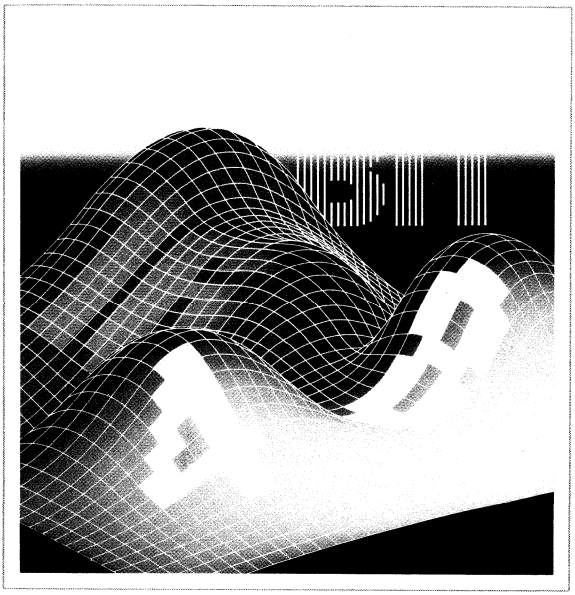
μCOM 87 Family Development Tools



User's Manual

NEC

THE μCOM-87 FAMILY

DEVELOPMENT TOOLS

USER'S MANUAL

Table of Contents

µCOM-87 Hardware Tools

Book	1	EVAKIT-7811 EVAKIT-87AD-1 (µCOM-87AD)	User's Manual
Book	2	The µCOM-87 Family In-Circuit-Emulator	User's Manual
Book	3	PG-1000	User's Manual
Book	4	PG-1003	User's Manual

µCOM-87 Software Tools

Book	5	RA-87 / RA-110 Relocatable Assembler	User's Manual
Book	6	µCOM-87AD Floating Point Arithmetic Library	Application Note
Book	7	µCOM-87AD Easy Arithmetic Software (Part 1)	Application Note

Book 1 Table of Contents

Section A

	Page
CHAPTER 1 GENERAL	1-1
1.1 Outline	1-1
1.2 Features	1-2
CHAPTER 2 SYSTEM CONFIGURATION	2-1
2.1 Configuration of the EVAKIT-87AD	2-2
2.2 EVAKIT-87AD Specifications	2-3
2.2.1 Major LSIs	2-3
2.2.2 Input and output	2-4
2.2.3 Environmental conditions	2-4
2.2.4 Power supply	2-4
CHAPTER 3 OPERATION	3-1
3.1 EVAKIT-87AD	3-1
3.1.1 Connectors	3-1
3.1.2 DIP switches	3-1
3.1.3 Sockets	3-9
3.2 Power supply	3-10
3.3 Connection to the user system	3-11
3.3.1 J3 pins	3-12
3.3.2 J4 pins	3-13
3.4 Connection of the console	3-14
3.4.1 TTY (ASR-33)	3-14
3.4.2 RS-232C console	3-16
3.4.3 TYPUTER 550TT (502T)	3-17
3.4.4 CRT terminal (Anritsu DDY-86)	3-18
3.5 Hexadecimal Keyboard	3-20
3.6 PROM mounting	3-21
3.7 Break Point Setting	3-22
3.8 Memory Card Connection	3-23
CHAPTER 4 MONITOR COMMANDS	4-1
4.1 Console Commands	4-1
4.1.1 Input format	4-1
4.1.2 List of Monitor Commands	4-2
4.1.3 Description of monitor commands	4-5
4.2 Hexadecimal Keyboard Commands	4-28
CHAPTER 5 COMPARISON WITH μPD7811G and μPD7810G	5-1
5.1 MM Register and MF Register	5-1
5.2 Mode Terminals	5-1
5.3 Ports D and F and ALE, WR and RD	5-1
5.4 User Program Memory	5-2

Page

Appendix I μCOM-87AD Instruction Set 5-3
Appendix II μCOM-87AD Instruction Application Table 5-7
Appendix III μCOM-87AD Mode Register Application Table 5-13

CHAPTER 6 MONITOR FOR DEBUG OF μPD7816G .....
6.1 Start-Up of Monitor For μPD7816G 6-1
6.2 Monitor Command For μPD7816G 6-1
6.3 Note At Emulation of μPD7816G 6-4

Section B

Page

CHAPTER 1 SYSTEM OVERVIEW
1.1 General 1
1.2 Features 1

CHAPTER 2 SYSTEM CONFIGURATION
2.1 System Configuration 2
2.2 EV87ADRTT Block Diagram 3
2.3 EV87ADRTT Specifications 3
2.3.1 Main LSIs 3
2.3.2 Environmental Conditions 3
2.3.3 Power Requirement 4
2.3.4 External Dimensions 4

CHAPTER 3 BEFORE OPERATION
3.1 External View of EV87ADRTT 5
3.2 System Connection 6
3.2.1 Setting the EVAKIT-87AD DIP switches and jumpers 6
3.2.2 Connecting the EV87ADRTT to the EVAKIT-87AD 6
3.3 Commands 7
3.3.1 Input format 7
3.3.2 Trace command 8
3.3.3 Expansion of the break command 10
3.4 EV87ADRTT Operation 11

CHAPTER 4 OPERATION
4.1 Power ON to Monitor Start-up 13
4.2 Command Examples 13
4.3 Display of Trace Data 17

APPENDIX 19

Book 2 Table of Contents

Section A Page

CHAPTER 1 GENERAL 1-1
1.1 Preface 1-1
1.2 Introduction 1-2
1.3 IE-7811H Specifications 1-3
1.4 IE-7811H Hardware Specifications 1-7
1.5 IE-7811H Software Specifications 1-11
1.6 Using the IE-7811H 1-22

CHAPTER 2 INSTALLATION 2-1
2.1 Outline 2-1
2.2 Unpacking 2-1
2.3 Installation 2-6
2.4 Connecting With Target System 2-17

CHAPTER 3 DETAILED DESCRIPTION OF MONITOR COMMANDS 3-1
3.1 Initial Setting 3-1
3.2 MAP (Mapping) Command 3-9
3.3 Memory Command 3-15
3.4 REG (Register) Command 3-25
3.5 Mode Register Command 3-32
3.6 Special Register Command 3-37
3.7 LOD (Load) Command 3-42
3.8 SAV (Save) Command 3-44
3.9 RUN (Emulation) Command 3-46
3.10 BR? (Break) Command 3-58
3.11 TR? (Trace) Command 3-67
3.12 CLK (Clock) Command 3-80
3.13 RES (Reset) Command 3-81
3.14 MAT (Operation) Command 3-83
3.15 SUF (Numeric Value Radix Specification) Command ... 3-86
3.16 ASM (Assemble) Command 3-87
3.17 DAS (Disassemble) Command 3-91
3.18 MOV (Memory Transfer) Command 3-91
3.19 Self-diagnostic Command (DIG) 3-96
3.20 PGM (PROM Programmer Control) Command 3-98

Page

CHAPTER 4 SERIAL INTERFACE 4-1
4.1 Outline 4-1
4.2 Basic Specifications 4-1
4.3 RS-232C Pin Description 4-3
4.4 Setting of Serial Protocol 4-4
4.5 Handshaking I/O Standards 4-5
4.6 Baud Rate Specification 4-6
4.7 Interfacing 4-7
4.8 Interfacing With Other Models 4-8

APPENDIX

A 1.1 Mechanical/Electrical Viation
A 1.2 CPU clock rate and CPU in emulator
A 1.3 IE-78C11 Test Points
A 1.4 IE-7809 driver & control block diagrams
A 1.5 IE-78C11 Target Probe
A 1.6 IE-7809/IE-87AD Clock switch
A 1.7 IE-7809 Mode Register and Special Register
A 1.8 IE-7809 Mapping
A 1.9 CLK Command
A 1.10 DIG Command
A 1.11 IP2 Connections for RS232
A 1.12 Instruction Set Differences
A 1.13 IE-78C11 Special Register Variance
A 1.14 IE-78C11 Mapping
A 1.15 IE-87AD-M Variation

Section B Page

PREFACE:
CHAPTER 1 SYSTEM OVERVIEW 1-1
1.1 Introduction 1-1

CHAPTER 2 OVERVIEW 2-1
2.1 Hardware Specifications 2-1
2.2 Software Specifications 2-7

CHAPTER 3 SYSTEM CONFIGURATION 3-1

CHAPTER 4 OPERATION OF IE-87AD 4-1
4.1 How to Connect the IE-87AD 4-1
4.2 Cautions 4-2

CHAPTER 5 PROTOTYPE SYSTEM DEVELOPMENT PROCEDURES 5-1
5.1 Flowchart for Prototype System Development 5-1

CHAPTER 6 COMMANDS 6-1

CHAPTER 7 DESCRIPTION OF RESPECTIVE COMMANDS 7-1
7.1 Outline 7-1
7.2 Commands 7-2
(1) IE87AD command 7-2
(2) EXIT command 7-15
(3) LOAD command 7-15
(4) SAVE command 7-17
(5) LIST command 7-18
(6) Self-Diagnosis command 7-20
(7) Mapping command 7-21
(8) Clock command 7-24
(9) Processor register command 7-25
(10) Mode register command 7-28
(11) Port command 7-29
(12) CAUSE command 7-31
(13) Mask register command 7-32
(14) Timer register command 7-33
(15) Timer event counter register command 7-33
(16) CR register command 7-34
(17) Serial command 7-34
(18) Memory command 7-35
(19) Symbol commands 7-37
(20) Break register command 7-40
(21) GO command 7-48
(22) STEP command 7-55
(23) Trace control command 7-56
(24) RESET CPU command 7-63
(25) RESET HARDWARE command 7-63
(26) On-line assembler command 7-63
(27) Reverse assemble command 7-65

CHAPTER 8 ELECTRICAL CHARACTERISTICS 8-1

CHAPTER 9 COMMAND FORMAT TABLE 9-1

Page

CHAPTER 10 μCOM87AD INSTRUCTION APPLICATION TABLE 10-1

CHAPTER 11 μCOM87-AD PIN CONFIGURATION 11-1

CHAPTER 12 ERROR CODE TABLE 12-1

CHAPTER 13 COMMAND EXECUTION EXAMPLES 13-1
(1) Self-Diagnosis command 13-1
(2) Register command 13-2
① Display example 13-2
② Setting example 13-4
(3) Mode register command 13-6
(4) Mask register command 13-7
(5) Timer register command 13-7
(6) Even timer register command 13-8
(7) Condition register command 13-8
(8) Serial command 13-9
(9) Port command 13-9
1 Display example 13-9
2 Setting example 13-10
(10) Mapping command 13-11
(11) Break register command 13-12
① All registers display 13-12
② Address break register 13-12
③ Data break register 13-13
④ External break register 13-15
⑤ Timer break register 13-15
⑥ Loop break register 13-15
(12) Memory command 13-16
① BYTE type 13-16
② WORD type 13-17
(13) Symbol command 13-18
(14) LOAD command 13-20
(15) GO command (Break condition setting) 13-21
(16) STEP command 13-34
(17) Trace command 13-36

APPENDIX A: Variations of IE-7809-I
(A1.1-A1.9) with respect to IE-87AD-I

APPENDIX B: μPD7809 Instruction Set
and Pin Configuration

APPENDIX C: μPD7809 Electrical Specification

Book 3 Table of Contents

		Page
PREFACE	.....	1
PART I	SYSTEM INSTALLATION	3
SECTION 1	SYSTEM OVERVIEW	4
1.1	PG-1000 Hardware Specifications	4
1.2	Operating Environment	4
1.3	Operating Modes	5
1.4	Appearance	6
SECTION 2	PACKING	7
2.1	Unpacking	7
2.2	Description of Items Contained in the Package	7
SECTION 3	PANEL FUNCTIONS	9
3.1	Front Panel	10
3.1.1	Display and mode specification LED	10
3.1.2	Key switches	12
3.2	Rear Panel	17
3.3	Bottom Panel	18
3.3.1	Quadruplet switch	19
3.3.2	Octuplet switch	19
SECTION 4	EXTERNAL INTERFACES	20
4.1	Serial Interface	20
4.1.1	Pin arrangements	20

4.1.2 Interface circuits 22
4.1.3 Interface switching 24
4.1.4 Baud rate switching 25
4.1.5 Parity check 27
4.1.6 Sending/receiving data format 28
4.1.7 Handshake system 29
4.1.8 Connection 34
4.2 Parallel Interface 35
4.2.1 Pin arrangements 36
4.2.2 Two-wire handshake 36
4.2.3 Connections 37

SECTION 5 CONNECTION TO MD-080/086 40
5.1 Selection of MD-080/086 Serial Channels 40
5.2 MD-080/086 Jumper Setting 41
5.2.1 Setting of MD-080/086-10 41
5.2.2 Setting the MD-080/086 50
5.3 Setting the PG-1000 58
5.4 Cable Connection 59

SECTION 6 PERSONAL MODULES 62

PART II OPERATION 64

SECTION 1 OPERATION OVERVIEW 65

		Page
SECTION 2	HOST CONNECTION MODE	66
2.1	General	66
2.2	Setting	66
2.3	Starting	67
2.3.1	Starting the MD-080/086	67
2.3.2	Starting the PG-1000	67
2.3.3	Starting the program (PGC)	68
2.4	Auto Mode	73
2.4.1	Specifying a file name	73
2.4.2	Specifying the separation of odd and even addresses	74
2.4.3	Loading	74
2.4.4	Specifying writing	75
2.5	Manual Mode	82
2.5.1	A command	83
2.5.2	E command	85
2.5.3	F command	86
2.5.4	G command	87
2.5.5	J command	90
2.5.6	K command	90
2.5.7	L command	93
2.5.8	M command	93
2.5.9	O command	94
2.5.10	P command	95
2.5.11	Q command	96
2.5.12	R command	97
2.5.13	S command	98
2.5.14	V command	99

Page

2.5.15 W command 101
2.5.16 Y command 103
2.5.17 Z command 103
2.6 System Control Character 104

SECTION 3 CONSOLE OPERATION MODE 105
3.1 General 105
3.2 Setting 105
3.3 Starting 105
3.4 Commands 106
3.4.1 I command 108
3.4.2 T command 108
3.5 System Control Characters 108

SECTION 4 STAND-ALONE MODE 109
4.1 Control Commands 110
4.1.1 Switching between auto and step modes 112
4.1.2 Indication of normal termination 113
4.1.3 Copy command 114
4.1.4 Blank check command 115
4.1.5 Program command 116
4.1.6 Verify command 118
4.1.7 Continuous command 119
4.2 Panel Command 121
4.2.1 PAE mode 123
4.2.2 FMT mode 132
4.2.3 CH mode 133
4.2.4 INS mode 139

Page

4.2.5 DEL mode 144

4.2.6 INIT mode 151

4.2.7 CMP mode 152

APPENDIXES 154

Appendix A 155

List of commands for host mode and console mode

Appendix B 156

List of error codes

Appendix C 157

HEX format file

Book 4 Table of Contents

		Page
CHAPTER 1	OUTLINE	1
1.1	Operating Environment	1
CHAPTER 2	ATTACHMENTS	2
CHAPTER 3	APPEARANCE	3
3.1	Front View	3
3.2	Bottom View	4
CHAPTER 4	CONNECTION TO PG-1000	5
CHAPTER 5	SELECTION OF FROM TO BE USED	9
5.1	Selection Procedures	9
5.2	Setting	10
CHAPTER 6	VOLTAGE REQUIREMENTS	11
APPENDIX A	WRITING TIME	12
APPENDIX B	EXAMPLE OF USED OF PG-1003	13

Book 5 Table of Contents

		Page
Preface		5
1.	RaXXX Relocatable Assembler Package Summary	7
1.1.	Software Supply Form	7
1.2.	System Environment	8
1.3	Text Editors	10
2.	Structure of Assembler RA87/RA310/RA110/RA210	10
2.0	General Structure	10
2.0.1	Tools for the modular construction of programs for the uCOM87/uCOM78xx family processors	11
2.0.2	Program Development	12
2.1	Source-file format	13
2.1.1.	Module Structure	13
2.1.1.1.	Module Header	13
2.1.1.2.	Module Body	14
2.1.1.3.	Module Tail	14
2.1.2.	Instruction Set	14
2.1.3.	Instruction Format and Length	16
2.1.4.	Symbols	16
2.1.5.	Data	17
2.1.5.1.	Simple Data Types	17
2.1.5.2.	Data Fields	17
2.1.5.3.	Data declarations	18
2.1.6.	Numerical Formats	18
2.1.7.	Procedures	18
2.1.8.	Macros	20
2.1.8.1.	Macro definition	20
2.1.8.2.	Macro activation	21
2.1.9.	Statements	21
2.1.9.1.	Labels	22
2.1.9.2.	Opcode Field	23
2.1.9.3.1.	Numerical Constants	23
2.1.9.3.2.	Register Names	24
2.1.9.3.3.	Handling of Symbols	29
2.1.9.3.4.	Expressions	29
2.1.9.3.5.	Location Counter	34
2.1.9.4.	Comment Field	35
2.1.10.	RA87/ RA310 /RA110 /RA210 Control Commands	35
2.1.10.1.	Control Command Line	35
2.1.10.2.	Primary-controls	36
2.1.10.2.1.	DEBUG/ NODEBUG	36
2.1.10.2.2.	OBJECT/ NOOBJECT	36
2.1.10.2.3.	PAGING/ NOPAGING	37
2.1.10.2.4.	PAGELENGTH	37
2.1.10.2.5.	PAGEWIDTH	37
2.1.10.2.6.	PRINT/ NOPRINT	38
2.1.10.2.7.	TITLE	38
2.1.10.2.8.	WORKFILE	38
2.1.10.2.9.	XREF/ NOXREF	39
2.1.10.2.10.	PROCESSOR	39
2.1.10.2.11.	OPTIMIZE	40
2.1.10.2.12.	DATE	40
2.1.10.2.13.	ERRORPRINT	40
2.1.10.2.14.	TAB	41
2.1.10.3.	General Controls	41
2.1.10.3.1.	EJECT	41
2.1.10.3.2.	INCLUDE	41

2.1.10.3.3.	LIST/ NOLIST	42
2.1.10.3.4.	SUBTITLE	42
2.1.10.3.5.	SET	43
2.1.10.3.6.	RESET	43
2.1.10.3.7.	IF/ ELSE/ ELSEIF/ ENDIF	43
2.1.10.4.	Priorities	45
2.1.11.	Assignment directives	45
2.1.11.1.	Segment Directives	45
2.1.11.1.1.	DSEG/ ENDS-Directive	46
2.1.11.1.2.	VSEG/ ENDS-Directive (RA87 only)	46
2.1.11.1.3.	CSEG/ ENDS-Directive	48
2.1.11.1.3.1.	CSEG FIXED/ ENDS Directive	48
2.1.11.1.3.2.	CSEG CALLT0 (RA310 / RA110) CSEG CALLT1 (RA310) CSEG CALLT (RA87 / RA210)	49
2.1.11.2.	Directives for Allocation of Constants	49
2.1.11.2.1.	EQU Assignment	49
2.1.11.2.2.	SET Assignment	50
2.1.11.3.	Directives for Memory reservation	51
2.1.11.3.1.	Bit memory reservation	51
2.1.11.3.1.1	DBIT Assignment for RA87 (uPD7809 only)	51
2.1.11.3.1.2	DBIT Assignment for RA310 / RA110 / RA210	51
2.1.11.3.2.	DB Assignment	52
2.1.11.3.3.	DW Assignment	52
2.1.11.3.4.	DS Assignment	53
2.1.11.4.	Directives for Describing End of Program	54
2.1.11.4.1.	END Assignment	54
2.1.11.5.	Directives for Link Run	54
2.1.11.5.1.	NAME Assignment	54
2.1.11.5.2.	PUBLIC Assignment	55
2.1.11.5.3.	EXTRN/EXTBIT assignment	55
2.1.11.6.	Directives for Locator Run	56
2.1.11.6.1.	ORG Assignment	56
2.1.11.7.	Macro-Directives	57
2.1.11.7.1.	MACRO Assignment	57
2.1.11.7.2	LOCAL Assignment	58
2.1.11.7.3.	REPT Assignment	59
2.1.11.7.4	IRP assignment	59
2.1.11.7.5.	ENDM assignment	60
2.1.11.7.6.	EXITM Assignment	60
2.1.11.7.7.	EXTBIT-Directive (RA310,RA210,RA110)	61
2.1.11.7.8.	BSEG/ENDS-Directive (not for RA87)	62
2.1.11.7.9.	Extended EQU-Assignment (RA310/RA210/RA110)	63
2.1.11.8.	Directives for use of Register banks (not for RA87)	63
2.1.11.8.1.	USING Directive (only for RA310)	63
2.1.11.9.	Assignment Directives of Functional and Absolute Names	63
2.1.11.9.1.	RSS-Directive (only RA310)	63
2.1.12.	RA87/RA310/RA110/RA210 Key words (reserved Words)	64
2.1.13.	Boundary Characters	65
2.1.14.	Segments	67
2.2.	Stack	68
2.2.1.	Stack Structure	68
2.2.2.	Stack size	68
2.3.	Character Set	69
2.4.	Limitations	70

3.	**Environment of Assemblers RA87/RA310/RA110/RA210**	71
3.1.	**Assembling**	71
3.1.1.	**File presence**	71
3.1.2.	**Assembler Call**	71
3.1.3.	Control commands in Call	72
3.2.	Output	72
3.2.1.	Output Equipment	72
3.2.1.1.	Terminal Equipment	72
3.2.1.2.	Printer Output	73
3.2.1.3.	Floppy Disc Output	73
3.2.2.	Output Files	73
3.2.2.1.	Listfile	73
3.2.2.2.	Cross Reference List	76
3.2.2.2.1.	Format of the Cross Reference List	76
3.3.	Assembler Error Messages	78
3.3.1.	Error Recognition and Error Classes	78
3.3.2.	Format of Error Messages	78
3.3.3.	List of Error Messages	78
4.	Linker Function	81
4.1.	Linkage Process	81
4.1.1.	Structure of a Translator Object File	81
4.1.2.	Combining Translator Modules to a Bound Module	83
4.1.3.	Handling of Library Modules during the Linkage Process	84
4.1.4.	Example of the Combination of Segments by Linking	85
4.2.	System Environment	85
4.3.	Restrictions	86
5.	Linker Operation	87
5.1.	Linker Activation	87
5.1.1.	Activation with an Explicit Command Line	87
5.1.2.	Activation with an Implicit Command Line (Command File)	88
5.2.	Linker Controls	89
5.3.	Startup Message	90
5.4.	List File Format	91
5.5.	Error Messages	92
5.5.1.	Operation Error Messages	92
5.5.2.	System Error Messages	98
5.6.	Warnings	99
6.	LOCxx-Function	100
6.1.	Method of operation of the Locater	100
6.1.1.	Locating a Relocatable Object File	100

6.1.2.	Structure of a Relocatable Object File	100
6.1.3.	Segment types	102
6.1.4.	Locating the Segments	108
6.1.4.1.	Principles	108
6.1.4.2.	Example	109
6.2.	System Environment	112
6.3	Restrictions	113
7.	LOCxx-Operation	114
7.1.	LOCxx-Call	415
7.1.1.	Activation with an explicite Command Line	114
7.1.2.	Activation with an implicite Command Line (Command File)	115
7.2.	LOCxx Control Commands	117
7.3.	List-File	119
7.4.	Error Messages	119
8.	List-File-Format	127
9.	The Librarian	130
9.1.	LB310 - Function	130
9.1.1.	Library creation process	130
9.1.2.	Using a Library	131
9.1.3.	System Environment	132
9.1.4.	Limitations	132
9.2.	Using LB310	133
9.2.1.	LB310 Call	133
9.2.2.	LB310 Control Commands	133
9.2.2.1.	CREATE	134
9.2.2.2.	ADD	134
9.2.2.3.	DELETE	135
9.2.2.4.	LIST	136
9.2.2.5.	EXIT	137
9.2.3.	LB310 Error Messages	137
9.2.3.1.	User Error Messages	137
9.2.4.	LB310 Warnings	144
APPENDIX A	An Assembler Example using RA310	145
APPENDIX B	References to uPD78112	166
APPENDIX C	References to uPD 78210/220/224	168

Book 6 Table of Contents

		Page
CHAPTER I	FLOATING POINT OPERATION SUBROUTINE	1
1.	Representation of Floating-point Binary Numbers	1
1-1	The Mantissa	1
1-2	The Exponent	2
2.	Variables	3
2-1	Binary Floating-point Accumulator (B.F.P.ACC)	3
2-2	Operand (OPE.)	4
2-3	SF	4
2-4	FSN	5
3.	Arithmetic Subroutines	5
3-1	Arithmetic Subroutines	5
3-1-1	BFADD (addition) subroutine	6
3-1-2	BFSUB (Subtruction) Subroutine	14
3-1-3	BFMUL (Multiplication) subroutine	15
3-1-4	BFDIV (Division) subroutine	21
3-2	Function Subroutine	32
3-2-1	SIN (Sine Function) subroutine	33
3-2-2	COS (Consine Function) subroutine	38
3-2-3	TAN (Tangent Function) subroutine	40
3-2-4	LNX (Natural Logarithmic Function) subroutine	44
3-2-5	LOG (Common Logarithmic Function) subroutine	48
3-2-6	EXP (Exponential Function) subroutine	50
3-2-7	ARCTAN (Arc Tangent) subroutine	55

Page

3-3-8 ARCSIN (Arc Sine) subroutine 58

3-2-9 ARCCOS (Arc Cosine) subroutine 61

3-2-10 POWER (Power Function) subroutine 64

3-2-11 SQR (Square Root) subroutine 69

3-2-12 TRACA (Polar coordinates ——→ Rectangular coordinates) subroutine 70

3-2-13 TRACB (Rectangular coordinates ——→ Polar coordinates) subroutine 74

3-3 Description of Subroutines for Converting a Character Constant into a Floating-point Binary Number and Vice Versa 79

3-3-1 BFINP (Character constant ——→ Floating-point binary number) subroutine 79

3-3-2 BFOUT (Floating-point binary number ——→ Character constant) subroutine 88

4. Construction of Subroutines for Arithmetic Operations 96

5. Common Subroutine 110

CHAPTER II APPLICATION 113

1. System Configuration 114

2. Evaluation of Expressions 115

3. Variables 122

4. Explanation of Program 124

CHAPTER III RESULT OF EVALUATION 131

CHAPTER IV SOURCE LIST 149

Table of Contents

Page

CHAPTER 1 ADDITION/SUBTRACTION/MULTIPLICATION/DIVISION 1
1.1 Binary Operation 1
1.1.1 Binary addition 1
1.1.2 Binary subtraction 3
1.1.3 Binary multiplication 5
1.1.4 Binary division 11
1.2 Decimal Operation 15
1.2.1 Decimal addition 15
1.2.2 Decimal subtraction 17
1.2.3 Decimal multiplication 19
1.2.4 Decimal division 24

CHAPTER 2 DATA TRANSFER 28

CHAPTER 3 SHIFT PROCESSING 30
3.1 N-Byte Data Shift Right 30
3.2 N-Byte Data Shift Left 31
3.3 N-Digit Data Shift Right 33
3.4 N-Digit Data Shift Left 34

CHAPTER 4 DATA CONVERSION PROCESSING 35
4.1 Converting HEX to BCD 35
4.2 Converting BCD to HEX 40
4.3 Converting ASCII to HEX 45
4.4 Converting HEX to ASCII 47

CHAPTER 5 COMPARISON PROCESSING 49
5.1 16-bit (2-byte) Data comparison 49
5.2 Data Retrieval 50
5.3 Bit Test and Set/Reset in Memory 55

CHAPTER 6 CONDITIONAL BRANCH PROCESSING 57

CHAPTER 7 TABLE REFERENCE PROCESSING 59

Book 1

EVAKIT-7811

EVAKIT-87AD-1

(μCOM-87AD)

REAL TIME TRACER

EV-87AD-RTT

(μCOM-87AD)

USER'S MANUAL

FOREWORD

This manual explains the operation of versions CB and CC of the EVAKIT-87AD. Users of the CA version should contact the sales personnel of NEC or an authorized NEC dealer for details of the operation of the CA version.
Check the lot number located near the IC147 to ascertain the version of your EVAKIT-87AD.

CAXXXXXXXX → CA version
CBXXXXXXXX → CB version
CCXXXXXXXX → CC version

CHAPTER 1. GENERAL

1.1 Outline

The EVAKIT-87AD is a hardware support board that is used for the efficient program development of a μCOM-87AD 8-bit single-chip microcomputer. As the μCOM-87AD incorporates a mask ROM within its single chip, it is necessary for the user to fully debug the data to be burned into the ROM before ordering the mask for the μCOM-87AD. In the EVAKIT-87AD, μPD7810G is used as the ROM and by rewriting the external memory and using the break functions, 1-step execution, trace functions, etc., provided, efficient debugging of the user program is made possible. The EVAKIT-87AD can also be connected to the user system to debug hardware and programs.

Progam debug may be performed using the EVAKIT-87AD and the hexadecimal keyboard included as an accessory. However, for more efficient debugging, it is recommended that you connect and use a TTY (or Miniprinter or Typuter) or PTR/PTP, etc.

1.2 Features

(1) Powerful monitor function

(2) 8K-byte user memory mounted. Connection of up to 64K bytes possible.

(3) Memory contents can be displayed and changed by a console, such as TTY or the hexadecimal keyboard.

(4) Contents of the internal registers and flags of the μPD7810G can be displayed and changed.

(5) Conditions can be freely set for the break point as the data, address and loop conditions.

(6) Single steps of a program may be executed.

(7) Program trace function to display the contents of the registers, flags, stack pointer and program counter.

(8) PROM writer (2716, 2732, 2732A, 2764)

(9) Connection of Real-time tracer is possible.

(10) MULTIBUS® connector provided. By using a general-purpose memory card, expansion of the external memory is possible.

MULTIBUS® is the registered trademark of Intel Corporation.

CHAPTER 2 SYSTEM CONFIGURATION

2.1 Configuration of the EVAKIT-87AD

The EVAKIT-87AD comprises the 8 blocks shown in Fig. 2-1.

(1) Supervisor block

This block controls the entire EVAKIT-87AD system. μPD8085AC is used as the supervisor and entire management of all the other blocks is performed by a 12K-byte monitor program.

(2) Program memory/control memory block

8K bytes of the built-in memory (EVACHIP addresses 0000 to 1FFFF) are allocated as the user program memory.

(3) μCOM-87AD emulation block

Execution of the user program and emulation of the terminal functions is performed by the μPD7810G. Note, however, that with the μPD7810G, trace of built-in RAM addresses FF00H to FFFFH and use of these addresses as address break conditions cannot be performed. Also, when setting break conditions, the monitor uses 3 bytes of the user memory. To trace the above addresses, consult NEC concerning the use of μPD78PG11E.

(4) Key input, LED display block

When using the EVAKIT-87AD as a stand-alone, key input and data display can be performed using the optional compact hexadecimal keyboard.

(5) Serial interface block

This block is used to connect an external console. The following types of interfaces are possible: TTL level, 30mA current loop and RS-232C. Transmission speeds may be set in the range 110 to 9,600 bps.

(6) PROM programmer block

This block is used to read and write the PROM (2716, 2732, 2732A or 2764). Externally provided 30V is necessary to write a PROM but is not required for reading PROMs.

(7) Break control block

This block is used during program debug to confirm the address of the program being executed, and the address of the memory being read or written.

The following two options are available for the setting of break points. Up to 4 continuous points may be set for serial break and up to 15 points, any one of which will cause a break when it is passed, can also be set for parallel break.

Also, any of the following may be used to provide the timing for the address or data break condition: OP code fetch, read, write and address latch.

Delays may be built into the break point so that break execution will be delayed up to 255 instructions after the break condition is met or only when the break condition has been met a specified number of times (up to 255). This is called the loop break condition.

(8) External memory block

The EVAKIT-87AD may be expanded by up to 64K bytes by using an external memory board for program development.

The supervisor (8085) and the EVACHIP (7810) can access a general-purpose memory board via the MULTIBUS.

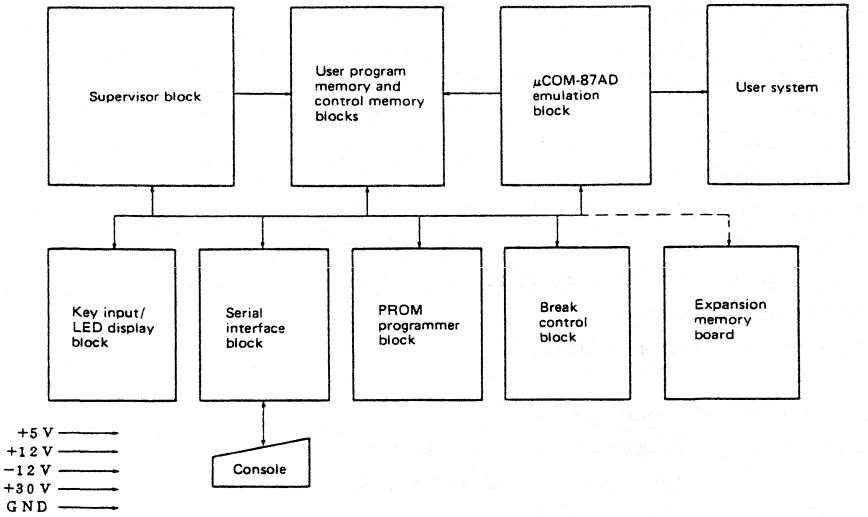


Fig. 2-1 EVAKIT-87AD Block Diagram

2.2 EVAKIT-87AD specifications

2.2.1 Major LSIs

System supervisor	μPD8085AC	x1
Evaluation chip	μPD7810G	x1
Monitor ROM	μPD2732D	x3
Monitor RAM	μPD2114LC	x2
Control RAM	μPD2149LC	x2
User program memory	μPD2167D	x8
Serial interface	μPD8251C	x1
Baud rate generator	μPD253C-5	x1
User PROM programmer	μPD2716	
	μPD2732	
	μPD2732A	
	μPD2764	

2.2.2 Input and output

(1) Input

Hexadecimal keyboard, TTY, etc.

(2) Output

7-segment LED, TTY, etc.

External interfaces

(1) Serial interface

TTL level, 20mA current loop or RS-232C.

Baud rate: 110 to 9600 bps.

(2) MULTIBUS interface

(3) Real-time tracer (option)

2.2.3 Environmental conditions

Operating:	Temperature:	0 to 40°C
	Humidity	: 10 to 90% RH
Non-operating:	Temperature:	-10 to 50°C
	Humidity	: 10 to 90% RH

2.2.4 Power supply

+5V, 6A

+12V, 0.2A (when using TTY)

$\pm$12V, 0.2A (when using RS-232C)

+30V, 0.3A (for writing PROMs)

CHAPTER 3 OPERATION

This chapter describes the function and operation of each element of the EVAKIT-87AD.

3.1 EVAKIT-87AD

The following is a description of the functions and setting methods for the on-board switches and connectors of the EVAKIT-87AD.

3.1.1 Connectors

Symbol	Type	Function
J1	50-pin flat cable connectors	To connect the EVAKIT-87AD to a Real-time tracer
J2		
J3	64-pin flat cable connector	Connects to the terminals of the μCOM-87AD. This connector is used to connect with the user system.
J4		
J5	50-pin flat cable connector	Used for connection with the key/display board.
J6	14-pin connector	Used for connection to the console (TTY, typuter, etc.)
J7	RS-232C connector	Used for connection with the RS-232C port.

3.1.2 DIP switches

SN1 Used for setting of the data transmission speed (baud rate), console and PROM.

SN2 Used for setting of the internal and external memories and the EVACHIP.

SN3 Corresponds to memory mapping register MM. This switch is used to perform memory mapping (port allocation) and control internal RAM access. The setting of this switch should be the same as the contents of register MM.

SN4 This switch is used when accessing an external memory via Port D (address/data bus) and Port F (address bus) to set the external memory as the memory in the user system or the memory on the MULTIBUS.

SN5 Corresponds to Mode F register MF. This switch is used to specify input/output of Port F. The setting of this switch should be the same as the contents of register MF.

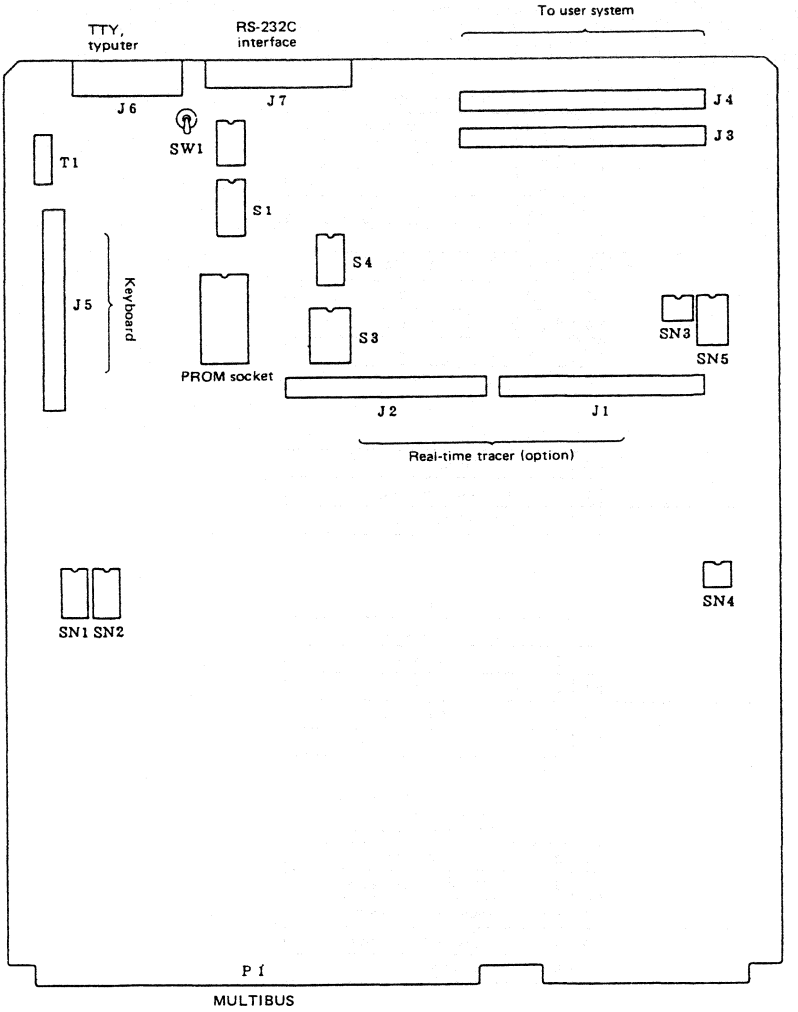


Fig. 3-1 Schematic of EVAKIT-87AD

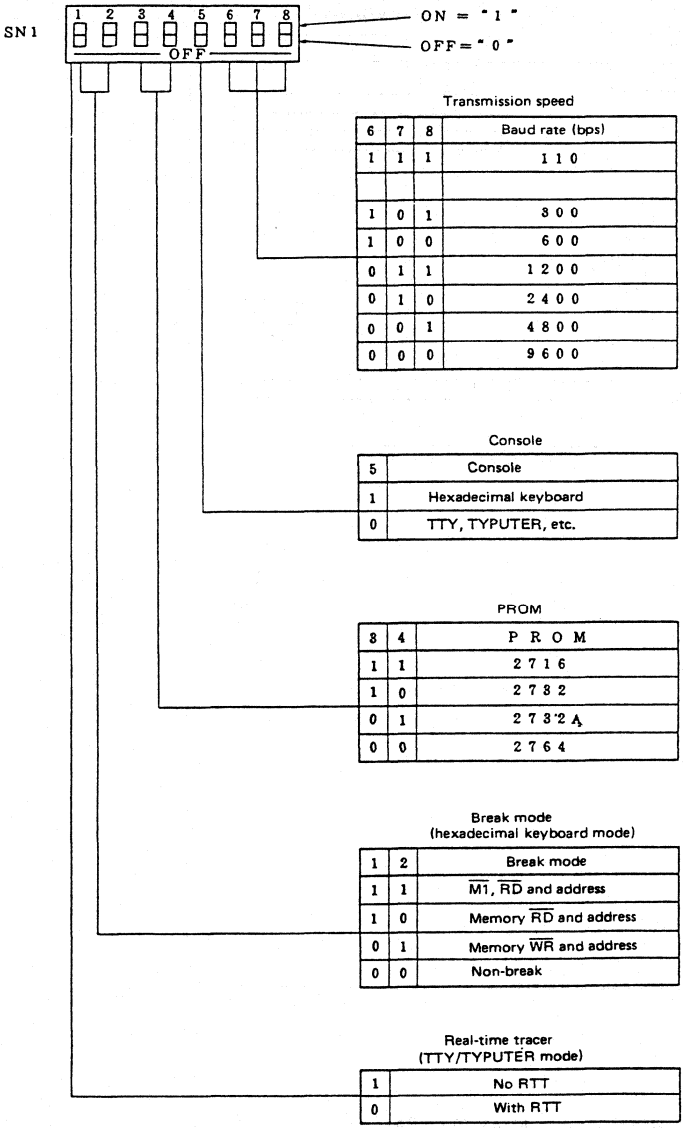
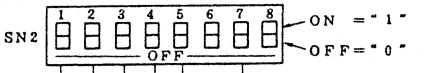


Fig. 3-2 Setting SN1



Buzzer

7	Buzzer
1	ON
0	OFF

Internal RAM access
(when using μPD78PG11)

5	Internal RAM access
1	Disable
0	Enable

EVACHIP

4	EVACHIP
1	μPD78PG11E使用のとき
0	μPD7810G

External memory size *

2	3	External memory size
1	1	4K bytes
0	1	16K bytes
1	0	μPD7811G mode
0	0	64K bytes

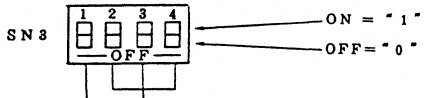
corresponds to the M1 terminal of μPD7811G and μPD7810G.
corresponds to the M0 terminal of μPD7811G and μPD7810G.
(When the CRT and typuter are used, the setting and display are reversed.)

Internal memory size

1	Internal memory size
1	8K bytes
0	4K bytes

* Even when used in the μPD7810G mode, addresses 0 to FFFFH access the meory on the EVAKIT.

Fig. 3-3 Setting SN2



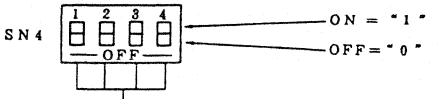
Mapping of external memory

2	3	4			
0	0	0	Port mode	Single chip	PD7–0 Input port PF7–0 Port mode
0	0	1	Port mode	Single chip	PD7–0 Output port PF7–0 Port mode
0	1	0	Expansion mode	256 bytes	PD7–0 Expansion mode PF7–0 Port mode
1	0	0	Expansion mode	4K bytes	PD7–0, PF3–0 } Expansion mode PF7–4 Port mode
1	1	0	Expansion mode	16K bytes	PD7–0, PF5–0 } Expansion mode PF7–6 Port mode
1	1	1	Expansion mode	60K bytes	PD7–0, PF7–0 } Expansion mode

Internal ROM access

1	Internal RAM access
0	Disabled
1	Enabled

Fig. 3-4 Setting SN3



MULTIBUS/Port switching

1	2	3	4	
0	0	0	0	All addresses accessed via MULTIBUS
1	0	0	0	Addresses 0 to 1FFF accessed via ports
0	1	0	0	Addresses 0 to 2FFF accessed via ports
1	1	0	0	Addresses 0 to 3FFF accessed via ports
0	0	1	0	Addresses 0 to 4FFF accessed via ports
1	0	1	0	Addresses 0 to 5FFF accessed via ports
0	1	1	0	Addresses 0 to 6FFF accessed via ports
1	1	1	0	Addresses 0 to 7FFF accessed via ports
0	0	0	1	Addresses 0 to 8FFF accessed via ports
1	0	0	1	Addresses 0 to 9FFF accessed via ports
0	1	0	1	Addresses 0 to AFFF accessed via ports
1	1	0	1	Addresses 0 to BFFF accessed via ports
0	0	1	1	Addresses 0 to CFFF accessed via ports
1	0	1	1	Addresses 0 to DFFF accessed via ports
0	1	1	1	Addresses 0 to EFFF accessed via ports
1	1	1	1	All addresses accessed via ports

Addresses 0 to FFF access the memory on the EVAKIT.

Fig. 3-5 Setting SN4

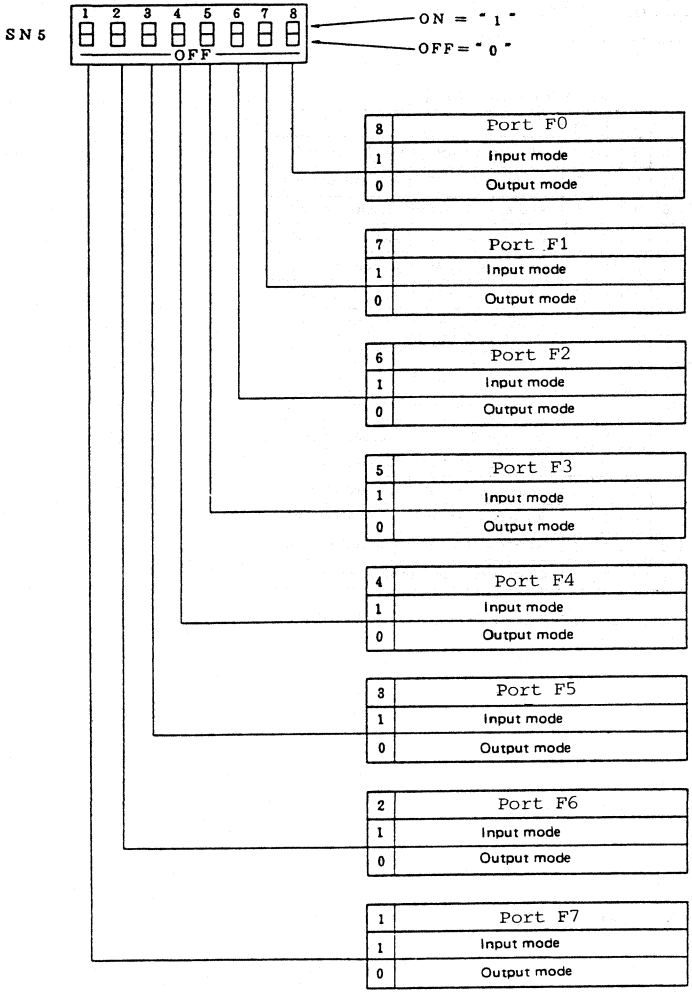
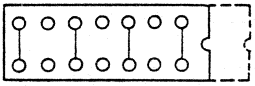


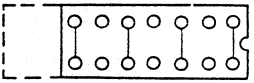
Fig. 3-6 Setting SN5

3.1.3 Sockets

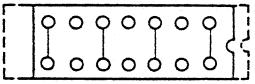
S1 This socket is used to set whether the interfacing of the EVAKIT with peripheral devices will be performed by RS-232C, 20mA current loop or TTL level.



When RS-232C (J7) is selected



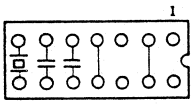
When 20mA current (J6) loop is selected



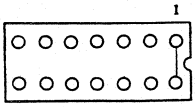
When TTL level (J6) is selected

Fig. 3-7 Socket S1

S3 This socket is used to select whether the crystal oscillator on the EVAKIT board or the oscillator in the user system is to be used as the oscillator circuit for the evaluation chip.



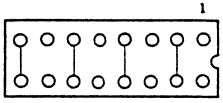
EVAKIT crystal
oscillator selected



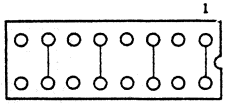
Oscillator in the
User system selected.

Fig. 3-8 Socket S3

S4 This socket is used to select whether the power for the evaluation chip is to be supplied from the power source on the EVAKIT board or from the power supply of the user system.



The power supply of
the EVAKIT is selected.



The power supply of
the user system is
selected.

Fig. 3-9 Socket S4

3.2 Power supply

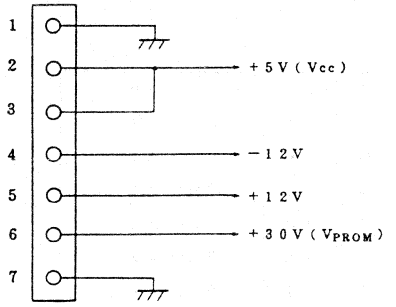
The EVAKIT-87AD will function in the normal use mode if +5V power is supplied.

In TTL level mode, +12V should be supplied. When using the RS-232C interface, ±12V should be supplied. To perform PROM write operations, +30V is necessary. Power should be applied in the order shown below. When turning the power supply OFF, the reverse order should be followed.

① +5V
② ±12V (+12V)
③ +30V

NOTE: If +30V power is supplied when the +5V power supply is not present, the PROM programmers may be damaged.

Connections of power supply terminal (T1)



When power is applied or when the system reset switch (SW1) is pressed, the monitor displays the following on the CRT to indicate the current state of the EVAKIT DIP switches.

```
EVAKIT-87AD MONITOR  VER. 1. 0
 MODE1 MODE0  RAE MM2  MM1  MM0     MF
   X     X     X   X    X    X   XXXXXXXX
 IN   -   nk
 USER -   mk
 BUS  - yyyy
*
```

For details of each item, refer to the description of the MD command.

If the EVACHIP (μPD7810G) is not functioning normally, the message "SYSTEM ERROR" will be displayed at this time. If this message is displayed, the setting of the oscillator (S3) and the power supply (S4) should be confirmed and the $\overline{NMI}$ terminal should be checked for noise.

3.3 Connection to the user system

A 64-pin connector (J3, J4) is provided to connect the EVAKIT-87AD to the user system. In addition, a cable with terminals compatible with the terminals of the μPD7811G is also provided.

Therefore, if the necessary connection to the user system is provided for the μPD7811G, connection of the EVAKIT-87AD to such a user system can be performed.

3.3.1 J3 pins

GND	1	2	Port A0
GND	3	4	Port A1
GND	5	6	Port A2
GND	7	8	Port A3
GND	9	10	Port A4
GND	11	12	Port A5
GND	13	14	Port A6
GND	15	16	Port A7
GND	17	18	Port B0
GND	19	20	Port B1
GND	21	22	Port B2
GND	23	24	Port B3
GND	25	26	Port B4
GND	27	28	Port B5
GND	29	30	Port B6
GND	31	32	Port B7
GND	33	34	Port C0
GND	35	36	Port C1
GND	37	38	Port C2
GND	39	40	Port C3
GND	41	42	Port C4
GND	43	44	Port C5
GND	45	46	Port C6
GND	47	48	Port C7
GND	49	50	$\overline{\text{NMI}}$
GND	51	52	INT1
GND	53	54	MODE1
GND	55	56	$\overline{\text{RESET}}$
GND	57	58	MODE0
GND	59	60	X2
GND	61	62	X1
GND	63	64	Vss

3.3.2 J4 pins

GND	1	2	Vcc
GND	3	4	V_{DD}
GND	5	6	Port D7
GND	7	8	Port D6
GND	9	10	Port D5
GND	11	12	Port D4
GND	13	14	Port D3
GND	15	16	Port D2
GND	17	18	Port D1
GND	19	20	Port D0
GND	21	22	Port F7
GND	23	24	Port F6
GND	25	26	Port F5
GND	27	28	Port F4
GND	29	30	Port F3
GND	31	32	Port F2
GND	33	34	Port F1
GND	35	36	Port F0
GND	37	38	ALE
GND	39	40	$\overline{WR}$
GND	41	42	$\overline{RD}$
GND	43	44	AVcc
GND	45	46	V_{AREF}
GND	47	48	AN7
GND	49	50	AN6
GND	51	52	AN5
GND	53	54	AN4
GND	55	56	AN3
GND	57	58	AN2
GND	59	60	AN1
GND	61	62	AN0
GND	63	64	AVss

3.4 Connection of the console

The EVAKIT-87AD can be connected to consoles which use the word configuration described below.

(1) Start bit 1 bit
(2) Data length 8 bits
(3) Parity None
(4) Stop bit 2 bits

If the above conditions are met, the interfaces listed in (1) below may be used at any of the baud rates listed in (2).

(1) Interfaces RS-232C
TTL level
20mA current loop

(2) Baud rate 110 bps
300 bps
600 bps
1,200 bps
2,400 bps
4,800 bps
9,600 bps

Representative consoles and the corresponding interfacing conditions are shown below.

(1) TTY (ASR-33)
110 bps, 20mA current loop
(2) Miniprinter (NA-2000 (M-30))
2,400 bps, 20mA current loop
(3) TYPUTER (550TT, 502T)
4,800 bps, TTL level

3.4.1 TTY (ASR-33)

To connect a TTY (ASR-33) to the EVAKIT-87AD, the wiring within the TTY must be changed. The steps for effecting the required changes are described below.

TTY internal rewiring

(1) Change the current source resistance to 1,450 ohms. The blue line connected to the 750-ohm terminal should be rewired to connect to the 1,450-ohm terminal.

(2) Change the source current level to 20mA.
Rewire the purple line connected to the No. 8 terminal to the No. 9 terminal.

(3) Change the mode of the console from half-duplex to full-duplex.
Rewire the brown/yellow line connected to the No. 3 terminal to the No. 5 terminal. Rewire the white/blue line connected to the No. 4 terminal to the No. 5 terminal.

(4) Add a relay to the paper tape reader circuit.
(a) Mount a 12V mercury relay with spark killer in the TTY.
(b) Connect one end of the mercury relay to terminal 1 of the mode switch.
(c) Connect the other end of the mercury relay to terminal L2 of the mode switch.
(d) Connect Pin 3 and Pin 11 of connector plug P4.
(e) Connect Pin 11 of connector plug P4 to terminal L2 of the mode switch.

(5) Connect terminals 3, 4, 6 and 7 on the terminal board to Pins 8, 1, 10 and 3, respectively, of connector 57-30140.
Connect ⊕ of the mercury relay coli to pin 5 of connector 57-30140 and ⊖, to pin 12.

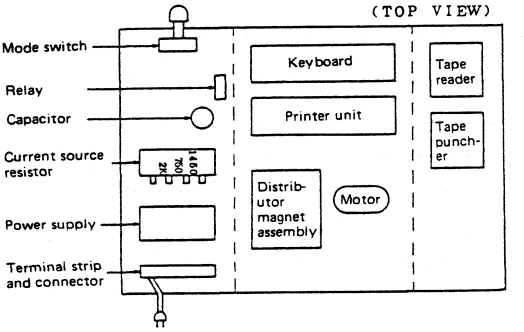


Fig. 3-10 TTY (ASR-33) Configuration

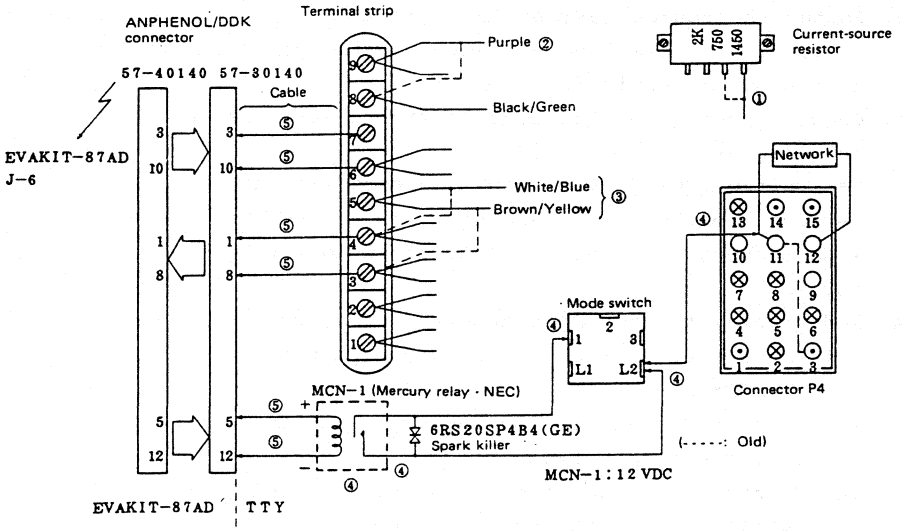


Fig. 3-11 TTY (ASR-33) Connection Change

3.4.2 RS-232C console

The EVAKIT-87AD can be connected to a console conforming to RS-232C specifications via the 25-pin connector (DBC-35S-AA) on the EVAKIT board. The signals used are the standard RS-232C signals shown below.

Connector pin No.	Signal name	Direction (EVAKIT — Console)	Active level
2	TxD	←	L
3	RxD	→	L
4	RTS	←	H
5	CTS	→	H
6	DSR	→	H
7	SG		

All other pins are open.

° Description of signals

TxD (Transmitted Data)	Data signal sent from the console to the EVAKIT.
RxD (Received Data)	Data signal sent from the EVAKIT to the console.
RTS (Request To Send)	Data send request signal sent to the EVAKIT. When this signal is low, the EVAKIT will not send data.
CTS (Clear To Send)	Enables data transmission from the console. This signal is always active after power ON.
DSR (Device Status Ready)	Used by the console to set the output condition of signal RTS.
SG (Signal Ground)	Signal ground terminal

3.4.3 TYPUTER 550TT (502T)

The TYPUTER is provided with a Hirose P-1628G-CA interface port which can be connected to the 57-30140 of the EVAKIT-87AD as described below.

EVAKIT-87AD

Signal name	Pin No.
$\overline{\text{SER BUSY}}$	6
GND	9
SER OUT	8
GND	9
SER IN	1
GND	10
REDR	12
GND	10
SERRST	13
GND	11

57-30140

TYPUTER 550TT(502T)

Pin No.	Signal name
13	$\overline{\text{DATA BUSY}}$
14	GND
3	DATA IN
4	GND
1	DATA OUT
2	GND
5	RDR STA
6	GND
23	IRST
24	GND

P-1628G-CA

The actual connections are shown in the figure below.

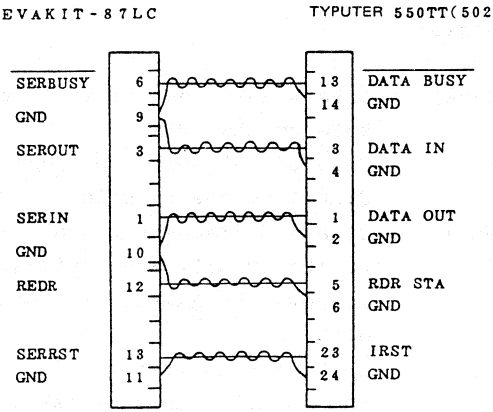


Fig. 3-12 Connection of TYPUTER and EVAKIT-87AD

3.4.4 CRT terminal (Anritsu DDY-86)

For a CRT terminal, the Anritsu DDY-86 is recommended. Connection of this terminal to the EVAKIT-87AD can be achieved simply by connecting the DDY-86 to the console terminal of the EVAKIT by a cable.

The CRT is provided with a JAE DB25P connector which is connected to the EVAKIT as shown below.

EVAKIT-87 DDY-86

Signal name	Pin No.	Pin No.	Signal name
$\overline{\text{SER BUSY}}$	6	16	$\overline{\text{TBSY}}$
GND	9	7	SG
SER OUT	3	13	R^+
GND	10	12	R^-
		23	Vcc(+5V)
		7	SG
SER IN	1	14	$\overline{\text{TXTTL}}$
GND	11	7	SG

57-30140 DB25P

The actual connections are shown in the figure below.

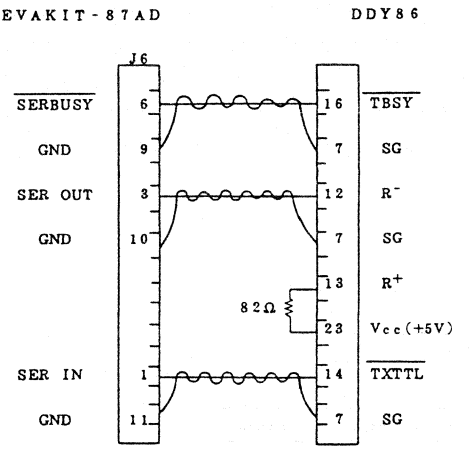


Fig. 3-13 Connection of DDY-86 CRT and EVAKIT-87AD

3.5 Hexadecimal Keyboard

Use the 50-pin flat cable provided to connect the hexadecimal keyboard to the EVAKIT-87AD as shown in Fig. 3-12.

At the hexadecimal keyboard side, the " " mark of the flat cable must match that on the hexadecimal keyboard connector. At the EVAKIT-87AD side, the mark " " of the cable should not match that of the keyboard connector.

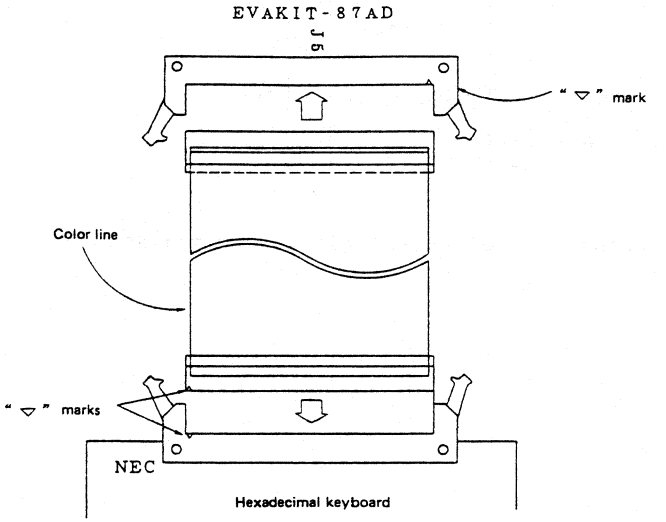


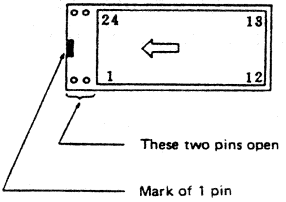
Fig. 3-14 Connecting the Hexadecimal Keyboard and EVAKIT-87AD

3.6 PROM mounting

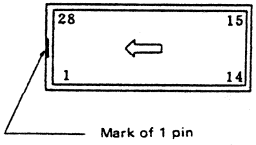
With the EVAKIT-87AD, it is possible to load programs from a PROM or to write the contents of the user program memory to a PROM.

PROMs are mounted as shown in the figure below.

°μPD2716, 2732, 2732A.



°μPD2764 (normal mounting)



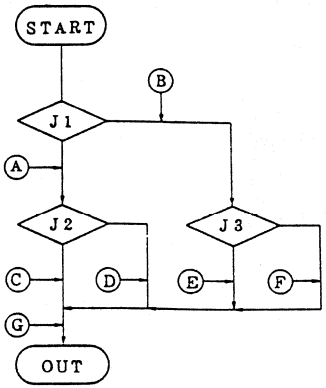
NOTE: If the PROM is mounted incorrectly, the PROM programmer may be damaged.

Fig. 3-15

3.7 Break Point Setting

In the EVAKIT-87AD, up to four break points can be set serially and in the console mode, break points can be set so that break will be applied when any one of up to fifteen break points is passed. The former type of break point setting is referred to as series break and the latter type as parallel break.

The series and parallel multibreak functions of the EVAKIT-87AD are explained using the following flowchart.



In the above flowchart, a program that begins execution at (START) and ends execution at (OUT) can have several program flows: Ⓐ → Ⓒ → Ⓖ, for example, or Ⓑ → Ⓕ → Ⓖ. Suppose that Ⓑ, Ⓕ, and Ⓖ have been specified as the break points using the EVAKIT-87AD series break function. In this case, a break occurs only when the program is executed in the sequence Ⓑ, Ⓕ, and Ⓖ. This flow can be easily understood if Ⓑ and Ⓕ are thought of as execution check points. A series break is especially effective to check the predicted program flow.

In contrast, a parallel break is effective when it is difficult to predict which of points Ⓒ, Ⓓ, Ⓔ or Ⓕ will be passed by a program that begins execution at (START). In other words, by setting points Ⓒ, Ⓓ, Ⓔ, and Ⓕ as parallel break points, a break will occur when program execution passes any one of these points.
In series break, up to four break points can be set while in parallel break, up to 15 break points can be set within a 1K-byte memory area.
In this case, the high-order six bits of addresses are fixed while the low-order ten bits are variable (0 to 3FFH, 400H to 7FFH, 800H to BFFH, etc.).
The EVAKIT-87AD also provides a delay break function that generates a break only when the specified number of steps have been executed after the break condition has been met. This type of break condition may be set for both series and parallel breaks.
Either, $\overline{MI}.\overline{RD}$ (operational code fetch), $\overline{RD}.\overline{WR}$ or ($\overline{RD}$ U $\overline{WR}$) of the μPD7810G may be selected as the trigger pulse for the break point.
To sum up, the selection of the trigger pulse for break points and use of series, parallel and delay breaks enable highly efficient debugging.

3.8 Memory Card Connection

A MULTIBUS-format memory card can be connected to the MULTIBUS of the EVAKIT-87AD PC board.
The following table shows connections of the MULTIBUS P1 side. The P2 side is not used. NEC SB-2032, etc., can be used as the memory card. Note that in this case, however, the operating frequency of the μPD7810 EVACHIP must be 6 MHz or below.

Power supply	1	GND	2	GND
	3	+5V	4	+5V
	5	+5V	6	+5V
	11	GND	12	GND
Bus control	13	$\overline{\text{BCLK}}$	14	$\overline{\text{INIT}}$
	19	$\overline{\text{MRDC}}$	20	$\overline{\text{MWTC}}$
	31	$\overline{\text{CCLK}}$		
Address	43	$\overline{\text{ADRE}}$	44	$\overline{\text{ADRF}}$
	45	$\overline{\text{ADRC}}$	46	$\overline{\text{ADRD}}$
	47	$\overline{\text{ADRA}}$	48	$\overline{\text{ADRB}}$
	49	$\overline{\text{ADR8}}$	50	$\overline{\text{ADR9}}$
	51	$\overline{\text{ADR6}}$	52	$\overline{\text{ADR7}}$
	53	$\overline{\text{ADR4}}$	54	$\overline{\text{ADR5}}$
	55	$\overline{\text{ADR2}}$	56	$\overline{\text{ADR3}}$
	57	$\overline{\text{ADR0}}$	58	$\overline{\text{ADR1}}$
Data	67	$\overline{\text{DAT6}}$	68	$\overline{\text{DAT7}}$
	69	$\overline{\text{DAT4}}$	70	$\overline{\text{DAT5}}$
	71	$\overline{\text{DAT2}}$	72	$\overline{\text{DAT3}}$
	73	$\overline{\text{DAT0}}$	74	$\overline{\text{DAT1}}$
Power supply	75	GND	76	GND
	81	+5V	82	+5V
	83	+5V	84	+5V
	85	GND	86	GND

NOTE: A 9.83 MHz clock is supplied to BCLK and CCLK.

CHAPTER 4 MONITOR COMMANDS

The monitor is a system program that communicates with the hexadecimal keyboard or console to enable program development. The monitor commands when the console is used differ from those available when the hexadecimal keyboard is used; these two sets of commands are described separately.

4.1 Console Commands

When the monitor program is activated at power application or by pressing the system reset switch after the console mode has been selected by DIP switch setting, the command prompt mark "*" is output to indicate that the monitor is waiting for input of a command. Commands are input in the format described below.

4.1.1 Input format

(1) All input is hexadecimal. If the input value is a 2-byte value, the low-order four digits are recognized. For a 1-byte value, the low-order two digits are recognized. If the input value is less than four digits, the value will be left-justified as shown below.

When four digits are recognized:

If 10000 is input, it will be taken as 0000.

If 123 is input, it will be taken as 0123.

(2) If the item to be input is already stored in the EVAKIT-87AD, it can be reinput pressing the space key (indicated by "Δ").

(3) Plural data are delimited by commas.

(4) Input of special keys

CTRL/C	Terminates execution and returns control to the monitor command level (input wait state).
CTRL/S	Temporarily halts output to the console.
CTRL/Q	Restarts the output that was halted by CTRL/S.
"↵"	Indicates C_R (carriage return).
"Δ"	Indicates a space.
~	Indicates output from the monitor.

4.1.2 List of Monitor Commands

	Command	Operation	Function
Memory list commands	CM	CM-XXXX↵	Rewrites the contents of address XXXX.
	DM	EM-XXXX,YYYY↵	Displays the contents of addresses XXXX to YYYY.
	IM	IM-XXXX,YYYY, ZZ↵	Sets data ZZ in addresses XXXX to YYYY.
	MM	MM-XXXX,YYYY, ZZZZ↵	Moves the contents of addresses XXXX to YYYY to a new location starting from address ZZZZ. memory range addresses XXXX to YYYY.
	SM	SM-XXXX,YYYY,ZZ↵	Searches for data ZZ in the memory range addresses XXXX to YYYY.
	VM	VM-XXXX,YYYY, ZZZZ↵	Compares the contents of memory of addresses XXXX to YYYY with the contents of memory following address ZZZZ.
	DA	DA-XXXX,YYYY↵	Disassembles the contents of addresses XXXX to YYYY.
	MD	MD↵	Displays the currently set memory mapping.
Register operation commands	DR	DR↵	Displays the internal registers.
	DQ	DQ↵	Displays which of the special registers can be be loaded.
	CA	CA XXXX-XXXX'↵	Rewrites the contents of the VA register.
	CB	CB XXXX-XXXX'↵	Rewrites the contents of the BC register.
	CD	CD XXXX-XXXX'↵	Rewrites the contents of the DE register.

	Command	Operation	Function
Register operation commands	CH	CH XXXX-XXXX'	Rewrites the contents of the HL register.
	CE	CE XXXX-XXXX'	Rewrites the contents of the EA register.
	CA'	CA'XXXX-XXXX'	Rewrites the contents of the V'A' register.
	CB'	CB'XXXX-XXXX'	Rewrites the contents of the B'C' register.
	CD'	CD'XXXX-XXXX'	Rewrites the contents of the D'E' register.
	CH'	CH'XXXX-XXXX'	Rewrites the contents of the H'L' register.
	CE'	CE'XXXX-XXXX'	Rewrites the contents of the EA' register.
	CS	CS XXXX-XXXX'	Rewrites the contents of the stack pointer.
	CP	CP XXXX-XXXX'	Rewrites the contents of the program counter.
	CQ	CQ AAA	Rewrites the contents of the special register.
	CZ	CZ X-X'	Sets/resets the Z flag.
	CC	CC X-X'	Sets/resets the C flag.
	CF	CF XXXX-XXXX'	Sets/resets the HC, Sk, Ll and L0 flags.
	ER	ER	Exchanges the contents of the main register and the alternate register.
I/O commands	IA	IA	Displays the contents of μCOM-87AD PA.
	IB	IB	Displays the contents of μCOM-87AD PB.
	IC	IC	Displays the contents of μCOM-87AD PC.
	ID	ID	Displays the conte ts of μCOM-87AD PD.

	Command	Operation	Function
I/O commands	IF	IF↵	Displays the conte ts of μCOM-87AD PF.
	OA	OA-XX↵	Outputs data XX to μCOM-87AD PA.
	OB	OB-XX↵	Outputs data XX to μCOM-87AD PB.
	OC	OC-XX↵	Outputs data XX to μCOM-87AD PC.
	OD	OD-XX↵	Outputs data XX to μCOM-87AD PD.
	OF	OF-XX↵	Outputs data XX to μCOM-87AD PF.
PROM operation commands	LH	LH-XXXX↵	Adds a bias of XXXX to the contents a HEX tape and loads the contents into the program memory.
	PH	PH-XXXX,YYYY↵	Outputs the contents of program memory addresses XXXX to YYYY to a HEX tape.
	VH	VH-XXXX↵	Adds a bias of XXXX to the contents of a HEX tape and compares the contents against the contetns of the program memory.
	DP	DP AAAA-XXXX, YYYY,ZZZZ↵	Displays the contents of PROM addresses XXXX to YYYY.
	LP	LP AAAA-XXXX, YYYY,ZZZZ↵	Loads the contents of PROM addresses XXXX to YYYY into the program memory from address ZZZZ.
	VP	VP AAAA-XXXX, YYYY,ZZZZ↵	Compares the contents of PROM addresses XXXX to YYYY against the contents of the program memory from address ZZZZ.

	Command	Operation	Function
PROM operation commands	ZP	ZP AAAA↵	Checks if the PROM has been erased.
	WP	WP AAAA-XXXX, YYYY,ZZZZ↵	Writes the contents of PROM addresses XXXX to YYYY into the program memory from address ZZZZ.
	GO	GO-XXXX↵	Executes the user program from address XXXX.
Execution commands	GB	GB-XXXX↵	Sets the break conditions specified by the BM and BP commands. Executes the user program from address XXXX. After the break is applied execution moves to the 1-step mode.
	GS	GS-XXXX,YY↵	Executes the user program from address XXXX for YY steps.
	GT	GT-XXXX,YY,N,n↵	Executes the user program from address XXXX for YY step or until the contents of register N match data n. The contents of the internal registers and displayed for each execution step until one of the above conditions is met. AFter that, execution moves to the 1-step mode.
Break commands	BM	BM aa,bb,cc-aa',bb',cc'↵	Sets the break mode, break points and delay count.
	BP	BP↵	Sets the address, data and mask conditions of the break point.
	BL	BL XX-XX'↵	Sets the number of break loop.
Other commands	RT	RT↵	Resets the μCOM-87AD.
	TM	TM-XXXX,YYYY↵	Performs a memory test for addresses XXXX to YYYY.

4.1.3 Description of monitor commands

(1) Memory commands

(a) CM Change Memory

When CM followed by "*" is input from the console after "*" has been output, "-" will be output indicating that you should input as a hexadecimal value the address whose contents you wish to change. When the address has been correctly input and followed by "↵", the console will perform line feed/carriage return and the address which you indicated above will be output along with the contents of that address in the format shown below. The "-" mark is also output indicating the input wait state. If the address was not correctly input, line feed/carriage return would be performed and "?" would be output on the console. Line feed/carriage return would be performed once more followed by output of "*" to indicated that control has returned to the command input wait state.

```
* CM-12↵
    0012 XX-
```

° When the address specified is less that 0FFF.

```
  (I)1012 XX-
```

° When the address is in the range 1000 to FFFF and the MULTIBUS board is selected for memory mapping.

```
  (E)1012 XX-
```

° When the address is in the range 1000 to FFFF and Ports PD and PF are selected for memory mapping.

```
    FF12 XX-
```

° When the address is FF00 or above.

In this state, when new data for the specified address is input as a two-digit hexadecimal number, the next address will be output and the program will wait for input of new data. It is possible to change the contents of the memory, one address after another, in this way. If the low-order 3 bits of the address are all '0' (for example, addresses ending in 08, 10, 18, 20, etc.) line feed/carriage return will be performed automatically and revision of data contents may be resumed after the new address is output. When you have completed changing the contents of the memory, simply input ↵ without any new data for the next address displayed and line feed/carriage return will be performed and control will return to the command input wait state (input prompt "*" will be output).

After "-" is output to prompt input of new memory data, if the following special inputs are performed, the functions described below will be activated.

^ input	To output the original data without changing it. Execution moves to the revision of the next address.
SHIFT + H input	To return to and revise the contents of the directly preceding address.
Back slash input	To cause the program to output "CM-" and enable the user to specify a new starting address for revision.

(b) DM Display Memory

When DM followed by is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting and end addresses, delimited by a comma, of the memory area which you wish to display. When these parameters are input followed by ↵ , the contents of the specified memory area will be output in the format shown below.

```
*DM-FFE,101F↵
    0FFE  XX XX
  (E)1000  XX XX.....
  (E)1010  XX XX.....
```

(c) EM Exchange Memory

When EM followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting and end addresses of the memory area to be exchanged along with the corresponding starting address of the memory area with which you wish to effect the exchange. All of these parameters should be delimited with commas. Then, when you input ↵ after making certain that two areas to be exchanged do not overlap, the contents of the two areas will be exchanged. When execution of this command has completed, "*" will be output.

```
*EM-XXXX,XXXX,XXXX↵
*
```

(d) IM Initialize Memory

When IM followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting and end addresses of the memory area which you wish to initialize followed by the data with which you wish to initialize this area. Then, when ↵ is input, this memory area will be initialized with with specified data. When execution of this command is completed, "*" will be output.

```
*IM-XXXX,XXXX,XX↵
*
```

(e) MM Move Memory

When MM followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting and end addresses of the memory area whose contents you wish to move along with the starting address of the memory area to which you wish to move these contents. All of these parameters should be delimited with commas. Then, when is input, the contents of the memory specified by the starting and end addresses will be checked and moved to the area starting with the destination starting address. When execution of this command has completed, "*" will be output. Unlike the EM command, there is no problem if the two memory areas overlap.

*MM~XXXX,XXXX,XXXX↵
*

(f) SM Search Memory

When SM followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting and end addresses of the memory area to be searched along with the search data. All of these parameters should be delimited with commas. Then, when ↵ is input, the contents of the specified memory area will first be checked and then searched.

(g) VM Verify Memory

When VM followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting and end addresses of the memory area to be used for verification and the starting address of the memory

compared. All of these parameters should be delimited with commas. Then, when ↵ is input, the contents of the two memory blocks will be compared. If the contents of the two blocks do not match, the address as well as the contents of the location in the block being verified will be output along with the corresponding verification data from the other block.
Unlike the SM command, verification by comparing the contents of the two blocks will continue until the specified end address is reached. To stop and restart execution, or to return to the monitor, CTRL/S, CTRL/Q and CTRL/C should be used.

```
*VM,XXXX,XXXX,XXXX↵
  0123 AB-CD
  1234 CD-EF
*
```

(h) DA Disassemble

When DA followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting and end addresses, delimited by a comma, of the program memory area which you wish to disassemble. When is input, the contents of the program memory ↵ within the specified range will be disassembled and output as shown below.

```
*DA,00C0,00CF↵
  PC    INSTRUCTION  MNEM   OPERAND
00C0    4854         SKIT   SB
00C2    54E000       JMP    00E0
00C5    6908         MVI    A,08
00C7    4DD0         MOV    MM,A
00C9    040000       LXI    SP,0
          :
*
```

(i) MD Mapping Display

When MD followed by ↵ is input from the console after "*" has been output, line feed/carriage return will be performed and the memory mapping set by DIP switches SN2 to SN4 will be output along with the I/O status of ports D and F as shown below.

```
MODE1 MODE0 RAE MM2 MM1 MM0      MF
  X     X     X   X   X   X   XXXXXXXX
IN    - nk
USER  - mk
BUS   - YYYY
*
```

MODE1 and MODE0 here indicate the set status of SN2 bits 2 and 3 (however, setting and display are reversed), RAE and MM2 to MM0 indicate the status of SN3, and MF indicates the status of SN5. n indicates the range of the internal memory set by SN2 bit 1. In normal mode, the size of the internal memory is 4K -1. m indicates the external access range of the user area set by SN2 bits 2 and 3 and SN3. The size of the user area is from nK to mK -1. When the memory area is expanded, SSSS is output. YYYY indicates the size of the expanded memory area that can be accessed via the MULTIBUS. The expanded memory range is from address YYYY (set by SN4) to address FEFF.

(2) Register Operation Commands

(a) DR Display Registers

When DR followed by ↵ is input from the console after "*" has been output, the contents of the program counter, registers and flags will be output as shown below.

```
△△△△△PC△△Z S H L L C△V△△A△△B△……L'△△E A'△△S P△
△○○○XXXX△XXXXXX△XX△XX△XX……XX△XXXX△XXXX
```

NOTES: 1. L is output in the order L0, L1.
2. "'" denotes an alternate register.

(b) DQ Display special registers
When DQ followed by ↵ is input from the console after "*" has been output, the special registers will be output in the format shown below.

```
△△△MK△△SMH△E0M△TMM△RXB△·····CR3△ECNT△ECPT
△△△XXX△XX△△XX△△XX△△XX△△·····XX△△XXXX△XXXX
```

(c) CA Display VA register
When CA followed by ↵ is input from the console after "*" has been output, the contents of registers V and A will be output as "△XXXX-". To change the contents of this register, input new data for the register followed by ↵. At this point if you input ↵ without inputting any new data, the register will be cleared to 0. Also, if "△" is input the contents of the register will not be changed but will be output a second time as they are and control will return to the command input wait state ("*").

(d) CB Change contents of register BC
(e) CD Change contents of register DE
(f) CH Change contents of register HL
(g) CE Change contents of register EA
(h) CS Change contents of the Stack pointer (SP)
(i) CP Change contents of the Program counter (PC)
(j) CA' Change the contents of register V'A'
When C is input and the A is input while holding down the CTRL key, followed by ↵, the contents of register V'A' will be output as "A'△XXXX-".
To change the contents of this register, input the new contents of the register followed by ↵.
(k) CB' Change the contents of register B'C'
(l) CD' Change the contents of register D'E'
(m) CH' Change the contents of register H'L'
(n) CE' Change the contents of register E'A'

(o) CZ Set and reset flag Z

When CZ followed by ↵ is input from the console after "*" has been output, the contents of flag Z will be output as "ΔX-". To set this flag, input '1' and to reset this flag input '0' followed by . At this time, if you input ↵ without inputting any data, the flag will be reset. If "Δ" is input, the contents of the flag will remain unchanged.

(p) CC Set and reset flag C.

(q) CF Change the contents of PSWs other than flags Z and C.

When CF followed by is input from the console after "*" has been output, the contents of PSWs SK, HC, L1 and L2 will be output as the 4-digit data "ΔXXXX-" with each of the above PSW being represented by one digit. Input '1' to set, and '0' to reset these flags.

(r) CQ Change special registers

When CQ followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the name of the register which you wish to change followed by ↵ . Line feed/carriage return will be performed and if the register is readable, the contents of the register will be output followed by "-" and if the register is unreadable, only "-" will be output. By inputting new data after "-" followed by ↵, the contents of each of these registers can be rewritten.

The PSW registers are as shown below.

MK	Interrupt mask register	R/W	16 bits
SMH	Serial mode high register	R/W	8 bits
SML	Serial mode low register	W	8 bits
ANM	A/D Channel register	R/W	8 bits
EOM	T/EC output mode register	R/W	8 bits
TMM	Timer mode register	R/W	8 bits

ETMM	T/EC mode register	W	8 bits
MM	Memory mapping register	W	8 bits
MCC	Mode control register C	W	8 bits
MA	Mode A register	W	8 bits
MB	Mode B register	W	8 bits
MC	Mode C register	W	8 bits
MF	Mode F register	W	8 bits
TXB	TX buffer	W	8 bits
TM0	Timer register 0	W	8 bits
TM1	Timer register 1	W	8 bits
ETM0	T/EC register 0	W	16 bits
ETM1	T/EC register 1	W	16 bits

(s) ER Exchange Registers

When ER followed by ↵ is input from the console after "*" has been output, the contents of the main, and alternate registers will be exchanged.

A — A'
B — B'
C — C'
D — D'
E — E'
H — H'
L — L'
EA—EA'

*ER↵
*

(3) Input/Output commands

(a) IA Port A Input

When IA followed by ↵ is input from the console after "*" has been output, the contents of Port A will be read and output. If Port A has been set in the output mode, the contents of the output latch will be output. If "Δ" is input, the same operation will be executed a second time and if ↵ is input, control will return to the command input wait state. The number of times that the same operation can be executed by inputting "Δ" is the number of console lines.

(b) IB Port B Input
(c) IC Port C Input
(d) ID Port D Input
(e) IF Port F Input
NOTE: This command can only be executed in the Port Mode.

(f) OA Port A Output
When OA followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, if you input the data that you wish to output via Port A as a 2-digit hexadecimal number, that data will be output when you input ↵. After the specified data has been output, the console will again output "-" so this operation may be performed repeatedly.
If "Δ" is input, control will return to the command input wait state.
(g) OB Port B Output
(h) OC Port C Output
(i) OD Port D Output
(j) OF Port F Output
NOTE: This command can only be executed in the Port Mode.

(4) Paper tape commands
(a) LH Load HEX tape
When LH followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the bias as a hexadecimal number. When ↵ is input, load from the HEX tape will begin.
The memory area that will be loaded is from the starting address of the program on the HEX tape plus the bias. If the user memory area is exceeded, loading will terminate, "?" will be

output and control will return to the command input wait state.

* LH=XXXX)

If a checksum error occurs, "?" will be output and control will return to the command input wait state.

(b) PH Punch HEX tape

When PH followed by) is input from the console after "*" has been output, "-" will be output. At this time you should input the starting and end addresses of the memory area to be output on the HEX tape delimited by a comma. When) is input, HEX tape punch will begin and when it has finished, the console will perform line feed/carriage return and output "*".
Check that the mode of the console is "punch ON" before executing this command.

* PH-XXXX,YYYY)

(c) VH Verify HEX tape

When VH followed by) is input from the console after "*" has been output, "-" will be output. At this time you should input the bias for tape verify as a hexadecimal number. When) is input, the bias value will be added to the addresses on the HEX tape and the contents of the HEX tape will be compared against the corresponding contents of the memory.
If the data does not match, the address of the HEX tape where the error occurred, the error data of the HEX tape and the corresponding contents of memory are output in that order.

(5) PROM commands

(a) DP Display PROM contents

When DP followed by ↵ is input from the console after "*" has been output, the type of PROM currently mounted will be output as "ΔXXXX-". At this time, if you input the starting and end addresses of the memory PROM memory which you wish to output delimited by a comma and followed by ↵ , the contents of the specified range will be output in the format shown below.

First the type of PROM is output as one of the following values.

"2716"
"2732"
"2732A"
"2764"

```
*DP2716-12E.140↵
ΔΔΔΔΔ12EΔΔΔXXΔXX
ΔΔΔΔΔ130ΔΔΔXXΔXX
ΔΔΔΔΔ140ΔΔΔXX
*
```

Note that if Δ is input while the contents of the PROM are being output, the current line will be output and then output will terminate.

(b) LP Load PROM

When LP followed by ↵ is input from the console after "*" has been output, the type of PROM currently mounted will be output as "ΔXXXX-". At this time you should input the starting and end addresses of the PROM area into which you wish to load your program. Next, input the starting address of the program which you wish to load. All of these parameters should be delimited with commas. When ↵ is input, starting address ≤ end address will be checked and the contents of the program memory area from the specified starting address will be loaded into the PROM. When loading has completed, "*" will be output.

```
* LP2732-XXX,YYY,ZZZ↵
```

(c) VP Verify PROM contents

When VP followed by ↵ is input from the console after "*" has been output, the type of PROM currently mounted will be output as "ΔXXXX-". At this time, you should input the starting and end addresses of the PROM which you wish to verify, along with the starting address of the memory area against which you wish to compare the contents of the PROM. All of these parameters should be delimited with commas. Then, when you input ↵ , verification will begin. If an error occurs, the console will perform line feed/carriage return and then output the PROM address where the error occurred, the contents of the PROM and the corresponding memory contents. After this output, verification will continue. To return control to the command input wait state, input CTRL/C.

```
* VP2764 110,2FF,1110↵
ΔΔΔΔ0127ΔFF-00
ΔΔΔΔ013FΔFF-32
*
```

(d) ZP PROM Zero check

When ZP followed by ↵ is input from the console after "*" has been output, the type of PROM currently mounted will be output as "ΔXXXX-". If ↵ is input at this time, the PROM zero check will be performed. If write data is encountered, the corresponding address and contents are output and the check continues. To return control to the command input wait state, input CTRL/C.

```
* ZPΔXXXX↵
ΔΔΔΔXXXXΔYY
*
```

(e) WP Write the contents of the memory to the PROM

When WP followed by ↵ is input from the console after "*" has been output, the type of PROM currently mounted will be output as "△XXXX-". At this time, you should input the starting and end addresses of the memory PROM memory to which you wish to write along with the starting address of the contents of the memory to be written. All of these parameters should be delimited with commas. Then, when you input ↵, the contents of the specified range of memory will be written to the PROM. Verification of the data will be performed at the same time.

If an error occurs, the PROM address and data and the corresponding contents of memory will be output in that order.

```
* WP △XXXX-1000,1FFF,,0↵
△△△△1234△56-78
*
```

(6) Execution commands

(a) GO Execution without break

When GO followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, if you input the starting address followed by ↵, execution will begin from the specified address. If only ↵ is input without a starting address, execution will begin from the address stored in the program counter. Break is applied by pressing either the CTRL/C or the ESC key. When break is applied in this manner, the contents of the registers at the time of break are output in the format shown below. Control then returns to the command input wait state.

```
* GO~XXXX↵
△△△△△PC△△△INST
△OOOXXXX△△XX△(XX)△(XX)△(XX)△△△
MNEMONIC          CZHLLC△V△△A△
AAAAAA……(24桁)……XXXXXX△XX△XX
                                  H△△L△△△E△△△△SP△
                                  XX△XX△XXXX△XXXX
*
```

(b) GB Go to Break

When GB followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting address for execution followed by ↵ . The break data will be referenced and after the break circuits have been set accordingly, execution will begin from the specified address. If only ↵ is input without a starting address, execution will begin from the address stored in the program counter.

When the set break condition is met, the contents of the registers are displayed in the same format as for the GO command with the difference that, after the contents of the registers are output, control does not return to the command input wait state indicated by the prompt "*". Instead "." is output to indicate the input wait state. If Δ is input at this time, one step of the program will be executed (1-step execution mode) and the result of execution will be output in the format shown below.

.○○○XXXX△△XX△(XX)△(XX)△(XX)△△△AAAAAA……

If "/" is input, line feed/carriage return will be performed and the output of the contents of the registers at the time of break will be repeated. If ↵ is input, control will return to the command wait input state indicated by the output of "*".

(c) GS Go Step

When GS followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the execution start address and the number of steps for which the program is to be executed delimited by a comma. When ↵ is input, program execution will begin and will continue for the specified number of steps.

When the specified number of steps have been executed, execution breaks.
Starting address and step count can be specified in the following four combinations.

① * GS=XXXX,YY)	Execution for YY steps from address XXXX
② * GS=,YY)	Execution for YY steps from the address stored in the program counter
③ * GS=XXXX)	Execution for one step from address XXXX
④ * GS=)	Execution for one step from the address stored in the program counter

The processing after the break is applied is the same as for the GB command.

(d) GT Go Trace

When GT followed by) is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting address, the number of trace steps and the trace conditions. This last parameter is optional and all parameters should be delimited with commas. Then, when) is input, the contents of the registers will be output for each execution step in the same format as when break is applied during execution of the GB command. Execution will continue for the specified number of trace steps or until the break condition is satisfied. After break is applied, "." will be output. At this time you have the same option as with the GB command of inputting Δ to execute the program in the 1-step mode. The input of the break condition and the contents of each condition are as described below.

,0,n To break when the contents of register V are n.
,1,n To break when the contents of register A are n.

,2,n Break when the contents of register B are n
,3,n Break when the contents of register C are n
,4,n Break when the contents of register D are n
,5,n Break when the contents of register E are n
,6,n Break when the contents of register H are n
,7,n Break when the contents of register L are n
,8,n Break when the contents of register EA are n
,9,n Break when the contents of register PC are n
,A,n Break when the contents of register SP are n
,B,n Break when the contents of flag Z are n
,C,n Break when the contents of flag SK are n
,D,n Break when the contents of flag HC are n
,E,n Break when the contents of flag L1 are n
,F,n Break when the contents of flag L0 are n
,10,n Break when the contents of flag C are n

* GT~,YYYY)	Execution for YYYY steps from the address stored in the program counter
* GT~,YYYY,1)	Execution for YYYY steps or until the contents of register A are '0' from the address stored in the program counter
* GT~XXXX,YYYY,B,1)	Execution for YYYY steps or until flag Z is '1' from address XXXX

(7) Break commands

Break commands are used to set data in the EVAKIT-87AD to control the execution of the EVACHIP. When the EVAKIT receives an execution command (GB, GS, GT), it uses the break condition data attached to the command to set the break conditions by hardware.

The break commands are BM, BP and BL. The BM command sets the break mode, the number of break points and the delay count. The BP command set the address, data and data mask conditions of the break point. The BL command sets the break loop condition.

Execution proceeds automatically from the BM command, so it is possible to set make a series of settings. It is also possible to start execution from either the BP or BL command.

(a) BM Set Break Mode register

When BM followed by ↵ is input from the console after "*" has been output, the break mode, number of break points and delay count will be output as "Δaa,bb,cc-".

The significance of each of these items is as follows:

° aa break mode (break trigger conditions)

- 0 $\overline{M1}$ · $\overline{RD}$ with address
- 1 $\overline{M1}$ · $\overline{RD}$ with data
- 2* $\overline{M1}$ · $\overline{RD}$ with data and address
- 4 Memory $\overline{RD}$ with address
- 5 Memory $\overline{RD}$ with data
- 6* Memory $\overline{RD}$ with data and address
- 8 Memory $\overline{WR}$ with address
- 9 Memory $\overline{WR}$ with data
- A* Memory $\overline{WR}$ with data and address
- C $\overline{RD}$ or $\overline{WR}$ with address
- D $\overline{RD}$ or $\overline{WR}$ with data
- E* $\overline{RD}$ or $\overline{WR}$ with data and address

- 0 Parallel break
- 1 Parallel break (with delay)
- 2 Serial break
- 3 Serial break (with delay)
- 4 Real-time tracer timer break

* Available only during serial break.

° bb Number of break points

Serial break : 4 break points max.

Parallel break : Up to 15 break points may be specified when when only the low-order 10 bits are variable.

When specifying both data and address break conditions, be sure the same number of break points exists for both conditions.

° cc Delay count

The delay count may be specified up to 255 instructions.

Each of these break command parameters should be input as hexadecimal values and delimited with commas. If you input "Δ" at this time, the contents of the break registers will remain unchanged. If you input ↵ , the register will be cleared to zero. However, during execution, the value for .bb and cc will be changed internally to 1. After all of the break conditions have been specified, if you input ↵ , line feed/carriage return will be performed and the following will be output.

≐BP1≐XXXX,YY,ZZ←

Where XXXX is the address break condition, YY is the data break condition and ZZ is the mask condition (available only when the data condition is set).

When the number of break points has been specified, the BP command automatically proceeds to the execution of the BL command.

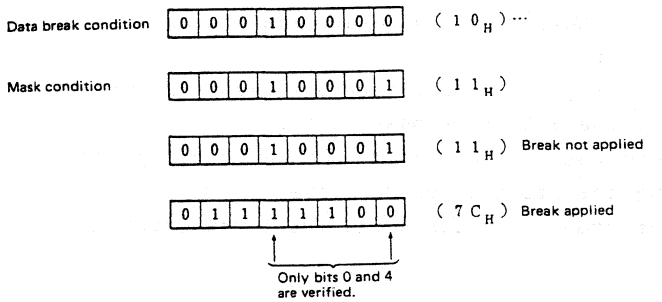
(b) BP Break Point setting

When BP is input from the console followed by the input of the pointer number, if this value does not conflict with the number of break points set in the BM command, the currently set address, data and mask conditions will be output followed by "-". At this point, you may input new data for each of these items.

*BP1△XXXX,YY,ZZ-XXXX',YY',ZZ')

Where XXXX is the address break condition, YY is the data break condition and ZZ is the mask condition (available only when the data break condition is set).

1) When only the address break condition is set as the break mode, break will be applied when address XXXX is accessed.
2) When setting the data break condition, please be sure to set the mask condition at the same time. When you perform this type of setting, the contents of the data bus will be compared against data YY and break will be applied when the two data are identical. However, the bits which have been masked by data ZZ as described below will not be compared.



(c) BL Set number of Break Loops

When BL followed by is input from the console after "*" has been output, currently set loop value will be displayed, so if you want to change the number of break loops, this is the time to do it.

* BL△XX-XX')

The following is a setting example for serial break with delay. Break will occur when 8 steps after bits 4 of addresses FF00 and FF110 have become '1' and bits 0 of the same addresses have become '0'.

```
*BM-aa,bb,cc-8A,02,08)
 BP1 XXXX,YY,ZZ-FF00,10,11)
 BP2 XXXX,YY,ZZ-FF10,10,11)
 BL *XX-01
*
```

(8) Other commands

(a) RT EVACHIP reset

When RT followed by) is input from the console after "*" has been output, the EVACHIP will be reset and the EVAKIT-87AD system will be initialized as follows:

- INTERRUPT ENABLE F/F is reset, inhibiting interrupts.
- All interrupt mask registers are reset to '0', causing the interrupt mask status.
- The interrupt request flag is reset causing any pending interrupt to be cancelled.
- All the program status words (PSW) are reset to '0'.

- Data 0000H is set in the program counter (PC).
- Data FFH is set in the Mode A, Mode B, Mode C and Mode F registers. Bits MM0, MM1 and MM2 of mode control register C and memory mapping register are reset and Ports A, B, C, D and F become input ports (output: high impedance).
- All the test flags with the exception of the SB flag are reset to '0'.
- The mode registers of the timer/event counter (ETTM, EOM) are reset to '0'.
- The serial mode high register (SMH) is reset and the serial mode low register (SML) is set to 48H.
- The A/D channel mode register of the A/D converter is reset.
- The $\overline{RD}/\overline{WR}$ signals become high level.
- At reset, the contents of the data memory as well as the contents of the following registers become undefined.
  Stack pointer (SP)
  Expanded accumulator (EA)
  Accumulator (A)
  General-purpose registers B, C, D, E, H, L, B', C', D', E', H' and L'
  All output port latches
  Timer registers 0 and 1 (TM0 and TM1)
  Timer/event counter registers 0 and 1 (ETM0 and ETM1)
  Memory mapping register bit REA.
  SB flag (test flag)

(b) TM Test program Memory

When TM followed by ↵ is input from the console after "*" has been output, "-" will be output. At this time, you should input the starting and end addresses of the memory area to be tested. When ↵ is input, the memory test will begin. Note that the contents of the memory are destroyed when this test is performed.

4.2 Hexadecimal Keyboard Commands

When the monitor program is activated at power application or by pressing the system reset switch after the keyboard mode has been selected by DIP switch setting, the command prompt "-" is output on the LED.
The display of this prompt indicates that the monitor program is waiting for input of commands or data.
The keyboard layout is shown in Fig. 4-1 and the function of each key in Table 4-1.

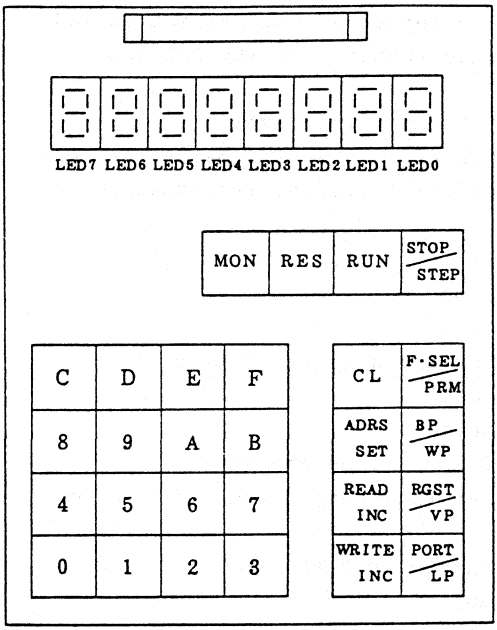


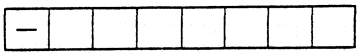
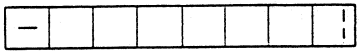
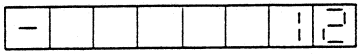
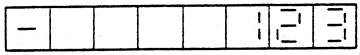
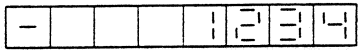
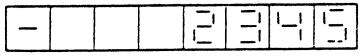
Fig. 4-1 Keyboard Layout

Key	Description
[0] - [F]	Hexadecimal keys for data input
[MON]	Monitor key. Returns to "-" prompt.
[RES]	Resets the EVACHIP.
[RUN]	Starts execution of the user program.
[STOP/STEP]	Stops program execution and enters the 1-step execution mode.
[CL]	Clears the contents of the user program memory to '0'.
[ADRS SET]	Sets and read program address.
[READ INC]	Increments and reads memory address.
[WRITE INC]	Increments and writes memory address.
[F. SEL/PRM]	Selects [BP/WP], [RGST/VP], [PORT/LP] key functions.
[BP WP]	Sets break points or writes to the PROM.
[RGST/VP]	Sets and reads register contents or verifies the PROM contents.
[PORT/LP]	Sets and reads the ports or loads the contents of the PROM.

° Explanation of keyboard commands

(1) Hexadecimal key input

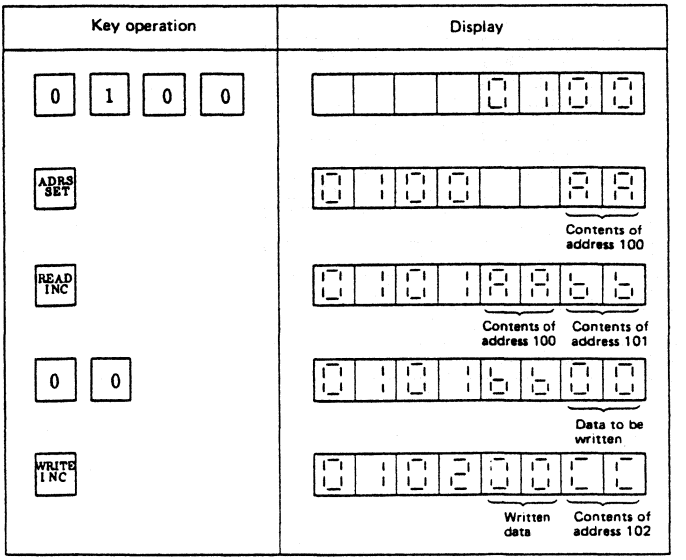
When any of hexadecimal keys [0] ~ [F] is pressed, the depressed key number will appear in the LED0 position (extreme right-hand side) of the 8-digit LED display. Any data that has already been input and is being displayed on the LED display will be shifted one digit to the left.

Key operation	Display
	
[1]	
[2]	
[3]	
[4]	
[5]	

(2) Memory Commands

When an address is input from the keyboard and the [ADRS SET] key is pressed, the address will be displayed on the 4 high-order digits of the LED display (LED7 ~ 4) and the contents of the address in the 2 low-order digits of the LED (LED1 ~ 0). At this time, if you press the [READ INC] key, the address being displayed on the LED will be incremented by 1 and the contents of the new address will be displayed in the 2 low-order digits of the LED display. The contents of the previous address will be displayed by LED3 ~ 2.

The contents of the memory can be revised by setting an address and displaying the contents of that address as described above and then using the hexadecimal keyboard to write (rewrite) the data of that address. The data which you have input from the keyboard will be written to the indicated address when you press the [WRITE INC] key. After the new data has been written, the address will be incremented and the contents of the next address will be displayed by LED1 ~ 0 and the contents which have just been written to the previous address by LED3 ~ 2.

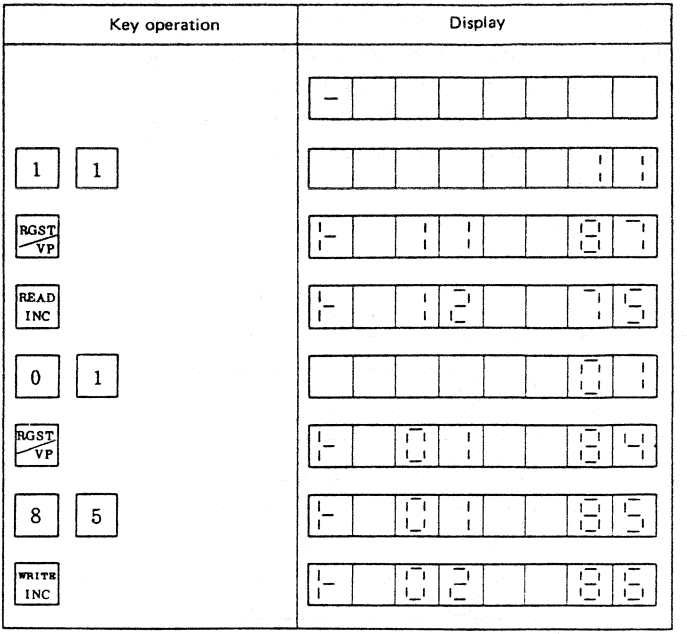


(3) Register commands

If you input the name of the desired register in the hexadecimal code shown below when the prompt mark is being displayed by LED7 and then press the RGST/VP key, the register mark ⊢ will be displayed by LED7, the register code by LED5 ~ 4 and the contents of the register by LED3 ~ 0. Then, if the READ/INC key is pressed, the register code number will be incremented and the contents of the new register will be displayed.

The contents of the registers may be changed in the same way that the contents of the memory may be rewritten. If new data is input and the WRITE/INC key is pressed while the contents of the register are being displayed, the contents of the register will be rewritten. The register code will be incremented and the contents of the corresponding register will be displayed.

Register name	Code	Register name	Code
V	$(00)_H$	V'	$(10)_H$
A	$(01)_H$	A'	$(11)_H$
B	$(02)_H$	B'	$(12)_H$
C	$(03)_H$	C'	$(13)_H$
D	$(04)_H$	D'	$(14)_H$
E	$(05)_H$	E'	$(15)_H$
H	$(06)_H$	H'	$(16)_H$
L	$(07)_H$	L'	$(17)_H$
EA	$(08)_H$	EA'	$(18)_H$
PC	$(09)_H$		
SP	$(0A)_H$		
PSW	$(0B)_H$		



(4) Port commands

If the PORT LP key is pressed while the prompt mark is being displayed by LED7, "PA", indicating Port A, will be displayed by LED7 ~ 6 and the contents of Port A will be displayed by LED1 ~ 0. At this time, if you press the READ INC key, LED7 ~ 6 will display "PB" and the contents of Port B will be displayed by LED1 ~ 0. By repeating this operation, the contents of Ports C ~ F can be displayed. To rewrite the contents of a port (to output via the port) press the PORT LP key and the name of the port will be displayed by the 2 high-order digits of the LED display and the contents by the 2 low-order digits of the display.

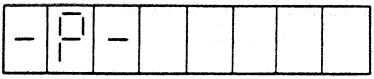
At this time, you should input the data that you wish to output via the port. Then, when you press the WRITE INC key, the data which you have specified will be output. The name of the port will be incremented (PA → PB) and the contents of Port B will be displayed by LED1 ~ 0. The previously output data will be shifted to LED3 ~ 2.

Key operation	Display
PORT/LP	PA AA
READ INC	Pb bb
READ INC	PC CC
READ INC	Pd dd
READ INC	PF FF
READ INC	PA AA

Key operation	Display
PORT/LP	PA AA
READ INC	Pb bb
5 5	Pb bb55
WRITE INC	PC 55CC
READ INC	Pd dd

(5) PROM commands

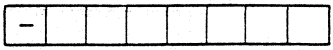
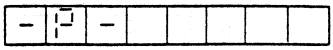
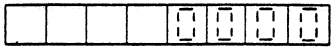
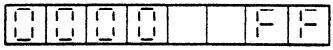
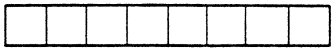
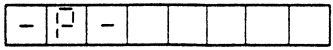
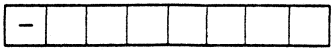
The following 4 keys control the PROM operations: F·SEL/PRM , BP/WP , RESET/VP and PORT/P . Of these, the F·SEL/PRM key is used to select the PROM monitor and when it has been pressed, the following prompt will be displayed on the LED.



The other 3 keys are used to select the PROM operation to be performed.

(a) Read (PROM to program memory)

After the hexadecimal keys and the ADRS/SET key have been used to set the starting address of program memory, if the PORT/LP key is pressed, the contents of the PROM (2K bytes for the 2716, 4K bytes for the 2732 and 2732A, and 8K bytes for the 2764) will be loaded into the program memory. The LED display is extinguished during the loading operation and when loading has completed, control returns to the PROM monitor.

Key operation	Display
	
F·SEL/PRM	
0 0 0 0	
ADRS/SET	
PORT/LP	
	
F·SEL/PRM	

(b) Verify PROM contents

After the hexadecimal keys and the [ADRS SET] key have been used to set the starting and end addresses of program memory, if the [RGST VP] key is pressed, the contents of the PROM will be compared against the contents of the program memory. If a verification error occurs, the address where the error occurred will be displayed by LED7 ~ 4, the contents of the program memory by LED3 ~ 2 and the contents of the PROM by LED1 ~ 0.

Verify operation will halt when and error occurs and can be restarted by pressing the [RGST VP] key.

Key operation	Display
	- P -
0	- P - 0
ADRS SET	0 0 0 0 4 8
0 F F F	0 0 0 0 F F F
RGST VP	0 1 2 3 8 3 8 7
RGST VP	- P -

(c) Write (program memory to PROM)

After the hexadecimal keys and the [ADPS SET] key have been used to set the starting address of program memory, if the [BP/WP] key is pressed, the contents of the specified memory area will be written to the PROM. During the write operation, the data that has been written to the PROM is read and check for whether or not it has been correctly written. If a write error is detected, the corresponding address is displayed by LED7 ~ 4, the correct data by LED3 ~ 2 and the error data by LED1 ~ 0. Write operation halts when an error occurs and can be restarted by pressing the [BP/WP] key a second time.

Key operation	Display
	- P -
0	- P - 0
ADRS SET	0 0 0 0 4 8
BP/WP	0 1 2 3 8 7 8 3
BP/WP	
	- P -

(6) Execution control commands

(a) MON key

Pressing the MON key causes system reset of the EVAKIT-87AD. However, the EVACHIP is not initialized and the condition at the time when the MON key was pressed is preserved. Also, the - prompt is displayed by LED7.

(b) RES key

Pressing this key causes reset of the EVACHIP. The data save area of the control RAM is cleared to '0' and the port latches, etc., are initialized. The contents of address 0 are displayed on the LED.

(c) RUN key

If the ADRS SET key is pressed after the hexadecimal keys and the RUN key have been used to set the starting address program execution will begin from the specified address For execution the RAM and register data of the EVACHIP that is withdrawn to the control RAM is restored During execution or the user program U will be displayed in the MSD of the LED display

(d) STOP/STEP key

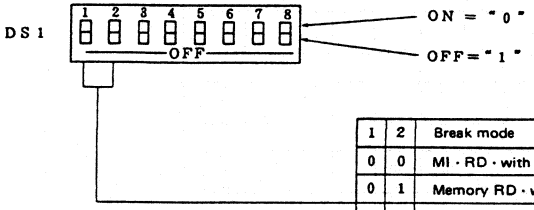
If this key is pressed during execution of the user program execution will halt and the 1-step execution mode will be entered. Commands which do not change the contents of the program counter can be executed while program execution is halted. Commands that change the contents of the other registers or the break conditions may be executed at this time. When execution is halted. the address at which execution was halted, the contents of the accumulator and the contents of the address are displayed by LED 7 ~ 4, 3 ~ 2 and 1 ~ 0, respectively.

Key operation	Display
RES	0 0 0 0 5 4
RUN	U
STOP/STEP	1 9 8 2 0 5 0 8
MON	-
1 9 8 2	- 1 9 8 2
ADRS SET	1 9 8 2 0 8
0 9	1 9 8 2 0 8 0 9
WRITE INC	1 9 8 3 0 9 4 d
MON	-
1 9 8 2	- 1 9 8 2
ADRS SET	1 9 8 2 0 9
RUN	U

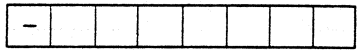
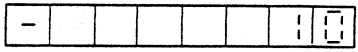
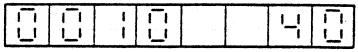
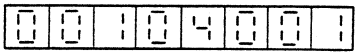
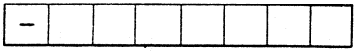
(7) Break point setting

The address break condition is set by first setting the address by using the hexadecimal keyboard and the [ADRS SET] key and then inputting a 2-digit value for the loop count and pressing the [BP/WP] key.

The break mode is set by manipulating DIP switch 1 (DS1).



1	2	Break mode
0	0	MI · RD · with address
0	1	Memory RD · with address
1	0	Memory WR · with address
1	1	Nonbreak

Key operation	Display
	
1 0	
ADRS SET	
0 1	
BP/WP	

CHAPTER 5 COMPARISON WITH μPD7811G and μPD7810G

The EVAKIT-87AD differs from the μPD7811G and μPD7810G in the following points. Also, please use a μPD78PG11E to check programs when ordering a ROM code.

5.1 MM Register and MF Register

In the EVAKIT-87AD, the setting of the memory mapping register MM and the mode F register MF is performed by manipulating DIP switches SN3 and SN5. This setting should agree with that of the program. Also note that it is therefore not possible to evaluate program which use dynamically change the memory mapping or the setting of Ports D and F.

5.2 Mode Terminals

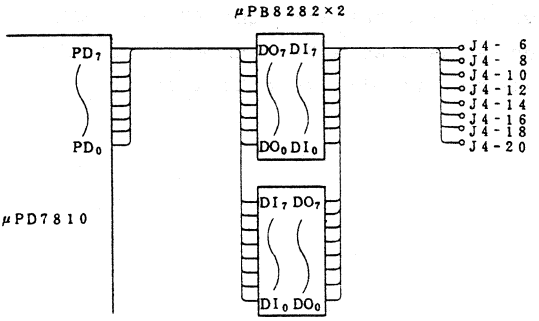
In the EVAKIT-87AD, the setting for Mode 0 and Mode 1 is performed by DIP switch S2. This setting should be the same as that of the user system.

However, even in the μPD7810 mode, the 4K bytes of addresses 0 to FFFH are always accessed on the EVAKIT memory so in fact it is not possible to evaluate the μPD7810.

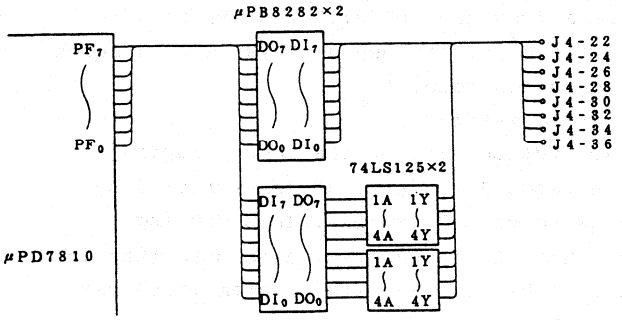
5.3 Ports D and F and ALE, WR and RD

In the EVAKIT-87AD Ports D and F and the ALE, WR and RD terminals are configured as shown below.

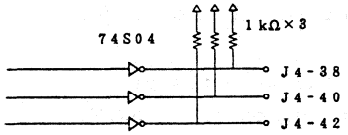
(1) Port D



(2) Port F



(3) ALE, WR and RD



5.4 User Program Memory

In the EVAKIT-87AD, the area corresponding to the internal ROM (0 to FFFH) of the μPD7811G is structured in the μPD2167D.

With the μPD7811G, it is not possible to write data to the internal ROM by software but with the EVAKIT-87AD, this memory area can be used as a RAM. However, should the user program overrun, it will rewrite itself.

Appendix I μCOM-87AD INSTRUCTION SET

Operand format/description

Format	Description
r r1 r2	V, A, B, C, D, E, H, L EAH, EAL, B, C, D, E, H, L A, B, C
sr sr1 sr2 sr3 sr4	PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, SML, EOM, ETMM, TMM, MM, MCC, MA, MB, MC, MF, TXB, TM0, TM1 PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, TMM, RXB, CR0, CR1, CR2, CR3 PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, TMM ETM0, ETM1 ECNT, ECPT
rp rp1 rp2 rp3	SP, B, D, H V, B, D, H, EA SP, B, D, H, EA B, D, H
rpa rpa1 rpa2 rpa3	B, D, H, D+, H+, D−, H− B, D, H B, D, H, D+, H+, D−, H−, D+byte, H+A, H+B, H+EA, H+byte D, H, D++, H++, D+byte, H+A, H+B, H+EA, H+byte
wa	8-bit immediate data
word byte bit	16-bit immediate data 8-bit immediate data 3-bit immediate data
f	CY, HC, Z
irf	FNMI, FT0, FT1, F1, F2, FE0, FE1, FEIN, FAD, FSR, FST, ER, OV, AN4, AN5, AN6, AN7, SB

Remarks

1. sr~sr4(special register)

PA	: PORT A	ETMM	: TIMER/ EVENT COUNTER MODE
PB	: PORT B	EOM	: TIMER/ EVENT COUNTER OUTPUT MODE
PC	: PORT C	ANM	: A/D CHANNEL MODE
PD	: PORT D	CR0	: A/D CONVERSION
PF	: PORT F	\	RESULT 0~3
MA	: MODE A	CR3	
MB	: MODE B	TXB	: Tx BUFFER
MC	: MODE C	RXB	: Rx BUFFER
MCC	: MODE CONTROL C	SMH	: SERIAL MODE High
MF	: MODE F	SML	: SERIAL MODE Low
MM	: MEMORY MAPPING	MKH	: MASK High
TM0	: TIMER REG0	MKL	: MASK Low
TM1	: TIMER REG1		
TMM	: TIMER MODE		
ETM0	: TIMER/ EVENT COUNTER REG0		
ETM1	: TIMER/ EVENT COUNTER REG1		
ECNT	: TIMER/ EVENT COUNTER UPCOUNTER		
ECPT	: TIMER/ EVENT COUNTER CAPTURE		

2. rp~rp3(register pair)

SP	: STACK POINTER
B	: BC
D	: DE
H	: HL
V	: VA
EA	: EXTENDED ACCUMULATOR

3. rpa~rpa3(rp addressing)

B	: (BC)
D	: (DE)
H	: (HL)
D+	: (DE)+
H+	: (HL)+
D−	: (DE)−
H−	: (HL)−
↑++	: (DE)++
H++	: (HL)++
D+byte	: (DE+byte)
H+A	: (HL+A)
H+B	: (HL+B)
H+EA	: (HL+EA)
H+byte	: (HL+byte)

4. f(flag)

CY	: CARRY
HC	: HALF CARRY
Z	: ZERO

5. irf(interrupt flag)

FNMI	: INTFNMI
FT0	: INTFT0
FT1	: INTFT1
F1	: INTF1
F2	: INTF2
FE0	: INTFE0
FE1	: INTFE1
FEIN	: INTFEIN
FAD	: INTFAD
FSR	: INTFSR
FST	: INTFST
ER	: ERROR
OV	: OVERFLOW
AN4	: ANALOG INPUT4~7
\	
AN7	
SB	: STANDBY

	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
8-bit data transfer instruction	MOV	r1, A	1	4	r1←A			
		A, r1	1	4	A←r1			
		sr, A	2	10	sr←A			
		A, sr1	2	10	A←sr1			
		r, word	4	17	r←(word)			
		word, r	4	17	(word)←r			
	MVI	r, byte	2	7	r←byte			
		sr2, byte	3	14	sr2←byte			
	MVIW	wa, byte	3	13	(V, wa)←byte			
	MVIX	rpa1,byte	2	10	(rpa1)←byte			
	STAW	wa	2	10	(V, wa)←A			
	LDAW	wa	2	10	A←(V, wa)			
	STAX	rpa2	½	⅜	(rpa2)←A			
	LDAX	rpa2	½	⅜	A←(rpa2)			
	EXX		1	4	B↔B', C↔C', D↔D' E↔E', H↔H', L↔L'			
	EXA		1	4	V, A↔V', A', EA↔EA'			
	EXH		1	4	H, L↔H', L'			
	BLOCK		1	13 (C+1)	(DE)'←(HL)'; C←C−1 End if borrow			
16-bit data transfer instruction	DMOV	rp3, EA	1	4	rp3←EAL, rp3+1←EAH			
		EA, rp3	1	4	EAL←rp3, EAH←rp3+1			
		sr3, EA	2	14	sr3←EA			
		EA, sr4	2	14	EA←sr4			
	SBCD	word	4	20	(word)←C, (word+1)←B			
	SDED	word	4	20	(word)←E, (word+1)←D			
	SHLD	word	4	20	(word)←L, (word+1)←H			
	SSPD	word	4	20	(word)←SPL, (word+1)←SPH			
	STEAX	rpa3	½	⅝	(rpa3)←EAL, (rpa3+1)←EAH			
	LBCD	word	4	20	C←(word), B←(word+1)			
	LDED	word	4	20	E←(word), D←(word+1)			
	LHLD	word	4	20	L←(word), H←(word+1)			
	LSPD	word	4	20	SPL←(word), SPH←(word+1)			
	LDEAX	rpa3	⅝	⅝	EAL←(rpa3), EAH←(rpa3+1)			
	PUSH	rp1	1	13	(SP−1)←rp1H (SP−2)←rp1L, SP←SP−2			
	POP	rp1	1	10	rp1L←(SP) rp1H←(SP+1), SP←SP+2			
	LXI	rp2, word	3	10	rp2←word			
	TABLE		2	17	C←(PC+3+A) B←(PC+3+A+1)			
8-bit operation instruction (register)	ADD	A, r	2	8	A←A+r		↕	↕
		r, A	2	8	r←r+A		↕	↕
	ADC	A, r	2	8	A←A+r+CY		↕	↕
		r, A	2	8	r←r+A+CY		↕	↕
	ADDNC	A, r	2	8	A←A+r	No Carry	↕	↕
		r, A	2	8	r←r+A	No Carry	↕	↕
	SUB	A, r	2	8	A←A−r		↕	↕
		r, A	2	8	r←r−A		↕	↕
	SBB	A, r	2	8	A←A−r−CY		↕	↕
		r, A	2	8	r←r−A−CY		↕	↕
	SUBNB	A, r	2	8	A←A−r	No Borrow	↕	↕
		r, A	2	8	r←r−A	No Borrow	↕	↕
	ANA	A, r	2	8	A←A∧r			↕
		r, A	2	8	r←r∧A			↕
	ORA	A, r	2	8	A←A∨r			↕
		r, A	2	8	r←r∨A			↕
	XRA	A, r	2	8	A←A⊻r			↕
		r, A	2	8	r←r⊻A			↕

	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
8-bit operation instruction (register)	GTA	A, r	2	8	A−r−1	No Borrow	↕	↕
		r, A	2	8	r−A−1	No Borrow	↕	↕
	LTA	A, r	2	8	A−r	Borrow	↕	↕
		r, A	2	8	r−A	Borrow	↕	↕
	NEA	A, r	2	8	A−r	No Zero	↕	↕
		r, A	2	8	r−A	No Zero	↕	↕
	EQA	A, r	2	8	A−r	Zero	↕	↕
		r, A	2	8	r−A	Zero	↕	↕
	ONA	A, r	2	8	A∧r	No Zero		↕
	OFFA	A, r	2	8	A∧r	Zero		↕
8-bit operation instruction (memory)	ADDX	rpa	2	11	A←A+(rpa)		↕	↕
	ADCX	rpa	2	11	A←A+(rpa)+CY		↕	↕
	ADDNCX	rpa	2	11	A←A+(rpa)	No Carry	↕	↕
	SUBX	rpa	2	11	A←A−(rpa)		↕	↕
	SBBX	rpa	2	11	A←A−(rpa)−CY		↕	↕
	SUBNBX	rpa	2	11	A←A−(rpa)	No Borrow	↕	↕
	ANAX	rpa	2	11	A←A∧(rpa)			↕
	ORAX	rpa	2	11	A←A∨(rpa)			↕
	XRAX	rpa	2	11	A←A⊻(rpa)			↕
	GTAX	rpa	2	11	A−(rpa)−1	No Borrow	↕	↕
	LTAX	rpa	2	11	A−(rpa)	Borrow	↕	↕
	NEAX	rpa	2	11	A−(rpa)	No Zero	↕	↕
	EQAX	rpa	2	11	A−(rpa)	Zero	↕	↕
	ONAX	rpa	2	11	A∧(rpa)	No Zero		↕
	OFFAX	rpa	2	11	A∧(rpa)	Zero		↕
Immediate data operation instruction	ADI	A, byte	2	7	A←A+byte		↕	↕
		r, byte	3	11	r←r+byte		↕	↕
		sr2, byte	3	20	sr2←sr2+byte		↕	↕
	ACI	A, byte	2	7	A←A+byte+CY		↕	↕
		r, byte	3	11	r←r+byte+CY		↕	↕
		sr2, byte	3	20	sr2←sr2+byte+CY		↕	↕
	ADINC	A, byte	2	7	A←A+byte	No Carry	↕	↕
		r, byte	3	11	r←r+byte	No Carry	↕	↕
		sr2, byte	3	20	sr2←sr2+byte	No Carry	↕	↕
	SUI	A, byte	2	7	A←A−byte		↕	↕
		r, byte	3	11	r←r−byte		↕	↕
		sr2, byte	3	20	sr2←sr2−byte		↕	↕
	SBI	A, byte	2	7	A←A−byte−CY		↕	↕
		r, byte	3	11	r←r−byte−CY		↕	↕
		sr2, byte	3	20	sr2←sr2−byte−CY		↕	↕
	SUINB	A, byte	2	7	A←A−byte	No Borrow	↕	↕
		r, byte	3	11	r←r−byte	No Borrow	↕	↕
		sr2, byte	3	20	sr2←sr2−byte	No Borrow	↕	↕
	ANI	A, byte	2	7	A←A∧byte			↕
		r, byte	3	11	r←r∧byte			↕
		sr2, byte	3	20	sr2←sr2∧byte			↕
	ORI	A, byte	2	7	A←A∨byte			↕
		r, byte	3	11	r←r∨byte			↕
		sr2, byte	3	20	sr2←sr2∨byte			↕
	XRI	A, byte	2	7	A←A⊻byte			↕
		r, byte	3	11	r←r⊻byte			↕
		sr2, byte	3	20	sr2←sr2⊻byte			↕
	GTI	A, byte	2	7	A−byte−1	No Borrow	↕	↕
		r, byte	3	11	r−byte−1	No Borrow	↕	↕

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
Immediate data operation instruction	GTI	sr2, byte	3	14	sr2 − byte − 1	No Borrow	↕	↕
	LTI	A, byte	2	7	A − byte	Borrow	↕	↕
		r, byte	3	11	r − byte	Borrow	↕	↕
		sr2, byte	3	14	sr2 − byte	Borrow	↕	↕
	NEI	A, byte	2	7	A − byte	No Zero	↕	↕
		r, byte	3	11	r − byte	No Zero	↕	↕
		sr2, byte	3	14	sr2 − byte	No Zero	↕	↕
	EQI	A, byte	2	7	A − byte	Zero	↕	↕
		r, byte	3	11	r − byte	Zero	↕	↕
		sr2, byte	3	14	sr2 − byte	Zero	↕	↕
	ONI	A, byte	2	7	A ∧ byte	No Zero		↕
		r, byte	3	11	r ∧ byte	No Zero		↕
		sr2, byte	3	14	sr2 ∧ byte	No Zero		↕
	OFFI	A, byte	2	7	A ∧ byte	Zero		↕
		r, byte	3	11	r ∧ byte	Zero		↕
		sr2, byte	3	14	sr2 ∧ byte	Zero		↕
Working register operation instruction	ADDW	wa	3	14	A ← A + (V, wa)		↕	↕
	ADCW	wa	3	14	A ← A + (V, wa) + CY		↕	↕
	ADDNCW	wa	3	14	A ← A + (V, wa)	No Carry	↕	↕
	SUBW	wa	3	14	A ← A − (V, wa)		↕	↕
	SBBW	wa	3	14	A ← A − (V, wa) − CY		↕	↕
	SUBNBW	wa	3	14	A ← A − (V, wa)	No Borrow	↕	↕
	ANAW	wa	3	14	A ← A ∧ (V, wa)			↕
	ORAW	wa	3	14	A ← A ∨ (V, wa)			↕
	XRAW	wa	3	14	A ← A ∨ (V, wa)			↕
	GTAW	wa	3	14	A − (V, wa) − 1	No Borrow	↕	↕
	LTAW	wa	3	14	A − (V, wa)	Borrow	↕	↕
	NEAW	wa	3	14	A − (V, wa)	No Zero	↕	↕
	EQAW	wa	3	14	A − (V, wa)	Zero	↕	↕
	ONAW	wa	3	14	A ∧ (V, wa)	No Zero		↕
	OFFAW	wa	3	14	A ∧ (V, wa)	Zero		↕
	ANIW	wa, byte	3	19	(V, wa) ← (V, wa) ∧ byte			↕
	ORIW	wa, byte	3	19	(V, wa) ← (V, wa) ∨ byte			↕
	GTIW	wa, byte	3	13	(V, wa) − byte − 1	No Borrow	↕	↕
	LTIW	wa, byte	3	13	(V, wa) − byte	Borrow	↕	↕
	NEIW	wa, byte	3	13	(V, wa) − byte	No Zero	↕	↕
	EQIW	wa, byte	3	13	(V, wa) − byte	Zero	↕	↕
	ONIW	wa, byte	3	13	(V, wa) ∧ byte	No Zero		↕
	OFFIW	wa, byte	3	13	(V, wa) ∧ byte	Zero		↕
16-bit operation instruction	EADD	EA, r2	2	11	EA ← EA + r2		↕	↕
	DADD	EA, rp3	2	11	EA ← EA + rp3		↕	↕
	DADC	EA, rp3	2	11	EA ← EA + rp3 + CY		↕	↕
	DADDNC	EA, rp3	2	11	EA ← EA + rp3	No Carry	↕	↕
	ESUB	EA, r2	2	11	EA ← EA − r2		↕	↕
	DSUB	EA, rp3	2	11	EA ← EA − rp3		↕	↕
	DSBB	EA, rp3	2	11	EA ← EA − rp3 − CY		↕	↕
	DSUBNB	EA, rp3	2	11	EA ← EA − rp3	No Borrow	↕	↕
	DAN	EA, rp3	2	11	EA ← EA ∧ rp3			↕
	DOR	EA, rp3	2	11	EA ← EA ∨ rp3			↕
	DXR	EA, rp3	2	11	EA ← EA ∨ rp3			↕
	DGT	EA, rp3	2	11	EA − rp3 − 1	No Borrow	↕	↕
	DLT	EA, rp3	2	11	EA − rp3	Borrow	↕	↕
	DNE	EA, rp3	2	11	EA − rp3	No Zero	↕	↕
	DEQ	EA, rp3	2	11	EA − rp3	Zero	↕	↕

16-bit operation instruction

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
Multiplication/devision instruction	DON	EA, rp3	2	11	EA ∧ rp3	No Zero		↕
	DOFF	EA, rp3	2	11	EA ∧ rp3	Zero		↕
	MUL	r2	2	32	EA ← A × r2			
	DIV	r2	2	38	EA ← EA ÷ r2, r2 ← (余り)			
Addition/subtraction instruction	INR	r2	1	4	r2 ← r2 + 1	Carry		↕
	INRW	wa	2	16	(V, wa) ← (V, wa) + 1	Carry		↕
	INX	rp	1	7	rp ← rp + 1			
		EA	1	7	EA ← EA + 1			
	DCR	r2	1	4	r2 ← r2 − 1	Borrow		↕
	DCRW	wa	2	16	(V, wa) ← (V, wa) − 1	Borrow		↕
	DCX	rp	1	7	rp ← rp − 1			
		EA	1	7	EA ← EA − 1			
Other operation instructions	DAA		1	4	Decimal Adjust Accumulator		↕	↕
	STC		2	8	CY ← 1		1	
	CLC		2	8	CY ← 0		0	
	NEGA		2	8	A ← Ā + 1			
Rotation shift instruction	RLD		2	17	Rotate Left Digit			
	RRD		2	17	Rotate Right Digit			
	RLL	r2	2	8	r2$_{n+1}$ ← r2$_n$, r2$_0$ ← CY, CY ← r2$_7$		↕	
	RLR	r2	2	8	r2$_{n-1}$ ← r2$_n$, r2$_7$ ← CY, CY ← r2$_0$		↕	
	SLL	r2	2	8	r2$_{n+1}$ ← r2$_n$, r2$_0$ ← 0, CY ← r2$_7$		↕	
	SLR	r2	2	8	r2$_{n-1}$ ← r2$_n$, r2$_7$ ← 0, CY ← r2$_0$		↕	
	SLLC	r2	2	8	r2$_{n+1}$ ← r2$_n$, r2$_0$ ← 0, CY ← r2$_7$	Carry	↕	
	SLRC	r2	2	8	r2$_{n-1}$ ← r2$_n$, r2$_7$ ← 0, CY ← r2$_0$	Carry	↕	
	DRLL	EA	2	8	EA$_{n+1}$ ← EA$_n$, EA$_0$ ← CY, CY ← EA$_{15}$		↕	
	DRLR	EA	2	8	EA$_{n-1}$ ← EA$_n$, EA$_{15}$ ← CY, CY ← EA$_0$		↕	
	DSLL	EA	2	8	EA$_{n+1}$ ← EA$_n$, EA$_0$ ← 0, CY ← EA$_{15}$		↕	
	DSLR	EA	2	8	EA$_{n-1}$ ← EA$_n$, EA$_{15}$ ← 0, CY ← EA$_0$		↕	
Jump instruction	JMP	word	3	10	PC ← word			
	JB		1	4	PC$_{n-0}$ ← B, PC$_{7-n}$ ← C			
	JR	word	1	10	PC ← PC + 1 + jdisp1			
	JRE	word	2	10	PC ← PC + 2 + jdisp			
	JEA		2	8	PC ← EA			
Call instruction	CALL	word	3	16	(SP − 1) ← (PC + 3)$_H$, (SP − 2) ← (PC + 3)$_L$, PC ← word			
	CALB		2	17	(SP − 1) ← (PC + 2)$_H$, (SP − 2) ← (PC + 2)$_L$, PC$_H$ ← B, PC$_L$ ← C			
	CALF	word	2	13	(SP − 1) ← (PC + 2)$_H$, (SP − 2) ← (PC + 2)$_L$, PC15 − 11 ← 00001, PC10 − 0 ← a			
	CALT	word	1	16	(SP − 1) ← (PC + 1)$_H$, (SP − 2) ← (PC + 1)$_L$, PC$_L$ ← (128 + 2a), PC$_H$ ← (129 + 2a)			
	SOFTI		1	16	(SP − 1) ← PSW, (SP − 2) ← (PC + 1)$_H$, (SP − 3) ← (PC + 1)$_L$, PC ← 003Bh			
Return instruction	RET		1	10	PC$_L$ ← (SP), PC$_H$ ← (SP + 1), SP ← SP + 2			
	RETS		1	10	PC$_L$ ← (SP), PC$_H$ ← (SP + 1), SP ← SP + 2, PC ← PC + a	無条件リターン		
	RETI		1	13	PC$_L$ ← (SP), PC$_H$ ← (SP + 1), PSW ← (SP + 2), SP ← SP + 3			
Skip instruction	BIT	bit, wa	2	10	Skip if (V, wa)bit = 1	(V, wa)bit = 1		
	SK	f	2	8	Skip if f = 1	f = 1		
	SKN	f	2	8	Skip if f = 0	f = 0		
	SKIT	irf	2	8	Skip if irf = 1 then reset irf	irf = 1		
	SKNIT	irf	2	8	Skip if irf = 0 otherwise reset irf	irf = 0		
CPU control instruction	NOP		1	4	No Operation			
	EI		1	4	Enable Interrupt			
	DI		1	4	Disable Interrupt			
	HLT		2	11	Halt			

NOTES: 1. In the expresions for bytes, rpa2, rpa3 to the right of the slash indicate D + byte and H + byte.

2. In the expresions for states, rpa2, rpa3 to the right of the slash indicate D + byte, H + A, H + B H + EA and H + byte.

TRANSFER AND ARITHMETIC LOGICAL OPERATION INSTRUCTION TABLE

8-bit transfer instruction

Addressing	A, rl rl, A	A, srl sr, A	r, word word, r	r, byte	sr2, byte	rpal, byte	wa	wa, byte	rpa2		
									Register, indirect	Autoincrement, autodecrement	Base
Mnemonic	MOV			MVI		MVIX	LDAW	MVIW	LDAX		
							STAW		STAX		

16-bit transfer instrction

Addressing	EA, rp3 rp3, EA	EA, sr4 sr3, EA	rp2, word	word	rpa3		
					Register indirect	Autoincrement, autodecrement	Base
Mnemonic	DMOV		LXI	LBCD LDED LHLD LSPD	LDEAX		
				SBCD SDED SHLD SSPD	STEAX		

Arithmetic logical operation instruction

Addressing		8-bit operation						16-bit operation		Skip condition
		A, r	r, A	A, byte r, byte sr2, byte	rpa (See Notes 1 and 2.)	wa (See Note 1.)	wa, byte	EA, rp3	EA, r2	
Arithmetic operation instruction		ADD		ADI	ADDX	ADDW		DADD	EADD	
		ADC		ACI	ADCX	ADCW		DADC		
		ADDNC		ADINC	ADDNCX	ADDNCW		DADDNC		No Carry
		SUB		SUI	SUBX	SUBW		DSUB	ESUB	
		SBB		SBI	SBBX	SBBW		DSBB		
		SUBNB		SUINB	SUBNBX	SUBNBW		DSUBNB		No Borrow
Logical operation instruction	Logic	ANA		ANI	ANAX	ANAW	ANIW	DAN		
		ORA		ORI	ORAX	ORAW	ORIW	DOR		
		XRA		XRI	XRAX	XRAW		DXR		
	Comparison	GTA		GTI	GTAX	GTAW	GTIW	DGT		1st operand > 2nd operand
		LTA		LTI	LTAX	LTAW	LTIW	DLT		1st operand < 2nd operand
	Match	NEA		NEI	NEAX	NEAW	NEIW	DNE		1st operand ≠ 2nd operand
		EQA		EQI	EQAX	EQAW	EQIW	DEQ		1st operand = 2nd operand
	Test	ONA		ONI	ONAX	ONAW	ONIW	DON		No Zero
		OFFA		OFFI	OFFAX	OFFAW	OFFIW	DOFF		Zero

NOTE 1. The 1st operand of skip condition is A (accumulator).
2. Register indirect and autoincrement/decrement possible.

Appendix II μCOM-87AD INSTRUCTION APPLICATION TABLE

μCOM-87AD machine language ⟷ mnemonic comparison table (alphabetic order)

ACI	A,byte	56××	EQAX	rpa	70F9-70FF	ONA	A,r	60C8-60CF
ACI	r,byte	7450××-7457××	EQI	A,byte	77××	ONAW	wa	74C8××
ACI	sr2,byte	6450-3,5-7;64D0,1,3,5××	EQI	r,byte	7478××-747F××	ONAX	rpa	70C9-70CF
ADC	A,r	60D0-60D7	EQI	sr2,byte	6478-B,D-F;64F8,9,B,D××	ONI	A,byte	47××
ADC	r,A	6050-6057	EQIW	wa,byte	75××××	ONI	r,byte	7448××-744F××
ADCW	wa	74D0××	ESUB	EA,r2	7061-7063	ONI	sr2,byte	6448-B,D-F;64C8,9,B,D××
ADCX	rpa	70D1-70D7	EXA		10	ONIW	wa,byte	45××××
ADD	A,r	60C0-60C7	EXH		50	ORA	A,r	6098-609F
ADD	r,A	6040-6047	EXX		11	ORA	r,A	6018-601F
ADDNC	A,r	60A0-60A7	GTA	A,r	60A8-60AF	ORAW	wa	7498××
ADDNC	r,A	6020-6027	GTA	r,A	6028-602F	ORAX	rpa	7099-709F
ADDNCW	wa	74A0××	GTAW	wa	74A8××	ORI	A,byte	17××
ADDNCX	rpa	70A1-70A7	GTAX	rpa	70A9-70AF	ORI	r,byte	7418××-741F××
ADDW	wa	74C0××	GTI	A,byte	27××	ORI	sr2,byte	6418-B,D-F;6498,9,B,D××
ADDX	rpa	70C1-70C7	GTI	r,byte	7428××-742F××	ORIW	wa,byte	15××××
ADI	A,byte	46××	GTI	sr2,byte	6428-B,D-F;64A8,9,B,D××	POP	rpl	A0-A4
ADI	r,byte	7440××-7447××	GTIW	wa,byte	25××××	PUSH	rpl	B0-B4
ADI	sr2,byte	6440-3,5-7;64C0,1,3,5××	HLT		483B	RET		B8
ADINC	A,byte	26××	INR	r2	41-43	RETI		62
ADINC	r,byte	7420××-7427××	INRW	wa	20××	RETS		B9
ADINC	sr2,byte	6420-3,5-7;64A0,1,3,5××	INX	EA	A8	RLD		4838
ANA	A,r	6088-608F	INX	rp	02,12,22,32	RLL	r2	4835-4837
ANA	r,A	6008-600F	JB		21	RLR	r2	4831-4833
ANAW	wa	7488××	JEA		4828	RRD		4839
ANAX	rpa	7089-708F	JMP	word	54××××	SBB	A,r	60F0-60F7
ANI	A,byte	07××	JR	word	C0-FF	SBB	r,A	6070-6077
ANI	r,byte	7408××-740F××	JRE	word	4E00-4FFF	SBBW	wa	74F0××
ANI	sr2,byte	6408-B,D-F;6488,9,B,D××	LBCD	word	701F××××	SBBX	rpa	70F1-70F7
ANIW	wa,byte	05××××	LDAW	wa	01××	SBCD	word	701E××××
BIT	bit,wa	58××-5F××	LDAX	rpa2	28-2F,A8××,AC-AE,AF××	SBI	A,byte	76××
BLOCK		31	LDEAX	rpa3	4882-5,B-F	SBI	r,byte	7470××-7477××
CALB		4829	LDED	word	702F××××	SBI	sr2,byte	6470-3,5-7;64F0,1,3,5××
CALF	word	7800-7FFF	LHLD	word	703F××××	SDED	word	702E××××
CALL	word	40××××	LSPD	word	700F××××	SHLD	word	703E××××
CALT	word	80-9F	LTA	A,r	60B8-60BF	SK	f	480A-480C
CLC		482A	LTA	r,A	6038-603F	SKIT	irf	4840-484C,4850-4854
DAA		61	LTAW	wa	74B8××	SKN	f	481A-481C
DADC	EA,rp3	74D5-74D7	LTAX	rpa	70B9-70BF	SKNIT	irf	4860-486C,4870-4874
DADD	EA,rp3	74C5-74C7	LTI	A,byte	37××	SLL	r2	4825-4827
DADDNC	EA,rp3	74A5-74A7	LTI	r,byte	7438××-743F××	SLLC	r2	4805-4807
DAN	EA,rp3	748D-748F	LTI	sr2,byte	6438-B,D-F;64B8,9,B,D××	SLR	r2	4821-4823
DCR	r2	51-53	LTIW	wa,byte	35××××	SLRC	r2	4801-4803
DCRW	wa	30××	LXI	rp2,word	04,14,24,34,44××××	SOFTI		72
DCX	EA	A9	MOV	r1,A	18-1F	SSPD	word	700E××××
DCX	rp	03,13,23,33	MOV	A,r1	08-0F	STAW	wa	63××
DEQ	EA,rp3	74FD-74FF	MOV	sr,A	4DC8-3,5-D;4DD8-A,7,8,A,B	STAX	rpa2	38-3F,B8××,BC-BE,BF××
DGT	EA,rp3	74AD-74AF	MOV	A,sr1	4CC0-3,5-9,B,D;4CD9;4CE0-3	STEAX	rpa3	4892-5,B-F
DI		BA	MOV	r,word	7068×××××-706F××××	STC		482B
DIV	r2	483D-483F	MOV	word,r	7078×××××-707F××××	SUB	A,r	60E0-60E7
DLT	EA,rp3	74BD-74BF	MUL	r2	482D-482F	SUB	r,A	6060-6067
DMOV	EA,rp3	A5-A7	MVI	sr2,byte	6400-3,5-7;6480,1,3,5××	SUBNB	A,r	60B0-60B7
DMOV	EA,sr4	48C0,48C1	MVI	r,byte	68××-6F××	SUBNB	r,A	6030-6037
DMOV	rp3,EA	B5-B7	MVIW	wa,byte	71××××	SUBNBW	wa	74B0××
DMOV	sr3,EA	48D2,48D3	MVIX	rpa1,byte	49××-4B××	SUBNBX	rpa	70B1-70B7
DNE	EA,rp3	74ED-74EF	NEA	A,r	60E8-60EF	SUBW	wa	74E0××
DOFF	EA,rp3	74DD-74DF	NEA	r,A	6068-606F	SUBX	rpa	70E1-70E7
DON	EA,rp3	74CD-74CF	NEAW	wa	74E8××	SUI	A,byte	66××
DOR	EA,rp3	749D-749F	NEAX	rpa	70E9-70EF	SUI	r,byte	7460××-7467××
DRLL	EA	48B4	NEGA		483A	SUI	sr2,byte	6460-3,5-7;64E0,1,3,5××
DRLR	EA	48B0	NEI	A,byte	67××	SUINB	A,byte	36××
DSBB	EA,rp3	74F5-74F7	NEI	r,byte	7468××-746F××	SUINB	r,byte	7430××-7437××
DSLL	EA	48A4	NEI	sr2,byte	6468-B,D-F;64E8,9,B,D××	SUINB	sr2,byte	6430-3,5-7;64B0,1,3,5××
DSLR	EA	48A0	NEIW	wa,byte	65××××	TABLE		48A8
DSUB	EA,rp3	74E5-74E7	NOP		00	XRA	A,r	6090-6097
DSUBNB	EA,rp3	74B5-74B7	OFFA	A,r	60D8-60DF	XRA	r,A	6010-6017
DXR	EA,rp3	7495-7497	OFFAW	wa	74D8××	XRAW	wa	7490××
EADD	EA,r2	7041-7043	OFFAX	rpa	70D9-70DF	XRAX	rpa	7091-7097
EI		AA	OFFI	A,byte	57××	XRI	A,byte	16××
EQA	A,r	60F8-60FF	OFFI	r,byte	7458××-745F××	XRI	r,byte	7410××-7417××
EQA	r,A	6078-607F	OFFI	sr2,byte	6458-B,D-F;64D8,9,B,D××	XRI	sr2,byte	6410-3,5-7;6490,1,3,5××
EQAW	wa	74F8××	OFFIW	wa,byte	55××××			

μCOM-87AD machine language ⟷ mnemonic comparison table (1/4)

00	NOP		40	CALL	word	4860	SKNIT	FNMI	4DC0	MOV	PA, A
01	LDAW	wa	41	INR	A	4861	SKNIT	FT0	4DC1	MOV	PB, A
02	INX	SP	42	INR	B	4862	SKNIT	FT1	4DC2	MOV	PC, A
03	DCX	SP	43	INR	C	4863	SKNIT	F1	4DC3	MOV	PD, A
04	LXI	SP, word	44	LXI	EA, word	4864	SKNIT	F2	4DC5	MOV	PF, A
05	ANIW	wa, byte	45	ONIW	wa, byte	4865	SKNIT	FE0	4DC6	MOV	MKH, A
06			46	ADI	A, byte	4866	SKNIT	FE1	4DC7	MOV	MKL, A
07	ANI	A, byte	47	ONI	A, byte	4867	SKNIT	FEIN	4DC8	MOV	ANM, A
08	MOV	A, EAH	4801	SLRC	A	4868	SKNIT	FAD	4DC9	MOV	SMH, A
09	MOV	A, EAL	4802	SLRC	B	4869	SKNIT	FSR	4DCA	MOV	SML, A
0A	MOV	A, B	4803	SLRC	C	486A	SKNIT	FST	4DCB	MOV	EOM, A
0B	MOV	A, C	4805	SLLC	A	486B	SKNIT	ER	4DCC	MOV	ETMM, A
0C	MOV	A, D	4806	SLLC	B	486C	SKNIT	OV	4DCD	MOV	TMM, A
0D	MOV	A, E	4807	SLLC	C	4870	SKNIT	AN4	4DD0	MOV	MM, A
0E	MOV	A, H	480A	SK	CY	4871	SKNIT	AN5	4DD1	MOV	MCC, A
0F	MOV	A, L	480B	SK	HC	4872	SKNIT	AN6	4DD2	MOV	MA, A
10	EXA		480C	SK	Z	4873	SKNIT	AN7	4DD3	MOV	MB, A
11	EXX		481A	SKN	CY	4874	SKNIT	SB	4DD4	MOV	MC, A
12	INX	B	481B	SKN	HC	4882	LDEAX	D	4DD7	MOV	MF, A
13	DCX	B	481C	SKN	Z	4883	LDEAX	H	4DD8	MOV	TXB, A
14	LXI	B, word	4821	SLR	A	4884	LDEAX	D++	4DDA	MOV	TM0, A
15	ORIW	wa, byte	4822	SLR	B	4885	LDEAX	H++	4DDB	MOV	TM1, A
16	XRI	A, byte	4823	SLR	C	488B	LDEAX	D+byte	4EXX	JRE	word
17	ORI	A, byte	4825	SLL	A	488C	LDEAX	H+A			
18	MOV	EAH, A	4826	SLL	B	488D	LDEAX	H+B	4FXX		
19	MOV	EAL, A	4827	SLL	C	488E	LDEAX	H+EA	50	EXH	
1A	MOV	B, A	4828	JEA		488F	LDEAX	H+byte	51	DCR	A
1B	MOV	C, A	4829	CALB		4892	STEAX	D	52	DCR	B
1C	MOV	D, A	482A	CLC		4893	STEAX	H	53	DCR	C
1D	MOV	E, A	482B	STC		4894	STEAX	D++	54	JMP	word
1E	MOV	H, A	482D	MUL	A	4895	STEAX	H++	55	OFFIW	wa, byte
1F	MOV	L, A	482E	MUL	B	489B	STEAX	D+byte	56	ACI	A, byte
20	INRW	wa	482F	MUL	C	489C	STEAX	H+A	57	OFFI	A, byte
21	JB		4831	RLR	A	489D	STEAX	H+B	58	BIT	0, wa
22	INX	D	4832	RLR	B	489E	STEAX	H+EA	59	BIT	1, wa
23	DCX	D	4833	RLR	C	489F	STEAX	H+byte	5A	BIT	2, wa
24	LXI	D, word	4835	RLL	A	48A0	DSLR	EA	5B	BIT	3, wa
25	GTIW	wa, byte	4836	RLL	B	48A4	DSLL	EA	5C	BIT	4, wa
26	ADINC	A, byte	4837	RLL	C	48A8	TABLE		5D	BIT	5, wa
27	GTI	A, byte	4838	RLD		48B0	DRLR	EA	5E	BIT	6, wa
28			4839	RRD		48B4	DRLL	EA	5F	BIT	7, wa
29	LDAX	B	483A	NEGA		48C0	DMOV	EA, ECNT	6008	ANA	V, A
2A	LDAX	D	483B	HLT		48C1	DMOV	EA, ECPT	6009	ANA	A, A
2B	LDAX	H	483D	DIV	A	48D2	DMOV	ETM0, EA	600A	ANA	B, A
2C	LDAX	D+	483E	DIV	B	48D3	DMOV	ETM1, EA	600B	ANA	C, A
2D	LDAX	H+	483F	DIV	C	49	MVIX	B, byte	600C	ANA	D, A
2E	LDAX	D−	4840	SKIT	FNMI	4A	MVIX	D, byte	600D	ANA	E, A
2F	LDAX	H−	4841	SKIT	FT0	4B	MVIX	H, byte	600E	ANA	H, A
30	DCRW	wa	4842	SKIT	FT1	4CC0	MOV	A, PA	600F	ANA	L, A
31	BLOCK		4843	SKIT	F1	4CC1	MOV	A, PB	6010	XRA	V, A
32	INX	H	4844	SKIT	F2	4CC2	MOV	A, PC	6011	XRA	A, A
33	DCX	H	4845	SKIT	FE0	4CC3	MOV	A, PD	6012	XRA	B, A
34	LXI	H, word	4846	SKIT	FE1	4CC5	MOV	A, PF	6013	XRA	C, A
35	LTIW	wa, byte	4847	SKIT	FEIN	4CC6	MOV	A, MKH	6014	XRA	D, A
36	SUINB	A, byte	4848	SKIT	FAD	4CC7	MOV	A, MKL	6015	XRA	E, A
37	LTI	A, byte	4849	SKIT	FSR	4CC8	MOV	A, ANM	6016	XRA	H, A
38			484A	SKIT	FST	4CC9	MOV	A, SMH	6017	XRA	L, A
39	STAX	B	484B	SKIT	ER	4CCB	MOV	A, EOM	6018	ORA	V, A
3A	STAX	D	484C	SKIT	OV	4CCD	MOV	A, TMM	6019	ORA	A, A
3B	STAX	H	4850	SKIT	AN4	4CD9	MOV	A, RXB	601A	ORA	B, A
3C	STAX	D+	4851	SKIT	AN5	4CE0	MOV	A, CR0	601B	ORA	C, A
3D	STAX	H+	4852	SKIT	AN6	4CE1	MOV	A, CR1	601C	ORA	D, A
3E	STAX	D−	4853	SKIT	AN7	4CE2	MOV	A, CR2	601D	ORA	E, A
3F	STAX	H−	4854	SKIT	SB	4CE3	MOV	A, CR3	601E	ORA	H, A

μCOM-87AD machine language ⟷ mnemonic comparison table (2/4)

601F	ORA	L, A	606F	NEA	L, A	60B7	SUBNB	A, L	60F7	SBB	A, L				
6020	ADDNC	V, A	6070	SBB	V, A	60B8	LTA	A, V	60F8	EQA	A, V				
6021	ADDNC	A, A	6071	SBB	A, A	60B9	LTA	A, A	60F9	EQA	A, A				
6022	ADDNC	B, A	6072	SBB	B, A	60BA	LTA	A, B	60FA	EQA	A, B				
6023	ADDNC	C, A	6073	SBB	C, A	60BB	LTA	A, C	60FB	EQA	A, C				
6024	ADDNC	D, A	6074	SBB	D, A	60BC	LTA	A, D	60FC	EQA	A, D				
6025	ADDNC	E, A	6075	SBB	E, A	60BD	LTA	A, E	60FD	EQA	A, E				
6026	ADDNC	H, A	6076	SBB	H, A	60BE	LTA	A, H	60FE	EQA	A, H				
6027	ADDNC	L, A	6077	SBB	L, A	60BF	LTA	A, L	60FF	EQA	A, L				
6028	GTA	V, A	6078	EQA	V, A	60C0	ADD	A, V	61	DAA					
6029	GTA	A, A	6079	EQA	A, A	60C1	ADD	A, A	62	RETI					
602A	GTA	B, A	607A	EQA	B, A	60C2	ADD	A, B	63	STAW	wa				
602B	GTA	C, A	607B	EQA	C, A	60C3	ADD	A, C	6400	MVI	PA, byte				
602C	GTA	D, A	607C	EQA	D, A	60C4	ADD	A, D	6401	MVI	PB, byte				
602D	GTA	E, A	607D	EQA	E, A	60C5	ADD	A, E	6402	MVI	PC, byte				
602E	GTA	H, A	607E	EQA	H, A	60C6	ADD	A, H	6403	MVI	PD, byte				
602F	GTA	L, A	607F	EQA	L, A	60C7	ADD	A, L	6405	MVI	PF, byte				
6030	SUBNB	V, A	6088	ANA	A, V	60C8	ONA	A, V	6406	MVI	MKH, byte				
6031	SUBNB	A, A	6089	ANA	A, A	60C9	ONA	A, A	6407	MVI	MKL, byte				
6032	SUBNB	B, A	608A	ANA	A, B	60CA	ONA	A, B	6408	ANI	PA, byte				
6033	SUBNB	C, A	608B	ANA	A, C	60CB	ONA	A, C	6409	ANI	PB, byte				
6034	SUBNB	D, A	608C	ANA	A, D	60CC	ONA	A, D	640A	ANI	PC, byte				
6035	SUBNB	E, A	608D	ANA	A, E	60CD	ONA	A, E	640B	ANI	PD, byte				
6036	SUBNB	H, A	608E	ANA	A, H	60CE	ONA	A, H	640D	ANI	PF, byte				
6037	SUBNB	L, A	608F	ANA	A, L	60CF	ONA	A, L	640E	ANI	MKH, byte				
6038	LTA	V, A	6090	XRA	A, V	60D0	ADC	A, V	640F	ANI	MKL, byte				
6039	LTA	A, A	6091	XRA	A, A	60D1	ADC	A, A	6410	XRI	PA, byte				
603A	LTA	B, A	6092	XRA	A, B	60D2	ADC	A, B	6411	XRI	PB, byte				
603B	LTA	C, A	6093	XRA	A, C	60D3	ADC	A, C	6412	XRI	PC, byte				
603C	LTA	D, A	6094	XRA	A, D	60D4	ADC	A, D	6413	XRI	PD, byte				
603D	LTA	E, A	6095	XRA	A, E	60D5	ADC	A, E	6415	XRI	PF, byte				
603E	LTA	H, A	6096	XRA	A, H	60D6	ADC	A, H	6416	XRI	MKH, byte				
603F	LTA	L, A	6097	XRA	A, L	60D7	ADC	A, L	6417	XRI	MKL, byte				
6040	ADD	V, A	6098	ORA	A, V	60D8	OFFA	A, V	6418	ORI	PA, byte				
6041	ADD	A, A	6099	ORA	A, A	60D9	OFFA	A, A	6419	ORI	PB, byte				
6042	ADD	B, A	609A	ORA	A, B	60DA	OFFA	A, B	641A	ORI	PC, byte				
6043	ADD	C, A	609B	ORA	A, C	60DB	OFFA	A, C	641B	ORI	PD, byte				
6044	ADD	D, A	609C	ORA	A, D	60DC	OFFA	A, D	641D	ORI	PF, byte				
6045	ADD	E, A	609D	ORA	A, E	60DD	OFFA	A, E	641E	ORI	MKH, byte				
6046	ADD	H, A	609E	ORA	A, H	60DE	OFFA	A, H	641F	ORI	MKL, byte				
6047	ADD	L, A	609F	ORA	A, L	60DF	OFFA	A, L	6420	ADINC	PA, byte				
6050	ADC	V, A	60A0	ADDNC	A, V	60E0	SUB	A, V	6421	ADINC	PB, byte				
6051	ADC	A, A	60A1	ADDNC	A, A	60E1	SUB	A, A	6422	ADINC	PC, byte				
6052	ADC	B, A	60A2	ADDNC	A, B	60E2	SUB	A, B	6423	ADINC	PD, byte				
6053	ADC	C, A	60A3	ADDNC	A, C	60E3	SUB	A, C	6425	ADINC	PF, byte				
6054	ADC	D, A	60A4	ADDNC	A, D	60E4	SUB	A, D	6426	ADINC	MKH, byte				
6055	ADC	E, A	60A5	ADDNC	A, E	60E5	SUB	A, E	6427	ADINC	MKL, byte				
6056	ADC	H, A	60A6	ADDNC	A, H	60E6	SUB	A, H	6428	GTI	PA, byte				
6057	ADC	L, A	60A7	ADDNC	A, L	60E7	SUB	A, L	6429	GTI	PB, byte				
6060	SUB	V, A	60A8	GTA	A, V	60E8	NEA	A, V	642A	GTI	PC, byte				
6061	SUB	A, A	60A9	GTA	A, A	60E9	NEA	A, A	642B	GTI	PD, byte				
6062	SUB	B, A	60AA	GTA	A, B	60EA	NEA	A, B	642D	GTI	PF, byte				
6063	SUB	C, A	60AB	GTA	A, C	60EB	NEA	A, C	642E	GTI	MKH, byte				
6064	SUB	D, A	60AC	GTA	A, D	60EC	NEA	A, D	642F	GTI	MKL, byte				
6065	SUB	E, A	60AD	GTA	A, E	60ED	NEA	A, E	6430	SUINB	PA, byte				
6066	SUB	H, A	60AE	GTA	A, H	60EE	NEA	A, H	6431	SUINB	PB, byte				
6067	SUB	L, A	60AF	GTA	A, L	60EF	NEA	A, L	6432	SUINB	PC, byte				
6068	NEA	V, A	60B0	SUBNB	A, V	60F0	SBB	A, V	6433	SUINB	PD, byte				
6069	NEA	A, A	60B1	SUBNB	A, A	60F1	SBB	A, A	6435	SUINB	PF, byte				
606A	NEA	B, A	60B2	SUBNB	A, B	60F2	SBB	A, B	6436	SUINB	MKH, byte				
606B	NEA	C, A	60B3	SUBNB	A, C	60F3	SBB	A, C	6437	SUINB	MKL, byte				
606C	NEA	D, A	60B4	SUBNB	A, D	60F4	SBB	A, D	6438	LTI	PA, byte				
606D	NEA	E, A	60B5	SUBNB	A, E	60F5	SBB	A, E	6439	LTI	PB, byte				
606E	NEA	H, A	60B6	SUBNB	A, H	60F6	SBB	A, H	643A	LTI	PC, byte				

μCOM-87AD machine language ⟷ mnemonic comparison table (3/4)

6 4 3 B	LTI	PD, byte	6 4 8 8	ANI	ANM, byte	6 9	MVI	A, byte	7 0 A 7	ADDNCX	H−
6 4 3 D	LTI	PF, byte	6 4 8 9	ANI	SMH, byte	6 A	MVI	B, byte	7 0 A 9	GTAX	B
6 4 3 E	LTI	MKH, byte	6 4 8 B	ANI	EOM, byte	6 B	MVI	C, byte	7 0 A A	GTAX	D
6 4 3 F	LTI	MKL, byte	6 4 8 D	ANI	TMM, byte	6 C	MVI	D, byte	7 0 A B	GTAX	H
6 4 4 0	ADI	PA, byte	6 4 9 0	XRI	ANM, byte	6 D	MVI	E, byte	7 0 A C	GTAX	D+
6 4 4 1	ADI	PB, byte	6 4 9 1	XRI	SMH, byte	6 E	MVI	H, byte	7 0 A D	GTAX	H+
6 4 4 2	ADI	PC, byte	6 4 9 3	XRI	EOM, byte	6 F	MVI	L, byte	7 0 A E	GTAX	D−
6 4 4 3	ADI	PD, byte	6 4 9 5	XRI	TMM, byte	7 0 0 E	SSPD	word	7 0 A F	GTAX	H−
6 4 4 5	ADI	PF, byte	6 4 9 8	ORI	ANM, byte	7 0 0 F	LSPD	word	7 0 B 1	SUBNBX	B
6 4 4 6	ADI	MKH, byte	6 4 9 9	ORI	SMH, byte	7 0 1 E	SBCD	word	7 0 B 2	SUBNBX	D
6 4 4 7	ADI	MKL, byte	6 4 9 B	ORI	EOM, byte	7 0 1 F	LBCD	word	7 0 B 3	SUBNBX	H
6 4 4 8	ONI	PA, byte	6 4 9 D	ORI	TMM, byte	7 0 2 E	SDED	word	7 0 B 4	SUBNBX	D+
6 4 4 9	ONI	PB, byte	6 4 A 0	ADINC	ANM, byte	7 0 2 F	LDED	word	7 0 B 5	SUBNBX	H+
6 4 4 A	ONI	PC, byte	6 4 A 1	ADINC	SMH, byte	7 0 3 E	SHLD	word	7 0 B 6	SUBNBX	D−
6 4 4 B	ONI	PD, byte	6 4 A 3	ADINC	EOM, byte	7 0 3 F	LHLD	word	7 0 B 7	SUBNBX	H−
6 4 4 D	ONI	PF, byte	6 4 A 5	ADINC	TMM, byte	7 0 4 1	EADD	EA, A	7 0 B 9	LTAX	B
6 4 4 E	ONI	MKH, byte	6 4 A 8	GTI	ANM, byte	7 0 4 2	EADD	EA, B	7 0 B A	LTAX	D
6 4 4 F	ONI	MKL, byte	6 4 A 9	GTI	SMH, byte	7 0 4 3	EADD	EA, C	7 0 B B	LTAX	H
6 4 5 0	ACI	PA, byte	6 4 A B	GTI	EOM, byte	7 0 6 1	ESUB	EA, A	7 0 B C	LTAX	D+
6 4 5 1	ACI	PB, byte	6 4 A D	GTI	TMM, byte	7 0 6 2	ESUB	EA, B	7 0 B D	LTAX	H+
6 4 5 2	ACI	PC, byte	6 4 B 0	SUINB	ANM, byte	7 0 6 3	ESUB	EA, C	7 0 B E	LTAX	D−
6 4 5 3	ACI	PD, byte	6 4 B 1	SUINB	SMH, byte	7 0 6 8	MOV	V, word	7 0 B F	LTAX	H−
6 4 5 5	ACI	PF, byte	6 4 B 3	SUINB	EOM, byte	7 0 6 9	MOV	A, word	7 0 C 1	ADDX	B
6 4 5 6	ACI	MKH, byte	6 4 B 5	SUINB	TMM, byte	7 0 6 A	MOV	B, word	7 0 C 2	ADDX	D
6 4 5 7	ACI	MKL, byte	6 4 B 8	LTI	ANM, byte	7 0 6 B	MOV	C, word	7 0 C 3	ADDX	H
6 4 5 8	OFFI	PA, byte	6 4 B 9	LTI	SMH, byte	7 0 6 C	MOV	D, word	7 0 C 4	ADDX	D+
6 4 5 9	OFFI	PB, byte	6 4 B B	LTI	EOM, byte	7 0 6 D	MOV	E, word	7 0 C 5	ADDX	H+
6 4 5 A	OFFI	PC, byte	6 4 B D	LTI	TMM, byte	7 0 6 E	MOV	H, word	7 0 C 6	ADDX	D−
6 4 5 B	OFFI	PD, byte	6 4 C 0	ADI	ANM, byte	7 0 6 F	MOV	L, word	7 0 C 7	ADDX	H−
6 4 5 D	OFFI	PF, byte	6 4 C 1	ADI	SMH, byte	7 0 7 8	MOV	word, V	7 0 C 9	ONAX	B
6 4 5 E	OFFI	MKH, byte	6 4 C 3	ADI	EOM, byte	7 0 7 9	MOV	word, A	7 0 C A	ONAX	D
6 4 5 F	OFFI	MKL, byte	6 4 C 5	ADI	TMM, byte	7 0 7 A	MOV	word, B	7 0 C B	ONAX	H
6 4 6 0	SUI	PA, byte	6 4 C 8	ONI	ANM, byte	7 0 7 B	MOV	word, C	7 0 C C	ONAX	D+
6 4 6 1	SUI	PB, byte	6 4 C 9	ONI	SMH, byte	7 0 7 C	MOV	word, D	7 0 C D	ONAX	H+
6 4 6 2	SUI	PC, byte	6 4 C B	ONI	EOM, byte	7 0 7 D	MOV	word, E	7 0 C E	ONAX	D−
6 4 6 3	SUI	PD, byte	6 4 C D	ONI	TMM, byte	7 0 7 E	MOV	word, H	7 0 C F	ONAX	H−
6 4 6 5	SUI	PF, byte	6 4 D 0	ACI	ANM, byte	7 0 7 F	MOV	word, L	7 0 D 1	ADCX	B
6 4 6 6	SUI	MKH, byte	6 4 D 1	ACI	SMH, byte	7 0 8 9	ANAX	B	7 0 D 2	ADCX	D
6 4 6 7	SUI	MKL, byte	6 4 D 3	ACI	EOM, byte	7 0 8 A	ANAX	D	7 0 D 3	ADCX	H
6 4 6 8	NEI	PA, byte	6 4 D 5	ACI	TMM, byte	7 0 8 B	ANAX	H	7 0 D 4	ADCX	D+
6 4 6 9	NEI	PB, byte	6 4 D 8	OFFI	ANM, byte	7 0 8 C	ANAX	D+	7 0 D 5	ADCX	H+
6 4 6 A	NEI	PC, byte	6 4 D 9	OFFI	SMH, byte	7 0 8 D	ANAX	H+	7 0 D 6	ADCX	D−
6 4 6 B	NEI	PD, byte	6 4 D B	OFFI	EOM, byte	7 0 8 E	ANAX	D−	7 0 D 7	ADCX	H−
6 4 6 D	NEI	PF, byte	6 4 D D	OFFI	TMM, byte	7 0 8 F	ANAX	H−	7 0 D 9	OFFAX	B
6 4 6 E	NEI	MKH, byte	6 4 E 0	SUI	ANM, byte	7 0 9 1	XRAX	B	7 0 D A	OFFAX	D
6 4 6 F	NEI	MKL, byte	6 4 E 1	SUI	SMH, byte	7 0 9 2	XRAX	D	7 0 D B	OFFAX	H
6 4 7 0	SBI	PA, byte	6 4 E 3	SUI	EOM, byte	7 0 9 3	XRAX	H	7 0 D C	OFFAX	D+
6 4 7 1	SBI	PB, byte	6 4 E 5	SUI	TMM, byte	7 0 9 4	XRAX	D+	7 0 D D	OFFAX	H+
6 4 7 2	SBI	PC, byte	6 4 E 8	NEI	ANM, byte	7 0 9 5	XRAX	H+	7 0 D E	OFFAX	D−
6 4 7 3	SBI	PD, byte	6 4 E 9	NEI	SMH, byte	7 0 9 6	XRAX	D−	7 0 D F	OFFAX	H−
6 4 7 5	SBI	PF, byte	6 4 E B	NEI	EOM, byte	7 0 9 7	XRAX	H−	7 0 E 1	SUBX	B
6 4 7 6	SBI	MKH, byte	6 4 E D	NEI	TMM, byte	7 0 9 9	ORAX	B	7 0 E 2	SUBX	D
6 4 7 7	SBI	MKL, byte	6 4 F 0	SBI	ANM, byte	7 0 9 A	ORAX	D	7 0 E 3	SUBX	H
6 4 7 8	EQI	PA, byte	6 4 F 1	SBI	SMH, byte	7 0 9 B	ORAX	H	7 0 E 4	SUBX	D+
6 4 7 9	EQI	PB, byte	6 4 F 3	SBI	EOM, byte	7 0 9 C	ORAX	D+	7 0 E 5	SUBX	H+
6 4 7 A	EQI	PC, byte	6 4 F 5	SBI	TMM, byte	7 0 9 D	ORAX	H+	7 0 E 6	SUBX	D−
6 4 7 B	EQI	PD, byte	6 4 F 8	EQI	ANM, byte	7 0 9 E	ORAX	D−	7 0 E 7	SUBX	H−
6 4 7 D	EQI	PF, byte	6 4 F 9	EQI	SMH, byte	7 0 9 F	ORAX	H−	7 0 E 9	NEAX	B
6 4 7 E	EQI	MKH, byte	6 4 F B	EQI	EOM, byte	7 0 A 1	ADDNCX	B	7 0 E A	NEAX	D
6 4 7 F	EQI	MKL, byte	6 4 F D	EQI	TMM, byte	7 0 A 2	ADDNCX	D	7 0 E B	NEAX	H
6 4 8 0	MVI	ANM, byte	6 5	NEIW	wa, byte	7 0 A 3	ADDNCX	H	7 0 E C	NEAX	D+
6 4 8 1	MVI	SMH, byte	6 6	SUI	A, byte	7 0 A 4	ADDNCX	D+	7 0 E D	NEAX	H+
6 4 8 3	MVI	EOM, byte	6 7	NEI	A, byte	7 0 A 5	ADDNCX	H+	7 0 E E	NEAX	D−
6 4 8 5	MVI	TMM, byte	6 8	MVI	V, byte	7 0 A 6	ADDNCX	D−	7 0 E F	NEAX	H−

μCOM-87AD machine language ↔ mnemonic comparison table (4/4)

70F1	SBBX	B	7438	LTI	V, byte	7478	EQI	V, byte	74F8	EQAW	wa
70F2	SBBX	D	7439	LTI	A, byte	7479	EQI	A, byte	74FD	DEQ	EA, B
70F3	SBBX	H	743A	LTI	B, byte	747A	EQI	B, byte	74FE	DEQ	EA, D
70F4	SBBX	D+	743B	LTI	C, byte	747B	EQI	C, byte	74FF	DEQ	EA, H
70F5	SBBX	H+	743C	LTI	D, byte	747C	EQI	D, byte	75	EQIW	wa, byte
70F6	SBBX	D−	743D	LTI	E, byte	747D	EQI	E, byte	76	SBI	A, byte
70F7	SBBX	H−	743E	LTI	H, byte	747E	EQI	H, byte	77	EQI	A, byte
70F9	EQAX	B	743F	LTI	L, byte	747F	EQI	L, byte	78	CALF	word
70FA	EQAX	D	7440	ADI	V, byte	7488	ANAW	wa	↓		
70FB	EQAX	H	7441	ADI	A, byte	748D	DAN	EA, B	7F		
70FC	EQAX	D+	7442	ADI	B, byte	748E	DAN	EA, D	80	CALT	word
70FD	EQAX	H+	7443	ADI	C, byte	748F	DAN	EA, H	↓		
70FE	EQAX	D−	7444	ADI	D, byte	7490	XRAW	wa	9F		
70FF	EQAX	H−	7445	ADI	E, byte	7495	DXR	EA, B	A0	POP	V
71	MVIW	wa, byte	7446	ADI	H, byte	7496	DXR	EA, D	A1	POP	B
72	SOFTI		7447	ADI	L, byte	7497	DXR	EA, H	A2	POP	D
7408	ANI	V, byte	7448	ONI	V, byte	7498	ORAW	wa	A3	POP	H
7409	ANI	A, byte	7449	ONI	A, byte	749D	DOR	EA, B	A4	POP	EA
740A	ANI	B, byte	744A	ONI	B, byte	749E	DOR	EA, D	A5	DMOV	EA, B
740B	ANI	C, byte	744B	ONI	C, byte	749F	DOR	EA, H	A6	DMOV	EA, D
740C	ANI	D, byte	744C	ONI	D, byte	74A0	ADDNCW	wa	A7	DMOV	EA, H
740D	ANI	E, byte	744D	ONI	E, byte	74A5	DADDNC	EA, B	A8	INX	EA
740E	ANI	H, byte	744E	ONI	H, byte	74A6	DADDNC	EA, D	A9	DCX	EA
740F	ANI	L, byte	744F	ONI	L, byte	74A7	DADDNC	EA, H	AA	EI	
7410	XRI	V, byte	7450	ACI	V, byte	74A8	GTAW	wa	AB	LDAX	D+byte
7411	XRI	A, byte	7451	ACI	A, byte	74AD	DGT	EA, B	AC	LDAX	H+A
7412	XRI	B, byte	7452	ACI	B, byte	74AE	DGT	EA, D	AD	LDAX	H+B
7413	XRI	C, byte	7453	ACI	C, byte	74AF	DGT	EA, H	AE	LDAX	H+EA
7414	XRI	D, byte	7454	ACI	D, byte	74B0	SUBNBW	wa	AF	LDAX	H+byte
7415	XRI	E, byte	7455	ACI	E, byte	74B5	DSUBNB	EA, B	B0	PUSH	V
7416	XRI	H, byte	7456	ACI	H, byte	74B6	DSUBNB	EA, D	B1	PUSH	B
7417	XRI	L, byte	7457	ACI	L, byte	74B7	DSUBNB	EA, H	B2	PUSH	D
7418	ORI	V, byte	7458	OFFI	V, byte	74B8	LTAW	wa	B3	PUSH	H
7419	ORI	A, byte	7459	OFFI	A, byte	74BD	DLT	EA, B	B4	PUSH	EA
741A	ORI	B, byte	745A	OFFI	B, byte	74BE	DLT	EA, D	B5	DMOV	B, EA
741B	ORI	C, byte	745B	OFFI	C, byte	74BF	DLT	EA, H	B6	DMOV	D, EA
741C	ORI	D, byte	745C	OFFI	D, byte	74C0	ADDW	wa	B7	DMOV	H, EA
741D	ORI	E, byte	745D	OFFI	E, byte	74C5	DADD	EA, B	B8	RET	
741E	ORI	H, byte	745E	OFFI	H, byte	74C6	DADD	EA, D	B9	RETS	
741F	ORI	L, byte	745F	OFFI	L, byte	74C7	DADD	EA, H	BA	DI	
7420	ADINC	V, byte	7460	SUI	V, byte	74C8	ONAW	wa	BB	STAX	D+byte
7421	ADINC	A, byte	7461	SUI	A, byte	74CD	DON	EA, B	BC	STAX	H+A
7422	ADINC	B, byte	7462	SUI	B, byte	74CE	DON	EA, D	BD	STAX	H+B
7423	ADINC	C, byte	7463	SUI	C, byte	74CF	DON	EA, H	BE	STAX	H+EA
7424	ADINC	D, byte	7464	SUI	D, byte	74D0	ADCW	wa	BF	STAX	H+byte
7425	ADINC	E, byte	7465	SUI	E, byte	74D5	DADC	EA, B	C0	JR	word
7426	ADINC	H, byte	7466	SUI	H, byte	74D6	DADC	EA, D	↓		
7427	ADINC	L, byte	7467	SUI	L, byte	74D7	DADC	EA, H	FF		
7428	GTI	V, byte	7468	NEI	V, byte	74D8	OFFAW	wa			
7429	GTI	A, byte	7469	NEI	A, byte	74DD	DOFF	EA, B			
742A	GTI	B, byte	746A	NEI	B, byte	74DE	DOFF	EA, D			
742B	GTI	C, byte	746B	NEI	C, byte	74DF	DOFF	EA, H			
742C	GTI	D, byte	746C	NEI	D, byte	74E0	SUBW	wa			
742D	GTI	E, byte	746D	NEI	E, byte	74E5	DSUB	EA, B			
742E	GTI	H, byte	746E	NEI	H, byte	74E6	DSUB	EA, D			
742F	GTI	L, byte	746F	NEI	L, byte	74E7	DSUB	EA, H			
7430	SUINB	V, byte	7470	SBI	V, byte	74E8	NEAW	wa			
7431	SUINB	A, byte	7471	SBI	A, byte	74ED	DNE	EA, B			
7432	SUINB	B, byte	7472	SBI	B, byte	74EE	DNE	EA, D			
7433	SUINB	C, byte	7473	SBI	C, byte	74EF	DNE	EA, H			
7434	SUINB	D, byte	7474	SBI	D, byte	74F0	SBBW	wa			
7435	SUINB	E, byte	7475	SBI	E, byte	74F5	DSBB	EA, B			
7436	SUINB	H, byte	7476	SBI	H, byte	74F6	DSBB	EA, D			
7437	SUINB	L, byte	7477	SBI	L, byte	74F7	DSBB	EA, H			

Flag operation

Operation						D6	D5	D4	D3	D2	D0
reg, memory			immediate		skip	Z	SK	HC	L1	L0	CY
ADD ADC SUB SBB DADD DADC DSUB DSBB EADD ESUB	ADDW ADCW SUBW SBBW	ADDX ADCX SUBX SBBX	ADI ACI SUI SBI			↕	0	↕	0	0	↕
ANA ORA XRA DAN DOR DXR	ANAW ORAW XRAW	ANAX ORAX XRAX	ANI ORI XRI	ANIW ORIW		↕	0	●	0	0	●
ADDNC SUBNB GTA LTA DADDNC DSUBNB DGT DLT	ADDNCW SUBNBW GTAW LTAW	ADDNCX SUBNBX GTAX LTAX	ADINC SUINC GTI LTI	GTIW LTIW		↕	↕	↕	0	0	↕
ONA OFFA DON DOFF	ONAW OFFAW	ONAX OFFAX	ONI OFFI	ONIW OFFIW		↕	↕	●	0	0	●
NEA EQA DNE DEQ	NEAW EQAW	NEAX EQAX	NEI EQI	NEIW EQIW		↕	↕	↕	0	0	↕
INR DCR	INRW DCRW					↕	↕	↕	0	0	●
DAA						↕	0	↕	0	0	↕
RLR, RLL, SLR, SLL DRLR, DRLL, DSLR, DSLL						●	0	●	0	0	↕
SLRC, SLLC						●	↕	●	0	0	↕
STC						●	0	●	0	0	1
CLC						●	0	●	0	0	0
			MVI A, byte			●	0	●	1	0	●
			MVI L, byte LXI H, word			●	0	●	0	1	●
					BIT SK SKN SKIT SKNIT	●	↕	●	0	0	●
					RETS	●	1	●	0	0	●
All other instructions						●	0	●	0	0	●

↕ ····· Affected (set or reset)
1 ····· Set
0 ····· Reset
● ····· Not affected

Appendix III μCOM-87AD MODE REGISTER APPLICATION TABLE

Special register operation instruction table

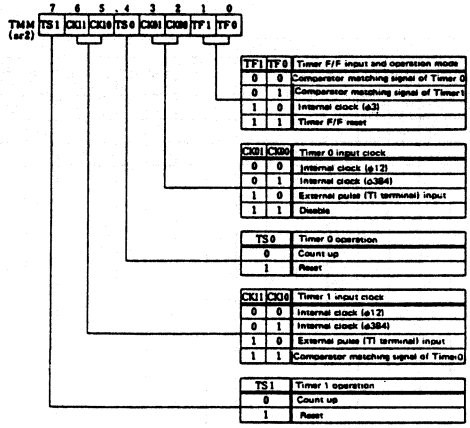
Instruction	Instruction code B1	Instruction code B2	Instruction code B3	Special register PA, PB, PC PD, PF	MKH MKL	ANM	SMH	SML	EOM	ETMM	TMM	MM	MCC	MA	MB	MC	MF	TXB	RXB	TM0 TM1	CR0 ↓ CR3
MOV sr, A	01001101	110S4S3S2S1S0		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○		○	
MOV A, sr1	01001100	11S4S3S2S1S0		○	○	○	○		○		○								○		○
MVI sr2,byte	01100100	S2000S2S1S0	Data	○	○	○	○		○		○										
ADI sr2,byte	01100100	S2100S2S1S0	Data																		
ACI sr2,byte		S2101S2S1S0																			
ADINC sr2,byte		S2010S2S1S0																			
SUI sr2,byte		S2110S2S1S0																			
SBI sr2,byte		S2111S2S1S0																			
SUINB sr2,byte		S2011S2S1S0																			
ANI sr2,byte		S2001S2S1S0		○	○	○	○		○		○										
ORI sr2,byte		S2011S2S1S0																			
XRI sr2,byte		S2010S2S1S0																			
GTI sr2,byte		S2101S2S1S0																			
LTI sr2,byte		S2111S2S1S0																			
NEI sr2,byte		S2101S2S1S0																			
EQI sr2,byte		S2111S2S1S0																			
ONI sr2,byte		S2001S2S1S0																			
OFFI sr2,byte		S2011S2S1S0																			

Special register table

Special register	sr—sr2	Code S4 S3 S2 S1 S0	sr	sr1	sr2
PORT A	PA	0 0 0 0 0	○	○	○
PORT B	PB	0 0 0 0 1	○	○	○
PORT C	PC	0 0 0 1 0	○	○	○
PORT D	PD	0 0 0 1 1	○	○	○
PORT F	PF	0 0 0 1 0 1	○	○	○
MASK High	MKH	0 0 0 1 1 0	○	○	○
MASK Low	MKL	0 0 0 1 1 1	○	○	○
A/D CHANNEL MODE	ANM	0 0 1 0 0 0	○	○	○
SERIAL MODE High	SMH	0 0 1 0 0 1	○	○	○
SERIAL MODE Low	SML	0 0 1 0 1 0	○		
TIMER/EVENT COUNTER OUTPUT MODE	EOM	0 0 1 0 1 1	○	○	○
TIMER/EVENT COUNTER MODE	ETMM	0 0 1 1 0 0	○		
TIMER MODE	TMM	0 0 1 1 0 1	○	○	○
MEMORY MAPPING	MM	0 1 0 0 0 0	○		
MODE CONTROL C	MCC	0 1 0 0 0 1	○		
MODE A	MA	0 1 0 0 1 0	○		
MODE B	MB	0 1 0 0 1 1	○		
MODE C	MC	0 1 0 1 0 0	○		
MODE F	MF	0 1 0 1 1 1	○		
Tx BUFFER	TXB	0 1 1 0 0 0	○		
Rx BUFFER	RXB	0 1 1 0 0 1		○	
TIMER REG 0	TM0	0 1 1 0 1 0	○		
TIMER REG 1	TM1	0 1 1 0 1 1	○		
A/D CONVERSION RESULT 0	CR0	1 0 0 0 0 0		○	
A/D CONVERSION RESULT 1	CR1	1 0 0 0 0 1		○	
A/D CONVERSION RESULT 2	CR2	1 0 0 0 1 0		○	
A/D CONVERSION RESULT 3	CR3	1 0 0 0 1 1		○	

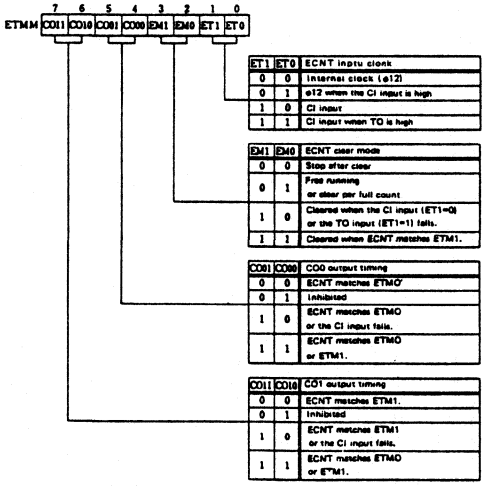
1. Timer

Timer mode register

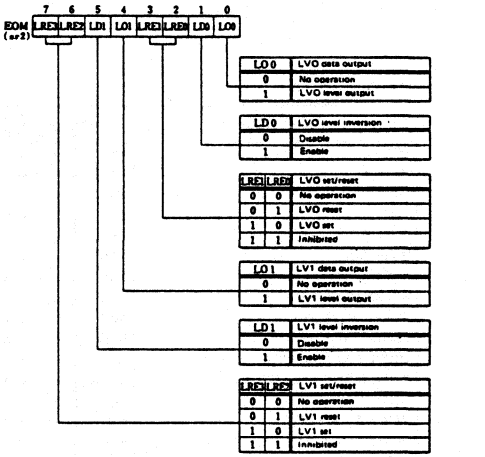


2. Timer/event counter

Timer/event counter mode register



Timer/event counter output mode register



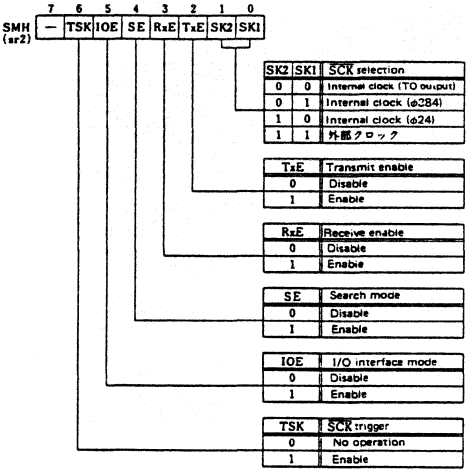
Note: To reset the output latch, reset the level output of level F/F (LV0, LV1) and level F/F (LV0, LV1).

Note: ECPT latches the ECNT contents:

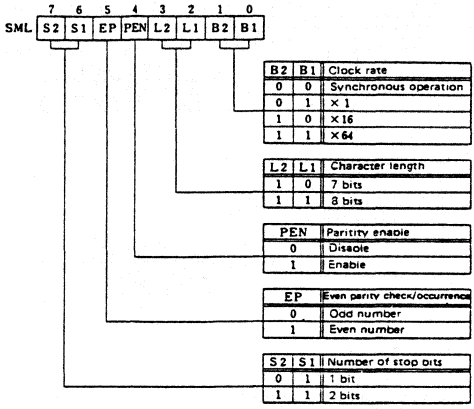
- o When the ET1 bit is 0 and the CI input falls.
- o When the ET1 bit is 1 and TO falls.

3. Serial interface

Serial mode high register



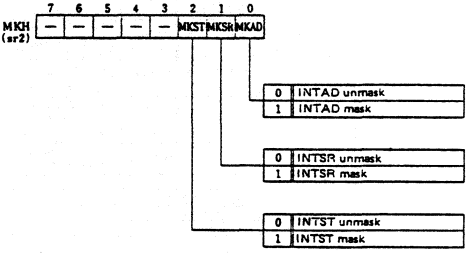
Serial mode low register



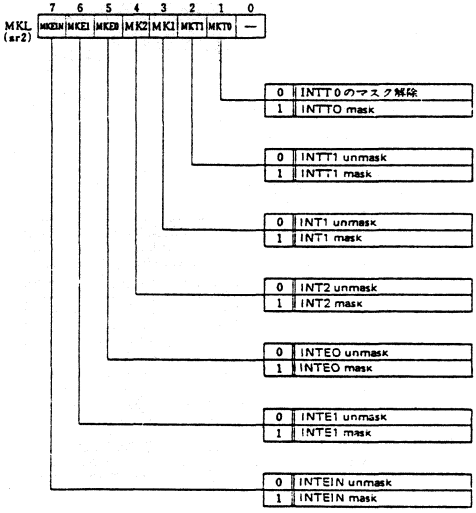
NOTE: Bit TSK is used to receive serial data in I/O interface mode.

4. Interrupt control

Interrupt mask high register



Interrupt mask low register



5. Port

Mode A register

7	6	5	4	3	2	1	0
MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0

MAn	PAn input/output
0	PAn : Output port
1	PAn : Input port

(n=0~7)

Mode B register

7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

MBn	PBn input/output
0	PBn : Output port
1	PBn : Input port

(n=0~7)

Mode C register

7	6	5	4	3	2	1	0
MC7	MC6	MC5	MC4	MC3	MC2	MC1	MC0

MCn	PCn input/output
0	PCn : Output port
1	PCn : Input port

(n=0~7)

Mode F register

7	6	5	4	3	2	1	0
MF7	MF6	MF5	MF4	MF3	MF2	MF1	MF0

MFn	PFn input/output
0	PFn : Output port
1	PFn : Input port

(n=0~7)

Mode control C register

7	6	5	4	3	2	1	0
MCC7	MCC6	MCC5	MCC4	MCC3	MCC2	MCC1	MCC0

0	PC0 : Port mode
1	PC0 : TxD output
0	PC1 : Port mode
1	PC1 : RxD input
0	PC2 : Port mode
1	PC2 : SCKinput/output
0	PC3 : Port mode
1	PC3 : INT2/T1 input
0	PC4 : Port mode
1	PC4 : T0 output
0	PC5 : Port mode
1	PC5 : CI input
0	PC6 : Port mode
1	PC6 : CO0 output
0	PC7 : Port mode
1	PC7 : CO1 output

Memory mapping register

7	6	5	4	3	2	1	0
—	—	—	—	RAE	MM2	MM1	MM0

MM2	MM1	MM0	PF and PF operation	
0	0	0	Single-chip	PD7-0 : Input port PF7-0 : Port mode
0	0	1		PD7-0 : Output port PF7-0 : Port mode
0	1	0	256 bytes	PD7-0 : Extension mode PF7-0 : Port mode
1	0	0	4K bytes	PD7-0 PF3-0 : Extension mode PF7-4 : Port mode
1	1	0	16K bytes	PF7-0 PD5-0 : Extension mode PF7-6 : Port mode
1	1	1	60K bytes	PD7-0 PF7-0 : Extension mode

RAE	Built-in RAM access
0	Disable
1	Enable

Note: Set bits MM0, MM1, and MM2 of the memory mapping register (MM) to 0 when μPD7810 is used.

6. Analog/digical converter

A/D channel mode register

7	6	5	4	3	2	1	0
—	—	—	FR	ANI2	ANI1	ANI0	MS

MS	Operating mode
0	Scan mode
1	Select mode

ANI2	ANI1	ANI0	Analog input
0	0	0	AN0
0	0	1	AN1
0	1	0	AN2
0	1	1	AN3
1	0	0	AN4
1	0	1	AN5
1	1	0	AN6
1	1	1	AN7

FR	Conversion speed specification
0	Oscillation frequency > 9MHz
1	Oscillation frequency < 9 MHz

CHAPTER 6 MONITOR FOR DEBUG OF μPD7816G

EVAKIT-87AD can debug μPD7816G by exchanging monitor PROM for μPD7816G. The monitor for 7816G consits of 4 PROMs of 2732 x 3 PCS for EVAKIT-87AD and 2716 x 1 for EV87AD RTT And the PROMs are attached to the EVAKIT at shipment.

6.1 Start-Up of Monitor For μPD7816G

(1) Exchange of Monitor PROM

Monitor PROMs for EVAKIT-87AD are taken off from IC 13 and 14 and 15 sockets and monitor PROMs for 7816 are put in the sockets.

(2) Setting of Jumpers and DIP switches

Setting of jumpers and DIP switches is in same way as in case of debug of μPD7811G mode).

6.2 Monitor Command For μPD7816G

In μPD7816G number of registers is less than in μPD7811G and the monitor for 7816 is different from that for 7811 as follows.

(1) Console Mode

Command	Monitor For EVAKIT-87AD	Monitor For 7816
CE	Change of Contents in EA Register	None
CA'	Change of Contents in V' and A' Registers	None
CB'	Change of Contents in B' and C' Registers	None
CD'	Change of Contents in D' and E' Registers	None
CH'	Change of Contents in H' and L' Registers	None
CE'	Change of Contents in EA' Register	None
ER	Exchange of Contents in Main Register and ALT Register	None
DR	Display of Contents in Register (PC,SP,EA,V, A,B,C,D,E,H,L,EA',V',A', B',C',D',E',H',L')	Display of Contents in Register (PC,SP,A,B,C,D, E,H,L)
DQ	Display of Contents in Special Register (MK,ANM,SMH,EOM,TMM,RXB, CR0,CR1,CR2,CR3,ECNT, ECPT)	Display of Contents in Special Register (MK,ANM,SMH,TMM,RXB,CR)
CQ	Change of Contents in Special Register (MK,SMH, SML,ANM,EOM,TMM,ETMM,MM, MCC,MA,MB,MC,MF,TXB,TM0, TM1,ETM0,ETM1)	Change of Contents in Special Register (MK,SMH, ANM,TMM,MM,MCC,MA,MB,MC, MF,TXB,TM0,TM1)
CA	Change Of Contents In V And A Registers	Change Of Contents In Accumulator

The other commands than the above ones are same in both monitors for evakit-87AD and 7816.

(2) Key-Board Mode

Code number of register is different in register operation command.

EVAKIT-87AD Monitor

Register Name	Code	Register Name	Code
V	(00)H	V'	(10)H
A	(01)H	A'	(11)H
B	(02)H	B'	(12)H
C	(03)H	C'	(13)H
D	(04)H	D'	(14)H
E	(05)H	E'	(15)H
H	(06)H	H'	(16)H
L	(07)H	L'	(17)H
EA	(08)H	EA'	(18)H
PC	(09)H		
SP	(0A)H		
PSW	(0B)H		

7816 Monitor

Register Name	Codo
A	(00)H
B	(01)H
C	(02)H
D	(03)H
E	(04)H
H	(05)H
L	(06)H
PC	(07)H
SP	(08)H
PSW	(09)H

The other code numbers than the above are same in both monitors for EVAKIT-87AD and 7816.

6.3 Note At Emulation of μPD7816G

If commands which the 7816 does not have differently from 7810 are written in program memory, the commands will be executed like usable commands since 7810 is used as evaluation chip on EVAKIT. Full care is needed. This problem can be solved by disassemble display with DA command and the verification before execution of program.

μCOM-87 AD

(EV87AD-RTT)

REAL TIME TRACER

USER'S MANUAL

CHAPTER 1 SYSTEM OVERVIEW

1.1 General

The EVAKIT-87AD Real-time tracer (EV87ADRTT), when connected to the EVAKIT-87AD, is used to trace the result of real-time emulation. This tracer enables efficient development of μCOM-87AD microcomputers.

The functions of the Real-time tracer are effective to:

- discover the cause of program overrun.
- perform debugging when the target hardware is connected but single step emulation cannot be performed due to external conditions.

1.2 Features

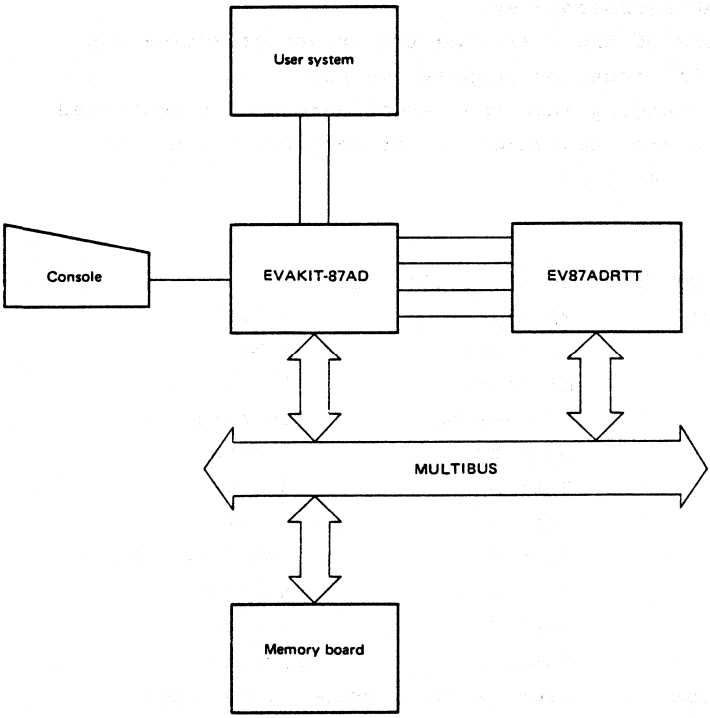
The EV87ADRTT features the following:

° Trace data	RD signal	1 bit
	WR signal	1 bit
	M1 signal	1 bit
	Address bus	16 bits
	Data bus	8 bits
	Port A	8 bits
	Port B	8 bits
	Port C	any 3 of 8 bits
	Port D	8 bits
	Port F	8 bits
	Total	62 bits
° Trace steps	Machine cycle mode	1,019 steps
	Instruction mode	1,022 steps

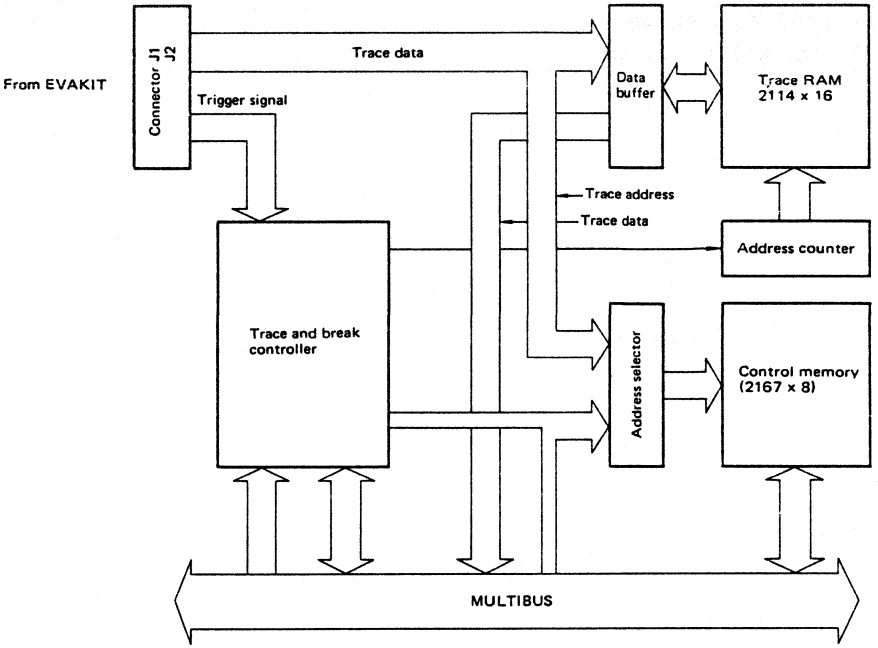
° Expanded break function
- Address break
- Timer break
- Step break

CHAPTER 2 SYSTEM CONFIGURATION

2.1 System Contiguration



2.2 EV87ADRTT Block Diagram



2.3 EV87ADRTT Specifications

2.3.1 Main LSIs

Trace RAM	: μPD2114LC5 x 16
Trace break control RAM	: μPD2167D-3 x 8
I/O port	: μPD8255AC-5 x 1
Timer/counter	: μPD8253C-5 x 1
Monitor ROM	: μPD2716D x 1

2.3.2 Environmental Conditions

Temperature	: 0° to 40°C
Storage	: -10° to 50 °C

2.3.3 Power Requirement

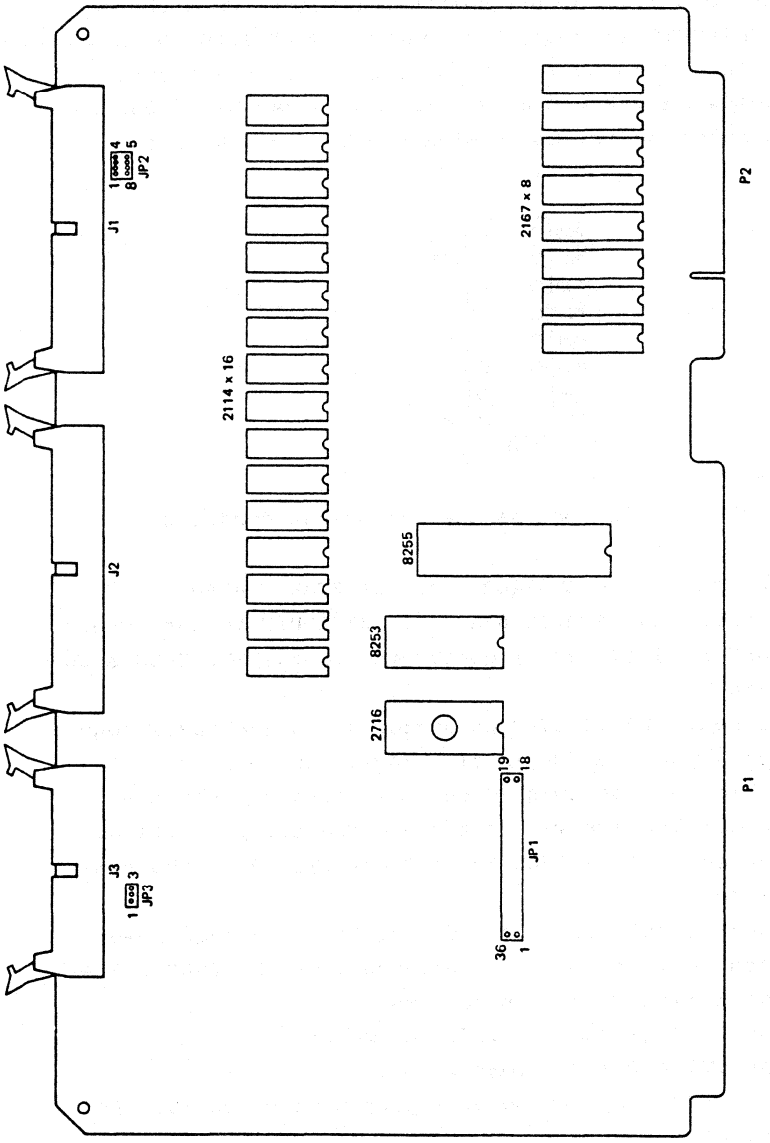
DC : +5V, ±5%, 3A (max.)

2.3.4 External Dimensions

305 (W) x 170 (L) mm

CHAPTER 3 BEFORE OPERATION

3.1 External View of EV87ADRTT

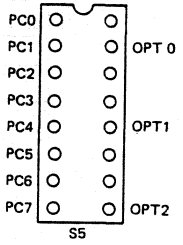


3.2 System Connection

3.2.1 Setting the EVAKIT-87AD DIP switches and jumpers

(1) Set pin No.1 of DIP switch SN1 to OFF when the EV87ADRTT is connected to the EVAKIT-87AD for tracing.

(2) EVAKIT-87AD socket S5 is as shown in the figure below. Any 3 of the 8 Port C bits can be traced by connecting option terminals OPT 0 to 3 as shown below.



Note:

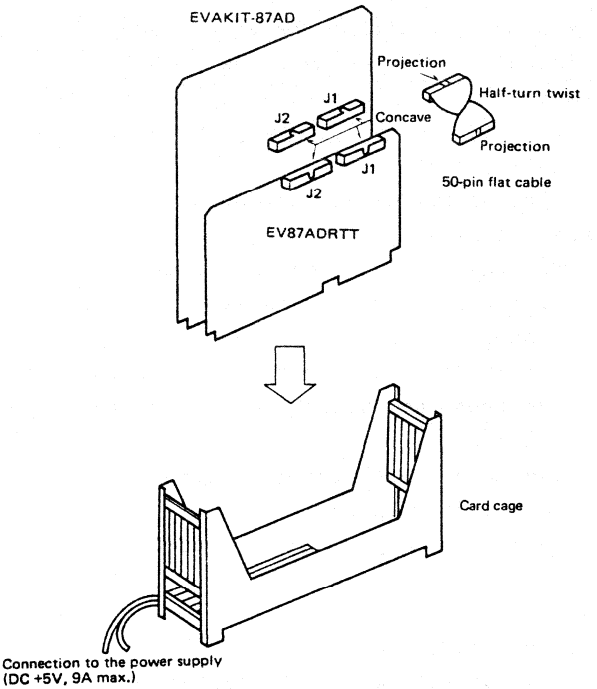
Socket S5 is open as the factor-set condition.

3.2.2 Connecting the EV87ADRTT to the EVAKIT-87AD

(1) Connect the EVAKIT-87AD and EV87ADRTT to card cages (NEC SB-9017, etc.) conforming to the MULTIBUS standard.

(2) Connect EVAKIT-87AD connector J1 to EV87ADRTT connector J1 with a 50-pin flat cable. The 50-pin flat cable must be twisted one half turn so that the header concave will fit the connector projection. Connect EVAKIT-87AD J2 to EV87ADRTT J2 in the same manner.

(3) Power should be supplied from the card cage power terminal. The power requirements of the EVAKIT-87AD and EV87ADRTT are as follows:
For EVAKIT-87AD: +5 VDC, 6A (max.)
For EV87ADRTT: 3A (max.)
Therefore, system power supply must be DC +5V, 9A (max.)



Connecting Diagram of EVAKIT-87AD and EV87ADRTT

3.3 Commands

This section explains the commands added to those already resident in the EVAKIT-87AD when the EV87ADRTT is connected.

3.3.1 Input format

(1) All input is hexadecimal. If the input value is a 2-byte value, the low-order four digits are recognized. For a 1-byte value, the low-order two digits are recognized. If the input value is less than four digits, the value will be left-justified as shown below.

When four digits are recognized:
If 10000 is input, it will be taken as 0000.
If 123 is input, it will be taken as 0123.

(2) If the item to be input is already stored in the EVAKIT-87AD, it can be reinput by pressing the space key (indicated by "Δ").

(3) Plural data are delimited by commas.

(4) Input of special keys

CTRL/C	Terminates execution and returns control to the monitor command level (input wait state).
CTRL/S	Temporarily halts output to the console.
CTRL/Q	Restarts the output that was halted by CTRL/S.
"↵"	Indicates C_R (carriage return).
"Δ"	Indicates a space.
~~	Indicates output from monitor.

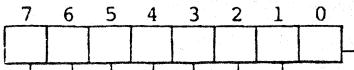
3.3.2 Trace command

(1) TF: Trace format setting

When "TF" is input following "*", "ΔXX-" will be output indicating the current trace format. When this message is output, you should input the desired trace format as a hexadecimal value.

When the new trace format has been correctly input and followed by "↵", the current trace format will be changed, line feed/carriage return will be performed and "*" will be output by the monitor to indicate that it has returned to command level (command input wait state).

Each bit of the 2-digit hexadecimal value for trace format setting is as follows:



Trace cycle
0: Machine cycle
1: Instruction

Port A trace data
0: Port A trace data not displayed.
1: Port A trace data displayed.

Port B trace data
0: Port B trace data not displayed.
1: Port B trace data displayed.

Port D trace data
0: Port D trace data not displayed.
1: Port D trace data displayed.

Port F trace data
0: Port F trace data not displayed.
1: Port F trace data displayed.

Option terminal trace data
0: Option terminal trace data not displayed.
1: Option terminal trace data displayed.

Disassembler list
0: Disassembler list not displayed.
1: Disassembler list displayed.

Port trace data output format
0: HEX format
1: BINARY format

Example:

*TF Δ00-7E ↵

(2) TD Trace data display

When TD followed by "↵" is input from the console after "*" has been output, "-" will be output by the monitor, prompting you to input the number of steps of trace data which you wish to display. When the number of trace steps has been correctly input and followed by "↵", line feed/carriage return will be performed and the specified number of steps of trace data will be displayed in the format specified by the TF command.

When the display of trace data has completed, "*" will be output by the monitor. When a value for display of trace data is input which exceeds the actual number of steps of trace performed, all of the trace data in the trace data RAM will be displayed.*

The maximum number of steps which can be traced in the machine cycle mode is 1,019 and in the instruction mode is 1,022.

Example:

```
*TD=6↵
ADRS DT M1 RD WR   PA   PB   MNEM.  OPERAND
0000 69  1  1  0   FF   FF   MVI    A,00H
0001 00  0  1  0   FF   FF
0002 4D  1  1  0   FF   FF   MOV    MA,A
0003 D2  0  1  0   FF   FF
0004 4D  1  1  0   FF   FF   MOV    MB,A
0005 D3  0  1  0   FF   FF
```

NOTE:

* If a value greater than the actual number of trace steps is input as a parameter for the TD command for the first trace operation after reset, garbage data may be displayed before the trace data.

3.3.3 Expansion of the break command

When the EV87ADRTT is connected to the EVAKIT-87AD, following 3 break modes will be added to the break command.

(1) Break Mode 40H Trace Address Break
This mode is essentially the same as break mode 00H except that the high-order 6 bits of each address need not agree.
Break loop condition may be set in the range 0000H to FFFFH. (If 0H is set, break will occur after the first loop.)

(2) Break Mode 41H Trace Timer Break
This mode is used to set the break condition according to the value of the timer on the EV87ADRTT board. In this mode, the number of break points cannot be set. Also note that break performed by the timer is independent of the address, data and mask conditions. Any value in the range 0000H to FFFFH (1 to FFFFH ms) may be set. A setting value of 0 will disable the timer break and time of timer is set with 'BL'.

(3) Break Mode 42H Trace Step Break
In this mode, trace will be performed for only the specified number of steps. The number of break points cannot be set. Break performed in this mode is independent of the address, data and mask conditions.
The number of steps is set by the BL command. The range of setting values is 0000H to FFFFH.
The number of steps cycles will be the number of trace cycles set at that time.

3.4 EV87ADRTT Operation

The EV87ADRTT Real-time tracer performs trace from the time that the EVACHIP starts execution of the user program (GB command) until break is applied. During this time, the signal states of the address and data buses, ports and control signals are traced continuously. Timing for trace is provided by either the RD or WR signals becoming active and trace can be performed per machine cycle or per instruction.

Determination of addresses for storage of these trace data is made by an incrementing counter. When this counter has made a complete loop, the maximum number of steps of trace data will have been written to the trace RAM.
The maximum number of trace steps is 1,019 in the machine cycle mode and 1,022 in the instruction mode. If trace is performed for a greater number of steps than is indicated by each of these values, the address counter will reset and write of trace data to the memory will continue from the first address. Therefore, if trace is performed for more than the specified maximum number of steps, only the most recent 1,019 or 1,022 steps of trace data will be retained, with the older data being progressively rewritten.

CHAPTER 4 OPERATION

4.1 Power ON to Monitor Start-up

(1) Before turning power ON, reconfirm that all connections in the system are correctly made (refer to Section 3.2). The DIP switches and jumpers of the EVAKIT-87AD should be set in accordance with the instructions of the EVAKIT-87AD operating manual.

(2) Apply power to the system. As the EVAKIT-87AD and the EV87ADRTT together draw a maximum current of 9A, be sure to use a large-capacity lead wire to prevent voltage drop. When power is supplied, the following message will be displayed on the screen of the EVAKIT monitor to indicate the current setting of the DIP switches.

```
EVAKIT-87AD MONITOR  VER.1.1

 MODE1 MODE0 RAE MM2 MM1 MM0     MF
   X     X     X   X   X   X   XXXXXXXX
IN    -  nK
USER  -  mK
BUS   -  yyyy
```

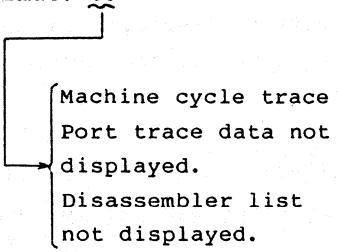
After this message has been displayed for a few seconds, the "*" prompt will appear indicating that the EVAKIT-87AD is at command level (command input wait state). If the "*" prompt is not displayed, or if it is displayed immediatedly after the above message appears, it could mean that pin No.1 of DIP switch SN1 pin is not OFF or that the connection of the MULTIBUS connector P1 is faulty.

4.2 Command Examples

(1) To execute the user program for the specified number of steps and to display the trace data.

First, set break mode 42H (trace step break). Then use the BL command to input the number of steps of the user program which you wish to execute. After using the TF command to set the trace format, use the GB command to start program execution. When the break is applied, the TD command is used to display the trace result data.

```
*BM 23,01 - 42          Trace Step Break
 BP 1 0000-0000
 BL 0002 - 6            Number of trace steps: 6
*TF 00 - 00             Trace format: 00
```




```
*GB-0
 RUN

     PC    INSTRUCTION MNEM.  OPERAND
   0006    4D D4       MOV    MC,A
•

                ZSHLLC V  A  B  C  D  E  H  L   EA   SP
                000000 8E 00 61 13 BD DD FF FF 2AF8 0400
```

```
*TD-6
   ADRS DT M1 RD WR
   0000 69  1  1  0
   0001 00  0  1  0
   0002 4D  1  1  0   } 6 steps (machine cycles)
   0003 D2  0  1  0
   0004 4D  1  1  0
   0005 D3  0  1  0
```

In the Trace Step Break mode, break is applied when the specified number of steps has been executed, regardless of the address or data. It is therefore useful for diagnosing program overrun.

(2) Example of delay break

Delay break is useful when the execution path beyond a certain address is not known. By setting the last known execution address as the break point and attaching a delay of several steps to the break point, the executed addresses beyond that point can be checked.

```
*BM23,01 - 10,1,5     Parallel break with 5-steps delay
 BP 1 0000-C          Break point: address 000C
 BL 01 - 01           Break loop: 1 loop

*TF 00 - 7F           Instruction trace
                      Port trace data displayed
                      Disassembler list displayed.
                      Port trace data output format: hex.
*GB-0
 RUN

    PC   INSTRUCTION MNEM.  OPERAND
  010C   69 03       MVI    A,03H
•

              ZSHLLC V  A  B  C  D  E  H  L   EA   SP
              000000 8E F0 01 00 AD DD ED B8 2AB8 0400

*TD-FF
   ADRS DT M1 RD WR  PA  PB  PD  PF  OPT MNEM.  OPERAND
   0000 69  1  1  0  01  00  FF  FF  000 MVI    A,0FFH
   0002 4D  1  1  0  01  00  FF  FF  000 MOV    MA,A
   0004 4D  1  1  0  01  00  FF  FF  111 MOV    MB,A
   0006 4C  1  1  0  01  00  FF  FF  111 MOV    A,PA
   0008 1A  1  1  0  01  00  FF  FF  111 MOV    B,A
   0009 4C  1  1  0  01  00  FF  FF  111 MOV    A,PB
   000B 1B  1  1  0  01  00  FF  FF  111 MOV    C,A
   000C 21  1  1  0  01  00  FF  FF  111 JB
   0100 69  1  1  0  01  00  FF  FF  111 MVI    A,0FH
   0102 4D  1  1  0  01  00  FF  FF  111 MOV    MM,A
   0104 70  1  1  0  01  00  FF  FF  111 MOV    0FF00H,D
   0108 69  1  1  0  01  00  FF  FF  111 MVI    A,0F0H
   010A 4D  1  1  0  01  00  FF  FF  111 MOV    PC,A
```

(3) Parallel break

Parallel break is useful in ascertaining which among several possible branch destination addresses is actually executed. By setting the branch destination addresses as break points, break will be applied at the address which is actually branched to.

```
*BM 23,01 - 00,3     Parallel break with 3 break points
 BP 1 0000-100       Break point 1: address 0100
 BP 2 0000-110       Break point 2: address 0110
 BP 3 0000-10        Break point 3: address 0010
 BL 01-01
```

```
*TF 00 - C3          Instruction trace
                     Port A data displayed
                     Disassembler list displayed
                     Port trace data output format: hex.
```

```
*GB-0
 RUN

     PC   INSTRUCTION MNEM.  OPERAND
   0110   FF          JR     0110H
.

              ZSHLLC V  A  B  C  D  E  H  L   EA   SP
              000000 00 01 55 11 A5 58 ED F9 2AB8 0400
```

```
*TD-F
   ADRS DT M1 RD WR    PA      MNEM.  OPERAND
   0000 69  1  1  0 00000001 MVI    A,0FFH
   0002 4D  1  1  0 00000001 MOV    MA,A
   0004 6A  1  1  0 00000001 MVI    B,55H
   0006 4C  1  1  0 00000001 MOV    A,PA
   0008 60  1  1  0 00000001 NEA    A,B
   000A 54  1  1  0 00000001 JMP    0100H
   000D 60  1  1  0 00000001 LTA    B,A
   000F 54  1  1  0 00000001 JMP    0110H
   0110 FF  1  1  0 00000001 JR     0110H
```

See NOTE on the next page.

NOTE:

* When the skip condition has been set, the following instruction is skipped and no internal operation is performed by the CPU; it merely waits for the number of idle states specified by the instruction. Note, however, that address and RD signals will be output normally so that in terms of the trace data, the result will be exactly the same as if the command were actually executed.

4.3 Display of trace data

(1) Vertically arranged instructions (MVI A, byte, MVI L, byte, LXI H, word)

When a series of instructions vertically arranged at the beginning of a program are executed, only the first instruction will be executed internally by the CPU. The remainder of the series of vertically arranged instructions will not be executed. Instead, the CPU will idle for the amount of time that would be required to execute each of the instructions without performing any operation. However, the CPU will continue to output address and read signals so that the result in terms of trace data will be exactly the same as if all of the instructions were actually executed.

```
*TD-B
   ADRS DT M1 RD WR    PA    MNEM.  OPERAND
   0004 6F  1  1  0    00    MVI    L,02H
   0006 6F  1  1  0    FF    MVI    L,03H   ┐
   0008 6F  1  1  0    FF    MVI    L,04H   ├ CPU idle
   000A 6F  1  1  0    FF    MVI    L,05H   ├
   000C 6F  1  1  0    FF    MVI    L,06H   ├
   000E 6F  1  1  0    FF    MVI    L,07H   ┘
   0010 69  1  1  0    FF    MVI    A,00H
   0012 4D  1  1  0    FF    MOV    MA,A
   0014 0F  1  1  0    FF    MOV    A,L
   0015 4D  1  1  0    FF    MOV    PA,A
   0017 FF  1  1  0    02    JR     0017H
```

(2) Skip command

When the skip command is executed, the skip condition is set, and the following command is skipped, the CPU will idle without performing any actual operation for the amount of time specified by that instruction. However, as in the case of vertically arranged instructions, the address and RD signals will continue to be output normally so that the result in terms of trace data will be exactly the same as if the instruction were actually executed.

APPENDIX

List of Parts Provided

- 50-pin flat cable (length: 20 cm) x2
- EV87ADRTT Operating Manual x1
- List of parts provided x1

Book 2

THE μCOM-87 FAMILY

IN-CIRCUIT EMULATOR

USER'S MANUAL

SECTION A

STAND ALONE

IE-7809-M

IE-7811-M

IE-78C11-M

IE-87AD-M

1.1 PREFACE

This manual describes the operation of IE-78llH-M Stand Alone System as a basis for an overall description for the operation of the following other IE-Systems:

IE-78Cll-M, IE-7809-M, IE-87AD-M

The IE-78Cll-M is used for emulation of uPD78C10/C11/C14 and IE-7809-M is used for emulation of uPD7807/08/09.
An Appendix to this manual describes general appearance of IE-78Cll-M and IE-7809-M plus variations in appearance and operation from IE-78llH-M.
The IE-87AD-M is used for emulation of uPD7810/11, but with an oscillator of 12 MHz instead of 15 MHz used in the IE-78llH-M.

CHAPTER 1 GENERAL

1.2 Introduction

The IE-7811H (an in-circuit emulator for the μCOM-7811H) is a hardware and software development support system for systems employing the μCOM-7811H as the CPU.
The IE-7811H consists of a driver, an emulation probe, and an IE control board and can emulate almost all the functions of the μCOM-7811H. In addition to various emulation, break, trace, and memory mapping functions described in this manual, the IE-7811H is provided with a function to communicate with the host system (MD series) for development of NEC microcomputers. Therefore, future system expansion can easily be carried out. In addition to the μCOM-7811H, the IE-7811H can also debug the μPD7810H and μPD7811H.

1.3 IE-7811H Specifications

IE-7811H specifications (stand-alone base)

CPU to be emulated		μPD7810H, μPD7811H	for other IE-Systems difficult, see APPEND.
CPU clock frequency		15MHz max.	
Memory mapping	User memory	Mapping in 256-byte units is possible in a 64K-byte address space (0 to FFFFH).	
	IE-7811H's internal memory (provided as standard)	Mapping (real-time execution) in 256-byte units is possible in the physical address space (64K bytes).	
Break	Break registers	1. Logical break registers (x4): BR0 to BR3 Physical break registers can be specified with these four registers. 2. Physical break registers (1) BRA An address, condition or data can be set. (2) BRD (Breaks by external sense clip data.) (3) BRE (Number of fetched OP codes) (4) BRT (Break timer) 3. Break mode register BRM Sets the break mode registers to hardware.	
	Setting conditions	1. Address Number of break points (two or more) Whether the break point is maskable and the range can be specified (within 64K bytes) 2. Condition OP code fetch, memory read, memory write, memory request, or unconditional.	

		3. Data Number of break points (two or more), whether the break point is maskable. 4. Number of OP code fetches (3 to FFFFH) 5. Number of loops (1 to FFH) 6. Timer (1 to FFFFH ms) 7. External sense data
	Break sources	1. Address 2. Data 3. Conditions (OP code fetch (M1 read), memory read, memory write, memory request, or unconditional) 4. External sense data 5. Instruction counter 6. Timer 7. Illegal (mapping breaks when an area other than that mapped is accessed) 8. Forced break (break by key input)
Trace	Tracing capacity	1023 frames
	Trace display	Instruction or machine cycle (selectable for display)
	Trace ON/OFF condition	One of the following is selected. 1. NON (will not trace) 2. ALL (all executions done by emulation CPU are traced) 3. TRX (traces according to the conditions specified by TRX)
	Contents of tracing	Address (16 lines)/data (8 lines)/ M1/RD/WR/External sense clips (8 lines)
External trace probes		8
Serial channels		2 (RS-232C) 110 to 9600 baud 1. TTY1 (for console) 25-pin connector 2. TTY2 (for sub I/O) 16-pin connector
Others		On-line assembling possible

(1) Emulation probe	Direct interfacing by inserting into 64-pin IC socket.
(2) CPU	μPD7810H (15MHz) this varies for other IE-System
(3) Signal pin interface	Ports D and F: With one HSCMOS buffer
(4) Control output signal	Real-time control during emulation CPU execution
(5) $\overline{\text{RESET}}$ and NMI signal inputs	Fully emulated
(6) CPU clock (varies for other IE-Systems)	4 to 15MHz (when an external clock is selected) 15MHz (when the internal clock is selected)
(7) MODE0, MODE1	Emulated

1.3.2 Break

All break functions are realized by hardware.

(1) Break condition — Execution of internal monitor program

(2) Number of break functions — five

(a) Break by three independent combinations of address, data, and control signals* (the number of loops can be specified).

(b) Break by accessing unspecified memory area.

(c) Break by conditions set by the eight external sense signals.

(d) Break by internal timer.

(e) Forced break by console operation.

* OPCODE FETCH (M1 read), MEMORY READ, MEMORY WRITE

1.3.3 Trace

(1) Three trace modes	(a) Nontrace
	(b) Trace by instruction
	(c) Trace by machine cycle
(2) Two trace operations	(a) Normal trace
	(b) Trace within specified address range
(3) Trace items	(a) Address and data buses
	(b) $\overline{M1}$, $\overline{RD}$, and $\overline{WR}$
	(c) Eight external sense signals

1.3.4 Mapping

(1) Mapping conditions	(a) Non-mapping (In the initial condition, the program memory is not mapped.)
	(b) External mapping (USER)
	(c) Internal mapping (IE)
(2) Area to be mapped	All 64K bytes of internal memory
(3) Mapping units	In 256-byte units.
(4) Internal memory resource	Program memory

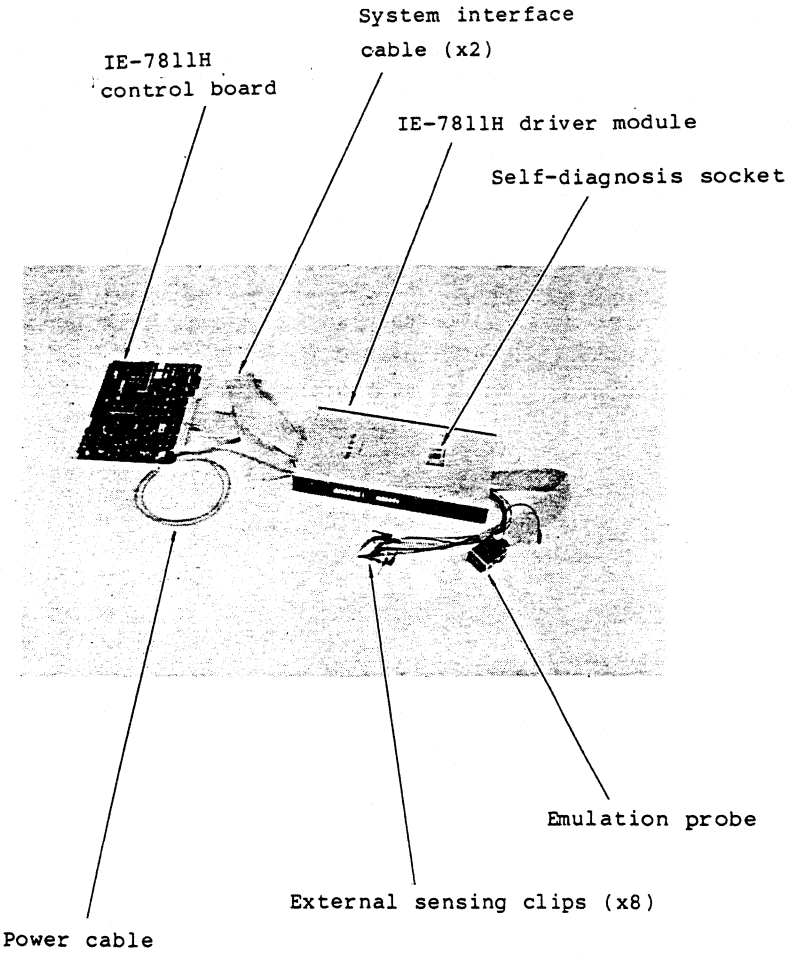
1.3.5 Other functions

(1) Multi IE function

Up to two IE-781lHs (or a combination of other IE systems and IE-781lHs) can be inserted into the slots of the host system (MD-086 series) so that more than one user can be accommodated. The MD-086EX expansion chassis is required for the MD-086FD. However, the chassis is not required for the MD-086FD-10.

1.4 IE-7811H Hardware Specifications

1.4.1 Appearance



1.4.2 Basic specifications

(1) Outline dimensions

IE control board	Depth	: 250mm
	Width	: 305mm
	Height	: Approx. 20mm
IE-781lH driver	Depth	: 400mm
	Width	: 230mm
	Height	: 48mm

(2) Weight

IE control board	550g
IE-781lH driver	2.6Kg

(3) Current

6.5A min. (+5V)
500mA min. (±12V)

(4) Operating temperature	10 to 40°C
(5) Storage temperature	-40 to +55°C
(6) Ambient humidity	10 to 90% (without condensation)

1.4.3 IE-7811H driver module block diagram

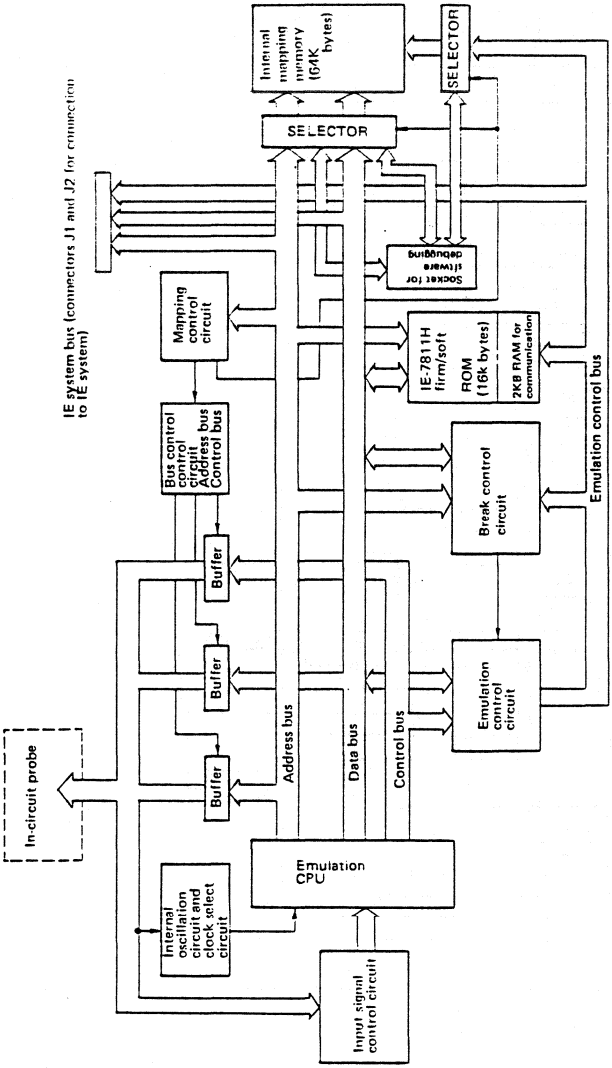


Fig. 1-1 IE-7811H Driver Block Diagram

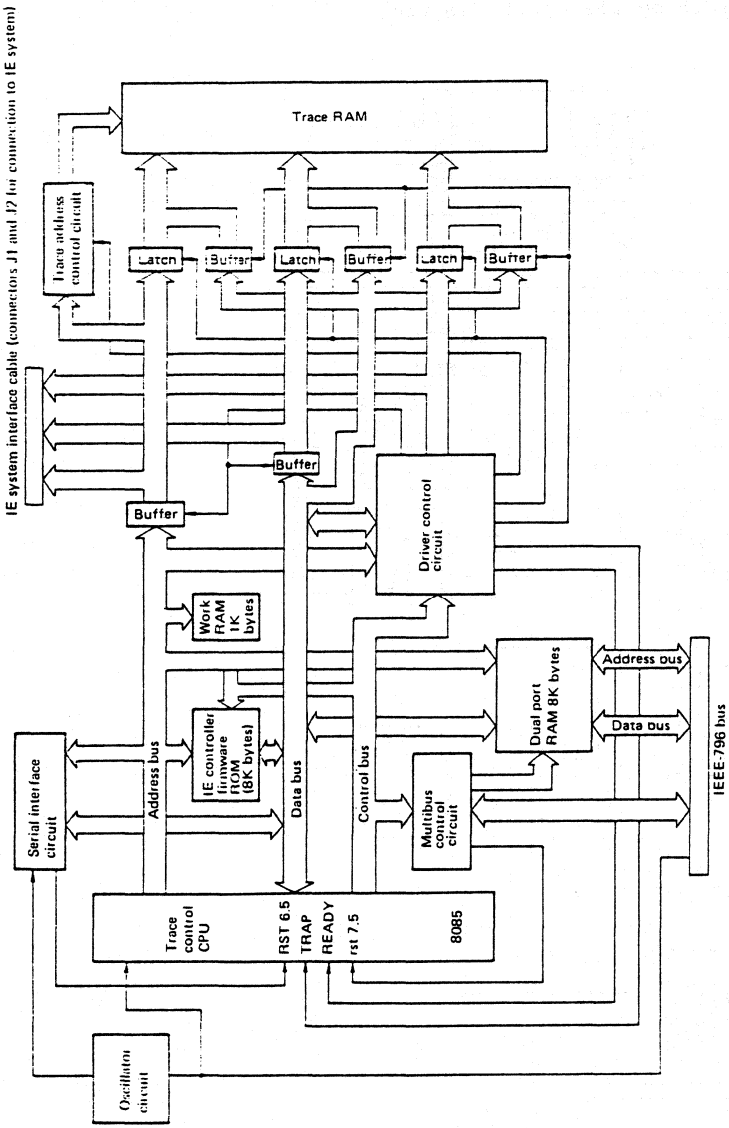
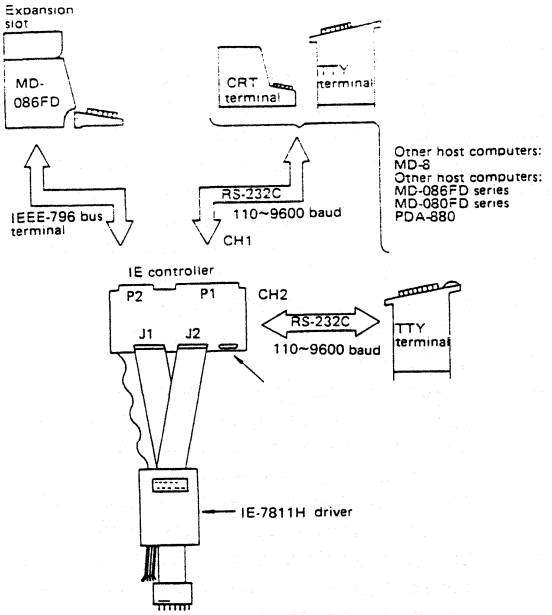


Fig. 1-2 IE Control Board Block Diagram

1.4.4 IE-7811H system configuration

The IE-7811H is used in a system with the configuration shown in Fig. 1-3.



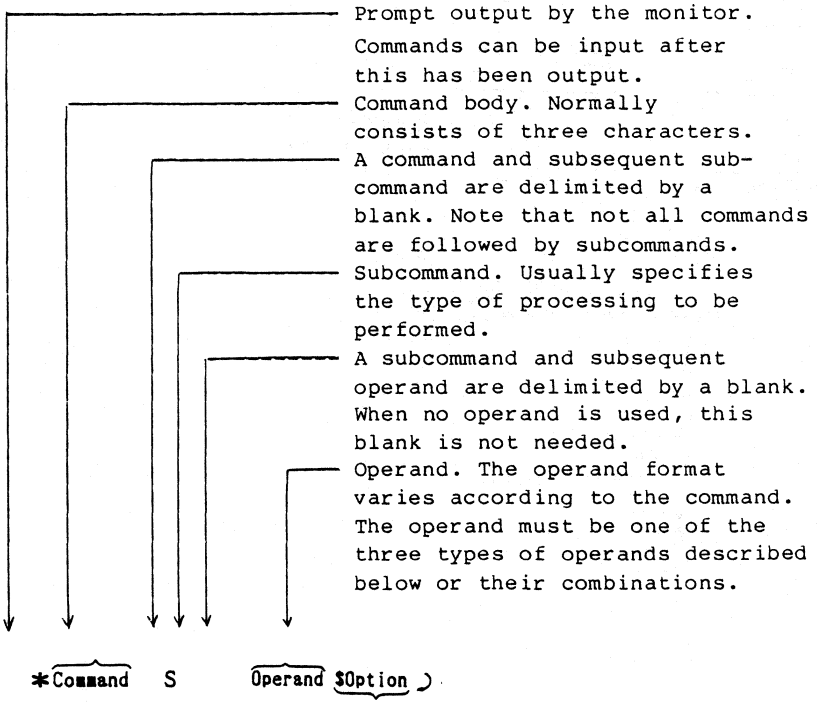
1.5 IE-7811H Software Specifications

1.5.1 Outline

The standard monitor program for the IE-7811H is provided on four EPROMs (μPD2764 x 1, μPD2716 x 2, μPD27128 x 1). Of these, the μPD27128 is mounted on the IE-7811H driver board, and the μPD2764 and two μPD2716s are mounted on the IE control board. The monitor program can be executed when power is applied to the system after it has been set up.

1.5.2 Command format

(1) As a rule, commands are input in the one-line format shown here. Inputting LF (line feed) indicates completion of command input.



Types of operands

(a) Keywords	Specific character strings that specify the processing to be performed.
(b) Numeric values	Specific numeric values or expressions.
(c) Numeric series	A series of numeric values. Processing is normally performed on the entire series.

2) In the command descriptions that follow, this format will be used:

Uppercase character	Indicates the actual key input of a character string or character.
Lowercase character	Input should be made to that location according to the explanation made by the lowercase characters (refer to Operands).
↵	Indicates line feed/carriage return.
␣*	Indicates a blank input.
[Character string ⋮ ⋮]	Indicates the user must select from among the character strings listed between braces.
{Character string ⋮ ⋮}	Indicates the user has the option of selecting one of the strings listed between the brackets or of making no input at all. The process performed when nothing is input is explained in the description of each command.

* Some of the descriptions use ony spaces instead of this symbol.

(3) Special characters

CTRL-H	Deletes one character (cursor moves one space left)
CTRL-U	Deletes one line (but still shown on screen)
CTRL-X	Deletes one line (erased from screen)
CTRL-E	Physical execution of [CR] [LF] command is not performed*
CTRL-R	Duplicates display of the input line
DEL	Deletes one character (cursor moves one space left)
CTRL-M	CR/LF
CTRL-J	CR/LF
CTRL-C	Aborts command execution (When the emulation CPU is operating, however, execution does not break and control returns to the "*" display.)
ESC	Same as CTRL-C
CTRL-S	Temporarily halts output
CTRL-Q	Restarts output
CTRL-P	Outputs to CH1 and CH2 simultaneously**
CTRL-D	Stops screen display (Screen display stops when this is input; however, executions are still performed. Inputting this character again restarts the screen display.)

*When CTRL-E is input during command execution, line feed/carriage return occurs on the screen. Command input, however, can continue.
By performing line feed at appropriate places, long command lines are made easier to read.

**Input CTRL-P to CH1 when a hardcopy of the trace results is desired; after CTRL-P is input from CH1, all subsequent output to CH1 will also be output to CH2. To cancel this, input CTRL-P again.

(4) Number of characters that can be input by key

A maximum 124 characters (including blanks) can be input.

1.5.3 Command list

(1) Main commands

MAP ; Mapping command
MEM ; Memory command
REG ; Register command
MDR ; Mode register command
SPR ; Special register command
LOD ; Load command
SAV ; Save command
RUN ; Emulation command
BR? ; Break command
(?; A, D, E, T, 0, 1, 2, 3, M)
TR? ; Trace command
(?; C, D, M, P, X, S)
CLK ; Clock command
RES ; Reset command
MAT ; Arithmetic operation command
SUF ; Numeric expression specification command
ASM ; On-line assemble command
DAS ; Disassemble command
MAV ; Memory transfer command
DIG ; Self-diagnosis command
PGM ; PROM programmer control command

(2) Subcommands

B ; Break
C ; Change, Chip
D ; Display, Dump
E ; Examination
F ; Fill
G ; Get
H ; Hard
I ; IE, Instruction
K ; Kill
M ; Move, Mode
N ; Normal
R ; internal ROM

S ; Step, Select
T ; Trace
U ; User
V ; Verify
W ; internal r/W memory
X ; eXchange

(3) Operands

addr	Numeric value that can be input by formula (2 bytes)
addrx	"X" numeric value that can be input (2 bytes)
partition	"X" numeric value that can be input
string	1-byte HEX data string delimited by commas
register-name	Register name
mode-register-name	Mode register name
special-register-name	Special register name
input-device-name	Input device name
output-device-name	Output device name
value	Numeric data (1 byte)
values	"X" numeric value that can be input (1 byte)
cond	Break condition Two or more conditions can be set.
step	Number of steps (1 byte)
time	Time unit (2 bytes) (M=msec)
data	1-byte data
line-No.	Number of trace lines
loop	Number of loops (1 byte)
count	Count value (2 bytes)

(4) Options

Options are always input following $.

° Memory command

(E: test) A: Simplified test

° Load command bias: Address bias value

° Emulate command

(T: trace) D: Display disable
(disassembled results will not be output)

E: Display enable
(disassembled results will be output, default value)

R: Register display

1.4.4 Command input format

(1) Mapping command

*MAP { R / W / U / K } { partition } ↵

R: Mapping to the IE-781lH's internal ROM
W: Mapping to the IE-781lH's internal RAM
U: Mapping to the user side
K: Releases mapping

(2) Memory command

*MEM C (addr) ↵
*MEM D (partition) ↵
*MEM E partition ($A) ↵

Changes memory (Change)
Displays memory (Dump)
Tests memory (Examination)
$A: Simplified test

*MEM { F / G } partition string ↵

F: Initializes memory (Fill)
G: Searches memory (Get)

*MEM { M / V / X } partition addr ↵

M: Transfers memory contents (Move)
V: Compares memory contents (Verify)
X: Exchanges memory contents (eXchange)

(3) Register command

```
*REG [ { C } ( register-name ) ]
       { D }                    ↵
```

C: Changes register
D: Displays register

(4) Mode register command

```
*MDR [ { C ( mode-register-name1) } ]
       { D ( mode-register-name  ) }  ↵
```

C: Changes mode register
D: Displays mode register

(5) Special register command

```
*SPR [ { C ( special-register-name1 ) } ]
       { D ( special-register-name2 ) }  ↵
```

C: Changes
D: Displays

(6) Load command

```
*LOD [ { = } input-device-name ] ( $bias) ↵
```

(7) Save command

```
*SAV ( output-device-name ) [ { = } partition ] ↵
```

(8) Emulation command

```
*RUN [ { N } ( addr ) ]
       { B }          ↵
```

N: Real-time execution without break
B: Real-time execution with break

```
*RUN S ( addr) (,step-No. ) ↵
```

S: Real-time step execution

```
*RUN T [ addr] [, { step-No. } ] [$ { D } ] [$R] ↵
                  {   ※     }       { E }
```

```
                   { =  }
                   { >  }
                   { <  }
                   { =< }
※=register-name    { >= }  value
                   { <> }
                   { >< }
                   { => }
                   { <= }
```

T: 1-step execution
D: Does not output disassembler
E: Outputs disassembler
R: Displays register

(9) Break command

```
*BRA [ A=addrx] [ V=values ]
        [ C=cond] [ L=loop] ↵
```

Sets address, data, condition, and loop

```
*BRD [ data] ↵
```

Sets external sense data

```
*BRE [ count ]↵
```

Sets number of opcode fetches

```
*BRT [ time]↵
```

Sets break timer

```
*BR { 0 } [ BRA.BRD-----] ↵
    { ~ }
    { 3 }
```

Sets physical break register to logical break register

```
*BRM [ BR0.BR1-----] ↵
```

Sets break register

(10) Trace command

```
*TRX [ A=addrx] [ V=values ] [ C=cond]
        [ PA=values ] [ PB=values ]↵
```

Sets address, data, condition, and port

```
*TRC [ { M } ] ↵
     [ { I } ]
```

Specifies display trace cycle
M: Machine cycle
I: Instruction

```
*TRD [ { ALL      } ] ↵
     [ { line-No. } ]
     [ { -line-No.} ]
```

Displays trace data
ALL: Dumps all

*TRD [{ O / N / line-No. / -line-No. }] ↵

Changes trace pointer
O: Old
N: New

*TRM [{ NON / ALL / TRX }] ↵

Specifies trace mode
NON: Does not trace
ALL: Traces all
TRX: Breaks according to the conditions of TRX command, and traces the specified address range

*TRS [{ PB / EX }] ↵

Trace selection
PB: Port B
EX: External port

(11) Clock command

*CLK [{ I / U }] ↵

Specifies CPU clock
I: Internal clock
U: External clock

(12) Reset command

*RES [H] ↵

H: Resets IE-7811H

(13) Arithmetic operation command

*MAT expression ↵

(14) Base specification command

*SUF [- { H / T / Q / Y }] ↵

H: HEX

T: Decimal
Q: Octal
Y: Binary

(15) Assemble command

*ASM [addr] ↵

(16) Disassemble command

*DAS [partition] ↵

(17) Memory transfer command

*MOV { U / I } partition addr ↵

U: Transfers memory contents of inside the IE to the user memory

I: Transfers user memory contents to the memory in the IE

(18) Self-diagnostic command

*DIG ↵

(19) PROM programmer command

*PGM ↵

1.6 Using the IE-7811H

1.6.1 Outline

In this system, the IE-7811H controller module performs all interfacing with the external system. To interface with the MD-086 host system, an IEEE-796 bus is provided. An RS-232C interface (CH1) is provided for standard external interfacing. In addition, another RS-232C port (CH2) is provided for use as a subchannel. The IE-7811H can be used in the following two ways:

(1) The IE-7811H is inserted in the multibus expansion slot of the host system.

(2) CH1 is used to connect the IE-7811H with the terminal as a stand-alone.

The monitor can be used for both; the desired mode is selected using the DIP switch setting described below. In this manual, a detailed description is given of the use of the IE-7811H as a stand-alone. For details of that application, refer to the operating manual for the system software, sold separately.

1.6.2 Using the IE-7811H as a stand-alone

Stand-alone mode is selected as the factory setting of the jumpers and DIP switches of the IE-7811H controller module; the baud rate of the serial channel is 4,800bps, and handshaking is performed under hardware control. Refer to Chapter 4 for details on the serial interface.
When using the IE-7811H as a stand-alone, it is recommended that the SB-9007 (IE controller case with power supply) be used.

1.6.3 Using the IE-7811H as a host subsystem

For users who have both MD-086 and IE-7811H systems, it is recommended that the separately available IE system software purchased. That way, the MD-086 can be used as an extremely powerful debugger for μPD7811H. Also, object programs for the μCOM-7811H created on the MD-086 system can be sent directly to the IE-7811H and programs in the IE-7811H can be stored as object file.

These are features of the IE system software:

(1) Symbolic debugging

(2) Object files on disk can be directly used by the IE system.

(3) By taking advantage of the multitasking functions provided, a single operator can operate multiple IE systems from a single console.

(4) Work from creating the source program to assembling and debugging can be done in a sequential manner.

1.6.4 Development procedure

A flowchart for typical system development is shown in Fig. 1-5.

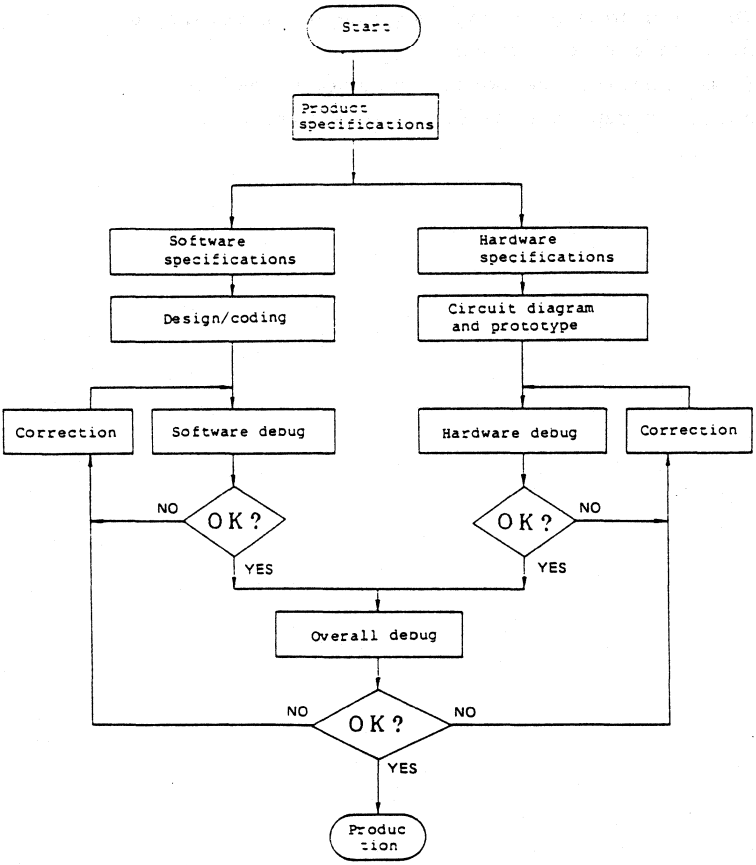


Fig. 1-5 Development Flow

Note: The IE-7811H can also be used to perform purely software-oriented debugging, i.e., debugging of software that is not hardware-dependent. Because there is no target hardware, therefore, the emulation target probe should be inserted in the self-diagnostic socket on the driver module. Both MODE0 and MODE1 should be set to 1.
The following settings should be made.
(1) Only the portion of the memory that will actually be used should be internally mapped. The rest of the memory should not be mapped.
(2) The clock should be fixed to 15MHz. (12MHz for IE-78C11 or IE-7809).

CHAPTER 2 INSTALLATION

2.1 Outline

This chapter discusses procedures for unpacking and setting up the IE-7811H system, and simple operation.

2.2 Unpacking

The IE-7811H is packed as shown in Fig. 2-1 and Fig. 2-2.

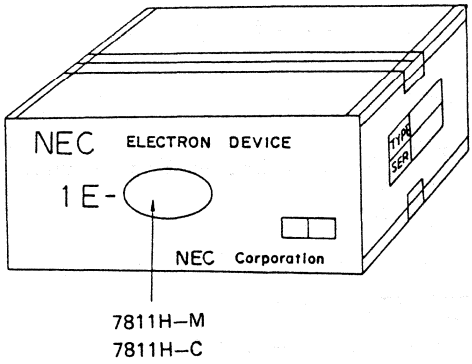


Fig. 2-1 Outside Appearance of Packing Carton

Note: The packaging method is subject to change.

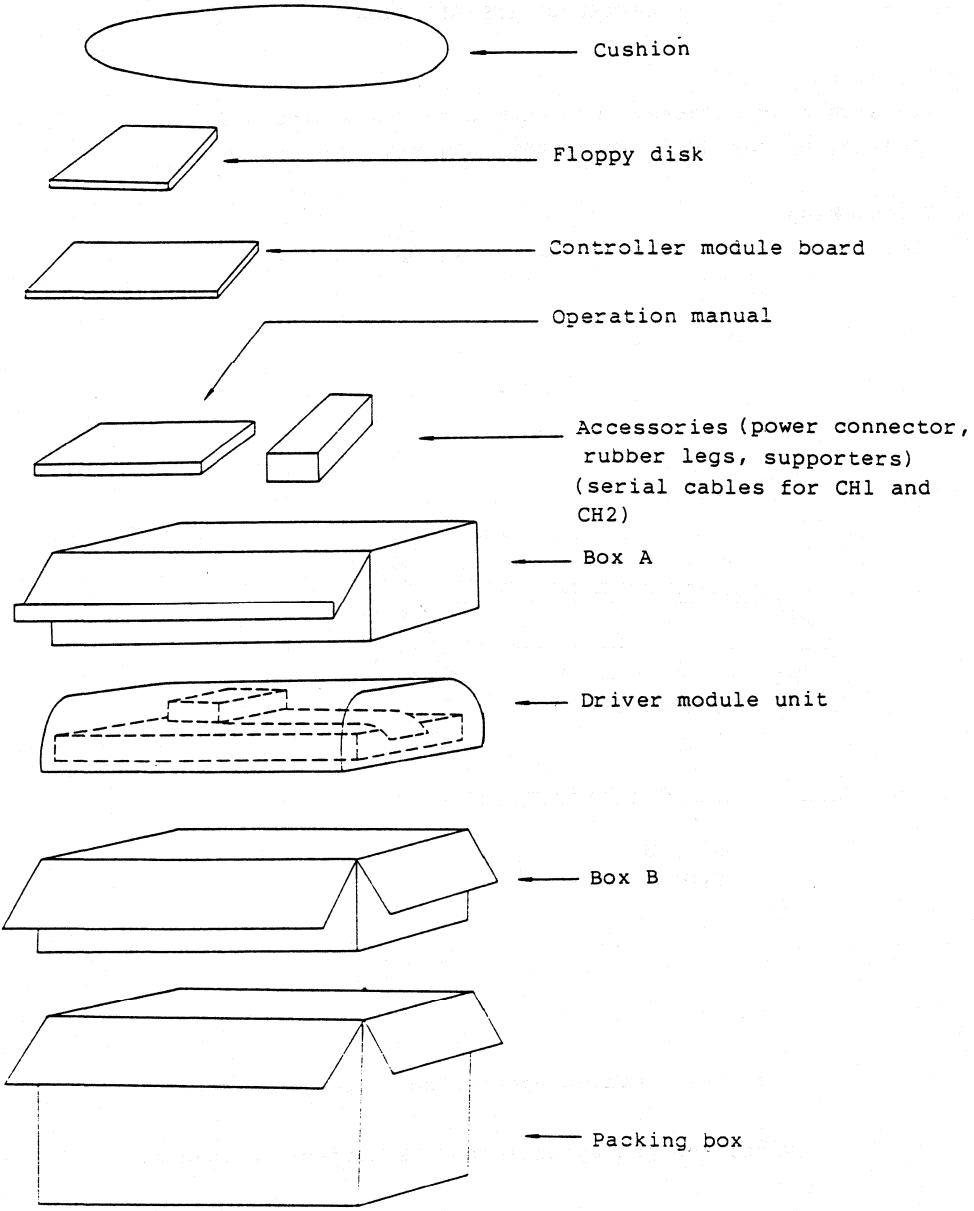
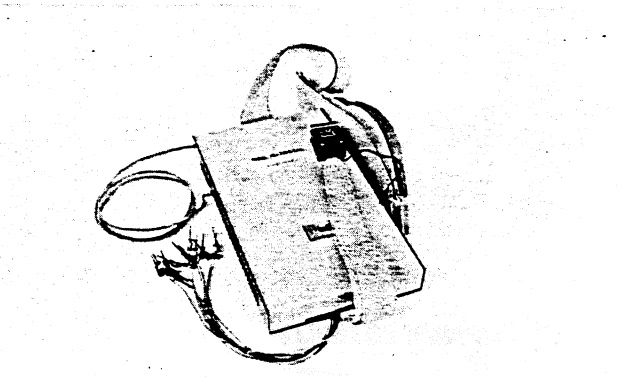


Fig. 2-2 Parts Contained in Shipping Carton

When unpacking the IE-781lH, make sure all components mentioned in the parts list have been included. Some of the components are small and the cables may become disconnected, so be careful when handling the parts. After checking components, exercise care not to lose them.

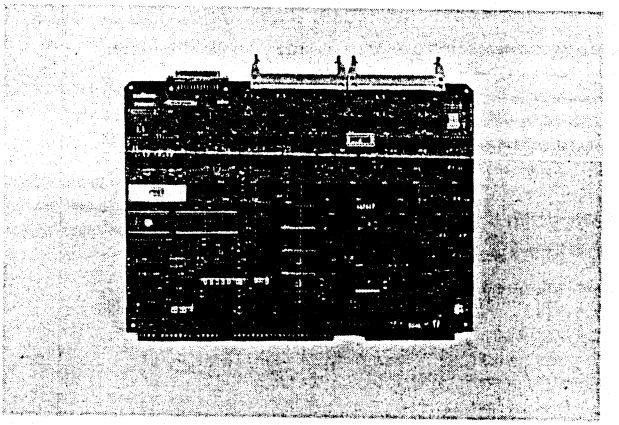
(1) IE-781lH driver

° Driver module	1	
° Power cable I	1	
° External sensing clip	1	(provided with driver
° System interface cables	2	module)
° Emulation target probe	1	



Photograph 2

(2) IE control board 1

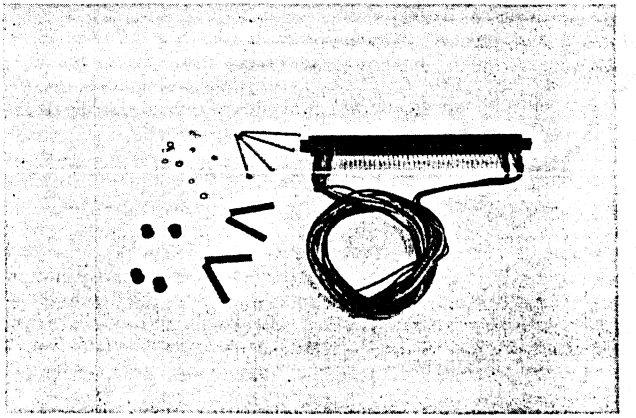


Photograph 3

(3) Accessories I

° Power cable II 1

° Spacers (controller module pads) 4



Photograph 4

(4) Accessories II

- Serial cable I (for CH1) 1
- Serial cable II (for CH2) 1

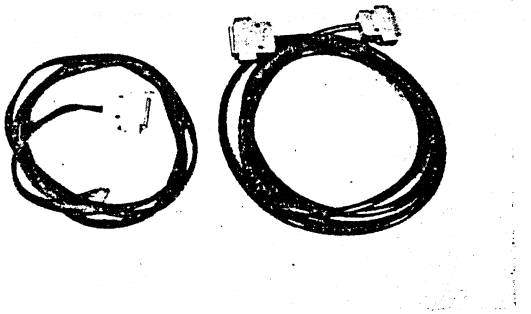


Photo. 5

(5) Other items

- Floppy disk (8-inch) 1
- Floppy disk protective file 1
- IE-781lH operation manual 1
- IE-781lH system software operation manual 1
- Warranty statement 1
- Contract for program product use 1

2.3 Installation

Follow the procedures given next when installing the IE-7811H.

2.3.1 Setting the IE controller module

The locations of the jumpers and switches are shown in Fig. 2-3 when setting the jumpers and switches of the IE controller module.

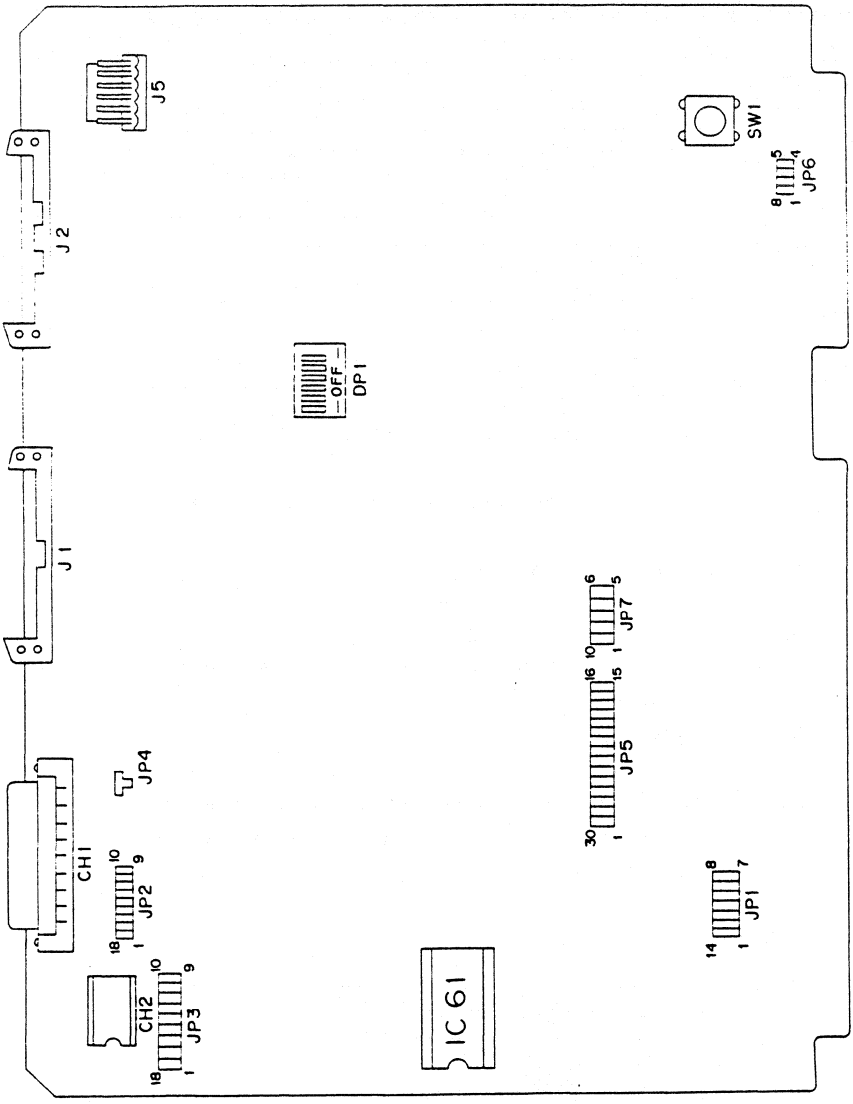
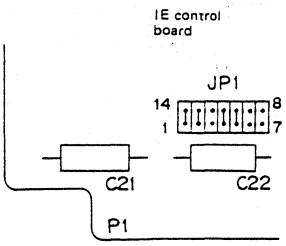


Fig. 2-3

(1) Confirm that JP1 to JP7 are as shown in Figs. 2-4 to 2-6.

(a) JP1



Position	When used as stand-alone	When connectd to MD-086 series
1-14	o	
2-13	o	
3-12		
4-11	o	o
5-10	o	
6-9		
7-8		

"o" indicates shorted.

Fig. 2-4 Jumper Connection When Used as Stand-alone

When used as stand-alone
3-12, 6-9, 7-8: Open
Others: Shorted
When connected to host machine (MD-086 series)
4-11 : Shorted
Others: Open

Note: When a termination resister is provided to pin 14 of P1 connecter on the IEEE-796 bus, 5-10 of JP1 should be open.

(b) JP2, JP3, JP4

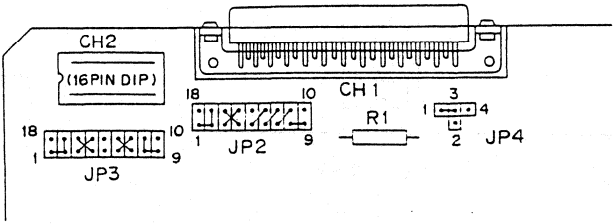


Fig. 2-5

JP2

1-2-17, 3-15, 4-16, 5-13, 6-12, 7-11, 8-9 : Shorted

10, 14, 18 Open

JP3

1-2-17, 3-15, 4-16, 6-12, 7-13, 8-9-11: Shorted

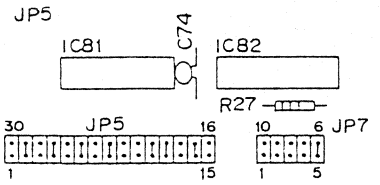
5, 10, 14, 18; Open

JP4

1-3; Shorted

Others; Open

(c) JP5



Position	Shorted
1-30	
2-29	o
3-28	
4-27	o
5-26	
6-25	o
7-24	
8-23	o
9-22	
10-21	
11-20	o
12-19	o
13-18	
14-17	o
15-16	

Fig. 2-6 "o" indicates shorted

1-30, 3-28, 5-26, 7-24, 9-22, 10-21, 13-18,

15-16: Open

Others: Shorted

(d) JP6

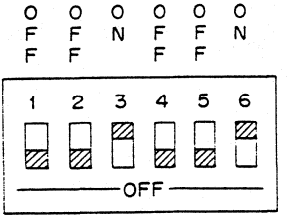
All jumper posts should be open

(e) JP7

5-6: Shorted

Others: Open

(2) Confirm that DP1 (DIP switch) is set correctly.



Note: DIP switches 1 to 3 set the baud rate, while switches 4 to 6 set serial protocol.

Fig. 2-7

These are the factory-set conditions: baud rate of 4,800 bps and handshaking performed by hardware. Refer to 4.6 for baud rate specification, and 4.4 for serial protocol.

(3) Jumper functions

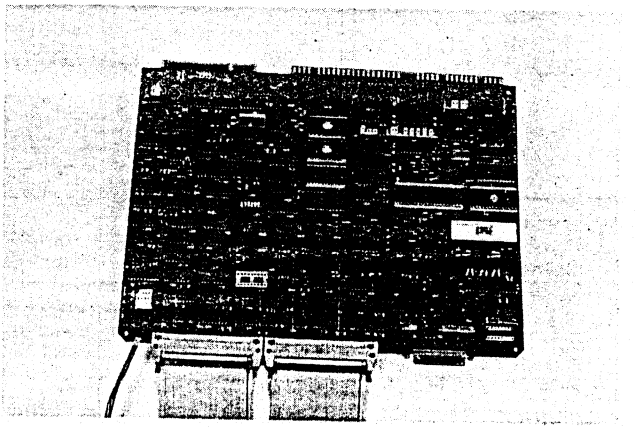
Jumper settings indicated by asterisk should not be modified.

Jumper number	Position	Function
JP1	1-14	Sets the $\overline{CCLK}$ signal. Inside the IE, $\overline{CCLK}$ signal of IEEE-796 bus is used. When the $\overline{CCLK}$ signal is output to the IEEE-796 bus from another board, this should be open. If the $\overline{CCLK}$ signal is not output to the IEEE-796 bus, this jumper setting should be connected so that the $\overline{CCLK}$ signal is output to the IEEE-796 bus from the IE controller module.
	2-13	Sets the $\overline{BCLK}$ signal. This setting has the same function as the 1-14 pin setting. When shorted, the $\overline{BCLK}$ signal is output to the IEEE-796 bus.
	3-12*	Sets the reset signal for IE ($\overline{INIT}$). 3-12 should be open, and 4-11 should be shorted.
	4-11*	
	5-10	Sets the reset signal for IE. When a termination resistor is provided for the $\overline{INIT}$ signal on the IEEE-796 bus, this jumper setting should be open, and should be shorted if the resistor is not provided.
	7-8	Sets the $\overline{BPRO}$ signal. When these pins are connected, the $\overline{BPRO}$ signal is output to the IEEE-796 bus, and not output when open.
JP2		Sets CH1 serial interface.
JP3		Sets CH2 serial interface.
JP4*		Selects the clock in the IE controller module. Only 1-3 should be shorted.
JP5		Sets IE number.
JP6*		Hardware interface in the IE controller module. All settings should be open.

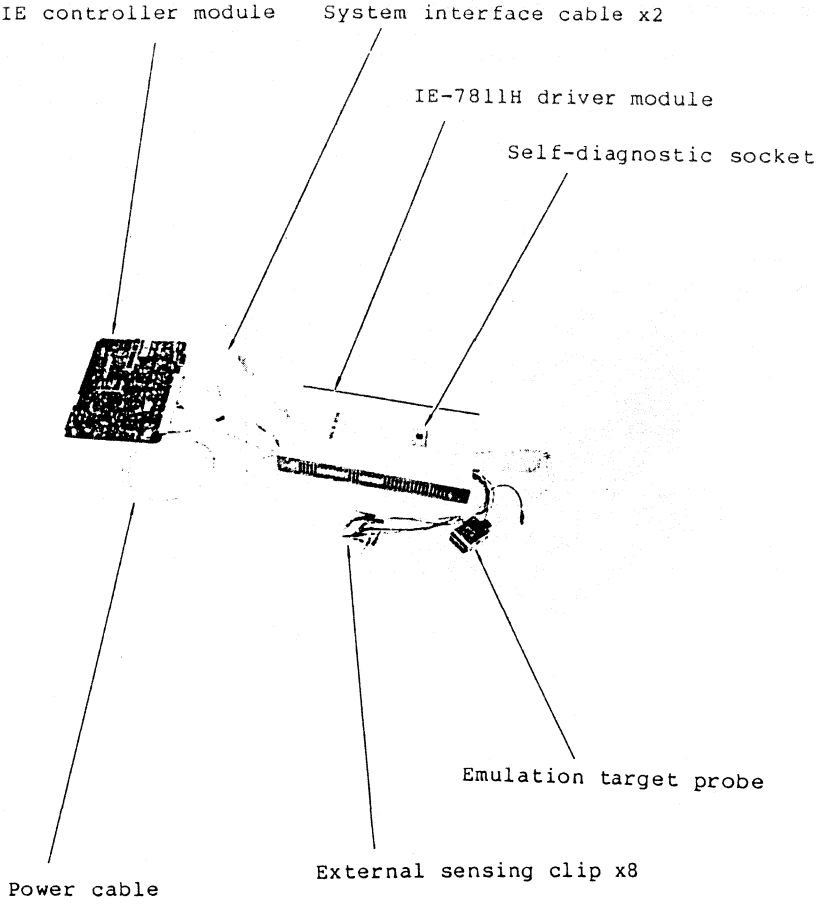
2.3.2 Connecting IE controller module to driver module

(1) Insert J1 and J2 connectors of the 50-pin flat cable connected to the IE-781lH driver module to the corresponding sockets "J1" and "J2" (refer to Photograph 6).

(2) In the same way, insert the connector of the power cable to the IE-781lH driver module to the power socket (refer to Photograph 6).

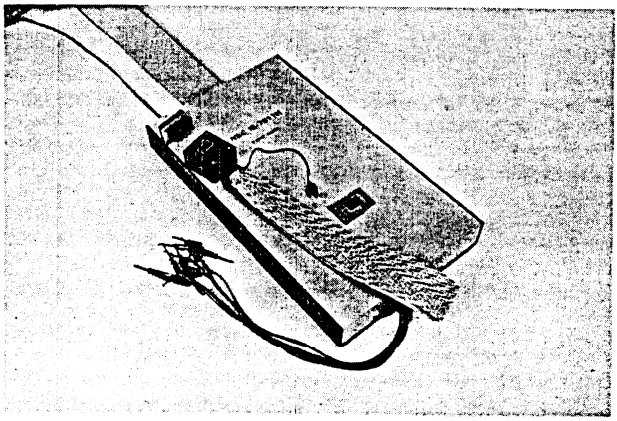


Photograph 6 External View of the IE-781lH Controller Module



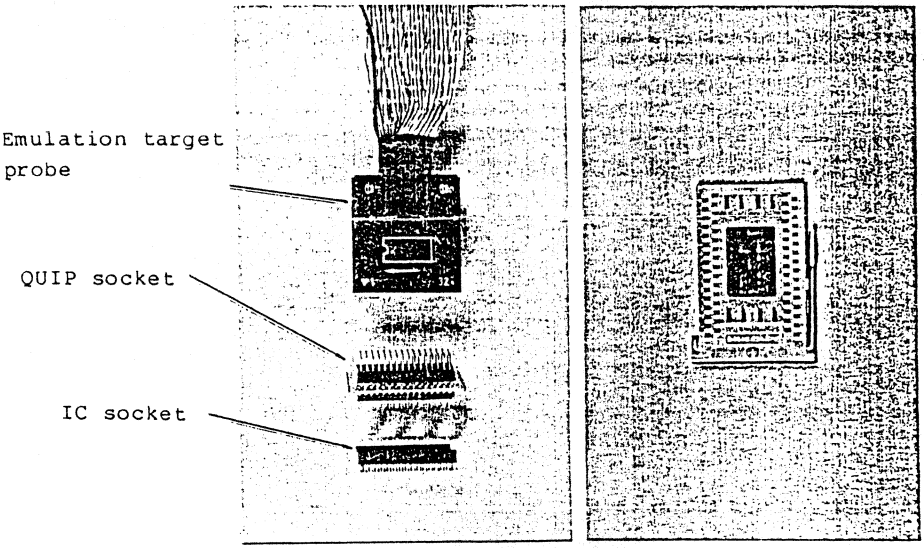
Photograph 7 External View of the IE-781lH

2.3.3 Setting up the IE-7811H driver module



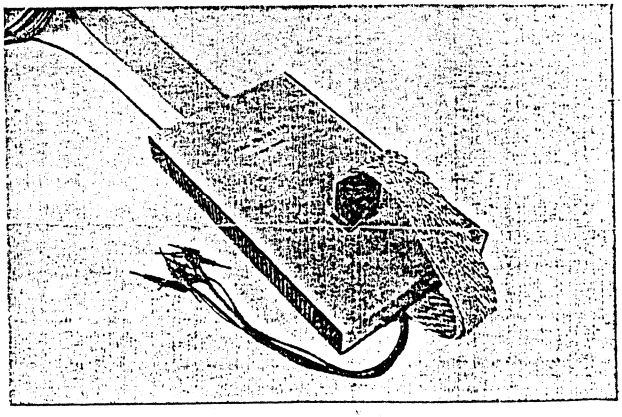
Photograph 8 External View of the IE-7811H Driver Module

(1) Set the clock selection switch on the emulation target probe to CLK.



Photograph 9 Emulation Target Probe and Self-diagnostic Socket

(2) Remove the emulation target probe from the QUIP socket and insert it in the self-diagnostic socket on the IE-7811H driver module.



Photograph 10

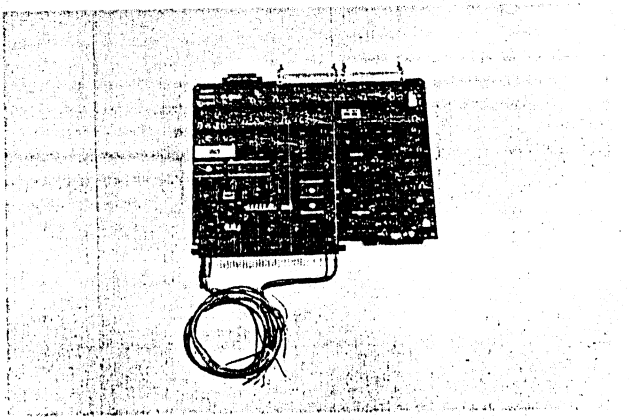
2.3.4 Connecting power cable of IE controller module

Currents of +5V, +12V, and -12V should be supplied through socket P1 of the IEEE-796 bus. The power supply must be able to supply a minimum 6.5A (+5V) and 500mA (±12V) (refer to Fig. 2-8).

Component side of IE controller module

PIN	Power supply	Cable color
1, 2, 85, 86	GND	Black
3, 4, 5, 6	+5V	Red
7, 8	+12V	Orange
79, 80	-12V	Blue

Fig. 2-8



Photograph 11

The following product is recommended as the edge connector P1.

Kel Corp. P/NO. 1258-086-032

SB-9017 (five-slot card cage) and SB-9007 (card cage with ±5V, ±12V power supply) are available from NEC.

2.3.5 Connecting IE controller module to console

Connect CH1 (RS-232C port) of the IE controller module and the RS-232C port of the console using a serial cable. Refer to Chapter 4 for this, because how the cables are connected may differ depending on the type of terminal or CRT used, possibly requiring readjustment of J2. (Examples are shown for three types of typically used terminals.)

2.4 Connecting With Target System

2.4.1 Connecting procedure and power-on sequence

Connect the IE-7811H to the target system in the following manner.

(1) Turn off the power supplies of both the IE-7811H and the target system.
(2) Insert the 64-pin emulation target probe in the 64-pin IC socket of the target system.
(3) Turn on the target system power supply.
(4) Turn on the IE-7811H system power supply.

When turning off these power supplies, turn off the one for the IE-7811H first, then that of the target system.

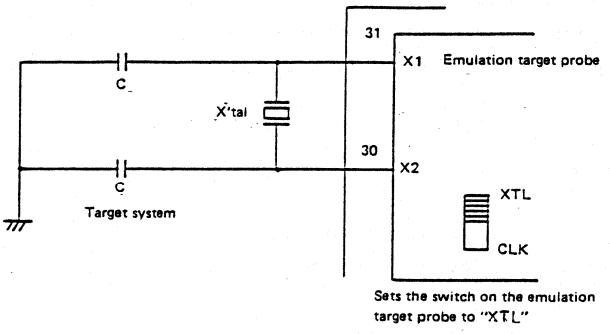
2.4.2 Setting the emulation target probe

A crystal oscillator is provided in the emulation target probe of the IE-7811H. Therefore, when using the oscillator of the target system as the CPU clock, the clock selection switch of the emulation target probe should be set according to the oscillator to be used. To use an

external clock, the external clock must be specified (CLK U) by executing a clock command (refer to Section 3.12). When using the internal clock (12MHz) of the IE-7811H system and the oscillator of the target system is not used, setting the clock selection switch is not required. The clock command carries out the necessary specification for the internal clock.

(1) When the oscillator of the target system consists only of a crystal oscillator* (X'tal), set the switch to "ETL".

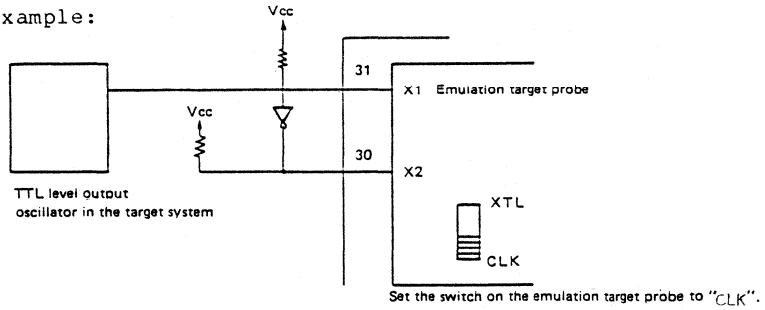
Example:



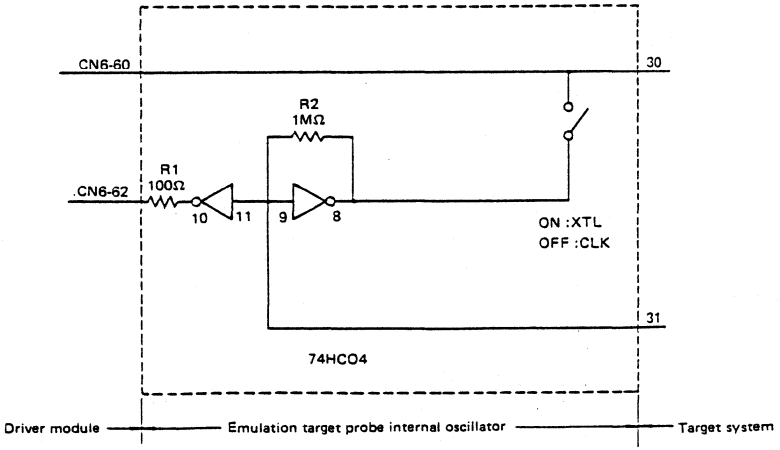
* A ceramic resonator cannot be used for the oscillator of the emulation target probe.

(2) When the oscillator of the target system consists of an oscillator for TTL level output, set the switch to "CLK".

Example:



(3) Emulation target probe internal circuit



N.B. The switch on IE-7809 target probe is different in appearance and function. See Appendix.

CHAPTER 3 DETAILED DESCRIPTION OF MONITOR COMMANDS

3.1 Initial Setting

When the IE-781lH monitor is started up, hexadecimal numeric input is specified as the default, and the memory mapping is cleared. The emulation CPU clock is set to the internal clock (15MHz), therefore, the MEM (memory), RUN (emulate), ASM (on-line assembler), DAS (disassembler), LOD (load), SAV (save), and MOV (memory transfer) commands that access the memory cannot be executed when the IE-781lH monitor is started up.
Radixes are hexadecimal, decimal, octal, and binary, and selected by SUF (suffix) command. Inputs are evaluated according to the radix selected. Note that some commands require that numeric input be in hexadecimal.
Set the mapping data before executing a monitor command.
When an asterisk (*) appears by CTRL-C during the emulation, the commands (MEM, RUN, LOD, SAV, ASM, DAS, MOV, TR?, REG, CLK, MAP, BRA, BRB, BRC, BRM, PGM) that start up the driver module cannot be executed. Press ESC key or execute the RES command before executing these commands.
To perform software debugging without having the target system connected, insert the emulation target probe in the self-diagnostic socket.

3.1.1 Numeric values

Inputting numeric values can be done by several types of numeric strings and sometimes by adding a radix at the end of the string. The radix specifies whether the numeric string is hexadecimal, decimal, octal, or binary, and is expressed by H, T, Q, and Y, respectively. If the radix is not specified in the numeric string, the radix specified by the SUF command is assumed. An example of each radix is shown in the following.

Example:

0FFFFH = FFFF	}	Hexadecimal suffix
0101H = 101H	}	
0632Q = 632	}	Octal suffix
07152Q = 7152Q	}	
0195T = 195	}	Decimal suffix
09999T = 9999T	}	
01101Y = 1101Y	}	Binary suffix
001100Y = 1100Y	}	

Error input

Q, H, T, Y } Radix specification directly appears without data

Input of unnecessary digits is ignored. However, if a nonnumeric character (other than 0 to 9 and A to F) is included in the ignored digits, an error is flagged.

Example:

12A4F6H	= A4F6H	16-bit input
1F59BCH	= BCH	8-bit input
252T	= 252T	8-bit input
11011010Y	= 11011010Y	8-bit input
1G66Q	—— Error	8- or 16-bit input

When the value input contradicts the specified suffix, an error is flagged.

Example:

Suffix - octal

2718

E195Q ~~ Underlined portions exceed octal number

Suffix - decimal

F910

AB7CT ~~ Underlined portions exceed decimal number

Suffix - binary

1234

0121Y ~~ Underlined portions exceed binary number

The effective number of digits and the maximum values for each input mode are shown in the table below.

Input Suffix	8 bit	10 bit	16 bit
H	2 digits	3 digits	4 digits
	FF	3FF	FFFF
T	3 digits	4 digits	5 digits
	255	1023	65535
Q	3 digits	4 digits	6 digits
	377	1777	177777
Y	8 digits	10 digits	16 digits
	11111111	1111111111	1111111111111111

3.1.2 Operand input format

(1) addr

This is the address designation input as a 16-bit value.

(2) addrx

This is the numeric range specification by address value. "X" can be substituted for the normal start/end address description.

Example:

MAP W 0, FF ↵ MAP W XX ↵

"XX" is described in this example. "X" represents 0 to F in hexadecimal. By modifying the suffix, "X" can represent 0 to 7 if octal is specified, and 0 to 1 if binary is specified. "X" cannot be specified, however, as a decimal number.

"X" input must be performed in sequence from the lower digits.
The "X" for the upper digits only, cannot be used if there is data in or after the "X".

Examples:

X1XY (binary) ——————→ Error
XX0Q (octal)——————→ Error
XX1234H (hexadecimal)——→ Error

In the case of XX1234H, an error will occur even if "X" had not been specified within the valid series of digits. As the definition of addrx, only "X" inputs are allowed; therefore, other input formats are not allowed. When addrx appears, input should be made according to the above conditions.

(3) partition
The "partition" part of the command designates the numeric range by "addr, addr" (start address and end address) or addrx (refer to (2)). A single addr expression is not allowed. When partition appears, input should be made according to the above condition.

(4) partitions
When "partitions" appears as part of a command, this designates that the partition should be divided into two or more partitions. This is done by delimiting the partitions with a space [partition _ partition...]. A single addr expression is not allowed. When partition appears, input should be made according to the above condition.

(5) addrs
The addrs part of the command appears in all format types -- addr, partition, and partitions.

(6) string

Indicates a data string up to a maximum of 10 numeric values delimited with commas. Only hexadecimal inputs can be used for the string. The bit length of the values used in the string is 8 bits.

(7) register-name

This is a register name (refer to Section 3.4). The following registers are provided.
PSW, V, A, B, C, D, E, H, L, EA, SP, PC, V', A', B', C', D', E', H', L', and EA'

(8) mode-register-name 1

This is a mode register name (refer to Section 3.5). The following mode registers are provided.
MA, MB, MCC, MC, MM, MF, TMM, ETMM, EOM, SML, SMH and ANM (See Appendix for variation IE-7809)

Note: All of these registers can be set; however, not all of them can be read out.

(9) mode-register-name 2

This is a mode register name of those listed in (8) above that can be read (refer to Section 3.5). The mode registers that can be read are:
TMM, EOM, SMH, and WDM

(10) special-register-name

This is a special register name (refer to Section 3.6). The following special registers are provided.
PA, PB, PC, PD, PF, MKH, MKL, TXB, TM0, TM1, ETM0 and ETM0

Note: All of these registers can be set; however, not all of them can be read out.

(11) special-register-name 2

This is a special register name of those listed in (10) that can be read (refer to Section 3.6). The special registers that can be read are:
PA, PB, PC, PC, PF, MKH, MKL, RXB, CR0, CR1, CR2, CR3, ECNT, ECPT

(12) input/output-device-name

TTY1: Corresponds to CH1 of the serial interface
TTY2: Corresponds to CH2 of the serial interface

(13) step-No.

To the step-No. part of the command, input is made in the numeric format described in Section 3.1.1. The bit length is 16 bits.

(14) value

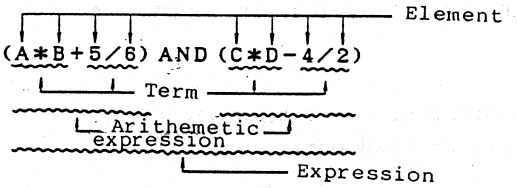
To the value part of the command, input is made in the numeric format described in Section 3.1.1. The bit length is 16 (or 8) bits.

(15) expression

The expression consists of the following.

<element>	: value (it is a value itself)
<term>	: quotient or product of <element>
<arithemetic expression>	: sum or remainder of <term>
<expression>	: comparison operation of two <term>s

Example:



Available operators are +, -, *, /, AND, OR, XOR, and NOT (refer to Section 3.14).

Note: Nested operations using parentheses are possible. The number of nestings (the number of pairs of parentheses) is limited to 32.

(16) line-No.

To the line-No. part of the command, input is made in the numeric format described in Section 3.1.1. The bit length is 16 (or 8) bits.

(17) values

As regards numeric expression, this is the same as (14). However, the mask condition can be used in the numeric string.

Example:

$$V=1X3Q \quad \left\{\begin{matrix}103\\113\\123\\133\\143\\153\\163\\173\end{matrix}\right\}$$

Normal values can also be used.

Note: Decimal numbers cannot be used.

(18) cond

This specifies the break and trace conditions (OP, RD, WR, MR, NC).

(19) loop

This sets the number of times that the break condi tion set by the address, data, and condition fields is satisfied causing break to occur. The bit length is 8 bits.

3.1.3 Terminator for command input

Any number of blanks can be used as the terminator for a command. However, at least one must be used.

Example: Correct and incorrect use of blanks

```
*MEM␣D 0.FF↵        Two blanks                          ; OK
*MEM D↵             Prompt is followed by a blank       ; OK
*M␣EM C↵            The blank divides the command       ; ERROR
*MEM D 0,␣FF↵       Comma is followed by a blank        ; ERROR
```

3.2 MAP (mapping) Commands

Format:

$$*MAP\left[\ \_\left\{\begin{matrix}R\\W\\U\\K\end{matrix}\right\}\left[\ \_partition\ \right]\right]\ \text{↵}$$

Function: Specifies, displays, and releases the memory mapping status.

Main command:	MAP		Mapping command
Subcommand	: R		Maps the IE-781lH's internal memory as ROM.
	: W		Maps the IE-781lH's internal memory as RAM.
	: U		Maps to the user side.
	: K		Releases mapping.
Operand	: Partition		Value that can be "X" input.

In the above format, partition input has the following meanings.

When no partition is input; Displays mapping data (when K is specified, all mapping is released)

When partition is input ; Sets mapping data (when K is specified, data in the partition is released)

3.2.1 Outline of mapping command

When the power is turned on, the reset switch is pushed, or command "RES H↵" is executed, the mapping is in cleared state (that is, the memory is not connected). After the monitor is started up, a command that accesses the memory cannot be executed unless the mapping has been specified. The memory can be mapped in 256-byte

units. The IE-781lH stand-alone can access up to 64K bytes of memory.
This memory can be mapped in 256-byte units either as the user target system memory or as the internal RAM of the IE-781lH stand-alone. A program of the user target system can be loaded into this memory.
The following three types of memory mapping are available:

* As internal RAM of the IE-781lH stand-alone, for which write protect can be used
* As memory mapping for the user target system
* As non-mapped

Other than by command, these three types of mapping can be designated either by CPU mode setting upon start up or automatically, by setting RAE.
When RAE is set to ENABLE, addresses FF00 to FFFFH are automatically mapped as the internal RAM; this cannot be modified by the mapping command. When set to DISABLE, the internal RAM area must be set by mapping command.
Internal ROM mapping by CPU mode setting upon start up can be done automatically (the same as above). This area also cannot later be modified. Refer to the IE-781lH System Software User's Manual for details on initialization.

3.2.2 Mapping commands

The following three types of mapping commands are available:

(1) For mapping setting of a specified area.
(2) For display of the current mapping data.
(3) For clearing the mapping of the specified area.

Mapping is performed in 256-byte units.
If the permissible range of memory for mapping specification does not conform to the conditions listed in the following (1) to (4), and exceeds the range, the message "MAPPING ERROR" is displayed and the command is ignored.

(1) When the target CPU is μPD7810H and the RAE bit is ENABLE

IE system RAM area	(W)	0000 to FFFFH
IE system ROM area	(R)	0000 to FEFFH

User system area (U) 0000 to FEFFH
Release (K) 0000 to FEFFH

(2) When the target CPU is μPD7810H and the RAE bit is DISABLE

IE system RAM area (W) 0000 to FFFFH
IE system ROM area (R) 0000 to FFFFH
User system area (U) 0000 to FFFFH
Release (K) 0000 to FFFFH

(3) When the target CPU is μCOM-7811H and the RAE bit is ENABLE

IE system RAM area (W) 1000 to FFFFH
IE system ROM area (R) 0000 to FEFFH
User system area (U) 1000 to FEFFH
Release (K) 1000 to FEFFH

(4) When the target CPU is μCOM-7811H and the RAE bit is DISABLE

IE system RAM area (W) 1000 to FFFFH
IE system ROM area (R) 0000 to FFFFH
User system area (U) 1000 to FFFFH
Release (K) 1000 to FFFFH

3.2.3 Mapping specification

MAP␣{W, R, U, K}␣partition)

Examples
*MAP W 1000H, 13FFH ↵ (1)
*MAP R 800H, FFFH ↵ (2)
*MAP U 0H, 7FFH ↵ (3)
*MAP K 1000H, 13FFH ↵ (4)

(1) Specifies program memory address space 1000 to 13FFH as IE-7811H system internal RAM.

(2) Specifies program memory address space 800 to FFFH as IE-7811H system internal ROM. The area specified as internal ROM is write protected during program execution

(3) Specifies program memory address space 0 to 7FFH as the user system memory.

(4) Clears the mapping specification in address space 1000 to 13FFH of the program memory.

Examples:

```
*MAP W 1000H.13FFH ↵
*MAP ↵
 $ IE-INTROM MAPPING $
 $ IE-INTRAM MAPPING $
 1000-13FF FF00-FFFF
 $ USER $
 $ NON MAPPING $
 0000-0FFF 1400-FEFF
```

← Specifies program memory address space 1000 to 13FFH as internal RAM.

```
*MAP R 800H.FFFH ↵
*MAP ↵

 $ IE-INTROM MAPPING $
 0800-0FFF
 $ IE-INTRAM MAPPING $
 1000-13FF FF00-FFFF
 $ USER $
 $ NON MAPPING $
 0000-07FF 1400-FEFF
```

← Specifies program memory address space 800 to FFFH as internal ROM.

```
*MAP U 0.7FFH ↵
*MAP ↵
 $ IE-INTROM MAPPING $
 0800-0FFF
 $ IE-INTRAM MAPPING $
 1000-13FF FF00-FFFF
 $ USER $
 0000-07FF
 $ NON MAPPING $
 1400-FEFF
```

← Specifies program memory address space of 2K bytes 0 to 7FFH as user memory.

```
*MAP K 1000H.13FFH ↵
*MAP ↵
 $ IE-INTROM MAPPING $
 0800-0FFF
 $ IE-INTRAM MAPPING $
 FF00-FFFF
 $ USER $
 0000-07FF
 $ NON MAPPING $
 1000-FEFF
```

← Releases mapping specification of program memory address space 1000 to 13FFH.

```
*MAP W XXXX ↵
      ↑
      'X' input
*MAP ↵

 $ IE-INTROM MAPPING $
 $ IE-INTRAM MAPPING $
 0000-FFFF
 $ USER $
 $ NON MAPPING $
```

← Specifies program memory address space 0 to FFFFH as internal RAM.

3.2.4 Display

MAP [␣ { W / R / U }] ↵

Examples:

*MAP W ↵ (1)
*MAP R ↵ (2)
*MAP U ↵ (3)
*MAP ↵ (4)

(1) Displays the area mapped as IE-781lH system internal RAM.
(2) Displays the area mapped as IE-781lH system internal ROM.
(3) Displays the area mapped as user system memory.
(4) Displays the entire mapped memory area.

Example:

```
*MAP W ↵
 $ IE-INTRAM MAPPING $
 FF00-FFFF
```
← Displays the mapping status of the internal RAM.

```
*MAP R ↵
 $ IE-INTROM MAPPING $
 0800-0FFF
```
← Displays the mapping status of the internal ROM.

```
*MAP U ↵
 $ USER $
 0000-07FF
```
← Displays the mapping status of the user memory.

```
*MAP ↵
 $ IE-INTROM MAPPING $
 0800-0FFF
 $ IE-INTRAM MAPPING $
 FF00-FFFF
 $ USER $
 0000-07FF
 $ NON MAPPING $

 1000-FEFF

*
```
← Displays all the mapping statuses.

3.2.5 Clear

MAP_K

Sets the mapping status which was set when the IE-781lH system was started up.

(1) When the target CPU is μPD7810H and the RAE bit is ENABLE

```
*MAP K ↵

*MAP ↵

 $ IE-INTROM MAPPING $
 $ IE-INTRAM MAPPING $
 FF00-FFFF
 $ USER $
 $ NON MAPPING $
 0000-FEFF
```

(2) When the target CPU is μPD7810H and the RAE bit is DISABLE

```
*MAP K ↵

*MAP ↵

 $ IE-INTROM MAPPING $
 $ IE-INTRAM MAPPING $
 $ USER $
 $ NON MAPPING $
 0000-FFFF
```

(3) When the target CPU is μCOM-781lH and the RAE bit is ENABLE

```
*MAP K ↵

*MAP ↵

 $ IE-INTROM MAPPING $
 0000-0FFF
 $ IE-INTRAM MAPPING $
 FF00-FFFF
 $ USER $
 $ NON MAPPING $
 1000-FEFF
```

(4) When the target CPU is μCOM-7811H and the RAE bit is DISABLE

```
*MAP K ↵

*MAP ↵

 $ IE-INTROM MAPPING $
 0000-0FFF
 $ IE-INTRAM MAPPING $
 $ USER $
 $ NON MAPPING $
 1000-FFFF
```

3.3 Memory Commands

Format

*MEM␣C [␣addr] ↵

*MEM␣D [␣partition] ↵

*MEM␣E␣partition [$A] ↵

*MEM␣{F / G}␣partition [␣string] ↵

*MEM␣{M / V / X}␣partition␣addr ↵

Function: The memory commands perform such operations as modification, display, test, initialization, search, transfer, comparison, and exchange of mapped memory.

Main command	: MEM	Memory command
sub command	: C	Memory change (Change)
(not	: D	Memory dump (Dump)
ommitable)	: E	Memory test (Examination)
	: F	Memory initialization (Fill)
	: G	Memory search (Get)
	: M	Memory move (Move)
	: V	Memory comparison (Verify)
	: X	Memory exchange (eXchange)
Operand	: addr	Value (2 bytes) that can be input by formula

	: partition	Area input that can be "X" input
	: string	Continuous input of 1-byte HEX data (up to 10 data can be input; use a comma as the delimiter)
Option	: A	A simplified test switch when subcommand E is specified.

3.3.1 Memory command

When executing the memory command, be sure that mapping is set either to inside or outside the IE-781lH system. If not mapped, "NON-MAP AREA ACCESS" is output, "*" is displayed, and the command will not be executed.

3.3.2 Memory change (Change)

MEM_C_addr↵

When HEX mode is used, the lower four digits of 0 to FFFFH are effective as address input. When the command is correctly input, the address of the memory and its contents are displayed, "-" is output, and the status becomes data input wait ("-" indicates output from the IE-781lH system).

Examples:

```
*MEM C 10↵
NON-MAP AREA ACCESS
*
```

If not mapped, an error occurs and "*" is displayed as shown at left.

Mapping should be specified and the command should be input again.

```
*MAP W 0,FFH↵
```

```
*MEM C 10T↵
000A 00-34 00-88 00-55 00-
*
```

Decimal input

HEX data input

After the memory contents are changed, the command can be terminated by inputting "-" following "↵".

```
*MEM C 10T↵
000A 34-34 88-88 55-55 00-
*
```

Space input

When space is input after "-", the contents of the memory at that address are not changed. The same data is echoed back onto the screen and the contents of the next memory address are displayed leaving the data unchanged.

```
*MEM C ↵
0000  00-12 00-FF 00-67 00-          '←'
0002  67-  ————————————————(ASCII 5FH)
0001  FF-88 67- ↵
*
```

If the address is not specified after "C" in the command input, address 0 is assumed. If "←" (ASCII 5FH) of ASCII standard is input after "-", display returns to the preceding address.

```
*MEM C 1209541000I0H ↵
0010 FF-FF FF-FF FF-FF FF- ↵
*
```

When addresses are input in HEX format, the lower-order four digits (up to 0FFFFH) are available.

```
*MEM C 10001Y ↵
0011  FF-FF FF-FX  ————————Input of non-HEX number
INPUT DATA ERROR
*
*MEM C 5Q ↵
0005  00- ↵
*
```

The type of the address input is determined by the numeric base indicator (H, T, Q, or Y) attached to the end of the addess specification. Input the contents of the memory in HEX code. If the memory contents are not input in HEX code, message "INPUT DATA ERROR" is output, and "*" is displayed.

```
*MEM 13Q ↵
000B 88-A ──────────── Space is input for second character
INPUT DATA ERROR
*

*MEM C 5↵
0005 00-Z ──────────── Input of non-HEX number
INPUT DATA ERROR
*
*MEM C10↵
     └──────────── No space
COMMAND FORMAT ERROR
*
```

3.3.3 Memory dump (Dump)

MEM_D_partition↵

Dumps the memory area specified by partition.

```
*MEM D 0,FH↵
0000   12 88 67 00 00 00 00 00  00 00 34 88 55 00 00 00
*
*MEM D XH↵
0000   12 88 67 00 00 00 00 00  00 00 34 88 55 00 00 00
*
*MEM D XQ↵
0000   12 88 67 00 00 00 00 00
*
*MEM D XY↵
0000   12 88
*
```

As shown in the above example, "X" can be input for the address parameter in partition.

```
*MEM D 2F0H,31FH↵
02F0   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF

0300   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
0310   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
```

When the lower-order two digits of the address become FF in dump display, the next line is left vacant.

```
*MEM D FFH,FFH↵
00FF   FF
*MEM D 3XH↵
0030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
*MEM D XXXH↵
0000   12 28 67 00 00 00 00 00  00 00 34 88 55 00 00 00
0010   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
0020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
0040   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0050   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
                                                  ↖CTRL-C or ESC key input
```

During program memory dump, if CTRL-C or ESC key is input, character display is stopped, and the display returns to "*".

```
*MAP K↵

*MAP W 0,FFH↵

*MEM D XXXH↵

NON-MAP AREA ACCESS

*
```

When non-mapped area is specified, message "NON-MAP AREA ACCESS" is output, and the display returns to "*".

```
*MEM D 10,5↵

 COMMAND FORMAT ERROR
*
*MEM FFH,FFH↵

 COMMAND FORMAT ERROR
*
*MEM D0,FFH↵
 COMMAND FORMAT ERROR
*
```

3.3.4 Memory test (Examination)

MEM_E_partition($A)↵

A test can be made of the memory area specified by partition. If an abnormality is found, the memory address and its contents are output. Following the test, the data in the area input by partition are not assured. However, a simplified test mode can be obtained by inputting "$A" after partition. Using this, the memory contents are not destroyed and the test only takes about 1 minute and 30 seconds to test a 64K-byte memory, as opposed to the standard memory test which requires about 7 minutes.

3.3.5 Memory initialization (Fill)

MEM_F_partition_string↵

Using commas as delimiters, up to 10 data can be specified in a string.

```
*MEM F 0,FFH 00↵

NON-MAP AREA ACCESS
*
*MAP W↵

*MEM F 0,FFH 00↵

*MEM D 0,63↵
0000   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0050   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060   00 00 00 00
*
```

00H initializes program memory addresses 0 to FFH, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 of string then initialize 0 to FFH.

```
*MEM F 0,FFH 1,2,3,4,5,6,7,8,9,0↵

*MEM D XXH↵
0000   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
0010   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
0020   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
0030   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
0040   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
0050   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
0060   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
0070   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
0080   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
0090   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
00A0   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
00B0   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
00C0   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
00D0   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
00E0   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
00F0   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
*
```

3.3.6 Memory search (Get)

MEM␣G␣partition␣string↵

Displays the start address of data which is arranged in the same way as the string in the partition area.
Using commas as delimiters, up to 10 data can be specified in a string.
The Fill command is then used to write the data string in the memory.

```
*MEM F 0H,110H 00↵

*MEM F F0,110H 55,AA,33,88↵

*MEM D F0H,120H ↵
00F0   55 AA 33 88 55 AA 33 88  55 AA 33 88 55 AA 33 88

0100   55 AA 33 88 55 AA 33 88  55 AA 33 88 55 AA 33 88
0110   55 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0120   FF
*
```

A data string with AA and 33 is then searched in the memory.

```
*MEM G 0,FFH AA,33 ↵
00F1
00F5
00F9
00FD

*MEM G 100H,13FH AA,33,88,55,AA,33 ↵
0101
0105
0109

*
```

3.3.7 Memory transfer (Move)

MEM␣M␣partition␣addr↵

Transfers the contents of the memory location specified by partition to the memory location whose start address is specified by addr.

The memory contents to be transferred are displayed first.

```
*MEM D 0,FFH ↵
0000   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
0010   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
0020   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
0030   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
0040   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
0050   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06

*                                          CTRL-C or ESC key input
```

The contents of the memory to which transfer is to be made are then displayed.

```
*MEM D 3000H,30FFH ↵
3000   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
3010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
3020   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
3030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
3040   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
3050   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
3060   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
3070   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
3080   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
3090   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00

*                                          CTRL-C or ESC key input
```

When the memory move command is executed, the memory contents of addresses 0 to FFH are copied to addresses 3000 to 30FFH. The memory contents of addresses 0 to FFH remain as they are.

```
*MEM M 0,FFH 3000H ↵

*MEM D 3000H,30FFH ↵
3000   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
3010   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
3020   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
3030   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
3040   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
3050   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
3060   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
3070   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
3080   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04

*                                          CTRL-C or ESC key input
```

3.3.8 Memory comparison (Verify)

MEM␣V␣partition␣addr↵

Compares the memory contents specified by partition and the memory contents whose start address is specified by addr. When they coincide, display returns to "*" without displaying anything. If they do not coincide, the address and its data are displayed.

The memory is initialized in the following manner.

```
*MEM F 0,FFH 1,2,3,4,5,6,7,8,9,0↵

*MEM D XXH↵
0000   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
0010   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
0020   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
0030   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
0040   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
0050   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
0060   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
0070   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
0080   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
0090   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
00A0   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
00B0   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
00C0   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
00D0   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
00E0   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
00F0   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
*
```

The memory contents of addresses 0 to FH are then compared with the memory contents of addresses 20 to 2FH.

```
*MEM V 0,FH 20H↵
0000 01 - 0020 03
0001 02 - 0021 04
0002 03 - 0022 05
0003 04 - 0023 06
0004 05 - 0024 07
0005 06 - 0025 08
0006 07 - 0026 09
0007 08 - 0027 00
0008 09 - 0028 01
0009 00 - 0029 02
000A 01 - 002A 03
000B 02 - 002B 04
000C 03 - 002C 05
000D 04 - 002D 06
000E 05 - 002E 07
000F 06 - 002F 08
*
```

When the data of the compared memories do not coincide, output is made as shown above. When they coincide, output is made in the following way.

```
*MEM V 0,1FH 64H ↵
*
```

3.3.9 Memory contents exchange (eXchange)

MEM_X_partition_addr↵

Exchanges the memory contents specified by partition with the memory contents specified by addr.
The following example exchanges the contents of memory area 0 to 3FH with the contents of memory area 2000 to 203FH.

```
*MEM D 0,3FH ↵
0000   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
0010   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
0020   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
0030   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
*
*MEM D 2000H,203FH ↵
2000   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
2010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
2020   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
2030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
*MEM X 0,3FH 2000H ↵


*MEM D 0,3FH ↵
0000   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
0010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
0030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
*MEM D 2000H,20FFH ↵
2000   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
2010   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
2020   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
2030   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
2040   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
2050   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
2060   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
2070   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*                                \ CTRL-C or ESC key input
```

3.4 REG (Register) Command

Format:

*REG [_{C / D} [_register-name]]

Function: Displays and changes the contents of a register of the IE-781lH system.

Main command: REG

Subcommand	: C	Register change
	: D	Register dump
Operand	: register-name	Register name
Option	: None	

3.4.1 Registers

The register names to be used in a command must be registered beforehand. The value input for the register must be based on the radix specified by SUF command. This input can be processed as expression.
(Regardless of the radix, the display is in HEX code.)

(1) Registers (register-name)

PSW	Flag register	PC	Program counter
V	V register	V'	V' register (alt.)
A	Accumulator	A'	A' register (alt.)
B	B register	B'	B' register (alt.)
C	C register	C'	C' register (alt.)
D	D register	D'	D' register (alt.)
E	E register	E'	E' register (alt.)
H	H register	H'	H' register (alt.)
L	L register	L'	L' register (alt.)
EA	Expansion accumulator	EA'	Expansion accumulator (alt.)
SP	Stack pointer		

3.4.2 Change

REG␣C [␣register-name] ↵

Changes the contents of the specified register. When register-name is not specified, the contents of all the registers are changed.

Examples: *REG C EA' ↵ (1)
*REG C ↵ (2)

(1) Changes the contents of the EA' register. If "CR" is input immediately after the contents of the EA' register are displayed, command execution terminates without changing the contents of the register.

(2) Changes the contents of all registers. If space is input after the contents of the register are displayed, the contents of the register are not changed and the next register is displayed. If "CR" is input, command execution terminates without changing the contents of the displayed register.

The contents of the registers are changed in this order: PSW, V, A, B, C, D, E, H, L, EA, SP, PC, V', A', B', C', D', E', H', L', EA'.

Command execution automatically terminates after the contents of the EA' register are changed.

An example of command execution is shown below.

```
*REG C ↵          <-------- When the register name is omitted,
                            all of the registers are changed.
 PSW 00=1 ↵
 V 8E=2 ↵
 A D6=3 ↵
 B 61=4 ↵
 C 11=5 ↵
 D 05=6 ↵
 E 58=7 ↵
 H EC=8 ↵
 L E9=9 ↵
 EA 2AB8=10 ↵
 SP 7000=11 ↵
 PC 0000=12 ↵
 V' 00=13 ↵
 A' 00=14 ↵
 B' 00=15 ↵
```

```
C' 00=16 ↵
D' 00=17 ↵
E' 00=18 ↵
H' 00=19 ↵
L' 00=20 ↵
EA' 0000=21↵
```

The result of changing the register contents is shown next.

[PROGRAM]

```
*REG ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 01  02 03 04 05 06 07 08 09 0010 0011 0012
     V' A' B' C' D' E' H' L' EA'
     13 14 15 16 17 18 19 20 0021

*
```

This is an example of when only the specified registers are changed.

```
*REG C PSW ↵
          └──────── Change contents of the PSW register
 PSW 01=FF ↵

*REG ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 7D  02 03 04 05 06 07 08 09 0010 0011 0012
     V' A' B' C' D' E' H' L' EA'
     13 14 15 16 17 18 19 20 0021

*

*REG C V ↵
         └──────── Change contents of the V register
 V 02=AA ↵

 REG ↵
 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 7D  AA 22 04 05 06 07 08 09 0010 0011 0012
     V' A' B' C' D' E' H' L' EA'
     13 14 15 16 17 18 19 20 0021

*REG C A ↵

 A 22=22
      └──────── Contents unchanged when space is input
*REG ↵
 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 7D  AA 22 04 05 06 07 08 09 0010 0011 0012
     V' A' B' C' D' E' H' L' EA'
*    13 14 15 16 17 18 19 20 0021
```

Errors will occur in these cases:

```
*REG C PSW' ↵
       ↑
       └──────────── If an unrecognized register-name is
COMMAND FORMAT ERROR input.

*REG CA ↵
      ↑
      └──────────── If there is no space before the
COMMAND FORMAT ERROR register-name.

*REG C A ↵

 A F1=FHG
       └──────────── If a non-HEX value is input
 INPUT DATA ERROR
                     In this case, the value of A is
*REG D ↵             retained.

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 49  0A F1 18 10 03 40 12 7E 0050 0003 003E
     V' A' B' C' D' E' H' L' EA'
     09 12 33 24 2D 36 3F 49 2492

*REG C ↵

 PSW 7D=00 ↵
 V AA=11 ↵
 A 22=22 ↵
 B 04=33 ↵
 C 05= ↵  ──────────── Input "CR" to terminate

*REG D ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  11 22 33 05 06 07 08 09 0010 0011 0012
     V' A' B' C' D' E' H' L' EA'
     13 14 15 16 17 18 19 20 0021

*REG C ↵

 PSW 00=1 ↵
 V 11=2 ↵
 A 22=3 ↵
 B 33=4 ↵
 C 05=GFY  ──────────── Input a non-HEX value to terminate

 INPUT DATA ERROR
```

```
*REG ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 01  02 03 04 05 06 07 08 09 0010 0011 0012
     V' A' B' C' D' E' H' L' EA'
     13 14 15 16 17 18 19 20 0021
```

These are examples of inputting non-HEX values:

```
*SUF ↵

 H
*REG C ↵ ———————— If there is no suffix specification
                  after the input value, HEX is displayed.
 PSW 01=1Q ↵ ———————————— Indicates an octal value
 V 02=2Q ↵
 A 03=4Q ↵
 B 04=10Q ↵
 C 05=20Q ↵
 D 06=40Q ↵
 E 07=100Q ↵
 H 08=200Q ↵
 L 09=300Q ↵
 EA 0010=177777Q ↵
 SP 0011=100000Q ↵
 PC 0012=4000Q ↵
 V' 13=11Q ↵
 A' 14=22Q ↵
 B' 15=33 ↵ ———————————— If there is no specification, HEX is
 C' 16=44Q ↵                assumed.
 D' 17=55Q ↵
 E' 18=66Q ↵
 H' 19=77Q ↵
 L' 20=111Q ↵
 EA' 0021=22222Q ↵

*REG ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 01  02 04 08 10 20 40 80 C0 FFFF 8000 0800   Converted to HEX
     V' A' B' C' D' E' H' L' EA'
     09 12 33 24 2D 36 3F 49 2492

*          └———— Value as is

*REG C ↵

 PSW 01=1234 ↵
 V 02=2345H ↵
 A 04=3456789H ↵
 B 08=BFDFBFCDE58F43C0A17574 ↵
 C 10= ↵

*REG ↵
```

```
PSW V  A  B  C  D  E  H  L  EA   SP   PC
34  45 89 74 10 20 40 80 C0 FFFF 8000 0800
    V' A' B' C' D' E' H' L' EA'
    09 12 33 24 2D 36 3F 49 2492

*
```

Because the PSW, V, A, B, and C registers are displayed as two digits, when the input is HEX, only the last two digits are valid.

```
*REG C ↵

 PSW 34=5471111Q ↵
 V 45=6450129 ↵
 A FF=67253361Q ↵
 B 18=5136547389 ↵

 INPUT DATA ERROR

*REG ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 49  0A F1 18 10 03 40 12 7E 0050 0003 003E
     V' A' B' C' D' E' H' L' EA'
     09 12 33 24 2D 36 3F 49 2492

*
```

Because the PSW, V, A, and B registers are displayed as two digits, when the input is octal, only the last three digits (000 to 377Q) are valid. In the example given above, the input for the B register is 738Q and an error occurs. The result is that the PSW, V, and A registers are changed but the contents of the B register remain unchanged.

3.4.3 Display

REG [_D [_register-name]] ↵

Displays the contents of the specified register. When register-name is omitted, the contents of all registers are displayed.

Examples:
```
*REG D A ↵        (1)
*REG D ↵          (2)
*REG ↵            (3)
```

(1) Displays the contents of the A register.
(2)(3) Display the contents of all registers are displayed.

Examples:

```
*REG ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 7D  AA 22 04 05 06 07 08 09 0010 0011 0012
     V' A' B' C' D' E' H' L' EA'
     13 14 15 16 17 18 19 20 0021

*REG D ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 7D  AA 22 04 05 06 07 08 09 0010 0011 0012
     V' A' B' C' D' E' H' L' EA'
     13 14 15 16 17 18 19 20 0021

*
```

These are examples of when the contents of specified registers are displayed:

```
*REG D PSW ↵

 PSW 49

*REG D V ↵

 V 0A

*REG D A ↵

 A F1

*
```

```
*REG D IY ↵
 COMMAND FORMAT ERROR
```

An unrecognized register has been specified and an error occurs.

```
*REG DA ↵
 COMMAND FORMAT ERROR
```

No space is input so an error occurs.

```
*
```

3.5 Mode Register Command

Format:

```
*MDR[ _{ C [_mode-register-name1] }]↵
     { D [_mode-register-name2] }
```

Function: Changes and displays the contents of the mode registers.

Main command: MDR

Subcommand	: C	Register change
	: D	Register dump
Operand	: mode-register-name1	Mode register name 1
	: mode-register-name2	Mode register name 2

3.5.1 IE-781lH mode registers

(1) mode-register-name1

- MA ; Mode A register
- MB ; Mode B register
- MCC ; Mode Control C register
- MC ; Mode C register
- MM ; Memory Mapping register
- MF ; Mode F register
- TMM ; Timer Mode register
- ETMM ; Timer/Event Counter Mode register
- EOM ; Timer/Event Counter Output Mode register
- SML ; Serial Mode register (low)
- SMH ; Serial Mode register (high)
- ANM ; A/D channel Mode register

Registers that belong to mode-register-name1 can be written.

(2) mode-register-name2

TMM ; Timer-mode-register
EOM ; Timer/Event Counter Output Mode register
SMH ; Serial Mode register (high)
ANM ; A/D Channel Mode register

Registers that belong to mode-register-name2 can be written/read.

3.5.2 Change

MDR_C [_mode-register-name1] ↵

Changes the contents of the specified mode register. When mode-register-name1 is omitted, the contents of all mode registers are changed.

Example: *MDR C MC ↵ (1)
*MDR C ↵ (2)

(1) Changes the contents of the MC register. If "CR" is input immediately after the contents of the MC register are displayed, command execution terminates without changing the contents of the register.

(2) Changes, in order, the contents of all mode registers. If a space is input after the contents of the register are displayed, the contents of the register are not changed and the contents of the next register are displayed. If "CR" is input, command execution terminates without changing the contents of the mode register. The mode registers are changed in this order:
MA, MB, MCC, MC, MM, MF, TMM, ETMM, EOM, SML, SMH, ANM
When the contents of ANM are changed, command execution automatically terminates.

3.5.3 Display

MDR [_D [_mode-register-name2]]↵

Displays the contents of the specified mode register. When mode-register-name2 is omitted, the contents of all mode-register-name2 registers are displayed.

Example:

```
*MDR D TMM ↵          (1)
*MDR D ↵              (2)
*MDR ↵                (3)
```

(1) Displays the contents of the TMM register.

(2) (3) Displays the contents of all mode-register-name2 registers.

Examples:

```
*MDR↵

 TMM EOM SMH ANM
 FF  00  00  00

*MDR D↵

 TMM EOM SMH ANM
 FF  00  00  00

*MDR D TMM↵

 TMM FF

*MDR D EOM↵

 EOM 00

*
```

These are examples of displaying and changing mode registers.

Examples:

```
*MDR C ↵

 MA --=12 ↵
 MB --=23 ↵
 MCC --=34 ↵
 MC --=45 ↵
 MM --=56 ↵
 MF --=67 ↵
 TMM FF=78 ↵
 ETMM --=89 ↵
 EOM 00=90 ↵
 SML --=AB ↵
 SMH 00=BC ↵
 ANM 00=CD ↵

*MDR ↵

 TMM EOM SMH ANM
 78  90  BC  CD

*MDR C MA ↵

 MA --=11 ↵

*MDR D MA ↵
 WRITE ONLY REGISTER

*

*MDR C ↵

 MA --=--
 MB --=--
 MCC --=--
 MC --=--
 MM --=--
 MF --=--
 TMM 88=88
 ETMM --=--
 EOM 90=90
 SML --=--
 SMH BC=BC
 ANM CD=CD

*

*MDR C AMN ↵
 COMMAND FORMAT ERROR

*
```

mode-register-namel cannot be displayed.

Inputting space retains original value.

An error occurs when unrecognized mode-register-name is specified.

```
*MDR C EOM ↵

 EOM 90=AA ↵

*MDR D EOM ↵

 EOM AA

*MDR C EOM ↵

 EOM AA=AA
          └──────────── Space input

*MDR C EOM ↵

 EOM AA= ↵  ──────────── When "CR" is input, change is
                         terminated.
*MDR ↵

 TMM EOM SMH ANM
 88  AA  BC  CD

*.

*SUF ↵  ──────────── When suffix is decimal

 T

*

*MDR C ↵

 MA --=12H ↵
 MB --=23H ↵
 MCC --=34Q ↵
 MC --=45Q ↵
 MM --=56T ↵
 MF --=67T ↵
 TMM 4E=78 ↵
 ETMM --=89 ↵
 EOM 5A=111010Y ↵
 SML --=1111Y ↵
 SMH EA=33 ↵
 ANM 7B=65H ↵
*MDR ↵

 TMM EOM SMH ANM
 4E  3A  21  65

*
```

As shown in the above, other suffixes can be used.

3.6 Special Register Command

Format:

```
*SPR[ ␣{ C [␣special-register-name1] }]↵
     { D [␣special-register-name2] }
```

Function: Displays and changes the contents of the special registers.

Main command: SPR

Subcommand	: C	Special register change
	: D	Special register dump
Operand	: special-register-name1	Special register name 1
	: special-register-name2	Special register name 2
Option	: None	

3.6.1 Special registers

(1) special-register-name1

```
PA   ; Port A register
PB   ; Port B register
PC   ; Port C register
PD   ; Port D register
PF   ; Port F register
MKH  ; Mask High register
MKL  ; Mask Low register
TXB  ; TX buffer register
TM0  ; Timer register 0
TM1  ; Timer register 1
ETM0 ; Timer/Event Counter Mode register 0
ETM1 ; Timer/Event Counter Mode register 1
```

Note: The registers that belong to special-register-name1 can be written.

3.6 Special Register Command

(2) special-register-name2

```
PA   ; Port A register
PB   ; Port B register
PC   ; Port C register
PD   ; Port D register
PF   ; Port F register
MKH  ; Mask High register
MKL  ; Mask Low register
CR0  ; A/D Conversion Result register 0
CR1  ; A/D Conversion Result register 1
CR2  ; A/D Conversion Result register 2
CR3  ; A/D Conversion Result register 3
ECNT ; Timer/Event Counter Upcounter
ECPT ; Timer/Event Counter Capture register
```

Note: The registers that belong to special-register-name2 can be read.

3.6.2 Change

SPR␣C [␣special-register-name1] ↵

Changes the contents of the specified special register. When special-register-name1 is omitted, the contents of all special registers are changed.

Examples: *SPR C PF ↵ (1)
*SPR C ↵ (2)

(1) Changes the contents of the PF register. If "special-register-name1" is omitted, the contents of all special registers are displayed in order.

(2) Displays the contents of all special registers in order. When space is input after the contents of the special register are displayed, the contents of the register are not changed and the contents of the next register are displayed. If "CR" is input, command

execution terminates without changing the currently displayed special register. Special registers are changed in this order:

PA, PB, PC, PD, PF, MKH, MKL, TXB, TM0, TM1, ETM0, ETM1.

When the contents of ETM1 are changed, command execution automatically terminates.

Example 1: When the change of the special register contents is affected by the contents of the mode register.

① *SPR C PA ↵

PA 00=12 ↵

② *SPR ↵

```
PA PB PC PD PF MKH MKL RXB CR0 CR1 CR2 CR3 ECNT ECPT
00 00 00 CB 00 07  FE  FF  FF  FF  FF  FF  0000 FFFF
```

③ *MDR C MA ↵

MA --=0 ↵

④ *SPR ↵

```
PA PB PC PD PF MKH MKL RXB CR0 CR1 CR2 CR3 ECNT ECPT
12 00 00 CB 00 07  FE  FF  FF  FF  FF  FF  0000 FFFF

*
```

In (1) the contents of the PA register are changed and displayed; in (2), however, no change in the contents of the PA register is seen. This is because the MA register had been set for input mode. In (3), the MA register is changed to output mode, and the contents of the special registers are displayed again in (4). This time, the contents of the PA register have changed. Note that the contents of PA, PB, PC, PD, and PF special registers are not always changed directly because of the contents of the MA, MB, MCC, MC, MM, and MF mode registers.

Example 2: When the contents of the MB register are first set to output mode after which the contents of the PB register are changed.

```
*SPR ↵

 PA PB PC PD PF MKH MKL RXB CR0 CR1 CR2 CR3 ECNT ECPT
 12 00 00 CB 00 07  FE  FF  FF  FF  FF  FF  0000 FFFF

*MDR C MB ↵

 MB --=0 ↵

*SPR C PB ↵

 PB 00=34 ↵

*SPR ↵

 PA PB PC PD PF MKH MKL RXB CR0 CR2 CR2 CR3 ECNT ECPT
 12 34 00 CB 00 07  FE  FF  FF  FF  FF  FF  0000 FFFF

*
```

Example 3: When special-register-namel change command is input and then space is input.

```
*SPR C ↵

 PA 00=00 ↵
 PB 00=00 ↵
 PC 00=00 ↵
 PD CB=CB ↵
 PF 00=00 ↵
 MKH 07=07 ↵
 MKL FE=FE ↵
 TXB --=-- ↵
 TM0 --=-- ↵
 TM1 --=-- ↵
 ETM0 --=-- ↵      ——— The special register not recognized
 ETM1 --=-- ↵            as special-register-name2 cannot be
                         read out, therefore, "--" is
*                        indicated.
```

Example 4: When an attempt is made to change a read-only special register.

```
*SPR C RXB ↵

 READ ONLY REGISTER

*
```

Special registers not listed under "special-register-namel" are read-only registers and can therefore not be changed.

3.6.3 Display

SPR [_D [_special-register-name2]]

Displays the contents of the specified special register. When "special-register-name2" is omitted, the contents of all special-register-name2 registers are displayed.

Examples:

*SPR D RXB ↵ (1)
*SPR D ↵ (2)
*SPR ↵ (3)

(1) Displays the contents of the RXB register.
(2)(3) Displays the contents of all special-register-name2 registers.

Examples:

```
*SPR ↵

 PA PB PC PD PF MKH MKL RXB CR0 CR1 CR2 CR3 ECNT ECPT
 00 00 00 DB 00 07  FE  FF  FF  FF  FF  FF  0000 FFFF

*

*SPR D↵

 PA PB PC PD PF MKH MKL RXB CR0 CR1 CR2 CR3 ECNT ECPT
 00 00 00 DB 00 07  FE  FF  FF  FF  FF  FF  0000 FFFF

*

*SPR D PA↵

 PA 00

*

*SPR D MKH↵

 MKH 07

*

*SPR D IXB ↵
 WRITE ONLY REGISTER
*
```

When special-register-name1 is specified, the following message is displayed.

WRITE ONLY REGISTER

3.7 LOD (Load) Command

Format

*LOD [{ _ / = } input-device-name] [␣&bias)↵]

Function: Loads the HEX format data to the mapped memory from the specified device. A bias can be specified.

Main command: LOD
Subcommand : None
Operand : input-device-name Input device name
Option : bias Bias address

Input devices are TTY1 and TTY2 of serial pins (RS-232C) on the IE controller module, and either one of them is specified.

3.7.1 Load

LOD␣TTY?$bias↵

* "TTY?" : TTY1 or TTY2.
* Space preceding "TTY?" can be "=".
* $bias is value input, and dependent on the suffix. When omitted, bias is 0.

Examples:

```
*MEM F 0,FFH 00↵

*MEM D XXH↵
0000   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0050   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00

*LOD TTY1↵
```

CTRL-C or ESC key input

```
*
*MEM D XXH ↵
0000   FF FF FF FF FF 7A 33 26  00 FF FF FF FF FF FF FF
0010   21 1E 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
0030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
0050   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
0060   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
0070   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
0080   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
0090   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
00A0   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
00B0   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
00C0   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
00D0   09 00 01 02 03 04 05 06  07 08 09 00 01 02 03 04
00E0   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
00F0   55 AA 33 88 55 AA 33 88  55 AA 33 88 55 AA 33 88

*

*MEM F 0.FFH 0 ↵

*MEM D XXH ↵
0000   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0050   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0080   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00A0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00B0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00C0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00D0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00E0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00F0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*

*MEM F XXX 0 ↵

*MEM D 800.8FF ↵
 0800  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 0810  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 0820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 0830  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 0840  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 0850  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 0860  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00

*LOD=TTY1$800 ↵                          CTRL-C or ESC key input

*MEM D 800.8FF ↵
 0800  11 22 33 44 55 66 77 88  11 22 33 44 55 66 77 88
 0810  11 22 33 44 55 66 77 88  11 22 33 44 55 66 77 88
 0820  11 22 33 44 55 66 77 88  11 22 33 44 55 66 77 88
 0830  11 22 33 44 55 66 77 88  11 22 33 44 55 66 77 88
 0840  11 22 33 44 55 66 77 88  11 22 33 44 55 66 77 88
 0850  11 22 33 44 55 66 77 88  11 22 33 44 55 66 77 88
 0860  1[ 22 33 44 55 66 77 88  11 22 33 44 55 66 77 88

*                                        CTRL-C or ESC key input
```

3.8 SAV (Save) Command

Format: *SAV [␣output-device-name] [{{-}{=}}partition]↵

Function: Saves the contents of the program memory specified by 'partition' in HEX format to the specified device.

Main command: SAV
Subcommand : None
Operand : output-device-name Output device name
: partition Save area
Option : None

3.8.1 Save

SAV␣TTY?␣partition↵

* "TTY?" is TTY1 (CH1) or TTY2 (CH2).
* Space between "TTY?" and "partition" can be "=".
* "partition" can be omitted; when omitted, all mapped area is saved.

Examples:

```
*MEM D XXH↵
0000   FF FF FF FF FF 7A 33 26  00 FF FF FF FF FF FF FF
0010   21 1E 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020   FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
0030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040   05 06 07 08 09 00 01 02  03 04 05 06 07 08 09 00
0050   01 02 03 04 05 06 07 08  09 00 01 02 03 04 05 06
0060   07 08 09 00 01 02 03 04  05 06 07 08 09 00 01 02
0070   03 04 05 06 07 08 09 00  01 02 03 04 05 06 07 08
                                      \ CTRL-C or ESC key
```

```
*SAV TTY1=0,FFH ↵
:10000000FFFFFFFFFF7A332600FFFFFFFFFFFFFF29
:10001000211E000000000000000000000000000A1
:10002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
:10003000000000000000000000000000000000C0
:10004000050607080900010203040506070809006O
:10005000010203040506070809000102030405065E
:10006000070809000102030405060708090001O248
:10007000030405060708090001020304050607O832
:10008000090001020304050607080900010203O430
:10009000050607080900010203040506070809O010
:1000A000010203040506070809000102030405060E
:1000B000070809000102030405060708090001O2F8
:1000C000030405060708090001020304050607O8F2
:1000D000090001020304050607080900010203O4E0
:1000E000050607080900010203040506070809OC0
:1000F00055AA338855AA338855AA338855AA338818
:00000001FF

*SAV TTY1 XXH ↵
:10000000FFFFFFFFFF7A332600FFFFFFFFFFFFFF29
:10001000211E000000000000000000000000000A1
:10002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
:10003000000000000000000000000000000000C0
:10004000050607080900010203040506070809006O
:10005000010203040506070809000102030405065E
:10006000070809000102030405060708090001O248
:10007000030405060708090001020304050607O832
:10008000090001020304050607080900010203O430
:10009000050607080900010203040506070809O010
:1000A000010203040506070809000102030405060E
:1000B000070809000102030405060708090001O2F8
:1000C000030405060708090001020304050607O8E2
:1000D000090001020304050607080900010203O4E0
:1000E000050607080900010203040506070809OC0
:1000F00055AA338855AA338855AA338855AA338818
:00000001FF

*SAV TTY2=0,FFH ↵

*
```

When a CRT is connected, codes are displayed in HEX format.

Data is sent to TTY2 after which display returns to "*".

3.9 RUN (Emulation) Command

Format:

*RUN [␣{N / B}] [␣addr] ↵

*RUN␣S [␣addr] [, step-No.] ↵

*RUN␣T [␣addr] [, {step-No. / ※}] [$ {D / E}] [$R] ↵

※=register-name {= / > / < / =< / >= / <> / >< / => / <=} value

Function: Performs real-time emulation or step emulation.

Main command:	RUN	
Subcommand	: N	Normal
	: B	Break
	: S	Step
	: T	Trace
Operand	: addr	Value (2 bytes) that can be input as expression
	: step-No.	Number of steps (1 byte)
	: value	Numeric value
Option	: D	Display disable
	: E	Display enable
	: R	Register display specification

3.9.1 Emulation commands

The following emulation commands are available:

*RUN N	Real-time execution without break
*RUN B	Real-time execution with break
*RUN S	Step execution
*RUN T	Single-step execution (with display)

3.9.2 RUN N (real-time execution without break)

RUN␣N [␣addr]↵

Examples:

*RUN ↵ (1)

*RUN N 0 ↵ (2)

(1) Starts real-time emulation from the address indicated by the current program counter.

(2) Starts real-time emulation from address 0.

Example:

```
*RUN N 0↵

USER-SYSTEM VDD-ON VCC-ON

JUST MOMENT
EMULATION START AT 0000

ESC BREAK        ——— Forced break by ESC key input
TERMINATED

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  08 59 18 18 26 25 00 00 0000 0603 1E0B
    V' A' B' C' D' E' H' L' EA'
    00 FF 20 30 00 04 06 FD 014C

*
```

* However, during HLT instruction execution, NON BREAK is displayed. The RES command must be input next. Program execution can be interrupted by inputting ESC key as shown in the above. Note that when CTRL-C is input, the status becomes command wait, although

the CPU continues execution. Most of the commands cannot be accepted in this condition.

3.9.3 RUN B (real-time execution with break)

```
RUN_B_addr↵
```

When this command is input, the IE-7811H checks the power supply and the clock on the user system, then outputs the message EMULATION START AT addr, and starts emulation at the input address (if addr is omitted, the current contents of the PC are taken as the start address specification). Forced break of emulation can be achieved by inputting ESC key. Inputting CTRL-C returns the display to "*"; however, the CPU continues emulation. When ESC key is input, the register and disassembler are dumped, and then wait for input. If space is then input, single-step emulation is performed (when single-step emulation is executed, trace data is not displayed even if the display command is executed). When CR is input, the display returns to "*".
The difference between the RUN N and RUN B commands is that the RUN N command executes emulation regardless of the break condition, while the RUN B command breaks when the break conditions specified in advance are satisfied during emulation. The causes of the breaks are then displayed.

Note: If RAE (E= enable, D= disable) is set to E when initializing, break cannot be made in the range 0FF00 to 0FFFFH. A break can be made, however, if the setting is changed to D and the break conditions are specified.

Examples:

```
*BRA A=XX↵

*BRO BRA ↵

*BRM BRO,BRA ↵

*RUN B 0 ↵

 USER-SYSTEM VDD-ON VCC-ON

 JUST MOMENT
 EMULATION START AT 0000

 NORMAL BREAK
 TERMINATED

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  08 59 18 18 26 25 00 00 0000 0603 0001
     V' A' B' C' D' E' H' L' EA'
     UU FF 2U 3U UU 04 06 FD 014C

 ONE STEP EMULATION STANDBY

 ADRS    OBJECT          MNEMONIC
 0001    00              NOP

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  08 59 18 18 26 25 00 00 0000 0603 0002
     V' A' B' C' D' E' H' L' EA'
     00 FF 20 30 00 04 06 FD 014C

 ADRS    OBJECT          MNEMONIC
 0002    00              NOP

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  08 59 18 18 26 25 00 00 0000 0603 0003
     V' A' B' C' D' E' H' L' EA'
     00 FF 20 30 00 04 06 FD 014C

 ADRS    OBJECT          MNEMONIC
 0003    00              NOP

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  08 59 18 18 26 25 00 00 0000 0603 0004
     V' A' B' C' D' E' H' L' EA'
     00 FF 20 30 00 04 06 FD 014C↵
*
```

Single-step execution by inputting space

CR is input to terminate single-step operation

The display returns to the prompt if the return key is immediately input. Single-step operation is achieved by inputting the space key. However, traced data are erased.

3.9.4 RUN S (real-time step execution)

RUN␣S [␣addr] [, step-No.]

After command wait state, input as "RUN S addr, step-No." following "*". The IE-781lH outputs the message EMULATION START AT addr, and executes emulation for the number of specified steps from the input addr (when addr is omitted, one step is executed from the current value of the program counter, when step No. is omitted, one step emulation is entered from the addr. A maximum 255 steps can be specified as step-No.

Examples: *RUN S 0, 60 ↵ (1)
*RUN S , 50 ↵ (2)
*RUN S 1000 ↵ (3)
*RUN S ↵ (4)

(1) Executes 60H steps of the program in real-time from address 0H and then enters single-step mode.
(2) Executes 50H steps in real-time from the current value of the program counter and then enters single-step mode.
(3) Single-step operation from address 1000H.
(4) Single-step operation from the current value of the program counter.

Examples:

```
*RUN S 0,60 ↵

 USER-SYSTEM VDD-ON VCC-ON

 JUST MOMENT
 EMULATION START AT 0000

 STEP BREAK
 TERMINATED

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  F6 1F D2 6F 01 20 A6 39 4CFD 0000 0006
```

```
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000    Space input for single-step
ONE STEP EMULATION STANDBY  ———     operation

ADRS      OBJECT                MNEMONIC
0006      41                    INR     A

PSW V  A  B  C  D  E  H  L  EA   SP   PC
10  F6 20 D2 6F 01 20 A6 39 4CFD 0000 0007
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000    ——— CR input to terminate single-step
                                        operation
*RUN S ,50 ↵

USER-SYSTEM VDD-ON VCC-ON

JUST MOMENT
EMULATION START AT 0007

STEP BREAK
TERMINATED

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 3A D2 6F 01 3B A6 39 4CFD 0000 0006
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000

ONE STEP EMULATION STANDBY  ——— Space input

ADRS      OBJECT                MNEMONIC
0006      41                    INR     A

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 3B D2 6F 01 3B A6 39 4CFD 0000 0007
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000    ——— CR input

*RUN S 1000 ↵

ONE STEP EMULATION START

ADRS      OBJECT                MNEMONIC
1000      00                    NOP

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 3B D2 6F 01 3B A6 39 4CFD 0000 1001
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000    ————————Space input


ADRS      OBJECT                MNEMONIC
1001      00                    NOP

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 3B D2 6F 01 3B A6 39 4CFD 0000 1002
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000    ——— CR input


*
```

[PROGRAM]

```
*RUN S ↵

 ONE STEP EMULATION START

 ADRS      OBJECT                MNEMONIC
 1002      00                    NOP

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  F6 3B D2 6F 01 3B A6 39 4CFD 0000 1003
     V' A' B' C' D' E' H' L' EA'
     00 00 00 00 8E 27 8E 27 0000

 ADRS      OBJECT                MNEMONIC
 1003      00                    NOP

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  F6 3B D2 6F 01 3B A6 39 4CFD 0000 1004
     V' A' B' C' D' E' H' L' EA'
     00 00 00 00 8E 27 8E 27 0000

 ADRS      OBJECT                MNEMONIC
 1004      00                    NOP

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  F6 3B D2 6F 01 3B A6 39 4CFD 0000 1005
     V' A' B' C' D' E' H' L' EA'
     00 00 00 00 8E 27 8E 27 0000
*
```

Single-step operation by inputting space

CR input to terminate single-step operation

3.9.5 RUN T (single-step execution with display)

RUN_T [_addr] [, { step-No. / ※ }] [$ { D / E }] [$R]

※= register-name { = / > / < / =< / >= / <> / >< / => / <= } value

Examples: *RUN T 100, 10ER ↵ (1)
*RUN T , L = 0 ↵ (2)
*RUN T ↵ (3)

(1) Performs single-step emulation for 10H steps from address 100H displaying the addresses, data mnemonics, and register contents. Execution then enters the single-step mode.

(2) Executes the program from the current value of the program counter until the contents of the L register becomes 0. Execution then enters the single-step mode.

(3) The current contents of the program counter are specified as the start address, and single-step operation is performed.

Examples:

```
*RUN T 0,4$R ↵

 ADRS     OBJECT              MNEMONIC
 0000     240001              LXI     D,0010OH

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  F6 01 D2 6F 01 00 A6 39 4CFD 0000 0003
     V' A' B' C' D' E' H' L' EA'
     00 00 00 00 8E 27 8E 27 0000

 ADRS     OBJECT              MNEMONIC
 0003     6900                MVI     A,00000H
```

```
PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 00 D2 6F 01 00 A6 39 4CFD 0000 0005
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000
ADRS      OBJECT              MNEMONIC
0005      3C                  STAX    D+

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 00 D2 6F 01 01 A6 39 4CFD 0000 0006
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000

ADRS      OBJECT              MNEMONIC
0006      41                  INR     A

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 01 D2 6F 01 01 A6 39 4CFD 0000 0007
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000

ONE STEP EMULATION STANDBY  ——— Space input for single-step
                                operation
ADRS      OBJECT              MNEMONIC
0007      FD                  JR      00005H

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 01 D2 6F 01 01 A6 39 4CFD 0000 0005
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000    ——— CR input to terminate single-step
                                        operation
*RUN T 0,PC=6$R↵

ADRS      OBJECT              MNEMONIC
0000      240001              LXI     D,00100H

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 01 D2 6F 01 00 A6 39 4CFD 0000 0003
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000

ADRS      OBJECT              MNEMONIC
0003      6900                MVI     A,00000H

PSW V  A  B  C  D  E  H  L  EA   SP   PC
08  F6 00 D2 6F 01 00 A6 39 4CFD 0000 0005
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000

ADRS      OBJECT              MNEMONIC
0005      3C                  STAX    D+

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  F6 00 D2 6F 01 01 A6 39 4CFD 0000 0006
    V' A' B' C' D' E' H' L' EA'
    00 00 00 00 8E 27 8E 27 0000

ONE STEP EMULATION STANDBY  ——— Space input
```

```
 ADRS      OBJECT                MNEMONIC
 0006      41                    INR      A
 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  F6 10 D2 6F 01 01 A6 39 4CFD 0000 0007
     V' A' B' C' D' E' H' L' EA'
     00 00 00 00 8E 27 8E 27 0000      ——— CR input

*REG C A ↵

 A 01=0 ↵

*RUN T 0,A>2 ↵

 ADRS      OBJECT                MNEMONIC
 0000      240001                LXI      D,00100H

 ADRS      OBJECT                MNEMONIC
 0003      6900                  MVI      A,00000H

 ADRS      OBJECT                MNEMONIC
 0005      3C                    STAX     D+

 ADRS      OBJECT                MNEMONIC
 0006      41                    INR      A

 ADRS      OBJECT                MNEMONIC
 0007      FD                    JR       00005H

 ADRS      OBJECT                MNEMONIC
 0005      3C                    STAX     D+

 ADRS      OBJECT                MNEMONIC
 0006      41                    INR      A

 ADRS      OBJECT                MNEMONIC
 0007      FD                    JR       00005H

 ADRS      OBJECT                MNEMONIC
 0005      3C                    STAX     D+

 ADRS      OBJECT                MNEMONIC
 0006      41                    INR      A

 ONE STEP EMULATION STANDBY   ——— Space input

 ADRS      OBJECT                MNEMONIC
 0007      FD                    JR       00005H

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  F6 03 D2 6F 01 03 A6 39 4CFD 0000 0005
     V' A' B' C' D' E' H' L' EA'
     00 00 00 00 8E 27 8E 27 0000      ——— CR input
*REG ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  F6 03 D2 6F 01 03 A6 39 4CFD 0000 0005
     V' A' B' C' D' E' H' L' EA'
     00 00 00 00 8E 27 8E 27 0000

*
```

3.9.6 Single-step operation

Input the space key to start single-step operation. When the "CR" key is input, the single-step operation is terminated and the display returns to the prompt.

```
ONE STEP EMULATION START

ADRS      OBJECT              MNEMONIC
EB45      00                  NOP

PSW V  A  B  C  D  E  H  L  EA   SP   PC
00  08 59 18 18 26 25 00 00 0000 0603 EB46
    V' A' B' C' D' E' H' L' EA'
    00 FF 20 30 00 04 06 FD 014C      ——— After CR is input, status returns
*                                           to command wait state.


*RUN T 100,2$E ↵

ADRS      OBJECT              MNEMONIC
0100      340010              LXI     H,01000H

ADRS      OBJECT              MNEMONIC
0103      6091                XRA     A,A

ONE STEP EMULATION STANDBY          ——— Space input

ADRS      OBJECT              MNEMONIC
0105      4DD2                MOV     MA,A

PSW V  A  B  C  D  E  H  L  EA   SP   PC
40  1C 00 6E FE 18 90 10 00 0000 0700 0107
    V' A' B' C' D' E' H' L' EA'
    00 FF 00 10 07 00 90 F0 014C      ——— CR input

*RUN T 100,2$D ↵

ONE STEP EMULATION STANDBY          ——— Space input

ADRS      OBJECT              MNEMONIC
0105      4DD2                MOV     MA,A

PSW V  A  B  C  D  E  H  L  EA   SP   PC
40  1C 00 6E FE 18 90 10 00 0000 0700 0107
    V' A' B' C' D' E' H' L' EA'
    00 FF 00 10 07 00 90 F0 014C      ——— CR input
```

```
*RUN T 100,2$E$R ↵

 ADRS      OBJECT                MNEMONIC
 0100      340010                LXI     H,01000H

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 44  1C 00 6E FE 18 90 10 00 0000 0700 0103
     V' A' B' C' D' E' L' EA'
     00 FF 00 10 07 00 90 F0 014C

 ADRS      OBJECT                MNEMONIC
 0103      6091                  XRA     A,A

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 40  1C 00 6E FE 18 90 10 00 0000 0700 0105
     V' A' B' C' D' E' H' L' EA'
     00 FF 00 10 07 00 90 F0 014C

 ONE STEP EMULATION STANDBY      ←———— Space input

 ADRS      OBJECT                MNEMONIC
 0105      4DD2                  MOV     MA,A

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 40  1C 00 6E FE 18 90 10 00 0000 0700 0107
     V' A' B' C' D' E' H' L' EA'
     00 FF 00 10 07 00 90 F0 014C     ←——CR input

*RUN T 100,2$D$R ↵

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 44  1C 00 6E FE 18 90 10 00 0000 0700 0103
     V' A' B' C' D' E' H' L' EA'
     00 FF 00 10 07 00 90 F0 014C

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 40  1C 00 6E FE 18 90 10 00 0000 0700 0105
     V' A' B' C' D' E' H' L' EA'
     00 FF 00 10 07 00 90 F0 014C

 ONE STEP EMULATION STANDBY      ←—— Space input

 ADRS      OBJECT                MNEMONIC
 0105      4DD2                  MOV     MA,A

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 40  1C 00 6E FE 18 90 10 00 0000 0700 0107
     V' A' B' C' D' E' H' L' EA'
     00 FF 00 10 07 00 90 F0 014C     ←—— CR input

*
```

3.10 BR? (Break) Command

Format:

```
*BRA [_A=addrs] [_V=values]
                [_C=cond] [_L=loop]
*BRD [_data]
*BRE [_count]
*BRT [_time]
*BR{0/3} [_BRA, BRD.....]
*BRM [_BR0, BR1.....]
```

Function: Sets or displays the contents of various break registers.

Main command:	BR?		(?=A, D, E, T, M, 0 to 3)
Subcommand	: None		
Operand	: addrs		Address input with masking
	: cond		Break condition (two or more conditions can be set)
	: values		Break data conditions with masking (1 byte)
	: loop		Number of loops (1 to 255)
	: addr		Address input that can be treated as expression
	: count		2-byte input that can be treated as expression
	: data		1-byte input
	: time		Sets time (msec)
	: BR0 to BR3		Logical break register
	: BRA to BRT		Physical break register
	: BRM		Break mode register
Option	: None		

3.10.1 Outline of break command

The break commands can be divided into the following three types:

(1) Physical break registers

Physical break registers hold data that is to be directly set on hardware.

These registers are: BRA, BRD, BRE, and BRT.

(2) Logical break registers

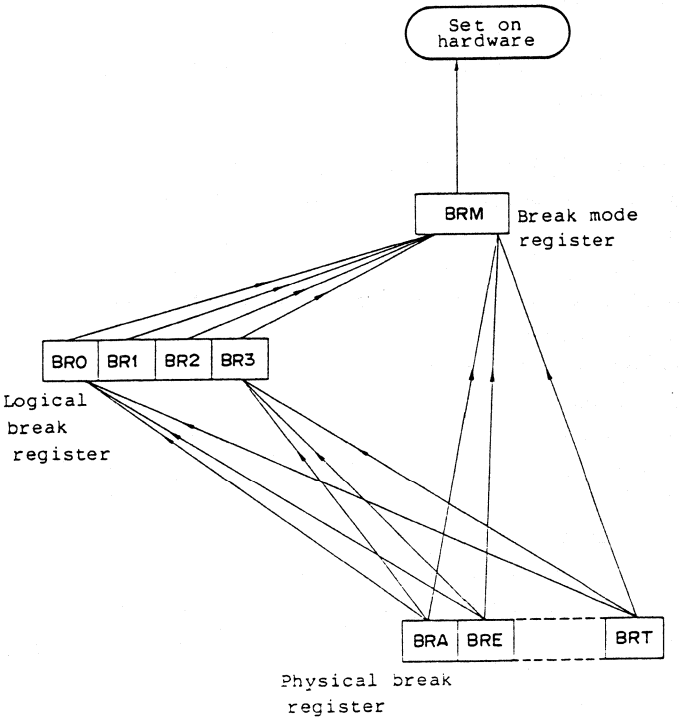
Logical break registers hold as data the logical combinations of the registers described in (1) above. These registers are: BR0, BR1, BR2, and BR3.

(3) Break mode register

This is the command that sets the break data indicated by (1) and (2) above on hardware. Until this command is executed, break data cannot be set on hardware even if data is modified as indicated in (1) and (2). The break mode register is BRM.

Note: When the power is turned on, all break registers are cleared and do not contain any data.

Simplified diagram of break commands



3.10.2 Physical break registers

The physical break registers are: BRA, BRD, BRE, and BRT. These registers hold data that is to be set on hardware. Break will not occur even if data is set on this register and RUN B command is executed.

(1) BRA command

BRA [␣A=addrs] [␣V=values]
[␣C=cond] [␣L=loop]↵

A ; Break address specification
V ; Break data specification
C ; Break condition specification
L ; Number of loops specification
addrs : Address field
Up to five addresses can be set by delimiting addr or partition with a space.
values; Data field
Data can be up to 8 bits in length, and can be expressed by X (however, decimal numbers cannot be expressed by X).
cond ; Condition field
These conditions may be selected (when omitted, no condition is assumed):
OP ; OP code fetch (M1 read)
RD ; Memory read
WR ; Memory write
MR ; Memory request
NC ; No condition
loop ; Loop specification field
Sets the number of loops of conditions set by the address, data, and condition fields. 1 to 255 loops can be set.

Examples:

*BRA A=10 ↵ (1)

*BRA ↵ (2)

*BRA A=0, 0FH, 30H, 3FH, 50H, 60H ↵ (3)

*BRA V=55T ↵ (4)

*BRA C=WR L=3 ↵ (5)

*BRA A=20H V=0 L=5T ↵ (6)

*BRA A=1XXH V=1X0X1Y C=RD L=2Q ↵ (7)

(1) Breaks when address 10 is accessed.
If the suffix specification is omitted, the current suffix is used to evaluate the address value. Therefore, if the current suffix is binary, address 2H is assumed; if octal, 8H; if decimal, AH; and if HEX, 10H.

(2) Displays the break condition stored in the BRA register.

(3) Breaks when addresses 0 to 0FH, 30 to 3FH, 50H, or 60H is accessed.

(4) Breaks when data 55T is accessed.

(5) Breaks when data is written three times.

(6) Breaks when data 0H is accessed five times at address 20H.

(7) Within address 100 to 1FFH, breaks when data 11H, 13H, 19H, or 1BH is read twice.

Example:

```
*BRA ↵
 --        ———————— Not specified
*BRA A=10,          ┌ Set condition ↵
*BRA ↵              └
 A=10H     ———— Displays suffix at the time of setting
*BRA A=1XXH V=1X0XY C=RD L=2Q, ↵
*BRA ↵              ——————————— Set condition
 A=1XXH V=1X0X1Y C=RD L=2Q
*
```

(2) BRD command

BRD data↵

The BRD command is an optional break command which sets a break condition using data of the eight external sense clips. Only 1-byte data can be input.

Examples: *BRD 5AH (1)
*BRD ↵ (2)

(1) Sets external sense clip data 5AH as the break condition.
(2) After command input prompt "*", inputting "BRD ↵ " performs carriage return and the data set by BRD command is displayed. Display then returns to "*". When no data has been set, "--" is displayed.

Example:

```
*BRD↵
 --
*BRD 5AH↵
*BRD↵
 5AH
*
```

(3) BRE command

BRE count↵

The BRE command sets the number of times (count) that an OP code is fetched as the break condition.

Examples: *BRE 1000H ↵ (1)
*BRE ↵ (2)

(1) Specifies 1000H as the OP code fetch count at which break occurs. The permissible range of values for "count" is 3 to 0FFFFH (2 bytes).

(2) After command input prompt "*", inputting "BRE ↵ " performs carriage return and the currently set data is displayed. The display then returns to "*". When no data have been set, "-" is displayed.

Example:

```
*BRE↵
 --
*BRE 1000↵
*BRE↵
 1000H
*
```

(4) BRT command

BRT time↵

When timer is used as break condition, this command sets the time from the start of emulation to the point at which break occurs. The range of the time that can be set is 1 to 0FFFFH ms.

Examples: *BRT 1300H (1)
*BRT ↵ (2)

(1) Specifies the break timer setting as 1300H ms. Time is specified as 2-byte data.
(2) After command input prompt "*", inputting "BRT ↵ " performs carriage return and currently set data is displayed. Display then returns to "*". If no data have been set, "--" is displayed.

Example:

```
*BRT↵
 --
*BRT 1300↵
*BRT↵
 1300H

*
```

3.10.3 Logical break registers

BR{0 | ... | 3} [␣BRA, BRD.....]↵

("BRA, BRD..." indicates that one or more registers can be selected from the BRA, BRD, BRE, and BRT physical break registers.)

BR0 to BR3 are available as logical break registers; all have the same functions. It is possible to simplify specification of the break condition by the BRM command. This is done by setting in the logical register those physical registers to be used to specify the break condition break. When the logical register is specified in the BRM command, it is as if all the physical registers had actually been specified.

Examples: *BR0 BRA, BRD, BRE, BRT ↵ (1)
*BR0 ↵ (2)

(1) Following the command input prompt "*", inputting "BR0 BRA ← " sets the physical break registers specified in the logical break register. When inputting two or more physical break registers, use a comma to delimit (the same applies to BR1 to BR3). The break registers break when any condition is satisfied.

(2) Following the command input prompt "*", inputting "BR0 ← " performs carriage return and the physical break registers currently stored in the logical break register are displayed. When no registers have been set, "--" is displayed and display returns to "*".

Example:

```
*BR0↵
 --
*BR0 BRA,BRD,BRE,BRT↵
*BR0↵
 BRA,BRD,BRE,BRT
*
```

3.10.4 Break mode register

BRM [BRA, BRD·····] ↵

("BRA, BRD..." indicates that one or more registers can be selected from BRA, BRD, BRE, BRT, BR0, BR1, BR2, and BR3)

The BRM command sets the break condition set in the physical break registers, and logical break registers on hardware before the execution of emulation with break (RUN B).

Examples:
```
*BRM BRA, BR0, BR1 ↵        } (1)
*BRM BR0, BR1, BR2, BR3 ↵   } (1)
*BRM ↵                        (2)
```

(1) Input as "BRM BRA, BR0, BR1 ↵" after command input prompt "*". Input physical and/or logical register names after "BRM ". When two or more register names are input, use "CR" to delimit. Registers can be input in any order.

(2) After command input prompt "*", inputting "BRM ↵" performs carriage return and the names of the currently set logical and physical registers are displayed.

```
*BRM ↵
 —
*BRM BRA,BR0,BR1 ↵
*BRM ↵
 BRA,BR0,BR1
*BRM BRA,BRD,BR0,BR1,BR2,BR3,BRT ↵
*BRM ↵
 BRA,BRD,BR0,BR1,BR2,BR3,BRT
*BRM BR0,BR1,BR2,BR3 ↵
*BRM ↵
 BR0,BR1,BR2,BR3
*

*BRA A=35 V=89 C=NC L=1 ↵

*BRT 300 ↵

*BRM BRA,BRT

*RUN B 0 ↵
```

```
USER-SYSTEM VDD-ON VCC-ON

JUST MOMENT
EMULATION START AT 0000

NORMAL BREAK
TERMINATED

PSW V  A  B  C  D  E  H  L  EA   SP   PC
48  08 AA 02 03 04 05 20 00 0000 3000 0037
    V' A' B' C' D' E' H' L' EA'
    80 10 20 30 40 50 60 70 014C

ONE STEP EMULATION START↵

*

*TRD -24↵

ADDR   OBJECT           MNEMONIC          PA PB
002E   41               INR     A         00 00
002F   FD               JR      00035H    00 00
002D   3D               STAX    H+        00 00
002E   41               INR     A         00 00
002F   FD               JR      00035H    00 00
002D   3D               STAX    H+        00 00
002E   41               INR     A         00 00
002F   FD               JR      00035H    00 00
002D   3D               STAX    H+        00 00
002E   41               INR     A         00 00
002F   FD               JR      00035H    00 00
002D   3D               STAX    H+        00 00
002E   41               INR     A         00 00
002F   FD               JR      00035H    00 00
002D   3D               STAX    H+        00 00
002E   41               INR     A         00 00
002F   FD               JR      00035H    00 00
002D   3D               STAX    H+        00 00
002E   41               INR     A         00 00
002F   FD               JR      00035H    00 00
0030   747E20           EQ1     H,00020H  00 00
0033   F9               JR      00031H    00 00
0034   C0               JR      00033H    00 00
0035   69AA             MV1     A,000AAH  00 00

*
```

3.11 TR? (Trace) Command

Format:

```
*TRC [_{M | I}] ↵

*TRD [_{ALL | line-No. | -line-No.}] ↵

*TRM [_{NON | ALL | TRX}] ↵

*TRP [_{O | N | line-No. | -line-No.}] ↵

*TRX [_A=addrx] [_V=value] [_C=cond]
                [_PA=values] [_PB=values]

*TRS [_{PB | EX}] ↵
```

Function: These commands perform various settings required for tracing the contents of the program executed by the emulation CPU.

Main command: TR? (? = C, D, M, P, X, S)
Subcommand : None
Operand : line-No. Number of pointers (number of lines) to be changed.
: O (old) Location at which tracing started
: N (new) Location at which tracing ended
: ALL Displays all tracing results
: M Displays machine cycle trace
: I Displays instruction trace
: PB Port B
: EX External sense clip
Option : None

3.11.1 Outline of trace commands

The IE-781lH has these six trace commands: TRC, TRD, TRM, TRP, TRX, and TRS.

TRC : Sets the trace cycle displayed
TRD : Displays the trace data
TRM : Specifies the trace mode
TRP : Moves the trace pointer
TRX : Sets the trace condition
TRS : Selects the trace source (either PB or EX)

The trace RAM traces the result of emulation execution per machine cycle for up to 1023 levels.
When power is applied to the IE-781lH, the trace cycle (TRC) is set to machine cycle (M) and the trace mode (TRM) is set to non-trace (NON). If the TRD command is executed when there is no data to be traced, the message "NON TRACE DATA" is output.

3.11.2 TRC command (trace cycle)

TRC [_ { M / I }] ↵

The TRC command selects the format for the dump trace data (TRD command). That is, whether the data is to be displayed for machine cycle or instruction cycle.

M : Machine cycle
I : Instruction cycle

The current display specification can be displayed by inputting "TRC ↵". M or I is displayed after the carriage return.

Examples:

```
*TRD ALL ↵                    ———————— Displays in machine cycle

 ADRS  DATA  CYC.  PA  PB
 0100  34    OP    00  00
 0101  00    RD    00  00
 0102  10    RD    00  00
 0103  60    OP    00  00
 0104  91    RD    00  00
 0105  4D    OP    00  00
 0106  D2    RD    00  00
 0107  6B    OP    FF  FF
 0108  FF    RD    FF  FF
 0109  2D    OP    FF  FF
 1000  11    RD    FF  FF
 010A  4D    OP    FF  FF
 010B  C0    RD    FF  FF
 010C  53    OP    11  11
 010D  54    OP    11  11
 010E  09    RD    11  11
 010F  01    RD    11  11

*TRC I ↵
*TRD ALL ↵                    ———————— Displays in instruction cycle

 ADRS    OBECT      MNEMONIC          PA  PB
 0100    340010     LXI    H,01000H   00  00
 0103    6091       XRA    A,A        00  00
 0105    4DD2       MOV    MA,A       00  00
 0107    6BFF       MVI    C,000FFH   FF  FF
 0109    2D         LDAX   H+         FF  FF
 010A    4DC0       MOV    PA,A       FF  FF
 010C    53         DCR    C          11  11
 010D    540901     JMP    00109H     11  11

*
```

3.11.3 TRD command (trace dump)

```
TRD[ {ALL       } ]↵
     {line-No.  }
     {-line-No. }
```

The TRD command displays the result of real time-execution (RUN N/B/S). Refer to the description of the TRM command for the trace condition.

When "TRD line-No. ↵" or "TRD -line No. ↵" is input after the command input waiting "*", a carriage return is performed, a header is output, and the specified number of lines of trace data are then displayed. The trace pointer points to the last displayed line.

When "TRD ↵" is input, the trace data from the current pointer position to the end of the trace are displayed.

Input "TRD ALL ↵" to display all trace data.

(When the trace mode (TRM) has been set to NON, data will not be traced in real-time execution or in single-step execution. Thereore, if "TRD " is input, "NON TRACE DATA" is displayed and the display returns to "*".)

Input CTRL-C or ESC to abort the trace display. The point at which an abort is made becomes the current trace pointer position.

Examples:

```
*TRD ALL ↵
 ADRS  DATA  CYC.   PA  EX
 1011  12    RD     13  F6
 1012  13    RD     10  F7
 1013  14    RD     13  E6
 1014  21    RD     14  F5
 1019  12    FD     13  F2
 101A  13    RD     10  F3
 101B  14    RD     13  E0
 101C  21    RD     14  F1

*TRD -3 ↵

 ADRS  DATA  CYC.   PA  EX
 101A  13    RD     10  F3
 101B  14    RD     13  E0
 101C  21    RD     14  F1

*TRD ↵

 ADRS  DATA  CYC.   PA  EX
 101C  21    RD     14  F1

*TRD 3 ↵

 ADRS  DATA  CYC.   PA  EX
 101C  21    RD     14  F1
 NEWEST

*TRD -1 ↵

 ADRS  DATA  CYC.   PA  EX
 101C  21    RD     14  F1

*
```

3.11.4 TRP Command (trace pointer)

```
TRP [ _ { 0
          N
          line-No.
          -line-No. } ] ↵
```

The TRP command moves the trace pointer from its current location.

Immediately after real-time execution (RUN N, RUN B, or or RUN S), the trace pointer points to the latest trace data. The following changes can be made in the command input wait state ("*" is displayed):

(1) "TRP O ↵"

Moves the trace pointer to the oldest trace data.

(2) "TRP N ↵"

Moves the trace pointer to the newest (latest) trace data.

(3) "TRP line-No. ↵" or "TRP -line-No. ↵"

Moves the trace pointer the specified number of lines from the current location. However, ±1 indicates the current pointer location. Inputting "+" is not required to move the pointer toward the newest data.

```
*TRD ALL↵

 ADRS  DATA  CYC.   PA  PB
 0100  34    OP     00  00
 0101  00    RD     00  00
 0102  10    RD     00  00
 0103  60    OP     00  00
 0104  91    RD     00  00
 0105  4D    OP     00  00
 0106  D2    RD     00  00
 0107  6B    OP     00  00
 0108  FF    RD     00  00
 0109  2D    OP     00  00
 1000  00    RD     00  00

*TRP N↵

*TRD↵

 ADRS  DATA  CYC.   PA  PB
 1000  00    RD     00  00
```

```
*TRP 0 ↵

*TRD ↵

 ADRS  DATA  CYC.   PA  PB
 0100  34    OP     00  00
 0101  00    RD     00  00
 0102  10    RD     00  00
 0103  60    OP     00  00
 0104  91    RD     00  00
 0105  4D    OP     00  00
 0106  D2    RD     00  00
 0107  6B    OP     00  00
 0108  FF    RD     00  00
 0109  2D    OP     00  00
 1000  00    RD     00  00

*TRP ↵

 000B

*TRP -5 ↵

*TRD ↵

 ADRS  DATA  CYC.   PA  PB
 0106  D2    RD     00  00
 0107  6B    OP     00  00
 0108  FF    RD     00  00
 0109  2D    OP     00  00
 1000  00    RD     00  00

*
```

3.11.5 TRM command

$$\text{TRM}\left[\ \_\left\{\begin{matrix}\text{NON}\\ \text{ALL}\\ \text{TRX}\end{matrix}\right\}\right]\ \leftarrow$$

The TRM command specifies the trace condition.
The trace conditions are:

NON : Trace not performed.
ALL : Traces all trace data.
TRX : Traces according to the trace condition set by the TRX command.

Examples:

```
*TRM NON ↵
*TRM ALL ↵
*TRM TRX ↵
*TRM ↵
```

Input "TRM NON ↵ ", "TRM ALL ↵ ", or "TRM TRX ↵ " in the command input wait state ("*" is displayed). The specified mode is set and the display returns to "*".
Input "TRM ↵" to display the current condition. The currently specified trace condition is displayed and the display then returns to "*" after the carriage return.

3.11.6 TRX command

```
TRX [_A=addrs] [_V=values] [_C=cond]
                [_PA=values] [_PB=values] ↵
```

A : Sets the trace address
V : Sets the trace data
C : Sets the trace condition
PA : Sets the trace of port A
PB : Sets the trace of port B

addrs : Address field
The address or partition can be delimited with a space. A maximum of five locations can be specified.

values: Data field
"X" expressions are possible for up to 8 bits. (Decimal "X" expressions, however, are prohibited.)

cond : Condition field
These conditions can be selected. When omitted, "no condition" is assumed.
OP : OP code fetch (M1 read)
RD : Memory read
WR : Memory write
NC : No condition

Examples:
```
*TRX A=10 ↵                          (1)
*TRX ↵                               (2)
*TRX A=0, OFH 30H 3FH 50H 60H ↵      (3)
*TRX V=55T ↵                         (4)
*TRX C=WR ↵                          (5)
*TRX A=20H V=0 ↵                     (6)
*TRX A=1XXH V=1X0X1Y C=RD ↵          (7)
```

(1) Traces when address 10 is accessed. When the suffix is omitted, the address value is evaluated with the current suffix. If the current suffix is decimal, the address is 2H; if octal, 8H; if decimal, AH; and if HEX, 10H.
(2) Displays the trace condition specified by the TRX command.
(3) Traces when address 0 to 0FH, 30 to 3FH, 50H, or 60H is accessed.
(4) Traces when data 55T is accessed.
(5) Traces data write.
(6) Traces when data 0H is accessed at address 20H.
(7) Traces when data 11H, 13H, 19H, or 1BH is accessed within address range 100 to 1FFH.

Examples:

```
*TRX↵
 —
*TRX A=0,1FFF 3000 V=0 C=RD PA=0↵
*TRX↵
 A=0H,1FFFH 3000 V=0 C=RD PA=0
*
```

Trace Command Example:
The following examples show how to use the TRX command.

```
*TRX A=10XX↵

*TRM TRX↵

*BRA A=1010↵

*BRM BRA↵

*RUN B 1000↵

 USER-SYSTEM VDD-ON,VCC-ON

 JUST MOMENT
 EMULATION START AT 1000

 NORMAL BREAK
 TERMINATED

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 00  F6 3B D2 6F 01 3B A6 39 4CFD 0000 1011
     V' A' B' C' D' E' H' L' EA'
     00 00 00 00 8E 27 8E 27 0000
```

```
ONE STEP EMULATION STANDBY ↵

  *TRD ALL↵

 ADRS   DATA    CYC.   PA PB
 1000   00      OP     00 00
 1001   00      OP     00 00
 1002   00      OP     00 00
 1003   00      OP     00 00
 1004   00      OP     00 00
 1005   00      OP     00 00
 1006   00      OP     00 00
 1007   00      OP     00 00
 1008   00      OP     00 00
 1009   00      OP     00 00
 100A   00      OP     00 00
 100B   00      OP     00 00
 100C   00      OP     00 00
 100D   00      OP     00 00
 100E   00      OP     00 00
 100F   00      OP     00 00
 1010   00      OP     00 00

*TRD -10 ↵

 ADRS   DATA    CYC.   PA PB
 1007   00      OP     00 00
 1008   00      OP     00 00
 1009   00      OP     00 00
 100A   00      OP     00 00
 100B   00      OP     00 00
 100C   00      OP     00 00
 100D   00      OP     00 00
 100E   00      OP     00 00
 100F   00      OP     00 00
 1010   00      OP     00 00

*TRD 5 ↵

 ADRS   DATA    CYC.   PA PB
 1010   00      OP     00 00
 NEWEST

*
```

3.11.7 TRS command

TRS [{ PB / EX }] ↵

PB : Port B
EX : External sense clip

Examples: *TRS EX ↵ (1)
*TRS ↵ (2)

(1) Traces the part connected to the external sense clip.
(2) Displays the currently selected source name.

Examples:

```
*DAS 100,10F↵

 ADRS   OBJECT          MNEMONIC
 0100   340010          LXI     H,01000H
 0103   6091            XRA     A,A
 0105   4DD2            MOV     MA,A
 0107   6BFF            MVI     C,000FFH
 0109   2D              LDAX    H+
 010A   4DC0            MOV     PA,A
 010C   53              DCR     C
 010D   540901          JMP     00109H

*BRA A=110↵

*BRM BRA↵

*TRX A=101XH C=RD PA=1XH↵

*TRM TRX↵

*TRS PB↵          ————————Traces port B

*RUN B 100↵

 USER-SYSTEM VDD-ON VCC-ON

 JUST MOMENT
 EMULATION START AT 0100

 NORMAL BREAK
 TERMINATED
```

```
 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 14  02 24 80 FF 80 80 20 00 0000 0700 0113
     V' A' B' C' D' E' H' L' EA'
     00 FF BF FF 00 3D 00 00 014C

 ONE STEP EMULATION STANDBY

*TRD ALL ↵

 ADRS  DATA   CYC.    PA PB
 1011  12     RD      13 13
 1012  13     RD      10 10
 1013  14     RD      13 13
 1014  21     RD      14 14
 1019  12     RD      13 13
 101A  13     RD      10 10
 101B  14     RD      13 13
 101C  21     RD      14 14

*TRS EX ↵      ————————Traces external sense clip

*RUN B 100 ↵

 USER-SYSTEM VDD-ON VCC-ON

 JUST MOMENT
 EMULATION START AT 0100

 NORMAL BREAK
 TERMINATED

 PSW V  A  B  C  D  E  H  L  EA   SP   PC
 14  02 00 80 FF 80 80 20 00 0000 0700 0113
     V' A' B' C' D' E' H' L' EA'
     00 FF BF FF 00 3D 00 00 014C

 ONE STEP EMULATION STANDBY

*TRD ALL ↵

 ADRS  DATA   CYC.    PA EX
 1011  12     RD      13 F3
 1012  13     RD      10 F2
 1013  14     RD      13 F3
 1014  21     RD      14 F2
 1019  12     RD      13 F1
 101A  13     RD      10 F0
 101B  14     RD      13 F1
 101C  21     RD      14 E1


*
```

3.12 CLK (Clock) Command

Format:

*CLK [␣{I / U}]↵

Function: This command specifies the clock source for the emulation CPU.

Main command: CLK

Subcommand : I Internal clock : When the internal clock (fixed to 15MHz) of the IE-7811H is used.

: U External clock : When the clock of the target system is used. The clock frequency must be betweeen 4 to 15MHz.

3.12.1 Clock specification

CLK␣{U / I}↵

Specifies either the IE internal clock (15MHz) or the target system clock (4 to 15MHz) as the emulation CPU clock.

Examples: *CLK U ↵ (1)
*CLK I ↵ (2)

(1) Specifies the target system clock as the emulation CPU clock. For this, refer to "Emulation probe setting" to set the emulation probe switch.

(2) Specifies the IE system internal clock (15MHz) as the emulation CPU clock.

Note: When the clock is specified by the CLK command, the emulation CPU is simultaneously reset.

3.12.2 Clock display

CLK↵

Displays the name of the currently specified clock source.

Examples:

```
*CLK ↵
  I
*CLK U ↵
  U
```

3.13 RES (Reset) Command

Format: *RES [_H]↵

Function: This command resets the emulation CPU or the IE-781lH system hardware.

Main command: RES
Subcommand : H Resets IE-781lH system
Operand : None
Option : None

3.13.1 Emulation CPU reset

RES↵

Resets the emulation CPU.

Example:

```
*RES ↵
*
```

3.13.2 Hardware reset

RES H↵

Resets the IE-7811H system. This has the same function as the reset switch on the controller module.

Examples:

```
*RES H↵
 IE-7811H MONITOR VX.X[XX XXX XX] •
    COPY RIGHT (C) 198X NEC CORPORATION

 CPU 7811H
 EXTERNAL MEMORY SIZE 64K BYTE
 RAE (E:ENABLE.D:DISABLE)=E

 USER SYSTEM VDD-ON VCC-ON

*
```

3.14 MAT (Operation) Command

Format:

*MAT␣expression)

Function: Performs the operation of the input expression, and displays the result in hexadecimal, decimal, octal, and binary format.

Main command: MAT
Subcommand : None
Operand : expression — Expression of numeric value that can be processed as expression
Option : None

3.14.1 Operation command

MAT␣expression)

After this command is input, the result is displayed in hexadecimal, decimal, octal, and binary format. Afterward, the display returns to "*". The operation values are each composed of 2 bytes. The following shows typical examples.

Examples:

```
*MAT ((A*3)-2)*111Y↵
C4H,196T,304Q,11000100Y
*

*MAT 0AH+(18Q*10T)*11Y+100H↵
          └──────Only 0 to 7 can be used for octal format
 INPUT DATA ERROR
*

*MAT 10+10↵
11H,17T,21Q,10001Y
*MAT  10Q*2↵
10H,16T,21Q,10000Y
*MAT 10+(10Q*2)↵
20H,32T,40Q,100000Y
*

*MAT 0AH+(12Q*10T)11Y+1000H↵
                  └──────Operator is missing
 INPUT DATA ERROR
*

*MAT 0AH+(12Q*10T)*11Y+1000H↵
1136H,4406T,10466Q,1000100110110Y
*

*MAT F00H+100H↵
1000H,4096T,10000Q,1000000000000Y
*

*MAT 5*5*5*5↵
271H,625T,1161Q,1001110001Y
*MAT 5*5*5*5*5↵
C35H,3125T,6065Q,1100001101101Y
*MAT 5*5*5*5*5*5↵
3D09H,15625T,36411Q,11110100001001Y
*MAT 5*5*5*5*5*5*5↵
312DH,12589T,30455Q,1100010010110 1Y
*MAT 1234H AND 0FFH↵
34H,52T,64Q,110100Y
*
```

The MAT command can perform arithmetic, parenthetical, or logical operations.
When this command is executed, if no radix (H, T, Q, or Y) is attached to the end of the numeric value, the specification made by the SUF command is assumed.

Operators:

Arithmetic operators: + - * /

Logical operators : AND, OR, XOR, NOT

A maximum of 32 pairs of parentheses can be used.
However, when input values or the result of the operation exceeds 2 bytes, only the data of the lower 2 bytes are effective.

Operator priority

1	*, /
2	+, -
3	NOT
4	AND
5	OR, XOR

3.15 SUF (Numeric Value Radix Specification) Command

Format:

*SUF [_{H T Q Y}] ↵

Function: Specifies or displays the radix of the input value.

Main command	: SUF	
Subcommand	: H	Hexadecimal
	: T	Decimal
	: Q	Octal
	: Y	Binary
Operand	: None	
Option	: None	

3.15.1 Display and change of SUF (numeric value radix)

SUF ↵

Displays the radix of the input value when the current command is input. The display then returns to "*".

Example:

```
*SUF ↵
 H          : HEX is assumed as the current input value.
*
```

To change the radix of the numeric value, input "SUF A ↵" (A=H, T, Q, Y) after "*" (where A is either H, T, Q, or Y). The display then returns to "*" with nothing displayed. The values of the commands input afterward are processed with the radix specified by the SUF command unless H, T, Q, or Y is attached to the end of the value.

```
*SUF ↵
H
*MAT 10↵
10H.16T.20Q.10000Y
*SUF T↵

*MAT 10↵
AH.10T.12Q.1010Y
*SUF Q↵

*MAT 10↵
8H.8T.10Q.1000Y
*SUF Y↵

*MAT 10↵
2H.2T.2Q.10Y
*
*SUF A↵

 COMMAND FORMAT ERROR
*
```

3.16 ASM (Assemble) Command

Format:

*ASM [␣addr]↵

Function: Changes the memory contents after the specified address in assembler format.

Main command : ASM
Subcommand : None
Operand : addr (start address)
Option : None

3.16.1 Description of ASM command

The ASM (assemble) command can change the mapped program memory (RAM) in assembler format. However, this command cannot be executed for an unmapped program memory; if an attempt is made to do so, the message "NON-MAP AREA ACCESS" is output. The following describes the procedure.

(1) In the command input wait state ("*" is displayed), input "ASM addr ↵". The ASM command starts from the address specified by addr.

(2) The instruction of the address is disassembled, and the status returns to the command input wait state after "=" is output.

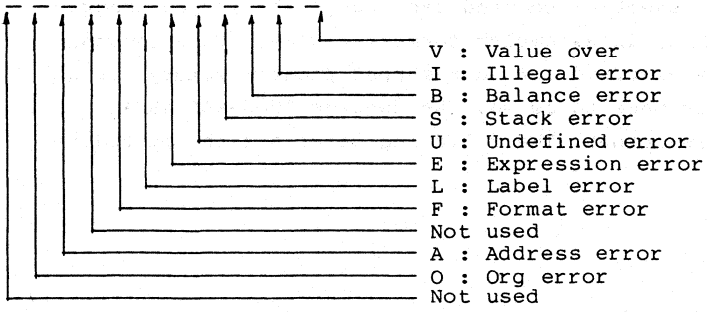
(3) When no change is desired, input "↵" to move to the next address, then go back to step (2). To make a change, input the mnemonic of the instruction, followed by "↵". Input "END↵ " to terminate the assembler command, after which the display returns to "*".

(4) If the mnemonic has been correctly input, its object code is output. The process then enters the command input wait state.
If the input mnemonic should be in error, the 4-byte NOP object error flag is output. The process then enters the command input wait state.

(5) When "↵" is input, object code is written to the memory, and the address is shifted to the next instruction. The process then returns to (2). When the space key is input, the assembled object is cancelled, and the process returns to (2).

3.16.2 Error flag description

When the input assembler has an error, "-" or an abbreviation of the error condition is output. The abbreviation may be for one of the following 10 errors:



When an error has occurred during assembling, the error code abbreviation is output instead of "-". The format error is output in this way:

```
- - - - F - - - - - - - -
```

The numeric value input for assembler conforms to the radix specified by the SUF command unless one of the radixes (H, T, Q, or Y) is attached to the end of the input value. The numeric value input can be processed as an expression, therefore, the same operators used for the MAT command can be used.

Value over	: Numeric value input in the operand field has overflowed.
Illegal error	: Illegal character is written.
Balance error	: Parentheses or quotation marks are incorrectly used.
Stack error	: Expression is too complex.
Undefined error	: Input not defined by assembler.
Expression error	: Operator is written incorrectly.
Label error	: Mnemonic that cannot be analyzed is input.
Format error	: Incorrect input format is used.
Address error	: Address input exceeds FFFFH.
Org error	: Indicated when the address value specified by org is lower than the current address value.

```
*MEM D X ↵

 0000  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F

*ASM ↵

 ADDR   OBJECT          MNEMONIC
 0000   00              NOP                     = ↵
 0001   0102            LDAU    00002H          = ↵
 0003   03              DCX     SP              = ↵
 0004   040506          LXI     SP,00605H       = ↵
 0007   0708            ANI     A,00008H        =ANA V,A ↵
 0007   60 08 ↵                                                FORMAT
 0009   09              MOV     A,EAL           =ADD A,30 ↵     ERROR
 0009   00 00 00 00 ----F------- ---- Space
 0009   09              MOV     A,EAL   input   =ADD A,30H ↵
 0009   00 00 00 00 ----F------- ---- Space
 000D   0D              MOV     A,E     input   =XRA L,A ↵
 000D   60 17 ↵
 000F   0F              MOV     A,L             = ↵
 0010   FF              JR      00010H          = ↵
 0011   FF              JR      00011H          = ↵
 0012   FF              JR      00012H          =END ↵
```

When a NON-MAP memory area is accessed the following message is output:

```
*MAP W 0,FF ↵

*MAP K 100,1FF ↵

ASM FD ↵

 ADDR   OBJECT          MNEMONIC
 00FD   0B              JR      00109H          =MOV A,B ↵
 00FD   0A ↵
 00FE   4741            ONI     A,00041H        = ↵

 NON-MAP AREA ACCESS

*
```

3.17 DAS (Disassemble) Command

Format:

*DAS [␣partition]↵

Function: This command disassembles and displays the contents of the memory of the specified address range.

Main command : DAS
Subcommand : None
Operand : partition (disassembler range)
Option : None

3.17.1 Disassemble command description

When "DAS partition ↵" is input in the command input wait state ("*" is displayed), the memory contents of the range specified by partition are displayed. When a memory area that has not been mapped is accessed for disasembly, the message "NON-MAP AREA ACCESS" is output and the display returns to "*".
Execution of this command can be aborted by inputting CTRL-C or the ESC key.

3.18 MOV (Memory Transfer) Command

Format:

*MOV␣{I / U}␣partition addr↵

Function: This command performs transfer between the internal memory of the IE-7811H system and the memory of the user system. The memory contents of the address range specified by partition are transferred to the memory area whose start address is specified by addr.

Main command : MOV
Subcommand : I Transfers the contents of the memory of the user system to the internal mapped memory of the IE-7811H system.
: U Transfers the contents of the internal mapped memory of the IE-7811H system to the mapped memory of user system.
Operand : partition and addr
Option : None

3.18.1 MOV command description

The MOV command transfers the contents of the mapped memory of the IE-7811H system to the user system in 256-byte block units or transfers the contents of the mapped memory of the user system to the IE-7811H system. To execute the MOV command, the destination address needs to be mapped while the source address needs not.

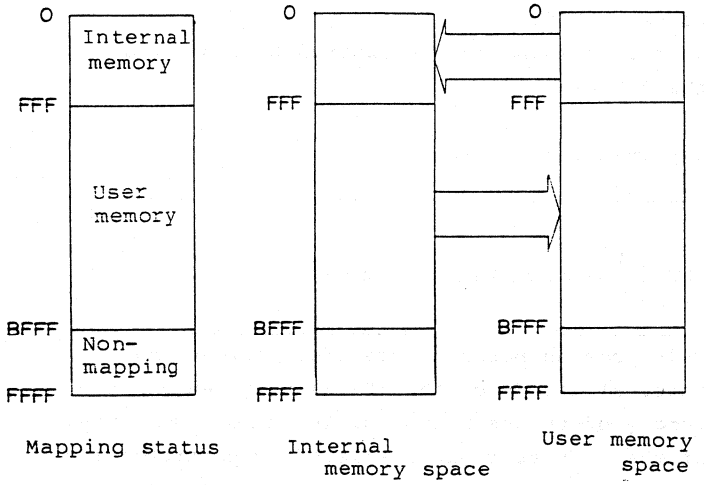


Fig. 3-1 Transfer of Memory Contents

The above diagram shows transfer of the data from the non-mapped memory to the mapped memory of the same address space.
Command execution can be aborted by inputting CTRL-C or the ESC key. However, the memory contents may be destroyed.

3.18.2 Transferring the contents of the memory from the IE-7811H system to the user system

MOV_I_partition_addr↵

This command transfers the contents of the user system memory whose address range is specified by partition to the IE-7811H system memory whose start address of the address range is specified by addr.

Examples:

The contents of the user system memory of addresses 0 to FFH are transfered to the IE-7811H system starting from address 0H.

```
*MAP W XXXX↵
*MAP U XX↵          ————————————Maps the user area
*MAP↵
$ IE-INTROM MAPPING $
$ IE-INTRAM MAPPING $
0100-FFFF
$ USER $
0000-00FF
$ NON MAPPING $

*MEM D XX↵

0000   00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F
0010   10 11 12 13 14 15 16 17   18 19 1A 1B 1C 1D 1E 1F
0020   20 21 22 23 24 25 26 27   28 29 2A 2B 2C 2D 2E 2F
0030   30 31 32 33 34 35 36 37   38 39 3A 3B 3C 3D 3E 3F
0040   40 41 42 43 44 45 46 47   48 49 4A 4B 4C 4D 4E 4F
0050   50 51 52 53 54 55 56 57   58 59 5A 5B 5C 5D 5E 5F
0060   60 61 62 63 64 65 66 67   68 69 6A 6B 6C 6D 6E 6F
0070   70 71 72 73 74 75 76 77   78 79 7A 7B 7C 7D 7E 7F
0080   80 81 82 83 84 85 86 87   88 89 8A 8B 8C 8D 8E 8F
0090   90 91 92 93 94 95 96 97   98 99 9A 9B 9C 9D 9E 9F
00A0   A0 A1 A2 A3 A4 A5 A6 A7   A8 A9 AA AB AC AD AE AF
00B0   B0 B1 B2 B3 B4 B5 B6 B7   B8 B9 BA BB BC BD BE BF
00C0   C0 C1 C2 C3 C4 C5 C6 C7   C8 C9 CA CB CC CD CE CF
00D0   D0 D1 D2 D3 D4 D5 D6 D7   D8 D9 DA DB DC DD DE DF
00E0   E0 E1 E2 E3 E4 E5 E6 E7   E8 E9 EA EB EC ED EE EF
00F0   F0 F1 F2 F3 F4 F5 F6 F7   F8 F9 FA FB FC FD FD FF
```

```
*MAP W XX ↵

*MEM D XX ↵

 0000   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 0010   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 0020   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 0030   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 0040   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 0050   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 0060   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 0070   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 0080   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 0090   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 00A0   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 00B0   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 00C0   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 00D0   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 00E0   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
 00F0   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00

*MOV I 0.FF 0 ↵

*MEM D XX ↵

 0000   00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F
 0010   10 11 12 13 14 15 16 17   18 19 1A 1B 1C 1D 1E 1F
 0020   20 21 22 23 24 25 26 27   28 29 2A 2B 2C 2D 2E 2F
 0030   30 31 32 33 34 35 36 37   38 39 3A 3B 3C 3D 3E 3F
 0040   40 41 42 43 44 45 46 47   48 49 4A 4B 4C 4D 4E 4F
 0050   50 51 52 53 54 55 56 57   58 59 5A 5B 5C 5D 5E 5F
 0060   60 61 62 63 64 65 66 67   68 69 6A 6B 6C 7D 6E 6F
 0070   70 71 72 73 74 75 76 77   78 79 7A 7B 7C 7D 7E 7F
 0080   80 81 82 83 84 85 86 87   88 89 8A 8B 8C 8D 8E 8F
 0090   90 91 92 93 94 95 96 97   98 99 9A 9B 9C 9D 9E 9F
 00A0   A0 A1 A2 A3 A4 A5 A6 A7   A8 A9 AA AB AC AD AE AF
 00B0   B0 B1 B2 B3 B4 B5 B6 B7   B8 B9 BA BB BC BD BE BF
 00C0   C0 C1 C2 C3 C4 C5 C6 C7   C8 C9 CA CB CC CD CE CF
 00D0   D0 D1 D2 D3 D4 D5 D6 D7   D8 D9 DA DB DC DD DE DF
 00E0   E0 E1 E2 E3 E4 E5 E6 E7   E8 E9 EA EB EC ED EE EF
 00F0   F0 F1 F2 F3 F4 F4 F6 F7   F8 F9 FA FB FC FD FE FF

*
```

3.18.3 Transferring the contents of the memory from the user system to the IE-7811H system

MOV_U_partition_addr↵

Execution of this command transfers the contents of the IE-7811H system memory whose adress range is specified by partition to the memory of the user system whose start address of the address range is specified by addr.

Examples:

```
*MAP W XXXX↵

*MEM F XXX 0,11,22,33,44,55↵

*MAP U XXX↵

*MEM F XX AA,BB,CC,DD,EE,FF↵

*MEM D X↵

0000   AA BB CC DD EE FF AA BB   CC DD EE FF AA BB CC DD

*MOV U XX 0↵

*MEM D XX↵

0000   00 11 22 33 44 55 00 11   22 33 44 55 00 11 22 33
0010   44 55 00 11 22 33 44 55   00 11 22 33 44 55 00 11
0020   22 33 44 55 00 11 22 33   44 55 00 11 22 33 44 55
0030   00 11 22 33 44 55 00 11   22 33 44 55 00 11 22 33
0040   44 55 00 11 22 33 44 55   00 11 22 33 44 55 00 11
0050   22 33 44 55 00 11 22 33   44 55 00 11 22 33 44 55
0060   00 11 22 33 44 55 00 11   22 33 44 55 00 11 22 33
0070   44 55 00 11 22 33 44 55   00 11 22 33 44 55 00 11
0080   22 33 44 55 00 11 22 33   44 55 00 11 22 33 44 55
0090   00 11 22 33 44 55 00 11   22 33 44 55 00 11 22 33
00A0   44 55 00 11 22 33 44 55   00 11 22 33 44 55 00 11
00B0   22 33 44 55 00 11 22 33   44 55 00 11 22 33 44 55
00C0   00 11 22 33 44 55 00 11   22 33 44 55 00 11 22 33
00D0   44 55 00 11 22 33 44 55   00 11 22 33 44 55 00 11
00E0   22 33 44 55 00 11 22 33   44 55 00 11 22 33 44 55
00F0   00 11 22 33 44 55 00 11   22 33 44 55 00 11 22 33

*MAP↵

$ IE-INTROM MAPPING $
$ IE-INTRAM MAPPING_$
1000-FFFF
$ USER $
0000-0FFF
$ NON MAPPING $

*
```

3.19 Self-diagnostic Command (DIG)

Format: *DIG↵

Main command: DIG
Subcommand : None
Operand : None
Option : None

The IE-7811H has a function that makes it possible for the IE-7811H to check its own hardware system. This command can be executed to diagnose the hardware system whenever the IE-7811H functions abnormally.
The self-diagnostic function checks these items:
(1) Internal memory
(2) MODE0, MODE1
(3) Port D, Port F
(4) Port A, Port B, Port C
(5) A/D input port (AN0 to AN7) (See Appendix for IE-7809 variance)
(6) Serial input
To perform the diagnosis, turn OFF the power supply of the IE-7811H and insert the 64-pin probe to the 64-pin socket on the driver module, then turn ON the power supply of the IE-7811H. The following message will be output.

[MESSAGE]

```
IE-7811H MONITOR VX.X[XX XXX XX]
   COPYRIGHT (C) 198X NEC CORPORATION

 CPU 7811H
 EXTERNAL MEMORY SIZE 64K BYTE
 RAE (E:ENABLE,D:DISABLE)=
```

Then set REA to D (disable). After the prompt "*" is output, input the DIG command to activate the self-diagnostic function. System reset is performed after the self-diagnostic operation is complete. The following is an example of this command's execution.

[PROGRAM]

```
*DIG↵

 ALTERNATE RAM TEST..OK
 USER MEMORY TEST....OK
 HARDWARE TEST
   MO.M1.......OK
   PORT D.F....OK
   PORT A.B.C..OK
   ANALOG IN...OK
   SERIAL I/O..OK

 DIAGNOSTIC COMPLETE
```

(See Appendix for IE-7809 variance)

3.20 PGM (PROM Programmer Control) Command

Format: *PGM↵

Function: This command enables the console used for the IE-7811H system to directly control the PG (PROM programmer) connected to CH2.

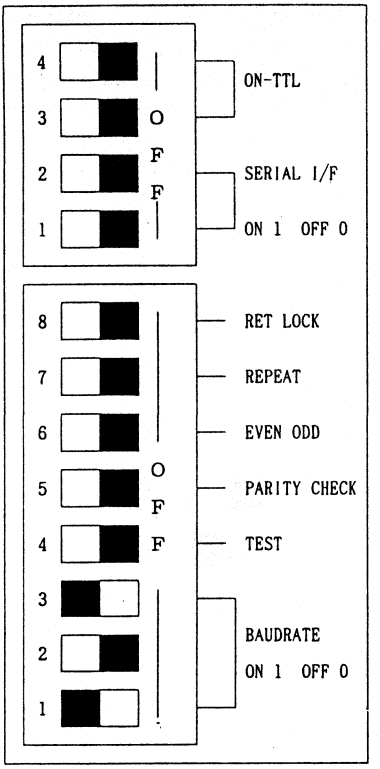
Main command: PGM
Subcommand : None

3.20.1 Connecting IE-7811H to PG-1000 or PG-2000

This section describes how to connect the IE-7811H to the PROM programmer (PG-1000 or PG-2000) and how to set the hardware.

(1) The baud rate of the IE-7811H, PG-1000, PG-2000, and terminal should be set to the same value. Refer to Chapter 4 for the baud rate setting of the IE-7811H. Refer to Tables 3-1 (for PG-1000) and 3-3 (for PG-2000) for the baud rate setting of the PROM programmer.

(2) When using the PG-1000, set the DIP switch located at the bottom of the PG-1000 according to Fig. 3-2. The baud rate should be set to the same setting as the IE controller module.



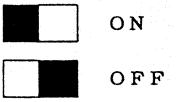


Fig. 3-2 Configuration of PG-1000 DIP Switch

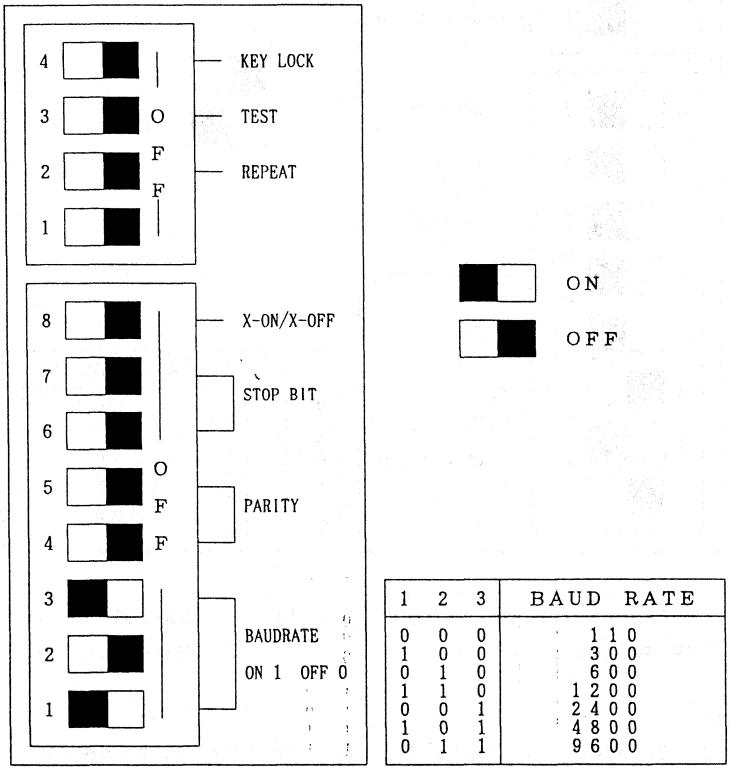
1	2	3	BAUD RATE
0	0	0	110
0	0	1	300
0	1	0	600
0	1	1	1200
1	0	0	2400
1	0	1	4800
1	1	0	9600
1	1	1	—

Table 3-1 PG-1000 Baud Rate Setting

Note:

1. 1, 2, and 3 in Table 3-1 indicate DIP switches 1, 2, and 3 of DIP switches 1 to 8.
2 In Fig. 3-2, 1 indicates ON, and 0 indicates OFF.

(3) When using the PG-2000, set the DIP switch located at the bottom of the PG-1000 according to Fig. 3-3. The baud rate should be set to the same setting as the IE controller module.



1	2	3	BAUD RATE
0	0	0	110
1	0	0	300
0	1	0	600
1	1	0	1200
0	0	1	2400
1	0	1	4800
0	1	1	9600

Fig. 3-3 Configuration of PG-2000 DIP Switch

Table 3-2 PG-2000 Baud Rate Setting

Note:

1. 1, 2, and 3 in Table 3-2 indicate DIP switches 1, 2, and 3 of DIP switches 1 to 8.
2 In Fig. 3-3, 1 indicates ON, and 0 indicates OFF.

(4) Setting of IE-7811H

(a) Change the jumper settings of JP3 according to Fig. 3-5.

(b) Set DIP switch No. 6 of DIP switch DP1 on the IE controller module to ON.

Note: Use hardware control mode for reader process on CH1 side.

(c) Use serial cable II (for CH2) that comes with the IE-7811H to connect the IE-7811H to the PG-1000 or PG-2000. Fig. 3-6 shows how to connect the IE-7811H using serial cable II (for IE-7811H CH2). If communications cannot be done, check according to Fig. 3-6.

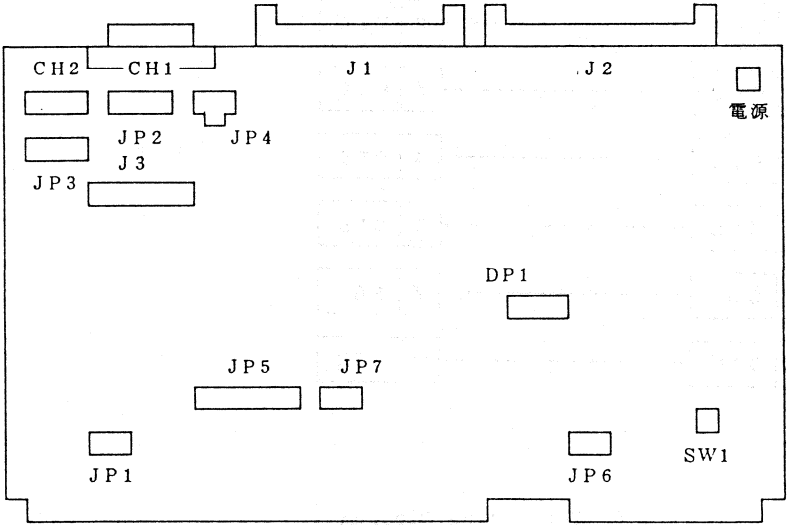


Fig. 3-4 IE-7811H Controller Module

Set JP3 as shown in Fig. 3-5.

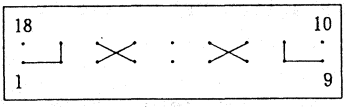


Fig. 3-5 JP3 Jumper Setting

The connecting diagram is shown in Fig. 3-6.

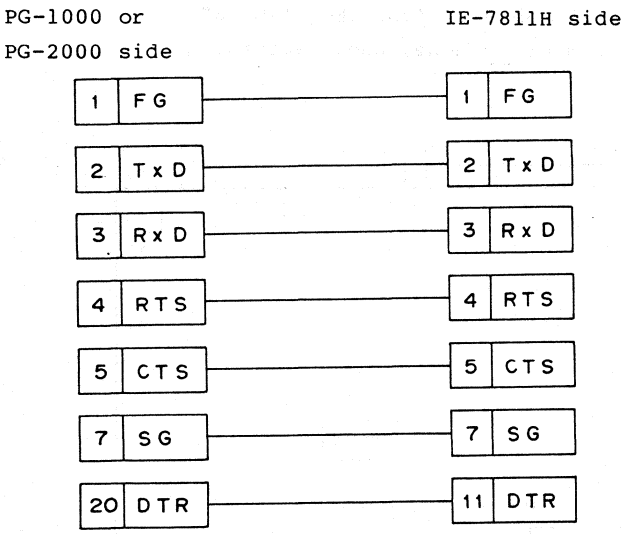


Fig. 3-6 Connecting Diagram

(5) Serial mode selection of PG

(a) Note that the operating procedures differ between the PG-1000 and PG-2000.

(b) Serial communication start method between the IE-7811H and PG-1000 or PG-2000.

1) When using PG-1000

RESET key → SERIAL B key

The IE-7811H and PG-1000 are connected by a serial communication line.

2) When using PG-2000

RESET key → -REM B key → START key

The IE-7811H and PG-2000 are connected by a serial communication line.

Key arrangements on PG-1000 and PG-2000 are shown in Figs. 3-7 and 3-8.

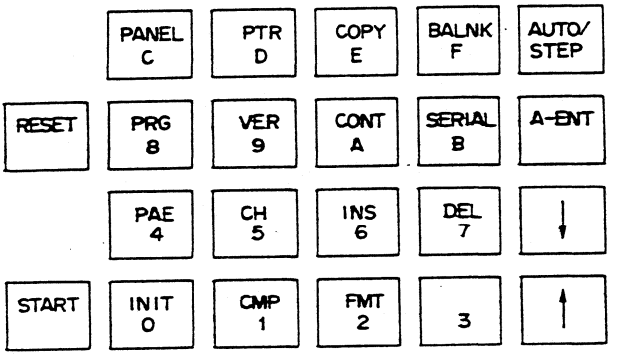


Fig. 3-7 Key Arrangement of PG-1000

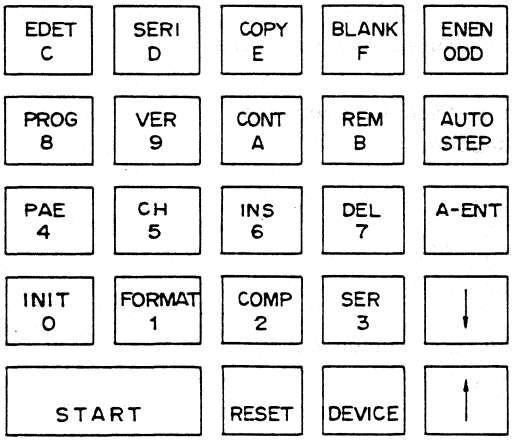


Fig. 3-8 Key Arrangement of PG-2000

3.20.2 Outline of PG-1000 and PG-2000

This section lists the ROMs that can be read and written by the PG-1000 and PG-2000.

For the PG-1000, the ROMs that can be read and written differ depending on the module used -- PG-1001, PG-1002, or PG-1003.

Refer to the PG-1001, PG-1002, PG-1003, and PG-2000 operating manuals.

(1) When using PG-1000

- When PG-1001 module is used
  μPD8741A, μPD8748, μPD8748H, μPD8749H, μPD8755A
- When PG-1002 module is used
  μPB403, μPB405, μPB406, μPB409, μPB417, μPB419, μPB423, μPB425, μPB426, μPB428, μPB429
- When PG-1003 module is used
  μPD78P09R

(2) When using PG-2000

μPD2716, μPD2732, μPD2732A, μPD2764, μPD27128, μPD27C64, μPD27256, μPD27C256

3.20.3 Starting and stopping PGM command

(1) When the PGM command is input, the "*" prompt is output from the PG.

```
*PGM ↵
PGM MODE
*
```

When the "*" prompt is not output, try the procedures described in (1) (for PG-1000) or (2) (for PG-2000) in 3.20.1 (5) (b). If the prompt is still not output, confirm the settings (1) to (4).

```
*PGM ↵
PGM MODE
■ ← Status in which the cursor is stopped and
    prompt is not output.
```

(2) The mode is transient (no echo back to console) when the PGM command is started. When "I ↵ " is input, the mode changes to intelligent and echo back is made to the console.

(3) Carry out the PG command (refer to 3.20.5 for execution).

(4) Input CTRL-Z (1AH) to terminate the PGM command execution. The normal IE command can then be used. The following is an example of the display when inputting CTRL-Z.

```
*←CTRL-Z input
PGM END
*
```

Note: The prompt is "*" for both IE and PG.

If CTRL-Z is input while the command is being input or executed, the PGM command is terminated. However, CTRL-Z cannot be used during execution of the L or P command (refer to 3.20.4 (2) L command, P command).

Note: When PG is connected, CTRL-P (CH1 and CH2 simultaneous output function) should not be input during normal manipulation of the IE command. If CTRL-P is input, data is output to the PG side, which is an abnormal operation the PG does not expect. CTRL-P mode is automatically released when the PGM command is executed, and remains released even after the PGM command is terminated.

The sequence from hardware setting to the termination of the PGM command execution is shown in the flowchart on the following page (Fig. 3-9).

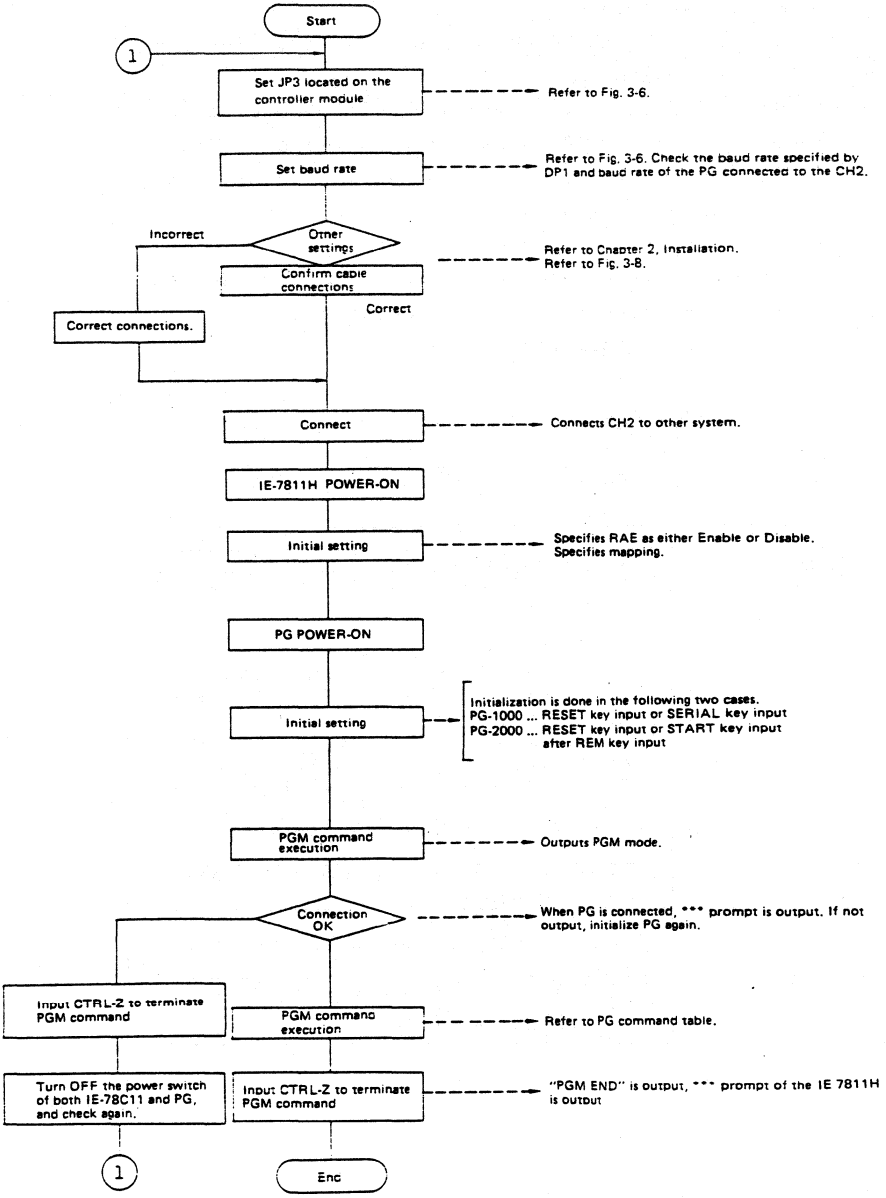


Fig. 3-9 PG Connection, Start-up, Termination Procedures

3.20.4 Details of PGM command

The PGM command enables the console used by the IE-7811H to control the commands of the PG.

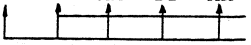
Additionally, by using this command, the contents of the mapped memory of the IE-7811H can be transferred to the PG, or the memory contents of the PG can be transferred to the mapped memory of the IE-7811H.

PG commands are listed in Table 3-3.

Commands other than the L and P commands are the same as commands of the PG console mode. For details on these commands, see the PG operation manual.

(1) PG-1000 and PG-2000 command table

Table 3-3 PG-1000 and PG-2000 Command Table

Command	Format	Function
A	*As, e, r↵	Sets parameters
E	*Er ↵	Changes the contents of the PG buffer. The format is: *Er r XX- YY XX- YY XX-  Input data Current data Data input format ° Input data ("?" is displayed when other than HEX coded data is input) ° Input space key (data will not change)
F	*Fr, re, d ↵	The buffer of the PG is initialized by d.
I	*I ↵	Changes to intelligent mode (echoes back to console).
J	*J ↵	Input from PTR "?" is displayed when a parity error occurs.
O	*Or, re ↵	Displays the contents of the PG buffer. Displaying format: r 00 00 00 00 00 00 00

R	*Rr, re ↵	Transfers the contents of the PROM to the PG buffer.
S	*S ↵	Selects PROM.
T	*T ↵	Changes to transient mode (no echo back to console).
V	*Vs, e, r ↵	Compares the contents of the PROM and PG buffer. "?" is displayed when data differ.
W	*Ws, e, r ↵	Writes the contents of the PG buffer to the PROM. "?" is displayed when a write error occurs.
Y	*Y ↵	Parameters currently set can be displayed and set. Start and end addresses of the PROM, and the start address of the PG buffer can be set and displayed.
Z	*Z ↵	Checks whether data is written to the PROM. Nothing is displayed when no data is written. "?" is displayed if data is written.
L	*Lbias ↵ PARTITION= i,ie ↵	Transfers the contents of address i to ie of mapped memory of the IE-781lH to the address i+bias to ie+bias of the PG buffer.
P	*Pr, re ↵ bias=i ↵	Transfers the contents of address r to re of the PG buffer to address r+i to re+i of the mapped memory of the IE-781lH.

Note: s : PROM start address

r : PG buffer start address

re: PG buffer end address

i : IE-781lH start address

ie: IE-781lH end address

d : Data

Error condition

r > re

i > ie

Input other than hexadecimal

(2) Details of L command and P command

(a) L command

```
*LXXXX ↵                : XXXX...  Load bias of PG
PARTITION=YYYY,ZZZZ       YYYY...  Transfer start
*                                  address of the
                                   memory of the
                                   IE-7811H.
                          ZZZZ...  Transfer end
                                   address of the
                                   memory of the
                                   IE-7811H.
                                   However, XXXX,
                                   YYYY, and ZZZZ
                                   are HEX and the
                                   lower 4 digits
                                   are effective.
```

This command transfers the contents of the mapped memory (from YYYY to ZZZZ) of the IE-7811H to the buffer ((XXXX + YYYY) to (XXXX + ZZZZ)) of the PG. Input ESC key to abort the execution of this command. When the ESC key is input, PG command input wait state is assumed. PARTITION input cannot be omitted. It takes about 30 seconds to transfer 4K bytes of data from the IE-7811H memory to the PG buffer.

Examples:

(1) Normal use of L command

```
*LO ↵                  : Sets the PG memory bias to 0, and
PARTITION=0,FF ↵         loads the contents of the IE memory
                         address 0 to FFH to the PG.
*
```

(2) When an error is input

```
*LO ↵
PARTITION=0,FX ↵       : PARTITION input error
INPUT DATA ERROR
*
```

```
*LO ↵
PARTITION=0,FF ↵    : An error occurs because the specified
                      IE
NON MAP AREA ACCESS   memory has not been mapped.

*LO ↵
PARTITION=0,FF ↵    : Data cannot be transferred correctly
?                     during data transfer from IE to PG
*                     (such as parity error).

*L8000 ↵            : When data to be transferred exceeds
?                     the PG memory address range, or
*                     data exceeds the memory address
                      range during transfer.
```

PG command input state "*" is assumed after an error is displayed.

(b) P command

```
*PXXXX, YYYY ↵          : XXXX... Transfer start address of the
BIAS=ZZZZ ↵                       PG buffer.
                          YYYY... Transfer end address of the
*                                 PG buffer.
                          ZZZZ... Load bias of the IE-7811H
                                  XXXX, YYYY, and ZZZZ are HEX
                                  and the lower 4 digits are
                                  effective.
```

Transfers the memory contents (XXXX to YYYY) of the PG to the mapped memory ((XXXX + ZZZZ) to (YYYY + ZZZZ)) of the IE-7811H. "*" is displayed when the transfer is terminated normally. Input ESC key to abort this command execution. When ESC key is input, PG command input wait state is assumed. BIAS input cannot be omitted. It takes about 30 seconds to transfer 4K bytes of data from the memory of the IE-7811H to the PG buffer.

Examples:

(1) Normal use of P command

```
*P0, FF ↵          : Transfers the contents of address 0 to
BIAS=0 ↵             FFH of PG to the memory of the IE-7811H
                     with bias 0.
*
```

(2) Display of error

```
*PO,EF ↵           : BIAS input error.
BIAS=X ↵
INPUT DATA ERROR
*

*PO,EF ↵           : An error occurs because the memory of
BIAS=100 ↵           the IE-7811H has not been mapped.
NON-MAP AREA ACCESS
*

*PO,1EF ↵          : Check sum error occurred during transfer.
BIAS=25 ↵
CHECK SUM ERROR
*

*P35, FF ↵         : Characters other than HEX code number
BIAS=0 ↵             were transferred.
BAD CHARACTER
*

*PO,1FF ↵          : Data that exceed the memory area of the
BIAS=FF20 ↵          IE-7811H are to be transferred.
ADDRESS OVER
*

*PO,8000 ↵         : Address that exceeds the PG memory
?                    address is to be set.
*
```

After an error is displayed, PG command input wait state is assumed and "*" is displayed.

3.20.5 PGM command execution example

The following explains the PGM command execution examples:

(1) When the power of the IE-7811H is turned ON, this message appears:

```
IE-7811H MONITOR V2,0 [XX XXX XX]
   COPYRIGHT (C) 198X NEC CORPORATION

CPU 7810H
EXTERNAL MEMORY SIZE 64K BYTE
RAE (E:ENABLE,D:DISABLE)= D

USER-SYSTEM VDD-ON, VCC-ON
*
```

(2) After "*" indicating the command input wait state is output, specify mapping to clear the memory.

```
*MAP W XXXX ↵        : Maps address 0 to FFFFH as
                       IE-7811H internal RAM.
*MEM F XXXX 0 ↵      : Clears address 0 to FFFFH of
                       the IE-7811H with 0.
```

(3) Executes the PGM command. After the execution, transfers the contents of the ROM to the PB buffer by the R command.

```
*PGM ↵          Note: After the PGM mode is set, "*"
 PGM MODE              is the PG prompt.

*                    : Input "I ↵" (refer to Table
                       3-1 I command).
*FO,1FFF,0 ↵         : Clears PG memory.
*R ↵                 : Transfers the contents of the
                       ROM for the amount of addresses
                       specified by the Y command to
                       the PG memory.
```

(4) Transfer the contents read out from the ROM to the IE memory, and terminate the PGM command execution.

```
*00.3F↵          : Displays the contents read out from the ROM
0000   69 00 6A 01 6B 02 6C 03   6D 04 6E 05 6F 06 68 07
0010   11 10 69 10 6A 11 6B 12   6C 13 6D 14 6E 15 6F 16
0020   68 17 54 22 00 00 00 00   00 00 00 00 00 00 00 00
0030   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
*P0.27↵
BIAS =0↵
                    : Transfers the contents of the PGM
*                     memory to the IE-7811H memory.
PGM END             : Input of CTRL-Z to terminate the PGM
*                     command execution.
                    : Prompt "*" after "PGM END" is for the
                      IE-7811H monitor program.
```

(5) After the execution of the PGM command, display and confirm the contents of the memory to which data is transferred.

```
*MEM D 0.3F↵        : Displays the memory contents
  0000   69 00 6A 01 6B 02 6C 03   6D 04 6E 05 6F 06 68 07
  0010   11 10 69 10 6A 11 6B 12   6C 13 6D 14 6E 15 6F 16
  0020   68 17 54 22 00 00 00 00   00 00 00 00 00 00 00 00
  0030   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00

*
```

(6) Change the memory contents using the assemble command.

```
*ASM 12↵

 ADRS   OBJECT          MNEMONIC
 0012   6910            MVI     A,00010H      =MVI A,70↵
 0012   6970↵
 0014   6A11            MVI     B,00011H      =MVI B,71↵
 0014   6A71↵
 0016   6B12            MVI     C,00012H      =MVI C,72↵
 0016   6B72↵
 0018   6C13            MVI     D,00013H      =MVI D,73↵
 0018   6C73↵
 001A   6D14            MVI     E,00014H      =MVI E,74↵
 001A   6D74↵
 001C   6E15            MVI     H,00015H      =MVI H,75↵
 001C   6E75↵
 001E   6F16            MVI     L,00016H      =MVI L,76↵
 001E   6F76↵
 0020   6817            MVI     V,00017H      =END↵
```

(7) Transfer the changed memory contents to the PG memory.

```
*PGM ↵
PGM MODE
*LO ↵                ---- Transfers the contents of the IE-7811H
PARTITION =0.2F           memory to the PG memory.
*
```

(8) Incorrect part is corrected and written to the PROM to complete the operation.

```
*00,3F ↵                  : When displayed, address 21 is not corrected.
0000   69 00 6A 01 6B 02 6C 03   6D 04 6E 05 6F 06 68 07
0010   11 10 69 70 6A 71 6B 72   6C 73 6D 74 6E 75 6F 76
0020   68 17 54 22 00 00 00 00   00 00 00 00 00 00 00 00
0030   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
*E20 ↵
0020   68-68 17-17 54-
*W ↵                      : Changes the contents of
*V ↵                        address 21 to 77H.

*                         : Writes the data for the
PGM END                     amount of addresses specified
*                           by the command to the PROM
                            and verifies.
                          : Input CTRL-Z to terminate the
                            PGM command.
                          : Note that the prompt "*" after
                            "PGM END" is the IE-7811H
                            monitor prompt.
```

Command Table I-1

Main command	Sub-command	Manipulation	Function
Mapping (MAP)	Def	MAP ↵	Displays mapping
	W	MAP W partition ↵	Maps the area specified by partition as IE-7811H internal RAM
	R	MAP R partition ↵	Maps the area specified by pattition as IE internal ROM
	U	MAP U partition ↵	Maps the area specified by partition to the user system
	K	MAP K partition ↵	Clears the mapped area specified by partition
Memory (MEM)	C	MEM C addr ↵	Changes the contents of the address specified by addr
	D	MEM D partition ↵	Displays the contents of the area specified by partition
	E	MEM E partition ↵	Tests the memory of the area specified by partition
	F	MEM F partition string ↵	Initalizes the contents of the area specified by partition by string
	G	MEM G partition string ↵	Displays the start address at which the contents of the area specified by parttion and string coincide
	M	MEM M partition addr ↵	Transfers the contents of the area specified by partition to the address whose starting address is specified by addr

Command Table I-2

Main command	Sub-command	Manipulation	Function
Memory (MEM) cont'd	V	MEM V partition addr ↵	Compares the contents of the area specified by partition to the contents of the address whose starting address is specified by addr
	X	MEM X partition addr ↵	Exchanges the contents of the area specified by partition with the contents of the address whose starting address is specified by addr
Register (REG)	C	REG C register-name ↵	Changes the contents of the specified register
	D	REG D register-name ↵	Displays the contents of the specified register
	Def	REG ↵	Displays all register
Mode register (MDR)	C	MDR C mode-register-name1	Changes the contents of the specified mode register
	D	MDR D mode-register-name2	Displays the contents of the specified mode register
	Def	MDR ↵	Displays all mode register
Special register (SPR)	C	SPR C special-register-name1 ↵	Changes the contents of the specified special register
	D	SPR D special-register-name2 ↵	Displays the contents of the specified special register
	Def	SPR ↵	Displays all special registers

Note: Def is the abbreviation for default, used only when the main command is input.

Command Table II-1

Main command	Sub-command	Manipulation	Function
Load (LOD)	-	LOD TTY?$addr ↵	Loads the data from TTY? to the program memory with bias addr
Save (SAV)	-	SAV TTY? partition ↵	Saves the contents of the area specified by partition to TTY?.
Emulation (RUN)	N	RUN N addr ↵	Executes the program from the address specified by addr
	B	RUN B addr ↵	Executes the program from the address specified by addr and breaks accroding to the break condition
	S	RUN S addr,step ↵	Executes the program from the address specified by addr for the number of times specified by step
	T	RUN T condl $as $R ↵	Executes the program, and displays at each step (condl=addr,step or addr,register-name ... value) (... is comparison operator)
	Def	RUN ↵	Executes the program from the current address
Physical break (BR?)	A	BRA A=addrs V=values C=cond L=loop ↵	Sets break condition: address, data, condition, and number of loops
	D	BRD data ↵	Sets option break data
	E	BRE count ↵	External break or step
	T	BRT count ↵	Sets timer data
	Def	BRA ↵	Displays data

Command Table II-2

Main command	Sub-command	Manipulation	Function
Logical break (BR?)	0 3	BRO BRA,... ↵	Combined data of physical and logical register
	Def	BR? (?=0~3)	Displays data
Break mode (BRM)	-	BRM BRA,BRO,... ↵	Hardware setting of physical/logical break data
	Def	BRM ↵	Displays and sets to hardware the data set to BRM
Trace (TR?)	C	TRC a ↵ (a=I or M)	Specifies trace data display cycle
	D	TRD b ↵ (b=No. of traces, ALL)	Trace dump
	M	TRM c ↵ (c-NON,ALL,TRX)	Sets trace mode
	P	TRP d ↵ (d=No. of traces, O, N)	Moves trace pointer
	X	TRX A=addrs V=values C=cond PA=values PB=values ↵	Sets trace condition; address, data, condition, port
	S	TRS e ↵ (e=PB,EX)	Selects source
	Def	TR? ↵	Displays data (except TRP)

Command Table III-1

Main command	Sub-command	Manipulation	Function
Clock (CLK)	I	CLK I ↵	Specifies internal clock
	U	CLK U ↵	Specifies external clock
	Def	CLK ↵	Displays current clock
Reset (RES)	H	RES H ↵	Resets IE-781lA
	Def	RES ↵	Resets emulation CPU
Operation (MAT)	-	MAT expression ↵	Operation command
Suffix (SUF)	H	SUF H ↵	Treats the input numeric value as HEX.
	T	SUF T ↵	Treats the input numeric value as DEC.
	Q	SUF Q ↵	Treats the input numeric value as OCT.
	Y	SUF Y ↵	Treats the input numeric value as BIN.
	Def	SUF ↵	Displays current suffix
Assembler (ASM)	-	ASM addr ↵	Performs on-line assembling from the program memory address specified by addr
Disassembler (DAS)	-	DAS partition ↵	Disassembles the contents of the area specified by partition
Memory transfer (MOV)	I	MOV U partition addr ↵	Transfers the contents in the address range of the internal mapped memory specified by partition to the user memory and set the mapped memory to the user memory space.

Command Table III-2

Main command	Sub-command	Manipulation	Function
Memory transfer (MOV)	U	MOV I partition addr ↵	Transfers the contents in the address range of the user memory specified by partition to the intenral memory and set the mapped memory to the internal memory space.
Self-diagnostic (DIG)	-	DIG ↵	Self-diagnoses own hardware
PROM programmer control (PGM)	-	PGM ↵	Enables the IE-7811A to control the PROM programmer

ERROR Messages

Error Message	Description
COMMAND FORMAT ERROR	Indicates an error in the command format
MEMORY FAILURE	Program memory is abnormal
CHECK SUM ERROR	Indicates incorrect data is input when loading program
INPUT DATA ERROR	Incorrect data input
NON-MAP AREA ACCESS	The area that has not been mapped is accessed
MAPPING ERROR	Mapping address is incorrectly specified
SYSTEM DOWN	System is abnormal. EVACHIP is not functioning correctly
COMMAND TOO LONG	Indicates the input number of character exceeds the maximum value (Output when 124 or more characters are input.)
NON BREAK	This is output when break does not occur when even a forced break is attempted, and indicates the firmware is not functioning correctly
E-CPU RUN	This is output when CTRL-C is input during the user program execution and status becomes command input wait state. The emulation CPU is executing the user program, therefore, to input a command again, input forced break or reset command.
NON TRACE DATA	This is displayed when there is no trace data.
POINTER OVER SET	This is displayed when a number exceeding the number of trace data is input by the TRP command
ADDRESS OVER	This is displayed when the data transferred by the LOD or PGM command exceeds IE memory address 0FFFH

CHAPTER 4 SERIAL INTERFACE

4.1 Outline

The serial interfaces used in the IE-7811H system conforms to the RS-232C standard. Because the handshaking and code protocol of the RS-232C differs slightly from product to product, the IE-7811H is designed so that by the combined setting of the DIP switches, jumpers (JP2 and JP3), and wiring of the cables, interfacing is possible with almost any commercially available terminal (TTY, PTP, PTR, etc.).

This will now be explained and interfacing examples with three terminals conforming to the RS-232C standard are presented.

4.2 Basic Specifications

1) Serial controller	μPD7201 multiprotocol serial controller
2) Number of channels	Two: CH1 .... For terminal CH2 .... For subinterface to I/O
3) Baud rate	110 to 9600bps (9600, 4800, 2400, 1200, 600, 300, 110)
4) Communication	Asynchronous
5) Transmission format	Start bits : 1 bit Character length : 8 bits Parity : No parity Stop bits : 2 bits
6) Character code	8-bit ASCII (MSB masked by software)
7) Handshaking protocol	Code control and hardware control by CTS and RTS

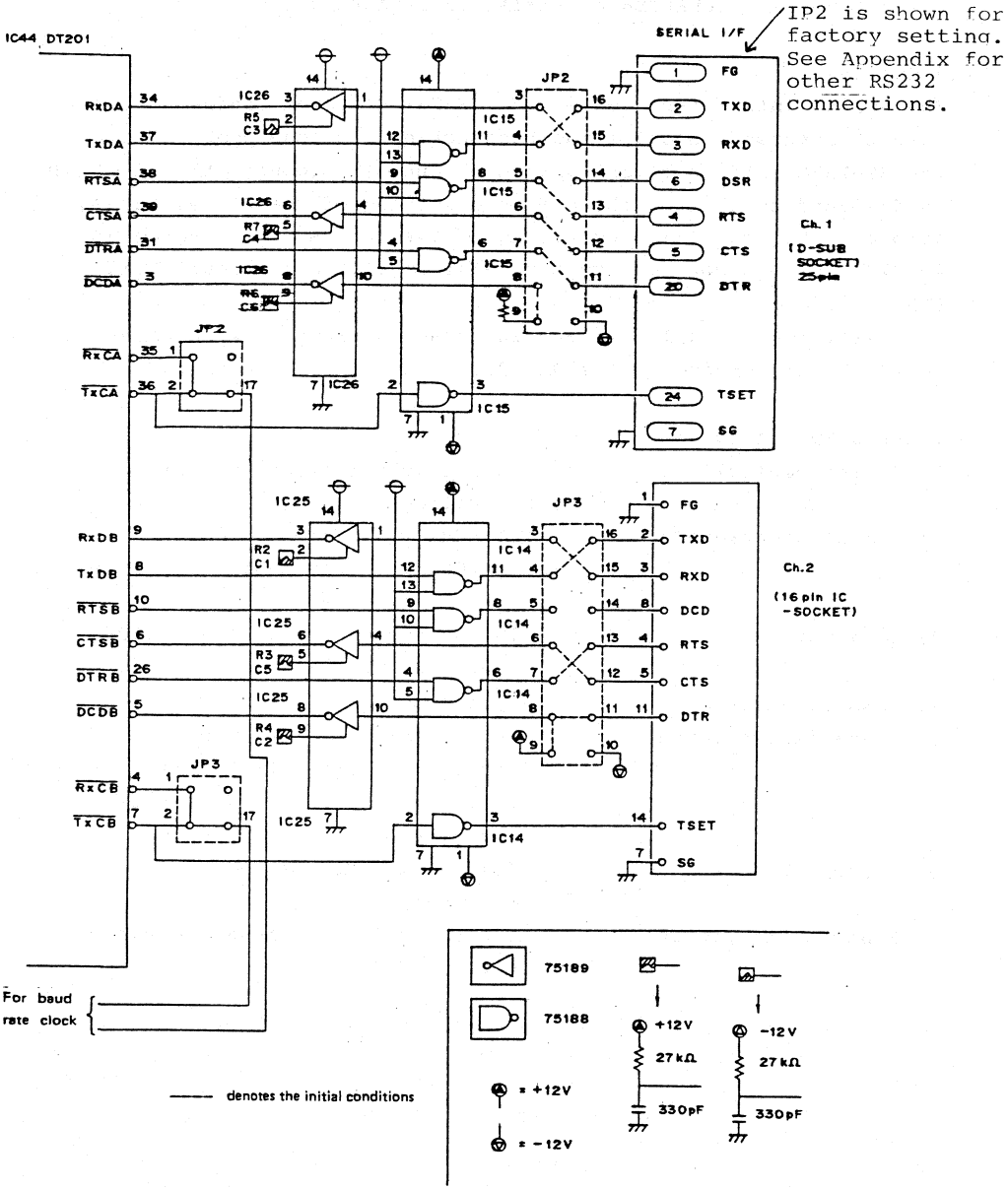


Fig. 4-1 Serial Circuit

4.3 RS-232C Pin Description

Table 4-1

Pin No. (D sub-connector	Signal name	Output/ input*	Description	
2	TxD (SD)	O	Serial output pin	The significance of these signals can be reversed by changing the JP2 jumper setting.
3	TxD (RD)	I	Serial input pin	
5	CTS	I	Input pin for the terminal receive enable output. High level indicates ready.	The significance of these signals can be reversed by changing the JP2 jumper setting
20	DTR	O	Output pin to indicate that the controller is ready to receive. High level indicates ready.	
4	RTS	O	Output pin for the transmission request signals (motor ON, etc.) to the reader of the terminal. High level indicates request output. However, if reader control which (described later) is performed by codes, this is always low.	
8	DCD	I	Input pin used to confirm that the terminal is ready for transmission or reception. The IE-78C11 does not operate unless this signal is high level. If the terminal outputs this signal, there is no harm in connecting it. It can be pulled up inside the IE-78C11 if desired. This signal is pulled up to high level as the factory-set condition, and not output to the channel pin.	
1	FG	-	Frame ground (connected to 0V power supply of board).	
7	SG	-	Signal ground (connected to 0V power supply of board).	

Note: "O" indicates output and "I" indicates input.

4.4 Setting of Serial Protocol

Serial interface handshaking is specified by the setting of pins 4 to 6 of DP1 (refer to Figs. 2-2 and 2-6).

Table 4-2

No.	ON	OFF
4	Specification for CH1. Handshaking is performed by hardware control with CTS and DTR, and is also performed by transferring X-ON (11H) and X-OFF (13H) codes. If handshaking is to be performed only by transferring X-ON and X-OFF codes, CTS must be pulled up inside the IE-7811H and DTR should not be connected.	Specification for CH1. Handshaking is performed only by CTS and DTR.
5	Specification for CH2. Functions in the same way as Pin No. 4.	Specification for CH2. Functions in the same way as Pin No. 4
6	Transmits request to the terminal assigned as the reader or punch of CH1 or CH2 by turning ON/OFF the RTS signal.	Transmit request performed by X-ON and X-OFF codes.

Note: When handshaking for the channel assigned as the reader or punch is performed by X-ON and X-OFF, always set DIP6 to ON and use a console that can handle code control.

4.5 Handshaking I/O Standards

(1) Output standards

When CTS input is OFF	Transmission ends with the code being currently output.
When X-OFF (13H) is output	From the point at which X-OFF code is received, outputs for a maximum 4 bytes including the code being currently output.

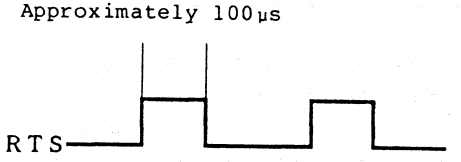
(2) Input standards

When DTR output is off	Input is enabled within a maximum 4 bytes after the DTR output goes off.
When X-OFF (13H) is output	Input is enabled within a maximum 10 bytes after X-OFF is output.

(3) Reader standards

Hardware handshaking

The reader reads characters in byte units as controlled by the ON-OFF state of the RTS signal.



X-ON, X-OFF reader control

When X-ON (11H) is output

Input is enabled within 4 bytes of data immediately after X-ON is output.

When X-OFF (13H) is output

Input is enabled within 4 bytes of data after X-OFF is output.

Normally, when handshaking is performed by control codes, there is a danger that 2 or 3 bytes of blank data will be sent. The IE-7811H has adequate allowance for input from the console to compensate for this. However, when data are sent from the IE-7811H there is a chance that a maximum of 4 bytes of blank data might be sent. Therefore, when using control codes to perform handshaking, always use a console that has a buffer function to take care of this problem. Most TTY specifications and CRTs designed for commercial use are provided with this kind of feature.

4.6 Baud Rate Specification

Baud rate is specified by the setting of switches 1 to 3 of DP1 (refer to Figs. 2-2 and 2-6).

Table 4-3 DP1

Switch number			Description
1	2	3	
ON	ON	ON	When connected to MD-086.
ON	ON	OFF	110bps
ON	OFF	ON	300bps
ON	OFF	OFF	600bps
OFF	ON	ON	1200bps
OFF	ON	OFF	2400bps
OFF	OFF	ON	4800bps
OFF	OFF	OFF	9600bps

Note: Baud rates cannot be set independently for CH1 and CH2. When different devices are connected to CH1 and CH2, therefore, both devices must operate at the same baud rate. If a clock frequency 16 times the desired baud rate can be sent from the terminal, by changing the connections of JP2 and JP3 (refer to Fig. 2-4) to 1-2-18 from 1-2-17, that terminal can be used.

4.7 Setting of serial interface

4.7.1 Initial setting

The serial interface is factory-set to the following operational conditions:

Mode	CH1	modem mode
	CH2	terminal mode
Baud rate	CH1, CH2	4,800 bps
Handshake protocol	CH1, CH2	hardware control (using RTS and CTS lines)

With the setting above, verious equipments can be connected to each channel by the cables supplied with the IE-78C11.

(1) Channel 1

NEC terminal MD-910TM
NEC host computer MD-080 series (The serial interface of the MD should be set to terminal mode.)
NEC host computer PDA-880 (NPM-951, the general purpose serial port, is required as an option.)

(2) Channel 2

NEC PROM writer PG series
NEC host computer MD-080 series
NEC host computer PDA-880

Detailed information on connection with the host computers is provided in the IE-7811H system software operation manual.

4.8 Interfacing With Other Models

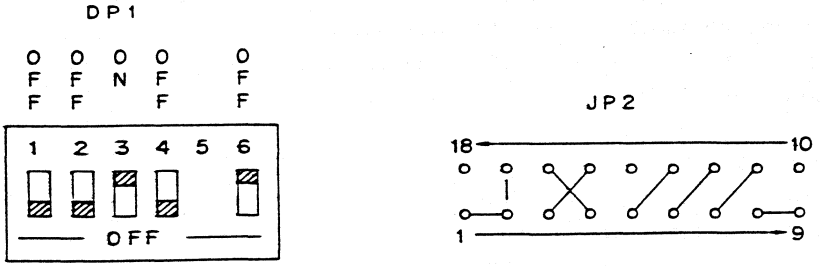
Interfacing to the following three commercially available models conforming to the RS-232C standard are described here.

(1) Citizen protyper MODEL 7652
(2) Anritsu character display terminal DDY86, DDY-870
(3) Casio Typuter MODEL650

Settings of DIP switch (DP1) and jumper JP2 on the IE-781lH's IE controller module and RS-232C cable connection are described here.

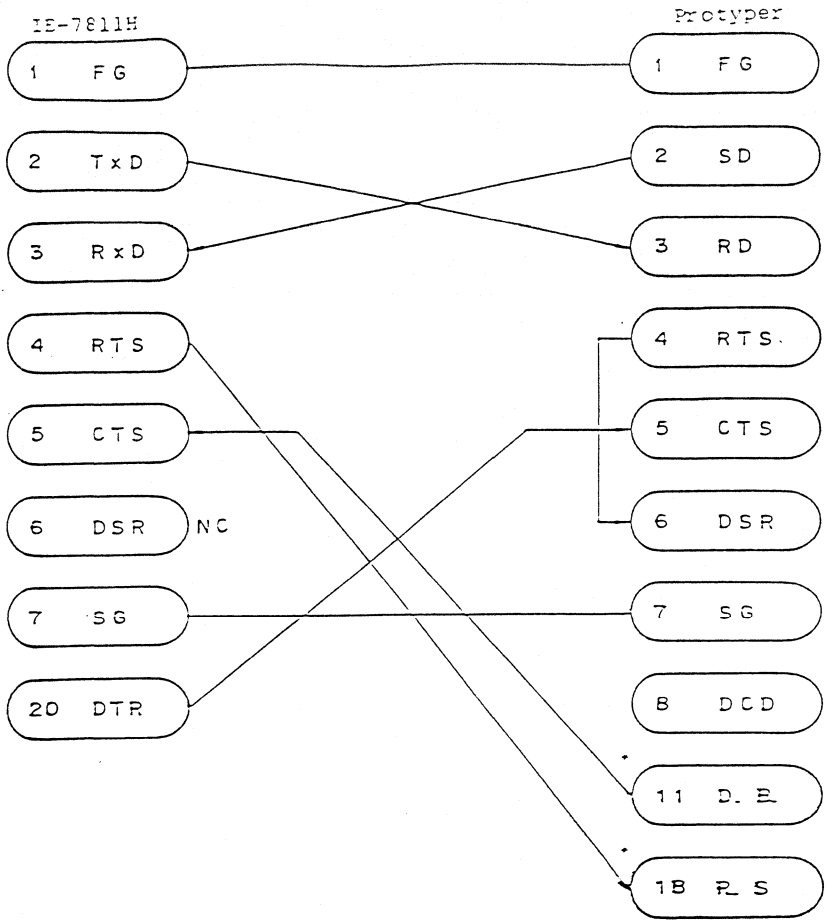
4.8.1 Interfacing with Citizen Protyper Model 7652

(1) When all handshaking is performed by hardware
(Settings for when handshaking is performed by CTS and DTR, and RTS is used for the punch and reader.)



1. Switch No. 5 of DP1 is undefined (specification for CH2).
2. In this example, the baud rate is 4800bps.

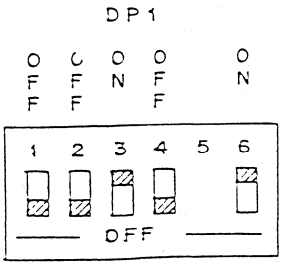
Fig. 4-3 IE-781lH Setting



* Used only by the Protyper

Fig. 4-4 Cable Connection

(2) When reader control is performed by codes



Switch No. 5 of DP1 is undefined (specification for CH2)

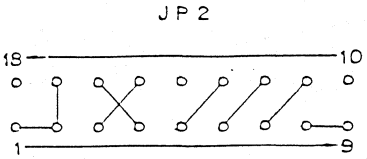
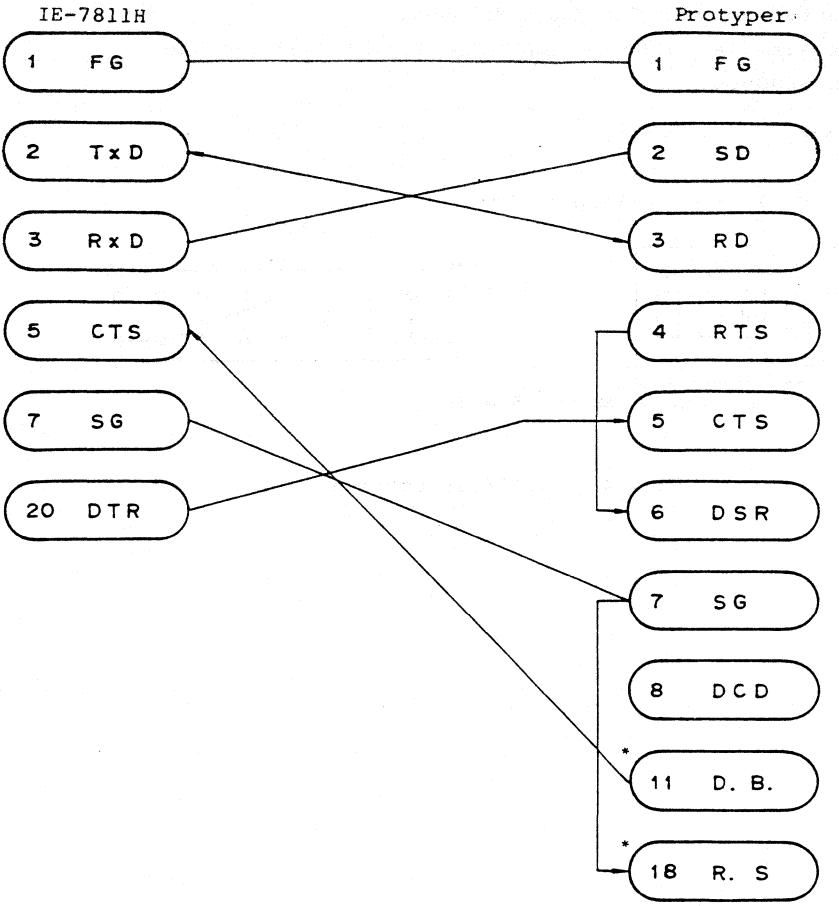


Fig. 4-5 IE-7811H setting

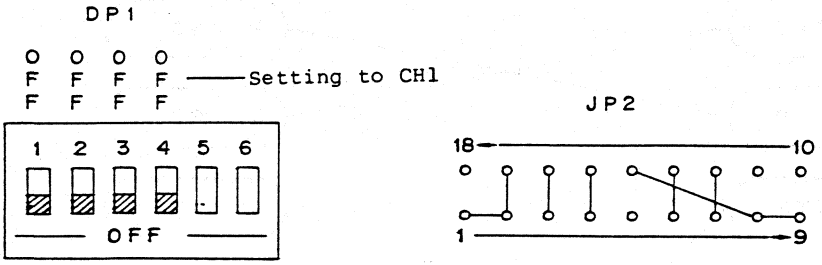


* Used only by the Protyper

Fig. 4-6 Cable Connection

4.8.2 Interfacing with Anritsu DDY86 Series

In this example, DDY86 is connected to CH1, and the baud rate is 9600bps.



Set switches 5 and 6 as necessary.

Fig. 4-7 IE-7811H Setting

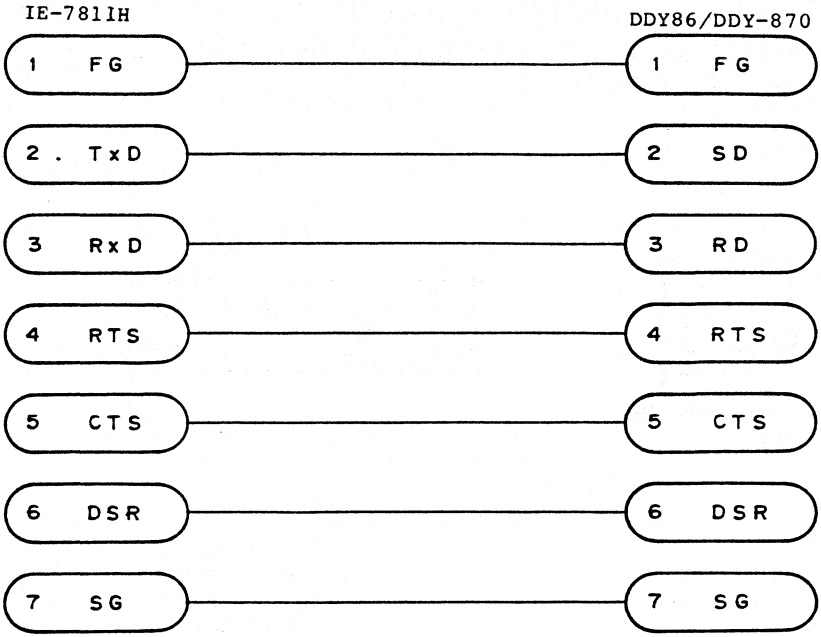


Fig. 4-8 Cable Connection

4.8.3 Interfacing with Casio Typuter Model 650

In this example, Model 650 is connected to CH1, and the baud rate is 4800bps.

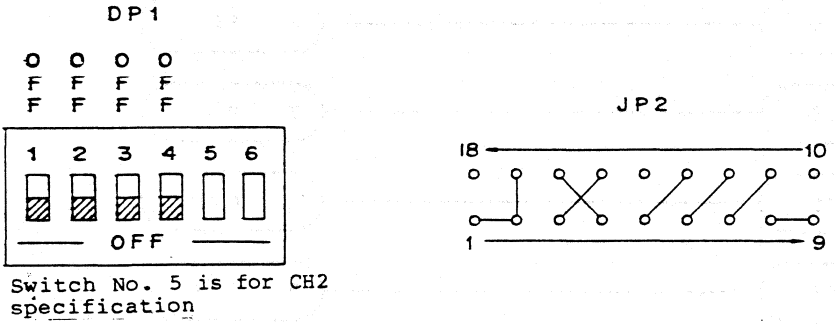


Fig. 4-9 IE-7811H Setting

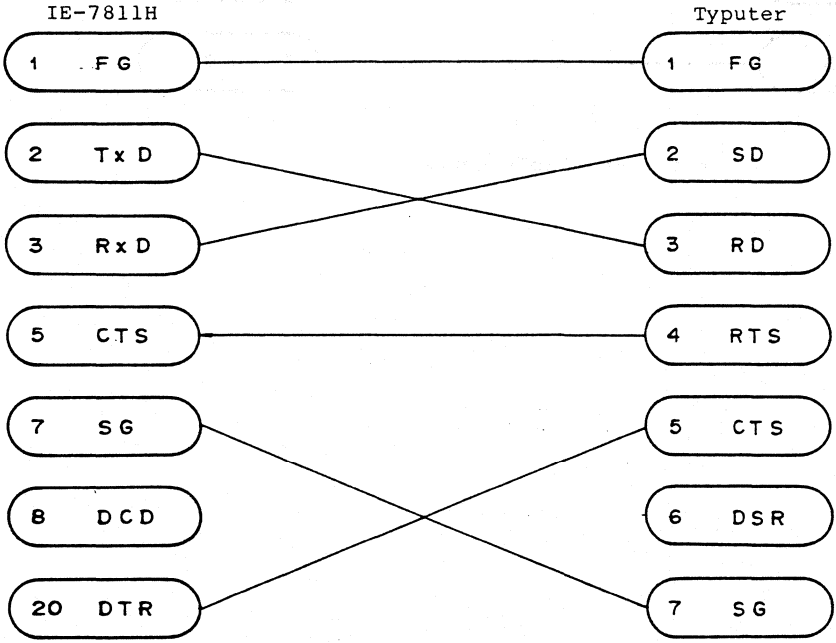


Fig. 4-10 Cable Connection

A P P E N D I X

A 1.1 Mechanical/Electrical Variation

The IE-7809/IE-87AD-M stand alone has the same physical appearance as IE-7811-H; except for the switch on target probe.
The IE-78C11, however, has some minor differences:
- (i) leds to indicate whether the emulator CPU is in H/W stop, S/W stop, Halt or in latch up condition (ii), test points for Ov, power supply control, break trigger in and break trigger out.
(iii) The diagnostic plug in driver housing is covered by a flap.
(iv) The power supply cable between controller card and driver housing has 6 pole socket.

N.B. Obviously the monitor firmware on controller card (2 x uPd2716 + 1 x uPD2764) will be for appropriate IE system.

A 1.2 CPU clock rate & CPU in emulator

The maximum clock rate for IE-7809/IE-78C11 is 12 Hz.
In the IE-7809 the emulator processor is uPD7807 and in the IE-78C11 the emulator processor is uPD78C10.

A 1.3 IE-78C11 Test points

TP1 Break trigger out. Output control from IE-78C11 active (L); occurs on break condition of emulator

TP2 Break trigger in. Input from external H/W, active (L) level. This will cause a break within emulator when active (L).

TP3 Power supply control. Output signal active (L) occurs when emulator enters latch up condition. Thus, external current may switch off power supplies when latch up detected.

TP4 Ground internally connected to Ov of emulator.

A 1.4 IE-7809 Driver & Control Block Diagrams

(*take from IE-7809 manual)

A 1.5 IE-78C11 Target Probe

To use with diagnostic plug in driver housing, the Quic plug must be removed from target probe.

A 1.6 IE-7809 / IE-87AD Clock Switch

On the target probe for IE-7809 / IE-87AD a slide switch selects between internal (INT) or external (EXT) clock. the external clock can either be a quartz oscillator or an external clock on the target hardware.

A 1.7 IE-7809 Mode Register & Special Register

The IE-7809 has WDM (watchdog timer mode register) instead of ANM (A/D mode register) and has M-register (mode register for comparator). The special register, type 2 for IE-7809 are PA, PB, PC, PD, PF, MKH, MKL, RXB, ESNT, ECPT0, ECPT1

| Note that uPD7807/7809 has two capture registers for |
| Event/Timer counter |

A 1.8 IE-7809 Mapping
When using map command, note that the maximum size of internal ROM for uPD7809 is 8 K (0 - 1FFFH) as opposed to 4K (0 - FFFH) for uPD78C11 / uPD7808 or uPD7811H.

A 1.9 CLK Command

For both IE-7809 & IE78C11

CLK I results in selection of an internal clock = 12MHz

CLK U results in selction of an external clock of 4 - 12 MHz range

A 1.10 IE-7809 D=G Command

IE-7809 diagnostic tests "Port T" instead of "Analog In"; this is seen on monitor. Other diagnostic tests are the same.

A 1.11 IP2 Connections for RS232

On the controller board of all IE systems (IE-7811H, IE-78C11, IE-7809) IP2 jumpers may be altered if a 1:1 cable connections is desired between IE system and terminal. Thus IP2 would be connected as follows:

Pins 1 + 2	to 17		
" 3	to 16		
" 4	to 15	N.B.	This jumpening
" 5	open		functions for use
" 6	to 12 + 13		with IBM-XT or AT.
" 7	to 11		
" 8	to 9		

A 1.12 Instruction Set Differences

The uPD7809 has some extra instructions with respect to uPD7811H. These common use of watchdog timer and comparator port and, of special interest, bit manipulation functions on internal RAM.
The uPD78C11 has one extra instruction compared to uPD7811H. This is STOP instruction.

A 1.13 IE-78C11 Special Register Vonience

The uPD78C11/C10 has one extra special register compared with uPD7810/11/10H/11H; i.e. ZCM register (zero-crossing mode).

A 1.14 IE-78C11-M Mapping

Note that the IE-78C11-M when being used to emulate uPD78C14 can have a mapped ROM max. 16K (0 - 3FFFH).

A 1.15 IE-87AD-M Variation

The IE-87AD-M is used for emulation of uPD7810/11 and therefore has max. 12 MHz clock frequency. Thus quartz crystal in emulator is 12 MHz instead of 15 MHz.
Obviously, the monitor firmware for IE-87AD-M is different when compared to IE-7811H-M firmware.

SECTION B

INTEL BASED

IE-7809-I

IE-87AD-I

P R E F A C E

This manual applies to both IE-87AD-I and IE-7809-I emulators. The IE-87AD-I is used for emulation of uPD7810/11 and IE-7809-I is used for emulation of uPD7807/09. Both are for use on Intel MDS running ISISII Operating System.
The basis for this manual is IE-87AD-I and, therefore, reference is made basically to this emulator. In most cases for IE-87AD-I one simply reads IE-7809-I.

Where any variations occur for IE-7809-I details can be found in Appendix A.

Appendix B shows the instruction set and pin configuration for uPD7809.

Appendix C contains Electrical Specification for uPD7809.

CHAPTER 1 SYSTEM OVERVIEW

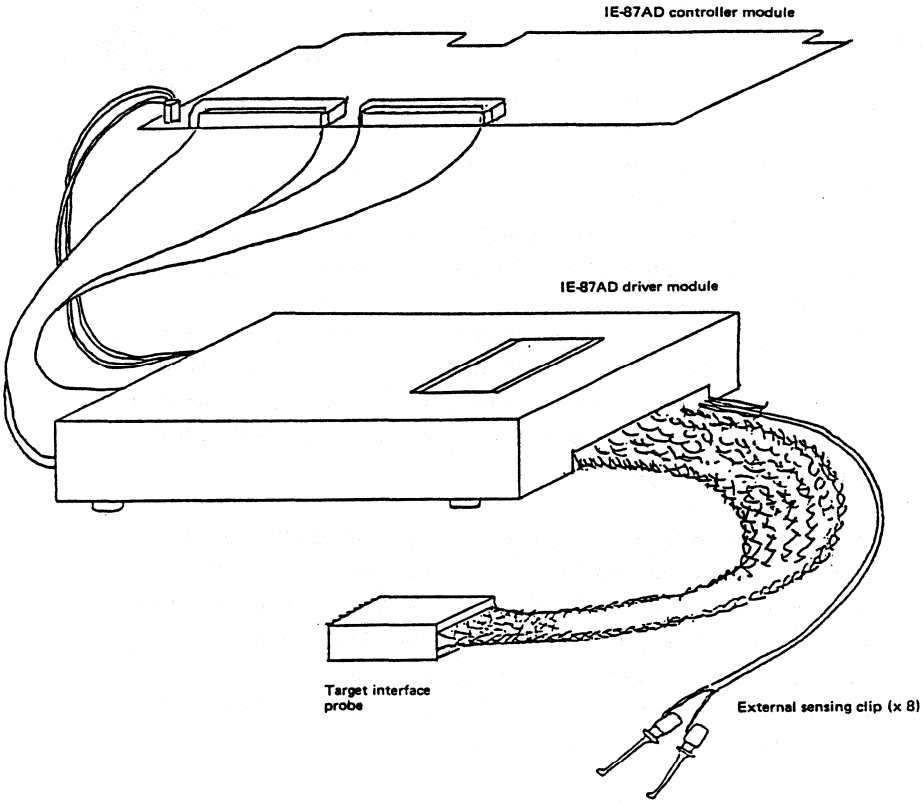
1.1 Introduction

The IE-87AD (In-Circuit Emulater for μCOM87AD) is a development support system designed for effectively developing the hardware and software using μCOM87AD. It utilizes an MDS (Microcomputer Development System) manufactured by the INTEL CORP. as a host machine.
By inserting a 64-pin connector into the CPU (μCOM87AD) socket of a prototype system instead of a CPU, the prototype system can operate as though a CPU was actually inserted there.
In addition, the IE-87AD system software supplied by the disk base provides the CPU with debugging capability. Efficient debugging can be guaranteed even if the prototype system has no debugging capability.

CHAPTER 2 OVERVIEW

2.1 Hardware Specifications

2.1.1 External view



2.1.2 IE-87AD module configuration

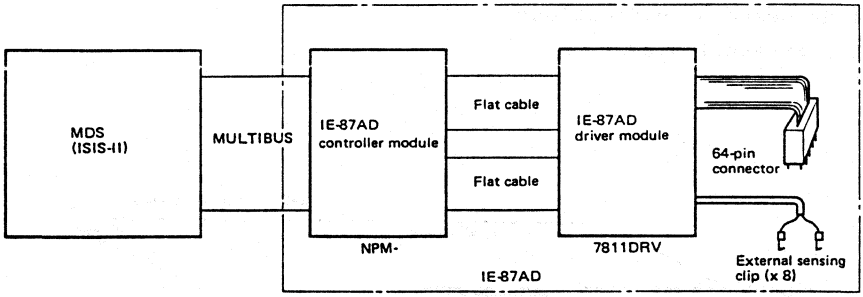


Fig. 2-1 IE-87AD Module Configuration

As shown in Fig. 2-1, the IE-87AD consists of two types of modules: the controller module and the driver module.

- Controller module
  The controller module mounted on the MDS expansion slot not only controls the IE-87AD driver module but also has a trace function that can trace up to 1,023 machine cycles. This module is exactly the same as the module for the NEC IE system.
- Driver module
  The driver module connected to the controller module with two flat cables and one power cable forms the main part of the IE-87AD system. A 64-pin connector and 8 external sense clips are provided for connection with the prototype system side of this module.
  The block diagrams of these two modules are shown in Figs. 2-2 and 2-3, respectively.

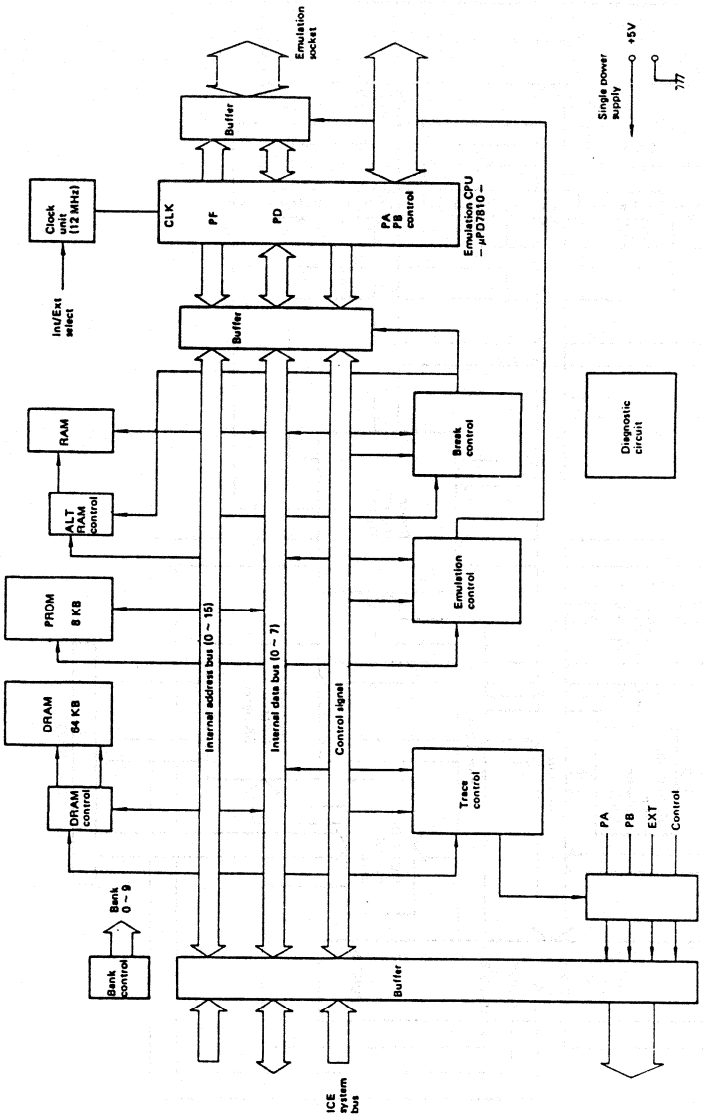


Fig. 2-2 Block Diagram of the IE-87AD Driver Module

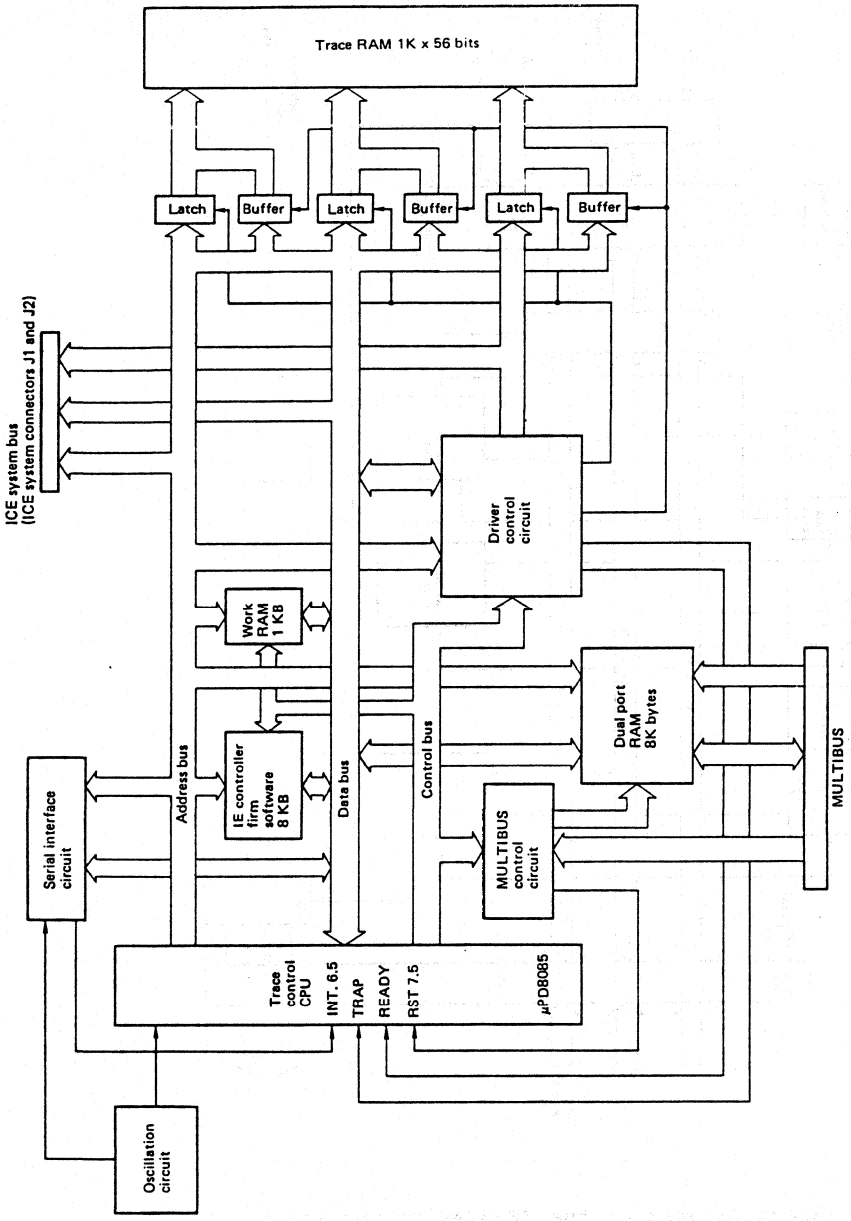


Fig. 2-3 Block Diagram of the IE-87AD Controller Module

2.1.3 Emulation

(1) Target interfacing

Direct interfacing with the target system via 64-pin IC socket

(2) CPU

μPD78P11

(3) MODE0, MODE1 terminals

The levels of these terminals are read by the prototype system and the mode is set accordingly. This mode setting cannot be performed by software. Therefore, when the prototype system is being designed, M0 and M1 must be determined by hardware. If no prototype system is connected, the values of M0 and M1 are undefined as is the mode setting. The following four modes are set by M0 and M1.

Table 2-1 Expansion Modes of μCOM87AD

M1	M0	Mode	CPU
1	1	External 64K bytes	μPD7810 mode
0	1	External 16K bytes	
0	0	External 4K bytes	
1	0	Internal ROM An internal ROM of 4K, 6K or 8K bytes may be selected. External memories of 256, 4K 16K or 64K bytes may be selected.	μPD7811 mode

NOTE: When using μPD7811 for the CPU, the internal 4K-byte ROM should be mounted. 6K- and 8K-byte ROM capacity is used for expansion.

(4) Clocks

Both internal and external clocks may be selected. The internal clock is fixed as 12MHz and for the external clock, the clock of the prototype system is used.

2.1.4 Break

Break control is performed entirely by hardware.

(1) In this system, the three types of breaks are performed. Address break is specified when read/write to the address set in the address register is performed. This register is completely independent of the memory and any number of addresses within a 64K-byte range may be specified as address break conditions. Data break conditions are set in the data register. Any data in the range 0 to 0FFH may be set. The third type of break is performed in response to a control signal.

(2) A loop counter is provided for use with any of the above breaks.

(3) If an unmapped memory location is accessed, break will be applied automatically.

(4) Break can also be performed according to conditions set for the external sensing signals. These signals are sent via 8 external clips connected to the driver module of the IE-87AD for hardware diagnosis. Each clip is provided with a ground clip.

(5) If the "ESC" key is pressed during emulation, forced break will be applied and emulation will be terminated.

(6) Timer break condition can also be set in the range 4ms to 65,535ms. Note, however, that the timer used for this has an aberration of 1ms.

2.1.5 Trace

(1) Trace mode

The following three trace modes are available.

(a) Non-trace mode

In this mode, none of the results of emulation are traced.

(b) Normal trace

In this mode, all of the trace items are traced while the CPU is performing emulation. However, if emulation is performed for more than 1,023 machine cycles, only trace data for the most

recent 1,023 machine cycles will remain in the memory.

(c) Conditional trace

In this mode, trace is performed only for the range specified by the TRACE ENABLE command. (refer to the description of commands.) These conditions are controlled by hardware. Therefore, when performing trace for emulation that exceeds 1,023 steps, by specifying trace only for specific data, data that would otherwise be lost as described in (b) above can be retained in the memory.

(2) Trace items

The following items can be traced.

(a) Address and data buses

(b) RD, WR and M1

(c) Port A

(d) Port B or external sensing signals

2.1.6 Mapping

The IE-87AD is provided with the following mapping functions.

(1) Mapping conditions

(a) NONMAP (upon power application)

In this state, no memory is mounted and therefore, mapping is not performed.

(b) External mapping

Memory allocation for the prototype system is performed.

(c) Internal RAM mapping

Memory allocation for RAM addresses in the IE-87A is performed.

(d) Internal ROM mapping

Memory allocation for ROM addresses in the IE-87AD is performed. Data in this area will not change even if write accessed during emulation.

(2) Memory area to be mapped
All of the 64K bytes of internal memory can be mapped.
(3) Mapping unit
Mapping is performed in units of 256 bytes.
(4) Internal memory resources
Dynamic RAM (μPD4164-3)

2.1.7 Self-diagnostic
The IE-87AD has a self-diagnostic function that checks the following items:
(1) Internal memory
(2) Terminals MODE0 and MODE1
(3) Ports D and F
(4) Ports A through C
(5) A/D input ports (AN0 through AN7)(Port T for IE-7809)
(6) Serial I/O

2.2 Software Specifications
2.2.1 Load/save
This function loads or save the HEX-format object or symbol files on a disk with LOAD or SAVE commands.

2.2.2 List output
This function enables the contents to be output by the IE-87AD to a line printer or a disk.

2.2.3 Register operation
This function displays or sets the contents of the μCOM-87AD registers.

2.2.4 Port operation
This function displays or sets the levels of each bit of the μCOM-87AD ports.

2.2.5 Memory command
The memory command is used to display or set the contents of the mapping memory in byte (8 bits) and word (16 bits) formats. Different from the memory image, in

the WORD format, the memory contents are displayed or modified starting from the highest-order byte.

2.2.6 Trace display

This function displays traced data in machine cycle (without dsiassembling) or instruction (with disassembling) format.

2.2.7 On-line assembler

This function allows the user to change the contents of the mapped memory by inputting mnemonic levels.

2.2.8 Disassembler

This function disassembles and displays the contents of the mapped memory.

2.2.9 Symbols

The functions related to symbols are as follows:

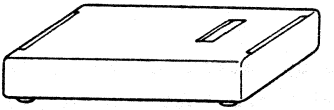
(1) Display of symbol addresses
(2) Alteration of symbol addresses
(3) Definition of new symbol addresses
(4) Deletion of symbols

2.2.10 Mapping

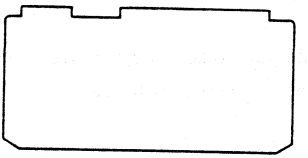
This function allows specification of memory ranges for access of the μCOM87AD. This specification is performed in units of 256 bytes and the specifications available are the internal and external RAM, the external ROM and NONMAP. However, in μPD7811 of other modes, or when the RAE setting by the μPD7811 is ON, memory will be automatically allocated as addresses 0 to 4K (6K or 8K) or 0FF00 to 0FFFF and it will not be possible to alter the result of memory mapping performed in this manner. In such a case, remapping must be performed by resetting terminals M0 and M1 and reinitializing the system by inputting the RESET HARD command.

CHAPTER 3 SYSTEM CONFIGURATION

The IE-87AD consists of the following components:



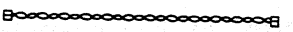
Driver module 8711DRV



Controller moudle NPM

Flat cable J1

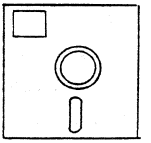
Flat cable J2



Power cable

Interface probe

External sensing clip



Floppy disk
(system software)

1. Driver module
   This module is the main element of the IE-87AD and provides hardware to support emulation, break, mapping functions, etc.

2. Controller module
   This module controls the operation of the IE-87AD driver module, communications with the host system (MDS), etc. This module is essentially the same as the IE controller manufactured by NEC.

3. Flat cables J1 and J2
   These 50-pin flat cables are used to connect the controller module and the driver module.

4. Power cable
   This cable is used to supply power to the driver module from the controller module. As all power necessary for the operation of the IE-87AD is supplied from the expansion unit of the MDS, do not connect another system to the MDS while the IE-87AD is connected to the MDS.

5. Interface probe
   This is a 64-pin probe from the driver module used for interfacing with the prototype system. Connection is performed by inserting this probe in the CPU socket of the prototype socket.

6. External sensing clips
   8 external sensing clips from the driver module are provided for hardware diagnosis. When these clips are connected to the prototype system and a trace select command specifying trace of external signals is input, the signal state of each clip is traced. Trace timing is that of the IE-87AD. This external trace data can be displayed by the trace display command.

The external sensing clips can also be used to perform break. When the signal states of the locations connected to the clips match those set as the external break condition, break is applied.

7. Floppy disk (system software for IE-87AD)
   This ISIS-II base floppy disk is used to perform communication between the IE-87AD and the MDS host. When the system software of the IE-87AD is initialized, this software is loaded into the memory of the MDS. As an overlay configuration is used, the execution time of each command varies to some extent.

CHAPTER 4 OPERATION OF IE-87AD

4.1 How to Connect the IE-87AD

Connect the IE-87AD in the following sequence:

(1) Turn OFF the power supplies of both the MDS and the prototype system.

(2) Insert the controller module into the MDS expansion slot.

(3) Connect the controller and driver modules with the two flat cables J1 and J2 and one power cable. Be sure to connect these flat cables correctly (Fig. 4-1).

(4) Remove the CPU (μCOM87AD) from the prototype system.

(5) Insert the 64-pin connector of the driver module into the CPU socket of the prototype system.

(6) Turn ON the power supply of the prototype system and then that of the MDS. When turning OFF these power supplies, be sure to first turn OFF the power supply of the prototype system and then that of the MDS.

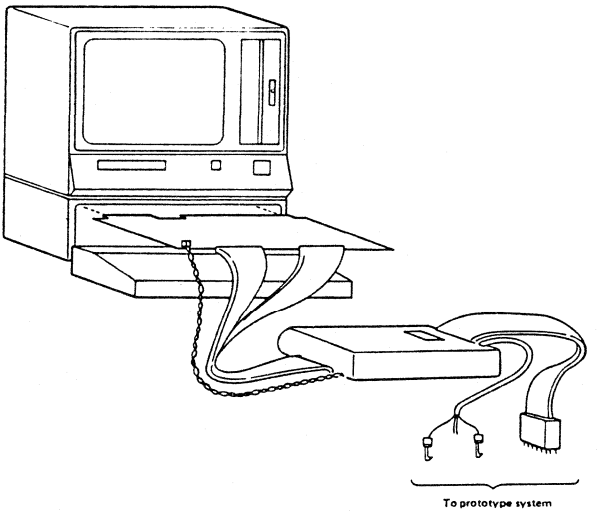


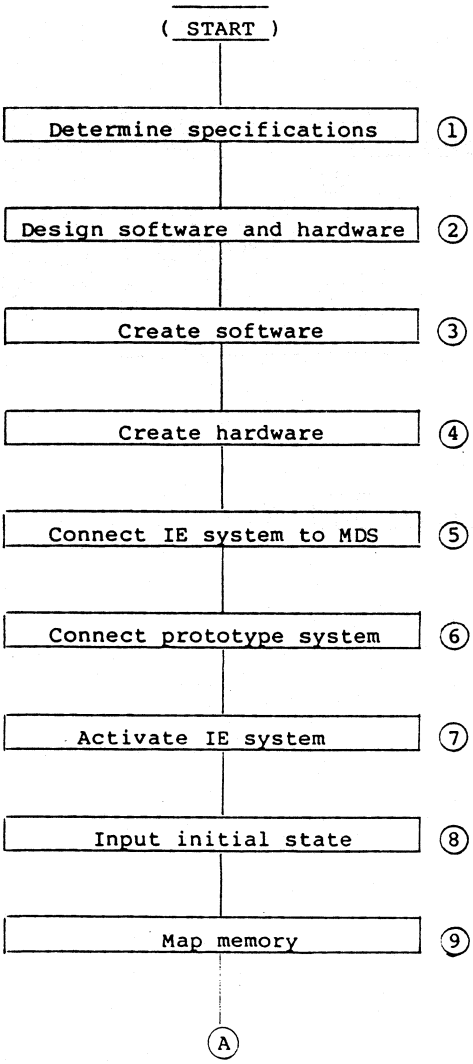
Fig. 4-1 Connection Diagram of IE-87AD

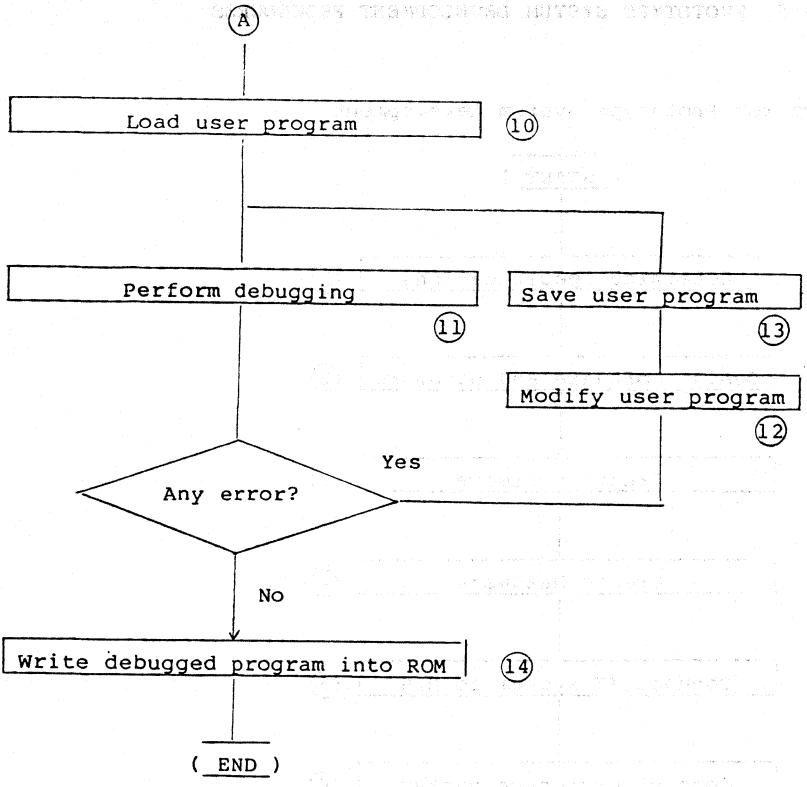
4.2 Cautions

- Be sure to set flat cables J1 and J2 correctly. Wrong connection of these cables can cause damage to the ICs used in the system.
- When inserting the controller module in an expansion unit connected to the IE-87AD, be sure that the total current required by all of the boards connected to the exapnsion unit does not exceed the capacity of the expansion unit. The average current used by the IE-87AD is 6.5A.

CHAPTER 5 PROTOTYPE SYSTEM DEVELOPMENT PROCEDURES

5.1 Flowchart for Prototype System Development




A
Load user program
10
Perform debugging
11
Save user program
13
Modify user program
12
Any error?
Yes
No
Write debugged program into ROM
14
(END)

① : Determine the specifications of the prototype system.
② : Design the software and hardware of the prototype system.
③ : Code the prototype system software with an assembly language, assemble it on the MDS, and create object and symbol files.
④ : Create the prototype system hardware.
⑤ : Connect the IE-87AD to the MDS.
⑥ : Connect the prototype system to the IE-87AD.
⑦ : Activate the IE-87AD in the ISIS-II mode.
⑧ : Input the initial state of the IE-87AD.
⑨ : Map memory with a MAP command. If the memory of the prototype system is incomplete, perform the mapping in the IE-87AD internal memory.
⑩ : Load the prototype software (object codes and symbol tables) to the mapped memory with a LOAD command.
⑪ : Debug the software and hardware of the prototype system with an IE-87AD command.
⑫ : If an error occurs, modify the user program.
⑬ : Save the modified program on the disk with a SAVE command.
⑭ : Write the debugged program into the PROM.
⑮ : Mount the CPU and PROM on the prototype system.

Cautions when developing prototype systems

- Set the values of registers MM and MF to the condition at the time of IE-87AD initialization.
- Set register MD in the same manner.
- If the cause (CAUSE) of a break is a memory error, this indicates that the internal RAM is defective. Should such an error occur, please contact NEC.
- Analog values vary in the range $\pm$ 2% although this figure is not guaranteed and is subject to further fluctuation depending on how the IE-87AD is handled.

° Execution of the HLT command will force termination of emulation. After a HLT command has been executed, pressing the "ESC" key to force break will be ignored. In this case, you should apply an external interrupt signal or a reset signal.

CHAPTER 6 COMMANDS

The IE-87AD console commands consist of keywords and special characters.
The special characters are:

- "."
- "/"
- ","
- "="

108 character strings are registered as keywords and the abbreviated forms of these keywords can be used.
The free abbreviation method is used for these abbreviations. In this method, the IE-87AD sequentially compares the input characters with the registered keywords from the beginning to check whether or not the keyword corresponding to the input characters exists. If there is even one keyword corresponding to the input characters, the IE-87AD recognizes that this keyword is the one corresponding to them. This means what the user should input only the minimum number of characters recognizable by the IE-87AD.
Take keyword ALL for example.
Keywords beginning with A are ALL, AND, ANM, and ASM. To enable the IE-87AD to recognize that A indicates ALL, input AL or ALL:

Symbols for IE-87AD keywords

ALL	AL	
AND	AND	
ANM	ANM	(not IE-7809)
ASM	AS	
BA0	BA0	
BA1	BA1	
BCNT	BC	
BDR	BD	
BEX	BE	
BR	BR	
BTMR	BT	
BYTE	BY	
CAUSE	CA	
CHECK	CH	
CLOCK	CL	
CPU	CP	
CR	CR	(not IE-7809)
CR0	CR0	
CR1	CR1	
CR2	CR2	
CR3	CR3	
CY	CY	
CYCLE	CYC	
DATA	DAT	
DASM	DAS	
DEFINE	DE	
DISABLE	DI	
EA	EA	
ECNT	ECN	
ECPT	ECP	(not IE-7809)
ENABLE	EN	
EOM	EO	
ETM0	ETM0	
ETM1	ETM1	
ETMM	ETMM	
EXIT	EXI	
EXTERNAL	EXT	
FETCHED	FE	
FOREVER	FO	
FROM	FR	
GO	GO	
GUARDED	GU	
HARDWARE	HA	
HC	HC	
IN	IN	
INSTRUCTION	INS	
INTERNAL	INT	
L0	L0	
L1	L1	
LENGTH	LE	
LIST	LI	
LOAD	LOA	
LOCATION	LOC	
LOOP		
MA	MA	
MAP	MAP	
MB	MB	
MC	MC	
MCC	MCC	
MF	MF	
MK	MK	
MKH	MKH	
MKL	MKL	
MM	MM	
MODE	MOD	
MOVE	MOV	
NEWEST	NE	
NOSYMBOL	NOS	
NOCODE	NOC	
OLDEST	OL	

OR ..... OR
OUT ..... OU
PC ..... PC
PORT ..... PORT
PORTA ..... PORTA
PORTB ..... PORTB
PORTC ..... PORTC
PORTD ..... PORTD
PORTF ..... PORTF
PRINT ..... PR
PSW ..... PS
RA ..... RA
RAM ..... RAM
RB ..... RB
RC ..... RC
RD ..... RD
RDE ..... RDE
RE ..... RE
READ ..... REA
REGISTER ..... REG
REMOVE ..... REM
RESET ..... RES
RH ..... RH
RHL ..... RHL
RL ..... RL
ROM ..... RO
RV ..... RV
RVA ..... RVA
SAVE ..... SA
SERIAL ..... SE
SK ..... SK
SMH ..... SMH
SML ..... SML
SP ..... SP
STEP ..... ST
SYMBOL ..... SY

TILL ..... TIL
TIMER ..... TIM
TM0 ..... TM0
TM1 ..... TM1
TMM ..... TMM
TO ..... TO
TRACE ..... TR
USER ..... U
WITH ..... WI
WORD ..... WO
WRITTEN ..... WR
Z ..... Z

IE-7809 only:

ECPT0/ ..... ECPTØ
ECPT1 ..... ECPT1
MT ..... MT
PORT1 ..... PORT1
WDM ..... WDM

CHAPTER 7 DESCRIPTION OF RESPECTIVE COMMANDS

7.1 Outline

This chapter details the IE-87AD commands. The representations used in this cnapter are:

- Partition

  Indicates the address range. The format of the partition is:

  s.adr[TO e.adr]
  s.adr[LENGTH size]
  s.adr: Start address
  e.adr: End address
  size: Memory size

  Those parameters enclosed in brackets can be omitted. If they are omitted, one address point is assumed. Public symbol references and operation expressions, etc., can be specified in the s.adr and e.adr.

  Example:

  ° 1000H
  ° 1000H TO 10FFH
  ° .ABC TO .ABC+10
  ° .ABC LENGTH 10

- Contents

  Indicates a constant and is used to set registers, etc. Digits and arithmetic expressions can be specified in the contents.

  For example, if RA=contents is specified, the digit specified in the contents is set in the RA.

  Binary, octal, decimal, and hexadecimal numbers can also be input. In this case, input B (binary), Q (octal), O (decimal), or H (hexadecimal) following the input digits as in the following example (O indicating the decimal input can be omitted):

  Example:

  ° Binary : 10001000B
  ° Octal : 30Q
  ° Decimal : 40
  ° Hexadecimal: 44H

Digits can be combined with operators as follows:

° 30 + 40G

° 100H - 42H

The following operators can be specified.

+: Addition

-: Subtraction

*: Multiplication

/: Division

7.2 Commands

(1) IE87AD command

This command initilizes the IE-87AD on the ISIS-II. When this command is accepted, the message "******* IE-87AD VX.X ******* is output.

If the message "SYSTEM FIRMWARE IS SICK" is output and the mode returns to the ISIS-II mode, check the hardware setting because the hardware has been set incorrectly.

When the hardware setting is correct, the IE-87AD next sets the initial state.

:drive:IE87AD

Example:

:F1:IE87AD

:drive: Indicates the disk drive number where the IE-87AD software is stored. The number must follow immediately after character F preceded by a colon (:) and a colon must also follow the number to enclose it.

IE87AD Indicates the program filename of the IE-87AD operating under control of the ISIS-II. The filename must immediately follow the disk drive number (:drive:). No blank is allowed between the drive number and the filename.

o Setting the initial state of the IE-87AD

When the IE-87AD command has been executed and the hardware check has been performed, the initial state of the IE-87AD will be set as follows.

(1) Display of M0 and M1

The levels of M0 and M1 are read from the target system and output to the console. A high level signal is displayed as '1' and a low level signal as '0'. If the target system is not connected to the IE-87AD, M0 and M1 will be unstable. They should therefore either be pulled up or grounded.

Example:

```
MODE0=1
MODE1=0
```

(2) Display and set of external memory

The size of the external memory selected by the values of M0 and M1 in (1) above will now be displayed.

Example:

```
EXT. MEMORY=16K
```

If M0 is low level and M1 is high level, the following message will be displayed.

```
EXT. MEMORY (0/256/4(K)/16(K)/64(K))=
```

Therefore, to set the size of the memory, input as follows.

```
No external memory    =0↵
256 byte ext. mem.    =256↵
4K byte ext. mem.     =4↵
16K byte ext. mem.    =16↵
64K byte ext. mem.    =64↵
```

This process sets which of the bits of PD and PF will be used as the address bus.

③ Setting access of the internal RAM
Next, the following message will be displayed.

RAE (ENABLE/DISABLE)=

If you wish to access the internal RAM input

=ENABLE ↵

and in all other cases input

=DISABLE ↵

At this point, if "ENABLE" is input, the internal RAM area will be mapped by the IE system. Once set, this mapping condition cannot be changed by input of a mapping command.

④ Setting the size of the internal ROM
When M0 is low level and M1 is high level, this setting can be performed. The message "INT.ROM=" will be displayed. If the target CPU is a μPD7811, input "=4 ↵".
This will specify an internal ROM size of 4K bytes. It is also possible to input "6" or "8" to specify 6K- and 8K-byte internal ROM sizes. However, these settings are for use with an expanded version.

⑤ Mode setting of Port D
This setting specifies whether Port D is to be used for input or for output. This setting can only be performed when no external memory is mounted.

PD (DISABLE/IN/OUT)=

When the above message is displayed, input as follows.

If you do not wish to use Port D =DISABLE↵
To use Port D for input =IN ↵
To use Port D for output =OUT ↵

⑥ Mode setting of Port F

Unless the external memory used is 64K bytes (in which case all the bits of both PD and PF will be used as the address bus), each bit of Port F can be specified for input or output. The following message will be displayed.

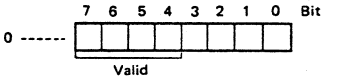
PFX-7 (0:IN/1:OUT)

In this message, the value of X will indicate the following external memory sizes.

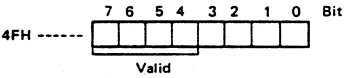
X: 0: External memory size is less that 256K bytes.
4: External memory size is 4K bytes.
6: External memory size is 16K bytes.

Example:
When the external size is 4K bytes, the following message will be displayed.

PF4-7 (0:IN/1:OUT)=



When "=0 ↵" is input, 0 is taken as an 8-bit value and bits 4 to 7 will be become '0' and will therefore be used for input. At this time, the input will be ignored for bits 0 to 3.

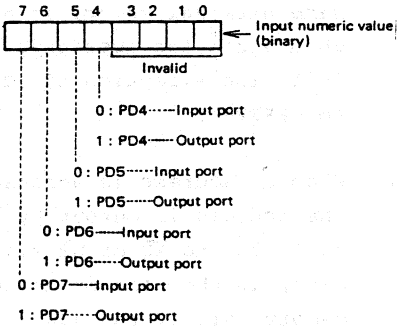


If "=4FH ↵" is input, this value will again be taken as an 8-bit value with the result that bit 6 will be set to '1' (output) and bits 7, 5 and 4 to '0' (input). This setting can also be performed using binary numbers (4FH for example would be input as 01001111B).

This completes the setting of initial values for the IE-87AD. The prompt sign "*" is displayed to indicate the the In-circuit emulator is ready for command input. IE-87AD mode is now entered. The input method to set the initial values for each mode are described below.

NOTE: Setting of MM, MF and MD in a user program is performed in the same manner as described above. When using the 7811 as the CPU, MM should be set to 0. Also, immediatly after reset, PD and PF will both be enabled for input. Each bit of PF takes the initial value of register MM. The setting of PD likewise takes the value of register MD when it is set.

In the 7810 mode, PD and PF take the values specified by the mode setting immediately after reset.

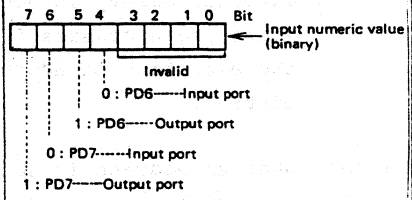
Setting of initial values when M0=0 and M1=1		
1.	Display of ext. memory EXT.MEMORY=	"4K" is displayed.
2.	Setting of ext. memory	None PF3 to 0 are automatically set in address mode.
3.	Setting of RAE RAE (ENABLE/DISABLE)=	ENABLE↵ Selects internal RAM of the IE-87AD. DISABLE↵ Selects use of an external memory.
4.	Setting memory size of internal ROM	None
5.	Mode setting of Port D	None Port D is automatically set in the bus mode.
6.	Mode setting of Port F PF4-7(0:IN/1:OUT)=	Bits 4 to 7 of the input value are valid.  Example: When 01010000B↵ is input, PD4 and PD6 are set as output ports and PD5 and PD7 are set as output ports.

Relation of mapping commands

When RAE is enabled by the procedure described on the preceding pages, memory area 0 to 0FEFFH can be mapped as the user desires. That is, the initial state of this area is NONMAP. Addresses 0FF00 to 0FFFFH are allocated as the RAM. When RAE is disabled, the entire memory area addresses 0 to 0FFFFH can be mapped in any way the user wishes. This entire area is in the NONMAP state after initialization. The procedure for accessing the respective memory areas is as described below.

Mapping external memory: When an address in one of these ranges is accessed by the CPU, the lower order bits (0 to 7) of the address are output via PD and the upper order bits (8 to 13) via PF. When the address has been specified, the $\overline{RD}$ or $\overline{WR}$ signal is output. Therefore, when address in the range 0n000H to OnFFFH,(n: 1 to F) the same external memory is accessed as when accessing addresses 0FF00 to 0FFFH (with the exception of addresses 0FF00H to 0FFFFH).

Mapping internal ROM and RAM memory: When an address is accessed by the CPU, the address is output by PD and PF (0 to 3) but no $\overline{RD}$ or $\overline{WR}$ signal is output. Also, in the case of an internal memory, all of the addresses 0 to 0FFFFH are treated as separate addresses.

Setting of initial values when M0=1 and M1=0		
1.	Display of ext. memory EXT.MEMORY=	"16K" is displayed.
2.	Setting of ext. memory	None PF5 to 0 are automatically set in address mode.
3.	Setting of RAE RAE (ENABLE/DISABLE)=	ENABLE↵ Selects internal RAM of the IE-87AD. DISABLE↵Selects use of external memory.
4.	Setting memory size of internal ROM	None
5.	Mode setting of Port D	None Port D is automatically set in the bus mode.
6.	Mode setting of Port F PF6-7(0:IN/1:OUT)=	Bits 6 and 7 of the input value are valid.  Example: When 0FFH↵is input, both PD6 and PD7 are set as output ports.

Relation of mapping commands

When RAE is enabled by the procedure described on the preceding pages, memory area 0 to 0FEFFH can be mapped as the user desires. That is, the initial state of this area is NONMAP. Addresses 0FF00 to 0FFFFH are allocated as the RAM. When RAE is disabled, the entire memory area addresses 0 to 0FFFFH can be mapped in any way the user wishes. This entire area is in the NONMAP state after initialization. The procedure for accessing the respective memory areas is as described below.

Mapping external memory: When an address in one of these ranges is accessed by the CPU, the lower order bits (0 to 7) of the address are output via PD and the upper order bits (8 to 13) via PF. When the address has been specified, the $\overline{RD}$ or $\overline{WR}$ signal is output. Therefore, when addresses 4000H to 7FFFH, 8000 to 0BFFFH and C000 to 0FFFFH are accessed, the same external memory is accessed as when accessing addresses 0 to 03FFFH (with the exception of addresses 0FF00H to 0FFFFH).

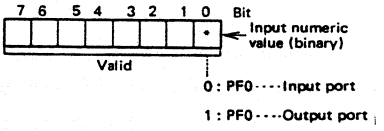
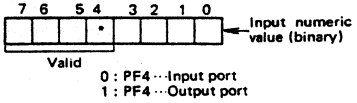
Mapping internal ROM and RAM memory: When an address is accessed by the CPU, the address is output by PD and PF (0 to 5) but no $\overline{RD}$ or $\overline{WR}$ signal is output. Also, in the case of an internal memory, all of the addresses 0 to 0FFFFH are treated as separate addresses.

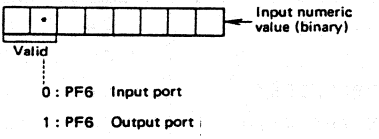
Setting of initial values when M0=1 and M1=1		
1.	Display of ext. memory EXT.MEMORY=	"64K" is displayed.
2.	Setting of ext. memory	None
3.	Setting of RAE RAE (ENABLE/DISABLE)=	ENABLE) Selects internal RAM of the IE-87AD. DISABLE) Selects use of an external memory.
4.	Setting memory size of internal ROM	None
5.	Mode setting of Port D	None
6.	Mode setting of Port F	None

Relation of mapping commands

When RAE is enabled by the procedure described on the preceding pages, memory area 0 to 0FEFFH can be mapped as the user desires. That is, the initial state of this area is NONMAP. Addresses 0FF00 to 0FFFFH are allocated as the RAM. When RAE is disabled, the entire memory area addresses 0 to 0FFFFH can be mapped in any way the user wishes. This entire area is in the NONMAP state after initialization. The procedure for mapping the respective memory areas is as described below.

Mapping external memory:	When an address in one of these ranges is accessed by the CPU, the lower order bits (0 to 7) of the address are output via PD and the upper order bits (8 to 15) via PF. When the address has been specified, the $\overline{RD}$ or $\overline{WR}$ signal is output.
Mapping internal ROM and RAM memory:	When an address is accessed by the CPU, the address is output by PD and PF (0 to 3) but no $\overline{RD}$ or $\overline{WR}$ signal is output. Also, in the case of an internal memory, all of the addresses 0 to 0FFFFH are treated as separate addresses.

Setting of initial values when M0=0 and M1=1		
1.	Display of ext. memory	None
2.	Setting of ext. memory EXT.MEMORY(0/256/4(K)/16(K)/64(K))=	0↵ No ext. memory 256↵ 256-byte ext. mem. 4↵ 4K-byte ext. mem. 16↵ 16K-byte ext. mem. 64↵ 64K-byte ext. mem.
3.	Setting of RAE RAE (ENABLE/DISABLE)=	ENABLE↵ Selects internal RAM of the IE-87AD. DISABLE↵ Selects use of an external memory.
4.	Setting memory size internal ROM	4↵ Specifies 4K-byte ROM (with 7810) 6↵ Specifies 6K-byte ROM (expanded version) 8↵ Specifies 8K-byte ROM (expanded version)
5.	Mode setting of PD(DISABLE/IN/OUT)	Valid only when 0 is input in 2. above. DISABLE PD not used. IN PD used as input port. OUT PD used as output port.
6.	Mode setting of Port F (a) When 0 or 256 is input in 2. above. PF0-7(0:IN/1:OUT)=	
	(b) When 4 is input in 2. above. PF4-7(0:IN/1:OUT)=	

Setting of initial values when M0=0 and m1=1		
	(c) When 16 is input in 2. above. PF6-7(0:IN/1:OUT)=	

Relation of mapping commands

When the size of the internal ROM is specified as 4K (when the 7811 is used), addresses 0 to 0FFFH are mapped as the ROM.
When RAE is enabled by the procedure described on the preceding pages, memory area 0FF00H is mapped as the internal RAM. Therefore, upon initialization, addresses 0 to 0FFFH are ROMMAP, addresses 1000 to 0FEFFH are NONMAP and addresses 0FF00 to 0FFFFH are RAMMAP.
The procedure for accessing the respective memory areas is as described below.

Mapping external memory: When an address is accessed by the CPU, the lower order bits (0 to 7) of the address are output via PD and the upper order bits (8 to 13) via PF. When the address has been specified, the RD or WR signal is output.

Mapping internal ROM and RAM memory: When an address is accessed by the CPU, the address is output by PD and PF (0 to 5) but no RD or WR signal is output. Also, in the case of an internal memory, all of the addresses 0 to 0FFFFH are treated as separate addresses.

(2) EXIT command

This command changes the mode from IE-87AD mode to ISIS-II mode.

EXIT

Example:

Exit

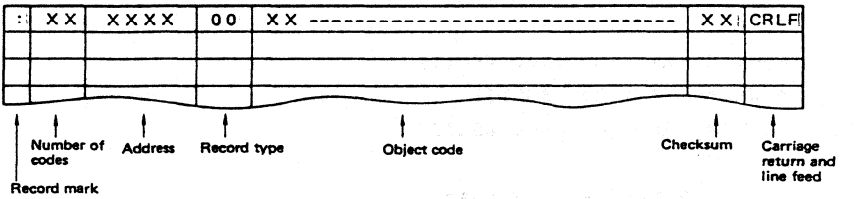
---

(3) LOAD command

This command loads the HEX-format object and symbol files created on a disk.

This command loads the HEX-format object file to the mapped memory of the ICE or the target system. If an unmapped memory exists, the IE-87AD outputs the message "MEMORY GUARDED", terminates processing, and returns to the command input wait state.

The format of the HEX-Format object file is:



Object File Format

The terminator must satisfy the following conditions:

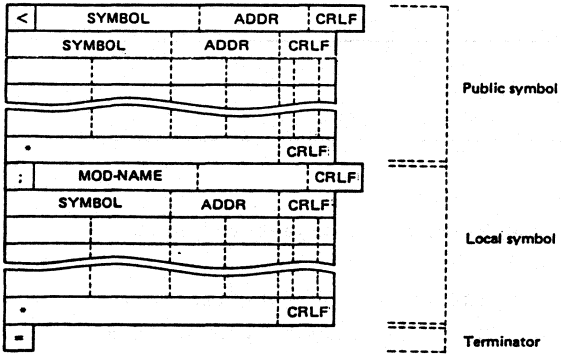
The record type is "01".

The number of codes is "00".

The checksum is consistent.

The symbol file is loaded to the MDS memory. If the symbol file is too large, the IE-87AD outputs the message "SYMBOL TABLE OVERFLOWED" and interrupts processing.

The format of the symbol file which consists of public and local symbols is:



<	Header of public symbol
;	Header of local symbol
*	Terminator of public/local symbol
=	Terminator of symbol file

SYMBOL	Symbol name (6 bytes ASCII)
ADDR	Address (4 bytes ASCII)
MOD-NAME	Module name (6 bytes ASCII)

Symbol File Format

Only one file (symbol or object) may be loaded. These object files are available separately from NEC. Eitherthe direct output from the μCOM87AD absolute assembler or the output of the μCOM87AD relocatable assembler can be input after being converted to a hex code by the hex converter.

LOAD command

```
LOAD:drive:filename [NOCODE  ]
                    [NOSYMBOL]
```

Example:

```
LOASD:F1:SAMPLE
LOAD:TEST1 NOSYMBOL
LOAD:TEST2 NOCODE
LOAD:F1:SAMPLE.H1 NOSYMBOL
```

LOAD	Command keyward that loads the object or the symbol table from the specified file of the specified disk.
:drive:	Indicates the disk drive number containing the disk file to be loaded. The default value is ":F0:".
filename	Indicates the the filename to be loaded. If no expansion identifier is specified, the object file is assumed to be "HEX" and the symbol file is assumed to be ".SYM".
NOCODE	Indicates that no object file is to be loaded (keyword).
NOSYMBOL	Indicates that no symbol file is to be loaded (keyword).

4) SAVE command

This command saves the object in the IE-87AD or target system memory and the symbol table in the MDS on the disk. The formats of the object and symbol files to be saved are the same as those for the LOAD command.

A partition is used to specify the range of the objects to be saved. If the partition is omitted, the range is from 0 through FFFFH. And if an unmapped area exists in the range, the IE-87AD outputs the message "MEMORY GUARDED" and terminates processing.

Either object files or symbol files can be saved.

SAVE command

```
SAVE:drive:filename |NOCODE/partition |
                    |NOSYMBOL         |
```

Example:

```
SAVE:F1:EXAMP
SAVE:F4:SAMPLE 0 to 0FFH
SAVE:F0:SAMP2 NOCODE
SAVE:F1"SAMPLE VP1 NOSYMBOL 128 LENGTH 100
```

SAVE	Command keyword that saves the specified object or symbol table onto the specified drive file.
:drive:	Indicates the disk drive number containing the disk file to be saved. The default value is ":F0:".
filename	Indicates the filename to be saved. An expansion identifier can also be specified. If no expansion identifier is specicied, the the object file is assumed to be "HEX" and the symbol file is assumed to be "SYM".
NOCODE	Keyword indicating that no object file is to be saved.
NOSYMBOL	Keyword indicating that no symbol table is to be saved.
partition	Specifies the object save range. The format is "s.adr TO e.adr" or "s.adr LENGTH size".

(5) LIST commands

This command outputs the processing results of the IE-87AD to a console, line printer, or disk file. The results are always output to the console even if the disk file or the line printer is specified. The names of these output units are:

```
Console      :CO:
Line printer :LP:
Disk file    :drive:filename
```

The disk filename must be the one conforming to the ISIS-II management. ":CO:"(console) is specified as an initial state.
This command can also be used as the LIST display command that displays the currently set device name.

LIST display command

LIST
Example:
LIST

LIST Command keyword that displays the name of the currently set device

Example:
*LIST
:LP:
*LIST
:F1:SAMPLE.LST

LIST setting command

LIST:device
LIST:drive:filename
Example:
LIST:LP:
LIST:CO:
LIST:F1:TEST.LST

LIST Command keyword that outputs the processing results of the IE-87AD to the specified device or disk file.
:device: Specifies the output device managed by the ISIS-II (:CO: and :LP:).
:drive: Indicates the disk drive number containing the disk file to be output. The default value is :F0:.

filename Indicates the output filename conforming to the ISIS-II management. The first 6 characters of the filename indicate the name and the remaining 3 characters indicate the expansion identifier.

(6) Self-diagnostic command

This command causes the IE-87AD to perform a self-diagnostic. Before executing this command, always make sure that the IE-87AD probe is inserted in the self-diagnostic socket. When the self-diagnostic is performed, the following items will be checked.

① Memory check
The user memory and the ICE memory are both checked. If any abnormality is discovered, the bank number and address of the abnormality will be displayed. If a memory error is detected, the self-diagnostic will terminate and no further checks will be performed.
BANK=0 User memory
BANK=1 Internal memory of ICE system

② M0 and M1 check

③ Port A check
I/O check of Port A is performed.

④ Port B check
I/O check of Port B is performed.

⑤ Port C check
I/O check of Port C is performed.

⑥ Port D check
I/O check of Port D is performed.

⑦ Port F check
I/O check of Port F is performed.

⑧ Analog input (AN0 to AN7) check
Check of analog input is performed and if any abnormalities are detected, the number of the analog input will be displayed.
} different for IE-7809 see APPENDIX

⑨ Serial I/O check
Checks the serial I/O.

PASSED (normal) or FAILED (abnormal) indicating the results is displayed for each check item.
If Port D or F is defective, FAILED is displayed for both ports but information indicating which port is defective is not displayed. FAILED is also displayed for Ports A through C and Serial I/O if Port D or F is defective.
Self-diagnostic command

CHECK
Example:
CHECK

CHECK Command keyword that executes the IE-87AD self-diagnosis.

(7) Mapping commands

The IE-87AD can access a maximum of 64K-byte memory. This memory can be mapped to the memory of the user target system or the RAM in the IE-87AD 256 bytes at a time. The user target system program can be loaded into this memory. The IE-87AD can map the memory (area):

- as the RAM.
- as the ROM.
- to the user target system.

The 64K-byte RAM in the IE-87AD can be used to debug software even if the target system is incomplete.
If the memory area is mapped as ROM and is written during emulation, a break will occur. The addresses of blocks 0 through 255 are separated as follows:

Block No.	Low address	High address
0	0H	0FFH
1	100H	1FFH
2	200H	2FFH
3	300H	3FFH
.	.	.
.	.	.
254	FE00H	FEFFH
255	FF00H	FFFFH

The following three mapping commands are available:

(a) Mapping display command

This command displays the current mapping state. The format of this command is:

0000-0FFF INT ROM 1000-2FFF INT RAM
4000-60FF USER

Addresses which do not include any hexadecimal numbers (for example, 0000, 4000, and 1000) are the first addresses to be mapped while those including hexadecimal numbers (0FFF, 60FF, and 2FFF) are the last addresses to be mapped.

"INT ROM" indicates that addresses 0000 through 0FFF have been mapped to the IE-87AD as the ROM.

"INT RAM" indicates that addresses 1000 through 2FFF have been mapped to the IE-87AD as the RAM.

"USER" indicates that the addresses 4000 through 60FF have been mapped to the user target system.

The undisplayed areas indicate that the addresses have not been mapped.

(b) Mapping setting command

This command maps the blocks from the start block including the start address to the block including the end address in units of 256 bytes.

(c) Reset map command

This command clears all of the mapping states (unconnected memory states) except for the ROM and RAM in the IE-87AD.

Mapping display command

MAP

Example:

MAP

MAP This command keyword displays the current mapping state.

Example:

*MAP↵

0000-0FFF INT ROM 1000-1FFF INT RAM
4000-60FF USER

The addresses 0H through FFFH are mapped to the IE-87AD as the ROM.
The addresses 1000H through 1FFFH are mapped to the IE-87AD as the RAM.
The addresses 4000H through 60FFH are mapped to the user target system.

Mapping setting command

```
MAP |s.adr TO e,adr    |=|INTERNAL|[RAM]|
    |s.adr LENGTH size|  |        | ROM |
                         |USER
                         |GUARDED
```

Example:

```
MAP 0 TO 0FFH=INTERNAL ROM (1)
MAP 2000H LENGTH 400H=USER (2)
MAP 3010H TO 3100H=GUARDED (3)
MAP 0FFH TO 0F100H=INTERNAL (4)
```

MAP	Command keyword indicating the mapping command
s.adr	Indicates the start address. Symbol files can also be specified in this parameter.
e.adr	Indicates the end address. Symbol files can be specified in this parameter.
size	Indicates the memory size from the start address.
TO	Keyword indicating that the next numeric value is an end address.
LENGTH	Keyword indicating that the next numeric value is the memory size.
INTERNAL	Keyword that maps the address specified by the start address to end address parameters into the IE-87AD.
RAM	Keyword that maps the addresses specified by the start address to end address parameters as the RAM. This keyword can be omitted.

ROM Keyword that maps the addresses specified by the start address to end address parameters. If this keyword is omitted, RAM is assumed.

USER Keyword that maps the addresses specified by the start address to end address parameters to the user target system.

GUARDED Command keyword that clears the mapping.

(1) The mapping setting command maps addresses 0H through 0FFH into the IE-87AD as the ROM.

(2) The command maps addresses 2000H through 23FFH to the user target system.

(3) The command clears the mapping of addresses 3000H through 31FFH.

(4) The command maps addresses F000H to H1FFH into the IE-87AD as the RAM.

RESET MAP command

RESET MAP

Example:

RESET MAP

RESET Command keyword indicating the reset command.

MAP Command keyword that clears the mapping. This command cannot be used to clear the ROM and RAM in the chip mapped when the IE-87AD was initialized.

(8) CLOCK commands

This command sets the clock that operates the emulation chip in the IE-87AD. The following two types of clocks are available:

- Internal clock: This clock is generated in the IE-87AD.
- External clock: This clock is generated in the target system.

This command can also be used to display the clock that is currently set. Note that the initial state is set in the internal clock.

Clock display command

CLOCK

Example:

CLOCK

CLOCK Command keyword indicating the currently set clock.
"INTERNAL" is displayed when the internal clock is set.
"EXTERNAL" is displayed when the external clock is set.

CLOCK setting commands

CLOCK= |INTERNAL|
       |EXTERNAL|

Example:

CLOCK=INTERNAL

CLOCK=EXTERNAL

CLOCK Command keyword setting the clock that operates emulation chip.

INTERNAL Command keyword that sets the internal clock.

EXTERNAL Command keyword that sets the external clock.

(9) Processor-register commands

This command displays and sets the values of the IE-87AD processor registers. The following processor registers are available:

IE-87AD 8-bit registers

Register-name		Register-name	
RV	Register V	RV'	Register V' (rear)
RA	Accumulator	RA'	Register A' (rear)
RB	Register B	RB'	Register B' (rear)
RC	Register C	RC'	Register C' (rear)
RD	Register D	RD'	Register D' (rear)
RE	Register E	RE'	Register E' (rear)
RH	Register H	RH'	Register H' (rear)
RL	Register L	RL'	Register L' (rear)

IE-87AD 16-bit registers

Pair-name		Pair-name	
RVA	Register pair VA	RVA'	Register pair VA' (rear)
RBC	Register pair BC	RBC'	Register pair BC' (rear)
RDE	Register pair DE	RDE'	Register pair DE' (rear)
RHL	Register pair HL	RHL'	Register pair HL' (rear)
EA	Extension accumulator	EA'	Extension accumulator (rear)
SP	Stack pointer		
PC	Program counter		

IE-87AD 1-bit status flags

Flag-name	
Z	Zero flag
SK	Skip flag
HC	Half carry flag
L1	L1 flag
L0	L0 flag
CY	Carry flag

The processor register command is used to display and set the contents of the abovementioned registers. A REGISTER command can collectively output the contents of these registers to the console but cannot display the contents of the pair register.

To set the contents of the registers, use the following format:
processor-register=contents

REGISTER command

REGISTER
Example:
REGISTER

REGISTER Command keyword that collectively outputs the contents of the processor registers

Processor-register display command

register-name
pair-name
flag-name
Example:
RA
RBC
EA'
SP

register-name	Indicates the name of the IE-87AD 8-bit register.
pair-name	Indicates the name of the IE-87AD 16-bit register.
flag-name	Indicates the name of the IE-87AD flag.

Processor-register setting command

processor-register=contents
Example:
RA=00H
RBC=.ABC
CY=1
RA'=10H

processor-register	Indicates the name of the 8- or 16-bit name or the flag name.
contents	Numeric value, arithmetic expression, and symbol reference can be specified in this parameter.

(10) Mode-register commands

This command displays and sets the contents of the IE-87AD mode registers.
The following mode registers are available:

IE-87AD mode registers (See APPENDIX for IE-7809 Variation)

MA (W)	Mode A register
MB (W)	Mode B register
MCC (W)	Mode control C register
MC (W)	Mode C register
MM (W)	Memory mapping register
MF (W)	Mode F register
TMM (R/W)	Timer mode register
ETMM (W)	Timer/event counter mode register
EOM (R/W)	Timer/event counter output mode register
SML (W)	Serial mode register
SMH (R/W)	
ANM (R/W)	A/D channel mode register

The registers that can be displayed and set are TMM, EOM, SMH, and ANM. Other registers are used only for setting.

Mode-register display command

```
mode-register 1
Example:
TMM
EOM
```

mode-register 1	Indicates the mode registers of IE-87AD that can be displayed and set (TMM, EOM, SMH, and ANM).

Mode-register setting command

---

mode-register=contents

Example:

MA=00H

EOM=0FH

SML=00001000B

---

mode-register Indicates the name of the mode register of the IE-87AD.

MODE command

MODE

Example:

MODE

MODE Command keyword that displays all of the mode registers that can be displayed.

(11) PORT commands

This command displays and sets the status of each IE-87AD port. The IE-87AD has five 8-bit ports: PORTA, PORTB, PORTC, PORTD, and PORTF. The command can be used to collectively display these ports at one time. The command can also be set and reset for each bit. Inputting a bit number after the port name also enables the IE-87AD to display the port status in units of 8 bits.

PORT command

PORT

Example:

PORT

PORT Command keyword that displays all of the statuses of the 5 ports of the IE-87AD on the console.

PORT display command

---

port-name
port-name bit-number
Example:
PORTA
PORTF
PORTB 7
PORTB 4

---

port-name Indicates the IE-87AD port name.
bit-number Indicates one of the effective bit numbers 0 through 7 in a port.
If this parameter is omitted, the contents of the 8-bit port are displayed in hexadecimal numbers.
If this parameter is specified, only the status of that particular port is displayed.

Example:
*PORTA
PORTA=11

*PORTA 0
PORTA 0=0

PORT setting command

---

port-name=contents
port-name bit-number= |0|
|1|

Example:
PORTA=00H
PORTF=0FFH
PORTA 7=0
PORTB 0=1

---

port-name Indicates the IE-87AD port name
contents Numeric value, arithmetic expression, and symbol reference can be specified in this parameter.

bit-number Indicates one of the effective bit numbers 0 through 7 in a port.

(12) CAUSE command

This command displays the break condition that occurs when the emulation terminates.

The following six break conditions are available:

① Ordinary break condition
This condition occurs when either the address break condition or the data break condition is matched.

② Software break condition
This condition occurs when the ESC key is pressed during emulation.

③ Timer break condition
Break occurs after execution has continued for the time set in the timer break register.

④ External break condition
This condition occurs when the level of the external break register matches that of the hardware diagnostic probe.

⑤ Non-map break condition
This condition occurs when the unmapped memory is accessed.

⑥ Memory error break condition
This condition occurs when a memory bit error, etc., occurs.

When a break is applied, one of the following messages will be displayed.

Type of break	Message
① Normal break	NORMAL BREAK
② Software break	ESC BREAK
③ Timer break	TIMER BREAK
④ External break	EXTERNAL BREAK
⑤ Non-map break	NON MAP BREAK
⑥ Memory error Break	MEMORY ERROR BREAK*

NOTE: * Normally, the memory error break message is not displayed. As this message indicates that a failure has occurred in the internal memory of the IE-87AD, contant your NEC dealer if this message is displayed.

CAUSE command

CAUSE

Example:

CAUSE

CAUSE Command keyword that displays the break cause.

(13) Mask register commands

This command displays and sets 87AD mask register MKL or MKH. The MKH and MKL are both represented by binary 8-bit codes.

MK command

MK

Example:

MK

MK Command keyword that displays the MKL or MKH Mask register display command.

Mask register display command

---

mask-name

Example:

MKH

MKL

---

mask-name Indicates mask register name MKH or MKL.

Example:

*MK↵

MKL=10011000B MKH=00000100B

*MKL↵

MKL=10011000B

Mask register setting command

---

mask-name=contents

Example:

MKH=00H

MKL=10000001B

---

mask-name Indicates mask register name MKH or MHL.

contents Numeric value, arithmetic expression, and symbol reference can be specified in this parameter.

(14) Timer register command

This command sets IE-87AD timer register TM0 or TM1. Neither the TM0 nor the TM1 can be displayed.

Timer register setting command

```
|TM0| =contents
|TM1|
Example:
TM0=0FFH
TM0=10
TM1=1110B
```

TM0 Command keyword that sets timer register 0.

TM1 Command keyword that sets timer register 1.

contents Numeric value, arithmetic expression, and symbols reference can be specified in this parameter.

(15) Timer/event counter register command (See APPENDIX f. IE-7809-I)

This command sets IE-87AD timer event counter register ETM0 or ETM1. Neither the ETM0 nor ETM1 can be displayed.

Timer/event counter register setting command

```
|ETM0| =contents
|ETM1|
Example:
ETM0=00H
ETM0=.ABC
ETM1=10000000B
```

ETM0 Command keyword that sets timer/event counter register 0.

ETM1 Command keyword that sets timer/event counter register 1.

contents Numeric value, arithmetic expression, and symbol reference can be specified in this parameter.

(16) CR register command (IE-87AD-I only)

This command displays the contents of one or all of the IE-87AD conversion result registers CR0 through CR3. This command, however, cannot be used to set these registers.

CR display command

CR
cr-name
Example:
CR
CR0
CR1

CR Command keyword that displays all of the contents of CR0 through CR9.

cr-name The followlng four CR register names can be specified: CR0, CR1, CR2, and CR3.

(17) SERIAL commands

This command displays the contents of the IE-87AD serial receive buffer register RXB or sets the contents of the IE-87AD serial transmit buffer register TXD.

SERIAL display command

SERIAL
Example:
SERIAL

SERIAL Command keyword that displays the contents of receive buffer register RXB.

SERIAL setting command

SERIAL=contents

Example:

SERIAL=10H

SERIAL=10001000B

SERIAL Command keyword that sets the contents of transmit buffer register TXB.

(18) Memory commands

These commands display and change the contents of the mapped memory.

There are two types of memory commands: memory display and memory change. These commands are further classified into a BYTE command which displays and the changes the contents per byte and a WORD command which displays or changes the contents for each 2 bytes.

In the WORD Format, in contrast to the memory image, the memory contents are displayed or modified starting from the highest-order byte.

- Display command

  A BYTE display command displays the memory contents of the specified address range per byte. A WORD display command displays the memory contents each 2 bytes.

- Change command

  (a) Memory change with the specified address range

  If BYTE partition=1,2,3 is specified, the Change command serially changes the memory contents to 1, 2, and 3 in that order starting from the beginning of the specified address range. The command changes the unchanged addresses in the partition to the last data 3.

  (b) Memory change with the specified first address

  If BYTE s.adr=1,2,3 is specified, the command changes the memory contents to 1, 2, and 3 in that order starting from the start address specified by the s.adr parameter.

Memory display command

```
BYTE|s.adr TO e.adr   |
WORD|s.adr LENGTH size|
```

Example:

```
BYTE 0 TO 0FFH
WORD .ABC TO .ABC+10
BYTE 0 LENGTH 10
WORD .ABC LENGTH 10
```

BYTE — Command keyword that displays the memory contents per byte.

WORD — Command keyword that displays the memory contents each 2 bytes. (In contrast to the memory image, the memory contents are sequentially displayed from the highest-order byte.)

s.edr — Indicates the start address.

e.adr — Indicates the end address.

size — Indicates the memory size.

TO — Keyword indicating that the end address follows this keyword.

LENGTH — Keyword indicating that the memory size follows this keyword

NOTE: Numeric value, arithmetic expression, and symbol reference can be specified in the s.adr, e.adr, and size parameters.

Memory modifying command

```
BYTE|s.adr TO e.adr   |=contents ,contents ...
WORD|s.adr LENGTH size|
BYTE s.adr=contents ,contents,...
WORD
```

Example:

```
BYTE 0 to 0FFH=0          ①
BYTE 0 LENGTH 10=1,2,3    ②
WORD .ABC=1,2,3           ③
WORD .ABC TO .ABC+3=0     ④
```

BYTE	Command keyword that modifies the memory contents per byte.
WORD	Command keyword that changes the memory contents each 2 bytes. (In contrast to the memory image, the memory contents are sequentially changed from the highest-order byte.)
s.adr	Indicates the start address.
e.adr	Indicates the end address.
size	Indicates the memory size.
TO	Keyword indicating that the end address follows this keyword.
LENGTH	Keyword indicating that the memory size follows the this keyword.
contents	Indicates the data to be modified. If two or more data are changed, use a comma (,) between each data.

① The memory modifying command modifies address 0H to address 0FFH to 0.

② The command sets 1 in address 0, 2 in address 1, and 3 in address 3 to address 10.

③ The command sequentially sets 1, 2, and 3 in the addresses with the symbol ABC.

④ The command sets 0 in four addresses with the symbol ABC.

(19) Symbol commands

The IE-87AD loads a symbol file for debugging. The format of the symbol file is the same as that explained in the LOAD command. The symbol file includes local and public symbols. The IE-87AD loads this symbol file from the disk into the MDS memory and performs symbol debugging.

The following four symbol commands are available:

- Symbol reference command
  This command displays the specified hexadecimal symbol address.
- Symbol modifying command
  This command modifies the defined symbol address.

- Symbol defining command
  This command defines a new symbol.
- Symbol deleting command
  This command deletes the defined symbol.

Note that the symbol file must be loaded before these commands are executed.

SYMBOL reference command

SYMBOL
..mod-name.sym-name
/p-sym-name
Example:
SYMBOL
..MOD1.START
/SWSET

SYMBOL	Keyword command that displays all of the symbols.
..	Indicates that the next alphanumeric string is a module name.
.	Indicates that the next alphanumeric string is a local symbol name.
mod-name	Indicates the local module name.
sym-name	Indicates the local symbol name.
/	Indicates that the next alphanumeric string is a public symbol name.
p-sym-name	Indicates the public symbol name.

Symbol modifying command

..mod-name.sym-name=contents
/p-sym-name=contents
Example:
..MOD1.STRAT=1000H
/SWSET=.SWRST+80H

..	Indicates that the next alphanumeric string is a module name.
.	Indicates that the next alphanumeric string is a local symbol name.

mod-name	Indicates the local module name.
sym-name	Indicates the local symbol name.
/	Indicates that the next alphanumeric string is a public symbol name.
p-sym-name	Indicates the public symbol name.
=	Indicates the symbol change command.
contents	Indicates the address to be modified. Numeric value, arithmetic expression, and public symbol reference can be specified in this parameter.

Symbol defining command

DEFINE | ..mod-name.sym-name | =contents
       | /p-sym-name |

Example:

```
DEFINE ..MOD1.STOP=4000H
DEFINE/ABC=.DEF+10H
```

DEFINE	Command keyword indicating that the command is a symbol defining command.
..	Indicates that the next alphanumeric string is a local module name.
.	Indicates that the next alphanumeric string is a local symbol name.
mod-name	Indicates the local module name.
sym-name	Indicates the local symbol name.
/	Indicates that the next alphanumeric string is a public symbol name.
p-sym-name	Indicates the public symbol name.
=	Indicates that the symbol address to be defined follows the equal sign.
contents	Indicates the symbol address to be defined. Numeric value, arithmetic expression, and public symbol reference can be specified in this parameter.

Symbol deleting command

```
REMOVE |SYMBOL                ||[ |            |     ]
       |..mod-name.sym-name   ||, |            |,....]
       |/p-sym-name           ||[ |            |     ]
```

Example:

REMOVE SYMBOL

REMOVE ..MOD1.START

REMOVE /ABC,..MOD1.STOP

REMOVE	Command keyword indicating that the command is a symbol deleting command.
SYMBOL	Command keyword that deletes all of the symbols.
..	Indicates that the next alphanumeric string is a local module name.
.	Indicates that the next alphanumeric string is a local symbol name.
mod-name	Indicates the local module name.
sym-name	Indicates the local symbol name.
/	Indicates that the next alphanumeric string is a public symbol name.
p-sym-name	Indicates the public symbol name.
,	Indicates that the symbols to be deleted still continue.

(20) Break register commands

The IE-87AD can use six break registers as the break functions. Only the registers specified in the break mode of a GO command can be used. A break occurs when the break register condition matches the condition of the break register specified in the effective break mode.

The following IE-87AD break registers are available:

- Address break register 0 (BA0)

  The address to be broken is set in this register as one continuous area. The availability of the Read or Write signal can also be set in this register.

BA1 Address break register 1
The address to be broken is set in this register as one continuous area. The availability of the Read or Write signal can also be set in this register.

BDR Data break register (BDR)
The status of the data bus to be broken is set in this register. The availability of the Read or Write signal or an address can also be set in this register. In other words, a break condition that occurs when an address is accessed, read, or written can be set.

BEX Name of the external sensing clip register 1.
Sets the conditions for the external sensing clip which you wish to use to perform break.

BTMR Timer break register
Contains the length of time for which emulation is to be performed.

BLOOP Loop count register
This register is used to keep track of the number of times that a break condition has been satisfied. Break is performed when the break condition has been met the specified number of times. However, break will not be performed if only this register is set.

The following commands are used to control the break registers.

- BR command
  This command displays the contents of all six break registers.
- Address break register display command
- Data break register display command
- External sensing clip register display command

- Timer break register display command
- Loop break register display command
- Address break register setting command
- Data break register setting command
- External sensing clip register setting command
- Timer break register setting command
- Loop break register setting command

The display commands cause the contents of the respective break registers to be displayed as a hexadecimal value.
Example:

```
BA0=0000-0100 W
BDR=60(0000-1000) R
```

"W" in the above example denotes the write signal and "R" the read signal. The numbers in parentheses in the display of the BDR contents indicate location addresses. The initial values of each of the break registers is as shown below.

- BA0 : 0
- BA1 : 0
- BDR : Nothing is set.
- BEX : 0
- BTMR : 0
- BLOOP: 1

BR command

BR

Example:

BR

BR Command keyword that displays all of the break registers.
This command displays the break registers as follows:

(Initial status)
BA0=0000
BA1=0000
BDR=XX
BEX=00
BTMR=0000
BLOOP=01

(Set status)
BA0=1100
BA1=2000-200F
BDR=60 (0000-000F) R
BEX=0F
BTMR=000A
BLOOP=09

Address break register display command

BA0 BA1

Example:
BA0
BA1

EA0 Indicates the name of address break register 0.
BA1 Indicates the name of address break register 1.
The contents of these registers are displayed in hexadecimal numbers.
When the Read signal is attached, "R" is displayed.
When the Write signal is attached, "W" is displayed.

Display example:
BA0=0010
BA0=0010-001F
BA1=1010 W
BA1=1020-1030 R

Data break register display command

BDR

Example:

BDR

BDR Indicates the name of the data break register.
When the Read signal is attached, "R" is displayed.
When the Write signal is attached, "W" is displayed.
The attached address is displayed in parentheses.

Display example:

BDR=XX
BDR=30
BDR=60 R
BDR=48 (0030) R
BDR=70 (0100-010F)R
BDR=XX. R
BDR=XX (1000-10FF)
(FIGURE)

External sensing clip register display command

BEX

Example:

BEX

BEX Indicates the name of the external sensing clip register. The contents of this register are displayed in hexadecimal numbers.

Display example:

BEX=0F
BEX=10

Timer break register display command

BTMR

Example:

BTMR

BTMR Indicates the name of the timer break register. The contents of this register are displayed in hexadecimal numbers.

Display example:
BTMR=000A
BTMR=00FF

Loop break register display command

BLOOP
Example:
BLOOP

BLOOP Name of the count break register. The contents of this register are displayed in hexadecimal numbers.

Display example:
BLOOP=01
BLOOP=0A

Address break register set command

BA0 / BA1 = partition [WRITTEN / READ]

Example:
BA0=1000H ①
BA1=1000H TO 2000H READ ②
BA0=1010H LENGTH 10H ③
BA1=2000H WRITTEN ④

BA0 Indicates the name of address break register 0.
BA1 Indicates the name of address break register 1.
= Indicates the address break register setting.

partition	Indicates the address range. The format of this parameter is [s.adr] TO e.adr or s.adr [LENGTH size].
WRITTEN	Keyword used to attach the Write signal to the break address condition.
READ	Keyword used to attach the Read signal to the break address condition.

The conditions in the above example are as follows.

① A break occurs when address 1000H is accessed.
② A break occurs when addresses 1000H to 2000H are read.
③ A break occurs when addresses 1010H to 101FH are accessed.
④ A break occurs when address 2000H is written.

Data break register setting command

```
BDR=[contents]  WRITTEN   WITH partition
                READ
Example:
BDR=60H                                    ①
BDR=60H READ                               ②
BDR=60H READ WITH 1000H                    ③
BDR=WRITTEN                                ④
BDR=READ WITH 1000H LENGTH 2000H           ⑤
BDR=WITH 1000H to 10FFH                    ⑥
```

BDR	Indicates the data break register name.
=	Indicates the data break register setting.
contents	Indicates the data bus condition to be set. If this parameter is omitted, no data bus condition is set.
WRITTEN	Keyword used to attach the Write signal to the data break condition.
READ	Keyword used to attach the Read signal to the data break condition.
WITH	Keyword used to attach an address condition to the data break condition.

partition Indicates the address range. The format of this parameter is s'adr [TO e.adr] or s.adr [LENGTH size].

The conditions in the above example are as follows:

① A break occurs when data 60H is accessed.
② A break occurs when data 00H is read.
③ A break occurs when data 60H is accessed at address 1000H.
④ A break occurs when data is written.
⑤ A break occurs when data read betweem addresses 1000H and 2FFFH.
⑥ A break occurs when the memory is accessed between addressed 1000H and 10FFH.

External :sensing clip register setting command

BEX=contents

Example:

BEX=0FH

BEX=10001010B

BEX Indicates the external sensing clip register name.

= Indicates the external sense clip register setting.

contents Indicates binary, octal, decinmal, or hexadecimal data to be set.

Timer break register seittng command

BTMR=contents

Example:

BTMR=10 ①

BTMR=100H ②

BTMR Indicates the timer break register name.

= Indicates the timer break register setting.

contents Indicates binary, octal, decimal, or hexdecimal data to be set.

① A break occurs 10ms after execution.
② A break occurs 100Hms after execution.

Loop break register setting command

```
BLOOP=contents
Example
BLOOP=10          (1)
BLOOP=0FFH        (2)
```

BLOOP	Name of the count break register.
=	Indicates that the contents of the loop break register are to be set
contents	Indicates the data to be set Input of this data can be performed in binary octal decimal or hexadecimal numbers

(1) Break will be performed when the break condition has been satisfied 10 times.
(2) Break will be performed when the break condition has been satisfied FFH times.

(21) GO command

This command instructs the IE-87AD to start the emulation from the address indicated by the program counter (PC).
When the emulation starts the message "EMULATION BEGUN" is output. The emulation continues until one break condition matches another, an irrecoverable error occurs or the ESC key is pressed.
When the emulation terminates the message "EMULATION TERMINATED #XXXX" is output. #XXXX indicates the PC.
The format of the GO command is:
GO [FROM s.adr][TILL break condition]
FROM s.adr is used to specify the start address.
TILL is used to set a break condition.
The break condition is set by a break register or the GO command.. The following five break conditions are available:

① Address break condition

This break is applied when the address break condition is satisfied.

Format

- Address break condition set by the GO command

  TILL[LOCATION=]partition[partition] [WRITTEN | READ]

  partition (s.adr[TO e.adr] | s.adr[LENGTH size])

  "partition" specifies the break range. Two partitions can be specified using a comma (,). READ is specified to attach the Read signal to the break condition. "WRITTEN" is specified to attach the Write signal to the break condition.

- Address break condition set by a break register

  TILL BA0

  TILL BA1

  TILL BA0 BA1

  An address break occurs when the contents of address break registers BA0 and BA1 match those of the address.

② Data break condition

Three factors of the data break condition are:

1) Data bus status
2) The availability of the Read and Write signals
3) Address range specification (A break occurs if the memory within this address range is accessed.)

Each one of 1) through 3) can be set. Two or more factors can also be set at the same time by using the AND condition.

For example the following conditions can be set together;

- To break when a specified data is read.
- To break when a location with a specified address range is accessed.

- To break when a specified data at a specified address is read.

Format

- Data break condition set by the GO command

  TILL DATA=[contents] [WRITTEN / READ] [WITH partition]

  contents Sets the data bus condition.
  WRITTEN Sets the availability of the Write
  READ and Read signals.
  partition Sets the address range.

- Data break condition set by a break register

  TILL DATA

  A data break occurs using the contents of data break register BDR. The contents of BDR are retained after execution.

③ External break condition

An external break occurs when the external sense clip status matches the external sense clip break condition.

The external break condition is set using the following methods:

- External break condition set by the GO command

  TILL EXTERNAL=contents

  contents: Sets the break condition.

- External break condition set by the break register

  TILL EXTERNAL

  Inputting "EXTERNAL" enables the external break to occur when the external sense clip status matches the external sense clip register (BEX) condition. The BEX contents remain stored even after execution.

④ Timer break condition

A timer break occurs only after the emulation is performed within the specified time period (1 ms to 65535 ms).

- Timer break condition set by the GO command
  TILL TIMER=contents
  A timer break occurs after the emulation is performed within the time (ms) period specified by "contents".
- Timer break condition set by a break register
  TILL TIMER
  Inputting "TIMER" enables the timer break to occur after the emulation is performed in the time set

⑤ Loop break condition

A loop break occurs when the break condition occurs the specified number of times. This value can be set in the range 1 to 255.

- Count break condition set by the GO command
  TILL LOOP=contents
  Break occurs when the break condition is satisfied the number of times specified by "contents".
- Count break condition set by the break register
  TILL LOOP
  By inputting LOOP, break will occur when the break condition is satisfied the number of times set in the loop counter BLOOP. The contents of BLOOP are retained after execution of the break.
  No data can be input from the keyboard during the emulation and only the ESC key can be used as a forcible break key.
  When the "ESC" key is depressed, the emulation is cancelled, causing the current state to return to the IE-87AD command input wait state.

GO command

```
GO |[FROM s.adr][TILL clause   [clause...]]|
   |[FOREVER]                               |
```

clause: 1 Address break
- [LOCATION]partition[,partition] WRITTEN / READ
- BA0
- BA1
- BA0 OR BA1

2 Data break
- DATA= contents [WRITTEN / READ] [WITH partition]
- DATA

3 External break
- EXTERNAL=contents
- EXTERNAL

4 Timer break
- TIMER=contents
- TIMER

5 Count break
- LOOP=contents
- LOOP

Example:

```
* GO                                              (1)
* GO FROM 1000H TILL 30H, 40H                     (2)
* GO TILL BA0    BA1                              (3)
* GO FROM .ABC TILL DATA                          (4)
* GO TILL DATA=30H READ WITH 0 TO 0FFH            (5)
* GO TILL EXTERNAL=80H                            (6)
* GO TILL EXTERNAL                                (7)
* GO TILL BA0    EXTERNAL                         (8)
* GO TILL DATA=WITH 0 TO 100H    TIMER=100        (9)
* GO TILL LOOP=3 DATA=60H READ                    (10)
* GO TILL DATA=WRITTEN    COUNT                   (11)
* GO TILL DATA=40                                 (12)
* GO TILL 100H                                    (13)
* GO TILL 100H    COUNT=3    TIMER                (14)
```

GO	Command keyword indicating the GO command
FROM	Keyword used to specify the emulation start address.
s.adr	Indicates the emulation start address. When this parameter is specified, the IE-87AD sets this address in the program counter (PC) and starts emulation.
TILL:	Keyword used to set a break condition.
clause	Indicates the break condition. Clauses 1 and 2 cannot be ORed.
FOREVER	Keyword used when no break condition is set. This keyword can be omitted.
LOCATION	Keyword used to set the address break condition. This keyword can be omitted.
partition	Specifies the address range. The format of this parameter is s.adr [TO e.adr] or s.adr [LENGTH size].
WRITTEN	Keyword used to attach the Write signal to the break condition.
READ	Keyword used to attach the Read signal to the break condition.
BA0	Address break register 0.
BA1	Address break register 1.
DATA	Keyword used to set the data break condition.
contents	Indicates the break condition.
WITH	Keyword used to specify the address range for the data break condition.
EXTERNAL	Keyword used to set the external break condition. If followed by "=contents", the value of the contents parameter will be used to set the external break condition. If only EXTERNAL is specified, the contents of the BEX register will be used for this purpose.
TIMER	Keyword used to set the timer break condition. If followed by "=contents", the value of the contents parameter will be

used to set the timer break condition. If only TIMER is specified, the contents of the BTMR register will be used for this purpose.

LOOP Keyword used to set the loop break condition. If followed by "=contents", the value of the contents parameter will be used to set the loop break condition. If only LOOP is specified, the contents of the BLOOP register will be used for this purpose.

Examples ① through ⑭ above signify the following:

① As in the case of the GO FOREVER command, no break will be performed. Forced break is applied when the ESC key is pressed.

② Emulation starts at address 1000H and break occurs when the address 30H or 40H is passed.

③ A break occurs when the condition stored in either registers BA0 or BA1 is satisfied.

④ Emulation starts at the address defined by symbol ABC. Break occurs when the condition stored in register BDR is satisfied.

⑤ Break occurs when data 30H is read at an address in the range 0H to 0FFH.

⑥ Break occurs when the status of the external sensing clips becomes 80H.

⑦ Break occurs when the status of the external sensing clips matches the values set in external sensing clip register BEX.

⑧ Break occurs when the value set in address break register 0 (BA0) or external sensing clip register (BEX) is matched.

⑨ Break occurs when the memory range 0H to 100H is accessed or when emulation has been performed for 100ms.

⑩ Break occurs when data 60H has been read three times.

⑪ Break occurs when memory write has been performed the number of times set in loop break register BLOOP.
⑫ Break occurs when data 40H is accessed.
⑬ Break occurs when address 100H is passed.
⑭ A break occurs either when address 100H has been passed three times or when emulation has been performed for the period of time (ms) set in timer break register ETMR.

(22) STEP command

This command performs emulation for one instruction only. All break conditions are invalidated when the STEP command is used. In other words, this command displays trace data and contents of the registers for each execution step. Note that trace data is disassembled before it is displayed. The display of trace data can also be cancelled.

```
STEP[FROM s.adr][COUNT contents]
STEP ENABLE
     DISABLE
Example:
STEP |ENABLE |
STEP |DISABLE|
STEP
STEP FROM 1000H
STEP FROM .ABC COUNT 3
```

STEP	Command keyword indicating that the command is a STEP command.
FROM	Keyword used to specify the start address.
s.adr	Indicates the start address.
COUNT	Keyword used to specify the STEP command execution count.
contents	Indicates the STEP command execution count.
ENABLE	Keyword used to display trace data.
DISABLE	Keyword used to cancel the trace data display.

(23) Trace control command

The IE-87AD is provided with a function which allows the result of the emulation to be displayed and retained. The trace control command specifies the data to be traced as well as the format for its display.

Trace data includes:

1. Address
2. Data
3. Port B or external sensing probe
4. Port A
5. Fetch cycle signal
6. Write signal
7. Read signal

These data are stored per machine cycle. The IE-87AD can store a maximum of 1,023 machine cycles of trace data in the trace RAM. If trace is performed for more than 1,023 machine cycles, the older trace data will be discarded and only the most recent trace data will be retained. Either Port B or the external sensing probe may be traced; trace of both these data is not permitted and this selection is made using the trace elect command.

The trace RAM is provided with a pointer which is moved to indicate which of the trace data is to be displayed. The OLDEST command moves the trace pointer to the location of the oldest trace data.

The NEWEST command moves the trace pointer to the location of the newest trace data.

The MOVE command moves the pointer from the current pointer position by the specified count. If a negative number is specified, the pointer is decremented and moves in the direction of the old traced data. If a positive number is specified, the pointer is incremented and moves in the direction of new traced data.

Traced data is displayed in the instruction mode or the cycle mode. A trace display format specification command is used to specify these modes.

The formats of the instruction and cycle modes are as follows:

- Instruction mode

```
ADRS                    INSTRUCTION   PA BP
100 4C C0       START1:MOV,A,PA       01 00
1002 60 8A              ANA A,B       01 00
1004 40 20 10           CALL TEST     01 00
```

- Cycle mode

ADRS	DATA	W	R	M1	PA	PB
1000	4C	1	0	1	01	00
1001	C0	1	0	0	01	00
1002	60	1	0	1	01	00
1003	8A	1	0	0	01	00
1004	40	1	0	1	01	00
1005	20	1	0	0	01	00
1006	10	1	0	0	01	00

ADRS	Displays address bus information.
INSTRUCTION	Diaassembles and displays instruction data.
PA	Displays Port A information.
PB	Displays Port B information.
DATA	Displays data bus information.
W	Displays the Write signal.
R	Displays the Read signal.
M1	Displays the fetch cycle.

The PRINT command displays traced data on the console by the specified count. The display format of the PRINT command is the same as that of the trace display format specification command.
If a negative number is specified, the trace pointer moves in the negative direction (in the direction of old data) by the specified count and displays traced data from that pointer position by the specified count. As a result, the pointer does not move at all.

If a positive number is specified, the command displays traced data in the positive direction from the current pointer position (in the direction of new data) by the specified count. As a result, the pointer is incremented by the specified count.
The TRACE ENABLE command specifies the trace conditions. The following five trace conditions can be specified by this command.

① Address condition
In this condition, the address range to be traced can be specified. If the address range is specified, 0 through FFFFH are to be traced. Data can be traced with the address range when the Read or Write signals are specified.

② Data condition
Data is traced only when the data bus status matches the specified status.
Data can also be traced when the Read or Write signal is specified.

③ Port B condition
Data is traced only when the Port B status matches the specified status.

④ Port A condition
Data is traced only when the Port A status matches the specified status.

⑤ Fetch cycle condition
Data can be traced only in the fetch cycle.

Two or more trace conditions ① through ⑤ above can be combined using the AND condition. For example, data can be traced when Port B is in a certain status, Port A is also in a certain status, and the cycle is in fetched.

The TRACE DISABLE command clears all of the trace conditions. The initial conditions of the trace control command must satisfy the following:

- Port B is traced but the external sense clip is not traced.
- Instruction mode is specified as the trace display mode.
- The TRACE DISABLE command is executed and no data is traced.

NOTE: When trace is performed using the data or fetch cycle conditions, correct trace results may not be obtained if the trace data is displayed by intruction type. Therefore, display such data by machine cycle type.

TRACE select command

```
TRACE |PORTB   |
      |EXTERNAL|
```

Example:

```
TRACE PORTB
TRACE EXTERNAL
```

TRACE	Command keyword indicating the trace select command.
PORTB	Command keyword that specifies the tracing of ports.
EXTERNAL	Command keyword that specifies the tracing of the external sense clip.

Trace pointer move command

```
MOVE[+/-]contents
OLDEST
NEWEST
```

Example:

```
MOVE 10
MOVE -30
OLDEST
NEWEST
```

MOVE	Command keyword that causes the trace pointer to move.
contents	Numeric value used to cause the pointer to move.
+	Keyword that causes the pointer to move in the direction of new traced data.
-	Keyword that moves the pointer in the direction of old traced data.
OLDEST	Command keyword that moves the pointer to the oldest traced data position.
NEWEST	Command keyword that moves the pointer to the newest traced data position.

NOTE: The trace pointer can only move within the trace range (OLDEST-NEWEST).

TRACE display format specification command

TRACE= |CYCLE |
|INSTRUCTION|

Example:

```
TRACE=CYCLE
TRACE=INSTRUCTION
```

TRACE	Command keyword that specifies the trace display format.
CYCLE	Keyword that displays traced data in the machine cycle mode.
INSTRUCTION	Keyword that displays traced data in the instruction mode.

PRINT command

```
PRINT ALL
PRINT[+/-]contents
```

Example:

```
PRINT ALL
PRINT 10
PRINT -20
```

PRINT Command keyword that displays traced data

ALL Displays all of the traced data

\+ Displays traced data in the direction of new data from the current trace pointer position.

\- Displays traced data in the direction of old data from the current trace pointer position.

TRACE ENABLE command

TRACE ENABLE[clause[clause...]]

clause:

① [LOCATION=]partition [READ / WRITTEN]

② DATA=contents [READ / WRITTEN]

③ PORTB=contents

④ PORTA=contents

⑤ FETCHED

Example:

```
TRACE ENABLE 0 TO 0FFH.
TRACE ENABLE DATA=60H READ
TRACE ENABLE PORTB=30H      FETCHED
TRACE ENABLE 0 TO 0FFH      DATA=60H AND FETCHED
TRACE ENABLE PORTA=30H      0 TO 0FFH WRITTEN
TRACE ENABLE FETCHED
```

TRACE Command keyword indicating the trace control command.

ENABLE Command keyword indicating the trace enable command.

clause Syntax that specifies the trace condition. Six trace conditions can be specified.

LOCATION Keyword that specifies the address to be traced. (This keyword can be omitted.)

READ Keyword that traces the Read signal.

WRITTEN Keyword that traces the Write signal.

partition Specifies the address trace range. The format of this parameter is "start address TO end address" or "start address LENGTH size".

contents Specifies the trace status.

DATA Keyword that specifies the data bus condition.

PORTB Keyword that specifies the Port B condition.

PORTA Keyword that specifies the Port A condition.

FETCHED Keyword that specifies the fetch cycle condition.

NOTES: 1. If no address range is specified by this command, the default value is addresses 0 through FFFFH. (In other words, inputting TRACE ENABLE alone will trace all of the addresses.)

2. The range of addresses to be traced as specified by this command will remain effective until a new TRACE ENABLE command is input. To specify more than one range of addresses for trace, input two consecutive TRACE ENABLE commands.
Example: To trace addresses 100H to 200H and 300H to 400H, input as shown below.

```
*TRACE ENABLE 100H TO 200H
*TRACE ENABLE 300H TO 400H
```

Trace conditions other than the address condition (for Port A, Port B, data or fetch cycle) will be as specified by the last TRACE ENABLE command. In the above example, none of these conditions has been specified and they will therefore all be invalid.

TRACE DISABLE command

TRACE DISABLE

Example:

TRACE DISABLE

TRACE Command keyword indicating the trace control command

DISABLE Command keyword indicating the trace disable command

(24) RESET CPU command

This command resets the IE-87AD emulation chip.

RESET CPU

Example:

RESET CPU

RESET Command keyword indicating the RESET command

CPU Command keyword indicating the RESET CPU command

(25) RESET HARDWARE command

This command resets the IE-87AD. When this command is accepted, the setting of the IE-87AD initialization starts. For details on how to input this command, see "IE87AD command".

RESET HARDWARE

Example:

RESET HARDWARE

RESET Command keyword indicating the RESET command

HARDWARE Keyword indicating the RESET HARDWARE command

(26) On-line assembler command

The IE-87AD has a mnemonic on-line assembler command that modifies the memory contents. The following describes how to use this command:

① First, input the assembler command in the ASM F20 FROM s.adr format to place the IE-87AD in the on-line assembler mode.

② The IE-87AD disassembles the contents from the address specified by the s.adr parameter by one instruction and displays them. The IE-87AD then displays "=" and enters the user key input wait state.

③ When a carriage return is input, the next instruction is displayed.

④ If a mnemonic code is input in assembler notation instead of the carriage return, the IE-87AD assembles it, outputs the results, and enters the user key input wait state again.

⑤ Input a carriage return to modify the memory contents using the assembled results. Input a space to invalidate the results. If a space is input, the IE-87AD disassembles the same instruction and enters the user key input wait state.

⑥ To terminate the online assembler, input "END". Once "END" is input, the IE-87AD enters the key input wait state.
The EQU, ORG, DB, and DW pseudo instructions can be used for the on-line assembler. The input format of the on-line assembler source must satisfy the following:

- The symbol consists of a maximum of 6 alphanumeric characters.
- The symbol of the label field is newly defined as the public symbol.
- The symbol in the operand field is predefined.
- Those input after a semicolon (;) are assumed to be comments.

On-line assember command

```
ASM FROM s.adr
Example:
ASM FROM 4000H
ASM FROM .ABC
```

ASM Command keyword that executes the on-line assembler command.

FROM Keyword indicating that the start address follows this keyword.

s.adr Indicates the start address.

(27) Disassemble command

This command disassembles and displays the memory contents in the specified address range.
If the symbol file is already loaded, this command also displays the symbols.

```
DASM |s.adr TO e.adr    |
     |s.adr LENGTH size |
Example:
DASSM 0 TO 0FH
DASM .ABC LENGTH 10
```

DASM Command keyword indicating the disassemble command.

s.adr Indicates the start address.

e.adr Indicates the end address.

TO Keyword indicating that the end adress follows this keyword.

LENGTH Keyword indicating the memory size follows this keyword.

size Indicates the memory size.

CHAPTER 8 ELECTRICAL CHARACTERISTICS

μPD7811

ABSOLUTE MAXIMUM (Ta = 25°C)

Parameter	Symbol	Test Condition	Limit	Unit
Power Supply Voltage	V_{CC}		-0.5 to +7.0	V
	V_{DD}		-0.5 to +7.0	V
	AV_{CC}		-0.5 to +7.0	V
	AV_{SS}		-0.5 to +0.5	V
Input Voltage	V_I		-0.5 to +7.0	V
Output Voltage	V_O		-0.5 to +7.0	V
Output Current Low	I_{OL}	All Output Pin	4.0	mA
		All Output Pin Total	100	mA
Output Current High	I_{OH}	All Output Pin	-0.5	mA
		All Output Pin Total	-20	mA
Reference Input Voltage	V_{AREF}		-0.5 to V_{CC}	V
Operating Temperature	T_{OPT}	$10MHz < f_{XTAL} \leqq 12MHz$	-10 to +70	°C
		$f_{XTAL} \leqq 10MHz$	-40 to +85	°C
Storage Temperature	T_{STG}		-65 to +150	°C

OPERATING CONDITION

Parameter / Osc. Freq.	Ta	Vcc, AVcc
$10MHz < f_{XTAL} \leqq 12MHz$	-10°C to +70°C	+5.0V ± 5%
$f_{XTAL} \leqq 10MHz$	-40°C to 85°C	+5.0V ± 10%

CAPACITANCE (Ta = 25°C, $V_{CC} = V_{DD} = V_{SS} = 0V$)

Parameter	Symbol	Test Condition	MIN	TYP	MAX	UNIT
Input Capacitance	C_I	f_c = 1MHz Unmeasured Pin Returned to 0V			10	pF
Output Capacitance	C_O				20	pF
I/O Capacitance	C_{IO}				20	pF

DC CHARACTERISTICS (Ta= -10°C to +70°C, V_{CC}= +5.0V ± 5%, V_{SS}= 0V, Vcc-0.8V≦V_{DD}≦Vcc)

Parameter	Symbol	Test Condition	MIN	TYP	MAX	UNIT
Input Low Voltage	V_{IL}		0		0.8	V
Input High Voltage	V_{IH1}	All except $\overline{SCK}$, $\overline{RESET}$ and X1	2.0		Vcc	V
	V_{IH2}	$\overline{SCK}$, X1	0.8Vcc		Vcc	V
	V_{IH3}	$\overline{RESET}$	0.8V_{DD}		Vcc	V
Output Low Voltage	V_{OL}	I_{OL} = 2.0mA			0.45	V
Output High Voltage	V_{OH}	I_{OH} = -200uA	2.4			V
Input Current	I_I	INT1, TI(PC3); +0.45V≦V_{IN}<Vcc			±200	uA
Input Leakage Current	I_{LI}	All except INT1, TI(PC3) 0V≦V_{IN}≦Vcc			±10	uA
Output Leakage Current	I_{LO}	+0.45V≦V_o≦Vcc			±10	uA
AVcc Supply Current	AI_{CC}			6	12	mA
V_{DD} Supply Current	I_{DD}	Ta = -40°C to +85°C		1.5*1	3.5	mA
Vcc Supply Current	Icc	Ta = -40°C to +85°C		110*1	220	mA
Data Retention Voltage	V_{DDDR}		3.7			V

*1 : Ta = 25°C, Vcc = V_{DD} = +5.0V

AC CHARACTERISTICS (Ta =-10°C to+70°C, Vcc =+5.0V±5%, Vss =0V, Vcc-0.3≦V_{DD}≦Vcc
READ/WRITE OPERATION

Parameter	Symbol	Test Condition	MIN	MAX	Unit
X1 Input Cycle Time	t_{CYC}		83	250	ns
Address Septup to ALE↓	t_{AL}		65		ns
Address Hold after ALE↓	t_{LA}		50		ns
Address to $\overline{RD}$↓ Delay Time	t_{AR}		150		ns
$\overline{RD}$↓ to Address Floating	t_{AFR}			20	ns
Address to Data Input	t_{AD}			360	ns
ALE↓ to Data Input	t_{LDR}			215	ns
$\overline{RD}$↓ to Data Input	t_{RD}			180	ns
ALE↓ to $\overline{RD}$↓ Delay Time	t_{LR}		35		ns
Data Hold Time to $\overline{RD}$↑	t_{RDH}		0		ns
$\overline{RD}$↑ to ALE↑ Delay Time	t_{RL}		115		ns
$\overline{RD}$ Width Low	t_{RR}	Data Read	280		ns
		OP Code Fetch	530		ns
ALE Width High	t_{LL}		125		ns
Address to $\overline{WR}$↓ Delay	t_{AW}		150		ns
ALE↓ to Data Output	t_{LDW}			195	ns
$\overline{WR}$↓ to Data Ouput	t_{WD}			100	ns
ALE↓ to $\overline{WR}$↓ Delay	t_{LW}		35		ns
Data Setup Time to $\overline{WR}$↑	t_{DW}		230		ns
Data Hold Time to $\overline{WR}$↑	t_{WDH}		95		ns
$\overline{WR}$↑ to ALE↑ Delay Time	t_{WL}		115		ns
$\overline{WR}$ Width Low	t_{WW}		280		ns

Note 1 : f_{XTAL} = 12MHz

2 : Load Capacitance ; C_L = 150pF

SERIAL OPERATION

Parameter	Symbol	Test Condition		MIN	MAX	Unit
$\overline{SCK}$ Cycle Time	t_{CYK}	$\overline{SCK}$ Input	*2	1		us
			*3	500		ns
		$\overline{SCK}$ Output		2		us
$\overline{SCK}$ Width Low	t_{KKL}	$\overline{SCK}$ Input	*2	400		ns
			*3	200		ns
		$\overline{SCK}$ Output		900		ns
$\overline{SCK}$ Width High	t_{KKH}	$\overline{SCK}$ Input	*2	400		ns
			*3	200		ns
		$\overline{SCK}$ Output		900		ns
RxD Setup Time to $\overline{SCK}\uparrow$	t_{RXK}	*2		80		ns
RxD Hold Time After $\overline{SCK}\uparrow$	t_{KRX}	*2		80		ns
$\overline{SCK}\downarrow$ to TxD Delay Time	t_{KTX}	*2			210	ns

*2. 1x Baud Rate in Asynchronous, Synchronous, I/O Interface Mode

*3. 16x Baud Rate or 64x Baud Rate in Asynchronous

ZERO-CROSS CHARACTERISTICS

(Ta =-10°C to +70°C, Vcc=+5.0V±5%, Vss=0V, Vcc-0.8V ≦ V_{DD} ≦ Vcc)

Parameter	Symbol	Test Condition	MIN	MAX	UNIT
Zero-Cross Detection Input	V_{ZX}	AC Coupled	1	3	VAC_{p-p}
Zero-Cross Accuracy	A_{ZX}	60 Hz Sine Wave		±135	mV
ZERO-Cross Detection Input Frequency	f_{ZX}		0.05	1	kHz

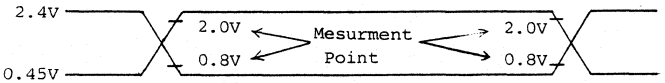
A/D CONVERTER CHARACTERISTICS

(Ta =-10°C to +70°C, Vcc =AVcc =+5.0V±5%, Vss =AVss =0V, AVcc-0.5V≦ V_{AREF} ≦AVcc)

Parameter	Symbol	Test Condition	MIN	TYP	MAX	UNIT
Resolution			8			Bits
Absolute Accuracy		Ta = -10°C to +50°C			0.4%±1/2	LSB
		Ta = -10°C to +70°C*4			0.6%±1/2	LSB
Conversion Time	t_{CONV}	83ns ≦ t_{CYC} ≦ 110ns	576			t_{CYC}
		110ns ≦ t_{CYC} ≦ 170ns	432			t_{CYC}
Sampling Time	t_{SAMP}	83ns ≦ t_{CYC} ≦ 110ns	96			t_{CYC}
		110ns ≦ t_{CYC} ≦ 170ns	72			t_{CYC}
Analog Input Voltage	V_{IAN}		0		V_{AREF}	V
Analog Input Impedance	R_{AN}			1000		MΩ
V_{AREF} Current	I_{AREF}		0.2	0.5	1.5	mA

*4. In case of f_{XTAL} ≦ 10MHz, Ta = -40°C to +85°C

AC TIMING MESURMENT POINT



Bus Timing depending on t_{CYC}

Symbol	Calculating Expression	MIN./MAX.	units
t_{AL}	2T - 100	MIN	ns
t_{LA}	T - 30	MIN	ns
t_{AR}	3T - 100	MIN	ns
t_{AD}	7T - 220	MAX	ns
t_{LDR}	5T - 200	MAX	ns
t_{RD}	4T - 150	MAX	ns
t_{LR}	T - 50	MIN	ns
t_{RL}	2T - 50	MIN	ns
t_{RR}	4T - 50 (Data Read) 7T - 50 (OP Code Fetch)	MIN	ns
t_{LL}	2T - 40	MIN	ns
t_{AW}	3T - 100	MIN	ns
t_{LDW}	T + 110	MAX	ns
t_{LW}	T - 50	MIN	ns
t_{DW}	4T - 100	MIN	ns
t_{WDH}	2T - 70	MIN	ns
t_{WL}	2T - 50	MIN	ns
t_{WW}	4T - 50	MIN	ns
t_{CYK}	12T ($\overline{SCK}$ Input) *1 24T ($\overline{SCK}$ Output)	MIN	ns
t_{KKL}	6T - 100 ($\overline{SCK}$ Input) *1 12T - 100 ($\overline{SCK}$ Output)	MIN	ns
t_{KKH}	6T - 100 ($\overline{SCK}$ Input) *1 12T - 100 ($\overline{SCK}$ Output)	MIN	ns

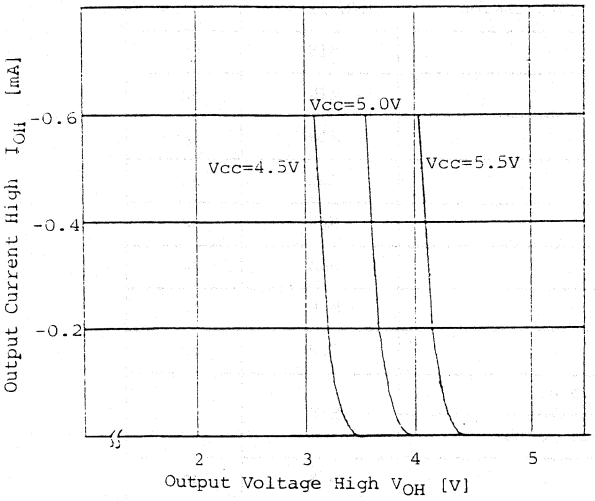
*1 1x Baud Rate in Asynchronous, Synchronous, I/O Interface Mode.

Note 1 : $T = t_{CYC} = 1/f_{XTAL}$

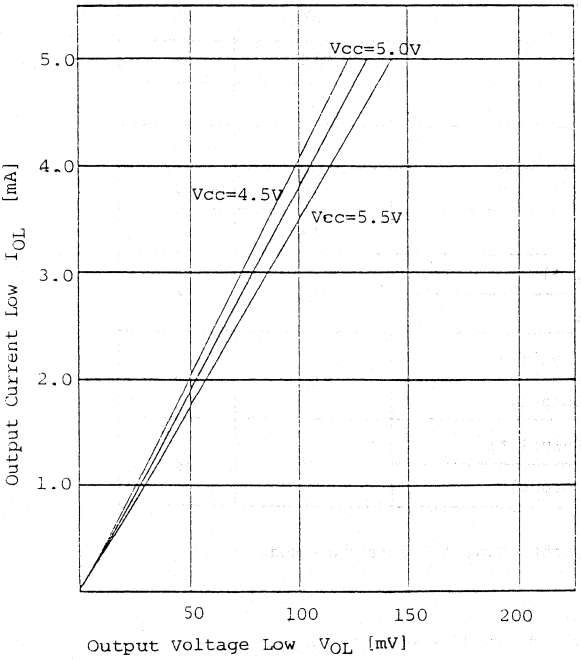
2 : The items out of this table are not dependent on f_{XTAL} .

CHARACTERISTICS CURVE (Ta = 25°C) --REFERENCE --

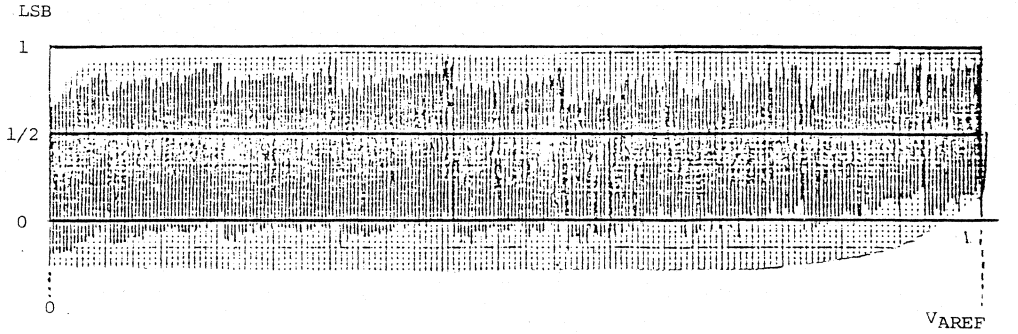
Output Voltage High vs Output Current High



Output Voltage Low vs Output Current Low

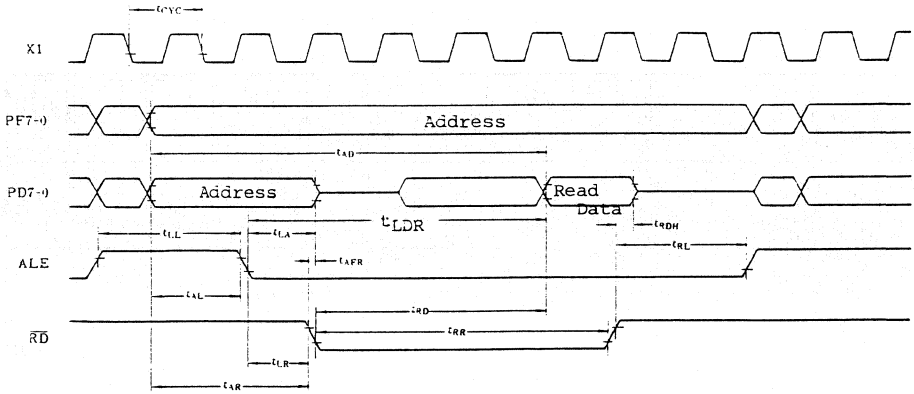


CHARACTERISTIC OF A/D CONVERTER

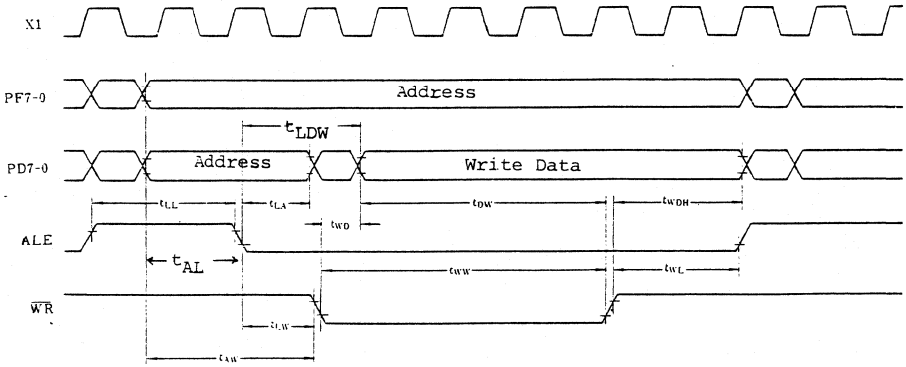


Timing Waveform

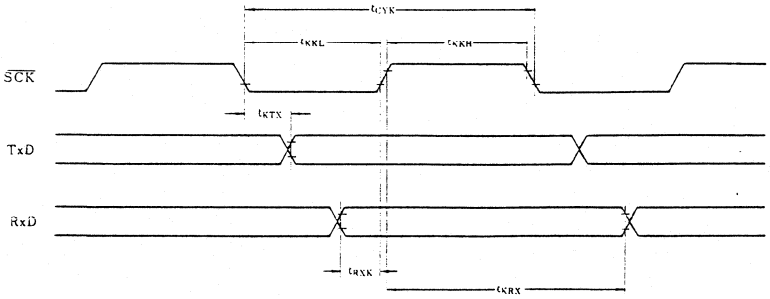
Read Operation



Write Operation



Serial Operation



CHAPTER 9 COMMAND FORMAT TABLE

This chapter list the IE-87AD commands. The format of the partition and content parameters are:

- partition: Specifies the address range.
  s.adr[TO e.adr]
  s.adr[LENGTH size]
    s.adr Start address
    e.adr End address
    size   Memory size

- contents: Indicates the constant.
  Numeric values, arithmetic expressions, binary, octal, and hexadecimal numbers can be specified in this parameter.

(1) IE87AD command (See APPENDIX for IE-7809)
    :drive:IE87AD

(2) EXIT command
    EXIT

(3) LOAD command
    LOAD:drive:filename [NOCODE / NOSYMBOL]

(4) SAVE command
    SAVE:drive:filename [NOCODE/partition / NOSYMBOL]

(5) LIST commands
    LIST
    LIST:device:
    LIST:drive:filename

(6) CHECK (self-diagnostic) command
    CHECK

(7) Mapping commands

MAP

```
MAP |s.adr TO e.adr    |=|INTERNAL|[RAM]|
    |s.adr LENGTH size|  |        |[ROM]|
                         |USER
                         |GUARDED
```

RESET MAP

(8) CLOCK commands

CLOCK

```
CLOCK= |INTERNAL|
       |EXTERNAL|
```

(9) Processor register commands

REGISTER

RV, RA, RB, RC RD, RE, RH, RL, RV', RA', RB', RC', RD', RE', RH', RL', RVA, RBC, RDE, RHL, EA, RVA', RBC', RDE', RHL', EA', SP, PC, Z, SK, HC, L1, L0, CY

The set command is set if "=contents" is specified in the above name.

(10) Mode registers (See APPENDIX for IE-7809)

MA, MB, MCC, MC, MM, MF, ETMM, SML, TMM[=contents], EOM[=contents], SMH[=contents], ANM[=contents], MODE

NOTE: TMM, EOM, SMH, and ANM can be displayed but other mode registers can only be set.

(11) PORT commands (See APPENDIX for IE-7809)

PORT

```
PORTA [=contents]    PORTA bit-no [=|0|]
                                  |1|
PORTB [=contents]    PORTB bit-no [=|0|]
                                  |1|
PORTC [=contents]    PORTC bit-no [=|0|]
                                  |1|
PORTD [=contents]    PORTD bit-no [=|0|]
                                  |1|
PORTF [=contents]    PORTF bit-no [=|0|]
                                  |1|
```

(12) CAUSE command

```
CAUSE
```

(13) Mask register commands

```
MK
MKH [=contents]
MKL [=contents]
```

(14) Timer register command

```
TM0 =contents
TM1
```

(15) Timer event counter register command (See APPENDIX f. IE-7809)

```
ETM0 =contents
ETM1
```

(16) CR register commands (IE-87AD only)

```
CR
CR0
CR1
CR2
CR3
```

(17) SERIAL commands

```
SERIAL
SERIAL=contents
```

(18) Memory commands

```
BYTE |s.adr TO e.adr    |
WORD |s.adr LENGTH size|
BYTE |s.adr TO e.adr    | =contents[,contents...]
WORD |s.adr LENGTH size|
|BYTE| s.adr=contents[,contents...]
|WORD|
```

(19) Symbol commands

```
SYMBOL
..mod-name.sym-name
..mod-name.sym-name=contents
/p-sym-name
/p-sym-name=contents
DEFINE |..mod-name.sym-name| =contents
       |/p-sym-name         |
REMOVE SYMBOL
       |..mod-name.sym-name| [,|         |,...]
       |/p-sym-name         |
```

(20) Break register commands

```
BR
BA0     BA0=partition [|WRITTEN|]
                      |READ   |
BA1     BA1=partition [|WRITTEN|]
                      |READ   |
BDR     BDR=[contents] [|WRITTEN|][WITH partition]
                       |READ   |
BEX     BEX=contents

BTMR    BTMR=contents
BLOOP   BLOOP=contents
```

(21) GO command

```
GO |[FROM s.adr][TILL clause[clause...]]|
   |[FOREVER]                            |
```

clause ① ° [LOCATION]partition[,partition] WRITTEN
READ

° BA0
° BA1
° BA0 BA1

② ° DATA=[contents] [|WRITTEN|][WITH partition]
|READ |

BDR

③ ° EXTERNAL=contents
° EXTERNAL

④ ° TIMER=contents
° TIMER

⑤ ° LOOP=contents
° LOOP

(22) STEP command

```
STEP[FROM s.adr][COUNT contents]
STEP |ENABLE |
     |DISABLE|
```

(23) Trace control commands

```
TRACE |PORTB   |
      |EXTERNAL|
MOVE |[+]| contents
     | - |
OLDEST
NEWEST
TRACE= |CYCLE      |
       |INSTRUCTION|
PRINT ALL
PRINT |[+]| contents
      | - |
TRACE ENABLE[clause[   clause...]]

clause ① [LOCATION]partition [|WRITTEN|]
                             [|READ   |]
       ② DATA=contents [|WRITTEN|]
                       [|READ   |]
       ③ PORTB=contents
       ④ PORTA=contents
       ⑤ FETCHED
TRACE DISABLE
```

(24) RESET CPU command

```
RESET CPU
```

(25) RESET HARDWARE command

```
RESET HARDWARE
```

(26) On-line assembler command

```
ASM FROM s.adr
```

(27) Disassembler command

```
DASM |s.adr TO e.adr   |
     |s.adr LENGTH size|
```

CHAPTER 10 μCOM87AD INSTRUCTION APPLICATION TABLE

μCOM-87AD INSTRUCTION APPLICATION TABLE

μCOM-87AD			machine language ⟷ mnemonic comparison table (alphabetic order)					
ACI	A,byte	56××	EQAX	rpa	70F9-70FF	ONA	A,r	60C8-60CF
ACI	r,byte	7450××-7457××	EQI	A,byte	77××	ONAW	wa	74C8××
ACI	sr2,byte	6450-3,5-7;64D0,1,3,5××	EQI	r,byte	7478××-747F××	ONAX	rpa	70C9-70CF
ADC	A,r	60D0-60D7	EQI	sr2,byte	6478-B,D-F;64F8,9,B,D××	ONI	A,byte	47××
ADC	r,A	6050-6057	EQIW	wa,byte	75××××	ONI	r,byte	7448××-744F××
ADCW	wa	74D0××	ESUB	EA,r2	7061-7063	ONI	sr2,byte	6448-B,D-F;64C8,9,B,D××
ADCX	rpa	70D1-70D7	EXA		10	ONIW	wa,byte	45××××
ADD	A,r	60C0-60C7	EXH		50	ORA	A,r	6098-609F
ADD	r,A	6040-6047	EXX		11	ORA	r,A	6018-601F
ADDNC	A,r	60A0-60A7	GTA	A,r	60A8-60AF	ORAW	wa	7498××
ADDNC	r,A	6020-6027	GTA	r,A	6028-602F	ORAX	rpa	7099-709F
ADDNCW	wa	74A0××	GTAW	wa	74A8××	ORI	A,byte	17××
ADDNCX	rpa	70A1-70A7	GTAX	rpa	70A9-70AF	ORI	r,byte	7418××-741F××
ADDW	wa	74C0××	GTF	A,byte	27××	ORI	sr2,byte	6418-B,D-F;6498,9,B,D××
ADDX	rpa	70C1-70C7	GTI	r,byte	7428××-742F××	ORIW	wa,byte	15××××
ADI	A,byte	46××	GTI	sr2,byte	6428-B,D-F;64A8,9,B,D××	POP	rp1	A0-A4
ADI	r,byte	7440××-7447××	GTIW	wa,byte	25××××	PUSH	rp1	B0-B4
ADI	sr2,byte	6440-3,5-7;64C8,1,3,5××	HLT		483B	RET		B8
ADINC	A,byte	26××	INR	r2	41-43	RETI		62
ADINC	r,byte	7420××-7427××	INRW	wa	20××	RETS		B9
ADINC	sr2,byte	6420-3,5-7;64A0,1,3,5××	INX	EA	A8	RLD		4838
ANA	A,r	6088-608F	INX	rp	02,12,22,32	RLL	r2	4835-4837
ANA	r,A	6008-600F	JB		21	RLR	r2	4831-4833
ANAW	wa	7488××	JEA		4828	RRD		4839
ANAX	rpa	7089-708F	JMP	word	54××××	SBB	A,r	60F0-60F7
ANI	A,byte	07××	JR	word	C0-FF	SBB	r,A	6070-6077
ANI	r,byte	7408××-740F××	JRE	word	4ED0-4FFF	SBBW	wa	74F0××
ANI	sr2,byte	6408-B,D-F;6488,9,B,D××	LBCD	word	701F××××	SBBX	rpa	70F1-70F7
ANIW	wa,byte	05××××	LDAW	wa	01××	SBCD	word	701E××××
BIT	bit,wa	58××-5F××	LDAX	rpa2	B-HF,AB××,AC-AE,AF××	SBI	A,byte	76××
BLOCK		31	LDEAX	rpa3	4882-5, B-F	SBI	r,byte	7470××-7477××
CALB		4829	LDED	word	702F××××	SBI	sr2,byte	6478-3,5-7;64F8,1,3,5××
CALF	word	7800-7FFF	LHLD	word	703F××××	SDED	word	702E××××
CALL	word	40××××	LSPD	word	700F××××	SHLD	word	703E××××
CALT	word	80-9F	LTA	A,r	60B8-60BF	SK	f	480A-480C
CLC		482A	LTA	r,A	6038-603F	SKIT	irf	4840-484C, 4850-4854
DAA		61	LTAW	wa	74B8××	SKN	f	481A-481C
DADC	EA,rp3	74D5-74D7	LTAX	rpa	70B9-70BF	SKNIT	irf	4860-486C, 4870-4874
DADD	EA,rp3	74C5-74C7	LTI	A,byte	37××	SLL	r2	4825-4827
DADDNC	EA,rp3	74A5-74A7	LTI	r,byte	7438××-743F××	SLLC	r2	4805-4807
DAN	EA,rp3	748D-748F	LTI	sr2,byte	6438-B,D-F;64B8,9,B,D××	SLR	r2	4821-4823
DCR	r2	51-53	LTIW	wa,byte	35××××	SLRC	r2	4801-4803
DCRW	wa	30××	LXI	rp2,word	04,14,24,34,44××××	SOFTI		72
DCX	EA	A9	MOV	r1,A	18-1F	SSPD	word	700E××××
DCX	rp	03,13,23,33	MOV	A,r1	08-0F	STAW	wa	63××
DEQ	EA,rp3	74FD-74FF	MOV	sr,A	4BC9-3,5-9;4BD9-A,F,8,A,B	STAX	rpa2	B-HF;BB××,BC-BE,BF××
DGT	EA,rp3	74AD-74AF	MOV	A,sr1	4CC9-3,5-9,B,D;4CD9;4CE9-3	STEAX	rpa3	4892-5, B-F
DI		BA	MOV	r,word	7868××××-786F××××	STC		482B
DIV	r2	483D-483F	MOV	word,r	7878××××-787F××××	SUB	A,r	60E0-60E7
DLT	EA,rp3	74BD-74BF	MUL	r2	482D-482F	SUB	r,A	6060-6067
DMOV	EA,rp3	A5-A7	MVI	sr2,byte	6480-3,5-7;6480,1,3,5××	SUBNB	A,r	60B0-60B7
DMOV	EA,sr4	48C0,48C1	MVI	r,byte	68××-6F××	SUBNB	r,A	6030-6037
DMOV	rp3,EA	B5-B7	MVIW	wa,byte	71××××	SUBNBW	wa	74B0××
DMOV	sr3,EA	48D2,48D3	MVIX	rpa1,byte	49××-4B××	SUBNBX	rpa	70B1-70B7
DNE	EA,rp3	74ED-74EF	NEA	A,r	60E8-60EF	SUBW	wa	74E0××
DOFF	EA,rp3	74DD-74DF	NEA	r,A	6068-606F	SUBX	rpa	70E1-70E7
DON	EA,rp3	74CD-74CF	NEAW	wa	74E8××	SUI	A,byte	66××
DOR	EA,rp3	749D-749F	NEAX	rpa	70E9-70EF	SUI	r,byte	7460××-7467××
DRLL	EA	48B4	NEGA		483A	SUI	sr2,byte	6460-3,5-7;64E0,1,3,5××
DRLR	EA	48B0	NEI	A,byte	67××	SUINB	A,byte	36××
DSBB	EA,rp3	74F5-74F7	NEI	r,byte	7468××-746F××	SUINB	r,byte	7430××-7437××
DSLL	EA	48A4	NEI	sr2,byte	6468-B,D-F;64E8,9,B,D××	SUINB	sr2,byte	6430-3,5-7;64B0,1,3,5××
DSLR	EA	48A0	NEIW	wa,byte	65××××	TABLE		48A8
DSUB	EA,rp3	74E5-74E7	NOP		00	XRA	A,r	6090-6097
DSUBNB	EA,rp3	74B5-74B7	OFFA	A,r	60D8-60DF	XRA	r,A	6010-6017
DXR	EA,rp3	7495-7497	OFFAW	wa	74D8××	XRAW	wa	7490××
EADD	EA,r2	7041-7043	OFFAX	rpa	70D9-70DF	XRAX	rpa	7091-7097
EI		AA	OFFI	A,byte	57××	XRI	A,byte	16××
EQA	A,r	60F8-60FF	OFFI	r,byte	7458××-745F××	XRI	r,byte	7410××-7417××
EQA	r,A	6078-607F	OFFI	sr2,byte	6458-B,D-F;64D8,9,B,D××	XRI	sr2,byte	6410-3,5-7;6490,1,3,5××
EQAW	wa	74F8××	OFFIW	wa,byte	55××××			

μCOM-87AD machine language ↔ mnemonic comparison table (1/4)

00	NOP		40	CALL	word	4860	SKNIT	FNMI	4DC0	MOV	PA, A
01	LDAW	wa	41	INR	A	4861	SKNIT	FT0	4DC1	MOV	PB, A
02	INX	SP	42	INR	B	4862	SKNIT	FT1	4DC2	MOV	PC, A
03	DCX	SP	43	INR	C	4863	SKNIT	F1	4DC3	MOV	PD, A
04	LXI	SP, word	44	LXI	EA, word	4864	SKNIT	F2	4DC5	MOV	PF, A
05	ANIW	wa, byte	45	ONIW	wa, byte	4865	SKNIT	FE0	4DC6	MOV	MKH, A
06			46	ADI	A, byte	4866	SKNIT	FE1	4DC7	MOV	MKL, A
07	ANI	A, byte	47	ONI	A, byte	4867	SKNIT	FEIN	4DC8	MOV	ANM, A
08	MOV	A, EAH	4801	SLRC	A	4868	SKNIT	FAD	4DC9	MOV	SMH, A
09	MOV	A, EAL	4802	SLRC	B	4869	SKNIT	FSR	4DCA	MOV	SML, A
0A	MOV	A, B	4803	SLRC	C	486A	SKNIT	FST	4DCB	MOV	EOM, A
0B	MOV	A, C	4805	SLLC	A	486B	SKNIT	ER	4DCC	MOV	ETMM, A
0C	MOV	A, D	4806	SLLC	B	486C	SKNIT	OV	4DCD	MOV	TMM, A
0D	MOV	A, E	4807	SLLC	C	4870	SKNIT	AN4	4DD0	MOV	MM, A
0E	MOV	A, H	480A	SK	CY	4871	SKNIT	AN5	4DD1	MOV	MCC, A
0F	MOV	A, L	480B	SK	HC	4872	SKNIT	AN6	4DD2	MOV	MA, A
10	EXA		480C	SK	Z	4873	SKNIT	AN7	4DD3	MOV	MB, A
11	EXX		481A	SKN	CY	4874	SKNIT	SB	4DD4	MOV	MC, A
12	INX	B	481B	SKN	HC	4882	LDEAX	D	4DD7	MOV	MF, A
13	DCX	B	481C	SKN	Z	4883	LDEAX	H	4DD8	MOV	TXB, A
14	LXI	B, word	4821	SLR	A	4884	LDEAX	D++	4DDA	MOV	TM0, A
15	ORIW	wa, byte	4822	SLR	B	4885	LDEAX	H++	4DDB	MOV	TM1, A
16	XRI	A, byte	4823	SLR	C	488B	LDEAX	D+byte	4E××	JRE	word
17	ORI	A, byte	4825	SLL	A	488C	LDEAX	H+A	↑		
18	MOV	EAH, A	4826	SLL	B	488D	LDEAX	H+B	4F××		
19	MOV	EAL, A	4827	SLL	C	488E	LDEAX	H+EA	50	EXH	
1A	MOV	B, A	4828	JEA		488F	LDEAX	H+byte	51	DCR	A
1B	MOV	C, A	4829	CALB		4892	STEAX	D	52	DCR	B
1C	MOV	D, A	482A	CLC		4893	STEAX	H	53	DCR	C
1D	MOV	E, A	482B	STC		4894	STEAX	D++	54	JMP	word
1E	MOV	H, A	482D	MUL	A	4895	STEAX	H++	55	OFFIW	wa, byte
1F	MOV	L, A	482E	MUL	B	489B	STEAX	D+byte	56	ACI	A, byte
20	INRW	wa	482F	MUL	C	489C	STEAX	H+A	57	OFFI	A, byte
21	JB		4831	RLR	A	489D	STEAX	H+B	58	BIT	0, wa
22	INX	D	4832	RLR	B	489E	STEAX	H+EA	59	BIT	1, wa
23	DCX	D	4833	RLR	C	489F	STEAX	H+byte	5A	BIT	2, wa
24	LXI	D, word	4835	RLL	A	48A0	DSLR	EA	5B	BIT	3, wa
25	GTIW	wa, byte	4836	RLL	B	48A4	DSLL	EA	5C	BIT	4, wa
26	ADINC	A, byte	4837	RLL	C	48A8	TABLE		5D	BIT	5, wa
27	GTI	A, byte	4838	RLD		48B0	DRLR	EA	5E	BIT	6, wa
28			4839	RRD		48B4	DRLL	EA	5F	BIT	7, wa
29	LDAX	B	483A	NEGA		48C0	DMOV	EA, ECNT	6008	ANA	V, A
2A	LDAX	D	483B	HLT		48C1	DMOV	EA, ECPT	6009	ANA	A, A
2B	LDAX	H	483D	DIV	A	48D2	DMOV	ETM0, EA	600A	ANA	B, A
2C	LDAX	D+	483E	DIV	B	48D3	DMOV	ETM1, EA	600B	ANA	C, A
2D	LDAX	H+	483F	DIV	C	49	MVIX	B, byte	600C	ANA	D, A
2E	LDAX	D−	4840	SKIT	FNMI	4A	MVIX	D, byte	600D	ANA	E, A
2F	LDAX	H−	4841	SKIT	FT0	4B	MVIX	H, byte	600E	ANA	H, A
30	DCRW	wa	4842	SKIT	FT1	4CC0	MOV	A, PA	600F	ANA	L, A
31	BLOCK		4843	SKIT	F1	4CC1	MOV	A, PB	6010	XRA	V, A
32	INX	H	4844	SKIT	F2	4CC2	MOV	A, PC	6011	XRA	A, A
33	DCX	H	4845	SKIT	FE0	4CC3	MOV	A, PD	6012	XRA	B, A
34	LXI	H, word	4846	SKIT	FE1	4CC5	MOV	A, PF	6013	XRA	C, A
35	LTIW	wa, byte	4847	SKIT	FEIN	4CC6	MOV	A, MKH	6014	XRA	D, A
36	SUINB	A, byte	4848	SKIT	FAD	4CC7	MOV	A, MKL	6015	XRA	E, A
37	LTI	A, byte	4849	SKIT	FSR	4CC8	MOV	A, ANM	6016	XRA	H, A
38			484A	SKIT	FST	4CC9	MOV	A, SMH	6017	XRA	L, A
39	STAX	B	484B	SKIT	ER	4CCB	MOV	A, EOM	6018	ORA	V, A
3A	STAX	D	484C	SKIT	OV	4CCD	MOV	A, TMM	6019	ORA	A, A
3B	STAX	H	4850	SKIT	AN4	4CD9	MOV	A, RXB	601A	ORA	B, A
3C	STAX	D+	4851	SKIT	AN5	4CE0	MOV	A, CR0	601B	ORA	C, A
3D	STAX	H+	4852	SKIT	AN6	4CE1	MOV	A, CR1	601C	ORA	D, A
3E	STAX	D−	4853	SKIT	AN7	4CE2	MOV	A, CR2	601D	ORA	E, A
3F	STAX	H−	4854	SKIT	SB	4CE3	MOV	A, CR3	601E	ORA	H, A

μCOM-87AD machine-language ⟷ mnemonic comparison table (2/4)

601F	ORA	L, A	606F	NEA	L, A	60B7	SUBNB	A, L	60F7	SBB	A, L				
6020	ADDNC	V, A	6070	SBB	V, A	60B8	LTA	A, V	60F8	EQA	A, V				
6021	ADDNC	A, A	6071	SBB	A, A	60B9	LTA	A, A	60F9	EQA	A, A				
6022	ADDNC	B, A	6072	SBB	B, A	60BA	LTA	A, B	60FA	EQA	A, B				
6023	ADDNC	C, A	6073	SBB	C, A	60BB	LTA	A, C	60FB	EQA	A, C				
6024	ADDNC	D, A	6074	SBB	D, A	60BC	LTA	A, D	60FC	EQA	A, D				
6025	ADDNC	E, A	6075	SBB	E, A	60BD	LTA	A, E	60FD	EQA	A, E				
6026	ADDNC	H, A	6076	SBB	H, A	60BE	LTA	A, H	60FE	EQA	A, H				
6027	ADDNC	L, A	6077	SBB	L, A	60BF	LTA	A, L	60FF	EQA	A, L				
6028	GTA	V, A	6078	EQA	V, A	60C0	ADD	A, V	61	DAA					
6029	GTA	A, A	6079	EQA	A, A	60C1	ADD	A, A	62	RETI					
602A	GTA	B, A	607A	EQA	B, A	60C2	ADD	A, B	63	STAW	wa				
602B	GTA	C, A	607B	EQA	C, A	60C3	ADD	A, C	6400	MVI	PA,	byte			
602C	GTA	D, A	607C	EQA	D, A	60C4	ADD	A, D	6401	MVI	PB,	byte			
602D	GTA	E, A	607D	EQA	E, A	60C5	ADD	A, E	6402	MVI	PC,	byte			
602E	GTA	H, A	607E	EQA	H, A	60C6	ADD	A, H	6403	MVI	PD,	byte			
602F	GTA	L, A	607F	EQA	L, A	60C7	ADD	A, L	6405	MVI	PF,	byte			
6030	SUBNB	V, A	6088	ANA	A, V	60C8	ONA	A, V	6406	MVI	MKH,	byte			
6031	SUBNB	A, A	6089	ANA	A, A	60C9	ONA	A, A	6407	MVI	MKL,	byte			
6032	SUBNB	B, A	608A	ANA	A, B	60CA	ONA	A, B	6408	ANI	PA,	byte			
6033	SUBNB	C, A	608B	ANA	A, C	60CB	ONA	A, C	6409	ANI	PB,	byte			
6034	SUBNB	D, A	608C	ANA	A, D	60CC	ONA	A, D	640A	ANI	PC,	byte			
6035	SUBNB	E, A	608D	ANA	A, E	60CD	ONA	A, E	640B	ANI	PD,	byte			
6036	SUBNB	H, A	608E	ANA	A, H	60CE	ONA	A, H	640D	ANI	PF,	byte			
6037	SUBNB	L, A	608F	ANA	A, L	60CF	ONA	A, L	640E	ANI	MKH,	byte			
6038	LTA	V, A	6090	XRA	A, V	60D0	ADC	A, V	640F	ANI	MKL,	byte			
6039	LTA	A, A	6091	XRA	A, A	60D1	ADC	A, A	6410	XRI	PA,	byte			
603A	LTA	B, A	6092	XRA	A, B	60D2	ADC	A, B	6411	XRI	PB,	byte			
603B	LTA	C, A	6093	XRA	A, C	60D3	ADC	A, C	6412	XRI	PC,	byte			
603C	LTA	D, A	6094	XRA	A, D	60D4	ADC	A, D	6413	XRI	PD,	byte			
603D	LTA	E, A	6095	XRA	A, E	60D5	ADC	A, E	6415	XRI	PF,	byte			
603E	LTA	H, A	6096	XRA	A, H	60D6	ADC	A, H	6416	XRI	MKH,	byte			
603F	LTA	L, A	6097	XRA	A, L	60D7	ADC	A, L	6417	XRI	MKL,	byte			
6040	ADD	V, A	6098	ORA	A, V	60D8	OFFA	A, V	6418	ORI	PA,	byte			
6041	ADD	A, A	6099	ORA	A, A	60D9	OFFA	A, A	6419	ORI	PB,	byte			
6042	ADD	B, A	609A	ORA	A, B	60DA	OFFA	A, B	641A	ORI	PC,	byte			
6043	ADD	C, A	609B	ORA	A, C	60DB	OFFA	A, C	641B	ORI	PD,	byte			
6044	ADD	D, A	609C	ORA	A, D	60DC	OFFA	A, D	641D	ORI	PF,	byte			
6045	ADD	E, A	609D	ORA	A, E	60DD	OFFA	A, E	641E	ORI	MKH,	byte			
6046	ADD	H, A	609E	ORA	A, H	60DE	OFFA	A, H	641F	ORI	MKL,	byte			
6047	ADD	L, A	609F	ORA	A, L	60DF	OFFA	A, L	6420	ADINC	PA,	byte			
6050	ADC	V, A	60A0	ADDNC	A, V	60E0	SUB	A, V	6421	ADINC	PB,	byte			
6051	ADC	A, A	60A1	ADDNC	A, A	60E1	SUB	A, A	6422	ADINC	PC,	byte			
6052	ADC	B, A	60A2	ADDNC	A, B	60E2	SUB	A, B	6423	ADINC	PD,	byte			
6053	ADC	C, A	60A3	ADDNC	A, C	60E3	SUB	A, C	6425	ADINC	PF,	byte			
6054	ADC	D, A	60A4	ADDNC	A, D	60E4	SUB	A, D	6426	ADINC	MKH,	byte			
6055	ADC	E, A	60A5	ADDNC	A, E	60E5	SUB	A, E	6427	ADINC	MKL,	byte			
6056	ADC	H, A	60A6	ADDNC	A, H	60E6	SUB	A, H	6428	GTI	PA,	byte			
6057	ADC	L, A	60A7	ADDNC	A, L	60E7	SUB	A, L	6429	GTI	PB,	byte			
6060	SUB	V, A	60A8	GTA	A, V	60E8	NEA	A, V	642A	GTI	PC,	byte			
6061	SUB	A, A	60A9	GTA	A, A	60E9	NEA	A, A	642B	GTI	PD,	byte			
6062	SUB	B, A	60AA	GTA	A, B	60EA	NEA	A, B	642D	GTI	PF,	byte			
6063	SUB	C, A	60AB	GTA	A, C	60EB	NEA	A, C	642E	GTI	MKH,	byte			
6064	SUB	D, A	60AC	GTA	A, D	60EC	NEA	A, D	642F	GTI	MKL,	byte			
6065	SUB	E, A	60AD	GTA	A, E	60ED	NEA	A, E	6430	SUINB	PA,	byte			
6066	SUB	H, A	60AE	GTA	A, H	60EE	NEA	A, H	6431	SUINB	PB,	byte			
6067	SUB	L, A	60AF	GTA	A, L	60EF	NEA	A, L	6432	SUINB	PC,	byte			
6068	NEA	V, A	60B0	SUBNB	A, V	60F0	SBB	A, V	6433	SUINB	PD,	byte			
6069	NEA	A, A	60B1	SUBNB	A, A	60F1	SBB	A, A	6435	SUINB	PF,	byte			
606A	NEA	B, A	60B2	SUBNB	A, B	60F2	SBB	A, B	6436	SUINB	MKH,	byte			
606B	NEA	C, A	60B3	SUBNB	A, C	60F3	SBB	A, C	6437	SUINB	MKL,	byte			
606C	NEA	D, A	60B4	SUBNB	A, D	60F4	SBB	A, D	6438	LTI	PA,	byte			
606D	NEA	E, A	60B5	SUBNB	A, E	60F5	SBB	A, E	6439	LTI	PB,	byte			
606E	NEA	H, A	60B6	SUBNB	A, H	60F6	SBB	A, H	643A	LTI	PC,	byte			

μCOM-87AD machine language ⟷ mnemonic comparison table (3/4)

643B	LTI	PD, byte	6488	ANI	ANM, byte	69	MVI	A, byte	70A7	ADDNCX	H−
643D	LTI	PF, byte	6489	ANI	SMH, byte	6A	MVI	B, byte	70A9	GTAX	B
643E	LTI	MKH, byte	648B	ANI	EOM, byte	6B	MVI	C, byte	70AA	GTAX	D
643F	LTI	MKL, byte	648D	ANI	TMM, byte	6C	MVI	D, byte	70AB	GTAX	H
6440	ADI	PA, byte	6490	XRI	ANM, byte	6D	MVI	E, byte	70AC	GTAX	D+
6441	ADI	PB, byte	6491	XRI	SMH, byte	6E	MVI	H, byte	70AD	GTAX	H+
6442	ADI	PC, byte	6493	XRI	EOM, byte	6F	MVI	L, byte	70AE	GTAX	D−
6443	ADI	PD, byte	6495	XRI	TMM, byte	700E	SSPD	word	70AF	GTAX	H−
6445	ADI	PF, byte	6498	ORI	ANM, byte	700F	LSPD	word	70B1	SUBNBX	B
6446	ADI	MKH, byte	6499	ORI	SMH, byte	701E	SBCD	word	70B2	SUBNBX	D
6447	ADI	MKL, byte	649B	ORI	EOM, byte	701F	LBCD	word	70B3	SUBNBX	H
6448	ONI	PA, byte	649D	ORI	TMM, byte	702E	SDED	word	70B4	SUBNBX	D+
6449	ONI	PB, byte	64A0	ADINC	ANM, byte	702F	LDED	word	70B5	SUBNBX	H+
644A	ONI	PC, byte	64A1	ADINC	SMH, byte	703E	SHLD	word	70B6	SUBNBX	D−
644B	ONI	PD, byte	64A3	ADINC	EOM, byte	703F	LHLD	word	70B7	SUBNBX	H−
644D	ONI	PF, byte	64A5	ADINC	TMM, byte	7041	EADD	EA, A	70B9	LTAX	B
644E	ONI	MKH, byte	64A8	GTI	ANM, byte	7042	EADD	EA, B	70BA	LTAX	D
644F	ONI	MKL, byte	64A9	GTI	SMH, byte	7043	EADD	EA, C	70BB	LTAX	H
6450	ACI	PA, byte	64AB	GTI	EOM, byte	7061	ESUB	EA, A	70BC	LTAX	D+
6451	ACI	PB, byte	64AD	GTI	TMM, byte	7062	ESUB	EA, B	70BD	LTAX	H+
6452	ACI	PC, byte	64B0	SUINB	ANM, byte	7063	ESUB	EA, C	70BE	LTAX	D−
6453	ACI	PD, byte	64B1	SUINB	SMH, byte	7068	MOV	V, word	70BF	LTAX	H−
6455	ACI	PF, byte	64B3	SUINB	EOM, byte	7069	MOV	A, word	70C1	ADDX	B
6456	ACI	MKH, byte	64B5	SUINB	TMM, byte	706A	MOV	B, word	70C2	ADDX	D
6457	ACI	MKL, byte	64B8	LTI	ANM, byte	706B	MOV	C, word	70C3	ADDX	H
6458	OFFI	PA, byte	64B9	LTI	SMH, byte	706C	MOV	D, word	70C4	ADDX	D+
6459	OFFI	PB, byte	64BB	LTI	EOM, byte	706D	MOV	E, word	70C5	ADDX	H+
645A	OFFI	PC, byte	64BD	LTI	TMM, byte	706E	MOV	H, word	70C6	ADDX	D−
645B	OFFI	PD, byte	64C0	ADI	ANM, byte	706F	MOV	L, word	70C7	ADDX	H−
645D	OFFI	PF, byte	64C1	ADI	SMH, byte	7078	MOV	word, V	70C9	ONAX	B
645E	OFFI	MKH, byte	64C3	ADI	EOM, byte	7079	MOV	word, A	70CA	ONAX	D
645F	OFFI	MKL, byte	64C5	ADI	TMM, byte	707A	MOV	word, B	70CB	ONAX	H
6460	SUI	PA, byte	64C8	ONI	ANM, byte	707B	MOV	word, C	70CC	ONAX	D+
6461	SUI	PB, byte	64C9	ONI	SMH, byte	707C	MOV	word, D	70CD	ONAX	H+
6462	SUI	PC, byte	64CB	ONI	EOM, byte	707D	MOV	word, E	70CE	ONAX	D−
6463	SUI	PD, byte	64CD	ONI	TMM, byte	707E	MOV	word, H	70CF	ONAX	H−
6465	SUI	PF, byte	64D0	ACI	ANM, byte	707F	MOV	word, L	70D1	ADCX	B
6466	SUI	MKH, byte	64D1	ACI	SMH, byte	7089	ANAX	B	70D2	ADCX	D
6467	SUI	MKL, byte	64D3	ACI	EOM, byte	708A	ANAX	D	70D3	ADCX	H
6468	NEI	PA, byte	64D5	ACI	TMM, byte	708B	ANAX	H	70D4	ADCX	D+
6469	NEI	PB, byte	64D8	OFFI	ANM, byte	708C	ANAX	D+	70D5	ADCX	H+
646A	NEI	PC, byte	64D9	OFFI	SMH, byte	708D	ANAX	H+	70D6	ADCX	D−
646B	NEI	PD, byte	64DB	OFFI	EOM, byte	708E	ANAX	D−	70D7	ADCX	H−
646D	NEI	PF, byte	64DD	OFFI	TMM, byte	708F	ANAX	H−	70D9	OFFAX	B
646E	NEI	MKH, byte	64E0	SUI	ANM, byte	7091	XRAX	B	70DA	OFFAX	D
646F	NEI	MKL, byte	64E1	SUI	SMH, byte	7092	XRAX	D	70DB	OFFAX	H
6470	SBI	PA, byte	64E3	SUI	EOM, byte	7093	XRAX	H	70DC	OFFAX	D+
6471	SBI	PB, byte	64E5	SUI	TMM, byte	7094	XRAX	D+	70DD	OFFAX	H+
6472	SBI	PC, byte	64E8	NEI	ANM, byte	7095	XRAX	H+	70DE	OFFAX	D−
6473	SBI	PD, byte	64E9	NEI	SMH, byte	7096	XRAX	D−	70DF	OFFAX	H−
6475	SBI	PF, byte	64EB	NEI	EOM, byte	7097	XRAX	H−	70E1	SUBX	B
6476	SBI	MKH, byte	64ED	NEI	TMM, byte	7099	ORAX	B	70E2	SUBX	D
6477	SBI	MKL, byte	64F0	SBI	ANM, byte	709A	ORAX	D	70E3	SUBX	H
6478	EQI	PA, byte	64F1	SBI	SMH, byte	709B	ORAX	H	70E4	SUBX	D+
6479	EQI	PB, byte	64F3	SBI	EOM, byte	709C	ORAX	D+	70E5	SUBX	H+
647A	EQI	PC, byte	64F5	SBI	TMM, byte	709D	ORAX	H+	70E6	SUBX	D−
647B	EQI	PD, byte	64F8	EQI	ANM, byte	709E	ORAX	D−	70E7	SUBX	H−
647D	EQI	PF, byte	64F9	EQI	SMH, byte	709F	ORAX	H−	70E9	NEAX	B
647E	EQI	MKH, byte	64FB	EQI	EOM, byte	70A1	ADDNCX	B	70EA	NEAX	D
647F	EQI	MKL, byte	64FD	EQI	TMM, byte	70A2	ADDNCX	D	70EB	NEAX	H
6480	MVI	ANM, byte	65	NEIW	wa, byte	70A3	ADDNCX	H	70EC	NEAX	D+
6481	MVI	SMH, byte	66	SUI	A, byte	70A4	ADDNCX	D+	70ED	NEAX	H+
6483	MVI	EOM, byte	67	NEI	A, byte	70A5	ADDNCX	H+	70EE	NEAX	D−
6485	MVI	TMM, byte	68	MVI	V, byte	70A6	ADDNCX	D−	70EF	NEAX	H−

μCOM-87AD machine language ⟷ mnemonic comparison table (4/4)

70F1	SBBX	B	7438	LTI	V, byte	7478	EQI	V, byte	74F8	EQAW	wa				
70F2	SBBX	D	7439	LTI	A, byte	7479	EQI	A, byte	74FD	DEQ	EA, B				
70F3	SBBX	H	743A	LTI	B, byte	747A	EQI	B, byte	74FE	DEQ	EA, D				
70F4	SBBX	D+	743B	LTI	C, byte	747B	EQI	C, byte	74FF	DEQ	EA, H				
70F5	SBBX	H+	743C	LTI	D, byte	747C	EQI	D, byte	75	EQIW	wa, byte				
70F6	SBBX	D−	743D	LTI	E, byte	747D	EQI	E, byte	76	SBI	A, byte				
70F7	SBBX	H−	743E	LTI	H, byte	747E	EQI	H, byte	77	EQI	A, byte				
70F9	EQAX	B	743F	LTI	L, byte	747F	EQI	L, byte	78	CALF	word				
70FA	EQAX	D	7440	ADI	V, byte	7488	ANAW	wa	↓						
70FB	EQAX	H	7441	ADI	A, byte	748D	DAN	EA, B	7F						
70FC	EQAX	D+	7442	ADI	B, byte	748E	DAN	EA, D	80	CALT	word				
70FD	EQAX	H+	7443	ADI	C, byte	748F	DAN	EA, H	↓						
70FE	EQAX	D−	7444	ADI	D, byte	7490	XRAW	wa	9F						
70FF	EQAX	H−	7445	ADI	E, byte	7495	DXR	EA, B	A0	POP	V				
71	MVIW	wa, byte	7446	ADI	H, byte	7496	DXR	EA, D	A1	POP	B				
72	SOFTI		7447	ADI	L byte	7497	DXR	EA, H	A2	POP	D				
7408	ANI	V, byte	7448	ONI	V, byte	7498	ORAW	wa	A3	POP	H				
7409	ANI	A, byte	7449	ONI	A, byte	749D	DOR	EA, B	A4	POP	EA				
740A	ANI	B, byte	744A	ONI	B, byte	749E	DOR	EA, D	A5	DMOV	EA, B				
740B	ANI	C, byte	744B	ONI	C, byte	749F	DOR	EA, H	A6	DMOV	EA, D				
740C	ANI	D, byte	744C	ONI	D, byte	74A0	ADDNCW	wa	A7	DMOV	EA, H				
740D	ANI	E, byte	744D	ONI	E, byte	74A5	DADDNC	EA, B	A8	INX	EA				
740E	ANI	H, byte	744E	ONI	H, byte	74A6	DADDNC	EA, D	A9	DCX	EA				
740F	ANI	L, byte	744F	ONI	L, byte	74A7	DADDNC	EA, H	AA	EI					
7410	XRI	V, byte	7450	ACI	V, byte	74A8	GTAW	wa	AB	LDAX	D+byte				
7411	XRI	A, byte	7451	ACI	A, byte	74AD	DGT	EA, B	AC	LDAX	H+A				
7412	XRI	B, byte	7452	ACI	B, byte	74AE	DGT	EA, D	AD	LDAX	H+B				
7413	XRI	C, byte	7453	ACI	C, byte	74AF	DGT	EA, H	AE	LDAX	H+EA				
7414	XRI	D, byte	7454	ACI	D, byte	74B0	SUBNBW	wa	AF	LDAX	H+byte				
7415	XRI	E, byte	7455	ACI	E, byte	74B5	DSUBNB	EA, B	B0	PUSH	V				
7416	XRI	H, byte	7456	ACI	H, byte	74B6	DSUBNB	EA, D	B1	PUSH	B				
7417	XRI	L, byte	7457	ACI	L, byte	74B7	DSUBNB	EA, H	B2	PUSH	D				
7418	ORI	V, byte	7458	OFFI	V, byte	74B8	LTAW	wa	B3	PUSH	H				
7419	ORI	A, byte	7459	OFFI	A, byte	74BD	DLT	EA, B	B4	PUSH	EA				
741A	ORI	B, byte	745A	OFFI	B, byte	74BE	DLT	EA, D	B5	DMOV	B, EA				
741B	ORI	C, byte	745B	OFFI	C, byte	74BF	DLT	EA, H	B6	DMOV	D, EA				
741C	ORI	D, byte	745C	OFFI	D, byte	74C0	ADDW	wa	B7	DMOV	H, EA				
741D	ORI	E, byte	745D	OFFI	E, byte	74C5	DADD	EA, B	B8	RET					
741E	ORI	H, byte	745E	OFFI	H, byte	74C6	DADD	EA, D	B9	RETS					
741F	ORI	L, byte	745F	OFFI	L, byte	74C7	DADD	EA, H	BA	DI					
7420	ADINC	V, byte	7460	SUI	V, byte	74C8	ONAW	wa	BB	STAX	D+byte				
7421	ADINC	A, byte	7461	SUI	A, byte	74CD	DON	EA, B	BC	STAX	H+A				
7422	ADINC	B, byte	7462	SUI	B, byte	74CE	DON	EA, D	BD	STAX	H+B				
7423	ADINC	C, byte	7463	SUI	C, byte	74CF	DON	EA, H	BE	STAX	H+EA				
7424	ADINC	D, byte	7464	SUI	D, byte	74D0	ADCW	wa	BF	STAX	H+byte				
7425	ADINC	E, byte	7465	SUI	E, byte	74D5	DADC	EA, B	C0	JR	word				
7426	ADINC	H, byte	7466	SUI	H, byte	74D6	DADC	EA, D	↓						
7427	ADINC	L, byte	7467	SUI	L, byte	74D7	DADC	EA, H	FF						
7428	GTI	V, byte	7468	NEI	V, byte	74D8	OFFAW	wa							
7429	GTI	A, byte	7469	NEI	A, byte	74DD	DOFF	EA, B							
742A	GTI	B, byte	746A	NEI	B, byte	74DE	DOFF	EA, D							
742B	GTI	C, byte	746B	NEI	C, byte	74DF	DOFF	EA, H							
742C	GTI	D, byte	746C	NEI	D, byte	74E0	SUBW	wa							
742D	GTI	E, byte	746D	NEI	E, byte	74E5	DSUB	EA, B							
742E	GTI	H, byte	746E	NEI	H, byte	74E6	DSUB	EA, D							
742F	GTI	L, byte	746F	NEI	L, byte	74E7	DSUB	EA, H							
7430	SUINB	V, byte	7470	SBI	V, byte	74E8	NEAW	wa							
7431	SUINB	A, byte	7471	SBI	A, byte	74ED	DNE	EA, B							
7432	SUINB	B, byte	7472	SBI	B, byte	74EE	DNE	EA, D							
7433	SUINB	C, byte	7473	SBI	C, byte	74EF	DNE	EA, H							
7434	SUINB	D, byte	7474	SBI	D, byte	74F0	SBBW	wa							
7435	SUINB	E, byte	7475	SBI	E, byte	74F5	DSBB	EA, B							
7436	SUINB	H, byte	7476	SBI	H, byte	74F6	DSBB	EA, D							
7437	SUINB	L, byte	7477	SBI	L, byte	74F7	DSBB	EA, H							

Flag operation

Operation						D6	D5	D4	D3	D2	D0
reg, memory			immediate		skip	Z	SK	HC	L1	L0	CY
ADD ADC SUB SBB DADD DADC DSUB DSBB EADD ESUB	ADDW ADCW SUBW SBBW	ADDX ADCX SUBX SBBX	ADI ACI SUI SBI			↕	0	↕	0	0	↕
ANA ORA XRA DAN DOR DXR	ANAW ORAW XRAW	ANAX ORAX XRAX	ANI ORI XRI	ANIW ORIW		↕	0	●	0	0	●
ADDNC SUBNB GTA LTA DADDNC DSUBNB DGT DLT	ADDNCW SUBNBW GTAW LTAW	ADDNCX SUBNBX GTAX LTAX	ADINC SUINC GTI LTI	GTIW LTIW		↕	↕	↕	0	0	↕
ONA OFFA DON DOFF	ONAW OFFAW	ONAX OFFAX	ONI OFFI	ONIW OFFIW		↕	↕	●	0	0	●
NEA EQA DNE DEQ	NEAW EQAW	NEAX EQAX	NEI EQI	NEIW EQIW		↕	↕	↕	0	0	↕
INR DCR	INRW DCRW					↕	↕	↕	0	0	●
DAA						↕	0	↕	0	0	↕
RLR, RLL, SLR, SLL DRLR, DRLL, DSLR, DSLL						●	0	●	0	0	↕
SLRC, SLLC						●	↕	●	0	0	↕
STC						●	0	●	0	0	1
CLC						●	0	●	0	0	0
			MVI A, byte			●	0	●	1	0	●
			MVI L, byte LXI H, word			●	0	●	0	1	●
					BIT SK SKN SKIT SKNIT	●	↕	●	0	0	●
					RETS	●	1	●	0	0	●
All other instructions						●	0	●	0	0	●

↕ ····· Affected (set or reset)

1 ····· Set

0 ····· Reset

● ····· Not affected

μCOM-87AD INSTRUCTION SET

Operand format/description

Format	Description
r r1 r2	V, A, B, C, D, E, H, L EAH, EAL, B, C, D, E, H, L A, B, C
sr sr1 sr2 sr3 sr4	PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, SML, EOM, ETMM, TMM, MM, MCC, MA, MB, MC, MF, TXB, TM0, TM1 PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, TMM, RXB, CR0, CR1, CR2, CR3 PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, TMM ETM0, ETM1 ECNT, ECPT
rp rp1 rp2 rp3	SP, B, D, H V, B, D, H, EA SP, B, D, H, EA B, D, H
rpa rpa1 rpa2 rpa3	B, D, H, D+, H+, D−, H− B, D, H B, D, H, D+, H+, D−, H−, D+byte, H+A, H+B, H+EA, H+byte D, H, D++, H++, D+byte, H+A, H+B, H+EA, H+byte
wa	8-bit immediate data
word byte bit	16-bit immediate data 8-bit immediate data 3-bit immediate data
f	CY, HC, Z
irf	FNMI, FT0, FT1, F1, F2, FE0, FE1, FEIN, FAD, FSR, FST, ER, OV, AN4, AN5, AN6, AN7, SB

Remarks

1. sr~sr4(special register)

PA : PORT A	ETMM : TIMER/ EVENT COUNTER MODE
PB : PORT B	EOM : TIMER/ EVENT COUNTER OUTPUT MODE
PC : PORT C	ANM : A/D CHANNEL MODE
PD : PORT D	CR0 : A/D CONVERSION RESULT 0~3
PF : PORT F	CR3
MA : MODE A	TXB : Tx BUFFER
MB : MODE B	RXB : Rx BUFFER
MC : MODE C	SMH : SERIAL MODE High
MCC : MODE CONTROL C	SML : SERIAL MODE Low
MF : MODE F	MKH : MASK High
MM : MEMORY MAPPING	MKL : MASK Low
TM0 : TIMER REG0	
TM1 : TIMER REG1	
TMM : TIMER MODE	
ETM0 : TIMER/ EVENT COUNTER REG0	
ETM1 : TIMER/ EVENT COUNTER REG1	
ECNT : TIMER/ EVENT COUNTER UPCOUNTER	
ECPT : TIMER/ EVENT COUNTER CAPTURE	

2. rp~rp3(register pair)

SP : STACK POINTER
B : BC
D : DE
H : HL
V : VA
EA : EXTENDED ACCUMULATOR

3. rpa~rpa3(rp addressing)

B	: (BC)
D	: (DE)
H	: (HL)
D+	: (DE)+
H+	: (HL)+
D−	: (DE)−
H−	: (HL)−
D++	: (DE)++
H++	: (HL)++
D+byte	: (DE+byte)
H+A	: (HL+A)
H+B	: (HL+B)
H+EA	: (HL+EA)
H+byte	: (HL+byte)

4. f(flag)

CY : CARRY
HC : HALF CARRY
Z : ZERO

5. irf(interrupt flag)

FNMI : INTFNMI	
FT0 : INTFT0	
FT1 : INTFT1	
F1 : INTF1	
F2 : INTF2	
FE0 : INTFE0	
FE1 : INTFE1	
FEIN : INTFEIN	
FAD : INTFAD	
FSR : INTFSR	
FST : INTFST	
ER : ERROR	
OV : OVERFLOW	
AN4 : ANALOG INPUT4 ~7	
AN7	
SB : STANDBY	

Instruction Group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
8-bit data transfer instruction	MOV	r1,A	1	4	r1←A			
		A,r1	1	4	A←r1			
		sr,A	2	10	sr←A			
		A,sr1	2	10	A←sr1			
		r,word	4	17	r←(word)			
		word,r	4	17	(word)←r			
	MVI	r,byte	2	7	r←byte			
		sr2,byte	3	14	sr2←byte			
	MVIW	Wa,byte	3	13	(V,Wa)←byte			
	MVIX	rpa1,byte	2	10	(rpa1) byte			
	STAW	Wa	2	10	(V,Wa)← A			
	LDAW	Wa	2	10	A←(V,Wa)			
	STAX	rpa2	1/2	7/13	(rpa2)←A			
	LDAX	rpa2	1/2	7/13	A←(rpa2)			
	EXX		1	4	B↔B',C↔C',D↔D' E↔E',H↔H',L↔L'			
	EXA		1	4	V,A↔V',A',EA↔EA'			
	EXH		1	4	H,L↔H',L'			
	BLOCK		1	13 (C+1)	(DE)$^{+}$←(HL)$^{+}$,C←C-1 End if borrow			
16-bit data transfer instruction	DMOV	rp3,EA	1	4	rp3$_L$←EAL,rp3$_H$←EAH			
		EA,rp3	1	4	EAL←rp3$_L$,EAH←rp3$_H$			
		sr3,EA	2	14	sr3←EA			
		EA,sr4	2	14	EA←sr4			
	SBCD	word	4	20	(word)←C,(word+1)←B			
	SDED	word	4	20	(word)←E,(word+1)←D			
	SHLD	word	4	20	(word)←L,(word+1)←H			
	SSPD	word	4	20	(word)←SP$_L$,(word+1)←SP$_H$			
	STEAX	rpa3	2/3	14/20	(rpa3)←EAL,(rpa3+1)←EAH			
	LBCD	word	4	20	C←(word),B←(word+1)			
	LDED	word	4	20	E←(word),D←(word+1)			
	LHLD	word	4	20	L←(word),H←(word+1)			
	LSPD	word	4	20	SP$_L$←(word),SP$_H$←(word+1)			
	LDEAX	rpa3	2/3	14/20	EAL←(rpa3),EAH←(rpa3+1)			
	PUSH	rp1	1	13	(SP-1)←rp1$_H$ (SP-2)←rp1$_L$,SP←SP-2			
	POP	rp1	1	10	rp1$_L$←(SP) rp1$_H$←(SP+1),SP←SP+2			
	LXI	rp2,word	3	10	rp2←(word)			
	TABLE		2	17	C←(PC+3+A) B←(PC+3+A+1)			
8-bit operation instruction register	ADD	A,r	2	8	A←A+r		↕	↕
		r,A	2	8	r←r+A		↕	↕
	ADC	A,r	2	8	A←A+r+CY		↕	↕
		r,A	2	8	r←r+A+CY		↕	↕
	ADDNC	A,r	2	8	A←A+r	No carry	↕	↕
		r,A	2	8	r←r+A	No carry	↕	↕
	SUB	A,r	2	8	A←A-r		↕	↕
		r,A	2	8	r←r-A		↕	↕
	SBB	A,r	2	8	A←A-r-CY		↕	↕
		r,A	2	8	r←r-A-CY		↕	↕
	SUBNB	A,r	2	8	A←A-r	No Borrow	↕	↕
		r,A	2	8	r←r-A	No Borrow	↕	↕
	ANA	A,r	2	8	A←A∧r			↕
		r,A	2	8	r←r∧A			↕
	ORA	A,r	2	8	A←A∨r			↕
		r,A	2	8	r←r∨A			↕
	XRA	A,r	2	8	A←A∨r			↕
		r,A	2	8	r←r∨A			↕

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
8-bit operation instruction (register)	GTA	A,r	2	8	A←r-1	No Borrow	↕	↕
		r,A	2	8	r←A-1	No Borrow	↕	↕
	LTA	A,r	2	8	A←r	Borrow	↕	↕
		r,A	2	8	r←A	Borrow	↕	↕
	NEA	A,r	2	8	A←r	No Zero	↕	↕
		r,A	2	8	r←A	No Zero	↕	↕
	EQA	A,r	2	8	A←r	Zero	↕	↕
		r,A	2	8	r←A	Zero	↕	↕
	ONA	A,r	2	8	A←A∧r	No Zero		↕
	OFFA	A,r	2	8	A∧r	Zero		↕
8-bit operation instruction (memory)	ADDX	rpa	2	11	A←A+(rpa)		↕	↕
	ADCX	rpa	2	11	A←A+(rpa)+CY		↕	↕
	ADDNCX	rpa	2	11	A←A+(rpa)	No carry	↕	↕
	SUBX	rpa	2	11	A←A-(rpa)		↕	↕
	SBBX	rpa	2	11	A←A-(rpa)-CY		↕	↕
	SUBNBX	rpa	2	11	A←A-(rpa)	No Borrow	↕	↕
	ANAX	rpa	2	11	A←A∧(rpa)			↕
	ORAX	rpa	2	11	A←A∨(rpa)			↕
	XRAX	rpa	2	11	A←A⊻(rpa)			↕
	GTAX	rpa	2	11	A-(rpa)-1	No Borrow	↕	↕
	LTAX	rpa	2	11	A-(rpa)	Borrow	↕	↕
	NEAX	rpa	2	11	A-(rpa)	No Zero	↕	↕
	EQAX	rpa	2	11	A-(rpa)	Zero	↕	↕
	ONAX	rpa	2	11	A∧(rpa)	No Zero		↕
	OFFAX	rpa	2	11	A∧(rpa)	Zero		↕
Immediate data operation instruction	ADI	A,byte	2	7	A←A+byte		↕	↕
		r,byte	3	11	r←r+byte		↕	↕
		sr2,byte	3	20	sr2←sr2+byte		↕	↕
	ACI	A,byte	2	7	A←A+byte+CY		↕	↕
		r,byte	3	11	r←r+byte+CY		↕	↕
		sr2,byte	3	20	sr2←sr2+byte+CY		↕	↕
	ADINC	A,byte	2	7	A←A+byte	No carry	↕	↕
		r,byte	3	11	r←r+byte	No carry	↕	↕
		sr2,byte	3	20	sr2←sr2+byte	No carry	↕	↕
	SUI	A,byte	2	7	A←A-byte		↕	↕
		r,byte	3	11	r←r-byte		↕	↕
		sr2,byte	3	20	sr2←sr2-byte		↕	↕
	SBI	A,byte	2	7	A←A-byte-CY		↕	↕
		r,byte	3	11	r←r-byte-CY		↕	↕
		sr2,byte	3	20	sr2←sr2-byte-CY		↕	↕
	SUINB	A,byte	2	7	A←A-byte	No Borrow	↕	↕
		r,byte	3	11	r←r-byte	No Borrow	↕	↕
		sr2,byte	3	20	sr2←sr2-byte	No Borrow	↕	↕
	ANI	A,byte	2	7	A←A∧byte			↕
		r,byte	3	11	r←r∧byte			↕
		sr2,byte	3	20	sr2←sr2∧byte			↕
	ORI	A,byte	2	7	A←A∨byte			↕
		r,byte	3	11	r←r∨byte			↕
		sr2,byte	3	20	sr2←sr2∨byte			↕
	XRI	A,byte	2	7	A←A⊻byte			↕
		r,byte	3	11	r←r⊻byte			↕
		sr2,byte	3	20	sr2←sr2⊻byte			↕
	GTI	A,byte	2	7	A-byte-1	No Borrow	↕	↕
		r,byte	3	11	r-byte-1	No Borrow	↕	↕

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	FLAG CY	FLAG Z
Immediate data operation instruction	GTI	sr2 byte	3	14	sr2-byte-1	No Borrow	↕	↕
	LTI	A,byte	2	7	A-byte	Borrow	↕	↕
		r,byte	3	11	r-byte	Borrow	↕	↕
		sr2,byte	3	14	sr2-byte	Borrow	↕	↕
	NEI	A,byte	2	7	A-byte	No Zero	↕	↕
		r,byte	3	11	r-byte	No Zero	↕	↕
		sr2,byte	3	14	sr2-byte	No Zero	↕	↕
	EQI	A,byte	2	7	A-byte	Zero	↕	↕
		r-byte	3	11	r-byte	Zero	↕	↕
		sr2,byte	3	14	sr2-byte	Zero	↕	↕
	ONI	A,byte	2	7	A∧byte	No Zero		↕
		r,byte	3	11	r∧byte	No Zero		↕
		sr2,byte	3	14	sr2∧byte	No Zero		↕
	OFFI	A, byte	2	7	A∧byte	Zero		↕
		r,byte	3	11	r∧byte	Zero		↕
		sr2,byte	3	14	sr2∧byte	Zero		↕
Working register operation instruction	ADDW	Wa	3	14	A←A+(V.Wa)		↕	↕
	ADCW	Wa	3	14	A←A+(V.Wa)+CY		↕	↕
	ADDNCW	Wa	3	14	A←A+(V.Wa)	No Carry	↕	↕
	SUBW	Wa	3	14	A←A-(V.Wa)		↕	↕
	SBBW	Wa	3	14	A←A-(V.Wa)-CY		↕	↕
	SUBNW	Wa	3	14	A←A-(V.Wa)	No Borrow	↕	↕
	ANAW	Wa	3	14	A←A∧(V.Wa)			↕
	ORAW	Wa	3	14	A←A∨(V.Wa)			↕
	XRAW	Wa	3	14	A←A∨(V.Wa)			↕
	GTAW	Wa	3	14	A-(V.Wa)-1	No Borrow	↕	↕
	LTAW	Wa	3	14	A-(V.Wa)	Borrow	↕	↕
	NEAW	Wa	3	14	A-(V.Wa)	No Zero	↕	↕
	EQAW	Wa	3	14	A-(V.Wa)	Zero	↕	↕
	ONAW	Wa	3	14	A∧(V.Wa)	No Zero		↕
	OFFAW	Wa	3	14	A∧(V.Wa)	Zero		↕
	ANIW	Wa,byte	3	19	(V.Wa)←(V.Wa)∧byte			↕
	ORIW	Wa,byte	3	19	(V.Wa)←(V.Wa)∨byte			↕
	GTIW	Wa,byte	3	13	(V.Wa)-byte-1	No Borrow	↕	↕
	LTIW	Wa,byte	3	13	(V.Wa)-byte	Borrow	↕	↕
	NEIW	Wa,byte	3	13	(V.Wa)-byte	No Zero	↕	↕
	EQIW	Wa,byte	3	13	(V.Wa)-byte	Zero	↕	↕
	ONIW	Wa,byte	3	13	(V.Wa)∧byte	No Zero		↕
	OFFIW	Wa,byte	3	13	(V.Wa)∧byte	Zero		↕
16-bit operation instruction	EADD	EA,r2	2	11	EA←EA+r2		↕	↕
	DADD	EA,rp3	2	11	EA←EA+rp3		↕	↕
	DADC	EA,rp3	2	11	EA←EA+rp3+CY		↕	↕
	DADDC	EA,rp3	2	11	EA←EA+rp3	No carry	↕	↕
	ESUB	EA,r2	2	11	EA←EA-r2		↕	↕
	DSUB	EA,rp3	2	11	EA←EA-rp3		↕	↕
	DSBD	EA,rp3	2	11	EA←EA-rp3-CY		↕	↕
	DSUBNB	EA,rp3	2	11	EA←EA-rp3	No Borrow	↕	↕
	DAN	EA,rp3	2	11	EA←EA∧rp3			↕
	DOR	EA,rp3	2	11	EA←EA∨rp3			↕
	DXR	EA,rp3	2	11	EA←EA∨rp3			↕
	DGT	EA,rp3	2	11	EA-rp3-1	No Borrow	↕	↕
	DLT	EA,rp3	2	11	EA-rp3	Borrow	↕	↕
	DNE	EA,rp3	2	11	EA-rp3	No Zero	↕	↕
	DEQ	EA,rp3	2	11	EA-rp3	Zero	↕	↕

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
16-bit operation instruction	DON	EA,rp3	2	11	EA∧rp3	No Zero		↕
	DOFF	EA,rp3	2	11	EA∧rp3	Zero		↕
Multiplication/devision instruction	MUL	r2	2	32	EA←A x r2			
	DIV	r2	2	59	EA←EA÷r2,r2←The rest			
Addition/subtraction instruction	INR	r2	1	4	r2←r2+1	Carry		↕
	INRW	wa	2	16	(V.Wa)←(V.Wa)+1	Carry		↕
	INX	rp	1	7	rp←rp+1			
		EA	1	7	EA←EA+1			
	DCR	r2	1	4	r2←r2-1	Borrow		↕
	DCRW	wa	2	16	(V.Wa)←(V.Wa)-1	Borrow		↕
	DCX	rp	1	7	rp←rp-1			
		EA	1	7	EA←EA-1			
	DAA		1	4	Decimal Adjust Accumulator		↕	↕
	STC		2	8	CY←1		↕	
	CLC		2	8	CY←0			
	NEGA		2	8	A A+1			
Rotation shift instruction	RLD		2	17	Rotate Left Digit			
	RRD		2	17	Rotate Right Digit			
	RLL	r2	2	8	$r2_{m+1}$←$r2_m$,$r2_0$←CY,CY←$r2_7$		↕	
	RLR	r2	2	8	$r2_{m-1}$←$r2_m$,$r2_7$←CY,CY←$r2_0$		↕	
	SLL	r2	2	8	$r2_{m+1}$←$r2_m$,$r2_0$←0,CY←$r2_7$		↕	
	SLR	r2	2	8	$r2_{m-1}$←$r2_m$,$r2_7$←0,CY←$r2_0$		↕	
	SLLC	r2	2	8	$r2_{m+1}$←$r2_m$,$r2_0$←0,CY←$r2_7$	Carry	↕	
	SLRC	r2	2	8	$r2_{m-1}$←$r2_m$,$r2_7$←0,CY←$r2_0$	Carry	↕	
	DRLL	EA	2	8	EA_{n+1}←EA_n,EA_0←CY,CY←EA_{15}		↕	
	DRLR	EA	2	8	EA_{n-1}←EA_n,EA_{15}←CY,CY←EA_0		↕	
	DSLL	EA	2	8	EA_{n+1}←EA_n,EA_0←0,CY←EA_{15}		↕	
	DSLR	EA	2	8	EAn-1←EA_n,EA_{15}←0,CY←EA_0		↕	
Jump instruction	JMP	word	3	10	PC←word			
	JB		1	4	PC_H←B,PC_L←C			
	JR	word	1	10	PC←PC+1+jdispl			
	JRE	word	2	10	PC←PC+2+jdisp			
	JEA		2	8	PC←EA			
Call instruction	CALL	word	3	16	(SP-1)←(PC+3)$_H$,(SP-2)←(PC+3)$_L$,PC←word			
	CALB		2	17	(SP-1)←(PC+2)$_H$,(SP-2)←(PC+2)$_L$,PC_H←B,PC_L←C			
	CALF	word	2	13	(SP-1)←(PC+2)$_H$,(SP-2)←(PC+2)$_L$ PC_{15-11}←00001,PC_{10-0}←fa			
	CALT	word	1	16	(SP-1)←(PC+1)$_H$,(SP-2)←(PC+1) PC_L←(128+2ta),PC_H←(129+2ta)			
	SOFT1		1	16	(SP-1)←PSW,(SP-2)←(PC+1)$_H$ (SP-3)←(PC+1)$_L$,PC←0060H			
Return instruction	RET		1	10	PC_L←(SP),PC_H←(SP+1) SP←SP+2			
	RETS		1	10	PC_L←(SP),PC_H←(SP+1) SP←SP+2,PC←PC+n	Uncondi-tional skip		
	RET1		1	13	PC_L←(SP),PC_H←(SP+1) PSW←(SP+2),SP←SP+3			

Instruction group	Mnemonic	Operand	Bytes	States	Operation	Skip condition	Flag CY	Flag Z
Skip instruction	BIT	bit,Wa	2	10	Skip if (V.Wa)bit=1	(V.Wa)Wa =1		
	SK	f	2	8	Skip if f=1	f=1		
	SKN	f	2	8	Skip if f=0	f=0		
	SKIT	irf	2	8	Skip if irf=1 then reset irf	irf=1		
	SKNIT	irf	2	8	Skip if irf=0 otherwise reset irf	irf=0		
CPU control instruction	NOP		1	4	No Operation			
	EI		1	4	Enable Interrupt			
	DI		1	4	Disable Interrupt			
	HLT		2	11	Halt			

NOTES: 1. In the expressions for bytes, rpa2, rpa3 to the right of the slash indicate D + byte and H + byte.

2. In the expressions for states, rpa2, rpa3 to the right of the slash indicate D + byte, H + A, H + B,H + E A and H + byte.

TRANSFER AND ARITHMETIC LOGICAL OPERATION INSTRUCTION TABLE

8-bit transfer instruction

Addressing	A, rl rl, A	A, srl sr, A	r, word word, r	r, byte	sr2, byte	rpal, byte	wa	wa, byte	rpa2		
									Register. indirect	Autoincrement, autodecrement	Base
Mnemonic	MOV			MVI		MVIX	LDAW	MVIW	LDAX		
							STAW		STAX		

16-bit transfer instrction

Addressing	EA, rp3 rp3, EA	EA, sr4 sr3, EA	rp2, word	word	rpa3		
					Register indirect	Autoincrement, autodecrement	Base
Mnemonic	DMOV		LXI	LBCD LDED LHLD LSPD	LDEAX		
				SBCD SDED SHLD SSPD	STEAX		

Arithmetic logical operation instruction

Addressing		8-bit operation							16-bit operation		Skip condition
		A, r	r, A	A, byte r, byte sr2, byte	rpa (See Notes 1 and 2.)	wa (See Note 1.)	wa, byte		EA, rp3	EA, r2	
Arithmetic operation instruction		ADD		ADI	ADDX	ADDW			DADD	EADD	
		ADC		ACI	ADCX	ADCW			DADC		
		ADDNC		ADINC	ADDNCX	ADDNCW			DADDNC		No Carry
		SUB		SUI	SUBX	SUBW			DSUB	ESUB	
		SBB		SBI	SBBX	SBBW			DSBB		
		SUBNB		SUINB	SUBNBX	SUBNBW			DSUBNB		No Borrow
Logical operation instruction	Logic	ANA		ANI	ANAX	ANAW		ANIW	DAN		
		ORA		ORI	ORAX	ORAW		ORIW	DOR		
		XRA		XRI	XRAX	XRAW			DXR		
	Comparison	GTA		GTI	GTAX	GTAW		GTIW	DGT		1st operand > 2nd operand
		LTA		LTI	LTAX	LTAW		LTIW	DLT		1st operand < 2nd operand
	Match	NEA		NEI	NEAX	NEAW		NEIW	DNE		1st operand ≠ 2nd operand
		EQA		EQI	EQAX	EQAW		EQIW	DEQ		1st operand = 2nd operand
	Test	ONA		ONI	ONAX	ONAW		ONIW	DON		No Zero
		OFFA		OFFI	OFFAX	OFFAW		OFFIW	DOFF		Zero

NOTE: 1. The 1st operand of skip condition is A (accumulator).
2. Register indirect and autoincrement/decrement possible.

μCOM-87AD MODE REGISTER APPLICATION TABLE

Special register operation instruction table

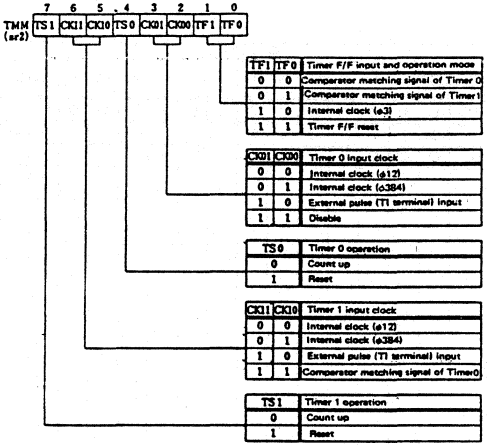
Instruction	Instruction code			Special register																
	B 1	B 2	B 3	PA, PB, PC PD, PF	MKH MKL	ANM	SMH	SML	EOM	ETMM	TMM	MM	MCC	MA	MB	MC	MF	TXB	RXB	TM0 TM1	CR0 ↓ CR3
MOV sr, A	01001101	110$S_4S_3S_2S_1S_0$		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○		○	
MOV A, sr1	01001100	110$S_4S_3S_2S_1S_0$		○	○	○	○		○		○								○		○
MVI sr2,byte	01100100	$S_4$0000$S_2S_1S_0$	Data	○	○	○	○		○		○										
ADI sr2,byte	01100100	$S_4$1000$S_2S_1S_0$	Data																		
ACI sr2,byte		$S_4$1010$S_2S_1S_0$																			
ADINC sr2,byte		$S_4$0100$S_2S_1S_0$																			
SUI sr2,byte		$S_4$1100$S_2S_1S_0$																			
SBI sr2,byte		$S_4$1110$S_2S_1S_0$																			
SUINB sr2,byte		$S_4$0110$S_2S_1S_0$																			
ANI sr2,byte		$S_4$0001$S_2S_1S_0$		○	○	○	○		○		○										
ORI sr2,byte		$S_4$0011$S_2S_1S_0$																			
XRI sr2,byte		$S_4$0010$S_2S_1S_0$																			
GTI sr2,byte		$S_4$0101$S_2S_1S_0$																			
LTI sr2,byte		$S_4$0111$S_2S_1S_0$																			
NEI sr2,byte		$S_4$1101$S_2S_1S_0$																			
EQI sr2,byte		$S_4$1111$S_2S_1S_0$																			
ONI sr2,byte		$S_4$1001$S_2S_1S_0$																			
OFFI sr2,byte		$S_4$1011$S_2S_1S_0$																			

Special register table

Special register	sr—sr2	Code S_6 S_4 S_3 S_2 S_1 S_0	sr	sr1	sr2
PORT A	PA	0 0 0 0 0 0	○	○	○
PORT B	PB	0 0 0 0 0 1	○	○	○
PORT C	PC	0 0 0 0 1 0	○	○	○
PORT D	PD	0 0 0 0 1 1	○	○	○
PORT F	PF	0 0 0 1 0 1	○	○	○
MASK High	MKH	0 0 0 1 1 0	○	○	○
MASK Low	MKL	0 0 0 1 1 1	○	○	○
A/D CHANNEL MODE	ANM	0 0 1 0 0 0	○	○	○
SERIAL MODE High	SMH	0 0 1 0 0 1	○	○	○
SERIAL MODE Low	SML	0 0 1 0 1 0	○		
TIMER/EVENT COUNTER OUTPUT MODE	EOM	0 0 1 0 1 1	○	○	○
TIMER/EVENT COUNTER MODE	ETMM	0 0 1 1 0 0	○		
TIMER MODE	TMM	0 0 1 1 0 1	○	○	○
MEMORY MAPPING	MM	0 1 0 0 0 0	○		
MODE CONTROL C	MCC	0 1 0 0 0 1	○		
MODE A	MA	0 1 0 0 1 0	○		
MODE B	MB	0 1 0 0 1 1	○		
MODE C	MC	0 1 0 1 0 0	○		
MODE F	MF	0 1 0 1 1 1	○		
Tx BUFFER	TXB	0 1 1 0 0 0	○		
Rx BUFFER	RXB	0 1 1 0 0 1		○	
TIMER REG 0	TM0	0 1 1 0 1 0	○		
TIMER REG 1	TM1	0 1 1 0 1 1	○		
A/D CONVERSION RESULT0	CR0	1 0 0 0 0 0		○	
A/D CONVERSION RESULT1	CR1	1 0 0 0 0 1		○	
A/D CONVERSION RESULT2	CR2	1 0 0 0 1 0		○	
A/D CONVERSION RESULT3	CR3	1 0 0 0 1 1		○	

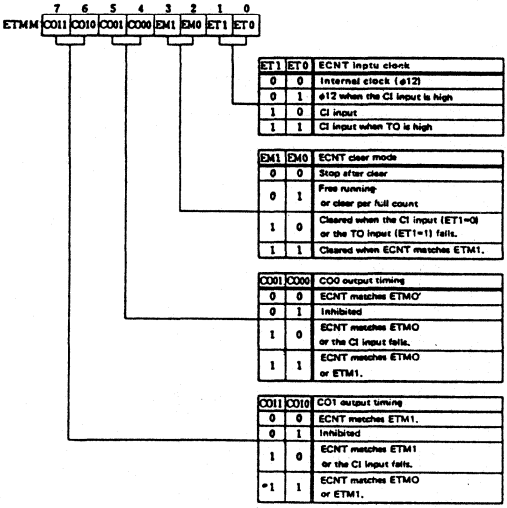
1. Timer

Timer mode register

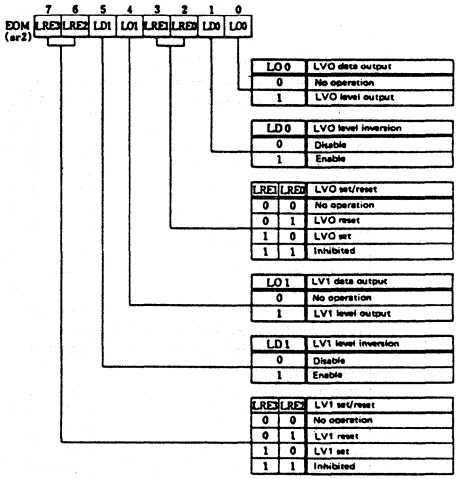


2. Timer/event counter

Timer/event counter mode register



Timer/event counter output mode register



Note: To reset the output latch, reset the level output of level F/F (LV0, LV1) and level F/F (LV0, LV1).

Note: ECPT latches the ECNT contents:

- o When the ET1 bit is 0 and the CI input falls.
- o When the ET1 bit is 1 and TO falls.

3. Serial interface

Serial mode high register

SMH (sr2)

7	6	5	4	3	2	1	0
—	TSK	IOE	SE	RxE	TxE	SK2	SK1

SK2	SK1	SCK selection
•0	0	Internal clock (TO output)
0	1	Internal clock (φ384)
1	0	Internal clock (φ24)
1	1	外部クロック

TxE	Transmit enable
0	Disable
1	Enable

RxE	Receive enable
0	Disable
1	Enable

SE	Search mode
0	Disable
1	Enable

IOE	I/O interface mode
0	Disable
1	Enable

TSK	SCK trigger
0	No operation
1	Enable

Serial mode low register

SML

7	6	5	4	3	2	1	0
S2	S1	EP	PEN	L2	L1	B2	B1

B2	B1	Clock rate
0	0	Synchronous operation
0	1	× 1
1	0	× 16
1	1	× 64

L2	L1	Character length
1	0	7 bits
1	1	8 bits

PEN	Paritity enable
0	Disable
1	Enable

EP	Even parity check/occurrence
0	Odd number
1	Even number

S2	S1	Number of stop bits
0	1	1 bit
1	1	2 bits

NOTE: Bit TSK is used to receive serial data in I/O interface mode.

4. Interrupt control

Interrupt mask high register

MKH (sr2)

7	6	5	4	3	2	1	0
—	—	—	—	—	MKST	MKSR	MKAD

0	INTAD unmask
1	INTAD mask

0	INTSR unmask
1	INTSR mask

0	INTST unmask
1	INTST mask

Interrupt mask low register

MKL (sr2)

7	6	5	4	3	2	1	0
MKEN	MKE1	MKE0	MK2	MK1	MKT1	MKT0	—

0	INTT0のマスク解除
1	INTTO mask

0	INTT1 unmask
1	INTT1 mask

0	INT1 unmask
1	INT1 mask

0	INT2 unmask
1	INT2 mask

0	INTEO unmask
1	INTEO mask

0	INTE1 unmask
1	INTE1 mask

0	INTEIN unmask
1	INTEIN mask

5. Port

Mode A register

7	6	5	4	3	2	1	0
MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0

MAn	PAn input/output
0	PAn : Output port
1	PAn : Input port

(n=0~7)

Mode B register

7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

MBn	PBn input/output
0	PBn : Output port
1	PBn : Input port

(n=0~7)

Mode C register

7	6	5	4	3	2	1	0
MC7	MC6	MC5	MC4	MC3	MC2	MC1	MC0

MCn	PCn input/output
0	PCn : Output port
1	PCn : Input port

(n=0~7)

Mode F register

7	6	5	4	3	2	1	0
MF7	MF6	MF5	MF4	MF3	MF2	MF1	MF0

MFn	PFn input/output
0	PFn : Output port
1	PFn : Input port

(n=0~7)

6. Analog/digical converter

A/D channel mode register

7	6	5	4	3	2	1	0
ANM (n=2)	—	—	—	FR	ANI2	ANI1	ANI0	MS

MS	Operating mode
0	Scan mode
1	Select mode

ANI2	ANI1	ANI0	Analog input
0	0	0	AN0
0	0	1	AN1
0	1	0	AN2
0	1	1	AN3
1	0	0	AN4
1	0	1	AN5
1	1	0	AN6
1	1	1	AN7

FR	Conversion speed specification
0	Oscillation frequency > 9MHz
1	Oscillation frequency ≤ 9 MHz

Mode control C register

7	6	5	4	3	2	1	0
MCC	MCC7	MCC6	MCC5	MCC4	MCC3	MCC2	MCC1	MCC0

0	PC0 : Port mode
1	PC0 : TxD output
0	PC1 : Port mode
1	PC1 : RxD input
0	PC2 : Port mode
1	PC2 : SCK input/output
0	PC3 : Port mode
1	PC3 : INT2/TI input
0	PC4 : Port mode
1	PC4 : TO output
0	PC5 : Port mode
1	PC5 : CI input
0	PC6 : Port mode
1	PC6 : CO0 output
0	PC7 : Port mode
1	PC7 : CO1 output

Memory mapping register

7	6	5	4	3	2	1	0
MM	—	—	—	—	RAE	MM2	MM1	MM0

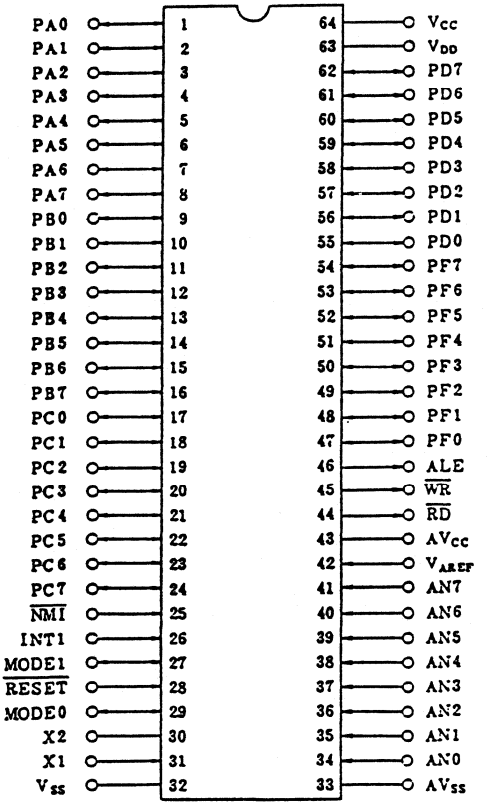
MM2	MM1	MM0	PF and PF operation	
0	0	0	Single-chip	PD7-0 : Input port PF7-0 : Port mode
0	0	1	Single-chip	PD7-0 : Output port PF7-0 : Port mode
0	1	0	256 bytes	PD7-0 : Extension mode PF7-0 : Port mode
1	0	0	4K bytes	PD7-0 PF3-0 : Extension mode PD7-4 : Port mode
1	1	0	16K bytes	PF7-0 PD5-0 : Extension mode PF7-6 : Port mode
1	1	1	60K bytes	PD7-0 PF7-0 : Extension mode

RAE	Built-in RAM access
0	Disable
1	Enable

Note: Set bits MM0, MM1, and MM2 of the memory mapping register (MM) to 0 when μPD7810 is used.

CHAPTER 11 μCOM87-AD PIN CONFIGURATION

(Top view)



PA7-0 : Port A
PB7-0 : Port B
PC7-0 : Port C
PD7-0 : Port D
PF7-0 : Port F
NMI : Non Maskable Interrupt
INT1 : Interrupt Request

MODE0, 1 : Mode0, 1
X1, X2 : Crystal
AN7-0 : Analog Input
RD : Read Strobe
WR : Write Strobe
ALE : Address Latch Enable
RESET : Reset
VAREF : Reference Voltage

CHAPTER 12 ERROR CODE TABLE

Error number	Description
1	MEMORY ERROR A user memory error or an internal IE-87AD memory error occurred.
30H	FORMAT ERROR The command input format is incorrect.
31H	SYMBOL TABLE OVERFLOWED A symbol table overflow occurred.
32H	WARNING A 2-byte jump instruction can be input for the on-line assembler.
33H	FILE ERROR The filename specified in the LOAD or SAVE command is invalid.
34H	I/O ERROR An error occurred during file I/O.
35H	LABEL OR NAME ERROR An invalid symbol was referenced in the online assembler.
36H	DUPLICATE SYMBOL ERROR The symbol was defined twice.
37H	ILLEGAL CHARACTER ERROR An unusable error was input.
38H	VALUE OVERFLOW ERROR The input numeric value exceeds the maximum value.
39H	UNDEFINED SYMBOL The specified symbol does not exist in the symbol table.
3AH	BALANCE ERROR A balance error occurred.
3CH	COMMAND NOT ALLOWED NOW The input command is not allowed now.
3DH	SYSTEM FIRMWARE IS SICK. The IE-87AD firmware is operating abnorml-ly. Reset and restart the firmware.

Error number	Description
3EH	SYMBOL TABLE NOT LOADED The symbol table is not loaded.
3FH	TOO MANY MODULES The number of modules to be loaded is too large.
40H	MEMORY GUARDED The memory to be accessed is not mapped.
41H	MAPPING ERROR A MAPPING command modification is not allowed.
42H	NO USER VCC No power is supplied to the prototype system.
43H	STACK ERROR The setting of the stack pointer is incorrect.
44H	EXPRESSION ERROR The numeric input method is incorrect.
45H	ORIGIN ERROR Use of the ORG pseudo instruction in the on-line assembler is incorrect.
46H	REFERENCE ERROR The address to be referenced by a BRANCH instruction in the on-line assembler is incorrect.
48H	AMBIGUOUS! The input is ambiguous.
49H	MISSING HARDWARE The connection of the IE-87AD hardware is incorrect. Check the hardware connection.
60H	NORMAL BREAK A break occurred because the break condition matched that of the user program.
61H	ESC BREAK A forcible break occurred because the user pressed the "ESC" key.

Error number	Description
60H	NORMAL BREAK A break occurred because the break condition matched that of the user program.
61H	ESC BREAK A forcible break occurred because the user pressed the "ESC" key.
62H	TIMER BREAK A break occurred because the timer condition was established.
63H	EXTERNAL BREAK A break occurred because the external break condition was established.
64H	NON MAP BREAK A break occurred because the unmapped memory was accessed.
65H	MEMORY ERROR BREAK A break occurred because an error occurred on the memory accessed during emulation.
70H	IMPOSSIBLE! No self-diagnosis was executed.
77H	NON TRACE DATA No traced data exists.
7AH	TRACE POINTER ERROR An error occurred because the trace pointer violated the movable range.

CHAPTER 13 COMMAND EXECUTION EXAMPLES

(1) Self-Diagnosis command

```
*CHECK
MEMORY  .....PASSED
M0/M1   .....PASSED
PORTD   .....PASSED
PORTF   .....PASSED
PORTA   .....PASSED
PORTB   .....PASSED
PORTC   .....PASSED
A/D     .....PASSED
SERIAL  .....PASSED
MODE0=1
MODE1=1
EXT. MEMORY = 64K
RAE (ENABLE/DISABLE)=DISABLE
*
```

(2) Register command

① Display example

```
*REGISTER                                        ; ALL REGISTER DISPLAY
RV =02    RA =00    RB =80    RC =80    RD =80    RE =80    RH =00    RL =00
RV'=00    RA'=FF    RB'=AA    RC'=00    RD'=00    RE'=3C    RH'=00    RL'=00
EA =0000  EA'=0000  PC =0000  SP =002C  PSW=000000008
*
*RV
RV=02
*
*RA
RA=00
*
*RB
RB=80
*
*RC
RC=80
*
*RD
RD=80
*
*RE
RE=80
*
*RH
RH=00
*
*RL
RL=00
*
*RV'
RV'=00
*
*RA'
RA'=FF
*
*RB'
RB'=AA
*
*RC'
RC'=00
*
*RD'
RD'=00
*
*RE'
RE'=3C
*
*RH'
RH'=00
*
*RL'
RL'=00
*
```

```
*RVA
RVA=0200
*
*RBC
RBC=8080
*
*RDE
RDE=8080
*
*RHL
RHL=0000
*
*RVA'
RVA'=00FF
*
*RBC'
RBC'=AA00
*
*RDE'
RDE'=003C
*
*RHL'
RHL'=0000
*
*SP
SP=002C
*
*PC
PC=0000
*
*PSW
PSW=00
*
*EA
EA=0000
*
*EA'
EA'=0000
*
*Z
Z=0
*
*SK
SK=0
*
*HC
HC=0
*
*L1
L1=0
*
*L0
L0=0
*
*CY
CY=0
```

② Setting example

```
*RV=1
*
*RA=0AH
*
*RB=0BH
*
*RC=0CH
*
*RD=0DH
*
*RE=0EH
*
*RH=2
*
*RL=3
*
*RV'=11H
*
*RA'=1AH
*
*RB'=1BH
*
*RC'=1CH
*
*RD'=1DH
*
*RE'=1EH
*
*RH'=12H
*
*RL'=13H
*
*REGISTER                                        ; ...........CHECK
RV =01     RA =0A     RB =0B     RC =0C     RD =0D     RE =0E     RH =02     RL =03
RV'=11     RA'=1A     RB'=1B     RC'=1C     RD'=1D     RE'=1E     RH'=12     RL'=13
EA =0000   EA'=0000   PC =0000   SP =002C   PSW=00000000B
*
*RVA=1234H
*
*RBC=5678H
*
*RDE=0ABCDH
*
*RHL=1256H
*
*EA=1234H
*
*RVA'=4321H
*
*RBC'=8765H
*
*RDE'=0DCBAH
*
*RHL'=6521H
*
```

```
*EA =0ABCDH
*
*SP=0FFF0H
*
*PC=1000H
*
*REGISTER
RV =12    RA =34    RB =56    RC =78    RD =AB    RE =CD    RH =12    RL =56
RV'=43    RA'=21    RB'=87    RC'=65    RD'=DC    RE'=BA    RH'=65    RL'=21
EA =1234  EA'=ABCD  PC =1000  SP =FFF0  PSW=00000000B
*
*PSW=0FFH
*
*PSW
PSW=FF
*
*CY=0
*
*PSW
PSW=FE
*
*L0=0
*
*PSW
PSW=FA
*
*L1=0
*
*PSW
PSW=F2
*
*HC=0
*
*PSW
PSW=E2
*
*SK=0
*
*PSW
PSW=C2
*
*Z=0
*
*PSW
PSW=82
*
```

(3) Mode register command

```
*MODE
SMH=00      EOM=00      TMM=FF      ANM=00
*
*SMH
SMH=00
*
*EOM
EOM=00
*
*TMM
TMM=FF
*
*ANM
ANM=00
*
*SMH=10H
*
*EOM=20H
*
*TMM=30H
*
*ANM=40H
*
*MODE
SMH=10      EOM=20      TMM=30      ANM=40
```

(4) Mask register command

```
*MK
MKH=00000111B  MKL=11111110B
*
*MKH
MKH=00000111B
*
*MKL
MKL=11111110B
*
*MKH=0AAH
*
*MKL=088H
*
*MK
MKH=10101010B  MKL=10111011B
```

(5) Timer register command

```
*TM0=10H
*
*TM1=20H
*
```

(6) Even timer register command

```
*ETM0=10H
*
*ETM1=20H
*
*ECNT
ECNT=0000
*
*ECPT
ECPT=FFFF
```

(7) Condition register command

```
*CR
CR0=FF    CR1=0F    CR2=3E    CR3=1F
*
*CR0
CR0=FF
*
*CR1
CR1=0F
*
*CR2
CR2=3E
*
*CR3
CR3=1F
```

(8) Serial command

```
*SERIAL
SERIAL=55
*
*SERIAL = 0AAH
*
*SERIAL
SERIAL=55
```

(9) Port command

① Display example

```
*PORT                                   ; DISPLAY PORT DATA
PORTA=00  PORTB=00  PORTC=00  PORTD=FF  PORTF=00
*
*PORTA                                  ; DISPLAY PORT A
PORTA=00
*
*PORTB                                  ; DISPLAY PORT B
PORTB=00
*
*PORTC                                  ; DISPLAY PORT C
PORTC=00
*
*PORTD                                  ; DISPLAY PORT D
PORTD=FF
*
*PORTF                                  ; DISPLAY PORT F
PORTF=00
```

② Setting example

```
*MA=0FFH                          ; PORTA 'IN'
*
*PORTA
PORTA=00
*
*PORTA = 0AAH                     ; OUTPUT '0AAH'
*
*PORTA                            ;
PORTA=00
*
*MA=0                             ; PORTA 'OUT'
*
*PORTA                            ;
PORTA=AA
*MB=0FFH                          ; PORTB 'IN'
*
*PORTB
PORTB=AA
*
*PORTB=0BBH                       ; OUTPUT '0BBH'
*
*PORTB                            ;
PORTB=AA
*
*MB=0                             ; PORTB 'OUT'
*
*PORTB                            ;
PORTB=BB
*MCC=0                            ; PORTC MODE = PORT MODE
*
*MC=0FFH                          ; PORTC 'IN'
*
*PORTC
PORTC=BB
*
*PORTC=0CCH                       ; OUTPUT '0CCH'
*
*PORTC                            ;
PORTC=BB
*
*MC=0                             ; PORTC 'OUT'
*
*PORTC                            ;
PORTC=CC
```

(10) Mapping command

```
*MAP
*
*MAP 0 TO 2000H = INTERNAL          ; MAP INTERNAL RAM
*
*MAP 3000H LENGTH 100H = INT RAM    ; MAP INTERNAL RAM
*
*MAP 4000H TO 4100H = INT ROM       ; MAP INTERNAL ROM
*
*MAP 5000H LENGTH 1000H = USER      ; MAP USER
*
*MAP
0000-20FF=INT RAM   3000-30FF=INT RAM   4000-41FF=INT ROM   5000-5FFF=USER
*
*MAP 5000H TO 50FFH = GUARDED       ; NON MAP
*
*MAP
0000-20FF=INT RAM   3000-30FF=INT RAM   4000-41FF=INT ROM   5100-5FFF=USER
*
```

(11) Break register command

1 All registers display

```
*BR
BA0=0000
BA1=0000
BOR=XX
BEX=00
BTMR=0000
BLOOP=01
*
```

② Address break register

```
*BA1
BA1=0000
*
*BA1=0
*
*BA1
BA1=0000 .
*
*BA1= 0 TO 0FFH
*
*BA1
BA1=0000-00FF
*
*BA1 = 0 TO 0FFH READ
*
*BA1
BA1=0000-00FF R
*
*BA1 = 0 LENGTH 0FFH WRITTEN
*
*BA1
BA1=0000-00FE W
*
*BA1 = READ
*
*BA1
BA1=0000 R
*
*BA1 = WRITTEN
*
*BA1
BA1=0000 W
*
```

③ Data break register

```
*BDR
BDR=XX
*
*BDR = 0
*
*BDR
BDR=00
*
*BDR = 0 READ
*
*BDR
BDR=00 R
*
*BDR = 0 WRITTEN
*
*BDR
BDR=00 W
*
*BDR = 0 WITH 1000H
*
*BDR
BDR=00(1000)
*
*BDR = 10H WITH 1000H LENGTH 10H
*
*BDR
BDR=10(1000-100F)
*
*BDR = 30H WITH 2000H TO 200FH
*
*BDR
BDR=30(2000-200F)
*
*BDR = 20H READ WITH 1000H
*
*BDR
BDR=20(1000) R
*
*BDR = 20H READ WITH 2000H LENGTH 10H
*
*BDR
BDR=20(2000-200F) R
*
```

```
*BDR = READ
*
*BDR
BDR=XX R
*
*BDP = WRITTEN
*
*BDR
BDR=XX W
*
*BDR = READ WITH 1000H TO 1030H
*
*BDR
BDR=XX(1000-1030) R
*
*BDR = WRITTEN WITH 2000H LENGTH 10H
*
*BDR
BDR=XX(2000-200F) W
*
*BDR = WITH 1000H
*
*BDR
BDR=XX(1000)
*
*BDR = WITH 2000H LENGTH 1
*
*BDR
BDR=XX(2000)
*
*BDR = WITH 3000H TO 300FH
*
*BDR
BDR=XX(3000-300F)
*
```

④ External break register

```
*BEX
BEX=00
*
*BEX = 0
*
*BEX
BEX=00
*
*BEX = 10000000B
*
*BEX
BEX=80
*
```

⑤ Timer break register

```
*BTMR
BTMR=0000
*
*BTMR = 0
*
*BTMR
BTMR=0000
*
*BTMR = 0FFFFH
*
*BTMR
BTMR=FFFF
*
```

⑥ Loop break register

```
*BLOOP
BLOOP=01
*
*BLOOP=10
*
*BLOOP
BLOOP=0A
*
*BLOOP=3
*
*BLOOP
BLOOP=03
*
```

(12) Memory command

① BYTE type

```
*BYTE 0 TO 0FFH = 0
*
*BYTE 0 TO 0FFH
0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
*BYTE 0 TO 0FH = 10H
*
*BYTE 10H TO 1FH = 20H
*
*BYTE 20H = 1, 2, 3, 4, 5, 6, 7, 8, 9
*
*BYTE 30H TO 3FH = 1, 2, 3, 4, 5, 6, 7, 8, 9
*
*BYTE 40H = 40H
*
*BYTE 50H = 50H
*
*BYTE 0 TO 200
0000  10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
0010  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0020  01 02 03 04 05 06 07 08 09 00 00 00 00 00 00 00
0030  01 02 03 04 05 06 07 08 09 09 09 09 09 09 09 09
0040  40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050  50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0  00 00 00 00 00 00 00 00 00
*
```

② WORD type

```
*
*WORD 0 TO 0FFH = 0
*
*WORD 0 TO 0FFH
0000  0000 0000 0000 0000 0000 0000 0000 0000
0010  0000 0000 0000 0000 0000 0000 0000 0000
0020  0000 0000 0000 0000 0000 0000 0000 0000
0030  0000 0000 0000 0000 0000 0000 0000 0000
0040  0000 0000 0000 0000 0000 0000 0000 0000
0050  0000 0000 0000 0000 0000 0000 0000 0000
0060  0000 0000 0000 0000 0000 0000 0000 0000
0070  0000 0000 0000 0000 0000 0000 0000 0000
0080  0000 0000 0000 0000 0000 0000 0000 0000
0090  0000 0000 0000 0000 0000 0000 0000 0000
00A0  0000 0000 0000 0000 0000 0000 0000 0000
00B0  0000 0000 0000 0000 0000 0000 0000 0000
00C0  0000 0000 0000 0000 0000 0000 0000 0000
00D0  0000 0000 0000 0000 0000 0000 0000 0000
00E0  0000 0000 0000 0000 0000 0000 0000 0000
00F0  0000 0000 0000 0000 0000 0000 0000 0000
*
*WORD 0 TO 0FH = 10H
*
*WORD 10H TO 1FH = 20H
*
*WORD 30H = 1, 2, 3, 4, 5, 6, 7, 8, 9, 1234H, 5678H
*
*WORD 30H TO 3FH = 1, 2, 3, 4, 5, 6, 7, 8, 9
WARNING
*
*WORD 40H = 0ABCDH
*
*WORD 50H = 9876H
*
*WORD 0 TO 200
0000  0010 0010 0010 0010 0010 0010 0010 0010
0010  0020 0020 0020 0020 0020 0020 0020 0020
0020  0000 0000 0000 0000 0000 0000 0000 0000
0030  0001 0002 0003 0004 0005 0006 0007 0008
0040  ABCD 1234 5678 0000 0000 0000 0000 0000
0050  9876 0000 0000 0000 0000 0000 0000 0000
0060  0000 0000 0000 0000 0000 0000 0000 0000
0070  0000 0000 0000 0000 0000 0000 0000 0000
0080  0000 0000 0000 0000 0000 0000 0000 0000
0090  0000 0000 0000 0000 0000 0000 0000 0000
00A0  0000 0000 0000 0000 0000 0000 0000 0000
00B0  0000 0000 0000 0000 0000 0000 0000 0000
00C0  0000 0000 0000 0000 0000
*
*BYTE 0 TO 200
0000  10 00 10 00 10 00 10 00 10 00 10 00 10 00 10 00
0010  20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030  01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00
0040  CD AB 34 12 78 56 00 00 00 00 00 00 00 00 00 00
0050  76 98 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0  00 00 00 00 00 00 00 00 00
*
```

(13) Symbol command

```
*SYMBOL                    ; DISPLAY ALL SYMBOL REFERENCE
SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS
PUBLIC
AB      0001  ABC     0002  ABCD    0003  ABCDE   0004  ABCDEF 0005
MOD1
MOD101 0010  MOD102 0020  MOD103 0030  MOD104 0040  MOD105 0050
MOD2
MD2ABC 0100  MD2BCD 0200  MD2CDE 0300  MD2PA0 0400  MD2PA1 0500  MD2STT 0600

*
*/ABC                     ; DISPLAY ONE SYMBOL REFERENCE
ABC    =0002

*..MOD1.MOD104            ; DISPLAY ONE SYMBOL REFERENCE
MOD104=0040
*
*..MOD2.MD2STT            ; DISPLAY ONE SYMBOL REFERENCE
MD2STT=0600
*
*/ABC = 1000H             ; CHANGE
*
*..MOD1.MOD103 = 2000H    ; CHANGE
*
*..MOD2.MD2PA0 = 3000H    ; CHANGE
*
*SYMBOL
SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS
PUBLIC
AB      0001  ABC     1000  ABCD    0003  ABCDE   0004  ABCDEF 0005
MOD1
MOD101 0010  MOD102 0020  MOD103 2000  MOD104 0040  MOD105 0050
  OD2
MD2ABC 0100  MD2BCD 0200  MD2CDE 0300  MD2PA0 3000  MD2PA1 0500  MD2STT 0600

*
```

```
*DEFINE /XYZ = 4000H      ; DEFINE SYMBOL
*
*DEFINE ..MOD1.VWX = 30H ; DEFINE SYMBOL
*
*DEFINE ..MOD2.STU = 90H ; DEFINE SYMBOL
*
*SYMBOL
SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS
PUBLIC
AB      0001  ABC     1000  ABCD    0003  ABCDE   0004  ABCDEF 0005  XYZ     4000

 MOD1
MOD101 0010  MOD102 0020  MOD103 2000  MOD104 0040  MOD105 0050  VWX     0030

MOD2
MD2ABC 0100  MD2BCD 0200  MD2CDE 0300  MD2PA0 3000  MD2PA1 0500  MD2STT 0600
STU     0090
*
*REMOVE /ABCD, ..MOD1.MOD101, ..MOD2.MD2STT
*                          ; REMOVE SOME SYMBOL
*
*SYMBOL
SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS
PUBLIC
AB      0001  ABC     1000  ABCDE   0004  ABCDEF 0005  XYZ     4000
MOD1
MOD102 0020  MOD103 2000  MOD104 0040  MOD105 0050  VWX     0030
MOD2
MD2ABC 0100  MD2BCD 0200  MD2CDE 0300  MD2PA0 3000  MD2PA1 0500  STU     0090


*REMOVE SYMBOL             ; REMOVE ALL SYMBOL
*
*SYMBOL
SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS
PUBLIC

*
```

(14) LOAD commnad

```
*;              LOAD FILE  ':F1:IADEXC.HEX'
*;                         ':F1:IADEXC.SYM'
*
*LOAD :F1:IADEXC           ; LOAD TEST PROGRAM
*
*DASM .START TO .TSTEND    ; DISPLAY TEST PROGRAM
0000 69 01        START :  MVI     A,01H
0002 6A 02                 MVI     B,02H
0004 6B 03                 MVI     C,03H
0006 6C 04                 MVI     D,04H
0008 6D 05                 MVI     E,05H
000A 6E 06                 MVI     H,06H
000C 6F 07                 MVI     L,07H
000E 68 08                 MVI     V,08H
0010 10           EXCHG1:  EXA
0011 11                    EXX
0012 69 10        CHGRG :  MVI     A,10H
0014 6A 20                 MVI     B,20H
0016 6B 30                 MVI     C,30H
0018 6C 40                 MVI     D,40H
001A 6D 50                 MVI     E,50H
001C 6E 60                 MVI     H,60H
001E 6F 70                 MVI     L,70H
0020 68 80                 MVI     V,80H
0022 10           EXCHG2:  EXA
0023 11                    EXX
0024 34 00 10     MEMTST:  LXI     H,1000H
0027 69 00        CLEARA:  MVI     A,00H
0029 30           STORE :  STAX    H+
002A 41                    INR     A
002B FD..                  JR      STORE
002C 74 7E 20     CHECK :  EQI     H,20H
002F F9                    JR      STORE
0030 C0                    JR      TSTEND
0031 69 AA        TSTEND:  MVI     A,0AAH
*
*SYMBOL                    ; DISPLAY TEST SYMBOL
SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS  SYM----ADRS
PUBLIC
CHECK -002C  CHGRG  0012  CLEARA 0027  EXCHG1 0010  EXCHG2 0022  MEMTST 0024
START  0000  STORE  0029  TSTEND 0031
*
```

(15) GO command (Break condition setting)

```
*;        (1) ADDRESS BREAK
*
*; < 1 >
*
*GO FROM 0 TILL 10H
EMULATION BEGUN
EMULATION TERMINATED, #0011
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01          START : MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10             EXCHG1: EXA                      00  00
*
*; < 2 >
*
*BA0 = 10H
*
*GO FROM 0 TILL BA0
EMULATION BEGUN
EMULATION TERMINATED, #0011
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01          START : MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10             EXCHG1: EXA                      00  00
*
```

```
*; < 3 >
*
*GO FROM 0 TILL 12H TO 16H
EMULATION BEGUN
EMULATION TERMINATED, #0014
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01        START  :  MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10           EXCHG1:   EXA                      00  00
0011 11                     EXX                      00  00
0012 69 10        CHGRG  :  MVI     A,10H            00  00
*
*; < 4 >
*
*BA1 = 12H TO 16H
*
*GO FROM 0 TILL BA1
EMULATION BEGUN
EMULATION TERMINATED, #0014
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01        START  :  MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10           EXCHG1:   EXA                      00  00
0011 11                     EXX                      00  00
0012 69 10        CHGRG  :  MVI     A,10H            00  00
```

```
*: < 5 >
*
*GO FROM 0 TILL 10H , 12H TO 16H
EMULATION BEGUN
EMULATION TERMINATED, #0011
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                    INSTRUCTION          PB  PA
0000 69 01        START : MVI     A,01H      00  00
0002 6A 02                MVI     B,02H      00  00
0004 6B 03                MVI     C,03H      00  00
0006 6C 04                MVI     D,04H      00  00
0008 6D 05                MVI     E,05H      00  00
000A 6E 06                MVI     H,06H      00  00
000C 6F 07                MVI     L,07H      00  00
000E 68 08                MVI     V,08H      00  00
0010 10           EXCHG1: EXA                00  00
*
*: < 6 >
*
*BA0
BA0=0010
*
*BA1
BA1=0012-0016
*
*GO FROM 0 TILL BA0 BA1
EMULATION BEGUN
EMULATION TERMINATED, #0011
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                    INSTRUCTION          PB  PA
0000 69 01        START : MVI     A,01H      00  00
0002 6A 02                MVI     B,02H      00  00
0004 6B 03                MVI     C,03H      00  00
0006 6C 04                MVI     D,04H      00  00
0008 6D 05                MVI     E,05H      00  00
000A 6E 06                MVI     H,06H      00  00
000C 6F 07                MVI     L,07H      00  00
000E 68 08                MVI     V,08H      00  00
0010 10           EXCHG1: EXA                00  00
*
*
```

```
*;        (2) DATA BREAK
*
*; < 1 >
*
*GO FROM .MEMTST TILL DATA = 2
EMULATION BEGUN
EMULATION TERMINATED, #002A
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0024 34 00 10     MEMTST: LXI     H,1000H            00  00
0027 69 00        CLEARA: MVI     A,00H              00  00
0029 3D           STORE : STAX    H+                 00  00
1000 00
002A 41                   INR     A                  00  00
002B FD                   JR      STORE              00  00
0029 3D           STORE : STAX    H+                 00  00
1001 01
002A 41                   INR     A                  00  00
002B FD                   JR      STORE              00  00
0029 3D           STORE : STAX    H+                 00  00
1002 02
*
*; < 2 >
*
*BDR = 2
*
*GO FROM .MEMTST TILL DATA
EMULATION BEGUN
EMULATION TERMINATED, #002A
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0024 34 00 10     MEMTST: LXI     H,1000H            00  00
0027 69 00        CLEARA: MVI     A,00H              00  00
0029 3D           STORE : STAX    H+                 00  00
1000 00
002A 41                   INR     A                  00  00
002B FD                   JR      STORE              00  00
0029 3D           STORE : STAX    H+                 00  00
1001 01
002A 41                   INR     A                  00  00
002B FD                   JR      STORE              00  00
0029 3D           STORE : STAX    H+                 00  00
1002 02
*
```

```
*; < 3 >
*
*GO FROM 0 TILL DATA = 6 READ
EMULATION BEGUN
EMULATION TERMINATED, #000C
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01          START : MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
*
*; < 4 >
*
*BDR = 6 READ
*
*GO FROM 0 TILL DATA
EMULATION BEGUN
EMULATION TERMINATED, #000C
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01          START : MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
*
```

```
*; < 5 >
*
*GO FROM .MEMTST TILL DATA = 0 WRITTEN
EMULATION BEGUN
EMULATION TERMINATED, #002A
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                    INSTRUCTION             PB  PA
0024 34 00 10    MEMTST: LXI     H,1000H         00  00
0027 69 00       CLEARA: MVI     A,00H           00  00
0029 3D          STORE : STAX    H+              00  00
1000 00
*
*; < 6 >
*
*BDR = 0 WRITTEN
*
*GO FROM .MEMTST TILL DATA
EMULATION BEGUN
EMULATION TERMINATED, #002A
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                    INSTRUCTION             PB  PA
0024 34 00 10    MEMTST: LXI     H,1000H         00  00
0027 69 00       CLEARA: MVI     A,00H           00  00
0029 3D          STORE : STAX    H+              00  00
1000 00
*
*; < 7 >
*
*GO FROM 0 TILL DATA = 69H WITH 12H
EMULATION BEGUN
EMULATION TERMINATED, #0014
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                    INSTRUCTION             PB  PA
0000 69 01       START : MVI     A,01H           00  00
0002 6A 02               MVI     B,02H           00  00
0004 6B 03               MVI     C,03H           00  00
0006 6C 04               MVI     D,04H           00  00
0008 6D 05               MVI     E,05H           00  00
000A 6E 06               MVI     H,06H           00  00
000C 6F 07               MVI     L,07H           00  00
000E 68 08               MVI     V,08H           00  00
0010 10          EXCHG1: EXA                     00  00
0011 11                  EXX                     00  00
0012 69 10       CHGRG : MVI     A,10H           00  00
*
```

```
*; < 8 >
*
*BDR = 69H WITH 12H
*
*GO FROM 0 TILL DATA
EMULATION BEGUN
EMULATION TERMINATED, #0014
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01        START  :  MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10           EXCHG1:  EXA                       00  00
0011 11                    EXX                       00  00
0012 69 10        CHGRG  :  MVI     A,10H            00  00
*
*; < 9 >
*
*GO FROM .MEMTST TILL DATA = 1 WRITTEN WITH 1100H TO 1200H
EMULATION BEGUN
EMULATION TERMINATED, #002A
*
*CAUSE
NORMAL BREAK
*
*PRINT -10
ADRS                        INSTRUCTION              PB  PA
002B FD                     JR      STORE            00  00
002C 74 7E 20     CHECK  :  EQI     H,20H            00  00
002F F9                     JR      STORE            00  00
0029 3D           STORE  :  STAX    H+               00  00
1100 00
002A 41                     INR     A                00  00
002B FD                     JR      STORE            00  00
0029 3D           STORE  :  STAX    H+               00  00
1101 01
*
```

```
*; < 10 >
*
*BDR = 1 WRITTEN WITH 1100H TO 1200H
*
*GO FROM .MEMTST TILL DATA
EMULATION BEGUN
EMULATION TERMINATED, #002A
*
*CAUSE
NORMAL BREAK
*
*PRINT -10
ADRS                    INSTRUCTION          PB  PA
002B FD                 JR      STORE        00  00
002C 74 7E 20   CHECK : EQI     H,20H        00  00
002F F9                 JR      STORE        00  00
0029 3D         STORE : STAX    H+           00  00
1100 00
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D         STORE : STAX    H+           00  00
1101 01
*
*
*;        (3) TIMER BREAK
*
*; < 1 >
*
*GO FROM 0 TILL TIMER = 1
EMULATION BEGUN
EMULATION TERMINATED, #0029
*
*CAUSE
TIMER BREAK
*
*PRINT -20
ADRS                    INSTRUCTION          PB  PA
002B FD                 JR      STORE        00  00
0029 3D         STORE : STAX    H+           00  00
11C0 C0
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D         STORE : STAX    H+           00  00
11C1 C1
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D         STORE : STAX    H+           00  00
11C2 C2
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D         STORE : STAX    H+           00  00
11C3 C3
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D         STORE : STAX    H+           00  00
11C4 C4
002A 41                 INR     A            00  00
*
```

```
*
*BTMR = 1
*
*GO FROM 0 TILL TIMER
EMULATION BEGUN
EMULATION TERMINATED, #002A
*
*CAUSE
TIMER BREAK
*
*PRINT -20
ADRS                    INSTRUCTION          PB  PA
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
1128 28
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
1129 29
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
112A 2A
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
112B 2B
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
112C 2C
*
*; < 3 >
*
*GO FROM 0 TILL TIMER = 2
EMULATION BEGUN
EMULATION TERMINATED, #0029
*
*CAUSE
TIMER BREAK
*
*PRINT -20
ADRS                    INSTRUCTION          PB  PA
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
11E5 E5
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
11E6 E6
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
11E7 E7
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
11E8 E8
002A 41                 INR     A            00  00
002B FD                 JR      STORE        00  00
0029 3D       STORE :   STAX    H+           00  00
11E9 E9
002A 41                 INR     A            00  00
```

```
*
*;        (4) LOOP COUNT BREAK
*
*; < 1 >
*
*GO FROM 0 TILL 0 TO 20H LOOP = 3
EMULATION BEGUN
EMULATION TERMINATED, #0004
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                    INSTRUCTION              PB  PA
0000 69 01        START : MVI     A,01H          00  00
0002 6A 02                MVI     B,02H          00  00
*
*; < 2 >
*
*BCNT = 3
*
*GO FROM 0 TILL DATA = 69H LOOP
EMULATION BEGUN
EMULATION TERMINATED, #0029
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                    INSTRUCTION              PB  PA
0000 69 01        START : MVI     A,01H          00  00
0002 6A 02                MVI     B,02H          00  00
0004 6B 03                MVI     C,03H          00  00
0006 6C 04                MVI     D,04H          00  00
0008 6D 05                MVI     E,05H          00  00
000A 6E 06                MVI     H,06H          00  00
000C 6F 07                MVI     L,07H          00  00
000E 68 08                MVI     V,08H          00  00
0010 10           EXCHG1: EXA                    00  00
0011 11                   EXX                    00  00
0012 69 10        CHGRG : MVI     A,10H          00  00
0014 6A 20                MVI     B,20H          00  00
0016 6B 30                MVI     C,30H          00  00
0018 6C 40                MVI     D,40H          00  00
001A 6D 50                MVI     E,50H          00  00
001C 6E 60                MVI     H,60H          00  00
001E 6F 70                MVI     L,70H          00  00
0020 68 80                MVI     V,80H          00  00
0022 10           EXCHG2: EXA                    00  00
0023 11                   EXX                    00  00
0024 34 00 10     MEMTST: LXI     H,1000H        00  00
0027 69 00        CLEARA: MVI     A,00H          00  00
*
```

```
*; < 3 >
*
*GO FROM .0 TILL DATA = 69H LOOP = 2
EMULATION BEGUN
EMULATION TERMINATED, #0014
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01        START  :  MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10           EXCHG1:   EXA                      00  00
0011 11                     EXX                      00  00
0012 69 10        CHGRG  :  MVI     A,10H            00  00
*
*
*;        (5) ADDRESS OR DATA BREAK
*
*GO FROM 0 TILL 10H DATA = 68H            ; ONLY ADDRESS BREAK
EMULATION BEGUN
EMULATION TERMINATED, #0011
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01        START  :  MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10           EXCHG1:   EXA                      00  00
*
*
```

```
*;          (6) ADDRESS OR TIMER BREAK
*
*GO FROM 0 TILL 10H TIMER = 1000H          ; ADDRESS BREAK
EMULATION BEGUN
EMULATION TERMINATED, #0011
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01          START : MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10             EXCHG1: EXA                      00  00
*
*GO FROM 0 TILL 4000H TIMER = 10           ; TIMER BREAK
EMULATION BEGUN
EMULATION TERMINATED, #0029
*
*CAUSE
TIMER BREAK
*
*PRINT -20
ADRS                        INSTRUCTION              PB  PA
002B FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
177D 7D
002A 41                     INR     A                00  00
002B FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
177E 7E
002A 41                     INR     A                00  00
002B FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
177F 7F
002A 41                     INR     A                00  00
002B FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
1780 80
002A 41                     INR     A                00  00
002B FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
1781 81
002A 41                     INR     A                00  00
*
*
```

```
*;        (7) DATA OR TIMER BREAK
*
*GO FROM 0 TILL DATA = 10H TIMER = 1000H ; DATA BREAK
EMULATION BEGUN
EMULATION TERMINATED, #0011
*
*CAUSE
NORMAL BREAK
*
*PRINT ALL
ADRS                        INSTRUCTION              PB  PA
0000 69 01          START : MVI     A,01H            00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10             EXCHG1: EXA                      00  00
*
*GO FROM 0 TILL DATA = 0F0H READ TIMER = 10   ; TIMER BREAK
EMULATION BEGUN
EMULATION TERMINATED, #0029
*
*CAUSE
TIMER BREAK
*
*PRINT -20
ADRS                        INSTRUCTION              PB  PA
0028 FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
17C9 C9
002A 41                     INR     A                00  00
0028 FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
17CA CA
002A 41                     INR     A                00  00
0028 FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
17CB CB
002A 41                     INR     A                00  00
0028 FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
17CC CC
002A 41                     INR     A                00  00
0028 FD                     JR      STORE            00  00
0029 3D             STORE : STAX    H+               00  00
17CD CD
002A 41                     INR     A                00  00
*
```

(16) STEP command

```
*LOAD :F1:IADEXC                ; LOAD TEST PROGRAM
*
*STEP FROM 0
EMULATION BEGUN
ADRS                        INSTRUCTION              PB  PA
0000 69 01        START : MVI     A,01H              00  00
EMULATION TERMINATED, #0002
*
*STEP
EMULATION BEGUN
ADRS                        INSTRUCTION              PB  PA
0002 6A 02                  MVI     B,02H            00  00
EMULATION TERMINATED, #0004
*
*STEP
EMULATION BEGUN
ADRS                        INSTRUCTION              PB  PA
0004 6B 03                  MVI     C,03H            00  00
EMULATION TERMINATED, #0006
*
*STEP FROM 0 COUNT 10
EMULATION BEGUN
ADRS                        INSTRUCTION              PB  PA
0000 69 01        START : MVI     A,01H              00  00
0002 6A 02                  MVI     B,02H            00  00
0004 6B 03                  MVI     C,03H            00  00
0006 6C 04                  MVI     D,04H            00  00
0008 6D 05                  MVI     E,05H            00  00
000A 6E 06                  MVI     H,06H            00  00
000C 6F 07                  MVI     L,07H            00  00
000E 68 08                  MVI     V,08H            00  00
0010 10           EXCHG1: EXA                      00  00
0011 11                   EXX                      00  00
EMULATION TERMINATED, #0012
*
*STEP COUNT 10
EMULATION BEGUN
ADRS                        INSTRUCTION              PB  PA
0012 69 10        CHGRG : MVI     A,10H              00  00
0014 6A 20                  MVI     B,20H            00  00
0016 6B 30                  MVI     C,30H            00  00
0018 6C 40                  MVI     D,40H            00  00
001A 6D 50                  MVI     E,50H            00  00
001C 6E 60                  MVI     H,60H            00  00
001E 6F 70                  MVI     L,70H            00  00
0020 68 80                  MVI     V,80H            00  00
0022 10           EXCHG2: EXA                      00  00
0023 11                   EXX                      00  00
EMULATION TERMINATED, #0024
*
```

```
*STEP ENABLE                    ; REGISTER DISPLAY
*
*STEP FROM 0
EMULATION BEGUN
ADRS                        INSTRUCTION          PB  PA
0000 69 01        START :   MVI    A,01H         00  00
 V=08  A=01  B=02  C=03  D=04  E=05  H=06  L=07 EA=0000 SP=3000 PC=0002
EMULATION TERMINATED, #0002
*
*STEP COUNT 10
EMULATION BEGUN
ADRS                        INSTRUCTION          PB  PA
0002 6A 02                  MVI    B,02H         00  00
 V=08  A=01  B=02  C=03  D=04  E=05  H=06  L=07 EA=0000 SP=3000 PC=0004
0004 6B 03                  MVI    C,03H         00  00
 V=08  A=01  B=02  C=03  D=04  E=05  H=06  L=07 EA=0000 SP=3000 PC=0006
0006 6C 04                  MVI    D,04H         00  00
 V=08  A=01  B=02  C=03  D=04  E=05  H=06  L=07 EA=0000 SP=3000 PC=0008
0008 6D 05                  MVI    E,05H         00  00
 V=08  A=01  B=02  C=03  D=04  E=05  H=06  L=07 EA=0000 SP=3000 PC=000A
000A 6E 06                  MVI    H,06H         00  00
 V=08  A=01  B=02  C=03  D=04  E=05  H=06  L=07 EA=0000 SP=3000 PC=000C
000C 6F 07                  MVI    L,07H         00  00
 V=08  A=01  B=02  C=03  D=04  E=05  H=06  L=07 EA=0000 SP=3000 PC=000E
000E 68 08                  MVI    V,08H         00  00
 V=08  A=01  B=02  C=03  D=04  E=05  H=06  L=07 EA=0000 SP=3000 PC=0010
0010 10           EXCHG1:   EXA                  00  00
 V=80  A=10  B=02  C=03  D=04  E=05  H=06  L=07 EA=0137 SP=3000 PC=0011
0011 11                     EXX                  00  00
 V=80  A=10  B=20  C=30  D=40  E=50  H=60  L=70 EA=0137 SP=3000 PC=0012
0012 69 10        CHGRG :   MVI    A,10H         00  00
 V=80  A=10  B=20  C=30  D=40  E=50  H=60  L=70 EA=0137 SP=3000 PC=0014
EMULATION TERMINATED, #0014
*
*STEP DISABLE
*
*STEP FROM 0
EMULATION BEGUN
ADRS                        INSTRUCTION          PB  PA
0000 69 01        START :   MVI    A,01H         00  00
EMULATION TERMINATED, #0002
*
*STEP
EMULATION BEGUN
ADRS                        INSTRUCTION          PB  PA
0002 6A 02                  MVI    B,02H         00  00
EMULATION TERMINATED, #0004
*
```

(17) Trace command

```
*LOAD :F1:IADEX2                ; LOAD TEST PROGRAM
*
*DASM .STMEM TO 27H
0000 69 0F        STMEM : MVI     A,0FH
0002 1A                   MOV     B,A
0003 1B                   MOV     C,A
0004 1C                   MOV     D,A
0005 34 00 10             LXI     H,1000H
0008 3D           STRMEM: STAX    H+
0009 51                   DCR     F
000A FD                   JR      STRMEM
0008 69 00        STPORT: MVI     A,00H
000D 4D D2                MOV     MA,A
000F 69 FF                MVI     A,0FFH
0011 4D D3                MOV     MB,A
0013 69 00                MVI     A,00H
0015 4D C0        OUTPA : MOV     PA,A
0017 41                   INR     F
0018 FC                   JR      OUTPA
0019 69 00                MVI     A,00H
001B 4D D3                MOV     MB,A
001D 69 FF                MVI     A,0FFH
001F 4D D2                MOV     MA,A
0021 69 00                MVI     A,00H
0023 4D C1        OUTPB : MOV     PB,A
0025 41                   INR     F
0026 FC                   JR      OUTPB
0027 54 00 00             JMP     STMEM
*;      (1) NON TRACE
*
*TRACE DISABLE
*
*GO FROM 0 TILL 8
EMULATION BEGUN
EMULATION TERMINATED, #0009
*
*PRINT ALL                       ; NON TRACE
ADRS                          INSTRUCTION            PB  PA
NON TRACE DATA
*
```

```
*
*;        (2) ADDRESS TRACE
*
*; < 1 >
*
*TRACE ENABLE 2 TO 4
*
*GO FROM 0 TILL 8
EMULATION BEGUN
EMULATION TERMINATED, #0009
*
*PRINT ALL                    ; 2H-4H TRACE
ADRS                      INSTRUCTION          PB  PA
0002 1A                   MOV     B,A          00  00
0003 1B                   MOV     C,A          00  00
0004 1C                   MOV     D,A          00  00
*
*; < 2 >
*
*TRACE DISABLE
*
*TRACE ENABLE 0 TO 8 WRITTEN
*
*GO FROM 0 TILL 8
EMULATION BEGUN
EMULATION TERMINATED, #0009
*
*PRINT ALL                    ; NON TRACE
ADRS                      INSTRUCTION          PB  PA
NON TRACE DATA
*
*; < 3 >
*
*TRACE DISABLE
*
*TRACE ENABLE 1000H TO 2000H WRITTEN
*
*GO FROM 0 TILL 0BH
EMULATION BEGUN
EMULATION TERMINATED, #000D
*
*TRACE = CYCLE
*
*PRINT ALL                          ; WRITE PALSE TRACE
ADRS  DATA  W  R  M1  PB  PA
1000   0F   1  0   0  00  00
1001   0E   1  0   0  00  00
1002   0D   1  0   0  00  00
1003   0C   1  0   0  00  00
1004   0B   1  0   0  00  00
1005   0A   1  0   0  00  00
1006   09   1  0   0  00  00
1007   08   1  0   0  00  00
1008   07   1  0   0  00  00
1009   06   1  0   0  00  00
100A   05   1  0   0  00  00
100B   04   1  0   0  00  00
100C   03   1  0   0  00  00
100D   02   1  0   0  00  00
100E   01   1  0   0  00  00
100F   00   1  0   0  00  00
*
```

```
*; < 4 >
*
*TRACE DISABLE
*
*TRACE ENABLE 2 TO 7 READ
*
*GO FROM 0 TILL 08H
EMULATION BEGUN
EMULATION TERMINATED, #000D
*
*TRACE = INSTRUCTION
*
*PRINT ALL                    ; 2H-7H ALL TRACE
ADRS                      INSTRUCTION          PB  PA
0002 1A                   MOV     B,A          00  00
0003 1B                   MOV     C,A          00  00
0004 1C                   MOV     D,A          00  00
0005 34 00 10             LXI     H,1000H      00  00
*
*
*;        (3) DATA TRACE
*
*; < 1 >
*
*TRACE DISABLE
*
*TRACE ENABLE DATA = 0
*
*GO FROM 0 TILL 08H
EMULATION BEGUN
EMULATION TERMINATED, #000D
*
*TRACE = CYCLE
*
*PRINT ALL
ADRS  DATA  W  R  M1  PB  PA
0006   00   0  1   0  00  00
1000   00   1  0   0  00  00
000C   00   0  1   0  00  00
*
*; < 2 >
*
*TRACE DISABLE
*
*TRACE ENABLE DATA = 0 READ
*
*GO FROM 0 TILL 08H
EMULATION BEGUN
EMULATION TERMINATED, #000D
*
*TRACE = CYCLE
*
*PRINT ALL
ADRS  DATA  W  R  M1  PB  PA
0006   00   0  1   0  00  00
000C   00   0  1   0  00  00
*
```

```
*; < 3 >
*
*TRACE DISABLE
*
*TRACE ENABLE DATA = 0 WRITTEN
*
*GO FROM 0 TILL 08H
EMULATION BEGUN
EMULATION TERMINATED, #0000
*
*TRACE = CYCLE
*
*PRINT ALL
ADRS  DATA  W  R  M1  PB  PA
1000   00   1  0   0  00  00
*
*; < 4 >
*
*TRACE DISABLE
*
*TRACE ENABLE DATA = 0EH WRITTEN WITH 1000H TO 100FH
*
*GO FROM 0 TILL 08H
EMULATION BEGUN
NO USER VCC
EMULATION TERMINATED, #0000
*
*TRACE = CYCLE
*
*PRINT ALL
ADRS  DATA  W  R  M1  PB  PA
1001   0E   1  0   0  00  00
*
*
*;        (4) PORTA TRACE
*
*TRACE DISABLE
*
*TRACE ENABLE PORTA = 10H
*
*GO FROM .STPORT TILL 27H
EMULATION BEGUN
NO USER VCC
EMULATION TERMINATED, #0000
*
*TRACE = CYCLE
*
*PRINT ALL
ADRS  DATA  W  R  M1  PB  PA
0017   41   0  1   1  00  10
0018   FC   0  1   1  00  10
0015   4D   0  1   1  00  10
0016   C0   0  1   0  00  10
*
*
```

```
*;        (5) PORTB TRACE
*
*TRACE DISABLE
*
*TRACE ENABLE PORTB = 20H
*
*GO FROM .STPORT TILL 27H
EMULATION BEGUN
NO USER VCC
EMULATION TERMINATED, #0000
*
*TRACE = CYCLE
*
*PRINT ALL
ADRS  DATA  W  R  M1  PB  PA
0025   41   0  1   1  20  FF
0026   FC   0  1   1  20  FF
0023   4D   0  1   1  20  FF
0024   C1   0  1   0  20  FF
*
*
*;        (6) EXTERNAL TRACE
*
*
*;        (7) FETCH TRACE
*
*TRACE DISABLE
*
*TRACE ENABLE FETCHED
*
*GO FROM 0 TILL 8
EMULATION BEGUN
NO USER VCC
EMULATION TERMINATED, #0009
*
*TRACE = CYCLE
*
*PRINT ALL
ADRS  DATA  W  R  M1  PB  PA
0000   69   0  1   1  FF  00
0002   1A   0  1   1  FF  00
0003   1B   0  1   1  FF  00
0004   1C   0  1   1  FF  00
0005   34   0  1   1  FF  00
0008   30   0  1   1  FF  00
*
```

A P P E N D I X A

Variations of IE-7809-I with respect to IE-87AD-I

A 1.1 IE-7809-I emulation chip is the uPD7807

A 1.2 IE-7809-I allows 8K (0-1FFFH) ROM to be mapped; since uPD7809 has 8K on chip ROM.

A 1.3 IE-7809-I System Software = IE7809

A 1.4 IE-7809-I self diagnostic is the same as IE-87AD-I but Port T is checked instead of analog inputs AN0-AN7.

A 1.5 IE-7809-I has WDM (R/W) mode register instead of ANM; WDM is watchdog timer mode register. Also an extra mode register MT (W) is used for Port T operation.
Mode registers display shows TMM, EOM, SMH and WDM for IE-7809-I

A 1.6 IE-7809-I Timer/Event extra commands

(i) ECNT displays contents of the timer/event counter up counter

(ii) ECPT0
ECPT1
ECPT0 command, displays the contents of timer/event counter capture register 0
ECPT1 command, displays the contents of timer/event counter capture register 1

A 1.7 IE-7809-I command format table vanience

(1) IE7809 command
: drive: IE7809

(10) mode registers
WDM instead of ANM
TMM, EOM, SMH and WDM displayed

(11) Port T can also be read

(15) ETM0/ETM1 = contents
ECNT ECPT0 ECPT1

A 1.8 IE7809 error code vanience

(a) error code 3DH: for IE-87AD
read IE7809

(b) error code 49H: for IE-87AD
read IE7809

(c) error code 4BH: CPU NOT STOP'
An error occurred because forced break could not be applied to the emulation CPU. If this error occurs, execute a hardware reset command.

A 1.9 IE-7809 command execution examples vanience

The command execution examples refer to IE-87AD-I operation and are, therefore, sometimes erroneous for IE-7809-I operation.
However, the basic principles of operation and command entries are the same for both IE-87AD-I and IE-7809-I.

APPENDIX B

μPD7809 Instruction Set and Pin Configuration

μPD7809/07 INSTRUCTION APPLICATION TABLE

μPD7809/07 machine language ⟷ mnemonic comparison table (alphabetic order)

ACI	A, byte	56××	DMOV	sr3, EA	48D2, 48D3
ACI	r, byte	7450××-7457××	DNE	EA, rp3	74ED-74EF
ACI	sr2, byte	6450-3, 5-7; 64D1, 3, 5××	DOFF	EA, rp3	74DD-74DF
ADC	A, r	60D8-60D7	DON	EA, rp3	74CD-74CF
ADC	r, A	6050-6057	DOR	EA, rp3	749D-749F
ADCW	wa	74D0××	DRLL	EA	48B4
ADCX	rpa	70D1-70D7	DRLR	EA	48B0
ADD	A, r	60C0-60C7	DSBB	EA, rp3	74F5-74F7
ADD	r, A	6040-6047	DSLL	EA	48A4
ADDNC	A, r	60A0-60A7	DSLR	EA	48A0
ADDNC	r, A	6020-6027	DSUB	EA, rp3	74E5-74E7
ADDNCW	wa	74A0××	DSUBNB	EA, rp3	74B5-74B7
ADDNCX	rpa	70A1-70A7	DXR	EA, rp3	7495-7497
ADDW	wa	74C0××	EADD	EA, r2	7041-7043
ADDX	rpa	70C1-70C7	EI		AA
ADI	A, byte	46××	EQA	A, r	60F8-60FF
ADI	r, byte	7440××-7447××	EQA	r, A	6078-607F
ADI	sr2, byte	6440-3, 5-7; 64C1, 3, 5××	EQAW	wa	74F8××
ADINC	A, byte	26××	EQAX	rpa	70F9-70FF
ADINC	r, byte	7420××-7427××	EQI	A, byte	77××
ADINC	sr2, byte	6420-3, 5-7; 64A1, 3, 5××	EQI	r, byte	7478××-747F××
ANA	A, r	6088-608F	EQI	sr5, byte	6478-B, D-F; 64F9, B, D, E××
ANA	r, A	6008-600F	EQIW	wa, byte	75××××
ANAW	wa	7488××	ESUB	EA, r2	7061-7063
ANAX	rpa	7089-708F	EXA		48AC
AND	CY, bit	31××	EX.H		48AE
ANI	A, byte	07××	EXR		48AD
ANI	r, byte	7408××-740F××	EXX		48AF
ANI	sr2, byte	6408-B, D-F; 6489, B, D××	GTA	A, r	60A8-60AF
ANIW	wa, byte	05××××	GTA	r, A	6028-602F
BLOCK	D+	10	GTAW	wa	74A8××
BLOCK	D−	11	GTAX	rpa	70A9-70AF
CALB		4829	GTI	A, byte	27××
CALF	word	7800-7FFF	GTI	r, byte	7428××-742F××
CALL	word	40××××	GTI	sr5, byte	6428-B, D-F; 64A9, B, D, E××
CALT	word	80-9F	GTIW	wa, byte	25××××
CLC		482A	HLT		483B
CLR	bit	5B	INR	r2	41-43
CMC		48AA	INRW	wa	20××
DAA		61	INX	EA	A8
DADC	EA, rp3	74D5-74D7	INX	rp	02, 12, 22, 32
DADD	EA, rp3	74C5-74C7	JB		21
DADDNC	EA, rp3	74A5-74A7	JEA		4828
DAN	EA, rp3	748D-748F	JMP	word	54××××
DCR	r2	51-53	JR	word	C0-FF
DCRW	wa	30××	JRE	word	4E00-4FFF
DCX	EA	A9	LBCD	word	701F××××
DCX	rp	03, 13, 23, 33	LDAW	wa	01××
DEQ	EA, rp3	74FD-74FF	LDAX	rpa2	29-2F, AB××, AC-AE, AF××
DGT	EA, rp3	74AD-74AF	LDEAX	rpa3	4882-5, B-F
DI		BA	LDED	word	702F××××
DIV	r2	483D-483F	LHLD	word	703F××××
DLT	EA, rp3	74BD-74BF	LSPD	word	700F××××
DMOV	EA, rp3	A5-A7	LTA	A, r	60B8-60BF
DMOV	EA, sr4	48C0-48C2	LTA	r, A	6038-603F
DMOV	rp3, EA	B5-B7	LTAW	wa	74B8××

Parts of this material may be changed without notice due to the introduction of new functions of products under development.

μPD7809/07 machine language ⟷ mnemonic comparison table (alphabetic order)

LTAX	rpa	70B9-70BF	RETI		62
LTI	A, byte	37×××	RETS		B9
LTI	r, byte	7438×××743F×××	RLD		4838
LTI	sr5, byte	6438-B, D-F; 64B9, B, D, E××	RLL	r2	4835-4837
LTIW	wa, byte	35××××	RLR	r2	4831-4833
LXI	rp2, byte	04, 14, 24, 34, 44××××	RRD		4839
MOV	A, r1	08-0F	SBB	A, r	60F0-60F7
MOV	A, sr1	4CC0-3, 5-7, 9, B, D, E; 4CD9; 4CE4	SBB	r, A	6070-6077
MOV	bit, CY	5A××	SBBW	wa	74F0××
MOV	CY, bit	5F××	SBBX	rpa	70F1-70F7
MOV	r1, A	18-1F	SBCD	word	701E××××
MOV	r, word	7068××××-706F××××	SBI	A, byte	76××
MOV	sr, A	4DC0-3, 5-7, 9-D; 4DD0-4, 7, 8, A, B; 4DE4, 5	SBI	r, byte	7470××-7477××
MOV	word, r	7078××××-707F××××	SBI	sr2, byte	6470-3, 5-7; 64F1, 3, 5××
MUL	r2	482D-482F	SDED	word	702E××××
MVI	r, byte	68××-6F××	SETB	bit	58××
MVI	sr2, byte	6400-3, 5-7; 6481, 3, 5××	SHLD	word	703E××××
MVIW	wa, byte	71××××	SK	bit	5D××
MVIX	rpa1, byte	49××-4B××	SK	f	480A-480C
NEA	A, r	60E8-60EF	SKIT	irf	4840-7, 9-C, F; 4854
NEA	r, A	6068-606F	SKN	bit	50××
NEAW	wa	74E8××	SKN	f	481A-481C
NEAX	rpa	70E9-70EF	SKNIT	irf	4860-7, 9-C, F; 4874
NEGA		483A	SLL	r2	4825-4827
NEI	A, byte	67××	SLLC	r2	4805-4807
NEI	r, byte	7468××-746F××	SLR	r2	4821-4823
NEI	sr5, byte	6468-B, D-F; 64E9, B, D, E××	SLRC	r2	4801-4803
NEIW	wa, byte	65××××	SOFTI		72
NOP		00	SSPD	word	700E××××
NOT	bit	59××	STAW	wa	63××
OFFA	A, r	60D8-60DF	STAX	rpa2	39-3F; BB××, BC-BE, BF××
OFFAW	wa	74D8××	STEAX	rpa3	4892-5, B-F
OFFAX	rpa	70D9-70DF	STC		482B
OFFI	A, byte	57××	SUB	A, r	60E0-60E7
OFFI	r, byte	7458××-745F××	SUB	r, A	6060-6067
OFFI	sr5, byte	6458-B, D-F; 64D9, B, D, E××	SUBNB	A, r	60B0-60B7
OFFIW	wa, byte	55××××	SUBNB	r, A	6030-6037
ONA	A, r	60C8-60CF	SUBNBW	wa	74B0××
ONAW	wa	74C8××	SUBNBX	rpa	70B1-70B7
ONAX	rpa	70C9-70CF	SUBW	wa	74E0××
ONI	A, byte	47××	SUBX	rpa	70E1-70E7
ONI	r, byte	7448××-744F××	SUI	A, byte	66××
ONI	sr5, byte	6448-B, D-F; 64C9, B, D, E××	SUI	r, byte	7460××-7467××
ONIW	wa, byte	45××××	SUI	sr2, byte	6460-3, 5-7; 64E1, 3, 5××
OR	CY, bit	5C××	SUINB	A, byte	36××
ORA	A, r	6098-609F	SUINB	r, byte	7430××-7437××
ORA	r, A	6018-601F	SUINB	sr2, byte	6430-3, 5-7; 64B1, 3, 5××
ORAW	wa	7498××	TABLE		48A8
ORAX	rpa	7099-709F	XOR	CY, bit	5E××
ORI	A, byte	17××	XRA	A, r	6090-6097
ORI	r, byte	7418××-741F××	XRA	r, A	6010-6017
ORI	sr2, byte	6418-B, D-F; 6499, B, D××	XRAW	wa	7490××
ORIW	wa, byte	15××××	XRAX	rpa	7091-7097
POP	rp1	A0-A4	XRI	A, byte	16××
PUSH	rp1	B0-B4	XRI	r, byte	7410××-7417××
RET		B8	XRI	sr2, byte	6410-3, 5-7; 6491, 3, 5××

μPD7809/07 machine language ⟷ mnemonic comparison table (2/4)

00	NOP		40	CALL	word	4864	SKNIT	F2	4DC7	MOV	MKL, A
01	LDAW	wa	41	INR	A	4865	SKNIT	FE0	4DC9	MOV	SMH, A
02	INX	SP	42	INR	B	4867	SKNIT	FEIN	4DCA	MOV	SML, A
03	DCX	SP	43	INR	C	4869	SKNIT	FSR	4DCB	MOV	EOM, A
04	LXI	SP, word	44	LXI	EA, word	486A	SKNIT	FST	4DCC	MOV	ETMM, A
05	ANIW	wa, byte	45	ONIW	wa, byte	486B	SKNIT	ER	4DCD	MOV	TMM, A
06			46	ADI	A, byte	486C	SKNIT	OV	4DD0	MOV	MM, A
07	ANI	A, byte	47	ONI	A, byte	486F	SKNIT	IEF2	4DD1	MOV	MCC, A
08	MOV	A, EAH	4801	SLRC	A	4874	SKNIT	SB	4DD2	MOV	MA, A
09	MOV	A, EAL	4802	SLRC	B	4882	LDEAX	D	4DD3	MOV	MB, A
0A	MOV	A, B	4803	SLRC	C	4883	LDEAX	H	4DD4	MOV	MC, A
0B	MOV	A, C	4805	SLLC	A	4884	LDEAX	D+ +	4DD7	MOV	MF, A
0C	MOV	A, D	4806	SLLC	B	4885	LDEAX	H+ +	4DD8	MOV	TXB, A
0D	MOV	A, E	4807	SLLC	C	488B	LDEAX	D+byte	4DDA	MOV	TM0, A
0E	MOV	A, H	480A	SK	CY	488C	LDEAX	H+A	4DDB	MOV	TM1, A
0F	MOV	A, L	480B	SK	HC	488D	LDEAX	H+B	4DE4	MOV	WDM
10	BLOCK	D-	480C	SK	Z	488E	LDEAX	H+EA	4DE5	MOV	MT
11	BLOCK	D−	481A	SKN	CY	488F	LDEAX	H+byte	4E××	JRE	word
12	INX	B	481B	SKN	HC	4892	STEAX	D			
13	DCX	B	481C	SKN	Z	4893	STEAX	H	4F××		
14	LXI	B, word	4821	SLR	A	4894	STEAX	D+ +	50	SKN	bit
15	ORIW	wa, byte	4822	SLR	B	4895	STEAX	H+ +	51	DCR	A
16	XRI	A, byte	4823	SLR	C	489B	STEAX	D+byte	52	DCR	B
17	ORI	A, byte	4825	SLL	A	489C	STEAX	H+A	53	DCR	C
18	MOV	EAH, A	4826	SLL	B	489D	STEAX	H+B	54	JMP	word
19	MOV	EAL, A	4827	SLL	C	489E	STEAX	H+EA	55	OFFIW	wa, byte
1A	MOV	B, A	4828	JEA		489F	STEAX	H+byte	56	ACI	A, byte
1B	MOV	C, A	4829	CALB		48A0	DSLR	EA	57	OFFI	A, byte
1C	MOV	D, A	482A	CLC		48A4	DSLL	EA	58	SETB	bit
1D	MOV	E, A	482B	STC		48A8	TABLE		59	NOT	bit
1E	MOV	H, A	482D	MUL	A	48AA	CMC		5A	MOV	bit, CY
1F	MOV	L, A	482E	MUL	B	48AC	EXA		5B	CLR	bit
20	INRW	wa	482F	MUL	C	48AD	EXR		5C	OR	CY, bit
21	JB		4831	RLR	A	48AE	EXH		5D	SK	bit
22	INX	D	4832	RLR	B	48AF	EXX		5E	XOR	CY, bit
23	DCX	D	4833	RLR	C	48B0	DRLR	EA	5F	MOV	CY, bit
24	LXI	D, word	4835	RLL	A	48B4	DRLL	EA	6008	ANA	V, A
25	GTIW	wa, byte	4836	RLL	B	48C0	DMOV	EA, ECNT	6009	ANA	A, A
26	ADINC	A, byte	4837	RLL	C	48C1	DMOV	EA, ECPT0	600A	ANA	B, A
27	GTI	A, byte	4838	RLD		48C2	DMOV	EA, ECPT1	600B	ANA	C, A
28			4839	RRD		48D2	DMOV	ETM0, EA	600C	ANA	D, A
29	LDAX	B	483A	NEGA		48D3	DMOV	ETM1, EA	600D	ANA	E, A
2A	LDAX	D	483B	HLT		49	MVIX	B, byte	600E	ANA	H, A
2B	LDAX	H	483D	DIV	A	4A	MVIX	D, byte	600F	ANA	L, A
2C	LDAX	D+	483E	DIV	B	4B	MVIX	H, byte	6010	XRA	V, A
2D	LDAX	H+	483F	DIV	C	4CC0	MOV	A, PA	6011	XRA	A, A
2E	LDAX	D−	4840	SKIT	FNMI	4CC1	MOV	A, PB	6012	XRA	B, A
2F	LDAX	H−	4841	SKIT	FT0	4CC2	MOV	A, PC	6013	XRA	C, A
30	DCRW	wa	4842	SKIT	FT1	4CC3	MOV	A, PD	6014	XRA	D, A
31	AND	CY, bit	4843	SKIT	F1	4CC5	MOV	A, PF	6015	XRA	E, A
32	INX	H	4844	SKIT	F2	4CC6	MOV	A, MKH	6016	XRA	H, A
33	DCX	H	4845	SKIT	FE0	4CC7	MOV	A, MKL	6017	XRA	L, A
34	LXI	H, word	4846	SKIT	FE1	4CC9	MOV	A, SMH	6018	ORA	V, A
35	LTIW	wa, byte	4847	SKIT	FEIN	4CCB	MOV	A, EOM	6019	ORA	A, A
36	SUINB	A, byte	4849	SKIT	FSR	4CCD	MOV	A, TMM	601A	ORA	B, A
37	LTI	A, byte	484A	SKIT	FST	4CCE	MOV	A, PT	601B	ORA	C, A
38			484B	SKIT	ER	4CD9	MOV	A, RXB	601C	ORA	D, A
39	STAX	B	484C	SKIT	OV	4CE4	MOV	A, WDM	601D	ORA	E, A
3A	STAX	D	484F	SKIT	IEF2	4DC0	MOV	PA, A	601E	ORA	H, A
3B	STAX	H	4854	SKIT	SB	4DC1	MOV	PB, A	601F	ORA	L, A
3C	STAX	D+	4860	SKNIT	FNMI	4DC2	MOV	PC, A	6020	ADDNC	V, A
3D	STAX	H+	4861	SKNIT	FT0	4DC3	MOV	PD, A	6021	ADDNC	A, A
3E	STAX	D−	4862	SKNIT	FT1	4DC5	MOV	PF, A	6022	ADDNC	B, A
3F	STAX	H−	4863	SKNIT	F1	4DC6	MOV	MKH, A	6023	ADDNC	C, A

μPD7809/07 machine language ⟷ mnemonic comparison table (1/4)

6024	ADDNC	D, A	6074	SBB	D, A	60BC	LTA	A, D	60FC	EQA	A, D
6025	ADDNC	E, A	6075	SBB	E, A	60BD	LTA	A, E	60FD	EQA	A, E
6026	ADDNC	H, A	6076	SBB	H, A	60BE	LTA	A, H	60FE	EQA	A, H
6027	ADDNC	L, A	6077	SBB	L, A	60BF	LTA	A, L	60FF	EQA	A, L
6028	GTA	V, A	6078	EQA	V, A	60C0	ADD	A, V	61	DAA	
6029	GTA	A, A	6079	EQA	A, A	60C1	ADD	A, A	62	RETI	
602A	GTA	B, A	607A	EQA	B, A	60C2	ADD	A, B	63	STAW	wa
602B	GTA	C, A	607B	EQA	C, A	60C3	ADD	A, C	6400	MVI	PA, byte
602C	GTA	D, A	607C	EQA	D, A	60C4	ADD	A, D	6401	MVI	PB, byte
602D	GTA	E, A	607D	EQA	E, A	60C5	ADD	A, E	6402	MVI	PC, byte
602E	GTA	H, A	607E	EQA	H, A	60C6	ADD	A, H	6403	MVI	PD, byte
602F	GTA	L, A	607F	EQA	L, A	60C7	ADD	A, L	6405	MVI	PF, byte
6030	SUBNB	V, A	6088	ANA	A, V	60C8	ONA	A, V	6406	MVI	MKH, byte
6031	SUBNB	A, A	6089	ANA	A, A	60C9	ONA	A, A	6407	MVI	MKL, byte
6032	SUBNB	B, A	608A	ANA	A, B	60CA	ONA	A, B	6408	ANI	PA, byte
6033	SUBNB	C, A	608B	ANA	A, C	60CB	ONA	A, C	6409	ANI	PB, byte
6034	SUBNB	D, A	608C	ANA	A, D	60CC	ONA	A, D	640A	ANI	PC, byte
6035	SUBNB	E, A	608D	ANA	A, E	60CD	ONA	A, E	640B	ANI	PD, byte
6036	SUBNB	H, A	608E	ANA	A, H	60CE	ONA	A, H	640D	ANI	PF, byte
6037	SUBNB	L, A	608F	ANA	A, L	60CF	ONA	A, L	640E	ANI	MKH, byte
6038	LTA	V, A	6090	XRA	A, V	60D0	ADC	A, V	640F	ANI	MKL, byte
6039	LTA	A, A	6091	XRA	A, A	60D1	ADC	A, A	6410	XRI	PA, byte
603A	LTA	B, A	6092	XRA	A, B	60D2	ADC	A, B	6411	XRI	PB, byte
603B	LTA	C, A	6093	XRA	A, C	60D3	ADC	A, C	6412	XRI	PC, byte
603C	LTA	D, A	6094	XRA	A, D	60D4	ADC	A, D	6413	XRI	PD, byte
603D	LTA	E, A	6095	XRA	A, E	60D5	ADC	A, E	6415	XRI	PF, byte
603E	LTA	H, A	6096	XRA	A, H	60D6	ADC	A, H	6416	XRI	MKH, byte
603F	LTA	L, A	6097	XRA	A, L	60D7	ADC	A, L	6417	XRI	MKL, byte
6040	ADD	V, A	6098	ORA	A, V	60D8	OFFA	A, V	6418	ORI	PA, byte
6041	ADD	A, A	6099	ORA	A, A	60D9	OFFA	A, A	6419	ORI	PB, byte
6042	ADD	B, A	609A	ORA	A, B	60DA	OFFA	A, B	641A	ORI	PC, byte
6043	ADD	C, A	609B	ORA	A, C	60DB	OFFA	A, C	641B	ORI	PD, byte
6044	ADD	D, A	609C	ORA	A, D	60DC	OFFA	A, D	641D	ORI	PF, byte
6045	ADD	E, A	609D	ORA	A, E	60DD	OFFA	A, E	641E	ORI	MKH, byte
6046	ADD	H, A	609E	ORA	A, H	60DE	OFFA	A, H	641F	ORI	MKL, byte
6047	ADD	L, A	609F	ORA	A, L	60DF	OFFA	A, L	6420	ADINC	PA, byte
6050	ADC	V, A	60A0	ADDNC	A, V	60E0	SUB	A, V	6421	ADINC	PB, byte
6051	ADC	A, A	60A1	ADDNC	A, A	60E1	SUB	A, A	6422	ADINC	PC, byte
6052	ADC	B, A	60A2	ADDNC	A, B	60E2	SUB	A, B	6423	ADINC	PD, byte
6053	ADC	C, A	60A3	ADDNC	A, C	60E3	SUB	A, C	6425	ADINC	PF, byte
6054	ADC	D, A	60A4	ADDNC	A, D	60E4	SUB	A, D	6426	ADINC	MKH, byte
6055	ADC	E, A	60A5	ADDNC	A, E	60E5	SUB	A, E	6427	ADINC	MKL, byte
6056	ADC	H, A	60A6	ADDNC	A, H	60E6	SUB	A, H	6428	GTI	PA, byte
6057	ADC	L, A	60A7	ADDNC	A, L	60E7	SUB	A, L	6429	GTI	PB, byte
6060	SUB	V, A	60A8	GTA	A, V	60E8	NEA	A, V	642A	GTI	PC, byte
6061	SUB	A, A	60A9	GTA	A, A	60E9	NEA	A, A	642B	GTI	PD, byte
6062	SUB	B, A	60AA	GTA	A, B	60EA	NEA	A, B	642D	GTI	PF, byte
6063	SUB	C, A	60AB	GTA	A, C	60EB	NEA	A, C	642E	GTI	MKH, byte
6064	SUB	D, A	60AC	GTA	A, D	60EC	NEA	A, D	642F	GTI	MKL, byte
6065	SUB	E, A	60AD	GTA	A, E	60ED	NEA	A, E	6430	SUINB	PA, byte
6066	SUB	H, A	60AE	GTA	A, H	60EE	NEA	A, H	6431	SUINB	PB, byte
6067	SUB	L, A	60AF	GTA	A, L	60EF	NEA	A, L	6432	SUINB	PC, byte
6068	NEA	V, A	60B0	SUBNB	A, V	60F0	SBB	A, V	6433	SUINB	PD, byte
6069	NEA	A, A	60B1	SUBNB	A, A	60F1	SBB	A, A	6435	SUINB	PF, byte
606A	NEA	B, A	60B2	SUBNB	A, B	60F2	SBB	A, B	6436	SUINB	MKH, byte
606B	NEA	C, A	60B3	SUBNB	A, C	60F3	SBB	A, C	6437	SUINB	MKL, byte
606C	NEA	D, A	60B4	SUBNB	A, D	60F4	SBB	A, D	6438	LTI	PA, byte
606D	NEA	E, A	60B5	SUBNB	A, E	60F5	SBB	A, E	6439	LTI	PB, byte
606E	NEA	H, A	60B6	SUBNB	A, H	60F6	SBB	A, H	643A	LTI	PC, byte
606F	NEA	L, A	60B7	SUBNB	A, L	60F7	SBB	A, L	643B	LTI	PD, byte
6070	SBB	V, A	60B8	LTA	A, V	60F8	EQA	A, V	643D	LTI	PF, byte
6071	SBB	A, A	60B9	LTA	A, A	60F9	EQA	A, A	643E	LTI	MKH, byte
6072	SBB	B, A	60BA	LTA	A, B	60FA	EQA	A, B	643F	LTI	MKL, byte
6073	SBB	C, A	60BB	LTA	A, C	60FB	EQA	A, C	6440	ADI	PA, byte

μPD7809/07 machine language ⟷ mnemonic comparison table (3/4)

6441	ADI	PB, byte	6499	ORI	SMH, byte	7041	EADD	EA, A	70B9	LTAX	B
6442	ADI	PC, byte	649B	ORI	EOM, byte	7042	EADD	EA, B	70BA	LTAX	D
6443	ADI	PD, byte	649D	ORI	TMM, byte	7043	EADD	EA, C	70BB	LTAX	H
6445	ADI	PF, byte	64A1	ADINC	SMH, byte	7061	ESUB	EA, A	70BC	LTAX	D+
6446	ADI	MKH, byte	64A3	ADINC	EOM, byte	7062	ESUB	EA, B	70BD	LTAX	H-
6447	ADI	MKL, byte	64A5	ADINC	TMM, byte	7063	ESUB	EA, C	70BE	LTAX	D-
6448	ONI	PA, byte	64A9	GTI	SMH, byte	7068	MOV	V, word	70BF	LTAX	H-
6449	ONI	PB, byte	64AB	GTI	EOM, byte	7069	MOV	A, word	70C1	ADDX	B
644A	ONI	PC, byte	64AD	GTI	TMM, byte	706A	MOV	B, word	70C2	ADDX	D
644B	ONI	PD, byte	64AE	GTI	PT, byte	706B	MOV	C, word	70C3	ADDX	H
644D	ONI	PF, byte	64B1	SUINB	SMH, byte	706C	MOV	D, word	70C4	ADDX	D+
644E	ONI	MKH, byte	64B3	SUINB	EOM, byte	706D	MOV	E, word	70C5	ADDX	H+
644F	ONI	MKL, byte	64B5	SUINB	TMM, byte	706E	MOV	H, word	70C6	ADDX	D-
6450	ACI	PA, byte	64B9	LTI	SMH, byte	706F	MOV	L, word	70C7	ADDX	H-
6451	ACI	PB, byte	64BB	LTI	EOM, byte	7078	MOV	word, V	70C9	ONAX	B
6452	ACI	PC, byte	64BD	LTI	TMM, byte	7079	MOV	word, A	70CA	ONAX	D
6453	ACI	PD, byte	64BE	LTI	PT, byte	707A	MOV	word, B	70CB	ONAX	H
6455	ACI	PF, byte	64C1	ADI	SMH, byte	707B	MOV	word, C	70CC	ONAX	D+
6456	ACI	MKH, byte	64C3	ADI	EOM, byte	707C	MOV	word, D	70CD	ONAX	H+
6457	ACI	MKL, byte	64C5	ADI	TMM, byte	707D	MOV	word, E	70CE	ONAX	D-
6458	OFFI	PA, byte	64C9	ONI	SMH, byte	707E	MOV	word, H	70CF	ONAX	H-
6459	OFFI	PB, byte	64CB	ONI	EOM, byte	707F	MOV	word, L	70D1	ADCX	B
645A	OFFI	PC, byte	64CD	ONI	TMM, byte	7089	ANAX	B	70D2	ADCX	D
645B	OFFI	PD, byte	64CE	ONI	PT, byte	708A	ANAX	D	70D3	ADCX	H
645D	OFFI	PF, byte	64D1	ACI	SMH, byte	708B	ANAX	H	70D4	ADCX	D+
645E	OFFI	MKH, byte	64D3	ACI	EOM, byte	708C	ANAX	D+	70D5	ADCX	H+
645F	OFFI	MKL, byte	64D5	ACI	TMM, byte	708D	ANAX	H+	70D6	ADCX	D-
6460	SUI	PA, byte	64D9	OFFI	SMH, byte	708E	ANAX	D-	70D7	ADCX	H-
6461	SUI	PB, byte	64DB	OFFI	EOM, byte	708F	ANAX	H-	70D9	OFFAX	B
6462	SUI	PC, byte	64DD	OFFI	TMM, byte	7091	XRAX	B	70DA	OFFAX	D
6463	SUI	PD, byte	64DE	OFFI	PT, byte	7092	XRAX	D	70DB	OFFAX	H
6465	SUI	PF, byte	64E1	SUI	SMH, byte	7093	XRAX	H	70DC	OFFAX	D+
6466	SUI	MKH, byte	64E3	SUI	EOM, byte	7094	XRAX	D+	70DD	OFFAX	H+
6467	SUI	MKL, byte	64E5	SUI	TMM, byte	7095	XRAX	H+	70DE	OFFAX	D-
6468	NEI	PA, byte	64E9	NEI	SMH, byte	7096	XRAX	D-	70DF	OFFAX	H-
6469	NEI	PB, byte	64EB	NEI	EOM, byte	7097	XRAX	H-	70E1	SUBX	B
646A	NEI	PC, byte	64ED	NEI	TMM, byte	7099	ORAX	B	70E2	SUBX	D
646B	NEI	PD, byte	64EE	NEI	PT, byte	709A	ORAX	D	70E3	SUBX	H
646D	NEI	PF, byte	64F1	SBI	SMH, byte	709B	ORAX	H	70E4	SUBX	D+
646E	NEI	MKH, byte	64F3	SBI	EOM, byte	709C	ORAX	D+	70E5	SUBX	H+
646F	NEI	MKL, byte	64F5	SBI	TMM, byte	709D	ORAX	H+	70E6	SUBX	D-
6470	SBI	PA, byte	64F9	EQI	SMH, byte	709E	ORAX	D-	70E7	SUBX	H-
6471	SBI	PB, byte	64FB	EQI	EOM, byte	709F	ORAX	H-	70E9	NEAX	B
6472	SBI	PC, byte	64FD	EQI	TMM, byte	70A1	ADDNCX	B	70EA	NEAX	D
6473	SBI	PD, byte	64FE	EQI	PT, byte	70A2	ADDNCX	D	70EB	NEAX	H
6475	SBI	PF, byte	65	NEIW	wa, byte	70A3	ADDNCX	H	70EC	NEAX	D+
6476	SBI	MKH, byte	66	SUI	A, byte	70A4	ADDNCX	D+	70ED	NEAX	H+
6477	SBI	MKL, byte	67	NEI	A, byte	70A5	ADDNCX	H+	70EE	NEAX	D-
6478	EQI	PA, byte	68	MVI	V, byte	70A6	ADDNCX	D-	70EF	NEAX	H-
6479	EQI	PB, byte	69	MVI	A, byte	70A7	ADDNCX	H-	70F1	SBBX	B
647A	EQI	PC, byte	6A	MVI	B, byte	70A9	GTAX	B	70F2	SBBX	D
647B	EQI	PD, byte	6B	MVI	C, byte	70AA	GTAX	D	70F3	SBBX	H
647D	EQI	PF, byte	6C	MVI	D, byte	70AB	GTAX	H	70F4	SBBX	D+
647E	EQI	MKH, byte	6D	MVI	E, byte	70AC	GTAX	D+	70F5	SBBX	H+
647F	EQI	MKL, byte	6E	MVI	H, byte	70AD	GTAX	H+	70F6	SBBX	D-
6481	MVI	SMH, byte	6F	MVI	L, byte	70AE	GTAX	D-	70F7	SBBX	H-
6483	MVI	EOM, byte	700E	SSPD	word	70AF	GTAX	H-	70F9	EQAX	B
6485	MVI	TMM, byte	700F	LSPD	word	70B1	SUBNBX	B	70FA	EQAX	D
6489	ANI	SMH, byte	701E	SBCD	word	70B2	SUBNBX	D	70FB	EQAX	H
648B	ANI	EOM, byte	701F	LBCD	word	70B3	SUBNBX	H	70FC	EQAX	D+
648D	ANI	TMM, byte	702E	SDED	word	70B4	SUBNBX	D+	70FD	EQAX	H+
6491	XRI	SMH, byte	702F	LDED	word	70B5	SUBNBX	H+	70FE	EQAX	D-
6493	XRI	EOM, byte	703E	SHLD	word	70B6	SUBNBX	D-	70FF	EQAX	H-
6495	XRI	TMM, byte	703F	LHLD	word	70B	SUBNBX	H-	71	MVIW	wa, byte

μPD7809/07 machine language ⟷ mnemonic comparison table (4/4)

7 2	SOFTI		7 4 4 7	ADI	L, byte	7 4 9 7	DXR	EA, H	A 2	POP	D
7 4 0 8	ANI	V, byte	7 4 4 8	ONI	V, byte	7 4 9 8	ORAW	wa	A 3	POP	H
7 4 0 9	ANI	A, byte	7 4 4 9	ONI	A, byte	7 4 9 D	DOR	EA, B	A 4	POP	EA
7 4 0 A	ANI	B, byte	7 4 4 A	ONI	B, byte	7 4 9 E	DOR	EA, D	A 5	DMOV	EA, B
7 4 0 B	ANI	C, byte	7 4 4 B	ONI	C, byte	7 4 9 F	DOR	EA, H	A 6	DMOV	EA, D
7 4 0 C	ANI	D, byte	7 4 4 C	ONI	D, byte	7 4 A 0	ADDNCW	wa	A 7	DMOV	EA, H
7 4 0 D	ANI	E, byte	7 4 4 D	ONI	E, byte	7 4 A 5	DADDNC	EA, B	A 8	INX	EA
7 4 0 E	ANI	H, byte	7 4 4 E	ONI	H, byte	7 4 A 6	DADDNC	EA, D	A 9	DCX	EA
7 4 0 F	ANI	L, byte	7 4 4 F	ONI	L, byte	7 4 A 7	DADDNC	EA, H	A A	EI	
7 4 1 0	XRI	V, byte	7 4 5 0	ACI	V, byte	7 4 A 8	GTAW	wa	A B	LDAX,	D+byte
7 4 1 1	XRI	A, byte	7 4 5 1	ACI	A, byte	7 4 A D	DGT	EA, B	A C	LDAX	H+A
7 4 1 2	XRI	B, byte	7 4 5 2	ACI	B, byte	7 4 A E	DGT	EA, D	A D	LDAX	H+B
7 4 1 3	XRI	C, byte	7 4 5 3	ACI	C, byte	7 4 A F	DGT	EA, H	A E	LDAX	H+EA
7 4 1 4	XRI	D, byte	7 4 5 4	ACI	D, byte	7 4 B 0	SUBNBW	wa	A F	LDAX	H+byte
7 4 1 5	XRI	E, byte	7 4 5 5	ACI	E, byte	7 4 B 5	DSUBNB	EA, B	B 0	PUSH	V
7 4 1 6	XRI	H, byte	7 4 5 6	ACI	H, byte	7 4 B 6	DSUBNB	EA, D	B 1	PUSH	B
7 4 1 7	XRI	L, byte	7 4 5 7	ACI	L, byte	7 4 B 7	DSUBNB	EA, H	B 2	PUSH	D
7 4 1 8	ORI	V, byte	7 4 5 8	OFFI	V, byte	7 4 B 8	LTAW	wa	B 3	PUSH	H
7 4 1 9	ORI	A, byte	7 4 5 9	OFFI	A, byte	7 4 B D	DLT	EA, B	B 4	PUSH	EA
7 4 1 A	ORI	B, byte	7 4 5 A	OFFI	B, byte	7 4 B E	DLT	EA, D	B 5	DMOV	B, EA
7 4 1 B	ORI	C, byte	7 4 5 B	OFFI	C, byte	7 4 B F	DLT	EA, H	B 6	DMOV	D, EA
7 4 1 C	ORI	D, byte	7 4 5 C	OFFI	D, byte	7 4 C 0	ADDW	wa	B 7	DMOV	H, EA
7 4 1 D	ORI	E, byte	7 4 5 D	OFFI	E, byte	7 4 C 5	DADD	EA, B	B 8	RET	
7 4 1 E	ORI	H, byte	7 4 5 E	OFFI	H, byte	7 4 C 6	DADD	EA, D	B 9	RETS	
7 4 1 F	ORI	L, byte	7 4 5 F	OFFI	L, byte	7 4 C 7	DADD	EA, H	B A	DI	
7 4 2 0	ADINC	V, byte	7 4 6 0	SUI	V, byte	7 4 C 8	ONAW	wa	B B	STAX	D+byte
7 4 2 1	ADINC	A, byte	7 4 6 1	SUI	A, byte	7 4 C D	DON	EA, B	B C	STAX	H+A
7 4 2 2	ADINC	B, byte	7 4 6 2	SUI	B, byte	7 4 C E	DON	EA, D	B D	STAX	H+B
7 4 2 3	ADINC	C, byte	7 4 6 3	SUI	C, byte	7 4 C F	DON	EA, H	B E	STAX	H+EA
7 4 2 4	ADINC	D, byte	7 4 6 4	SUI	D, byte	7 4 D 0	ADCW	wa	B F	STAX	H+byte
7 4 2 5	ADINC	E, byte	7 4 6 5	SUI	E, byte	7 4 D 5	DADC	EA, B	C 0	JR	word
7 4 2 6	ADINC	H, byte	7 4 6 6	SUI	H, byte	7 4 D 6	DADC	EA, D	\		
7 4 2 7	ADINC	L, byte	7 4 6 7	SUI	L, byte	7 4 D 7	DADC	EA, H	F F		
7 4 2 8	GTI	V, byte	7 4 6 8	NEI	V, byte	7 4 D 8	OFFAW	wa			
7 4 2 9	GTI	A, byte	7 4 6 9	NEI	A, byte	7 4 D D	DOFF	EA, B			
7 4 2 A	GTI	B, byte	7 4 6 A	NEI	B, byte	7 4 D E	DOFF	EA, D			
7 4 2 B	GTI	C, byte	7 4 6 B	NEI	C, byte	7 4 D F	DOFF	EA, H			
7 4 2 C	GTI	D, byte	7 4 6 C	NEI	D, byte	7 4 E 0	SUBW	wa			
7 4 2 D	GTI	E, byte	7 4 6 D	NEI	E, byte	7 4 E 5	DSUB	EA, B			
7 4 2 E	GTI	H, byte	7 4 6 E	NEI	H, byte	7 4 E 6	DSUB	EA, D			
7 4 2 F	GTI	L, byte	7 4 6 F	NEI	L, byte	7 4 E 7	DSUB	EA, H			
7 4 3 0	SUINB	V, byte	7 4 7 0	SBI	V, byte	7 4 E 8	NEAW	wa			
7 4 3 1	SUINB	A, byte	7 4 7 1	SBI	A, byte	7 4 E D	DNE	EA, B			
7 4 3 2	SUINB	B, byte	7 4 7 2	SBI	B, byte	7 4 E E	DNE	EA, D			
7 4 3 3	SUINB	C, byte	7 4 7 3	SBI	C, byte	7 4 E F	DNE	EA, H			
7 4 3 4	SUINB	D, byte	7 4 7 4	SBI	D, byte	7 4 F 0	SBBW	wa			
7 4 3 5	SUINB	E, byte	7 4 7 5	SBI	E, byte	7 4 F 5	DSBB	EA, B			
7 4 3 6	SUINB	H, byte	7 4 7 6	SBI	H, byte	7 4 F 6	DSBB	EA, D			
7 4 3 7	SUINB	L, byte	7 4 7 7	SBI	L, byte	7 4 F 7	DSBB	EA, H			
7 4 3 8	LTI	V, byte	7 4 7 8	EQI	V, byte	7 4 F 8	EQAW	wa			
7 4 3 9	LTI	A, byte	7 4 7 9	EQI	A, byte	7 4 F D	DEQ	EA, B			
7 4 3 A	LTI	B, byte	7 4 7 A	EQI	B, byte	7 4 F E	DEQ	EA, D			
7 4 3 B	LTI	C, byte	7 4 7 B	EQI	C, byte	7 4 F F	DEQ	EA, H			
7 4 3 C	LTI	D, byte	7 4 7 C	EQI	D, byte	7 5	EQIW	wa, byte			
7 4 3 D	LTI	E, byte	7 4 7 D	EQI	E, byte	7 6	SBI	A, byte			
7 4 3 E	LTI	H, byte	7 4 7 E	EQI	H, byte	7 7	EQI	A, byte			
7 4 3 F	LTI	L, byte	7 4 7 F	EQI	L, byte	7 8	CALF	word			
7 4 4 0	ADI	V, byte	7 4 8 8	ANAW	wa	\					
7 4 4 1	ADI	A, byte	7 4 8 D	DAN	EA, B	7 F					
7 4 4 2	ADI	B, byte	7 4 8 E	DAN	EA, D	8 0	CALT	word			
7 4 4 3	ADI	C, byte	7 4 8 F	DAN	EA, H	\					
7 4 4 4	ADI	D, byte	7 4 9 0	XRAW	wa	9 F					
7 4 4 5	ADI	E, byte	7 4 9 5	DXR	EA, B	A 0	POP	V			
7 4 4 6	ADI	H, byte	7 4 9 6	DXR	EA, D	A 1	POP	B			

μPD7809/07 INSTRUCTION SET

Operand format/description

Format	Description
r r1 r2	V, A, B, C, D, E, H, L EAH, EAL, B, C, D, E, H, L A, B, C
sr sr1 sr2 sr3 sr4 sr5	PA, PB, PC, PD, PF, MKH, MKL, SMH, SML, EOM, ETMM, TMM, MM, MCC, MA, MB, MC, MF, TXB, TM0, TM1, WDM, MT PA, PB, PC, PD, PF, MKH, MKL, SMH, EOM, TMM, PT, RXB, WDM PA, PB, PC, PD, PF, MKH, MKL, SMH, EOM, TMM ETM0, ETM1 ECNT, ECPT0, ECPT1 PA, PB, PC, PD, PF, MKH, MKL, SMH, EOM, TMM, PT
rp rp1 rp2 rp3	SP, B, D, H V, B, D, H, EA SP, B, D, H, EA B, D, H
rpa rpa1 rpa2 rpa3	B, D, H, D+, H+, D−, H− B, D, H B, D, H, D+, H+, D−, H−, D+byte, H+A, H+B, H+EA, H+byte D, H, D++, H++, D+byte, H+A, H+B, H+EA, H+byte
wa	8-bit immediate data
word byte	16-bit immediate data 8-bit immediate data
bit	8-bit immediate data
f	CY, HC, Z
irf	FNMI, FT0, FT1, F1, F2, FE0, FE1, FEIN, FSR, FST, ER, OV, IFE2, SB

Remarks

1. sr~sr5(special register)

PA	: PORT A	ECPT0	: TIMER/EVENT COUNTER CAPTURE0
PB	: PORT B	ECPT1	: TIMER/EVENT COUNTER CAPTURE1
PC	: PORT C	ETMM	: TIMER/ EVENT COUNTER MODE
PD	: PORT D	EOM	: TIMER/ EVENT COUNTER OUTPUT MODE
PF	: PORT F	WDM	: WATCHDOG TIMER MODE
PT	: PORT T	TXB	: Tx BUFFER
MA	: MODE A	RXB	: Rx BUFFER
MB	: MODE B	SMH	: SERIAL MODE High
MC	: MODE C	SML	: SERIAL MODE Low
MCC	: MODE CONTROL C	MKH	: MASK High
MF	: MODE F	MKL	: MASK Low
MM	: MEMORY MAPPING		
MT	: MODE T		
TM0	: TIMER REG0		
TM1	: TIMER REG1		
TMM	: TIMER MODE		
ETM0	: TIMER/ EVENT COUNTER REG0		
ETM1	: TIMER/ EVENT COUNTER REG1		
ECNT	: TIMER/ EVENT COUNTER UPCOUNTER		

2. rp~rp3(register pair)

SP	: STACK POINTER
B	: BC
D	: DE
H	: HL
V	: VA
EA	: EXTENDED ACCUMULATOR

3. rpa~rpa3(rp addressing)

B	: (BC)
D	: (DE)
H	: (HL)
D+	: (DE)+
H+	: (HL)+
D−	: (DE)−
H−	: (HL)−
D+ +	: (DE)+*
H+ +	: (HL)+*
D+byte	: (DE+byte)
H+A	: (HL+A)
H+B	: (HL+B)
H+EA	: (HL+EA)
H+byte	: (HL+byte)

4. f(flag)

CY	: CARRY
HC	: HALF CARRY
Z	: ZERO

5. irf(interrupt flag)

FNMI	: INTFNMI
FT0	: INTFT0
FT1	: INTFT1
F1	: INTF1
F2	: INTF2
FE0	: INTFE0
FE1	: INTFE1
FEIN	: INTFEIN
FSR	: INTFSR
FST	: INTFST
ER	: ERROR
OV	: OVERFLOW
IEF2	: INTERRUPT ENABLE F/F2
SB	: STANDBY

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
8-bit data transfer instruction	MOV	r1, A	1	4	r1←A			
		A, r1	1	4	A←r1			
		sr, A	2	10	sr←A			
		A, sr1	2	10	A←sr1			
		r, word	4	17	r←(word)			
		word, r	4	17	(word)←r			
	MVI	r, byte	2	7	r←byte			
		sr2, byte	3	14	sr2←byte			
	MVIW	wa, byte	3	13	(V, wa)←byte			
	MVIX	rpa1, byte	2	10	(rpa1)←byte			
	STAW	wa	2	10	(V, wa)←A			
	LDAW	wa	2	10	A←(V, wa)			
	STAX	rpa2	1	7	(rpa2)←A			
	LDAX	rpa2	1	7	A←(rpa2)			
	EXX		2	8	B↔B, C↔C, D↔D E↔E, H↔H, L↔L			
	EXA		2	8	V, A↔V, A, EA↔EA			
	EXH		2	8	H, L↔H', L'			
	EXR		2	8	A, B, C, D, E, H, L, EA ↔A, B, C, D, E, H, L, EA			
	BLOCK	D+	1	13 (C+1) End if borrow	(DE)+←(HL)+, C←C−1			
		D−	1	13 (C+1) End if borrow	(DE)−←(HL)−, C←C−1			
16-bit data transfer instruction	DMOV	rp3, EA	1	4	rp3₁←EAL, rp3ₘ← EAH			
		EA, rp3	1	4	EAL←rp3₁, EAH←rp3ₘ			
		sr3, EA	2	14	sr3←EA			
		EA, sr4	2	14	EA←sr4			
	SBCD	word	4	20	(word)←C, (word+1)←B			
	SDED	word	4	20	(word)←E, (word+1)←D			
	SHLD	word	4	20	(word)←L, (word+1)←H			
	SSPD	word	4	20	(word)←SP₁, (word+1)←SPₕ			
	STEAX	rpa3	2	9	(rpa3)←EAL, (rpa3+1)←EAH			
	LBCD	word	4	20	C←(word), B←(word+1)			
	LDED	word	4	20	E←(word), D←(word+1)			
	LHLD	word	4	20	L←(word), H←(word+1)			
	LSPD	word	4	20	SP₁←(word), SPₕ←(word+1)			
	LDEAX	rpa3	2	9	EAL←(rpa3), EAH←(rpa3+1)			
	PUSH	rp1	1	13	(SP−1)←rp1ₕ (SP−2)←rp1₁, SP←SP−2			
	POP	rp1	1	10	rp1₁←(SP) rp1ₕ←(SP+1), SP←SP−2			
	LXI	rp2, word	3	10	rp2←word			
	TABLE		2	17	C←(PC+3+A) B←(PC+3+A+1)			

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
8-bit operation instruction (register)	ADD	A, r	2	8	A←A+r		↕	↕
		r, A	2	8	r←r+A		↕	↕
	ADC	A, r	2	8	A←A+r+CY		↕	↕
		r, A	2	8	r←r+A+CY		↕	↕
	ADDNC	A, r	2	8	A←A+r	No Carry	↕	↕
		r, A	2	8	r←r+A	No Carry	↕	↕
	SUB	A, r	2	8	A←A−r		↕	↕
		r, A	2	8	r←r−A		↕	↕
	SBB	A, r	2	8	A←A−r−CY		↕	↕
		r, A	2	8	r←r−A−CY		↕	↕
	SUBNB	A, r	2	8	A←A−r	No Borrow	↕	↕
		r, A	2	8	r←r−A	No Borrow	↕	↕
	ANA	A, r	2	8	A←A∧r			↕
		r, A	2	8	r←r∧A			↕
	ORA	A, r	2	8	A←A∨r			↕
		r, A	2	8	r←r∨A			↕
	XRA	A, r	2	8	A←A⊻r			↕
		r, A	2	8	r←r⊻A			↕
	GTA	A, r	2	8	A−r−1	No Borrow	↕	↕
		r, A	2	8	r−A−1	No Borrow	↕	↕
	LTA	A, r	2	8	A−r	Borrow	↕	↕
		r, A	2	8	r−A	Borrow	↕	↕
	NEA	A, r	2	8	A−r	No Zero	↕	↕
		r, A	2	8	r−A	No Zero	↕	↕
	EQA	A, r	2	8	A−r	Zero	↕	↕
		r, A	2	8	r−A	Zero	↕	↕
	ONA	A, r	2	8	A∧r	No Zero		↕
	OFFA	A, r	2	8	A∧r	Zero		↕
Immediate date operation instruction	ADDX	rpa	2	11	A←A+(rpa)		↕	↕
	ADCX	rpa	2	11	A←A+(rpa)+CY		↕	↕
	ADDNCX	rpa	2	11	A←A+(rpa)	No Carry	↕	↕
	SUBX	rpa	2	11	A←A−(rpa)		↕	↕
	SBBX	rpa	2	11	A←A−(rpa)−CY		↕	↕
	SUBNBX	rpa	2	11	A←A−(rpa)	No Borrow	↕	↕
	ANAX	rpa	2	11	A←A∧(rpa)			↕
	ORAX	rpa	2	11	A←A∨(rpa)			↕
	XRAX	rpa	2	11	A←A⊻(rpa)			↕
	GTAX	rpa	2	11	A−(rpa)−1	No Borrow	↕	↕
	LTAX	rpa	2	11	A−(rpa)	Borrow	↕	↕
	NEAX	rpa	2	11	A−(rpa)	No Zero	↕	↕
	EQAX	rpa	2	11	A−(rpa)	Zero	↕	↕
	ONAX	rpa	2	11	A∧(rpa)	No Zero		↕
	OFFAX	rpa	2	11	A∧(rpa)	Zero		↕

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
Immediate data operation instruction	ADI	A, byte	2	7	A←A + byte		↕	↕
		r, byte	3	11	r←r + byte		↕	↕
		sr2, byte	3	20	sr2←sr2 + byte		↕	↕
	ACI	A, byte	2	7	A←A + byte + CY		↕	↕
		r, byte	3	11	r←r + byte + CY		↕	↕
		sr2, byte	3	20	sr2←sr2 + byte + CY		↕	↕
	ADINC	A, byte	2	7	A←A + byte	No Carry	↕	↕
		r, byte	3	11	r←r + byte	No Carry	↕	↕
		sr2, byte	3	20	sr2←sr2 + byte	No Carry	↕	↕
	SUI	A, byte	2	7	A←A − byte		↕	↕
		r, byte	3	11	r←r − byte		↕	↕
		sr2, byte	3	20	sr2←sr2 − byte		↕	↕
	SBI	A, byte	2	7	A←A − byte − CY		↕	↕
		r, byte	3	11	r←r − byte − CY		↕	↕
		sr2, byte	3	20	sr2←sr2 − byte − CY		↕	↕
	SUINB	A, byte	2	7	A←A − byte	No Borrow	↕	↕
		r, byte	3	11	r←r − byte	No Borrow	↕	↕
		sr2, byte	3	20	sr2←sr2 − byte	No Borrow	↕	↕
	ANI	A, byte	2	7	A←A∧byte			↕
		r, byte	3	11	r←r∧byte			↕
		sr2, byte	3	20	sr2←sr2∧byte			↕
	ORI	A, byte	2	7	A←A∨byte			↕
		r, byte	3	11	r←r∨byte			↕
		sr2, byte	3	20	sr2←sr2∨byte			↕
	XRI	A, byte	2	7	A←A∀byte			↕
		r, byte	3	11	r←r∀byte			↕
		sr2, byte	3	20	sr2←sr2∀byte			↕
	GTI	A, byte	2	7	A − byte − 1	No Borrow	↕	↕
		r, byte	3	11	r − byte − 1	No Borrow	↕	↕
		sr5, byte	3	14	sr5 − byte − 1	No Borrow	↕	↕
	LTI	A, byte	2	7	A − byte	Borrow	↕	↕
		r, byte	3	11	r − byte	Borrow	↕	↕
		sr5, byte	3	14	sr5 − byte	Borrow	↕	↕
	NEI	A, byte	2	7	A − byte	No Zero	↕	↕
		r, byte	3	11	r − byte	No Zero	↕	↕
		sr5, byte	3	14	sr5 − byte	No Zero	↕	↕
	EQI	A, byte	2	7	A − byte	Zero	↕	↕
		r, byte	3	11	r − byte	Zero	↕	↕
		sr5, byte	3	14	sr5 − byte	Zero	↕	↕
	ONI	A, byte	2	7	A∧byte	No Zero		↕
		r, byte	3	11	r∧byte	No Zero		↕
		sr5, byte	3	14	sr5∧byte	No Zero		↕
	OFFI	A, byte	2	7	A∧byte	Zero		↕
		r, byte	3	11	r∧byte	Zero		↕
		sr5, byte	3	14	sr5∧byte	Zero		↕

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
Working register operation instruction	ADDW	wa	3	14	A←A + (V, wa)		↕	↕
	ADCW	wa	3	14	A←A + (V, wa) + CY		↕	↕
	ADDNCW	wa	3	14	A←A + (V, wa)	No Carry	↕	↕
	SUBW	wa	3	14	A←A − (V, wa)		↕	↕
	SBBW	wa	3	14	A←A − (V, wa) − CY		↕	↕
	SUBNBW	wa	3	14	A←A − (V, wa)	No Borrow	↕	↕
	ANAW	wa	3	14	A←A∧(V, wa)			↕
	ORAW	wa	3	14	A←A∨(V, wa)			↕
	XRAW	wa	3	14	A←A∀(V, wa)			↕
	GTAW	wa	3	14	A − (V, wa) − 1	No Borrow	↕	↕
	LTAW	wa	3	14	A − (V, wa)	Borrow	↕	↕
	NEAW	wa	3	14	A − (V, wa)	No Zero	↕	↕
	EQAW	wa	3	14	A − (V, wa)	Zero	↕	↕
	ONAW	wa	3	14	A∧(V, wa)	No Zero		↕
	OFFAW	wa	3	14	A∧(V, wa)	Zero		↕
	ANIW	wa, byte	3	19	(V, wa)←(V,wa)∧byte			↕
	ORIW	wa, byte	3	19	(V, wa)←(V, wa)∨byte			↕
	GTIW	wa, byte	3	13	(V, wa) − byte − 1	No Borrow	↕	↕
	LTIW	wa, byte	3	13	(V, wa) − byte	Borrow	↕	↕
	NEIW	wa, byte	3	13	(V, wa) − byte	No Zero	↕	↕
	EQIW	wa, byte	3	13	(V, wa) − byte	Zero	↕	↕
	ONIW	wa, byte	3	13	(V, wa)∧byte	No Zero		↕
	OFFIW	wa, byte	3	13	(V, wa)∧byte	Zero		↕
16-bit operation instruction	EADD	EA, r2	2	11	EA←EA + r2		↕	↕
	DADD	EA, rp3	2	11	EA←EA + rp3		↕	↕
	DADC	EA, rp3	2	11	EA←EA + rp3 + CY		↕	↕
	DADDNC	EA, rp3	2	11	EA←EA + rp3	No Carry	↕	↕
	ESUB	EA, r2	2	11	EA←EA − r2		↕	↕
	DSUB	EA, rp3	2	11	EA←EA − rp3		↕	↕
	DSBB	EA, rp3	2	11	EA←EA − rp3 − CY		↕	↕
	DSUBNB	EA, rp3	2	11	EA←EA − rp3	No Borrow	↕	↕
	DAN	EA, rp3	2	11	EA←EA∧rp3			↕
	DOR	EA, rp3	2	11	EA←EA∨rp3			↕
	DXR	EA, rp3	2	11	EA←EA∀rp3			↕
	DGT	EA, rp3	2	11	EA − rp3 − 1	No Borrow	↕	↕
	DLT	EA, rp3	2	11	EA − rp3	Borrow	↕	↕
	DNE	EA, rp3	2	11	EA − rp3	No Zero	↕	↕
	DEQ	EA, rp3	2	11	EA − rp3	Zero	↕	↕
	DON	EA, rp3	2	11	EA∧rp3	No Zero		↕
	DOFF	EA, rp3	2	11	EA∧rp3	Zero		↕
Multiplication/devision instruction	MUL	r2	2	32	EA←A × r2			
	DIV	r2	2	59	EA←EA÷r2, r2←reset			

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
Pseudoinstructions	INR	r2	1	4	r2←r2+1	Carry		↕
	INRW	wa	2	16	(V, wa)←(V, wa)+1	Carry		↕
	INX	rp	1	7	rp←rp+1			
		EA	1	7	EA←EA+1			
	DCR	r2	1	4	r2←r2−1	Borrow		↕
	DCRW	wa	2	16	(V, wa)←(V, wa)−1	Borrow		↕
	DCX	rp	1	7	rp←rp−1			
		EA	1	7	EA←EA−1			
Bit operation instructions	MOV	CY,bit	2	10	CY←(bit)		↕	
		bit,CY	2	13	(bit)←CY		↕	
	AND	CY,bit	2	10	CY←CY∧(bit)		↕	
	OR	CY,bit	2	10	CY←CY∨(bit)		↕	
	XOR	CY,bit	2	10	CY←CY∨(bit)		↕	
	SETB	bit	2	13	(bit)←1			
	CLR	bit	2	13	(bit)←0			
	NOT	bit	2	13	(bit)←$\overline{(bit)}$			
	SK	bit	2	10	Skip if (bit)=1	(bit)=1		
	SKN	bit	2	10	Skip if (bit)=0	(bit)=0		
Other operation instructions	DAA		1	4	Decimal Adjust Accumulator		↕	↕
	STC		2	8	CY←1		1	
	CLC		2	8	CY←0		0	
	CMC		2	8	CY←$\overline{CY}$		↕	
	NEGA		2	8	A←$\overline{A}$+1			
Rotation shift instruction	RLD		2	17	Rotate Left Digit			
	RRD		2	17	Rotate Right Digit			
	RLL	r2	2	8	$r2_{n+1}$←$r2_n$, $r2_0$←CY, CY←$r2_7$		↕	
	RLR	r2	2	8	$r2_{n-1}$←$r2_n$, $r2_7$←CY, CY←$r2_0$		↕	
	SLL	r2	2	8	$r2_{n+1}$←$r2_n$, $r2_0$←0, CY←$r2_7$		↕	
	SLR	r2	2	8	$r2_{n-1}$←$r2_n$, $r2_7$←0, CY←$r2_0$		↕	
	SLLC	r2	2	8	$r2_{n+1}$←$r2_n$, $r2_0$←0, CY←$r2_7$	Carry	↕	
	SLRC	r2	2	8	$r2_{n-1}$←$r2_n$, $r2_7$←0, CY←$r2_0$	Carry	↕	
	DRLL	EA	2	8	EA_{n+1}←EA_n, EA_0←CY, CY←EA_8		↕	
	DRLR	EA	2	8	EA_{n-1}←EA_n, EA_8←CY, CY←EA_0		↕	
	DSLL	EA	2	8	EA_{n+1}←EA_n, EA_0←0, CY←EA_8		↕	
	DSLR	EA	2	8	EA_{n-1}←EA_n, EA_8←0, CY←EA_0		↕	
Jump instruction	JMP	word	3	10	PC←word			
	JB		1	4	PC_W←B, PC_L←C			
	JR	word	1	10	PC←PC+1+jdisp1			
	JRE	word	2	10	PC←PC+2+jdisp			
	JEA		2	8	PC←EA			

Instruction group	Mnemonic	Operand	Byte	State	Operation	Skip condition	Flag CY	Flag Z
Call instruction	CALL	word	3	16	$(SP-1)$←$(PC+3)_H$, $(SP-2)$←$(PC+3)_L$, PC←word, (SP)←$(SP-2)$			
	CALB		2	17	$(SP-1)$←$(PC+2)_H$, $(SP-2)$←$(PC+2)_L$, PC_H←0, PC_L←C, (SP)←$(SP-2)$			
	CALF	word	2	13	$(SP-1)$←$(PC+2)_H$, $(SP-2)$←$(PC+2)_L$, PC_{13-11}←000, PC_{10-0}←fa, (SP)←$(SP-2)$			
	CALT	word	1	16	$(SP-1)$←$(PC+1)_H$, $(SP-2)$←$(PC+1)_L$, PC_H←0, PC_L←$(128+2n+1)$, PC_H←$(129+2n)_H$, (SP)←$(SP-2)$			
	SOFTI		1	16	$(SP-1)$←PSW, $(SP-2)$←$(PC+1)_H$, $(SP-3)$←$(PC+1)_L$, PC←0060H, (SP)←$(SP-3)$			
Return instruction	RET		1	10	PC_L←(SP), PC_H←$(SP+1)$, SP←SP+2	Unconditional		
	RETS		1	10	PC←(SP), PC_H←$(SP+1)$, SP←SP+2, PC←PC+a	Unconditional		
	RETI		1	13	PC_L←(SP), PC_H←$(SP+1)$, PSW←$(SP+2)$, SP←SP+3			
Skip instruction	SK	f	2	8	Skip if f=1	f=1		
	SKN	f	2	8	Skip if f=0	f=0		
	SKIT	irf	2	8	Skip if irf=1 then reset irf	irf=1		
	SKNIT	irf	2	8	Skip if irf=0 otherwise reset irf	irf=0		
CPU control instruction	NOP		1	4	No Operation			
	EI		1	4	Enable Interrupt			
	DI		1	4	Disable Interrupt			
	HLT		2	11	Halt			

NOTES: 1. In the expresions for bytes, rpa2, rpa3 to the right of the slash indicate D + byte and H + byte.
2. In the expresions for states, rpa2, rpa3 to the right of the slash indicate D + byte, H + A, H + B H + EA and H + byte.

TRANSFER AND ARITHMETIC LOGICAL OPERATION INSTRUCTION TABLE

8-bit transfer instruction

Addressing	A, rl rl, A	A, srl sr, A	r, word word, r	r, byte	sr2, byte	rpal, byte	wa	wa, byte	rpa2		
									Register, indirect	Autoincrement, autodecrement	Base
Mnemonic	MOV			MVI		MVIX	LDAW	MVIW	LDAX		
							STAW		STAX		

16-bit transfer instruction

Addressing	EA, rp3 rp3, EA	EA, sr4 sr3, EA	rp2, word	word	rpa3		
					Register indirect	Autoincrement, autodecrement	Base
Mnemonic	DMOV		LXI	LBCD LDED LHLD LSPD	LDEAX		
				SBCD SDED SHLD SSPD	STEAX		

Arithmetic logical operation instruction

		1-bit calculation	8-bit operation							16-bit operation		(See NOTE 3.)
Addressing		CY, bit	A, r	r, A	A, byte r, byte sr2, byte	 sr5, byte	(See NOTE 1.) (See NOTE 2.) rpa	(See NOTE 1.) wa	wa, byte	EA, rp3	EA, r2	Skip condition
Arithmetic operation instruction			ADD		ADI		ADDX	ADDW		DADD	EADD	
			ADC		ACI		ADCX	ADCW		DADC		
			ADDNC		ADINC		ADDNCX	ADDNCW		DADDNC		No Carry
			SUB		SUI		SUBX	SUBW		DSUB	ESUB	
			SBB		SBI		SBBX	SBBW		DSBB		
			SUBNB		SUINB		SUBNBX	SUBNBW		DSUBNB		No Borrow
Logical operation instruction	Logic	AND	ANA		ANI		ANAX	ANAW	ANIW	DAN		
		OR	ORA		ORI		ORAX	ORAW	ORIW	DOR		
		XOR	XRA		XRI		XRAX	XRAW		DXR		
	Comparison		GTA			GTI	GTAX	GTAW	GTIW	DGT		1st opd>2nd opd
			LTA			LTI	LTAX	LTAW	LTIW	DLT		1st opd<2nd opd
	Match		NEA			NEI	NEAX	NEAW	NEIW	DNE		1st opd≠2nd opd
			EQA			EQI	EQAX	EQAW	EQIW	DEQ		1st opd=2nd opd
	Test		ONA			ONI	ONAX	ONAW	ONIW	DON		No Zero
			OFFA			OFFI	OFFAX	OFFAW	OFFIW	DOFF		Zero

NOTE: 1. The 1st operand of skip condition is A (accumulator).
2. Register indirect and autoincrement/decrement possible.

Operation							D6	D5	D4	D3	D2	D0
reg, memory			immediate		bit	skip	Z	SK	HC	L1	L0	CY
ADD ADC SUB SBB DADD DADC DSUB DSBB EADD ESUB	ADDW ADCW SUBW SBBW	ADDX ADCX SUBX SBBX	ADI ACI SUI SBI				↕	0	↕	0	0	↕
ANA ORA XRA DAN DOR DXR	ANAW ORAW XRAW	ANAX ORAX XRAX	ANI ORI XRI	ANIW ORIW			↕	0	●	0	0	●
ADDNC SUBNB GTA LTA DDNC SUBNB DGT DLT	ADDNCW SUBNBW GTAW LTAW	ADDNCX SUBNBX GTAX LTAX	ADINC SUINB GTI LTI	 GTIW LTIW			↕	↕	↕	0	0	↕
ONA OFFA DON DOFF	ONAW OFFAW	ONAX OFFAX	ONI OFFI	ONIW OFFIW			↕	↕	●	0	0	●
NEA EQA DNE DEQ	NEAW EQAW	NEAX EQAX	NEI EQI	NEIW EQIW			↕	↕	↕	0	0	↕
INR DCR	INRW DCRW						↕	↕	↕	0	0	●
DAA							↕	0	↕	0	0	↕
RLR RLL SLR SLL DRLR DRLL DSLR DSLL							●	0	●	0	0	↕
SLRC SLLC							●	↕	●	0	0	↕
STC							●	0	●	0	0	1
C							●	0	●	0	0	0
CMC							●	0	●	0	0	↕
			MVI A, byte				●	0	●	1	0	●
			MVI L, byte LXI H, word				●	0	●	0	1	●
					MOV CY, bit AND CY, bit OR CY, bit XOR CY, bit		●	0	●	0	0	↕
						SK SKN SKIT SKNIT	●	↕	●	0	0	●
						RETS	●	1	●	0	0	●
All other instructions							●	0	●	0	0	●

↕ ····· Affected (set or reset)
1 ····· Set
0 ····· Reset
● ····· Not affected

μPD7809/07 MODE REGISTER APPLICATION TABLE

Special register operation instruction table

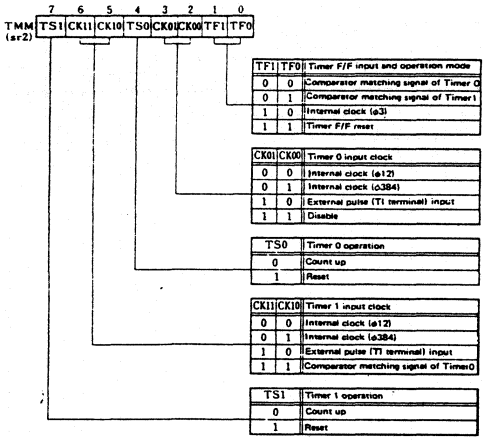
Instruction	Instruction code B1	Instruction code B2	Instruction code B3	Special register PA,PB,PC PD,PF	MKH MKL	SMH	SML	EOM	ETMM	TMM	PT	MM, MCC, MA MB, MC, MF	TXB	RXB	TM0 TM1	WDM	MT
MOV sr, A	0 1 0 0 1 1 0 1	1 1 S_3 S_4 S_3 S_2 S_1 S_0		○	○	○	○	○	○	○		○	○		○	○	○
MOV A, sr1	0 1 0 0 1 1 0 0	1 1 S_3 S_4 S_3 S_2 S_1 S_0		○	○	○		○		○	○			○		○	
MVI sr2, byte	0 1 1 0 0 1 0 0	S_3 0 0 0 0 S_2 S_1 S_0	Data	○	○	○		○		○							
ADI sr2, byte	0 1 1 0 0 1 0 0	S_3 1 0 0 0 S_2 S_1 S_0	Data														
ACI sr2, byte		S_3 1 0 1 0 S_2 S_1 S_0															
ADINC sr2, byte		S_3 0 1 0 0 S_2 S_1 S_0															
SUI sr2, byte		S_3 1 1 0 0 S_2 S_1 S_0															
SBI sr2, byte		S_3 1 1 1 0 S_2 S_1 S_0		○	○	○		○		○							
SUINB sr2, byte		S_3 0 1 1 0 S_2 S_1 S_0															
ANI sr2, byte		S_3 0 0 0 1 S_2 S_1 S_0															
ORI sr2, byte		S_3 0 0 1 1 S_2 S_1 S_0															
XRI sr2, byte		S_3 0 0 1 0 S_2 S_1 S_0															
GTI sr5, byte	0 1 1 0 0 1 0 0	S_3 0 1 0 1 S_2 S_1 S_0	Data														
LTI sr5, byte		S_3 0 1 1 1 S_2 S_1 S_0															
NEI sr5, byte		S_3 1 1 0 1 S_2 S_1 S_0		○	○	○		○		○	○						
EQI sr5, byte		S_3 1 1 1 1 S_2 S_1 S_0															
ONI sr5, byte		S_3 1 0 0 1 S_2 S_1 S_0															
OFFI sr5, byte		S_3 1 0 1 1 S_2 S_1 S_0															

Special register table

Special register		Code S_5 S_4 S_3 S_2 S_1 S_0	sr	sr1	sr2	sr5
PORT A	PA	0 0 0 0 0 0	○	○	○	○
PORT B	PB	0 0 0 0 0 1	○	○	○	○
PORT C	PC	0 0 0 0 1 0	○	○	○	○
PORT D	PD	0 0 0 0 1 1	○	○	○	○
PORT F	PF	0 0 0 1 0 1	○	○	○	○
MASK High	MKH	0 0 0 1 1 0	○	○	○	○
MASK Low	MKL	0 0 0 1 1 1	○	○	○	○
SERIAL MODE High	SMH	0 0 1 0 0 1	○	○	○	○
SERIAL MODE Low	SML	0 0 1 0 1 0	○			
TIMER/EVENT COUNTER OUTPUT MODE	EOM	0 0 1 0 1 1	○	○	○	○
TIMER/EVENT COUNTER MODE	ETMM	0 0 1 1 0 0	○			
TIMER MODE	TMM	0 0 1 1 0 1	○	○	○	○
PORT T	PT	0 0 1 1 1 0		○		○
MEMORY MAPPING	MM	0 1 0 0 0 0	○			
MODE CONTROL C	MCC	0 1 0 0 0 1	○			
MODE A	MA	0 1 0 0 1 0	○			
MODE B	MB	0 1 0 0 1 1	○			
MODE C	MC	0 1 0 1 0 0	○			
MODE F	MF	0 1 0 1 1 1	○			
Tx BUFFER	TXB	0 1 1 0 0 0	○			
Rx BUFFER	RXB	0 1 1 0 0 1		○		
TIMER REG 0	TM0	0 1 1 0 1 0	○			
TIMER REG 1	TM1	0 1 1 0 1 1	○			
WATCHDOG TIMER MODE	WDM	1 0 0 1 0 0	○	○		
MODE T	MT	1 0 0 1 0 1	○			

1. Timer

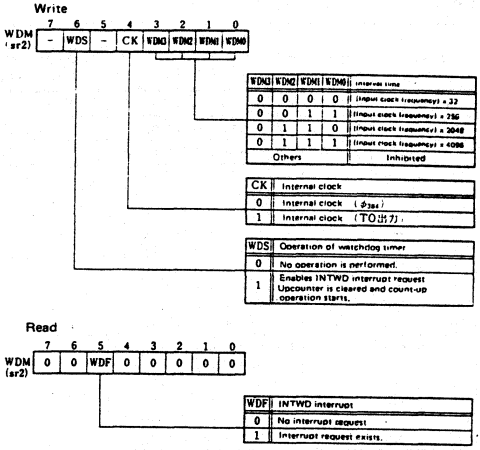
Timer mode register



2. Watchdog timer

Watchdog timer mode register

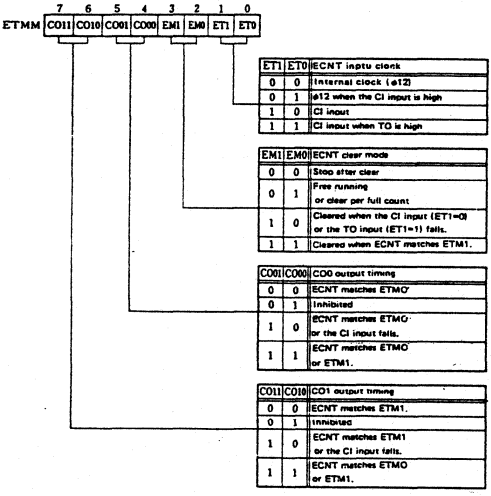
Write



Read

3. Timer/event counter

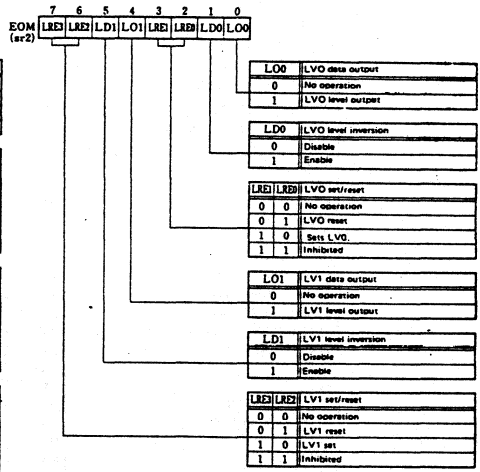
Timer/event counter mode register



Note: ECPT latches the ECNT contents:

- o When the ET1 bit is 0 and the CI input falls.
- o When the ET1 bit is 1 and TO falls.

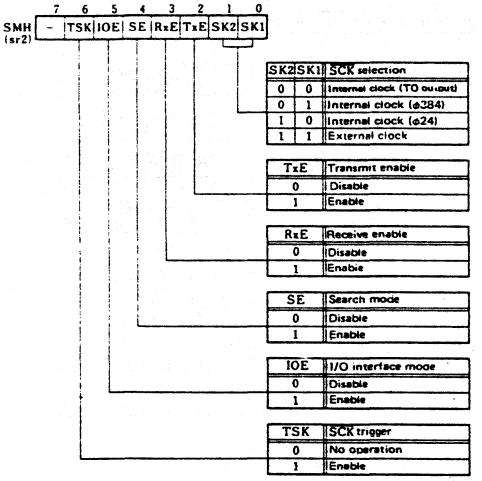
Timer/event counter output mode register



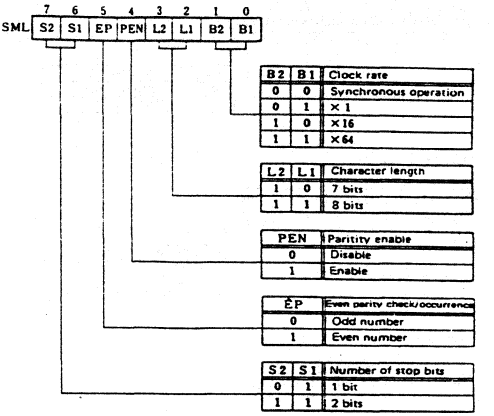
Note: To reset the output latch, reset the level output of level F/F (LV0, LV1) and level F/F (LV0, LV1).

4. Serial interface

Serial mode high register



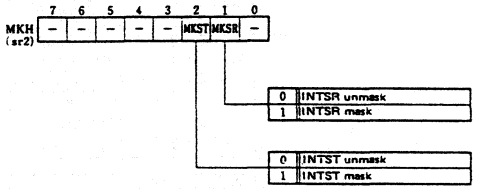
Serial mode low register



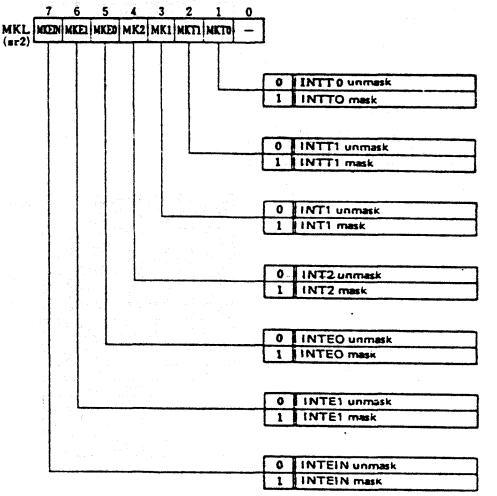
NOTE: Bit TSK is used to receive serial data in I/O interface mode.

5. Interrupt control

Interrupt mask high register



Interrupt mask low register



6. Port

Mode A register

7	6	5	4	3	2	1	0
MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0

MAn	PAn input/output
0	PAn : Output port
1	PAn : Input port

(n = 0 ~ 7)

Mode B register

7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

MBn	PBn input/output
0	PBn : Output port
1	PBn : Input port

(n = 0 ~ 7)

Mode C register

7	6	5	4	3	2	1	0
MC7	MC6	MC5	MC4	MC3	MC2	MC1	MC0

MCn	PCn input/output
0	PCn : Output port
1	PCn : Input port

(n = 0 ~ 7)

Mode F register

7	6	5	4	3	2	1	0
MF7	MF6	MF5	MF4	MF3	MF2	MF1	MF0

MFn	PFn input/output
0	PFn : Output port
1	PFn : Input port

(n = 0 ~ 7)

MODE T register

7	6	5	4	3	2	1	0
–	–	–	–	MT3	MT2	MT1	MT0

MT3	MT2	MT1	MT0	Treshold voltage
0	0	0	0	$V_{TH} \times 9/9$
0	0	0	1	$V_{TH} \times 1/9$
0	0	1	0	$V_{TH} \times 2/9$
0	0	1	1	$V_{TH} \times 3/9$
0	1	0	0	$V_{TH} \times 4/9$
0	1	0	1	$V_{TH} \times 5/9$
0	1	1	0	$V_{TH} \times 6/9$
0	1	1	1	$V_{TH} \times 7/9$
1	0	0	0	$V_{TH} \times 8/9$
1	0	0	1	$V_{TH} \times 9/9$
1	0	1	0	$V_{TH} \times 3/9$
1	0	1	1	$V_{TH} \times 4/9$
1	1	0	0	$V_{TH} \times 5/9$
1	1	0	1	$V_{TH} \times 6/9$
1	1	1	0	$V_{TH} \times 7/9$
1	1	1	1	$V_{TH} \times 8/9$

Mode control C register

7	6	5	4	3	2	1	0
MCC7	MCC6	MCC5	MCC4	MCC3	MCC2	MCC1	MCC0

0	PC0 : Port mode
1	PC0 : TxD output
0	PC1 : Port mode
1	PC1 : RxD input
0	PC2 : Port mode
1	PC2 : SCK input/output
0	PC3 : Port mode
1	PC3 : INT2/TI input
0	PC4 : Port mode
1	PC4 : TO output
0	PC5 : Port mode
1	PC5 : CI input
0	PC6 : Port mode
1	PC6 : CO0 output
0	PC7 : Port mode
1	PC7 : CO1 output

Memory mapping register

7	6	5	4	3	2	1	0
–	–	–	WTE	RAE	MM2	MM1	MM0

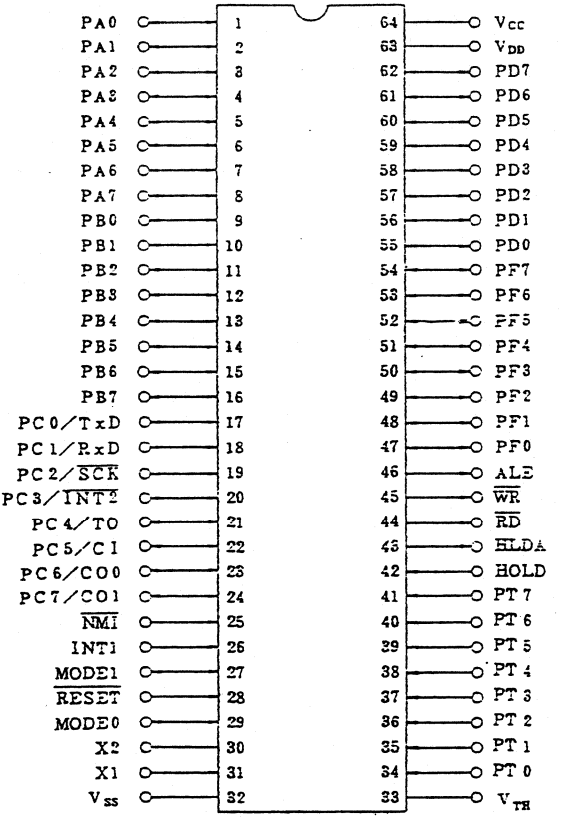
MM2	MM1	MM0	PD, PF operation	
0	0	0	Single-chip	PD7-0 : Input port PF7-0 : Port mode
0	0	1		PD7-0 : Output port PF7-0 : Port mode
0	1	0	256 bytes	PD7-0 : Extension mode PF7-0 : Port mode
1	0	0	4K bytes	PD7-0 / PF3-0 : Extension mode PF7-4 : Port mode
1	1	0	16K bytes	PD7-0 / PF5-0 : Extension mode PF7-6 : Port mode
1	1	1	56K bytes	PD7-0 / PF7-0 : Extension mode

RAE	Built-in RAM access
0	Disable
1	Enable

WTE	Inserts 1 wait when accessing external memory
0	Disable
1	Enable

NOTE: In the μPD7807, each bit of the memory mapping register MM, i.e., bits MM2, MM1 and MM0, is set to 0.

CHAPTER 11 μCOM7809 PIN CONFIGURATION



PA7-0 : PortA
PB7-0 : PortB
PC7-0 : PortC
PD7-0 : PortD
PF7-0 : PortF
NMI : Non Maskable Interrupt
INT1 : Interrupt Request
MODE0,1 : Mode0,1
X1,X2 : Crystal

PT7-0 : PortT
RD : Read Strobe
WR : Write Strobe
ALE : Address Latch Enable
RESET : Reset
V_{TH} : Threshold Voltage
HOLD : Hold Request
HLDA : Hold Acknowledge

APPENDIX C

μPD7809 Electrical Specification

ELECTRICAL SPECIFICATIONS

AND PACKAGE OUTLINES FOR

μPD7809

ABSOLUTE MAXIMUM RATINGS

(T_a = 25° C)

PARAMETER	SYMBOL	TEST CONDITIONS	RATINGS	UNITS
Power Supply Voltage	V_{CC}		−0.5 to +7.0	V
	V_{DD}		−0.5 to +7.0	V
Input Voltage	V_I		−0.5 to +7.0	V
Output Voltage	V_O		−0.5 to + 7.0	V
Output Current Low	I_{OL}	All Output Pin	4.0	mA
		All Output Pin Total	100	mA
Output Current High	I_{OH}	All Output Pin	−0.5	mA
		All Output Pin Total	−20	mA
Threshold Voltage	V_{TH}		−0.5 to V_{CC} + 0.1	V
Operating Temperature	T_{opt}	10 MHz < f_{XTAL} ≤ 12 MHz	−10 to +70	°C
		f_{XTAL} ≤ 10 MHz	−10 to +70	°C
Storage Temperature	T_{stg}		−40 to + 125	°C

OPERATING CONDITION

PARAMETER / OSC. FREQ.	T_a	V_{CC}, AV_{CC}
10 MHz < f_{XTAL} ≤ 12 MHz *1	−10°C to +70°C	+5.0V ± 5 %
f_{XTAL} ≤ 10 MHz *1	−10°C to +70°C	+5.0V ± 10 %

CAPACITANCE

(T_a = 25°C, V_{CC} = V_{DD} = V_{SS} = 0V)

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Input Capacitance	C_I	f_C = 1 MHz Unmeasured Pins Returned to 0V			10	pF
Output Capacitance	C_O				20	pF
I/O Capacitance	$C_{I/O}$				20	pF

DC CHARACTERISTICS

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 5 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC}

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Input Low Voltage	V_{IL}		0		0.8	V
Input High Voltage	V_{IH1}	All except $\overline{SCK}$, $\overline{RESET}$, X1	2.0		V_{CC}	V
	V_{IH2}	$\overline{SCK}$, X1 *8	0.8 V_{CC}		V_{CC}	V
	V_{IH3}	$\overline{RESET}$	0.8V_{DD}		V_{CC}	V
Output Low Voltage	V_{OL}	I_{OL} = 2.0mA			0.45	V
Output High Voltage	V_{OH}	I_{OH} = −200μA	2.4			V
Input Current	I_I	INT1, TI (PC3); +0.45V ≤ V_I ≤ V_{CC}			± 200	μA
Input Leakage Current	I_{LI}	All except INT1, TI (PC3) 0V ≤ V_I ≤ V_{CC}			± 10	μA
Output Leakage Current	I_{LO}	+0.45V ≤ V_O ≤ V_{CC}			± 10	μA
V_{TH} Input Current	I_{TH}	V_{TH} = V_{CC}		0.2*2	0.5	mA
V_{DD} Supply Current	I_{DD}			1.5 *2	3.2	mA
V_{CC} Supply Current	I_{CC}			150*2	200	mA

AC CHARACTERISTICS READ/WRITE OPERATION

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 5 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC})

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	MAX	UNITS
X1 Input Cycle Time	t_{CYC}		83	250	ns
Address Setup to ALE ↓	t_{AL}	*3, *5	65		ns
Address Hold from ALE ↓	t_{LA}	*3, *5	50		ns
Address to $\overline{RD}$ ↓ Delay Time	t_{AR}	*3, *5	150		ns
$\overline{RD}$ ↓ to Address Floating	t_{AFR}	*5		20	ns
Address to Data Input	t_{AD}	*3, *5		360	ns
ALE ↓ to Data Input	t_{LDR}	*3, *5		215	ns
$\overline{RD}$ ↓ to Data Input	t_{RD}	*3, *5		180	ns
ALE ↓ to $\overline{RD}$ ↓ Delay Time	t_{LR}	*3, *5	35		ns
Data Hold Time from $\overline{RD}$ ↑	t_{RDH}	*5	0		ns
$\overline{RD}$ ↑ to ALE ↑ Delay Time	t_{RL}	*3, *5	115		ns
$\overline{RD}$ Width Low	t_{RR}	Data Read *3, *5	280		ns
		OP Code Fetch *3, *5	530		ns
ALE Width High	t_{LL}	*3, *5	125		ns
$\overline{M1}$ Setup Time to $\overline{ALE}$ ↓	t_{ML}	*3	65		ns
$\overline{M1}$ Hold Time from ALE ↓	t_{LM}	*3	50		ns
$\overline{IO}$/M Setup Time to ALE ↓	t_{IL}	*3	65		ns
$\overline{IO}$/M Hold Time from ALE ↓	t_{LI}	*3	50		ns
Address to $\overline{WR}$ ↓ Delay	t_{AW}	*3, *5	150		ns
ALE ↓ to Data Output	t_{LDW}	*3, *5		195	ns
$\overline{WR}$ ↓ to Data Output	t_{WD}	*5		100	ns
ALE ↓ to $\overline{WR}$ ↓ Delay Time	t_{LW}	*3, *5	35		ns
Data Setup Time to $\overline{WR}$ ↑	t_{DW}	*3, *5	230		ns
Data Hold Time from $\overline{WR}$ ↑	t_{WDH}	*3, *5	95		ns
$\overline{WR}$ ↑ to ALE ↑ Delay Time	t_{WL}	*3, *5	115		ns
$\overline{WR}$ Width Low	t_{WW}	*3, *5	280		ns

SERIAL OPERATION

PARAMETER	SYMBOL	TEST CONDITIONS		MIN	MAX	UNITS
$\overline{SCK}$ Cycle Time	t_{CYK}	$\overline{SCK}$ Input	*6	1		μs
			*7	500		ns
		$\overline{SCK}$ Output		2		μs
$\overline{SCK}$ Width Low	t_{KKL}	$\overline{SCK}$ Input	*6	400		ns
			*7	200		ns
		$\overline{SCK}$ Output		900		ns
$\overline{SCK}$ Width High	t_{KKH}	$\overline{SCK}$ Input	*6	400		ns
			*7	200		ns
		$\overline{SCK}$ Output		900		ns
RxD Setup Time to $\overline{SCK}$ ↑	t_{RXK}	*6		80		ns
RxD Hold Time from $\overline{SCK}$ ↑	t_{KRX}	*6		80		ns
$\overline{SCK}$ ↓ to TxD Delay Time	t_{KTX}	*6			210	ns

DC CHARACTERISTICS

(T_a = −40°C to +70°C, V_{CC} = +5.0V ± 10 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC})

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Input Low Voltage	V_{IL}		0		0.8	V
Input High Voltage	V_{IH1}	All except $\overline{SCK}$, $\overline{RESET}$, X1	2.0		V_{CC}	V
	V_{IH2}	$\overline{SCK}$, X1 *8	0.8 V_{CC}		V_{CC}	V
	V_{IH3}	$\overline{RESET}$	0.8V_{DD}		V_{CC}	V
Output Low Voltage	V_{OL}	I_{OL} = 2.0mA			0.45	V
Output High Voltage	V_{OH}	I_{OH} = −200μA	2.4			V
Input Current	I_I	INT1, TI (PC3); +0.45V ≤ V_I ≤ V_{CC}			± 200	μA
Input Leakage Current	I_{LI}	All except INT1, TI (PC3); 0V ≤ V_I ≤ V_{CC}			± 10	μA
Output Leakage Current	I_{LO}	+0.45V ≤ V_O ≤ V_{CC}			± 10	μA
V_{TH} Input Current	I_{TH}	V_{TH} = V_{CC}		0.2	0.6	mA
V_{DD} Supply Current	I_{DD}			1.5 *2	3.5	mA
V_{CC} Supply Current	I_{CC}			150*2	220	mA

AC CHARACTERISTICS READ/WRITE OPERATION

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 10 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC})

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	MAX	UNITS
X1 Input Cycle Time	t_{CYC}		100	250	ns
Address Setup to ALE ↓	t_{AL}	*4, *5	100		ns
Address Hold from ALE ↓	t_{LA}	*4, *5	70		ns
Address to $\overline{RD}$ ↓ Delay Time	t_{AR}	*4, *5	200		ns
$\overline{RD}$ ↓ to Address Floating	t_{AFR}	*5		20	ns
Address to Data Input	t_{AD}	*4, *5		480	ns
ALE ↓ to Data Input	t_{LDR}	*4, *5		300	ns
$\overline{RD}$ ↓ to Data Input	t_{RD}	*4, *5		250	ns
ALE ↓ to $\overline{RD}$ ↓ Delay Time	t_{LR}	*4, *5	50		ns
Data Hold Time from $\overline{RD}$ ↑	t_{RDH}	*5	0		ns
$\overline{RD}$ ↑ to ALE ↑ Delay Time	t_{RL}	*4, *5	150		ns
$\overline{RD}$ Width Low	t_{RR}	Data Read *4, *5	350		ns
		OP Code Fetch *4, *5	650		ns
ALE Width High	t_{LL}	*4, *5	160		ns
$\overline{M1}$ Setup Time to ALE ↓	t_{ML}	*4	100		ns
$\overline{M1}$ Hold Time from ALE ↓	t_{LM}	*4	70		ns
$\overline{IO}$/M Setup Time to ALE ↓	t_{IL}	*4	100		ns
$\overline{IO}$/M Hold Time from ALE ↓	t_{LI}	*4	70		ns
Address to $\overline{WR}$ ↓ Delay	t_{AW}	*4, *5	200		ns
ALE ↓ to Data Output	t_{LDW}	*4, *5		210	ns
$\overline{WR}$ ↓ to Data Output	t_{WD}	*5		100	ns
ALE ↓ to $\overline{WR}$ ↓ Delay Time	t_{LW}	*4, *5	50		ns
Data Setup Time to $\overline{WR}$ ↑	t_{DW}	*4, *5	300		ns
Data Hold Time from $\overline{WR}$ ↑	t_{WDH}	*4, *5	130		ns
$\overline{WR}$ ↑ to ALE ↑ Delay Time	t_{WL}	*4, *5	150		ns
$\overline{WR}$ Width Low	t_{WW}	*4, *5	350		ns

SERIAL OPERATION

PARAMETER	SYMBOL	TEST CONDITIONS		MIN	MAX	UNITS
$\overline{SCK}$ Cycle Time	t_{CYK}	$\overline{SCK}$ Input	*6	1.2		μs
			*7	500		ns
		$\overline{SCK}$ Output		2.4		μs
$\overline{SCK}$ Width Low	t_{KKL}	$\overline{SCK}$ Input	*6	500		ns
			*7	200		ns
		$\overline{SCK}$ Output		1.1		μs
$\overline{SCK}$ Width High	t_{KKH}	$\overline{SCK}$ Input	*6	500		ns
			*7	200		ns
		$\overline{SCK}$ Output		1.1		ns
RxD Setup Time to $\overline{SCK}$ ↑	t_{RXK}	*6		80		ns
RxD Hold Time from $\overline{SCK}$ ↑	t_{KRX}	*6		80		ns
$\overline{SCK}$ ↓ to TxD Delay Time	t_{KTX}	*6			210	ns

HOLD OPERATION

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 5 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC}

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 10 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC}

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNITS
HOLD ↑ Setup Time to ALE ↑	t_{SHDL}		2T + 150			ns
ALE ↑ to HLDA ↑ Delay	t_{DLHA}				T + 150	ns
HLDA ↑ to BUS Floating	t_{FBHA}		0			ns
HOLD ↓ to HLDA ↓ Delay	t_{HDDA}		T −50		4T + 150	ns
HLDA ↓ to Bus Enable Time	t_{EHAB}		0			ns
Bus Setup Time to ALE	t_{BL}		2T −100			ns

COMPARATOR CHARACTERISTICS

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 5 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC}

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 10 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC}

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Comparison Accuracy	V_{ACOMP}				± 100	mV
Threshold Voltage	V_{TH}		0		V_{CC}+0.1	V
Comparison Time	t_{COMP}		144		145	T_{CYC}
PT Input Voltage	V_{IPT}		0		V_{CC}	V

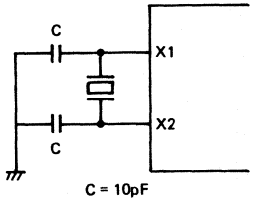
ZERO-CROSS CHARACTERISTICS

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 5 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC})

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 10 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC})

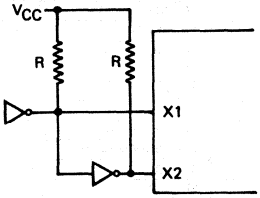
PARAMETER	SYMBOL	TEST CONDITIONS	MIN	MAX	UNITS
Zero-Cross Detection Input	V_{ZX}	AC Coupled	1	1.8	VAC_{p-p}
Zero-Cross Accuracy	A_{ZX}	60 Hz Sine Wave		± 135	mV
Zero-Cross Detextion Input Frequency	f_{ZX}		0.05	1	kHz

*1: XTAL oscillation circuit



C = 10pF

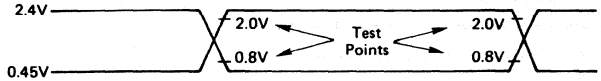
*8: External clock drive circuit



R = 5KΩ

*2: T_a = +25°C, V_{CC} = V_{DD} = 5V
*3: f_{XTAL} = 12 MHz
*4: f_{XTAL} = 10 MHz
*5: Load Capacitance: C_L = 150 pF
*6: Asynchronous mode with 1x baud rate, synchronous, I/O interface mode
*7: Asynchronous mode with 16x or 64x baud rate

AC TIMING TEST POINTS



EXTERNAL CLOCK

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 5 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC})

(T_a = −10°C to +70°C, V_{CC} = +5.0V ± 10 %, V_{SS} = 0V, V_{CC} −0.8V ≤ V_{DD} ≤ V_{CC})

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	MAX	UNITS
High Level Width	$t_{\phi H}$		30	250	ns
Low Level Width	$t_{\phi L}$		30	250	ns
Rising Time	t_r		0	30	ns
Falling Time	t_f		0	30	ns

DATA RETENTION CHARACTERISTICS

(T_a = −10 to +70°C, V_{CC} = 0V, V_{DD} = V_{DDDR})

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Data Retention Voltage	V_{DDDR}	RESET = V_{IL}	3.2		5.5	V
Data Retention Supply Current	I_{DDDR}	RESET = V_{IL}, V_{DDDR} = 3.2 V		1.3	3.0	mA

BUS TIMING DEPENDING ON tCYC

SYMBOL	CALCULATING EXPRESSION	MIN./MAX.	UNITS
t_{AL}	2T − 100	MIN	ns
t_{LA}	T − 30	MIN	ns
t_{AR}	3T − 100	MIN	ns
t_{AD}	7T − 220 *1	MAX	ns
t_{LDR}	5T − 200 *1	MAX	ns
t_{RD}	4T − 150 *1	MAX	ns
t_{LR}	T − 50	MIN	ns
t_{RL}	2T − 50	MIN	ns
t_{RR}	4T − 50 (Data Read) *1	MIN	ns
	7T − 50 (OP Code Fetch) *1		
t_{LL}	2T − 40	MIN	ns
t_{ML}	2T − 100	MIN	ns
t_{LM}	T − 30	MIN	ns
t_{IL}	2T − 100	MIN	ns
t_{LI}	T − 30	MIN	ns
t_{AW}	3T − 100	MIN	ns
t_{LDW}	T + 110	MAX	ns
t_{LW}	T − 50	MIN	ns
t_{DW}	4T − 100 *1	MIN	ns
t_{WDH}	2T − 70	MIN	ns
t_{WL}	2T − 50	MIN	ns
t_{WW}	4T − 50 *1	MIN	ns
t_{CYK}	12T ($\overline{SCK}$ Input) *2	MIN	ns
	24T ($\overline{SCK}$ Output)		
t_{KKL}	5T + 5 (SCK Input) *2	MIN	ns
	12T − 100 ($\overline{SCK}$ Output)		
t_{KKH}	5T + 5 ($\overline{SCK}$ Input) *2	MIN	ns
	12T − 100 ($\overline{SCK}$ Output)		

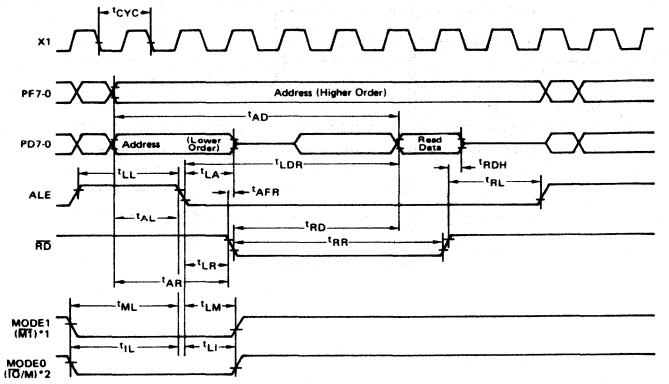
*1: Add 3T to each parameter in the case of external memory access using program WAIT function.

*2: Asynchronous mode with 1x baud rate, Synchronous, I/O Interface Mode

*3: T = t_{CYC} = 1/f_{XTAL}

*4: The items out of this table are not dependent on f_{XTAL}.

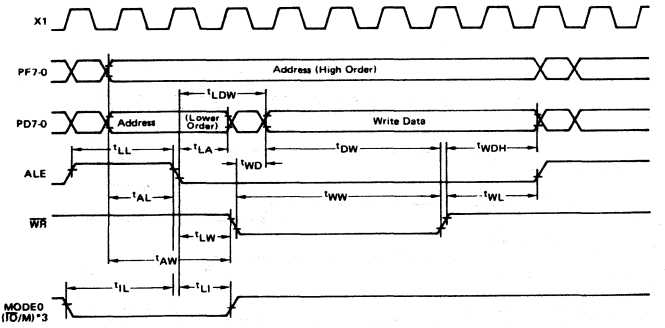
READ OPERATION



*1: $\overline{M1}$ signal is output to the MODE1 pin at 1st OP code fetch cycle when MODE 1 pin is pulled-up to V_{CC}.

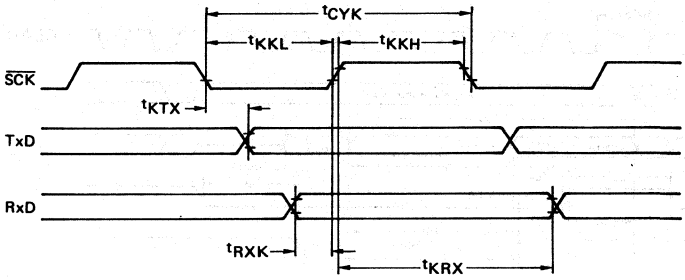
*2: $\overline{IO}$/M signal is output to the MODE0 pin at sr to sr2 register read timing when MODE0 pin is pulled-up to V_{CC}.

WRITE OPERATION

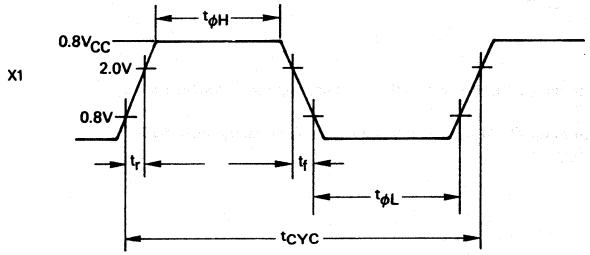


*3: $\overline{IO}$/M signal is output to the MODE0 pin at sr to sr2 register write timing when MODE0 pin is pulled-up to V_{CC}.

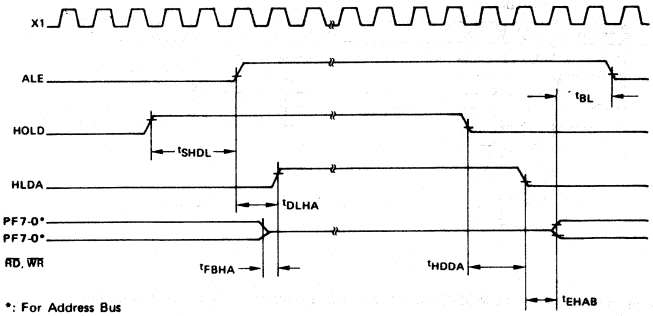
SERIAL OPERATION



X1 INPUT WAVEFORM

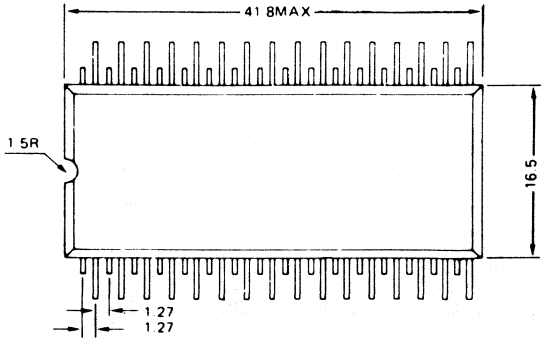


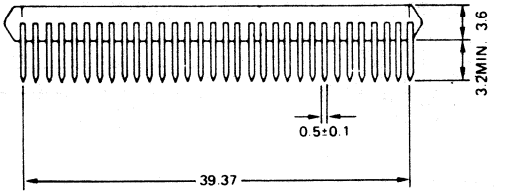
HOLD OPERATION



*: For Address Bus

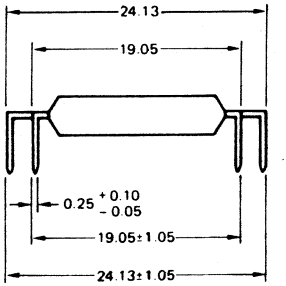
64 PIN PLASTIC
QUIP OUTLINE BENT LEADS
(Unit : mm)
μPD7809G



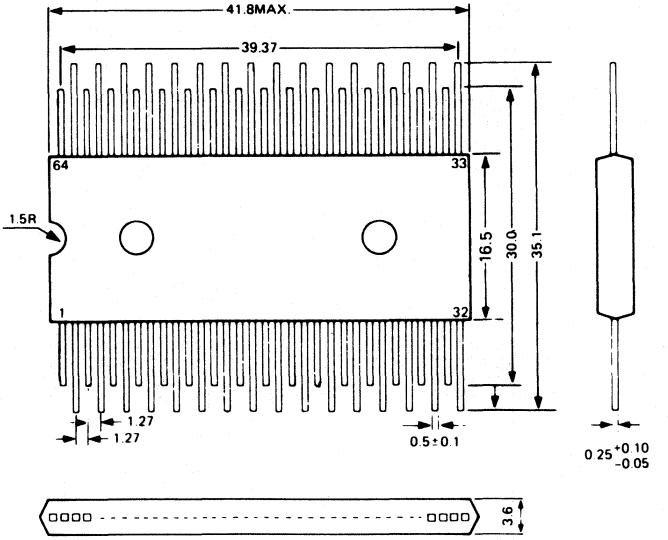


When ordering this package, specify as follows:

μPD7809G-xxx-36



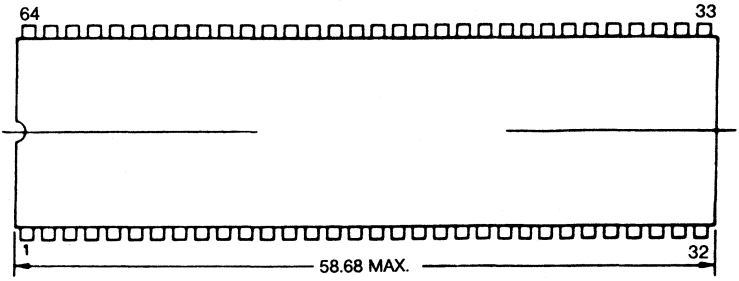
64 PIN PLASTIC
PACKAGE OUTLINE
STRAIGHT LEADS
(Unit : mm)
μPD7809



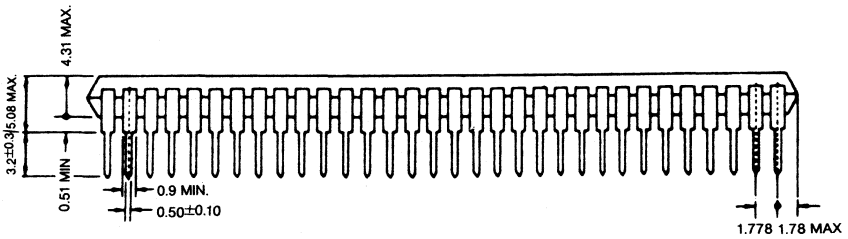
When ordering this package, specify as follows: μPD7809G-xxx-37

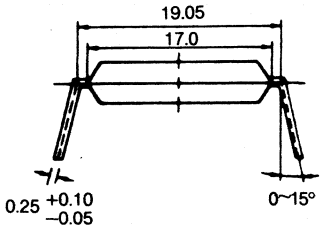
PACKAGE OUTLINE
μPD7809CW

64-PIN SHRINK DIP



When ordering this package, specify as follows:
μPD7807CW
μPD7808CW-xxx





Book 3

PG-1000

USER'S MANUAL

PREFACE

This manual is in two parts.

Part 1 deals with the method of installing the PG-1000. Please read this part carefully before installing.

Part 2 explains how to operate the PG-1000. As will be explained later, there are three operation modes for the PG-1000, namely, the host mode, console mode and stand-alone mode, which are discussed in detail in Chapters 2, 3 and 4, respectively.

The appendixes give a list of the commands for the host and console modes and an error code list.

The following are some of the terms used in this manual, especially for the host and console modes.

PGC:	PG-1000 control program for MD-080/086
MD-080/086:	Generic term for NEC's MD Series and MD Series Model-10 host machines
MD buffer:	Data area in PGC
PG buffer:	Data area in the PG-1000 (The PG-1000 has a 16K-byte buffer memory.)
_____:	A key entry from the host machine or the console.
↵ :	Pressing of the RETURN key.

PROPER USE OF THE PG-1000

Before using it, please read the following instructions carefully. The PG-1000 cannot be guaranteed for any problems resulting from these instructions being ignored.

1. Always use the regular power source.
2. When setting a personal module, pull out the power cable. If a personal module is set with the power cable plugged in, the module and the PG-1000 may be damaged and there is also danger of electric shock.
3. Don't turn the power switch on/off with a PROM inserted in the PROM programmer. This may cause the PROM and PROM programmer to be damaged.
4. Don't use the PG-1000 in high-temperature, high-humidity, or dusty environments. This may cause the PG-1000 to develop problems.
5. Don't use the PG-1000 in noisy environments. This may cause it to malfunction.

PART I SYSTEM INSTALLATION

SECTION 1 SYSTEM OVERVIEW

The PG-1000 is a PROM programmer designed for the MD-080 and MD-086 Series host machines.

By allowing its optional personal modules to be changed, the PG-1000 can serve as a single-chip microcomputer with a built-in PROM or a bipolar PROM programmer.

Also, the provision of key switches on the front and rear panels and a serial interface permits the PG-1000 to be used as a stand-alone PROM programmer or to be activated through a console connected to the serial interface.

1.1 PG-1000 Hardware Specifications

Table 1.1 lists the hardware specifications of the PG-1000.

Table 1.1 Hardware specifications

CPU	μPD780
Data RAM	16K bytes
Control ROM	4K bytes
Serial interface	RS-232C, TTL, current loop
Parallel interface	TTL (two-wire handshake)

1.2 Operating Environment

Table 1.2 lists the conditions for the operating environment.

Table 1.2 Operating environment conditions

Temperature	10 to 35°C
Humidity	20 to 80% RH

1.3 Operating Modes

The PG-1000 can be used in the host mode, console mode or stand-alone mode. The equipment configuration for each of these modes is shown in Figs. 1.1 to 1.3.

① Host mode

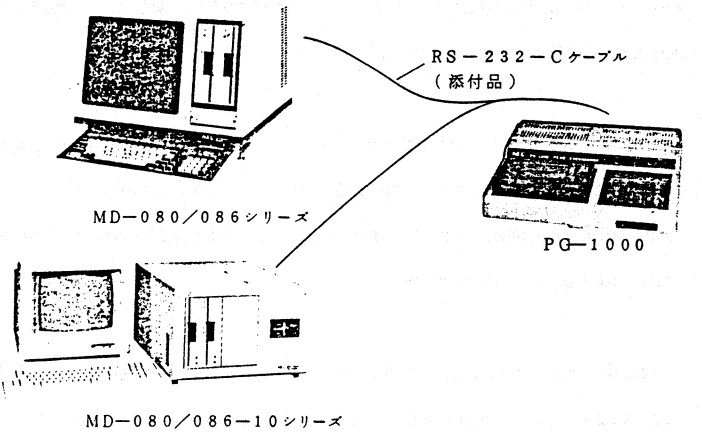


Fig. 1.1 Mode I (Host connection mode)

② Console mode

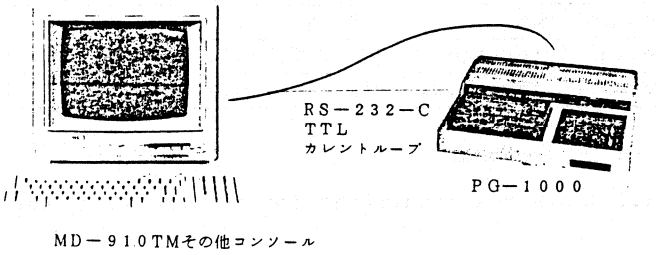
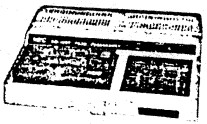


Fig. 1.2 Mode II (Console operation mode)

③ Stand-alone mode



PG—1000

Fig. 1.3 Mode III (Stand-alone mode)

1.4 Appearance

The appearance of the PG-1000 is shown in Fig. 1.4.

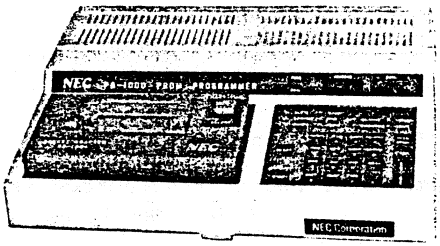


Fig. 1.4 Appearance of PG-1000

SECTION 2 PACKING

This chapter explains the items contained in the PG-1000 packing.

2.1 Unpacking

The packing contains the following items:

- o PG-1000
- o PG-1000 manual
- o Power cable
- o MD connecting cable
- o Floppy disk

If any defects are found, please report them to your dealer.

2.2 Description of Items Contained in the Package

Each of the items contained in the package is explained below.

① Program product licence

This is your contract for using the programs stored in the floppy disk. Please fill in the form and return it to us.

② Guarantee certificate

This is a certificate of guarantee for the product. Please fill in the form and return it to us.

③ PG-1000 manual

This describes how to handle and operate the product. Please read this prior to using the product to avoid misoperations that may damage the product.

④ Power cable

This cable is used to connect the PG-1000 to the power source. The voltage specifications on the rear panel must be strictly observed.

⑤ MD connection cable

This cable used when the product is used in the host mode. Please note that this cable cannot be used to connect to a console.

⑥ Floppy disk

This stores the following PG-1000 control programs that operate on the MD-080/086:

(i) PGC.CMD (MP/M-86*base)

(ii) PGC.COM (CP/M**base)

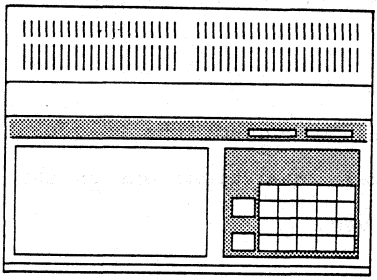
Please copy the programs from the attached floppy disk to your own floppy disk and keep the former as the master disk so that a backup is available if the program is accidentally destroyed.

* MP/M-86 is a registered trademark of Digital Research, U.S.A.

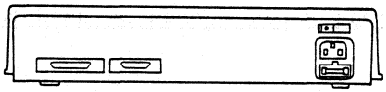
** CP/M is a registered trademark of Digital Research U.S.A.

SECTION 3 PANEL FUNCTIONS

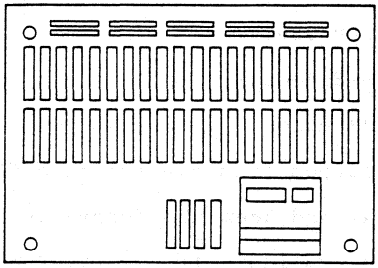
This chapter outlines the panel functions of the PG-1000.



Front panel



Rear panel



Bottom panel

Fig. 3.1 External views of the PG-1000

3.1 Front Panel

3.1.1 Display and mode specification LED

Figure 3.2 shows the display and mode specification LED.

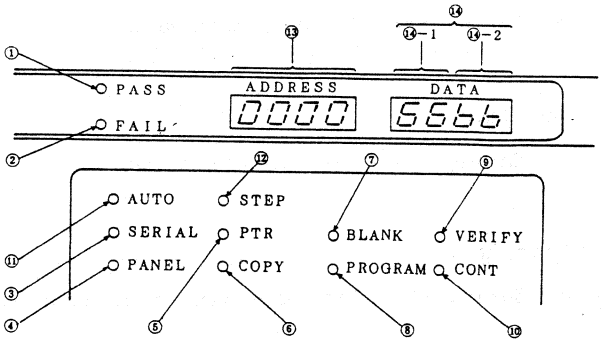


Fig. 3.2 Front panel LED

① PASS lamp

This shows whether operations are normal. It turns on when copying, blank checking, program execution, verification and continuance operations each terminate normally.

② FAIL lamp

This indicates errors in operation. It turns on if an error occurs in copying, blank checking, program execution, verification or continuance operations.

③ SERIAL operation lamp

This turns on during the SERIAL operation mode.

④ PANEL operation lamp

This turns on during the PANEL operation mode.

⑤ PTR operation lamp

This turns on during the PRT operation mode.

⑥ COPY operation lamp

This turns on during the COPY operation mode.

⑦ BLANK operation lamp

This turns on during the blank checking operation mode.

⑧ PROGRAM operation lamp

This turns on during the PROGRAM operation mode.

⑨ VERIFY operation lamp

This turns on during the VERIFY operation mode.

⑩ CONT operation lamp

This turns on during the continuance operation mode.

⑪ AUTO mode lamp

This turns on during the AUTO mode. When this LED is lit, all addresses are operated on continuously.

⑫ STEP mode lamp

This turns on during the STEP mode. When this LED is lit, the operation stops after execution at each address.

⑬ ADDRESS display

This displays the address to be given to the PROM. In the PANEL mode, it also displays the operation address. It is also indicates various messages.

⑭ DATA display

⑭-1 Indicates the contents of the buffer memory for data to be written.

⑭-2 Indicates the data read out from the PROM.

3.1.2 Key switches

Figure 3.3 shows the key switch panel.

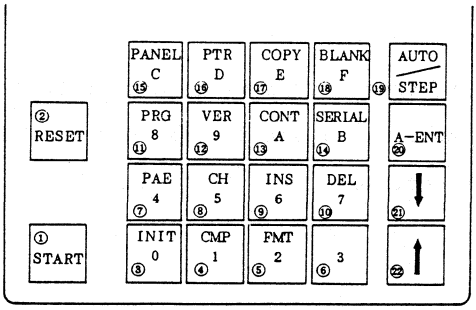


Fig. 3.3 Front panel key switches

① START key

In the COPY, BLANK CHECK, PROGRAM, VERIFY and CONTINUANCE modes, pressing this key starts the corresponding operation.

② RESET key

In the PANEL, COPY, BLANK CHECK, PROGRAM, VERIFY, and CONTINUANCE modes, pressing this key stops and resets the current operation.

(Note)

Key switches ③ to ⑩, and ⑳ to ㉒ below are effective only in the PANEL mode.

③ INIT 0 key

This is used to initialize the memory. It also is used to enter a hexadecimal zero (0) after a command has been entered.

④ CMP 1 key

This is used to reverse the contents of the buffer memory. It also is used to enter a hexadecimal one (1) after a command has been entered.

⑤ FMT 2 key

This is used to specify a tape format when reading data from the PTR. It also is used to enter a hexadecimal two (2) after a command has been entered.

⑥ 3 key

This is used to enter a hexadecimal three (3) after a command has been entered.

⑦ PAE 4 key

This is used to specify the address range in each operation mode. It also is used to enter a hexadecimal four (4) after a command has been entered.

⑧ CH 5 key

This is used to change the contents of the buffer memory. It also is used to enter a hexadecimal five (5) after a command has been entered.

⑨ INS 6 key

This key is used to store data in the buffer memory. It also is used to enter a hexadecimal six (6) after a command has been entered.

⑩ DEL 7 key

This is used to delete the data in the buffer memory. It also is used to enter a hexadecimal seven (7) after a command has been entered.

⑪ PRG 8 key

This is used to specify the PROGRAM mode. It also is used to enter a hexadecimal eight (8) after a command has been entered.

⑫ [VER 9] key

This is used to specify the VERIFY mode. It also is used to enter a hexadecimal nine (9) after a command has been entered.

⑬ [CONT A] key

This is used to specify the CONTINUANCE mode. It also is used to enter a hexadecimal A after a command has been entered.

⑭ [SERIAL B] key

This is used to specify the SERIAL I/O mode. It also is used to enter a hexadecimal B after a command has been entered.

⑮ [PANEL C] key

This key is used to specify the PANEL mode. It also is used to enter a hexadecimal C after a command has been entered.

⑯ [PTR D] key

This key is used to specify the PRT mode. It also is used to enter a hexadecimal D after a command has been entered.

⑰ [COPY E] key

This key is used to specify the COPY mode. It also is used to enter a hexadecimal E after a command has been entered.

⑱ [BLANK F] key

This key is used to specify the BLANK CHECK mode. It also is used to enter a hexadecimal F after a command has been entered.

⑲ [AUTO/STEP] key

This key is used to switch between the AUTO and STEP modes.

⑳ [A-ENT] key

This key is used to specify the entry of an address or to enter data.

㉑ [↓] key

This key is used to decrement the address.

㉒ [↑] key

This key is used to increment the address.

3.2 Rear Panel

Figure 3.4 shows the rear panel.

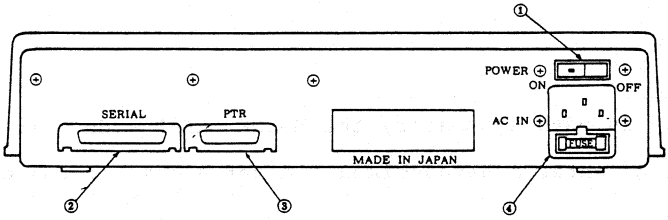


Fig. 3.4 Rear panel

① Power switch

This is a rocker switch. Pressing the left side turns the power on. Pressing the right side turns the power off.

② Serial interface connector

This is an EIA 25-pin (D-sub) connector for connecting the serial interface to an RS-232-C, TTL or current loop interface.

③ Parallel interface connector

This makes a connection to a paper tape reader. The applicable plug is a D-sub (15-pin) plug.

④ AC input connector

This contains a fuse holder (2A fuse) for the AC input.

3.3 Bottom Panel

Figure 3.5 shows the bottom panel switches.

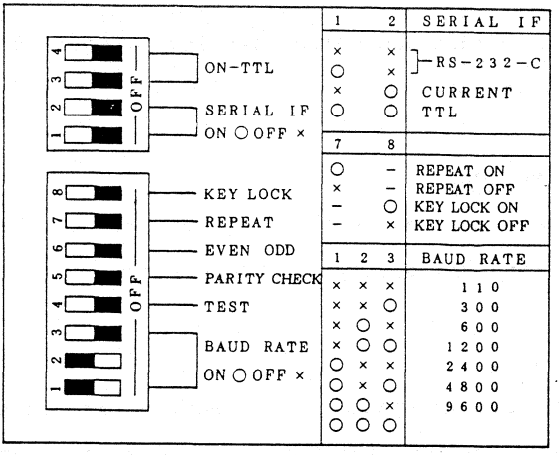
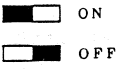


Fig. 3.5 Bottom panel switches

(Note)

In Fig. 3.4, the letters "o" and "x" represent ON and OFF, respectively. (The above figure shows the setting at the time the PG-1000 was shipped from the factory.)



The rear panel switches are used to switch between the serial interface operation modes.

3.3.1 Quadruplet switch

This is used to switch between interfaces.

For the interface and switch settings, refer to Section 4.1 Serial Interface.

3.3.2 Octuplet switch

This is used to specify the transfer rate of the serial interface and the parity check operation.

① Switches 1-3

These are used to specify the transfer rate of the serial interface. For the transfer rate and switch settings, refer to Section 4.1 Serial Interface.

② Switch 4

This is used for checking at the time the PG-1000 is shipped from the factory. Do not turn this switch ON.

③ Switches 5-6

These are used to specify an input/output data parity. Switch 5 is used to specify whether parity checking is to be performed. Switches 6 and 7 are used to specify the switching of EVEN and ODD parities, respectively.

④ Switches 7-8

These are used for checking at the time the PG-1000 is shipped from factory. Do not turn these switches ON.

SECTION 4 EXTERNAL INTERFACES

This chapter explains the rear panel interfaces.

4.1 Serial Interface

The PG-1000 has a start-stop serial interface to connect to the MD-910TM and other consoles or the MD-080/086 and other computers.

There are three interface circuits, namely, RS-232-C, TTL and 20 mA current loop, which can be switched by using the bottom panel quadruplet switch.

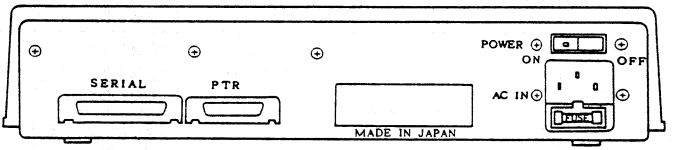


Fig. 4.1 Serial interface connector

4.1.1 Pin arrangements

The pin arrangements on the serial interface connector are as follows.

Table 4.1 Serial interface pin arrangements

Pin No.	Symbol	Type*	Direction PG-outside	Function**
1	FG	Common	——	Frame ground
2	TXD	Common	←——	Receiving data
3	RXD	Common	——→	Sending data
4	RTS	R	←——	Sendable at high level
5	CTS	R	——→	Receivable at high level
6	DSR	R	——→	Receivable at high level
7	SG	Common	——	Signal ground
10	DATA IN/DATA IN+	T/C	←——	Receiving data
11	DATA IN-	C	——→	Return route for receiving data
12	DATA OUT/DATA OUT+	T/C	——→	Sending data
13	DATA OUT-	C	←——	Return route for sending data
14	RS+	C	←——	Return route for paper tape reader control
15	RS/RS-	T/C	——→	Paper tape reader control
16	BUSY	T	←——	External device busy signal
20	DTR	R	←——	Receivable at high level

* Common: Common interface

R: RS-232-C

T: TTL

C: Current loop

** PG-1000 side operation

4.1.2 Interface circuits

(1) TTL interface

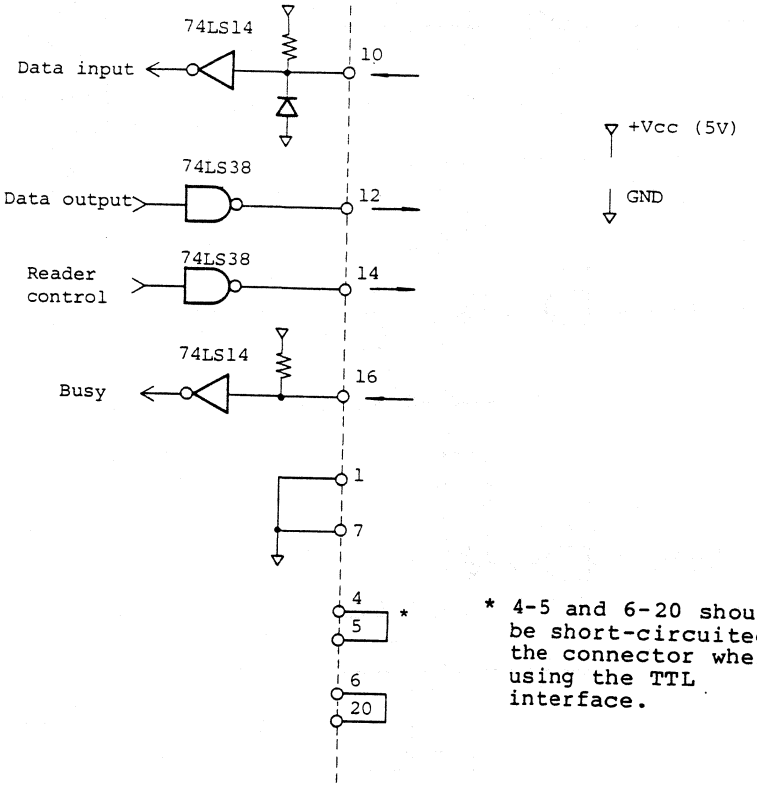


Fig. 4.2 TTL interface circuit

(2) Current loop interface

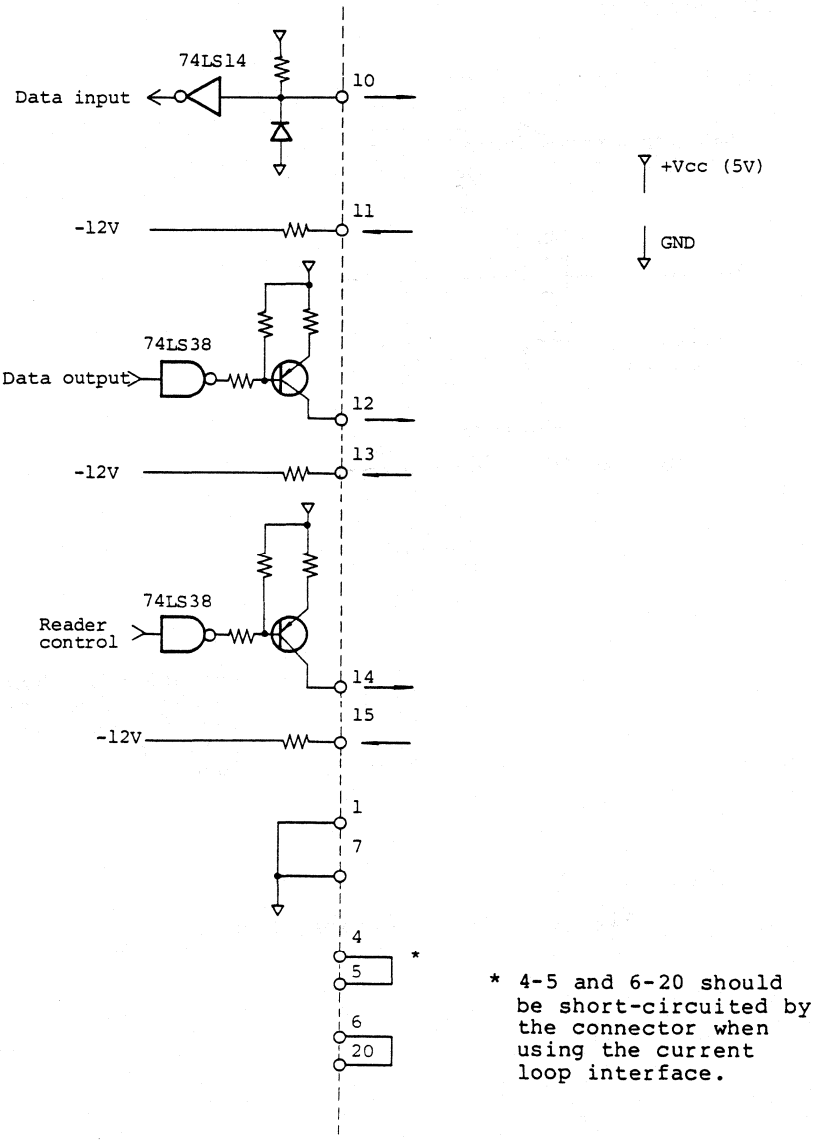


Fig. 4.3 Current loop interface circuit

4.1.3 Interface switching

The RS-232-C, TTL, and current loop interfaces are switched by using the bottom panel quadruplet dip switches. The functions of the dip switches are as follows:

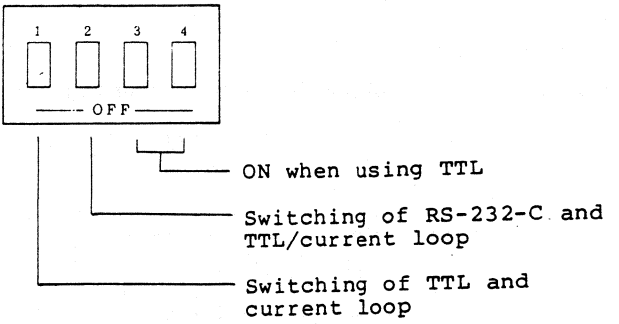
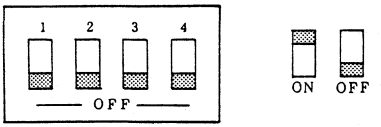


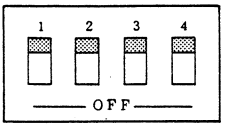
Fig. 4.4 Interface switches

The actual settings are as follows:

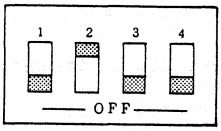
(1) RS-232-C interface



(2) TTL interface



(3) Current loop interface



4.1.4 Baud rate switching

The baud rate can be switched by using the octuplet dip switches (1-3) on the rear panel.

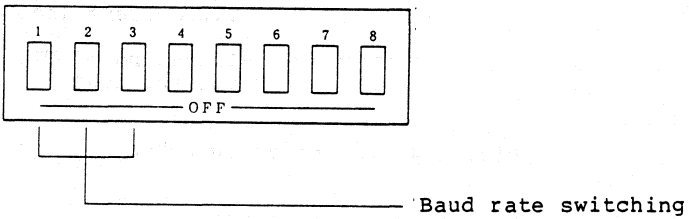
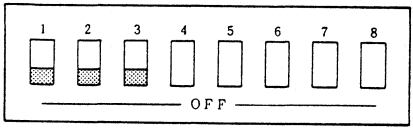
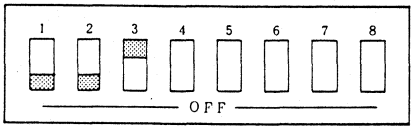


Fig. 4.5 Baud rate switches

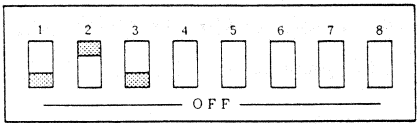
(1) 110 bauds



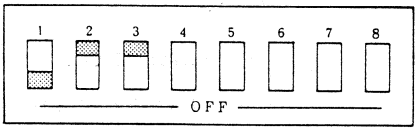
(2) 300 bauds



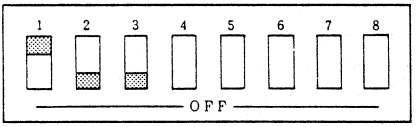
(3) 600 bauds



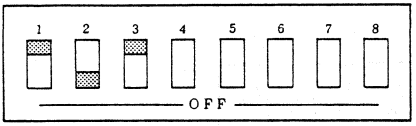
(4) 1200 bauds



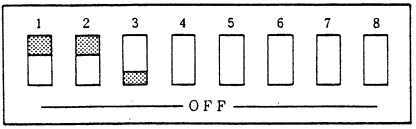
(5) 2400 bauds



(6) 4800 bauds



(7) 9600 bauds



4.1.5 Parity check

Parity checking is specified by using the octuplet dip switches (5 and 6) on the rear panel.

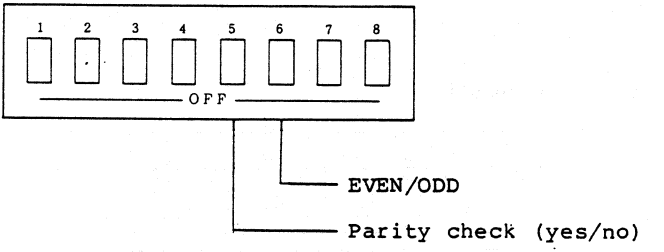
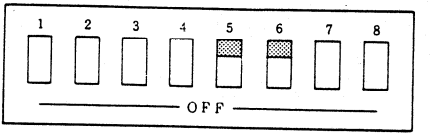
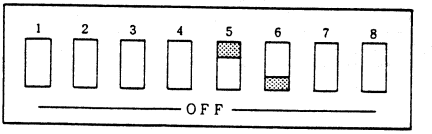


Fig. 4.6 Parity check switches

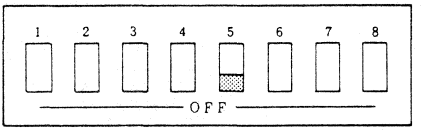
(1) EVEN parity check



(2) ODD parity check



(3) No parity check

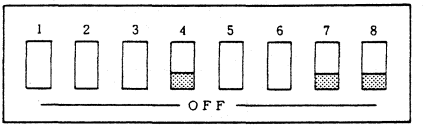


(Note)

Switch 6 may be set either to ON or OFF.

Switches 4, 7 and 8 each must always be set to OFF.

These switches are used for checking at shipment.



4.1.6 Sending/receiving data format

Data is sent or received with the start stop system.

Start bit	One bit
Stop bit	Two bits

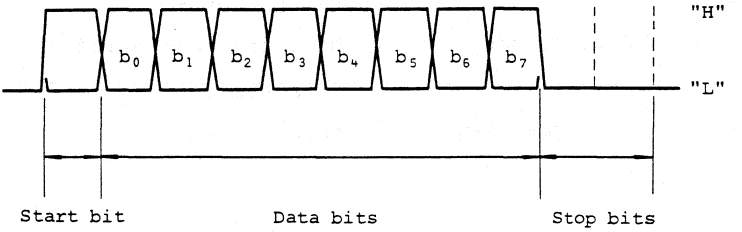


Fig. 4.7 Sending/receiving data format

4.1.7 Handshake system

(1) RS-232-C

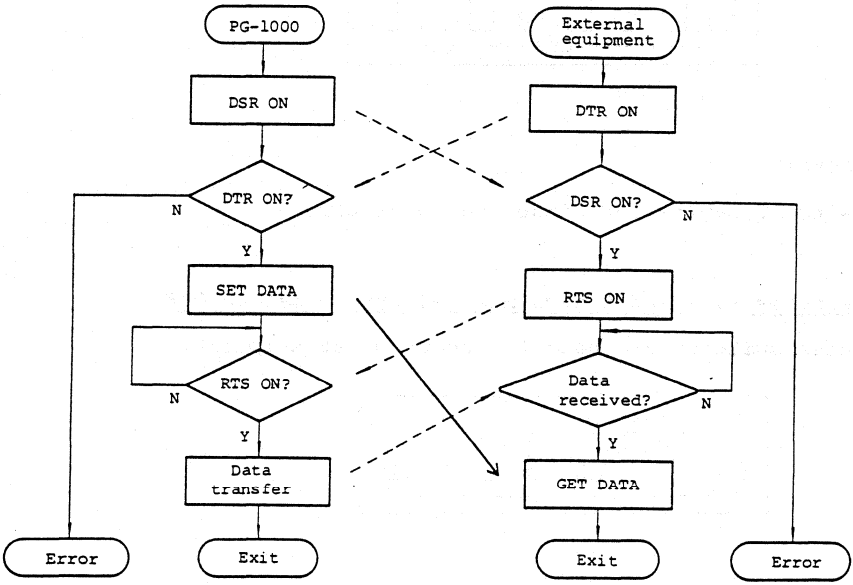


Fig. 4.8 PG-1000 data output

① DSR ON in serial mode
② Check DTR.
③ Set data.
④ Wait for RTS to be set to ON.
⑤ Start data transfer with RTS set to ON.

① DTR ON with power turned ON.
② Check DSR.
③ Set RTS to ON.*
④ Wait for completion of data receiving.
⑤ Receive data.

* The RTS signal is set to ON in the form of pulses.

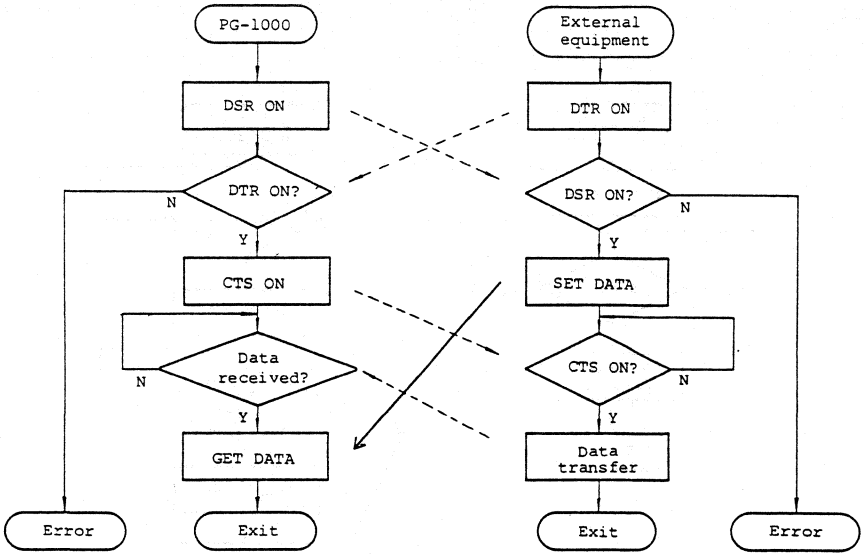


Fig. 4.9 PG-1000 data input

① DSR ON in serial mode
② Check DTR.
③ Set CTS to ON.*
④ Wait for complete of data receiving.
⑤ Receive data.

① DTR ON with power turned ON.
② Check DSR.
③ Set data.
④ Wait for CTS to be set to ON.
⑤ Transfer data with CTS to ON.

* The CTS signal is set to ON in the form of pulses.

(2) TTL

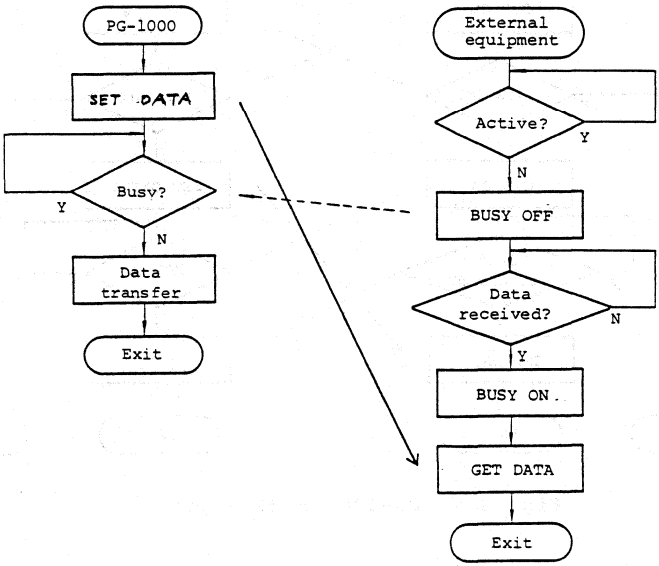


Fig. 4.10 PG-1000 data output

① Set data.
② Check BUSY signal.
③ Transfer data.

① Turn the BUSY signal OFF when the external equipment operations have been completed.
② Turn the BUSY signal ON when data is received.
③ Receive data.

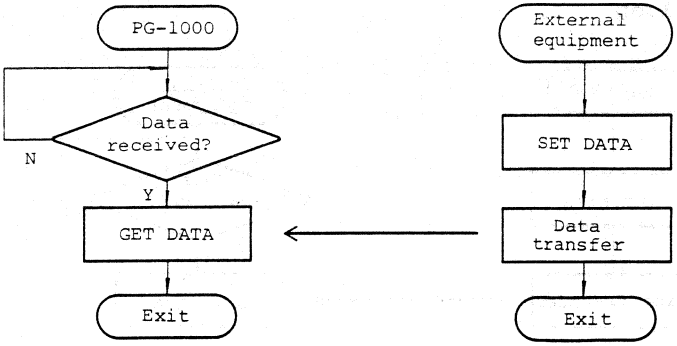


Fig. 4.11 PG-1000 data input

① Check the sending/receiving of data.

② Get data.

① Set data.

② Transfer data.

(Note)

With PG-1000 data input, no handshake signal is supported. In high-speed data transfer, therefore, data may be omitted. In such cases, the equipment must be connected at a lower transfer rate.

(3) Current loop

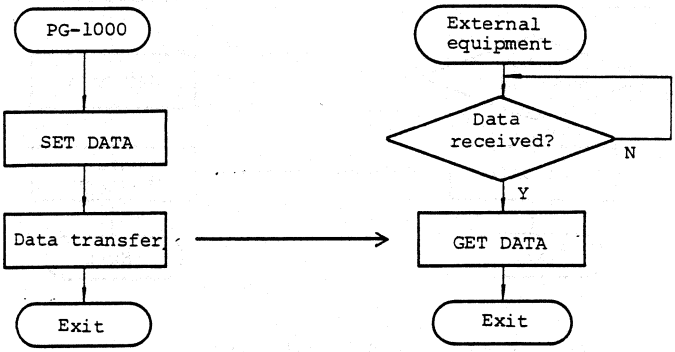


Fig. 4.12 PG-1000 data output

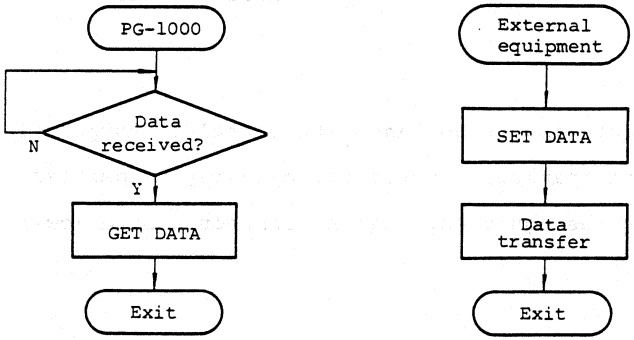


Fig. 4.13 PG-1000 data input

(Note)

The current loop interface does not support the handshake signal. In high-speed data transfer, therefore, data may be omitted. In such cases, the equipment must be connected at a lower transfer rate.

When the TTL or current loop interface is used, no handshake is performed in either direction. It is recommended that the RS-232-C interface be used as much as possible.

4.1.8 Connection

This section explains the method of connection using the RS-232-C interface, taking the MD-910TM as an example.

Set the MD-910TM in the RS-232-C mode and connect it using the following cable:

Table 4.2 Connection table (MD-910TM)

PG-1000		MD-910TM	
Symbol	Pin No.	Pin No.	Symbol
FG	1	1	FG
TXD	2	2	SD
RXD	3	3	RD
RTS	4	4	RS
CTS	5	5	CS
DSR	6	6	DR
SG	7	7	SG
DTR	20	21	RR Note)

(Note)

The RS-232-C specifications require connection to the ER signal of pin No. 20. But the MD-910TM uses the above connection.

4.2 Parallel Interface

The PG-1000 has a two-wire handshake eight-bit parallel input interface.

Negative TTL logic is used.

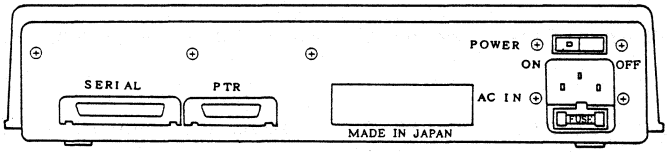


Fig. 4.14 Parallel interface connector

4.2.1 Pin arrangements

Table 4.3 Parallel interface pin arrangements

Pin No.	Symbol	Direction PG-outside	Function
1	$\overline{\text{DRIVE}}$	——→	Start signal from PG to external equipment
2	$\overline{\text{SPROCKET}}$	←——	Acknowledgement signal from external equipment to PG
3	$\overline{\text{DATA0}}$	←——	Input data (bit 0)
4	$\overline{\text{DATA1}}$		ditto (bit 1)
5	$\overline{\text{DATA2}}$		ditto (bit 2)
6	$\overline{\text{DATA3}}$		ditto (bit 3)
7	$\overline{\text{DATA4}}$		ditto (bit 4)
8	$\overline{\text{DATA5}}$		ditto (bit 5)
9	$\overline{\text{DATA6}}$		ditto (bit 6)
10	$\overline{\text{DATA7}}$		ditto (bit 7)
11	GND	——	Signal ground
12			
13			
14			
15			

4.2.2 Two-wire handshake

The following is the method of handshaking in the parallel interface.

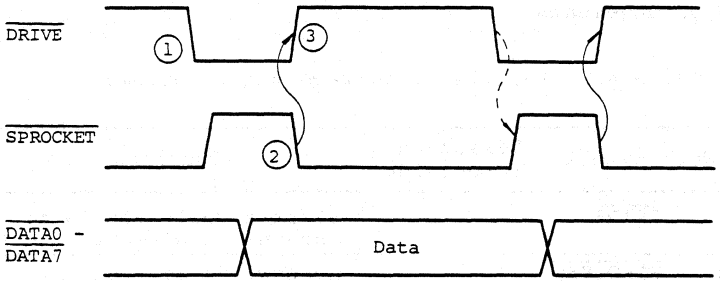


Fig. 4.15 Parallel interface timing

① If the parallel interface is activated, the $\overline{\text{DRIVE}}$ signal is made active (low level). The external equipment starts operation with the above signal.

② When the data setting operation has been completed, the external equipment makes the $\overline{\text{SPROCKET}}$ signal active.

③ With the fall the $\overline{\text{SPROCKET}}$ signal, the PG-1000 detects the completion of operations of the external equipment. It then makes the $\overline{\text{DRIVE}}$ signal non-active and stops the operation. At the same time, it reads the data from $\overline{\text{DATA0}}$ to $\overline{\text{DATA7}}$.

4.2.3 Connections

(1) DPT-6A (Anritsu Electric) negative logic specifications

Use the following cable to connect paper tape reader DPT-6A:

Table 4.4 Connection cable (DPT-6A)

PG-1000		DPT-6A	
Symbol	Pin No.	Pin No.	Symbol
$\overline{\text{DRIVE}}$	1	20	STA
$\overline{\text{SPROCKET}}$	2	4	SPR
$\overline{\text{DATA 0}}$	3	1	RD1
$\overline{\text{DATA 1}}$	4	2	RD2
$\overline{\text{DATA 2}}$	5	3	RD3
$\overline{\text{DATA 3}}$	6	5	RD4
$\overline{\text{DATA 4}}$	7	6	RD5
$\overline{\text{DATA 5}}$	8	7	RD6
$\overline{\text{DATA 6}}$	9	8	RD7
$\overline{\text{DATA 7}}$	10	9	RD8
GND	11 to 15	10 to 15	GND

(Note)

The FWD signal (pin 24) of the DPT-6A should be kept open.

(2) DPT-7A/B (Anritsu Electric)

The DPT-7A/B should be connected by using the following cable.

Table 4.5 Connection cable (DPT-7A/B)

PG-1000		DPT-7A/B	
Symbol	Pin No.	Pin No.	Symbol
$\overline{\text{DRIVE}}$	1	11	STA
$\overline{\text{SPROCKET}}$	2	4	SPR
$\overline{\text{DATA 0}}$	3	1	RD1
$\overline{\text{DATA 1}}$	4	2	RD2
$\overline{\text{DATA 2}}$	5	3	RD3
$\overline{\text{DATA 3}}$	6	5	RD4
$\overline{\text{DATA 4}}$	7	6	RD5
$\overline{\text{DATA 5}}$	8	7	RD6
$\overline{\text{DATA 6}}$	9	8	RD7
$\overline{\text{DATA 7}}$	10	9	RD8
GND	11 to 15	27, 28	GND

SECTION 5 CONNECTION TO MD-080/086

This chapter explains the method of connecting the MD-080/086 host machine to the PG-1000 used as a PROM programmer.

5.1 Selection of MD-080/086 Serial Channels

The MD-080/086 has five standard serial channel and optional four channels. Of these, the following can be connected to the PG-1000.

Table 5.1 MD-080/086 serial channels

Channel No.	MD-080	(Note) MD-086	
1	o	o	
2	o	o	
3	o	x	
4	o	x	
5	o	x	┐
6	o	x	Option channel
7	o	x	
8	o	x	┘
9	o	o	

o: Connectable
x: Not connectable

(Note)

This column shows the channel available when the MP/M-86* is used as the OS (operating system) of the MD-086. If the OS is switched over to CP/M by using the 'CPM' command, all channels can be used, as with the MD-080.

Select one of the channels available for connection with the PG-1000. However it is not desirable to use channel '9' as explained later.

5.2 MD-080/086 Jumper Setting

Set the MD-080/086 serial channel jumpers in the RS-232-C mode. For details, refer to the following material:

- o MD-080FD Hardware Manual
- o MD-086FD Hardware Manual
- o MD-080FD-10 Hardware Manual
- o MD-086FD-10 Hardware Manual

5.2.1 Setting of MD-080/086-10

(1) Channels 1 to 4

Channels 1 to 4 are set using the jumpers on the rear panel I/O board.

(a) RS-232-C/TTL switching jumpers

Open the jumper of the channel being used.

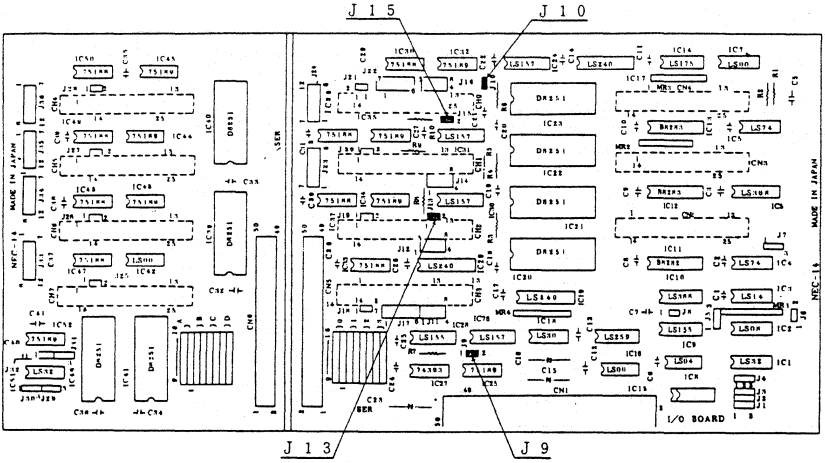


Fig. 5.1

Table 5.2

Channel No.	Jumper	Short pin
1	J10	Open
2	J15	
3	J13	
4	J9	

(b) Baud rate selection jumper

Set the baud rate to 9600 bauds.

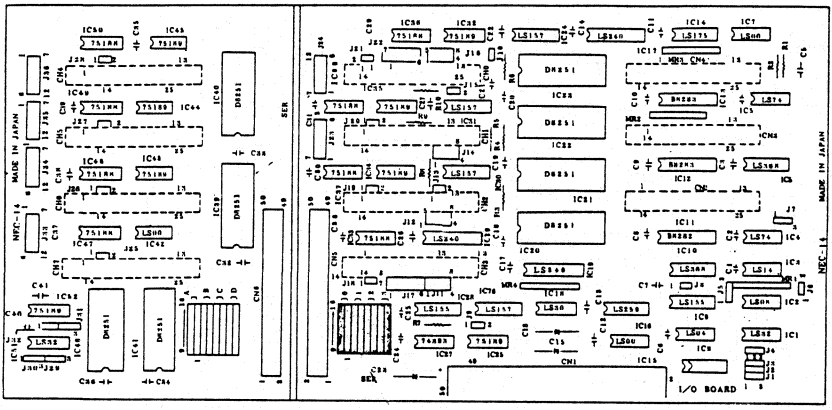


Fig. 5.2

Table 5.3

Channel No.	Jumper	Short pin
1	0	1 to 9
2	1	
3	2	
4	3	

(c) Modem/terminal mode selection jumpers

Set the channel being used in the modem mode.

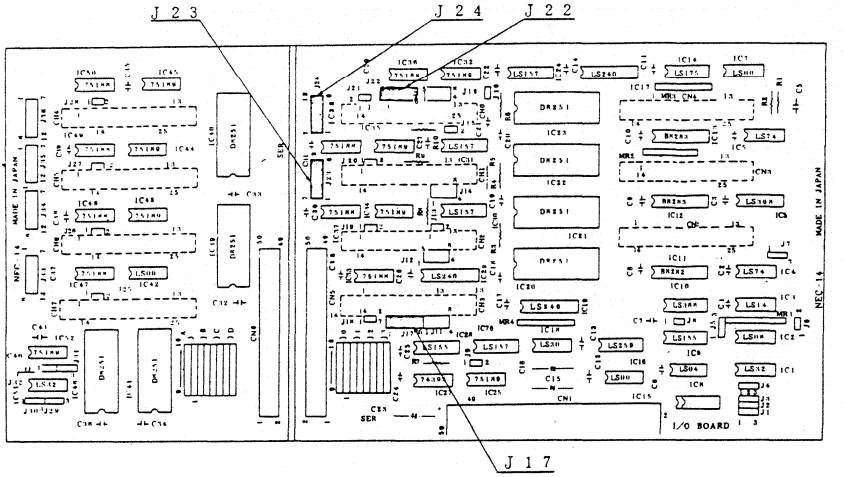


Fig. 5.3

Table 5.4

Channel No.	Jumper	Short pin
1	J22	1-2, 3-4 5-6, 7-8 9-10, 11-12
2	J24	
3	J23	
4	J17	

(d) RTS selection jumper

Output the RxRDY signal.

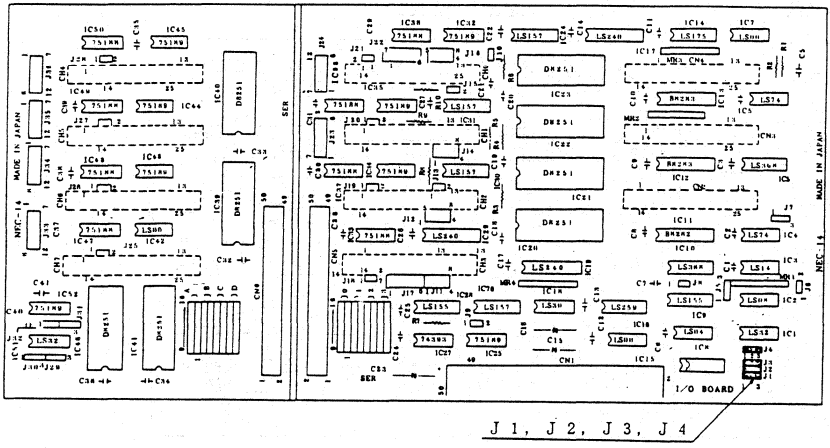


Fig. 5.4

Table 5.5

Channel No.	Jumper	Short pin
1	J4	2-3
2	J3	
3	J2	
4	J1	

(2) Channels 5 to 8

Channels 5 to 8 are set using the jumpers on the rear panel I/O board.

(a) Baud rate selection jumper

Set the baud rate to 9600 bauds.

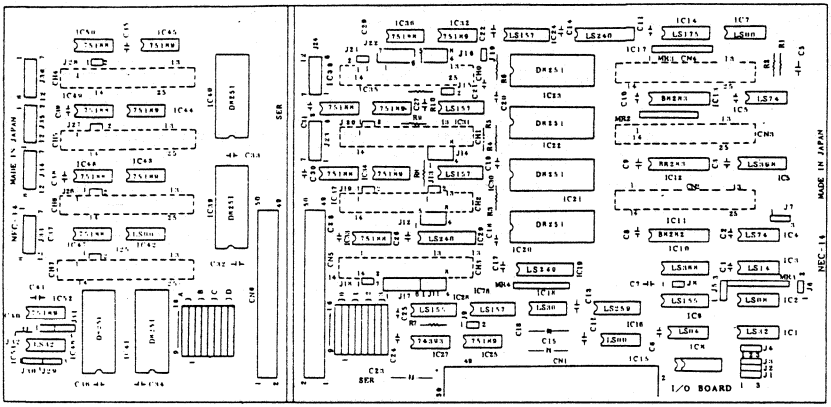


Fig. 5.5

Table 5.6

Channel No.	Jumper	Short pin
5	A	1-9
6	B	
7	C	
8	D	

(b) Modem/terminal mode selection jumpers

Set the channel being used in the modem mode.

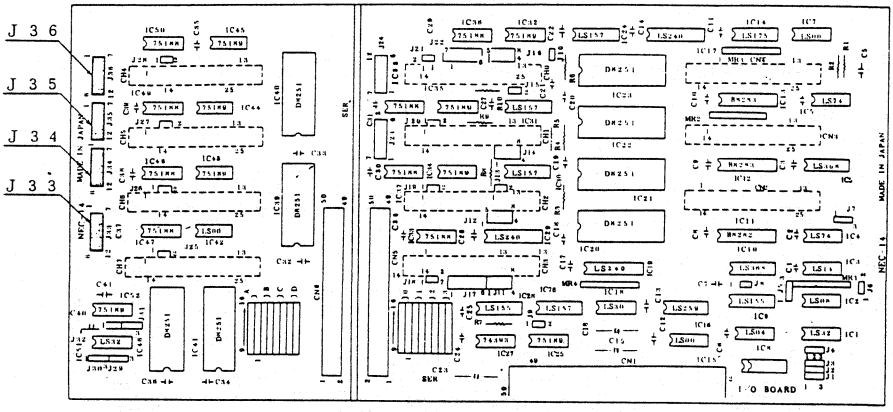


Fig. 5.6

Table 5.7

Channel No.	Jumper	Short pin
5	J36	1-2, 3-4
6	J35	5-6, 7-8
7	J34	9-10, 11-12
8	J33	

(c) RTS selection jumper

Output the RxRDY signal.

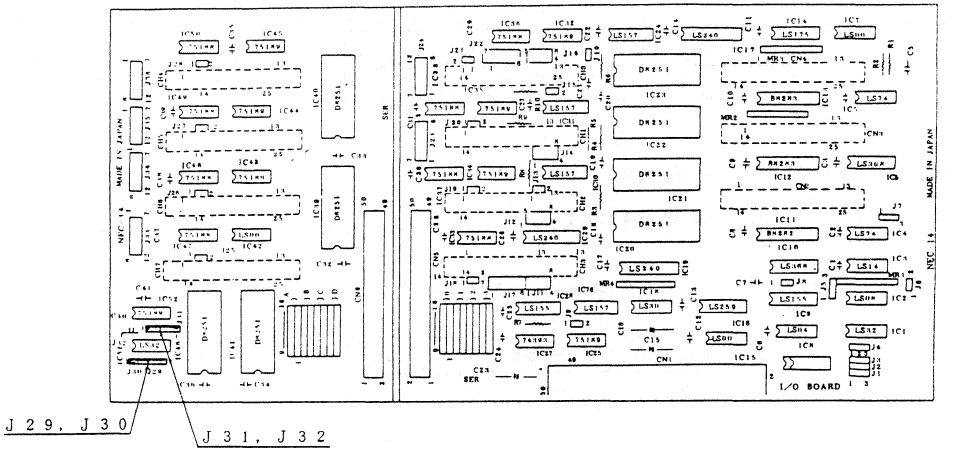


Fig. 5.7

Table 5.8

Channel No.	Jumper	Short pin
5	J32	2-3
6	J31	
7	J29	1-2
8	J30	

(3) Channel 9

Channel 9 is set by switching the jumpers on SCB in the MD-080/086-10.

(a) Baud rate selection jumper

Set the baud rate to 9600 bauds.

Table 5.9

Channel No.	Jumper	Short pin
9	JP3	1-12

(b) RTS selection jumper

Output the RxRDY signal.

Table 5.10

Channel No.	Jumper	Short pin
9	JP5	1-2

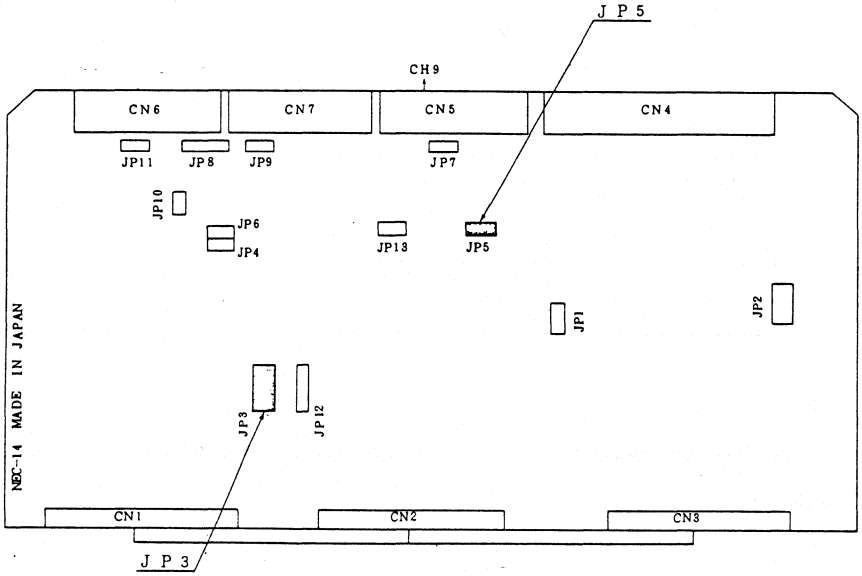


Fig. 5.8

5.2.2 Setting the MD-080/086

(1) Channels 1 to 4

Channels 1 to 4 are set by switching the jumpers on the rear panel I/O board.

(a) Baud rate selection jumper

Set the baud rate to 4800 bauds.

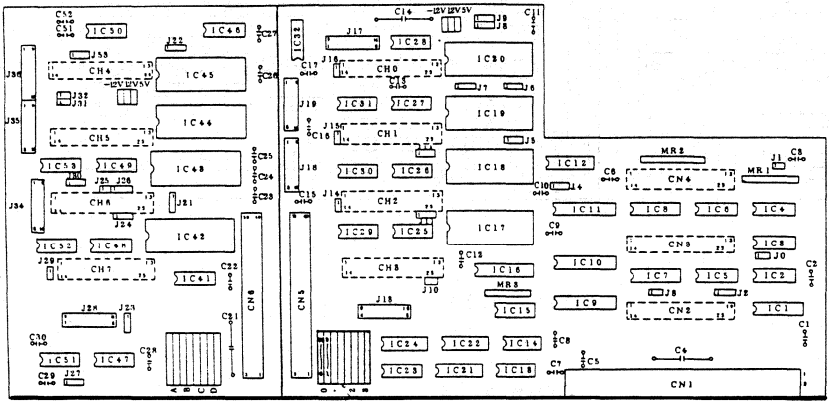


Fig. 5.9

Table 5.11

Channel No.	Jumper	Short pin
1	0	2-10
2	1	
3	2	
4	3	

(b) Modem/terminal mode selection jumper

Set the channel being used in the modem mode.

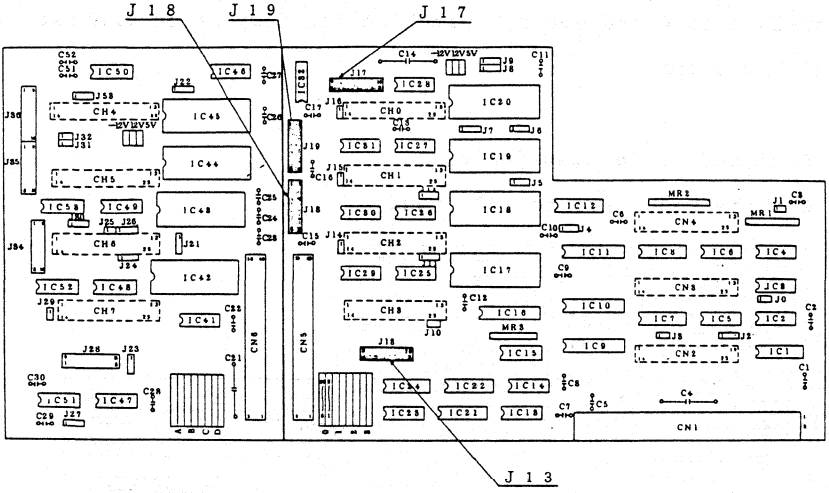


Fig. 5.10

Table 5.12

Channel No.	Jumper	Short pin
1	J17	1-2, 3-4
2	J19	5-6, 7-8
3	J18	9-10, 11-12
4	J13	13-14, 15-16

(c) RTS selection jumper

Output the RxRDY signal.

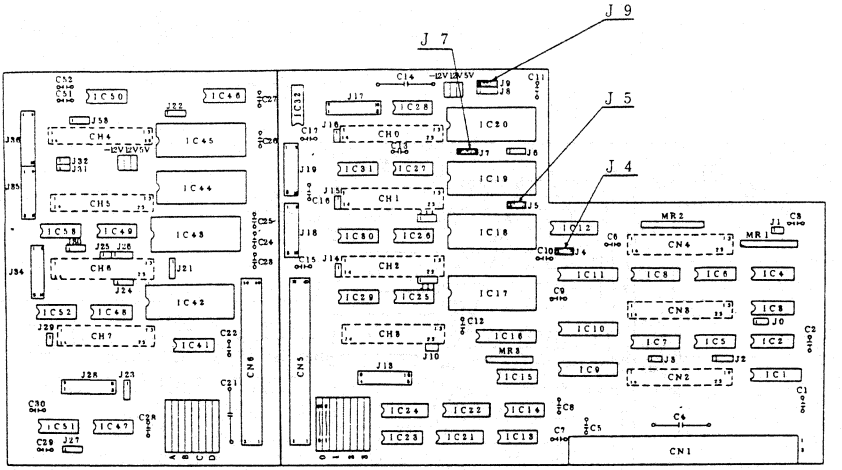


Fig. 5.11

Table 5.13

Channel No.	Jumper	Short pin
1	J9	1-2
2	J7	
3	J5	
4	J4	

(d) Clock selection jumper

Select the internal clock.

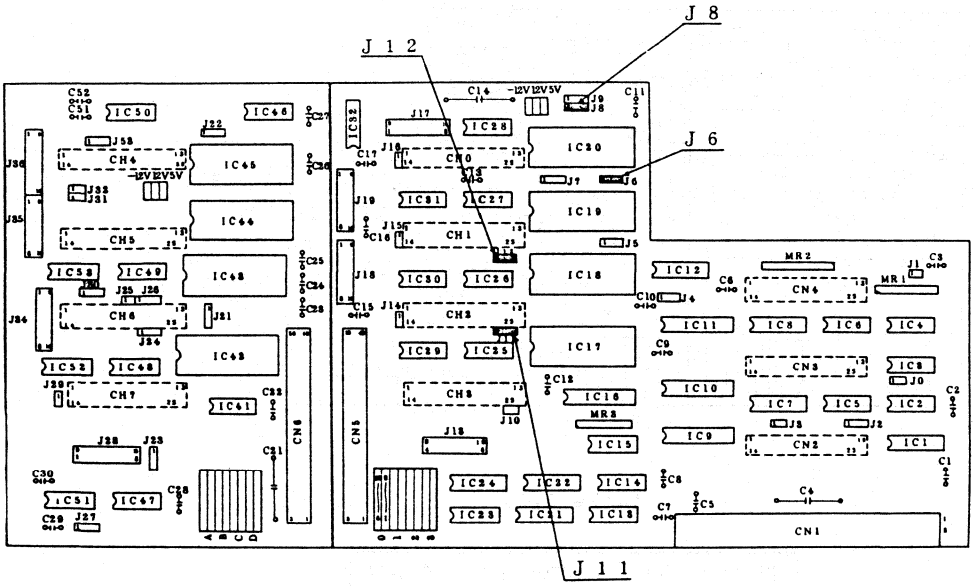


Fig. 5.12

Table 5.14

Channel No.	Jumper	Short pin
1	J8	2-3
2	J6	
3	J12	
4	J11	

(2) Channels 5 to 8

Channels 5 to 8 are set by switching the extended system I/O jumpers on the rear panel.

(a) Baud rate selection jumper

Set the baud rate to 4800 bauds.

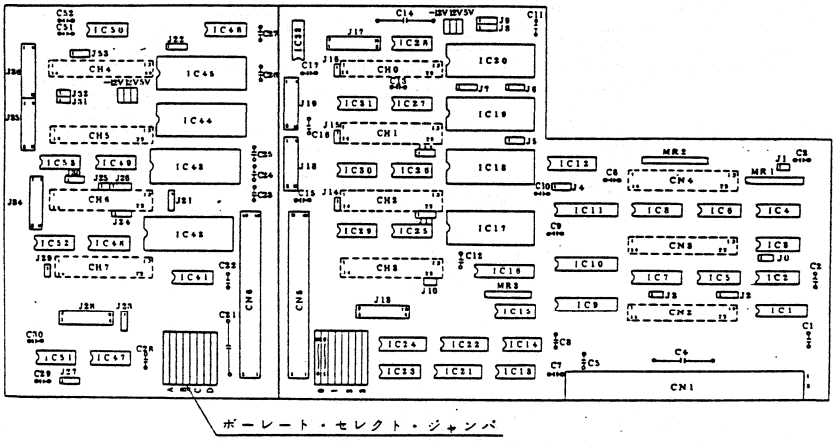


Fig. 5.13

Table 5.15

Channel No.	Jumper	Short pin
5	A	2-10
6	B	
7	C	
8	D	

(b) Modem/terminal mode selection jumpers

Set the channel being used in the modem mode.

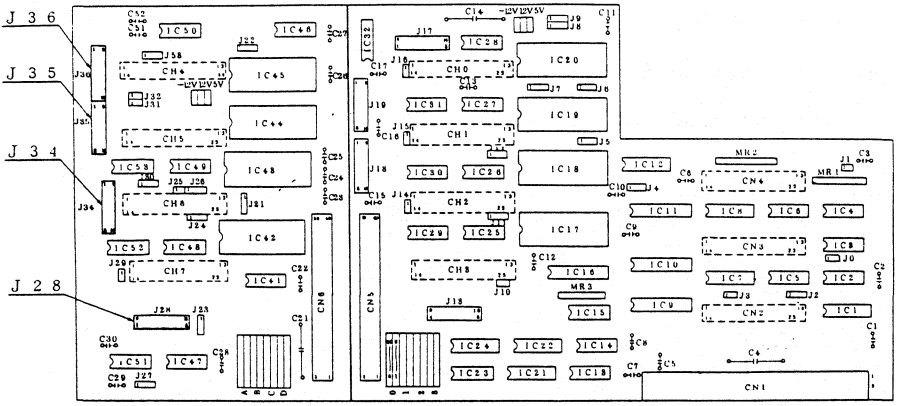


Fig. 5.14

Table 5.16

Channel No.	Jumper	Short pin
5	J36	1-2, 3-4
6	J35	5-6, 7-8
7	J34	9-10, 11-12
8	J28	13-14, 15-16

(c) RTS selection jumpers

Output the RxRDY signal.

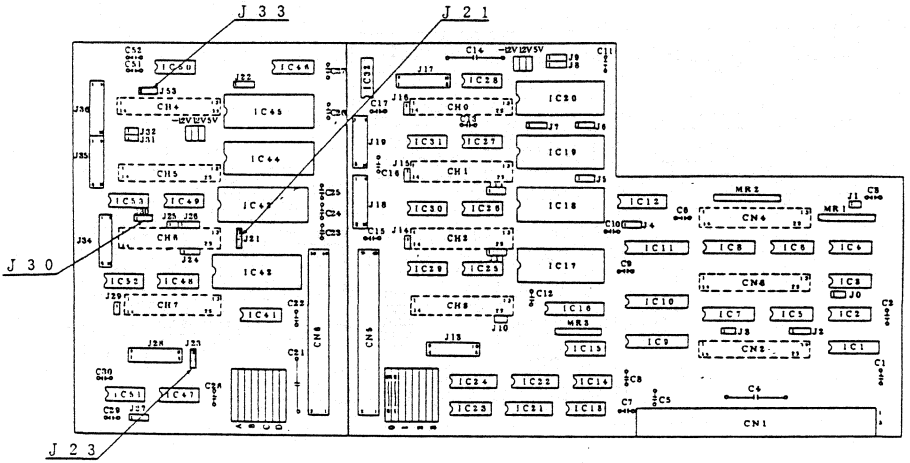


Fig. 5.15

Table 5.17

Channel No.	Jumper	Short pin
5	J33	1-2
6	J30	
7	J21	
8	J23	

(d) Clock selection jumpers

Select the internal clock.

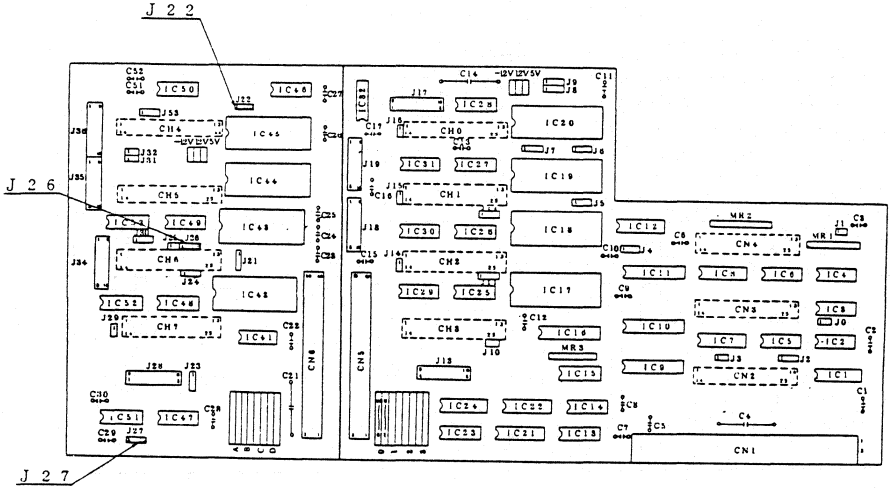


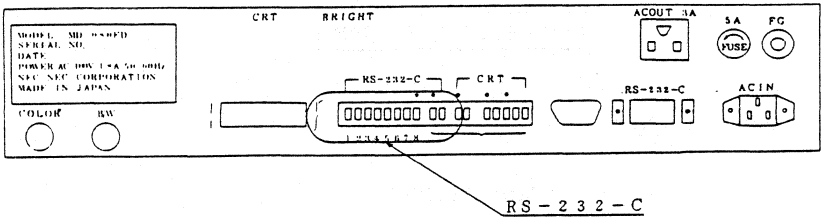
Fig. 5.16

Table 5.18

Channel No.	Jumper	Short pin
5	J22	2-3
6	J26	
7	J24	
8	J27	

(3) Channel 9

Channel 9 is set by switching the jumpers on the lower part of the rear panel.



RS-232-C jumper

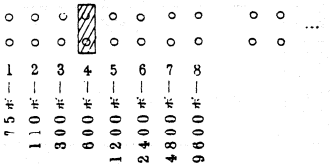


Fig. 5.17

5.3 Setting the PG-1000

Set the dip switches at the bottom of the PG-1000 as follows:

RS-232-C

600/4800/9600 bauds

Quadruplet switches

Table 5.19

Switch Number			
1	2	3	4
OFF	OFF	OFF	OFF

Octuplet switches

Table 5.20

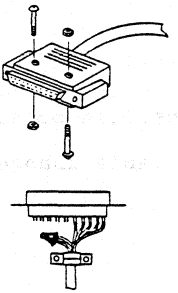
Host	Channel number	Switch number							
		1	2	3	4	5	6	7	8
MD-080 MD-086	1 to 8	ON	OFF	ON	OFF				
	9	OFF	ON	OFF					
MD-080-10 MD-086-10	1 to 8	ON	ON	OFF					
	9	ON	ON	OFF					

5.4 Cable Connection

When MD-080/086 and PG-1000 operations have been completed, connect the two units using the cable attached to the PG-1000. If channel 9 is used, however, the cable must be changed by the following procedure:

(1) Remove the MD-080/086-side RS-232-C connector cover.

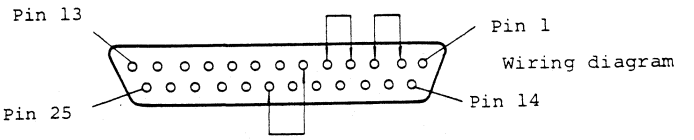
The difference between the modem mode cable and terminal mode cable is as follows:



PG-1000 side	MD-080/086 side	
	Channel 1 to 8	Channel 9
2 (RxD)	3 (TxD)	2 (TxD)
3 (TxD)	2 (RxD)	3 (RxD)
4 (CTS)	5 (RTS)	4 (RTS)
5 (RTS)	4 (CTS)	5 (CTS)
6 (DTR)	20 (DSR)	6 (DSR)
7 (SG)	7 (SG)	7 (SG)
20 (DSR)	6 (DTR)	20 (DTR)

The cable is originally set for use with channels 1 to 8.

(2) Change the wiring of pins 2-3, 4-5 and 6-20.



(Change the arrowed wiring)

(3) Attach the connector cover by reversing the procedure explained in (1) above.

(4) Remove the seal stick from the connector cover.

(Note)

Channels 1 to 8 should be used as much as possible because there are problems with channel 9 with respect to cable changes.

SECTION 6 PERSONAL MODULES

The PG-1000 can be used as a single-chip microcomputer with a variety of built-in PROMs or a bipolar PROM programmer by changing parts called personal modules.

For devices which can be programmed and their operations, refer to the manual of each personal module.

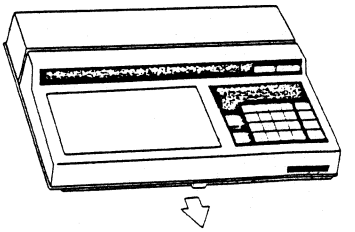
At present, the following modules are available.

Module	Device
PG-1001	μPD8755A, μPD8741A, μPD8748, μPD8748H, μPD8749H
PG-1002	μPB403, μPB423, μPB406, μPD426 μPB405, μPB425, μPB428, μPB417 μPB409, μPB429, μPB419
PG-1003	μPD78PQ9R

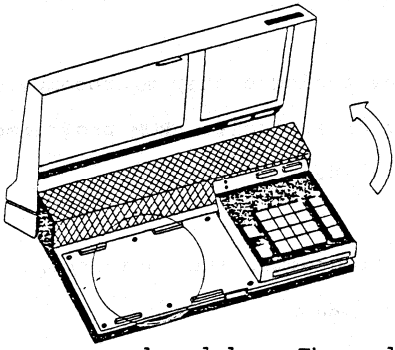
The method of changing personal modules is explained below.

① Remove the PROM, turn off the power switch and pull out the power cable.

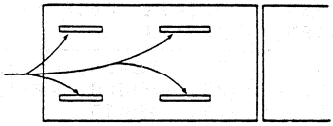
② Pull the catch at the lower part of the front side of the PG-1000 and unlock it.



③ Lift the case cover and remove the personal module from the left side.



④ Insert another personal module. The module must be fixed with the four connectors shown below.



⑤ Check the operations of the personal module using its manual.

PART II OPERATION

SECTION 1 OPERATION OVERVIEW

As explained in Part I, the PG-1000 can be used in the following three modes:

1. Host connection mode
2. Console operation mode
3. Stand-alone mode

In the host connection mode, the PG-1000 can be connected to an MD-080/086 host machine, making it possible to write programs developed on the MD-080/086 into a PROM. The contents of a master PROM can also be saved in the form of a HEX- or extended HEX-type disk file.

In the console operation mode, the PG-1000 can be connected to a console and controlled from it.

In the stand-alone mode, the PG-1000 can be used by itself for master PROM copying, program patching, etc. Because of its compact, portable design, the product can be used very effectively for PROM programming in a production line.

SECTION 2 HOST CONNECTION MODE

This chapter explains how to use the PG-1000 when it is connected to an MD-080/086 host machine.

2.1 General

In this mode, it is possible to connect the PG-1000 to the serial channel of the MD-080/086 and thereby control the PG-1000 from the console of the MD-080/086 using the PROM programmer control program called 'PGC'.

This mode provides functions for reading HEX- or extended HEX-type files stored on a floppy disk and writing them into a PROM, reading PROM data and storing it in the form of a HEX- or extended HEX-type file in a floppy disk, as well as the basic functions of PROM copying (read out), blank check (checking of deletion), programming (writing), verification (comparison), etc.

Moreover, if the auto mode is used, it is possible to considerably reduce the user's operations when floppy disk files are being written into a PROM.

2.2 Setting

Connect the serial channel of the MD-080/086 to the serial interface of the PG-1000 using the attached cable. For details, refer to PART I, CHAPTER 5 Connection with the MD-080/086.

2.3 Starting

2.3.1 Starting the MD-080/086

If the MD-080/086 has been properly set, turn the MD-080/086 power switch on. Then, set the system disk in drive 'A' of the MD-080/086 and reset the machine. With this, drive 'A' is started and CP/M or the MP/M-86 OS is activated with a displayed prompt ('A>' for CP/M; 'OA>' for MP/M-86).

2.3.2 Starting the PG-1000

Turn the PG-1000 power switch on. After about two seconds a buzzer sounds. Then select a PROM. The method of selecting a PROM differs slightly for each personal module. Please refer to the manual of the personal module being used.

After selecting a PROM, press the SERIAL key. The 'SERIAL' LED on the front panel turns on.

(Note)
When pressing the SERIAL key, if there is no power to the MD-080/086, the address display indicates 'E-11'. In this case, start again from 2.3.1. If 'E-11' is indicated even with power supplied to the MD-080/086, the machine may be out of order. In such cases, turn off the power, remove the cable and send the machine for repair.

2.3.3 Starting the program (PGC)

If the MD-080/086 and PG-1000 have been started, then start the PG-1000 control program (PGC) from the console of the MD-080/086 in the following manner. Set the floppy disk that stored the program in the drive and perform the following key entries:

MP/M-86

```
0A>[d:]PGC[␣n]↵   (n='1','2','9')
```

CP/M

```
A>[d:]PGC[␣n]↵    (n='1' - '9')
```

↵ RETURN key

␣ Key entry

[ ] Omissible

d Drive number d='A','B', ...

n Channel number

Active the PGC stored in the disk set in drive d and use channel n.

If 'd:' is omitted, the log-in disk is selected.

If '␣n' is omitted or if a character other than '1' to '9' is entered for n, channel '1' is selected as the default.

(Example)

Suppose CP/M is used.

A>PGC␣3↵ ... Activate the program on the login drive and use channel 3.

A>C:PGC↵ ... Activate the program on drive C and use channel 1.

The following are the channels to be used by the user and the method of starting the program. (It is assumed that the program is stored in the login disk.)

Table 2.1 Command starting methods

Channel number	Activating method
1	PGC↵ PGC␣1↵
2	PGC␣2↵
3	PGC␣3↵
4	PGC␣4↵
5	PGC␣5↵
6	PGC␣6↵
7	PGC␣7↵
8	PGC␣8↵
9	PGC␣9↵

With the above operation, the program is activated and communication with the PG-1000 is started. Communication is established in a few seconds and the following message is then displayed.

```
PG CONTROLLER (MD-086 SERIES) V1.0 [26 Jun 84]
   Copyright (C) 1984 NEC Corporation

ROM SELECT
(PG1000):
```

(Note)

This message is for Version 1.0.

If 'ROM SELECT' and the subsequent message are not displayed, please check the jumper setting, cable connection, PG-1000 operation mode and channel specification at the time the program was started.

If the above message is not displayed, it is possible to transfer control to the OS by entering CTRL-C. Since a PROM has been selected in 2.3.2, press the RETURN key on the MD-080/086 console.

```
ROM SELECT
(PG1000):↵
```

If the RETURN key has been pressed, initialization is performed and after a few seconds the following message is displayed:

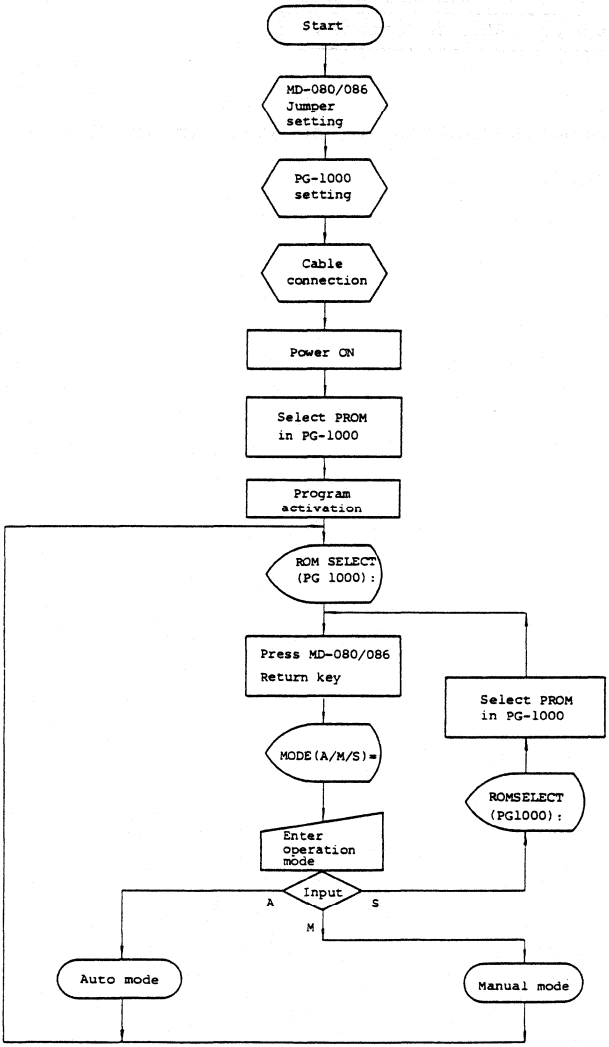
```
MODE(A/M/S)=
```

This is to specify either of the 'PGC' operation modes or to select a PROM again.

Key entry	Operation
A	Move to the auto mode.
M	Move to the manual mode.
S	Move to PROM selection.

If S is specified, the program changes to the state in which a PROM can be selected again.

Program start and processing flow



2.4 Auto Mode

If the auto mode is selected, the following message is displayed:

```
MODE(A/M/S)=A↵
*** AUTO MODE ***
```

2.4.1 Specifying a file name

Following the auto mode message, the following message is displayed, prompting the entry of a file name:

```
SOURCE=
```

Then, enter a file name in the following format:

```
[d:]filename[.[typ]]↵
```

d ---------- Drive (If omitted, the login drive is assumed.)

filename --- File name

typ -------- File type HEX or H86
(If omitted, HEX is assumed.)

(Example)

Specify 'TEST.H86' of the login drive:

```
SOURCE=TEST.H86↵
```

Specify 'T0.HEX' of drive B:

```
SOURCE=B:T0.HEX↵
SOURCE=B:T0.↵
SOURCE=B:T0↵
```

2.4.2 Specifying the separation of odd and even addresses

It is then necessary to specify whether to separate the addresses for writing. Since the following entry request message is displayed, enter 'Y' if separate writing is to be performed. Otherwise, enter 'N'.

```
CONV(Y/N)=
```

(Example)
Separate writing

```
CONV(Y/N)=Y↵
```

Normal writing

```
CONV(Y/N)=N↵
```

Returns to operation mode set if a character other than 'Y' or 'N' is input.

```
CONV(Y/N)=A↵
MODE(A/M/S)=
```

2.4.3 Loading

If the separation of addresses is specified, the file is read in. If the specified file does not exist, an error results and the message requesting the entry of a file name is displayed again.

```
SOURCE=T0.HEX↵
CONV(Y/N)=N↵
LOADING
NO SUCH FILE

SOURCE=
```

File loading continues until the state of the internal buffer satisfies any one of the following conditions:

① The number of data records read exceeds the capacity of the PROM.

② Addresses become discontinued.

③ EOF (end of file) of the file is detected.

When the loading has been completed, information about the data is displayed. The information includes the start address and end address (20-bit address) of the data, the number of data records that were read, and the write address (16-bit address) in the PROM.

```
LOADING
START ADDRESS=00000
STOP  ADDRESS=000FF
DATA  LENGTH = 0100

ROM ADDRESS=0000-00FF
```

.4 Specifying writing

When the loading has been completed, the following message is displayed, asking whether the data read is to be written in the PROM:

```
(W:WRITE/P:PASS/A=ABORT)=
```

Here, the following three entries are available.

Key entry	Operation
W	Write the data into the PROM.
P	Do not write the data.
A	Return to the mode setting.

If a character other than 'W', 'A' or 'P' is entered, the same message is output again.

If 'W' is entered, the following message is displayed, requesting the setting of the PROM.

```
(W:WRITE/P:PASS/A:ABORT)=W)
SET NEW ROM, THEN TYPE (CR)
```

Then, set a cleared PROM and press the RETURN key. This starts the writing. Pressing any key other than the RETURN key causes the system to display the first message again as follows:

```
(W:WRITE/P:PASS/A:ABORT)=W)
SET NEW ROM, THEN TYPE (CR)A
(W:WRITE/P:PASS/A:ABORT)=
```

After writing starts, files are automatically read and written as long as the capacity of the PROM is not exceeded.

If the capacity of the PROM is exceeded, the system asks whether the writing operation be continued in a new PROM. If so, the above operation is repeated until an EOF is detected.

(Example)

No separation of even/odd addresses

```
*** AUTO MODE ***

SOURCE=TEST.HEX)              --- File name specification
CONV(Y/N)=N)                  --- No address separation
LOADING
START ADDRESS=00000
STOP  ADDRESS=00006
DATA  LENGTH = 0007

ROM ADDRESS=0000-0006                    Writing in first block

(W:WRITE/P:PASS/A:ABORT)=W)
SET NEW ROM, THEN TYPE (CR)
TRANSFER
WRITING
LOADING
START ADDRESS=00011
STOP  ADDRESS=00016
DATA  LENGTH = 0006
                                         Writing in second block
ROM ADDRESS=0011-0016

TRANSFER
WRITING
LOADING
START ADDRESS=00021
STOP  ADDRESS=00025
DATA  LENGTH = 0005
                                         Writing in third block
ROM ADDRESS=0021-0025

TRANSFER
WRITING
LOADING
END OF FILE                   --- Terminate when EOF is detected.
```

Separation of even/odd addresses

If the separation of addresses is specified, the data in even number addresses is written and then the data in odd number addresses as follows:

```
*** AUTO MODE ***

SOURCE=TEST.HEX)          --- File name specification
CONV(Y/N)=Y)              --- Specification of separate writing
LOADING
START ADDRESS=00000
STOP  ADDRESS=00006
DATA  LENGTH = 0004
EVEN

ROM ADDRESS=0000-0003

(W:WRITE/P:PASS/A:ABORT)=W)
SET NEW ROM, THEN TYPE (CR))

TRANSFER
WRITING
LOADING
START ADDRESS=00012
STOP  ADDRESS=00016
DATA  LENGTH = 0003
EVEN                                  Writing of even-numbered
                                      addresses
ROM ADDRESS=0009-000B

TRANSFER
WRITING
LOADING
START ADDRESS=00022
STOP  ADDRESS=00024
DATA  LENGTH = 0002
EVEN

ROM ADDRESS=0011-0012

TRANSFER
WRITING
LOADING
END OF FILE
```

```
LOADING
START ADDRESS=00001
STOP  ADDRESS=00005
DATA  LENGTH = 0003
ODD

ROM ADDRESS=0000-0002

(W:WRITE/P:PASS/A:ABORT)=W↵
SET NEW ROM, THEN TYPE (CR)↵

TRANSFER
WRITING
LOADING
START ADDRESS=00011
STOP  ADDRESS=00015
DATA  LENGTH = 0003
ODD

ROM ADDRESS=0008-000A
TRANSFER
WRITING
LOADING
START ADDRESS=00021
STOP  ADDRESS=00025
DATA  LENGTH = 0003
ODD

ROM ADDRESS=0010-0012

TRANSFER
WRITING
LOADING
END OF FILE
```

Writing of odd-numbered addresses

(Note)

If an error occurs during data writing, the following message is displayed, requesting the next processing:

```
WRITING
WRITE ERROR
(R:REPEAT/P:PASS/A:ABORT)=
```

Here, the following three entries are available.

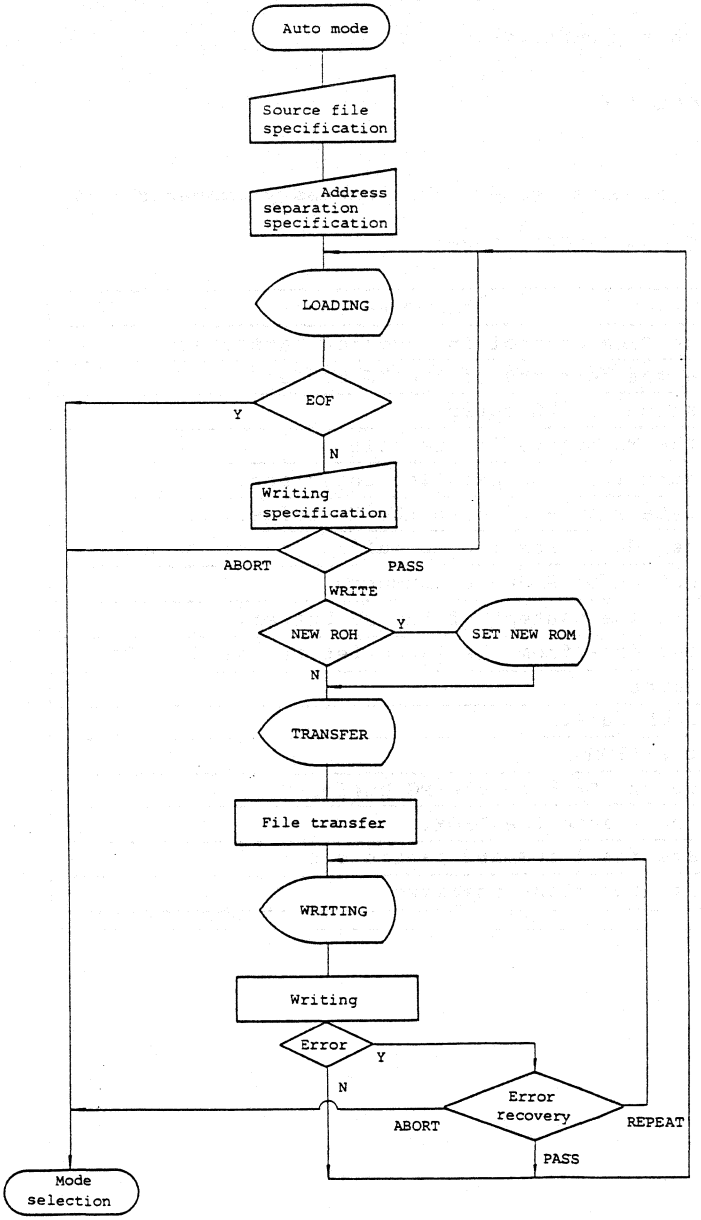
Key entry	Operation
R	Write data again.*
P	Do not write.
A	Return to mode setting.

* When resuming the writing operation, set a cleared PROM and enter 'R'.

(Note)

In the auto mode, data cannot be written into a four-bit PROM (μPB403, μPB406, μPB426, etc.)

Auto mode operation flow chart



2.5 Manual Mode

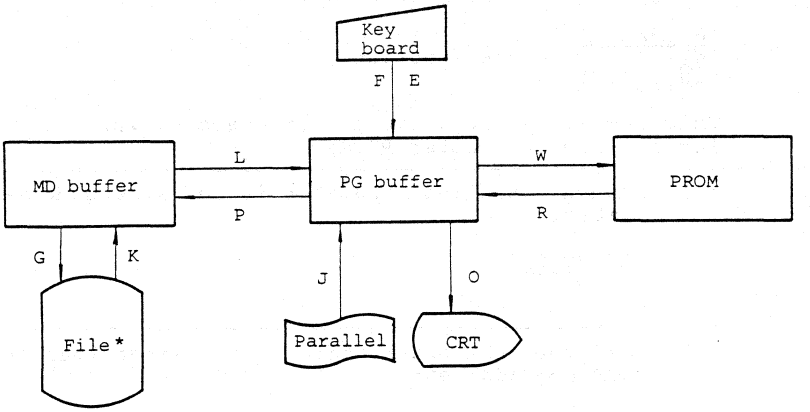
If the manual mode is selected, the following message is displayed with a prompt (*).

```
*** MANUAL MODE ***
*
```

If the prompt appears, select the necessary commands from the following and execute them:

Command	Function
A	Set the PROM control instruction parameters.
E	Change the contents of the PG buffer.
F	Initialize the PG buffer.
G	Save the MD buffer in the file.
J	Read data from the parallel interface.
K	Load data from a file to the MD buffer.
L	Transfer data from the MD buffer to the PG buffer.
M	Combination of K and L commands
O	Indicate the content of the PG buffer.
P	Transfer data from the PG buffer to the MD buffer.
Q	Set a mode.
R	Read PROM data.
S	Select a PROM.
V	Compare the PROM to the PG buffer.
W	Write data into the PROM.
Y	Indicate the A command parameters.
Z	Perform PROM blank checking.

The flow of data with each command is as follows:



* HEX- type or extended HEX-type file

The command format, function, and error control for each of the above commands follow.

2.5.1 A command

This command sets parameters for the PROM control instructions (W and V commands).

Command format	*As,e,r)
	s: PROM start address e: PROM end address r: PG-1000 buffer start address The parameters each should be specified with four or fewer hexadecimal digits. If more than four digits are specified, the last four digits are used.

Function	If all parameters are omitted in the W and V commands, the parameters specified in this command are used. *A0,3FF,0) *W) *W0,3FF,0) } Same function This command is convenient, for example, when copying a large number of master PROMs.
Error control	(1) When a specified parameter is not a hexadecimal number: *A0,3G,0) ? *
	(2) When the PROM start address is greater than the PROM end address: *A10,0,0) ? *
	(3) When the buffer start address is greater than the buffer end address (3FFF): *A0,F,4000) ? *
	(4) When the PROM end address is greater than the currently specified PROM end address: If μPD78P09R is selected: Max.address: 1FFF A0,2000,0) ? *
Note	If a PROM is reselected by using the S command, the parameters of the A command change as follows: s: 0000 e: Max. address of selected PROM r: 0000 If a parameter is omitted, 0 is assumed. If all parameters are omitted, however, they do not always change: *A,,) Same as A0, 0, 0) *A)

2.5.2 E command

This command is used to change the contents of the PG-1000 buffer memory. This is used when patching a program.

Command format	*Er) r: PG-1000 buffer memory address The parameter should be specified with four or fewer hexadecimal digits. If more than four digits are specified, the last four digits are used.
Function	Displays the buffer address, data and a prompt '-'. The data can be changed by the following entry: (1) Two hexadecimal numbers: Change the data. (2) Space: Make no change. (3) Return: Terminate the E command operation. *E10) 0010␣55-00␣55-55␣55-) * ↑ Space key
Error control	(1) When a hexadecimal character is entered for the address: *E3G) ? * (2) When the address exceeds the maximum address (3FFF) of the PG-1000 buffer: *E4000) ? * (3) When an illegal character is entered at the time the data is input: *E0) 0000 55-G ? *
Note	If the E command is activated with the parameter omitted, address '0' is assumed. *E) 0000 55-

2.5.3 F command

This command is used to initialize the PG-1000 buffer memory.

Command format	*Fs,e,d)
	s: Memory start address e: Memory end address d: Initialization data The addresses should be each specified with four or fewer hexadecimal digits. If more than four digits are specified, the last four digits are used. Data should be specified with one or two hexadecimal digits. If more than two digits are specified, the last two digits are used.
Function	Makes all the contents of the buffer memory, from the start address to the end address, initialization data. *F30,40,5) Makes the contents, from address 30 to address 40, '5'.
Error control	(1) When non-hexadecimal characters are specified for the addresses and data: *FG,4G,G) ? * (2) When the memory start address is greater than the memory end address: *F10,0,0) ? * (3) When the memory end address exceeds the last address (3FFF) of the PG-1000 buffer memory: *F0,4000,0) ? * (4) When an incorrect command format is used: *F0,100) ? *

Note	If the parameters are omitted, 0 is assumed. *F,,5) ⟩Same function *F0,0,5)

2.5.4 G command

This command is used to save the data that has been transferred into the MD buffer by a P command as a HEX-type or extended HEX-type file.

Command format	*G filename) *G)
	filename: Specifies a file by a drive name, file name and file type. If the drive name is omitted, the login drive is assumed. If the file type is omitted, 'HEX' is assumed.
Function	Saves the contents of the MD buffer as a file. The file type differs depending on the specification as follows:

File type	Type in which data is saved
HEX	HEX type
H86	Extended HEX type

If the specified file is found, the system asks whether the file be deleted:

```
*GB:TEST.HEX)
B:  TEST HEX ALREADY EXITS DELETE(Y/N)?
```

When the data has been saved in the file, the system asks whether the file is to be closed. If the writing operation is to be continued in the same file, it must not be closed.

When executing the G command with the existing file remaining open, no file name need be specified:

```
*G TEST.HEX)
CLOSE(Y/N)?N)
*P0, F)
*G) ←—— Continue writing to TEST. HEX
CLOSE(Y/N)?
```

Error control

(1) When no file name is specified:

```
*G↵
?
*
```

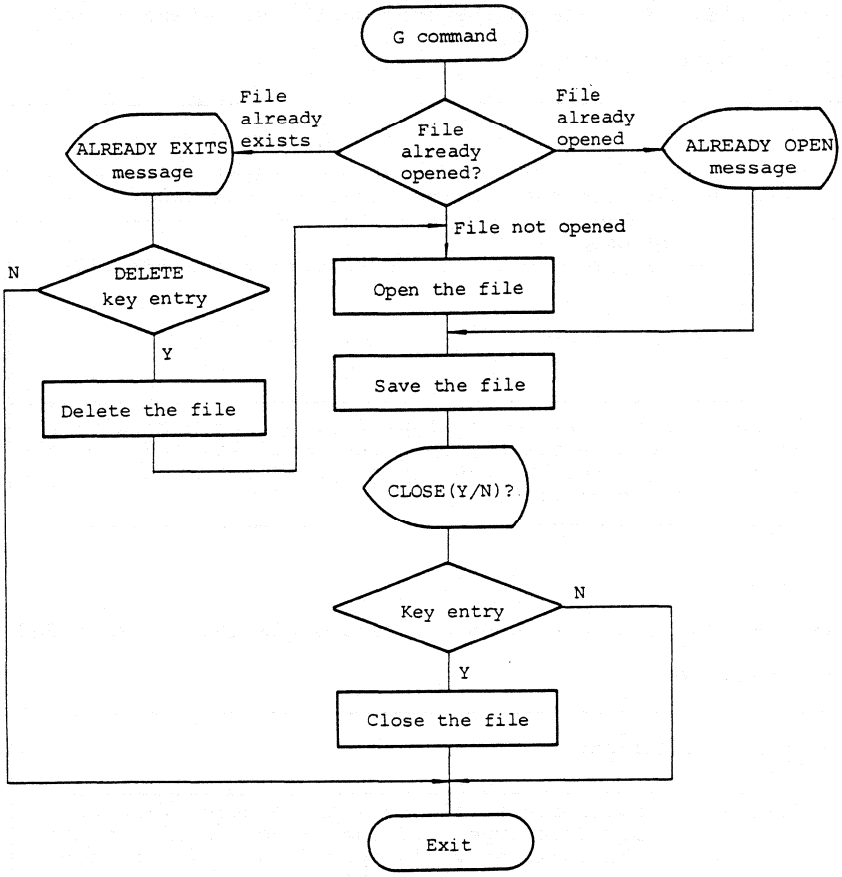
(2) When the G command is executed without execution of the P command or after execution of the L command:

```
*** MANUAL MODE ***
*GTEST.HEX↵
NOT EXECUTE P-COMMAND
*
```

(3) When a file name is specified with the existing file remaining open:

```
*GB:TEST.HEX↵
CLOSE(Y/N)?N↵
*GWORK.HEX↵
B:TEST.HEX IS ALREADY OPEN, TYPE (G ,CR)
*G↵
GB:TEST.HEX
CLOSE(Y/N)?
```

G command processing flow



2.5.5 J command

This command is used to take a HEX file from a device connected to the parallel interface on the rear panel and expand it into the PG-1000 buffer memory.

Command format	*J↵
Function	Reads a HEX file through the parallel interface.
Error control	(1) Parity check error When a parity error is found during the HEX file read: *J↵ ? *
Note	If no external device is connected, a hangup condition occurs. In such cases, press the RESET key and then the SERIAL key on the PG-1000 key panel to return the system to a normal condition.

2.5.6 K command

This command is used to open and read a HEX-type or extended HEX-type file.

Command format	*K filename,o,c↵ *K↵
	filename Specifies a file by a drive name, file name and file type. If the drive name is omitted, the login drive is specified. If the file type is omitted, 'HEX' is assumed. o: Offset address When reading the file, the data in addresses less than this address is read. This is specified with four or fewer hexadecimal digits. If this is omitted, 0 is assumed, that is all the data is loaded.

<table>
<tr><td></td><td>C: Specifies separation of even/odd addresses
There are three key words for address separation specification:

N: No separation
E: Load in even addresses only.
O: Load in odd addresses only.

If this is omitted, 'N' is used.</td></tr>
<tr><td>Function</td><td>Loads the file and expands it in the MD buffer. The type of the file to be loaded differs depending on the file type specification as follows:

<table><tr><th>File type</th><th>Type in which the file is loaded</th></tr><tr><td>HEX</td><td>HEX type</td></tr><tr><td>H86</td><td>Extended HEX type</td></tr><tr><td>Others</td><td>Binary type</td></tr></table>
The loading operation continues until any one of the following conditions is met:

(1) The size of the PROM is exceeded.
(2) Load addresses become discontinued.
(3) The file reaches the end.

When the loading has been completed, the start address, stop address, data length and PROM address for the loaded data are displayed:

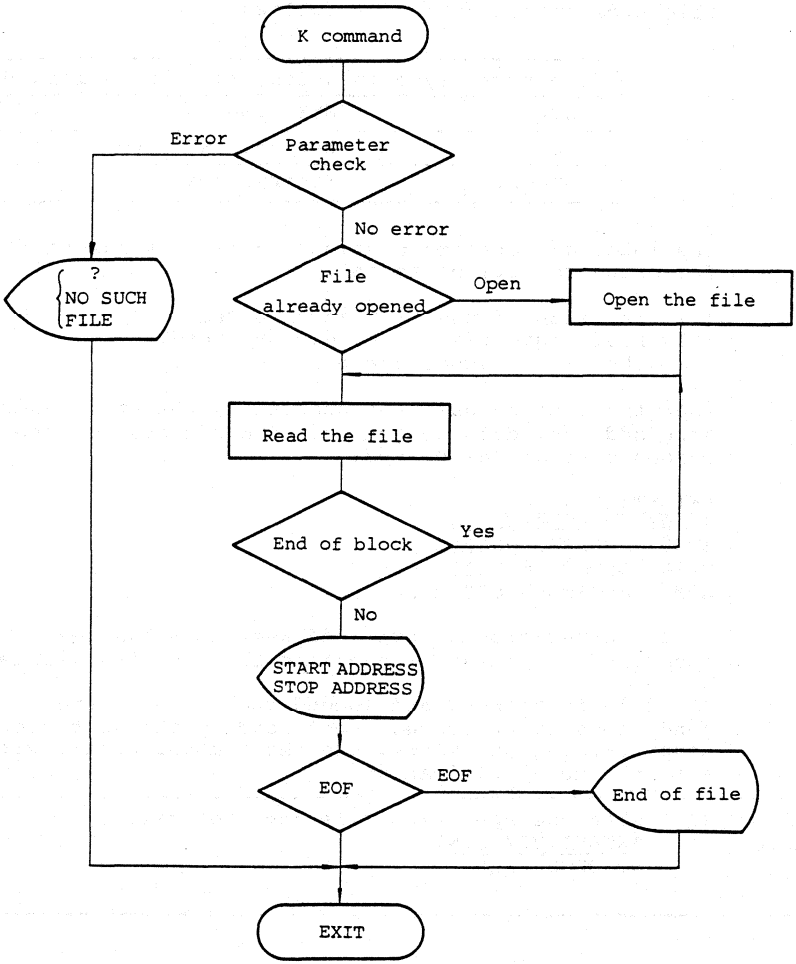
*KTEST.HEX,0,N)
START ADDRESS=00000
STOP ADDRESS=00006
DATA LENGTH = 0007
ROM ADDRESS=0000-0006

If the separation of even/odd addresses has been specified, a message indicating this is also displayed.

The file is blocked and loaded according to the conditions shown above. When loading the next block after reading one block, use the command format without a file name specification.</td></tr>
<tr><td>Error control</td><td>(1) When the specified file does not exist:
*KTEST.HEX,0,N)
NO SUCH FILE
*</td></tr>
</table>

Error control	(2) When there is an error in the parameter specification: *KTEST.HEX,G,A) ? *

K command processing flow



2.5.7 L command

The K command is used to transfer the data, that has been loaded into the MD buffer by the K command, into the PG-1000 buffer.

Command format	*L↵
Function	Transfers the data that has been loaded into the MD buffer by the K command into the PG-1000 buffer. A simple protocol is used for data transfer. The transfer operation can be interrupted by pressing the 'ESC' key. *L↵ BREAK ← Press the ESC key. *
Error control	(1) When a parameter is added: *L123↵ ? * (2) When no K command has been executed or when the L command is executed after execution of a P command: *** MANUAL MODE *** *L↵ FILE NOT READ *

2.5.8 M command

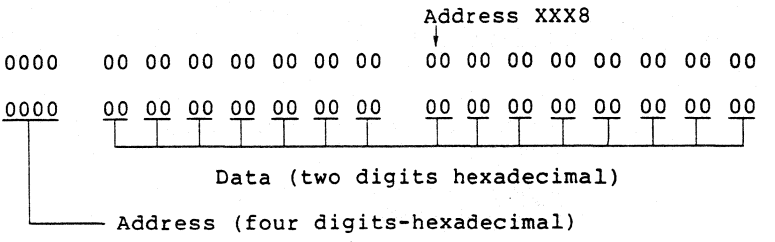
This is a combination of the K command and L command.

Command format	*M filename,δ,C↵ *M↵
Function	Executes the L command after execution of the K command. For details, refer to the K and L commands.

2.5.9 O command

This command is used to display the contents of the PG-1000 buffer memory in hexadecimal numbers.

Command format	*Os,e↵
	s: Buffer memory start address e: Buffer memory end address The parameter should each be specified with four or fewer hexadecimal digits. If more than four digits are specified, the last four digits are used.
Function	Displays the contents of the buffer memory. If the buffer memory address is omitted, only the contents specified by the start address are displayed. If both start and end addresses are omitted, only address '0' is displayed: *O100↵ 0100 55 *O↵ 0000 55 *
Error control	(1) When the start address is greater than the end address: *O100,0↵ ? * (2) When non-hexadecimal parameters are specified: *O0G,FG↵ ? * (3) When the end address exceeds the last address (3FFF) of the PG-1000 buffer memory: *O0, 4000↵ ? *

Note	Data is displayed in the following format: 

2.5.10 P command

The P command is used to transfer the contents of the PG-1000 buffer to the MD buffer.

Command format	*Ps,e) s: Buffer memory start address e: Buffer memory end address The parameters should each be specified with four or fewer hexadecimal digits. If more than four digits are specified, the last four digits are taken.
Function	Transfers the contents of the PG-1000 buffer memory to the MD buffer memory. A simple protocol is used for data transfer. The transfer operation can be interrupted by pressing the 'ESC' key. *P0,3FFF) Press the ESC key. BREAK * If the end address is omitted, only one byte specified by the start address is displayed. If both start and end addresses are omitted, only address '0' is transferred. *P100) *P) *
Error control	(1) When non-hexadecimal parameters are specified: *P10G,20G) ? *

Error control	(2) When the start address is greater than the end address: *P200,0) ? * (3) When the check sum in the simple protocol contains an error: *P0,FF) CHECK SUM ERROR *

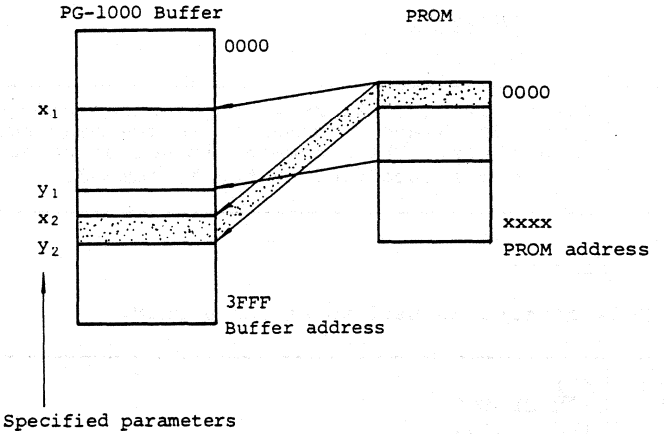
2.5.11 Q command

This command returns control to the mode setting.

Command format	*Q)
Function	Returns control to the setting of the auto mode or manual mode. *Q) MODE(A/M/S)= As explained previously, the following three key entries are available: A: Move to the auto mode. M: Move to the manual mode. S: Move to PROM selection.
Error control	(1) When a key other than 'A', 'M' or 'S' is pressed: *Q) MODE(A/M/S)=0) MODE(A/M/S)=

2.5.12 R command

This command is used to read out the contents of the PROM into the PG-1000 buffer.

Command format	*Rx,y)
	x: Buffer memory start address y: Buffer memory end address The parameters should each be specified with four or fewer hexadecimal digits. If more than four digits are specified, the last four digits are used.
Function	Reads the contents of address '0' of the PROM into the PG-1000 buffer memory from the start address to the end address.  If the end address is omitted, all data from the start address is read into the PG-1000 buffer. If both start and end addresses are omitted, all the PROM data from the start address (0) is read into the PG-1000 buffer.
Error control	(1) When non-hexadecimal parameters are specified: *R10G,GFF) ? *

Error control	(2) When the start address is greater than the end address: *R100,0) ? * (3) When the end address is greater than the maximum address of the PG-1000 buffer: *R3FF0,4000) ? * (4) When the size of the area specified by the start and end addresses exceeds the PROM size: If μPD8748 is selected: Buffer size: 400 *R0,400) ? *
Note	The R command differs from the other PROM control commands (W and V commands) in the meanings of the parameters. The R command is not affected by the parameters specified by the A command.

2.5.13 S command

This command is used to select a PROM.

Command format	*S) ROM SELECT (PG1000):) *
Function	Selects a PROM. With the execution of the S command, the following message is displayed: *S) ROM SELECT (PG1000): Then, select the desired PROM according to the following procedure: ① Press the RESET key on the PG-1000 key panel.

Function	② Select the desired PROM by making reference to the manual for each personal module. ③ Press the SERIAL key on the PG-1000 key panel. Confirm that the SERIAL LED is on. With the above procedure, the desired PROM is selected. Then, press the RETURN key on the MD-080/086 keyboard.
Note	The method of selecting a PROM differs slightly for each personal module. For details, refer to the manual for each personal module. If the RETURN key is pressed without the SERIAL key of the PG-1000 being pressed, a hangup condition arises. The SERIAL mode must be set before pressing the RETURN key. If the RETURN key is pressed prior to the SERIAL key by mistake, press the SERIAL key to escape from the hangup condition and then execute the S command again. If a new PROM is selected by executing the S command, the parameters of the A command change as follows: s: 0000 e: Maximum address of selected PROM r: 0000

2.5.14 V command

This command is used to compare the contents of the PG-1000 with the content of the PROM.

Command format	*V[s,e,r] *V
	s: PROM start address e: PROM end address r: PG buffer start address The parameters should each be specified with four or fewer hexadecimal digits. If more than four digits are specified, the last four digits are used. If all the parameters are omitted, the parameters specified by the A command are used.
Function	Compares the data written from the start address to end address of the PROM with the data from the start address of the PG-1000 buffer memory.

Function

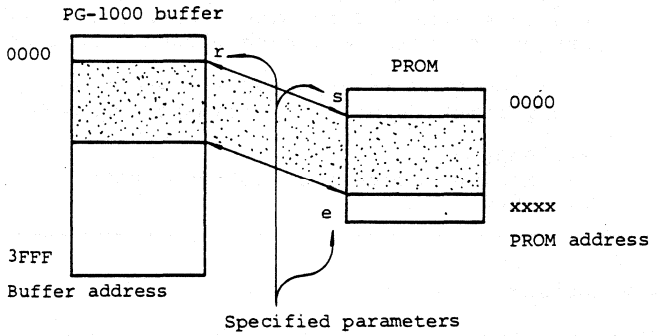
If different data is written, the system displays '?' together with the PROM address, PROM data and buffer memory data on the PG-1000 front panel.

If a parameter is omitted, it is assumed that 0 has been specified.

```
*V,,100↓
*V0,0,100↓      > Same function
```

If, however, all the parameters are omitted, the values specified by the A command are used:

```
*A0,F,1000↓
*V↓             ] Same function
*V0,F,1000↓
```



Error control

(1) When non-hexadecimal parameters are specified:

```
*VG,FG,10G↓
?
*
```

(2) When the start address is larger than the end address:

```
*V101,100,0↓
?
*
```

(3) When the buffer address exceeds the maximum address (3FFF) of the PG-1000 buffer:

```
*V0,F,4000↓
?
```

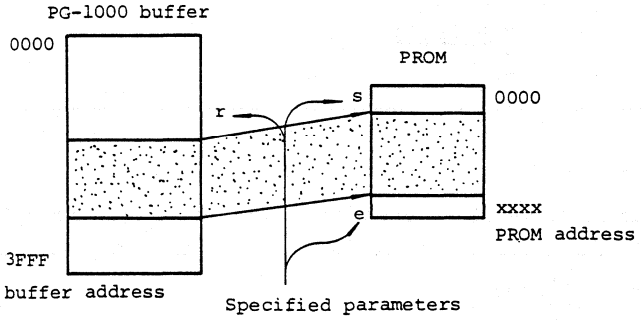
Error control	(4) When different data is found in the PG-1000 buffer and PROM: *V0,FF,0) ? *

2.5.15 W command

This command is used to write data into the PROM.

Command format	*WS,e,r) *W)
	s: PROM start address e: PROM end address r: PG buffer start address The parameters should each be specified with four or fewer hexadecimal digits. If more than four digits are specified, the last four digits are used. If all the parameters are omitted, the parameters specified by the A command are used.
Function	Writes the data from the PG-1000 buffer memory start address into the area specified by the PROM start address and end address. If a parameter is omitted, it is assumed that 0 has been specified: *W,,100) *W0,0,100) > Same function If, however, all the parameters are omitted, the values specified by the A command are used: *A0,F,1000) *W) *W0,F,1000) > Same function

Function



Error control

(1) When non-hexadecimal parameters are specified:

```
*WG,10G,GF↵
?
*
```

(2) When the start address is larger than the end address:

```
*W100,0,0↵
?
*
```

(3) When the buffer address exceeds the maximum address (3FFF) of the PG-1000:

```
*W0,F,4000↵
?
*
```

(4) When an error has occurred during the write operation:

```
*W↵
?
*
```

Note

W command operations have the following three stages:

① Blank check
② Write
③ Compare

Note	In the blank check above, the specified memory area is examined to determine whether the contents have been erased. No writing can be made into a PROM unless they have been erased.

2.5.16 Y command

This command is used to display the parameters set by the A command

Command format	*Y)
Function	Displays the values set by the A command. The display format is as follows: *Y) 0000 0000 0000 Buffer start address PROM end address PROM start address

2.5.17 Z command

The Z command checks the PROM to determine whether it has been erased or not.

Command format	*Z)
Function	Checks all the data of the PROM to determine whether it has been erased. If there is any unerased data, '?' is displayed. *Z) ? *
Note	If there is already written data, its address and contents are displayed on the PG-1000 front panel.

2.6 System Control Character

There are the five system control characters as follows:

Key entry	Function
CTRL-S	Stops the display operation.
CTRL-Q	Clears CTRL-S (stopping of display).
CTRL-A	Resets the PG-1000 to stop the processing.
Space	Stops the display operation.
CTRL-C	Aborts the control program ('PGC').

SECTION 3 CONSOLE OPERATION MODE

This chapter explains the methods of connecting the PG-1000 to a console and thereby controlling the PG-1000 from the console.

3.1 General

In this mode, the PG-1000 can be controlled from the console connected to the serial interface of the PG-1000.

The mode provides the basic operations including copying a PROM (reading); blank check (determining whether the PROM has been erased or not); programming (writing); and verification (comparison). Unlike the host mode, however, the console mode does not provide extended functions.

3.2 Setting

Check the interface of the console to be connected, prepare the PG-1000 serial interface connection cable, and connect the two units.

For details, refer to PART I, Section 4.1 Serial Interface.

3.3 Starting

When the equipment has been set, apply power to the PG-1000. Then, push the SERIAL key on the PG-1000 key panel. The SERIAL LED on the front panel will then turn on.

(Note)

When the SERIAL key is pressed, if power is not applied to the console, 'E-11' is displayed on the address display. If 'E-11' is displayed even with power applied to the console, check the cable connection. If the cable is properly connected, the equipment may be out of order. Turn off the power immediately and send the equipment for repair.

When the cable connection has been completed, a prompt(*) is displayed on the console. If no prompt appears, check the baud rate for consistency, the cable for proper connection and the SERIAL LED on the PG-1000 front panel for lighting. If these are all normal, the PG-1000 may have a fault. In such cases, please send it for repair.

3.4 Commands

When the PG-1000 starts, the system displays a prompt(*) waiting for command entry.

Command	Function
A	Sets the parameters of the PROM control commands.
E	Changes the contents of the PG buffer.
F	Initializes the PG buffer.
I	Moves to the intelligent mode*.
J	Reads data from the parallel interface.
O	Displays the contents of the PG buffer.
R	Reads the data out of the PROM.
S	Selects a PROM.
T	Moves to the terminal mode**.
V	Compares the PROM to the PG buffer.
W	Writes data into the PROM.

Command	Function
Y	Displays the A command parameters.
Z	Performs PROM blank checking.

* The intelligent mode echoes user key entries back to the console. This is used in the console operation mode.

** The terminal mode does not echo the user key entries. This is used in the host connection mode. This is the default mode at the time the power is turned on.

The above commands except 'I' and 'T' are the same as those available in the manual mode of the host mode.

This section, therefore, explains only the 'I' and 'T' commands. For other commands, refer to Section 2.5 Manual Mode.

(Note)
In the host mode, it is possible to delete a key entry by using the 'DEL' key. But the console mode does not support this function. Also, if an incorrect parameter is read, an error message is displayed immediately:

```
*A0,G
?
*
```

3.4.1 I command

This command is used to move to the intelligent mode.

Command format	*I↓
Function	Echoes the user's key entries:

Key entry	Display	Mode
ABCD↓	*	Terminal mode
I↓	*	↓
ABCD	*ABCD	Intelligent mode

3.4.2 T command

This command is used to move to the terminal mode.

Command format	*T↓
Function	Does not echo the user's key entries:

Key entry	Display	Mode
ABCD↓	*ABCD	Intelligent mode
I↓	*T↓	↓
ABCD	*	Terminal mode

3.5 System Control Characters

There are four system control characters as follows:

Key entry	Function
CTRL-S	Temporarily stops the display operation.
CTRL-Q	Clears CTRL-S (temporary stopping).
CTRL-C	Stops all processing.
Space	Stops the display operation.

SECTION 4 STAND-ALONE MODE

If the PG-1000 power is turned on after a personal module has been connected, the panel displays random data for a few seconds and then displays the currently used PROM name (when the personal module is the PG-1001 or 1002) or the writing method (when the personal module is the PG-1003). At this time, the PG-1000 can be used by itself without any device being connected. This state is called the stand-alone mode.

The commands used in the stand-alone mode can be broadly divided into two categories, that is, control commands used for data writing, etc. and panel commands for editing data in the buffer memory.

When the PG-1000 is used in the stand-alone mode, if the command being executed can be forcibly terminated by pressing the reset key. This puts the system in the command wait state. If an error arises during the reset operation, the following indication is given:

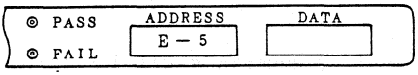


Fig. 4.1

If an error indication is given, perform the following:

① Remove the PROM from the module, turn off the PG-1000 power and check the personal module connections.

② Then, connect the personal unit to the PG-1000 properly and turn the power switch on.

③ Press the reset key. If an error is indicated again, the PG-1000 or the connected personal unit may be faulty. In such cases, please contact your dealer.

This chapter explains the commands for the PG-1000 used in the stand-alone mode. Before using these commands, it is necessary to connect your personal module and select a PROM or a writing method. For the methods of connecting a personal module and selecting a PROM, refer to the manual for each personal module.

4.1 Control Commands

Control commands are used to control the PROM directly, e.g. writing data into the PROM and comparing its data with the memory. The keys related to the control commands are those enclosed by a dotted line in Fig. 4.2.

The following control commands can be used:

Table 4.1

Command name	Key	Function
Copy	COPY	Reads data.
Blank check	BLANK	Checks the write status.
Program	PRG	Writes data.
Verify	VER	Compares data.
Continuous	CONT	Writes data, compares it and checks it status.

Key positions

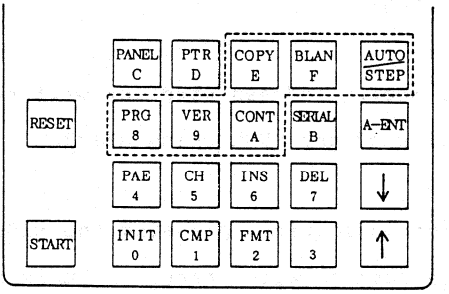


Fig. 4.2

A control command can be executed by pressing the corresponding key shown in Fig. 4.2 and then pressing the START key.

When the START key is pressed, program address is checked prior to the command being executed. If an error is detected, the following error indication is given:

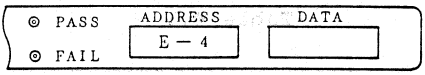


Fig. 4.3

If an error is indicated, check the following:

1. Does the last address of the data written exceed the maximum address of the PROM?

2. Is the start address larger than the end address for data writing?

3. Does the memory address used for writing exceed the memory size of the PROM being used?

After an error indication, pressing the RESET key puts the system in the command wait state. In this case, check the parameters by making reference to Section 4.2.1 PAE Mode.

4.1.1 Switching between auto and step modes

For the Blank Check, Program, Verify and Continuous commands, the auto and step modes can be switched only in the command wait state. The auto mode means executing commands continuously by pressing the START key. The step mode means executing commands for a single address each time the START key is pressed.

Operation:

① Prior to switching the auto and step modes, check both the AUTO and STEP lamps. The illuminated lamp shows the current mode.

② Press the AUTO and STEP key. The other lamp then turns on and the mode has been changed.

4.1.2 Indication of normal termination

If Copy, Program, Verify or Continuous command operations have terminated normally, the following indication is given:

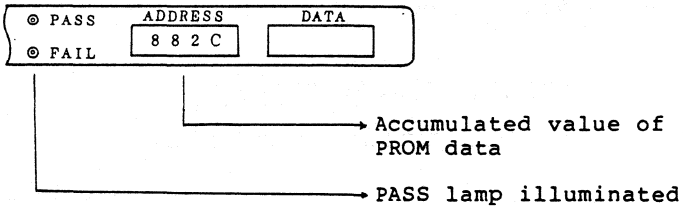


Fig. 4.4

For the Blank Check command, the following indication is given:

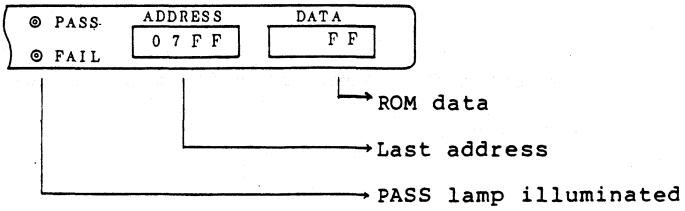


Fig. 4.5

For the indication of abnormal termination, refer to the explanation of each command.

4.1.3 Copy command

This command is used to read the data written in the PROM into the PG-1000 buffer memory.

To reduce misreading of data into the memory, the Copy command compares the data read into the memory with the PROM data after the reading operation. If an error is found, the FAIL lamp on the panel lights and the memory address in which the error was found and the data in the address are displayed as shown in Fig. 4.6

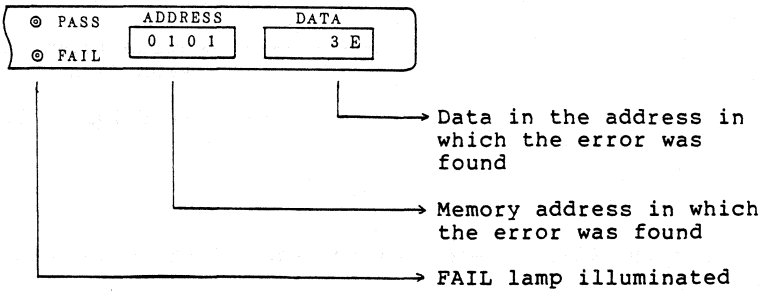


Fig. 4.6

Operations:

① Press the COPY key. This illuminates the COPY lamp on the panel.

② Press the START key. This causes the Copy command to be executed. If the PASS lamp lights after the execution, it shows that the operation has terminated normally. If the FAIL lamp lights during the execution, it means that an error has been found. In this case, place the system in the command wait state by pressing the RESET key. No other key entries are accepted.

4.1.4 Blank check command

This command is used to determine whether the PROM has been erased. If data is already written in it, the following indication is given:

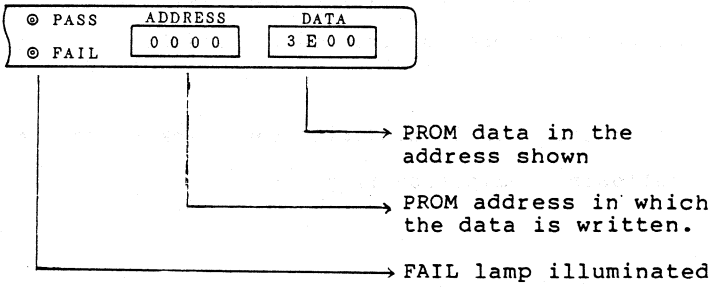


Fig. 4.7

Operation:

① Press the BLANK key. This causes the BLANK lamp to light.

② Press the START key. With this, the Blank Check command is executed. If the PASS lamp lights after the execution, it means that the PROM has been erased. If the FAIL lamp lights during the operation as shown in Fig. 4.7, it means that data has already been written in the address shown.

In this case, place the system in the command wait sate by pressing the RESET key. Any other key entry cannot be accepted.

4.1.5 Program command

This is for writing the data stored in the buffer memory into the PROM. When the writing has been completed, the command performs a verification check.

If an error is found during the checking operation, the following indication is given:

Verification check error

PROM data in the address in which the error was found

Memory data in the address in which the error was found

Memory address in which the error was found

FAIL lamp illuminated

Vpp error

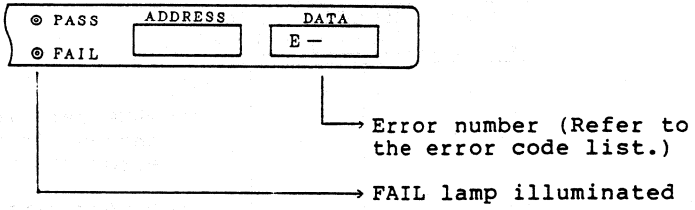


Fig. 4.8

Operation:

① Press the PRG key. This causes the PROGRAM lamp on the panel to light.

② Press the START key. With this, the PROGRAM command is executed. If the PASS lamp lights after the command has been executed, it means that the operation has been terminated normally.

If the FAIL lamp lights during the execution, it means that the error shown in Fig. 4.7 has been found. In this case, press the RESET key to place the system in the command wait state. No other key entries are accepted.

4.1.6 Verify command

This command compares the contents of the written PROM with the contents of the memory to determine whether the writing has been performed normally. If an error is found during the execution of the command, the following indication is given:

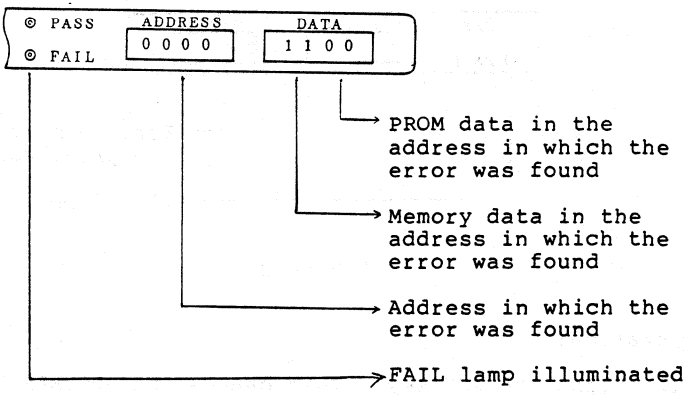


Fig. 4.9

Operation:

① Press the VER key. This causes the VERIFY lamp on the panel to light.

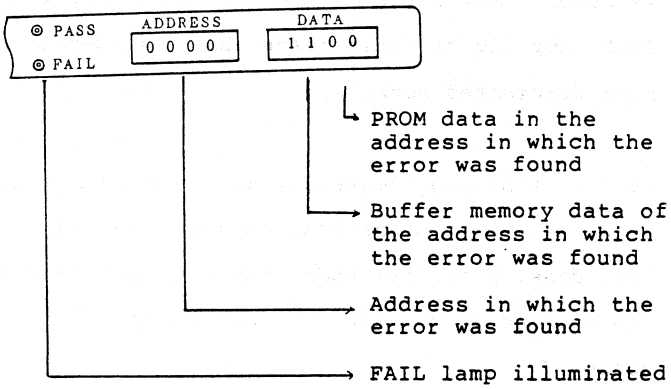
② Press the START key. With this the Verify command is executed. After the command has been executed, if the PASS lamp lights, this means that the operation has been terminated normally.

If the FAIL lamp lights during the execution as shown in Fig. 4.9, it means that an error has been found. In this case, press the RESET key to place the system in the command wait state. No other key entries are accepted.

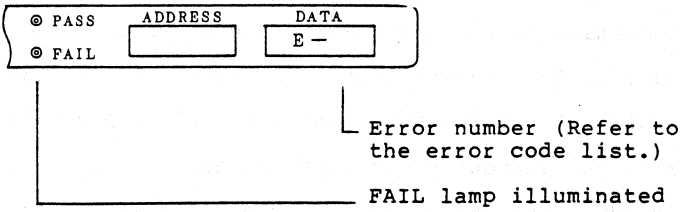
4.1.7 Continuous command

This command performs the Blank Check, Program and Verify operations in succession in this order. During the execution of the command, both the CONT lamp and the lamp showing the currently executed command are lit. If an error is found during the execution, the following indication is given:

Program or verify error



Program Vpp error



Blank check error

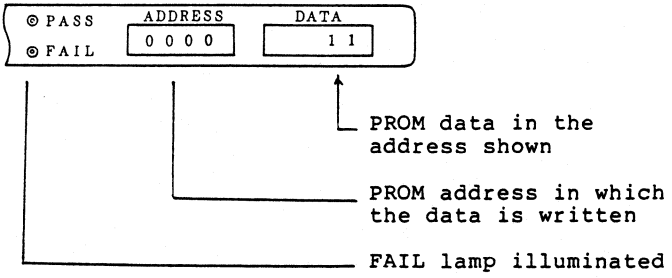


Fig. 4.10

Operation:

① Press the CONT key. This causes the CONT lamp on the panel to light.

② Press the START key. With this, the Continuous command is executed. If the PASS lamp lights after the execution, it means that the operation has been terminated normally.

If the FAIL lamp lights during the execution, it means that the error shown above has been found. In this case, press the RESET key to place the system in the command wait state. No other key entries are accepted.

4.2 Panel Command

Panel commands are used to edit the data in the PG-1000 buffer memory, set the range of addresses, etc. The keys related to the panel commands are those enclosed in a dotted line shown in Fig. 4.11.

The panel commands are as follows:

Table 4.2

Mode name	Key	Function
PAE mode	PAE	Sets the range of PROM operation, and the start and end addresses of the buffer memory.
FMT mode	FMT	Displays data format addresses.
CH mode	CH	Prepares new data and modifies the old data.
INS mode	INS	Inserts data.

Table 4.2 - continued

Mode name	Key	Function
DEL mode	DEL	Deletes data.
INIT mode	INIT	Initializes the data in the buffer memory.
CMP mode	CMP	Reverses the data in the buffer memory.

Key positions

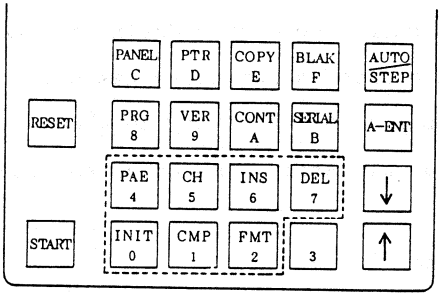


Fig. 4.11

Points to note for panel command execution:

1. When entering each panel command mode, press the corresponding key after ascertaining that the PANEL lamp is lit.

2. When entering an address from the keyboard, use four digits. When entering byte data, use two digits. If more than two digits are entered, the last two digits are used.

3. The entered value is shifted to the left by one digit each time another value is entered. If an incorrect value is entered, enter the correct data from the beginning to change this value to the correct value.

4. A control command key can be entered in the panel mode command wait state. In this case, the panel mode is automatically cleared.

4.2.1 PAE mode

This is used to specify or check the range of operation for data writing or reading.

The addresses that can be specified by this command are the start address (MS) and end address (MP) of the PROM, and the start address (MR) of the buffer memory.

When applying power to the PG-1000, the MS and MR are each set to address 0. For MP, the setting differs with each personal module. Please refer to the manual for each personal module.

Please note that when a PROM has been selected, a new MP that conforms to the PROM is set. For the newly established MP, refer to the manual for each personal module.

When reading data out of the PROM, the MR specifies the address of the buffer memory from which the data is read.

Operation:

1. When checking the established value:

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the PAE key. At this time, the panel display gives the following indication:

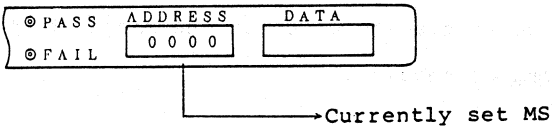


Fig. 4.12

③ Press the ↑ key or the A-ENT key. At this time, the panel display gives the following indication:

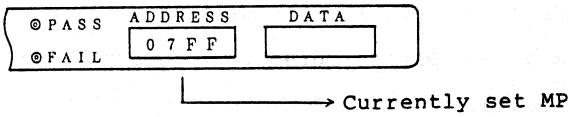


Fig. 4.13

④ Press the ↑ key or the A-ENT key. At this time, the panel display gives the following indication:

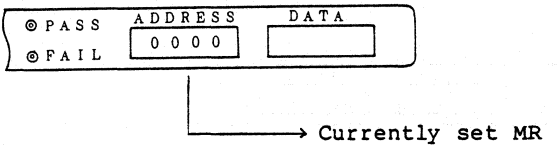


Fig. 4.14

⑤ Press the ↑ key or the A-ENT key. At this time, the PAE command terminates with the indication shown in Fig. 4.15.

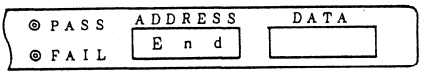


Fig. 4.15

The system is now in the PAE mode and can accept any panel command entry including the PAE command and any control command entry.

2. When entering the value to be set:

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the PAE key. At this time, the panel display indicates the currently established MS. (Refer to operation ② for checking the established value.)

③ Key in the MS to be established again. At this time, the panel display gives the following indication:

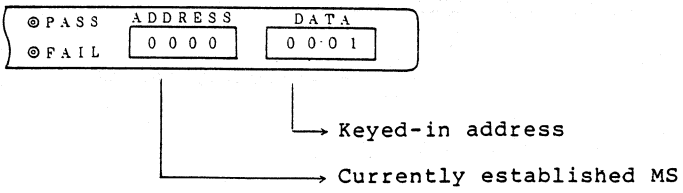


Fig. 4.16

④ Press the A-ENT key to set the keyed-in address as the MS. At this time, the panel display gives the following indication:

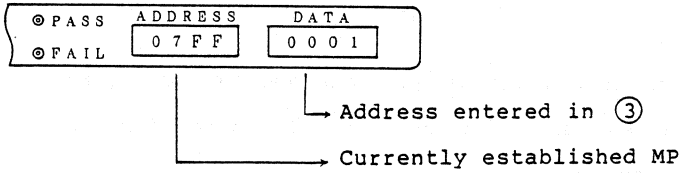


Fig. 4.17

(Note)

If the ↑ key is pressed without the A-ENT key being pressed, the keyed-in address is not accepted as the MS and the system waits for the next operation.

⑤ Key in the MP to be established again. At this time, the panel display gives the following indication:

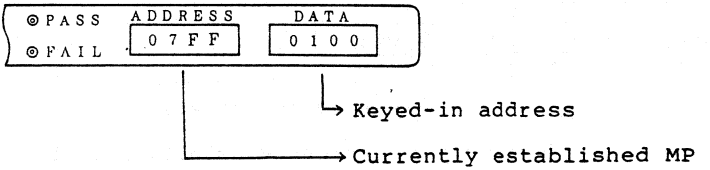


Fig. 4.18

⑥ Press the A-ENT key to set the keyed-in address as the MP. At this time, the panel display gives the following indication:

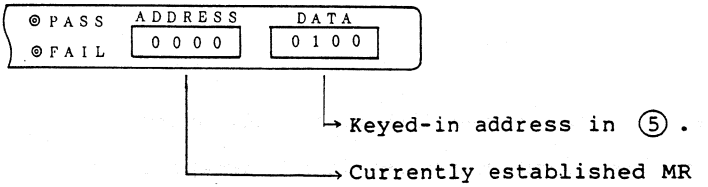


Fig. 4.19

(Note)
If the ↑ key is pressed without the A-ENT key being pressed, the keyed-in address is not accepted and the system waits for the next operation.

⑦ Key in the MR to be established again. At this time, the panel display gives the following indication:

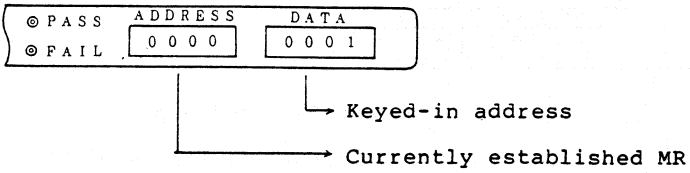


Fig. 4.20

⑧ Press the A-ENT key to set the keyed-in address as the MR. At this time, the panel display indicates "END". (Refer to operation ⑤ for checking the established value.)

(Note)
If the ↑ key is pressed without the key being pressed, the keyed-in address is not accepted and the system waits for the next operation.

It is not possible to return to the previous setting during the MS, MP or MR setting operation. If, for example, it is desired to reestablish the MP during the PR setting operation, it is necessary to execute the PAE command again.

Sample operation

Set addresses 0001, 001F and 0001 for the MS, MP and MR, respectively, and check them after they have been set. (The initial settings of the MS, MP and MR are assumed to be 0, 1FFF and 0, respectively.)

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the PAE key. With this, the panel display gives the following indication:

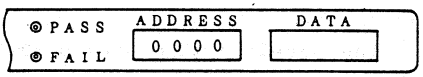


Fig. 4.21

③ Key in address 0001. The panel display gives the following indication:

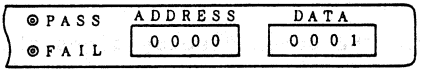


Fig. 4.22

④ Press the A-ENT key. The panel display gives the following indication:

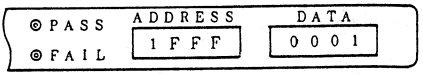


Fig. 4.23

⑤ Key in address 001F. The panel display gives the following indication:

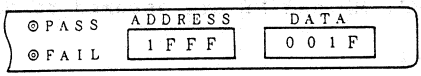


Fig. 4.24

⑥ Press the A-ENT key. The panel display gives the following indication:

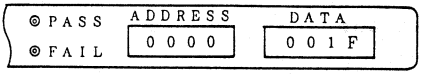


Fig. 4.25

⑦ Key in address 0001. The panel display gives the following indication:

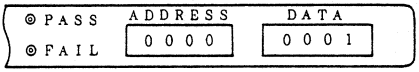


Fig. 4.26

⑧ Press the A-ENT key. The panel display gives the following indication, showing the termination of the PAE command.

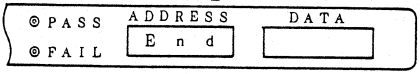


Fig. 4.27

⑨ Check the newly established addresses.

Press the PAE key. The panel display gives the following indication, making it possible to ascertain that the MS has been changed:

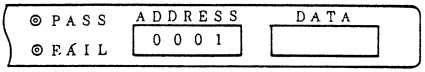


Fig. 4.28

⑩ Press the ↑ key. The panel display gives the following indication, making it possible to ascertain that the MP has been changed:

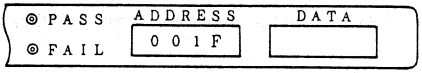


Fig. 4.29

⑪ Press the ↑ key. The panel display gives the following indication, making it possible to ascertain that the MR has been changed:

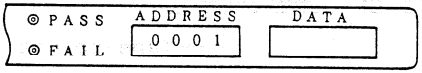


Fig. 4.30

⑫ Press the ↑ key. The panel display gives the following indication, showing the termination of the PAE command:

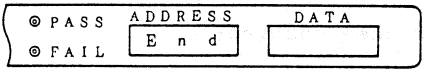


Fig. 4.31

4.2.2 FMT mode

This mode is used to specify a tape format when reading data from a paper type. However, since only the Intel-type format is available with the PG-1000, this command is currently meaningless.

Operation:

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the FMT key. The panel display gives the following indication:

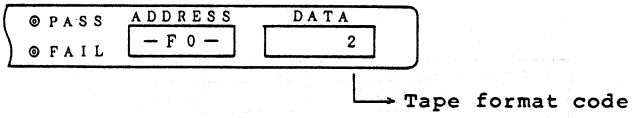


Fig. 4.32

Tape format code "2" shown here is the code number of the Intel HEX format. If you enter a number other than "2" and go to operation ③ below, the error message shown below will be given.

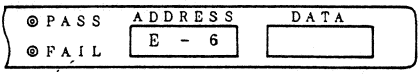


Fig. 4.33

③ Press the ↑ key. The FMT command is then terminated with the indication below:

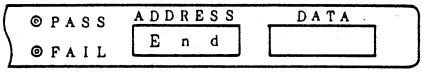


Fig. 4.34

4.2.3 CH mode

This mode is used to change or check the buffer memory data.

Operation:

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the CH key. The panel display gives the following indication:

Fig. 4.35

③ Key in the address to be changed or checked. The panel display gives the following indication:

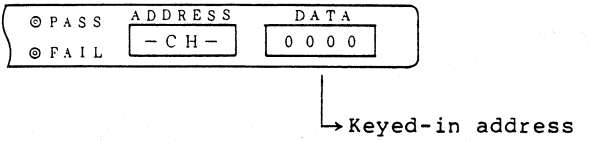


Fig. 4.36

④ Press the A-ENT key to set the keyed-in address. The panel display gives the following indication:

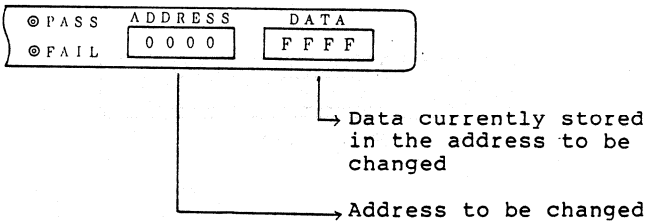


Fig. 4.37

(Note)

Data is indicated in four digits. This is because two data items in the address shown are indicated in bytes.

⑤ Key in the new data. The panel display gives the following indication:

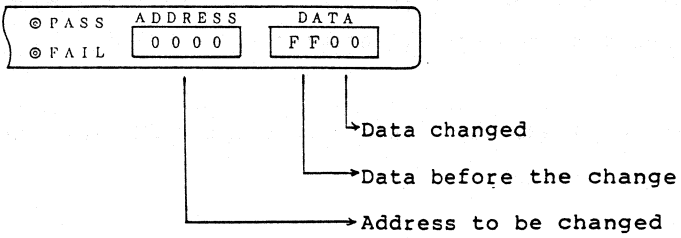


Fig. 4.38

⑥ Press the ↑ key to set the newly entered data. The panel display gives the following indication:

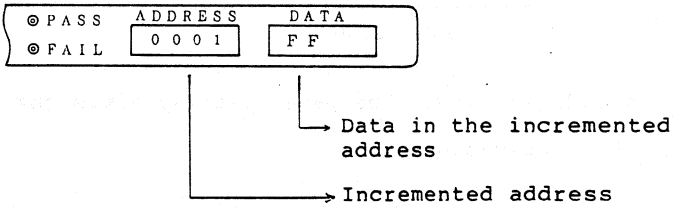


Fig. 4.39

If it is desired to change data in succession, repeat operations ⑤ and ⑥. To terminate the CH command, press the RESET key. At this time, the system is placed in the command wait state.

If it is desired to check the data, perform the operations except ⑤. If the key is pressed in operation ⑥, the decremented address is displayed. It is possible to combine the data change and checking operation.

Sample operation

Change the data in addresses 100, 101 and 102 to 3EH, 00H and AFH, respectively, and check the data after the change. (The initial settings are all assumed to be FFH.)

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the CH key. The panel display gives the following indication:

Fig. 4.40

③ Key in address 0100. The panel display gives the following indication:

Fig. 4.41

④ Press the A-ENT key. The panel display indicates the following:

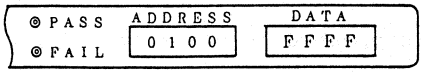


Fig. 4.42

⑤ Key in data 3E. The panel display indicates the following:

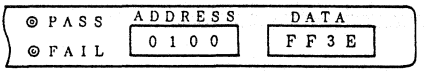


Fig. 4.43

⑥ Press the ↑ key. The panel display gives the following indication:

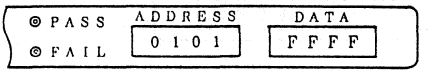


Fig. 4.44

⑦ Key in data 00. The panel display gives the following indication:

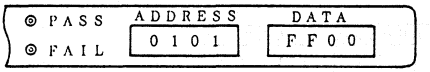


Fig. 4.45

⑧ Press the ↑ key. The panel display gives the following indication:

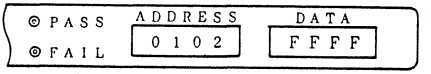


Fig. 4.46

⑨ Key in data AF. The panel display gives the following indication:

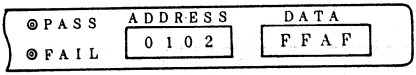


Fig. 4.47

⑩ Press the ↑ key. The panel display gives the following indication:

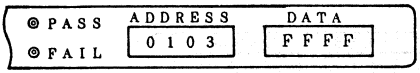


Fig. 4.48

⑪ Press the ↑ key. The panel display gives the following indication, making it possible to ascertain that the data has been changed:

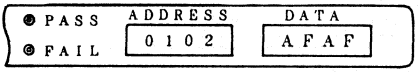


Fig. 4.49

⑫ Press the ↓ key. The panel display gives the following indication, making it possible to ascertain that the data has been changed:

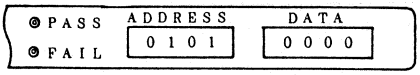


Fig. 4.50

⑬ Press the + key. The panel display gives the following indication, making it possible to ascertain that the data has been changed.

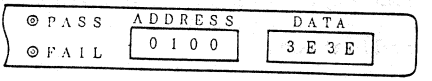


Fig. 4.51

⑭ Press the RESET key to terminate the CH command.

4.2.4 INS mode

This mode is used to insert new data in the data string into the buffer memory. In this mode, address of the data in and after the inserted address is incremented by 1. Therefore, the data in the last address of the buffer memory is deleted.

(Example)

Insert data FFH in address 1.

Before data insertion			After data insertion	
Address	Data		Address	Data
0000	00	Insert FF	0000	00
0001	01	→	0001	FF
0002	02	→	0002	01
⟆	⟆		0003	02
3FFD	FD	→	⟆	⟆
3FFE	FE	→	3FFE	FD
3FFF	FF	→	3FFF	FE
			Deleted	

Operation:

① Press the PANEL key. This causes the PANEL lamp on the panel to be light.

② Press the INS key. The panel display gives the following indication:

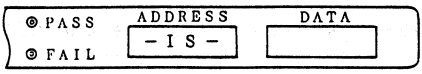


Fig. 4.52

③ Key in the address to be inserted. The panel display gives the following indication:

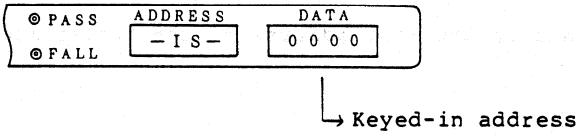


Fig. 4.53

④ Press the A-ENT key to set the keyed-in address. The panel display gives the following indication:

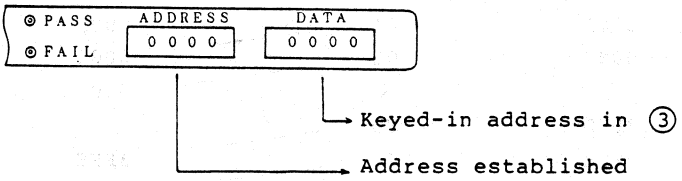


Fig. 4.54

⑤ Key in the data to be inserted. The panel display gives the following indication:

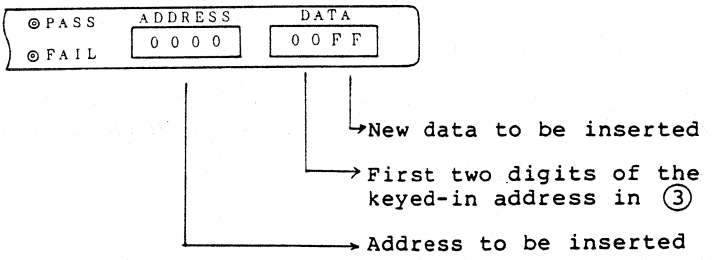


Fig. 4.55

⑥ Press the + key to set the data to be inserted. The panel display gives the following indication:

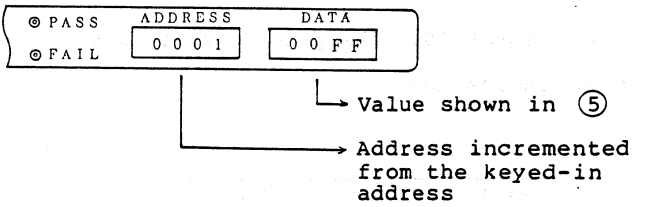


Fig. 4.56

If it is desired to insert data in succession, repeat operations ⑤ and ⑥. To terminate the INS command, press the RESET key. With this, the system is placed in the command wait state. The maximum address available for data insertion is 3FFF.

Sample operation

Insert data C3H, 00H and 10H in the addresses 100, 101 and 102, respectively. (The buffer memory is assumed to contain FFH.)

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the INS key. The panel display gives the following indication:

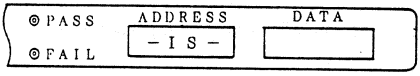


Fig. 4.57

③ Key in address 0100. The panel display gives the following indication:

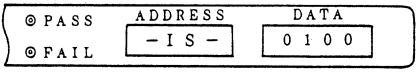


Fig. 4.58

④ Press the A-ENT key. The panel display gives the following indication:

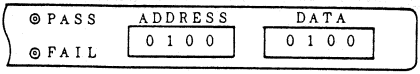


Fig. 4.59

⑤ Key in data C3 to be inserted. The panel display gives the following indication:

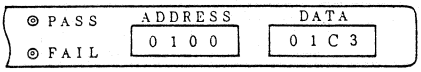


Fig. 4.60

⑥ Press the ↑ key. The panel display gives the following indication:

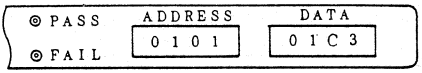


Fig. 4.61

⑦ Key in data 00 to be inserted. The panel display indicates the following:

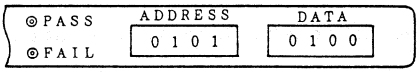


Fig. 4.62

⑧ Press the ↑ key. The panel display indicates the following:

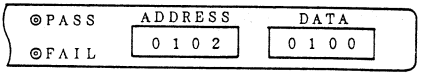


Fig. 4.63

⑨ Key in data 10 to be inserted. The panel display indicates the following:

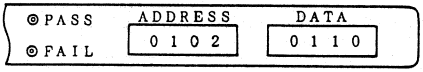


Fig. 4.64

⑩ Press the ↑ key. The panel display gives the following indication:

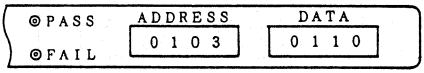


Fig. 4.65

⑪ Since the data insertion has been completed, press the RESET key to terminate the INS command:

4.2.5 DEL mode

This mode is used to delete data in the data string from the buffer memory. There are two types of deletion: single address deletion and block deletion.

In this mode, if the data in a certain address is deleted, the addresses of data in subsequent address is decremented by 1. Therefore, FFH is input into the empty address.

Example

Delete the data in address 1.

Before deletion

Before deletion			After deletion	
Address	Data		Address	Data
0000	00	Deleted	0000	00
0001	01		0001	02
0002	02		⟆	
⟆	⟆		7FFD	0E
7FFE	0E		7FFE	0F
7FFF	0F		7FFF	FF

Operation

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the DEL key. The panel display indicates the following:

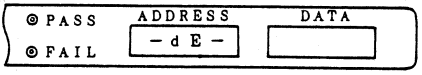


Fig. 4.66

③ Key in the address to be deleted or the first address of the block to be deleted. The panel display gives the following indication:

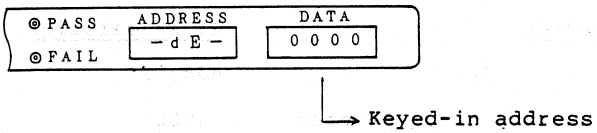


Fig. 4.67

④ Press the A-ENT Key to set the keyed-in address. The panel display gives the following indication:

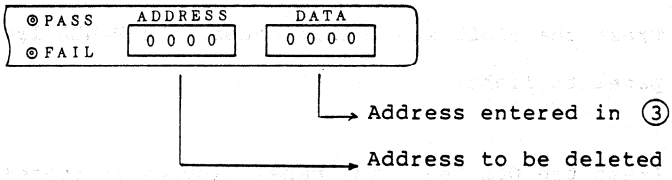


Fig. 4.68

If a single address is to be deleted, go to ⑦.

⑤ Key in the end address of the block to be deleted. The panel display gives the following indication:

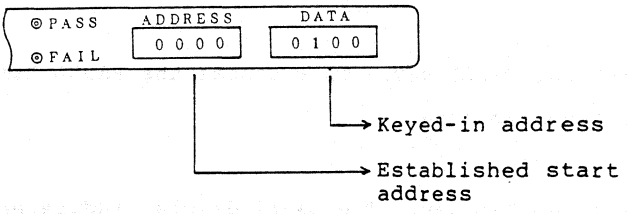


Fig. 4.69

⑥ Press the A-ENT key to establish the keyed-in address. The panel display gives the following indication:

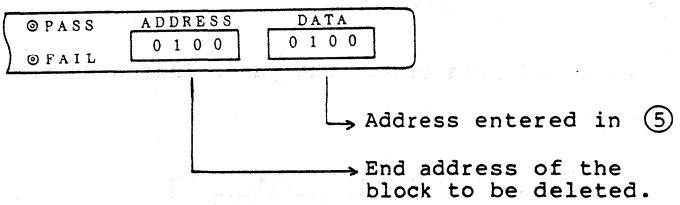


Fig. 4.70

⑦ Press the ↑ key to execute the DEL command. After the execution, the panel display gives the following indication, showing the termination of the DEL command:

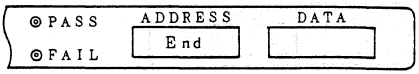


Fig. 4.71

Sample Operation

1. When deleting the content of address 100:

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the DEL key. The panel display indicates the following:

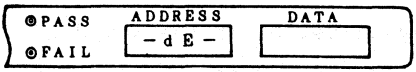


Fig. 4.72

③ Key in address 0100. The panel indicates the following:

Fig. 4.73

④ Press the A-ENT key. The panel display gives the following indication:

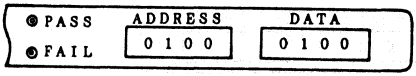


Fig. 4.74

⑤ Press the ↑ key to execute the DEL command. After the execution, the panel display indicates the following showing the termination of the DEL command:

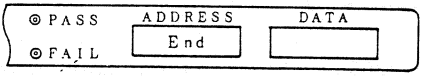


Fig. 4.75

2. When deleting the contents of address 100 to address 200:

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the DEL key. The panel display indicates the following:

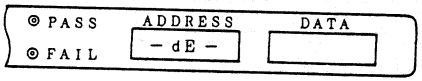


Fig. 4.76

③ Key in address 0100. The panel display gives the following indication:

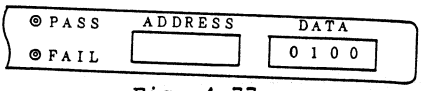


Fig. 4.77

④ Press the A-ENT key. The panel display gives the following indication:

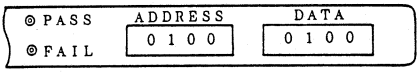


Fig. 4.78

⑤ Key in address 0200. The panel display gives the following indication:

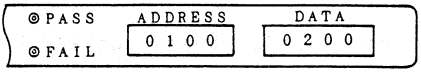


Fig. 4.79

⑥ Press the A-ENT key. The panel display gives the following indication:

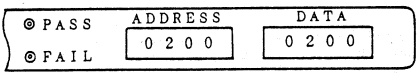


Fig. 4.80

⑦ Press the ↑ key to execute the DEL command. After the execution, the panel display gives the following indication, showing the termination of the DEL command:

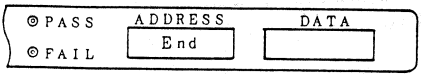


Fig. 4.81

4.2.6 INIT mode

This mode is used to initialize the buffer memory. The entire buffer memory is initialized and after that, all the contents of the buffer memory become FFH.

Operation:

① Press the PANEL key. This causes the PANEL lamp on the panel to light.

② Press the INIT key. The panel display gives the following indication:

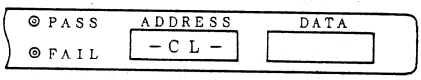


Fig.4.82

③ Press the ↑ key to execute the INIT command. After the execution, the panel display gives the following indication, showing the termination of the INIT command:

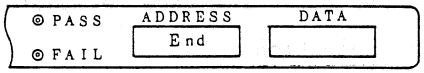


Fig. 4.83

In this state, the system is in the INIT mode. The system can accept all panel and control commands.

4.2.7 CMP mode

This mode is used to reverse the data string in the buffer memory one bit at a time.

(Example)

For 00H

00000000 00H

⇓ Reverse

11111111 FFH

for 31H

00110001 31H

⇓ Reverse

11001110 CEH

The CMP mode is effective in the range set by the PAE mode.

Operation:

① Press the PANEL key. This causes the PANEL lamp to light.

② Press the CMP key. The panel display gives the following indication:

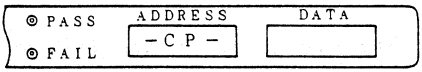


Fig. 4.84

③ Press the ↑ key. With this, the CMP command is executed. After the execution, the panel display gives the following indication, showing the termination of the CMP command.

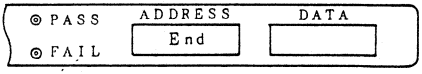


Fig. 4.85

In this state, the system is in the CMP mode and can accept all panel and control commands.

APPENDIXES

APPENDIX A LIST OF COMMANDS FOR HOST MODE AND CONSOLE MODE

Key-word	Host mode	Con-sole mode	Command format	Function
A	o	o	*As,e,r↓	Parameter setting
B				
C				
D				
E	o	o	*Er↓	Buffer data change
F	o	o	*Fs,e,d↓	Buffer initialization
G	o		*G filename↓	File save
H				
I		o	*I↓	Move to intelligent mode
J	o	o	*J↓	Parallel input
K	o		*K filename,o,c↓	File loading
L	o		*L↓	Data transfer from MD to PG
M	o		*M filename,o,c↓	(K command) + (L command)
N				
O	o	o	*Os,e↓	Buffer indication
P	o		*Ps,e↓	Data transfer from PG to MD
Q	o		*Q↓	Operation mode selection
R	o	o	*Rs,e,↓	PROM reading
S	o	o	*Sn↓	PROM selection
T		o	*T↓	Move to terminal mode
U				
V	o	o	*Vs,e,r↓	PROM and buffer comparison
W	o	o	*Ws,e,r↓	Writing data into PROM
X				
Y	o	o	*Y↓	A command parameter indication
Z	o	o	*Z↓	PROM blank check

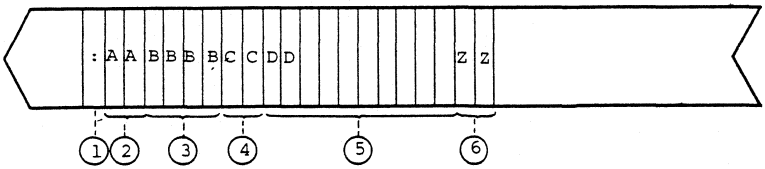
APPENDIX B LIST OF ERROR CODES

Indication	Error type
E-1	Memory error
E-2	Parity error
E-3	Format error
E-4	Address entry error
E-5	PROM unit not inserted
E-6	Format unspecified
E-7	Paper tape verification error
E-11	Vpp (5V) abnormality
E-12	Vpp (25V or 21V) abnormality
E-13	Vcc (5V) abnormality
*E-14	Reversed insertion
E-15	Other errors

* Even if the ROM is inserted normally, the reversed insertion message "E-14" may be indicated. In such cases, ensure that the ROM is inserted normally and, if so, press the START key again. The ROM then becomes operable in the selected mode.

APPENDIX C HEX FORMAT FILE

Files supported by the PG-1000 have a HEX format called the Intel format. The format consists of an ASCII character string as shown below:



Intel HEX format

① Record mark

Shows the beginning of records.

② Record length

Shows the length of data. If this is 0, the record is regarded as the last record.

③ Head address

Shows the head address in four digits.

④ Record type

The PG-1000 ignores this part.

⑤ Data area

The area that contains data. As much data as shown by the record length is stored.

⑥ Check sum

Shows the check sum in two digits. The area for check sum generation is from the record type to the end of the data.

Book 4

PG-1003

USER'S MANUAL

PREFACE

This manual explains PG-1003 used as a PROM programmer by connecting it to a PG-1000.

Chapter 2 lists the items accompanying this product; when unpacking, check the unit and attached documents. Chapter 3 explains the connection to the PG-1000. Connect this unit to the PG-1000 according to the procedure described in this chapter. Chapter 4 describes the PG-1003 writing modes. There are two types of writing modes.

When this product is connected to a PG-1000 for use as a PROM programmer, refer to the PG-1000 operation manual for the operating procedure.

NOTES ON USING THIS PRODUCT

The following cautions should be noted when using this product. NEC cannot accept responsibility for any malfunction resulting from the user ignoring these cautions.

1. Do not connect this unit to any device other than the PG-1000.

2. Do not connect and disconnect this unit to any from PG-1000 before removing the power plug of PG-1000.

3. Do not turn power to PG-1000 on and off when PROM is inserted in this unit, as it may damage both the PROM and this product.

4. Do not use this unit in an environment where there is electrical interference as this may cause a malfunction.

5. Avoid high temperature, humidity, and dust as such conditions may lead to a malfunction.

CHAPTER 1 OUTLINE

The PG-1003 is a personal module that can function as the PROM programmer for μPD78P09R by connecting it to a PG-1000. The PG-1003 connected to the PG-1000 is used in one of three modes: host mode when connected to the host machine MD-080/086, console mode when connected to a console, and stand alone mode when used alone. There are two types of writing modes: high-speed writing mode in which data is written rapidly and normal writing mode in which data is written surely.

1.1 Operating Environment

Table 1.1 Operating environment

Temperature	10 to 35°C	During operation
Humidity	20 to 80% RH	

CHAPTER 2 ATTACHMENTS

Check that the following items are contained in the carton. If any item is missing, contact your dealer.

1. PG-1003 unit
2. Operation manual
3. Warranty
4. Attachment list

Since the warranty is required when a trouble occurs, retain it carefully.

CHAPTER 3 APPEARANCE

3.1 Top View

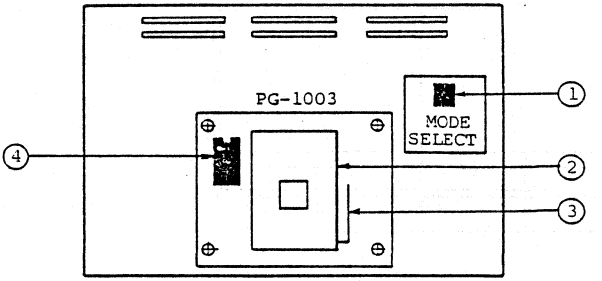


Fig. 3.1 Top view of PG-1003

① Mode select switch

This switch is used to select the writing mode.

② PROM socket

PROM is inserted into the PROM socket.

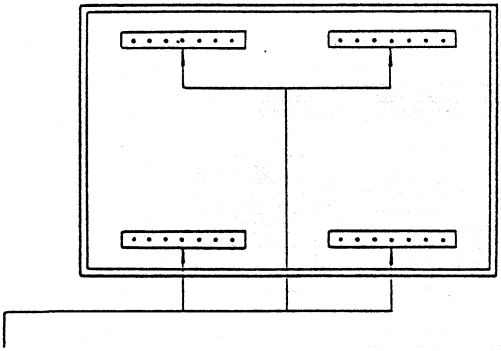
③ Lever

Before inserting PROM, raise this lever. After it has been inserted, lower this lever to lock.

④ PROM insert direction indicator

Indicates the direction in which PROM should be inserted.

3.2 Bottom View



Connector (used when connected to PG-1000)

Fig. 3.2 Bottom view of PG-1003

CHAPTER 4 CONNECTION TO PG-1000

This chapter explains the connection of PG-1003 to PG-1000. Perform connection according to the procedures described in (1) to (8) below:

(1) Disconnect power to PG-1000 and remove the power cable.

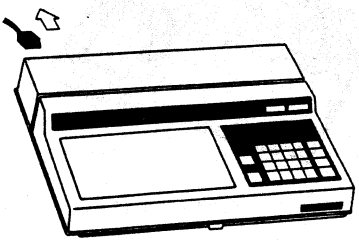


Fig. 4-1

(2) Unlock by pulling the tab at the bottom of the front of the PG-1000 shown in Figure 4.2.

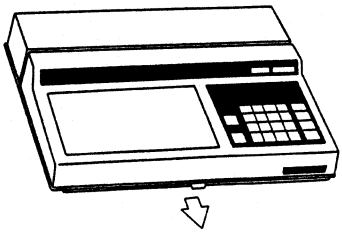


Fig. 4-2

(3) Raise the cover, and the PG-1000 will appear as shown in Figure 4.3 If another personal module is installed in the position indicated by the circle, raise it upward and remove it.

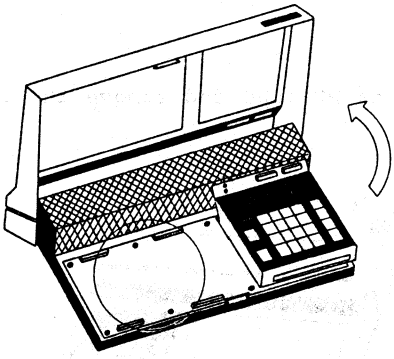


Fig. 4-3

(4) Connect PG-1003 to PG-1000 as shown in Figure 4.4 below. Note that bent or broken connector pins will cause bad connection, and lead to incorrect operation.

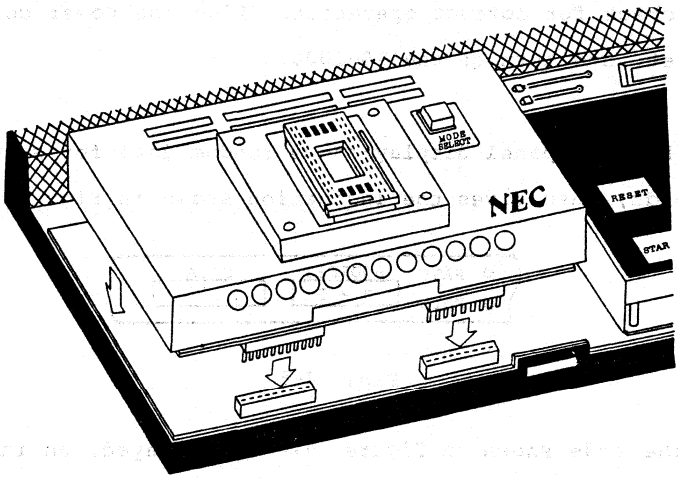


Fig. 4-4

Set the connectors of PG-1003 in the corresponding positions of PG-1000.

(5) Close the cover.

(6) Insert the power cord in PG-1000.

(7) This completes the connections.
Now check for correct operation. Plug the power cord in the socket, and turn on the PG-1000.

(8) The PG-1000 panel display shows random data for several seconds, then gives the indication shown in Figure 4.5

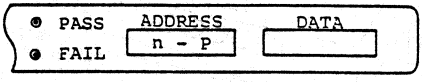


Fig. 4-5

If the code shown in Figure 4.6 displayed, an insertion error may have occurred. Carry out the above operations once again carefully.

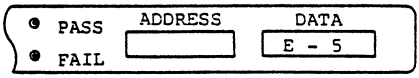


Fig. 4-6

Correct your dealer if other error codes are displayed, since PG-1003 or PG-1000 may be defective.

CHAPTER 5 SELECTION OF FROM TO BE USED

When PG-1003 has been connected to PG-1000, the writing mode can be selected when writing data to PROM. Table 5.1 lists the PROM used and writing modes.

Table 5.1

PROM used	μPD78P09R
Writing mode	High-speed writing mode
	Normal writing mode

5.1 Selection Procedures

(1) Press the MODE SELECT switch. The PG-1000 panel display gives the writing mode currently selected as shown in Figures 5.1 and 5.2.

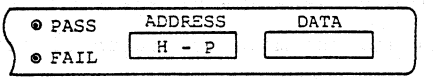


Fig. 5.1 Display in high-speed writing mode

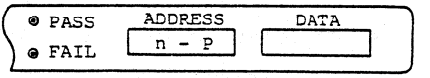


Fig. 5.2 Display in normal writing mode

When power is turned on, the normal writing mode is set.

(2) Select the writing mode using the [↑] key and [↓] key or [0] key and [1] key. The relationships between the key numbers and writing modes are shown in the following table.

Key number	Display	Writing mode
0	0 - F	Normal writing
1	0 - F	High-speed writing

① [0], [1] key

Displays the writing mode corresponding to the key number [0] and [1].

② [↑], [↓] key

Displays a writing mode different from the writing mode currently being displayed.

(3) Catalog the writing mode by pressing the [A-ENT] key. At the completion of cataloging, the PASS lamp of the PG-1000 panel lights and the mode selection operations terminate.

Note: Pressing the [RESET] key before pressing the [A-ENT] key returns to state (1).

5.2 Setting

When PG-1003 is connected to PG-1000, PG-1000 automatically sets the last address of PROM used by the PAE instruction. The last address is X'1FFF'.

CHAPTER 6 VOLTAGE REQUIREMENTS

PG-1003 sets the voltages to μPD78P09R V_{CC} (power supply terminal) in each operating mode as follows, thereby guaranteeing the margin of data written to PROM.

Table 6.1

Item / Mode	V_{CC} (V)	V_{REF} (V)
COPY	5.0	1.5
BLANK	5.25	0.6
PROGRAM	5.0	1.5
VERIFY (*1)	4.75/5.25	0.6/2.4

*1: VERIFY is performed in two conditions.

APPENDIX A WRITING TIME

As described earlier, PG-1003 has two types of writing mode. The writing times when writing is performed to all addresses of μPD78P09R in the respective modes are shown in the following table:

Table 1 Writing time

Writing mode	Time (seconds)
Normal writing mode	Approximately 410
High-speed writing mode (*1)	Approximately 16 minimum
	Approximately 330 maximum

*1 The time depends on the PROM characteristics.

APPENDIX B EXAMPLE OF USED OF PG-1003

The contents of all addresses of μPD78P09R are read into PG-1000. The contents X'3E' and X'55' at addresses X'8A' and X'83' are changed to X'00' and the changed contents are written to another erased μPD78P09R. (The high-speed writing mode is used.)

(1) Connect PG-1003 to PG-1000, plug in the power cord, then turn the power switch on. At this time, the panel display gives the indication shown in Figure 1, indicating that the PG-1000 is ready.

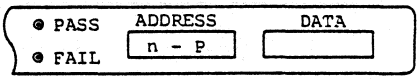


Fig. 1

(2) Insert μPD78P09R containing data into the PG-1003 socket.

(3) Press the [COPY / E] key. The COPY lamp on the panel lights.

(4) Press the [START] key. The copy instruction is executed.

(5) Press the [PANEL / C] key. The PANEL lamp on the panel lights.

(6) Press the [CH 5] key. The panel display gives the indication shown in Figure 2.

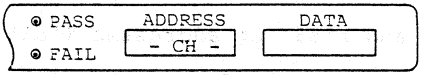


Fig. 2

(7) Key in address X'008A'. The panel display gives the indication shown in Figure 3.

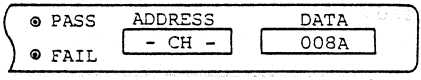


Fig. 3

(8) Press the [A-ENT] key. The panel display gives the indication shown in Figure 4.

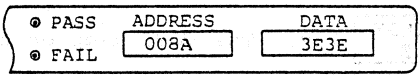


Fig. 4

(9) Key in data X'00'. The panel display gives the indication shown in Figure 5.

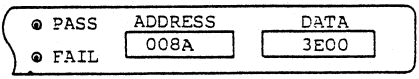


Fig. 5.

(10) Press the [↑] key. The panel gives the indication shown in Figure 6.

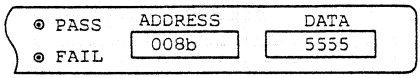


Fig. 6

(11) Key in data X'00'. The panel display gives the indication shown in Figure 7.

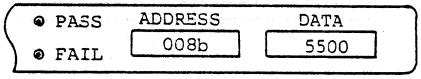


Fig. 7

(12) Press the [↑] key. The panel display gives the indication shown in Figures 8.

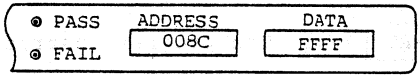


Fig. 8

(13) Press the [RESET] key to terminate the CH instruction.

(14) Remove the μPD78P09R inserted in the PG-1003 socket and insert another erased μPD78P09R into the socket.

(15) Press the mode select switch. The panel display gives the indication shown in Figure 9.

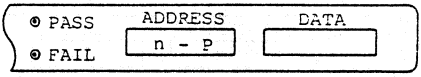


Fig. 9

(16) Press the [↑] key. The panel display gives the indication shown in Figure 10.

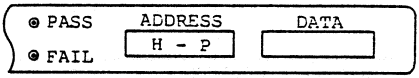


Fig. 10

(17) press the [A-ENT] key. The PASS lamp lights.

(18) Press the [PRG 8] key. The PROGRAM lamp on the panel lights.

(19) Press the [START] key. The program instruction is executed.

By the above operations, PROM data is read and changed, and data is written to PROM. For details of specific instructions, refer to the PG-1000 operation manual.

Book 5

RA-87 / RA-310

RA-210 / RA-110

RELOCATABLE ASSEMBLER

USER'S MANUAL

Preface

The RA87 relocatable assembler package supports the high end microcomputers of NEC's powerful 8-bit ucom-87 microcomputer family.

8 bit microcomputers supported by the assembler package are:

uPD7807, uPD7808, uPD7809
uPD7810, uPD7811,
uPD78c10, uPD78C11,uPD78C14

The RA310 relocatable assembler package supports the high end 16 bit microcomputers uPD78312/78310.

The RA210 relocatable assembler package supports the 8 bit microcomputers uPD78210, uPD78220/uPD78224.

The RA110 relocatable assembler package supports the 8 bit microcomputer uPD78112.

The relocatable assembler packages consist of an assembler, linker locator and a librarian. The assembler package translates an assembly language ASCII source program into an object file which can be executed by the target microcomputer.

The assembler packages are supported only by 16 bit operating systems as follows:

- CP/M-86 (*3) 16-BIT OS V1.0
- MS-DOS (*4) 16 BIT OS V2.11
  or PC-DOS (*7) 16 BIT OS V3.0 (or compatible with regard to this software
- UDI (*5) 16 BIT OS

Note: Previous RA87 version (V1.5) have been supported by 8 BIT OS like CP/M80 V2.2 or ISIS II V4.3. This support is not available any more for the software described in this manual. Please refer to manual of 7/85 V1.5.

RA310,RA210 and RA110 will not be supported by 8 BIT OS at all.

Note: UDI (*6) is the UNIVERSAL DEVELOPMENT INTERFACE defined by INTEL Corp.. It can be run on ISIS-IV (*8), ISIS-III (*8) and various UDI (*5) Emulators on IBM PC etc.

The assembler packages are as well suported by VAX based operating systems as follows:

- VMS V4.1 (*6)
- ULTRIX BERKELY V4.2 (*6)

Note: This manual refers to RA87/RA310/RA210/RA110. The operation of the RA210/RA110 is identical to RA310. Some small differences are explained in the appendixes or on the text. Please refer to it.

*3 CP/M-86 is a trade mark of Digital Research Corp.
*4 MS-DOS is a trade mark of Microsoft Corp.
*5 UDI is a trade mark of INTEL Corp.
*6 VMS/ULTRIX are trade marks of DIGITAL EQUIPMENT Corp.
*7 PC-DOS is a trade mark of IBML Corp.
*8: ISIS III/IV are trade marks of INTEL Corp.

1. RaXXX Relocatable Assembler Package Summary

1.1. Software Supply Form

The RA87/RA310/210/110 relocatable assembler package is supplied in form of a flexible disk for PCs which contains the machine executable program. Table 1.1 shows the flexible disk formats for different operating systems and the PC host development machines.
For VAX the software is supplied on magentic tape, 1600 bpi, EBCDIC Code.

RA87 Package Name	Operating System	Floppy Disk	Host Development Machine
RAC86-08SS-87	CP/M-86 *3	8" single sided single density	APC or any standard PC supporting CP/M-86 V 1.0
RAC86-I5DD-87	CP/M-86 *3	5 1/4" double sided double density	IBM-PC
RAMSD-I5DD-87	MS-DOS *4	5 1/4" double sided double density	IBM-PC
RAUDI-I5DD-87	MS-DOS *5	5 1/4" double sided double density	IBM-PC

RA310 Package Name	Operating System	Floppy Disk	Host Development Machine
RAC86-08SS-78310	CP/M-86 *3	8" single sided single density	APC or any standard PC supporting CP/M-86 V x.x
RAC86-I5DD-78310	CP/M-86 *3	5 1/4" double sided double density	IBM-PC
RAMSD-I5DD-78310	MS-DOS *4	5 1/4" double sided double density	IBM-PC
RAUDI-I5DD-78310	UDI *5	5 1/4" double sided double density	IBM-PC

RA210 Package Name	Operating System	Floppy Disk	Host Development Machine
RAC86-08SS-78210	CP/M-86 *3	8" single sided single density	APC or any standard PC supporting CP/M-86 V x.x
RAC86-I5DD-78210	CP/M-86 *3	5 1/4" double sided double density	IBM-PC
RAMSD-I5DD-78210	MS-DOS *4	5 1/4" double sided double density	IBM-PC
RAUDI-I5DD-78210	UDI *5	5 1/4" double sided double density	IBM-PC

RA110 Package Name	Operating System	Floppy Disk	Host Development Machine
RAC86-08SS-78110	CP/M-86 *3	8" single sided single density	APC or any standard PC supporting CP/M-86 V x.x
RAC86-I5DD-78110	CP/M-86 *3	5 1/4" double sided double density	IBM-PC
RAMSD-I5DD-78110	MS-DOS *4	5 1/4" double sided double density	IBM-PC
RAUDI-I5DD-78110	UDI *5	5 1/4" double sided double density	IBM-PC

Note: VMS/ULTRIX based software order numbers were not yet available during edition of this manual. Please ask your sales office.)

Table 1.1

1.2. System Environment

The RA87/RA310/210/110 assembler package is available for operation under different operating systems. Table 1.2 shows the requirements for the operating system

release versions and the minimum memory required to run the assembler package.

Operating System	Version	Minimum System Memory Needed
CP/M-86 *3	V 1.0	128K bytes
MS-DOS *4	V 2.11	128K bytes
UDI *5		128K bytes
VMS *6	V 4.1	t.b.d.
ULTRIX *6	V 4.2	t.b.d.

Table 1.2

Note: *3, *4, *5 ,*6 Please refer to Table 1.1

RA87/RA310/210/110 (CP/M86,MSDOS) can be run under CPM/86, PCDOS/MSDOS on the IBM PC with min. 128 KByte memory or compatible PCs.
RA87/RA310/210/110 (UDI) can be run under ISIS III/IV (*8) on INTEL MDS series III/IV or IBM PC (with UDI emulator software) with min. 128 KByte memory.

VAX VMS/ULTRIX requirements are to be defined later.

1.3 Text Editors

RA87/RA310/210/110 requires a Source file (Input-file) containing ASCII characters conforming to RA87/RA310/210/110 specification. Other characters, which are not recognized as RA87/RA310/210/110 Comments, result in Assembler-Error.

Different text editors supported by the RA87/RA310/210/110 assembler package are

ED	for CP/M-86
WORDSTAR	for CP/M-86, MS-DOS
ED	for VMS/Ultrix
CREDIT/AEDIT	for UDI

2. Structure of Assembler RA87/ RA310 / RA110 / RA210

2.0 General Structure

The RAXXX relocatable assembler translates the source file into a relocatable object file. Fig. 2.0.0 shows a RAXXX system configuration.

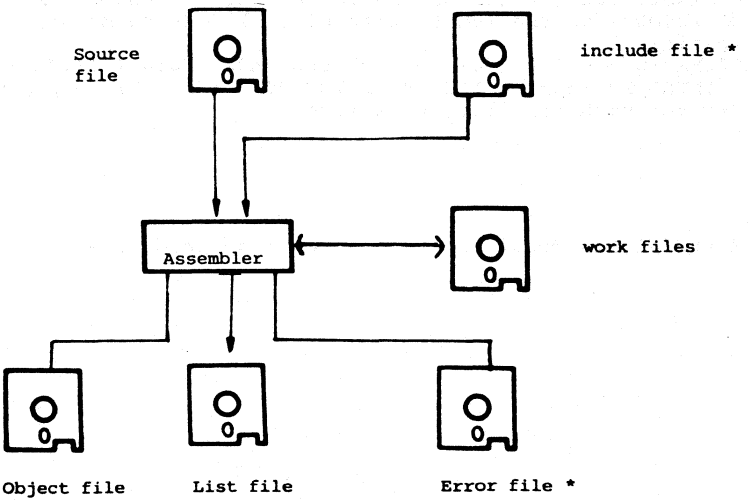


Fig. 2.0.0 RAXXX Configuration

The source file contains a program module and may call an include file. The include file itself is a source program module. The output of an assembled source file is a relocatable object file, a list file which contains the source code together with the object code and an (optional) error file. The following chapters would give the details about the construction of the individual modules.

2.0.1 Tools for the modular construction of programs for the uCOM87/uCOM78xx family processors

- Assembler
- Linker
- Locater
- Librarian
- In-circuit-Emulator

Here,the Assembler will be considered. The Assembler converts an ASCII Inputfile into a relocatable Object-file.

Input-File is:

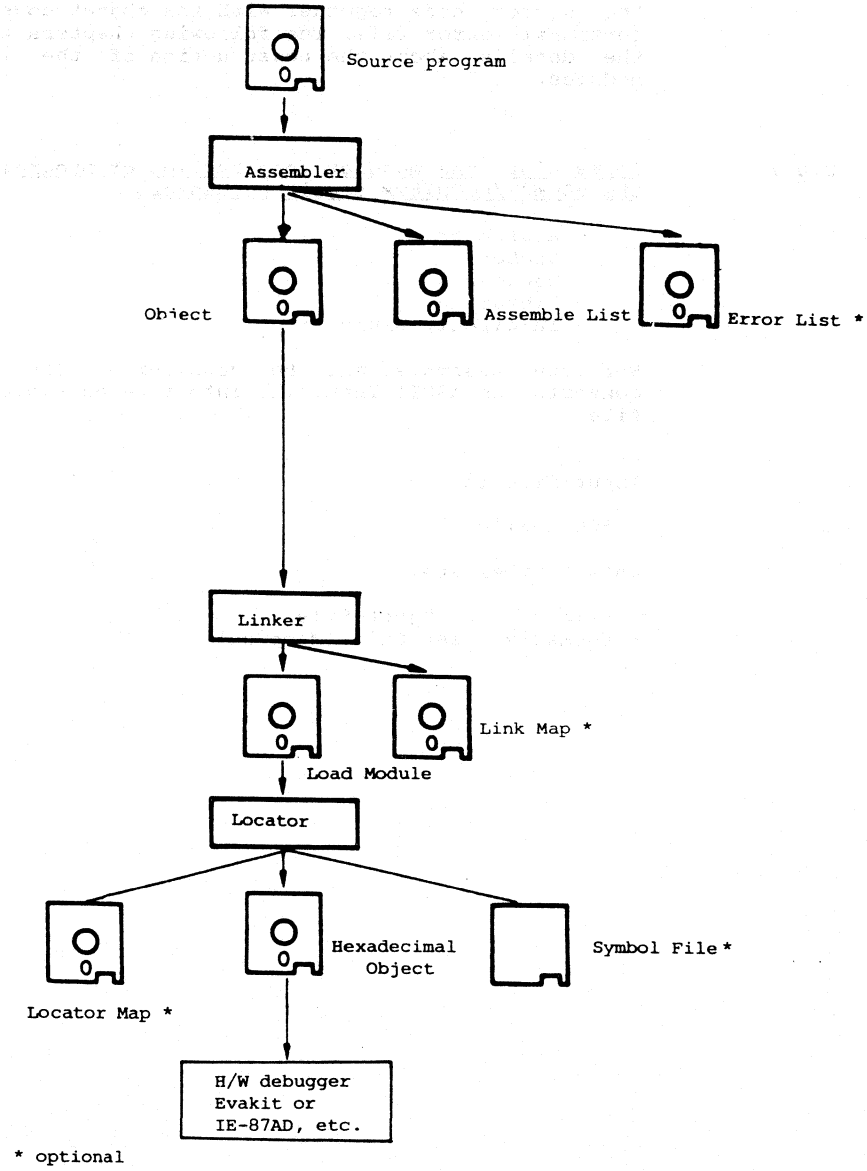
- ASCII-File

Output-Files are:

- relocatable Object-File
- formative List-File (Mapping)

2.0.2 Program Development

Fig. 2.0.2 shows the program development process with RAXXX assembler package.



2.1 Source-file format

2.1.1. Module Structure

An Assembler-Module consists of:

- Header
- Body
- Tail

Additionally comments can be made at any desired positions in Module.

2.1.1.1. Module Header

At first the Module Header must contain:

- Directive NAME (= 1. Non Command line)

The Module Header may contain:

- Control Command lines
  - - Primary Control Commands in front of NAME
  - - General Control Commands anywhere

- Global definitions:
  They only can appear in the Modul Header.
  - - PUBLIC-Directives
  - - EXTRN-Directives
  - - EXTBIT-Directives

- Constant definitions:
  - - EQU-Directives
  - - SET-Directives

Sequence in Module Header:

1.) Command Lines with (Primary) Control Commands
2.) Module definition:
- NAME
- PUBLIC-Directives or EXTRN-Directives or EXTBIT-Directives
- EQU-Directives or SET-Directives

After the Module definition NAME, any desired Command lines can be entered, which may also contain General Control Commands.

2.1.1.2. Module Body

The Module Body can contain:

- Control Command lines with General Control Commands
- A Code Segment
- A Data Segment
- RA87 only: 5 absolute-Segments
  1 relocatable V-Segment
- Ten Absolute-Segments (Through ORG-Directive)

> any desired sequence

- Directive: DBIT (only in V-Segment of uPD 7809 or in B-Segment of RA310 / RA110 / RA210))
- Directives: DB, DW, DS
- Directive: ENDS (at the end of a Segment)
- Macro-Directives
- Machine Instructions as per Instruction Set of Processor

2.1.1.3. Module Tail

The Module Tail signifies the end of a Module and can contain:
The Module Tail is optional.

- END-Directive

2.1.2. Instruction Set

RA87 supports the Instruction Sets of the Processors uPD7809, uPD7807, uPD7810, uPD78C10, uPD78C11, uPD78c14, uPD7811 as per NEC Product Description.
RA310 supports the Instruction Set of the Processor uPD78310.

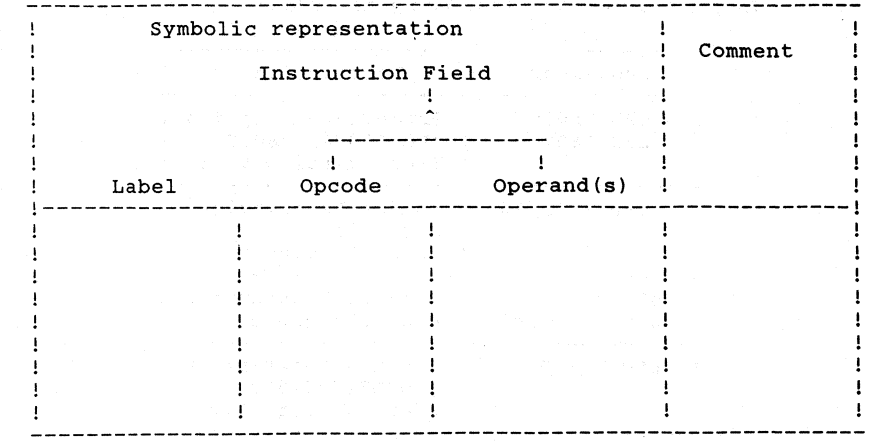
RA110 supports the Instruction Set of the Processor uPD78112.
RA210 supports the Instruction Set of the Processor uPD78210, uPD78220, uPD78224.

In particular the following is valid:

Processor	Description
uPD 7809G uPD 7807	Product-Description uPD 7809G/ 7807G 7/83, Version V1.1
uPD 7811/10	Product-Description uPD 7811 6/83, Version V2.0
uPD 78c14/ uPD 78c11/c10	Product-Description uPD78c14/c10/c11
uPD 78310/312	Product-Description uPD78310/312 8/86, Version 1.2
uPD 78112	Specification by 13 March 1986
uPD 78210	Specification by 3 June 1986
uPD 78220 uPD 78224	Specification by 3 June 1986

2.1.3. Instruction Format and Length

Instruction Format



1. Label Field: This Field is optional and can contain the Symbolic Address of the following instruction.

2. Instruction Field: This Field encompasses:
   - The mnemonic Opcode
   - dependent on Opcode 0/1/2 Operands (displacements)

3. Comment Field: This Field is optional.

- The Assembler translates only the Instruction Field in Opcode. This results in different Opcode-lengths in the order of 1-5 bytes.

- The Instruction format may be represented in BNF as follows:

  [[label:] [Opcode [DISPL1[,DISPL2]][;comment]]]<cr><lf>

2.1.4. Symbols

RA87/ RA310 / RA110 / RA210 allows symbolic Variables and Procedure References. A Symbol may contain a maximum of 6 Characters, which occur in the following limited ASCII Character Set.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789?@(Ampersand)_ (Underscore)
```

The following Characters can be used to seperate Symbols:
blank, CR, FF, HT.

The special symbol"$" can be used as an optical divide character, but is not significant by name.

The first Character must always be a letter.
A Symbol may not start with a number or "_".
The Assembler does not differ between Upper and Lower Case characters.
Key Words (s. 2.1.12.) are reserved. It is not allowed to use these words as Symbols.
Symbols are generally valid within the Modules in which they were arranged.
An extension of validity can only be achieved by an explicit declaration (PUBLIC, EXTRN, EXTBIT).

2.1.5. Data

2.1.5.1. Simple Data Types

Type	Range,decimal	Length
Bit	0, 1	1 Bit
Byte	0...255	1 Byte
Word	0...65535	2 Byte

Directives:

```
DBIT --> define bit
DB   --> define byte
DW   --> define word
```

```
Example:    ONE     DBIT
            TWO  :  DB
            THREE:  DW
```

2.1.5.2. Data Fields

RA87 / RA310 / RA110 / RA210 recognizes only 1-dimensional
Fields with the same Data type pro Field element.
Valid Types can be Byte-or Word sized.
Bit-Fields cannot be defined in V-Segments / B-Segments.
For definitions, see directives under 2.1.10.3.1.

```
Example:  ARRAY: DB (10)    ; defines ARRAY as Field with
                            ; size of 10 Bytes.
     or   FIELD: DS 10      ; defines FIELD as field with
                            ; size of 10 Bytes.
```

2.1.5.3. Data declarations

Segment	allowed are
Code-Segment	DB, DW, DS
Data-Segment	DB, DW, DS
V-Segment	DBIT, DB, DW, DS
B-Segment	DBIT [only RA310 / RA110 / RA210]

Data will be addressed via a Data declaration. Normally, this address will be relative to the start of the corresponding Segment. The directive ORG is used for Absolute Addressing.

```
Example:    Data Segment
              :
            VAR1: DW        ; Address of VAR1 lies rela-
                            ; tive to start of Data Seg-
                            ; ment
            Data Segment
            ORG  100H
              :
            VAR2:  DW       ;  Address of VAR2 lies at
                            ; absolute Address 0100H
```

2.1.6. Numerical Formats

The Assembler RA87 / RA310 / RA110 / RA210 supports 4 Numerical Formats:

Format	allowed numbers	Symbol
Binary	0 1	B
Octal	0 1 2 3 4 5 6 7	O
Decimal	0 1 2 3 4 5 6 7 8 9	[D]
Hex	0 1 2 3 4 5 6 7 8 9 A B C D E F	H

Hex values always begin with a number (see Example). The default Numerical Format is Decimal.

```
Example:    01111111B  -->  Binary number
            71O        -->  Octal number
            11         -->  Decimal number
            11D        -->  Decimal number
            2AH        -->  Hexadecimal number
            A2H        -->  Error(correct: 0A2H)
```

2.1.7. Procedures

The instructions shown here for handling of Procedures, have been taken from uPD7809/7811 instruction Set. For other Processors, other instruction mnemonics are valid.

Procedures are contained only in Code Segments. They consist of one or more Instructions. The last executable instruction of a Procedure is:

- RET
- RETI
- RETS
- As desired (before Directive END/ENDS --> Procedure is Main Program).

A Procedure can be called via:

- CALL
- CALB
- CALF
- CALT

A Symbolic label can be set before the first executable instruction of a Procedure. The Procedure can be activated by reference to this Symbol.

```
Example:    Proc1:  statement
                    statement
                       :
                     RET

      Call:e.g.:  CALL  Proc1
```

The Directive ORG fixes Procedures to absolute memory Addresses.
The management of a Stack (especially at Parameter transfer) belongs to the user.

```
Example:                          ; Procedure will be stored
          ORG       1000H         ; in memory starting at
      1) Proc1: statement         ; Address 1000H
                statement
                   :
                   :
                 RET

      2) Parameter transfer
         Proc1: :
                PUSH  par1        ; first   Parameter
                PUSH  par2        ; second  Parameter
                CALL  Proc2       ; will be pushed onto Stack
                RET

         Proc2: POP   par2        ; Parameters will be
                POP   par1        ; popped off Stack
                statement
                    :
                    :
                RET
```

2.1.8. Macros

RA87 / RA310 / RA110 / RA210 allows the definition of Macros. A Macro represents a sequence of instructions, which is activated in a similar way to a Program subroutine. Instead of the subroutine activation and subsequent increase of Program execution time, the Macro activation is via implicit Code operation.

- The instruction sequence defined in a Macro, will be copied by the Assembler into the Sourcetext for every Macro Call.

- Advantage is the reduced Program Run Time.

- Principally Macros can only be defined in the Code Segment of the actual Module. It is possible, however, per INCLUDE-Command, to load a Macro from an external File into the current Module. In doing this one must pay attention to the fact that the relevant INCLUDE-Command can only appear in Code Segment.

- Macros must already be defined before they can be activated for the first time.

- The Macro Name must obey the rules for construction of Symbol.
  The Name may not appear in the PUBLIC-, EXTRN-, EXTBIT-List.

- A label or symbol defined in Macro should be defined with the Local-Directive immediately after the definition of a Macro itself.
  Failing to do so, would result in redefinition of the symbol with each Macro call.

- The number of current Parameters at Macro Call must correspond with the number and Data type of the formal Parameters in the Macro definition and may not exceed 5 Parameters.

- The Assembler allows the nesting of Macros with Includes to a depth of 10.

- It is not allowed to nest Macros.

2.1.8.1. Macro definition

```
name   MACRO    Parameterlist
       :
       :
   ENDM
```

name: symbolic Macroname

Parameterlist: List of max. 5 Parameters, separated by commas.

2.1.8.2. Macro activation

```
      :
      :
      name  current Parameter list
      :
      :
```

name: symbolic Macroname

Current Parameter list: List of current Parameter (max. 5), separated by commas.

Note: 1) At Macro Call the number of Parameters must correspond to the number of Formal Parameters in the Macro definition; otherwise an Error Message occurs.
2) The current parameter mustn't be a reserved register name.

2.1.9. Statements

Every statement must be written on one program line, i.e. each statement in Source-Text must be ended by a <lf>.
A statement must conform to the instruction format mentioned in 2.1.3.
A statement contains 4 Fields:

- Label Field
- Opcode Field
- Operand Field
- Comment Field

The following combinations of the 4 Fields are allowed:

Label			
Label			Comment
Label	Opcode	Operand Field	
Label	Opcode	Operand Field	Comment
Label	Opcode		
Label	Opcode		Comment
	Opcode	Operand Field	
	Opcode	Operand Field	Comment
	Opcode		
	Opcode		Comment
			Comment

2.1.9.1. Labels

A label can be placed before every Assembler Statement.

A Label is generally forbidden before Assembler directives.

Exceptions:
- DW
- DB
- DBIT
- DS

The Label can be used as Reference for symbolic Jumps or for symbolic Subroutine Calls.

A Label is generally valid within the Module, in which it was defined. For expansion of it's validity, the Label can be given the attribute "PUBLIC" ("EXTRN" when referring to another Module).

If a Label appears before Directive DBIT, a switch will be made to the next byte (only RA87). In other cases it is forbidden.

The same rules have to apply to construction of a Label, as those for the construction of a Symbol. A Label is signified by it's ending: a colon.

Example:LABEL1: ...

2.1.9.2. Opcode Field

The Opcode Field may contain:

- A mnemonic Machine Instruction (As per Instruction Set)
- A Directive

In case the Machine Code or Directive Operand requires an operand at least one Blank as separation character between Opcode Field and Operand must be inserted.
A mnemonic Machine instruction will be translated by the Assembler into Machine Code. Directives will not be coded.

The Operand Field contains Operands (Displacements), as far as they are required by Opcode Field. Depending upon the machine instruction 0-2 displacements will be necessary.

Multiple Operand will be separated by commas.
Operands for Directives are described under paragraph 2.1.10.

Operand types:

Possible Operands are:

- numerical Constants
- Register-Names
- Symbols
- Expressions
- Location Counter

2.1.9.3.1. Numerical Constants

- binary Constants
- octal Constants
- decimal Constants
- hexadecimal Constants

Numerical Constants appear either directly in Displacement or will be defined indirectly via Constant Assignment EQU/SET.

```
Example:  1)  SUI   A, 07H        ; direct

          2)  BYTE   EQU   07H
                :
                :
              SUI  A, BYTE        ;indirect
```

2.1.9.3.2.

Register Names

a) general register
b) special register
c) register pairs
d) register pair address (indirect Addressing)
e) flags

a) General Registers

uP78C11 uP7811 uP7810 uP78C10 uP78C14	uP7809 uP7807	uPD78310 uPD78312	uPD78112	uPD78210	uPD78220 uPD78224
V A B C D E H L EAH EAL	V A B C D E H L EAH EAL	A X B C D E H L UPH UPL VPH VPL R0 : : R15	A X B C D E H L R0 : : R7	A X B C D E H L R0 : : R7	A X B C D E H L R0 : : R7

b) Special Registers

uP7811/10	uP7809/07	uP78C11 uP78C14 uP78C10	uPD78310 uPD78312	uPD78112	uPD78210	uPD78220/224
PA	PA	PA	Special function Registers (SFR) as Product-Description	Special function Registers (SFR) as Description	Special function Registers (SFR) as Description	Special function Registers (SFR) as Description
PB	PB	PB				
PC	PC	PC				
PD	PD	PD				
PF	PF	PF				
MKH	MKH	MKH				
MKL	MKL	MKL				
SMH	SMH	SMH				
SML	SML	SML				
EOM	EOM	EOM				
ETMM	ETMM	ETMM				
TMM	TMM	TMM				
*	PT	*				
MM	MM	MM				
MCC	MCC	MCC				
MA	MA	MA				
MB	MB	MB				
MC	MC	MC				
MF	MF	MF				
TXB	TXB	TXB				
RXB	RXB	RXB				
TM0	TM0	TM0				
TM1	TM1	TM1				
CR0	*	CR0				
CR1	*	CR1				
CR2	*	CR2				
CR3	*	CR3				
*	WDM	*				
*	MT	*				
ANM	*	ANM				
ETM0	ETM0	ETM0				
ETM1	ETM1	ETM1				
ECNT	ECNT	ECNT				
ECPT	*	ECPT				
*	ECPT0	*				
*	ECPT1	*				
	*	ZCM				

c) Register Pairs

uP78C11 uP78C14 uP78C10 uP7811/10	uP7809/07	Abbreviation	uPD78312 uPD78310	uPD78112	uPD78210	uPD78220 uPD78224
SP	SP	SP	AX	AX	AX	AX
BC	BC	B	BC	BC	BC	BC
DE	DE	D	DE	DE	DE	DE
HL	HL	H	HL	HL	HL	HL
EA	EA	EA	UP	RP0	RP0	RP0
VA	VA	V	VP	:	:	:
			RP0	:	:	:
			:	RP3	RP3	RP3
			:	SP	SP	SP
			RP7	PSW	PSW	PSW
			PSW			

d) Register Pair Address/ Mem-Address (RA310 / RA210)

uP78C11 uP7811/10 uP78C14 uP78C10	uP7807 uP7809	Abbreviation	uPD78310 uPD78312	uPD78112	uPD78210	uPD78220 uPD78224
-	-		[DE+]	[D]	[DE]	[DE]
BC	BC	B	[HL+]	[E]	[HL]	[HL]
DE	DE	D	[DE-]	[HL]	[DE+]	[DE+]
HL	HL	H	[HL-]	[E+]	[HL+]	[HL+]
DE+	DE+	D+	[DE]	word [B]	[DE-]	[DE-]
HL+	HL+	H+	[HL]	word [A]	[HL-]	[HL-]
DE-	DE-	D-	[VP]		[DE+byte]	[DE+byte]
HL-	HL-	H-	[UP]		[HL+byte]	[HL+byte]
HL+A	HL+A	H+A	[DE+A]		[SP+byte]	[SP+byte]
HL+B	HL+B	H+B	[HL+A]		word[A]	word[A]
HL+EA	HL+EA	H+EA	[DE+B]		word[B]	word[B]
HL+byte	HL+byte	H+byte	[HL+B]		word[DE]	word[DE]
DE++	DE++	D++	[VP+DE]		word[HL]	wprd[HL]
HL++	HL++	H++	[VP+HL]			
DE+byte	DE+byte	D+byte	[DE+byte]			
			[SP+byte]			
			[HL+byte]			
			[UP+byte]			
			[VP+byte]			
			word[A]			
			word[B]			
			word[DE]			
			word[HL]			

e) Flags

	uP7811	uP7809	uPD78310	uPD78112	uPD78210
	uP78C11	uP7807	uPD78312		uPD78220
	uP7810/11				uPD78224
	uP78C10				
	uP78C14				
PSW <	-	-			
	CY	CY	CY	CY	CY
	HC	HC			
	Z	Z			
	FNMI	FNMI			
	FT0	FT0			
	FT1	FT1			
	F1	F1			
	F2	F2			
	FE0	FE0			
	FE1	FE1			
	FEIN	FEIN			
	FAD	FAD			
	FSR	FSR			
	FST	FST			
	ER	ER			
	OV	OV			
	AN4	AN4			
	AN5	AN5			
	AN6	AN6			
	AN7	AN7			
	SB	SB			
	*	IEF2			

2.1.9.3.3. Handling of Symbols

If a Symbol appears in Operand Field, the value assigned to the Symbol (Address/EQU/SET/DB/DW/DBIT) will be used as value for the Operand. However,only Forward referencing is allowed in some cases.

```
Examples   1)   MARKE:  SBB   A, H
                        :
                        :
                        JMP   MARKE    ; The Symbol MARKE
                                       ; will be substi-
                                       ; tuted via the
                                       ; corresponding
                                       ; Address

           2)           JMP   MARKE1   ; Forward reference
                        :
                        :
                MARKE1: SBB   A, H

           3)   MARKE2  EQU   0F5H
                        :
                MARKE3: DB
                        :
                        SUI   A,MARKE2
                        MOV   MARKE3,A
```

2.1.9.3.4. Expressions

- Operands:

 -- Symbols (absolute, relocatable)
 -- $-Symbol (Current Location-Counter)
 -- Integer-Constants
 -- ASCII Char-Constants (1 Character enclosed by apostrophes)

- Operators:

 -- arithmetic Operators:

 + - * / MOD

 -- logical Operators:

 NOT AND OR XOR

 -- Compare Operators

 EQ NE GT GE LT LE

 -- Shift-Operators:

 SHR SHL

 -- Word-Operators:

 HIGH LOW

-- unary arithmetic Operators:

+ -

-- Bit-Operator:

- Priority Rules

-- Upto Version V1.5 (RA87)

The individual Operators of an Expression will be evaluated from left to right; i.e.in the sequence of their appearance.

The use of brackets is not allowed.
Unary Operators always refer to the total Expression.

-- From Version V1.6 (RA87) and V1.0 (RA310) and V1.0 (RA110)

The individual Operators of an Expression will be calculated as follows:

```
HTermF:   <hterm> --> <gterm>
                      <gterm>  OR <hterm>
                      <gterm> XOR <hterm>
GTermF:   <gterm> --> <fterm>
                      <fterm> AND <gterm>
FTermF:   <fterm> --> <eterm>
                      NOT <eterm>
ETermF:   <eterm> --> <dterm>
                      <dterm>  NE <dterm>
                      <dterm>  EQ <dterm>
                      <dterm>  GE <dterm>
                      <dterm>  GT <dterm>
                      <dterm>  LT <dterm>
                      <dterm>  LE <dterm>
DTermF:   <dterm> --> <cterm>
                      <cterm>   + <dterm>
                      <cterm>   - <dterm>
CTermF:   <cterm> --> <bterm>
                      <bterm>   * <cterm>
                      <bterm>   / <cterm>
                      <bterm> MOD <cterm>
                      <bterm> SHR <cterm>
                      <bterm> SHL <cterm>
BTermF:   <bterm> --> <aterm>
                            + <aterm>
                            - <aterm>
                         HIGH <aterm>
                          LOW <aterm>
HTermF:   <aterm> --> number
                      ident
                      string
                      $
                      ( <hterm> )
```

- Value of an Expression:

  The Components of an Expression will be handled as 16-Bit values. The Result of a calculation of an Expression will also be a 16- Bit value.

- Division by Zero:

  By Division or Modulo-Operation the 2.Operand will be checked for non-zero Status. If, however, 2.Operand is zero, an Error Message occurs.

- Arithmetic Operators:

+	Addition
-	Subtraction (binary, unary)
*	Multiplication
/	Division
MOD	modulo-Operation

- Logical Operators:

NOT	Negation (unary)
AND	logical Product
OR	logical Sum
XOR	Exclusive Or

```
Examples:   NOT  1234H --> 0001001000110100
            --> 1110110111001011 --> EDCBH

            1234H AND 5678H
            --> 0001001000110100 AND 0101011001111000
            --> 0001001000110000 --> 1230H

            1234H OR 5678H
            --> 0001001000110100 OR 0101011001111000
            --> 0101011001111100 --> 567CH

           1234H XOR 5678H
           --> 0001001000110100 OR 0101011001111000
           --> 0100010001001100 --> 444CH
```

- Compare Operators:

  Compare Operators are used for the comparison of 2 values. If the comparison is positive, then the result is given the value 0FFH, in other cases the result is given value 00H.

EQ	Equal
NE	Not Equal
GT	Greater Than
GE	Greater Equal
LT	Less Than
LE	Less Equal

- Shift-Operators:

SHR	Shift Right
SHL	Shift Left

```
Format:   a SHR b
          a SHL b

          'a' will be shifted 'b' places to the  right
          (left).
          Zeroes will fill resulting "free" positions.

Example:   01AFH SHR 5 --> 0000000110101111 SHR 5
           0000000000001101 --> 000DH
```

- Word-Operators:

```
HIGH   (unary)
LOW    (unary)
```

The Word-Operator delivers the higher (lower) value Byte of a Word.

```
Example:   HIGH  0FE03H --> 0FEH
           LOW   0FE03H --> 03H
```

HIGH and LOW can also be used on Expressions which contain only one label or only one external Symbol.

- unary arithmetic Operators:

```
+
-
```

The "+" Operator has no meaning for unary Operations.

Example: +5 --> 5

The "-" Operator delivers the negative value of an Operand in 2nd Complement.

- Bit-Operator (uPD 7809 only):

The Bit-Operator delivers a Bit Address for Adressing of Hardware-Registers:

```
PA
PB
PC
PD
PF
MKH
MKL
SMH
EOM
TMM
PT
```

The following Table gives Information about the fixed Hardware-Bit-Addresses:

7	6	5	4	3	2	1	0	Hardware.Symbol
87	86	85	84	83	82	81	80	PA
8F	8E	8D	8C	8B	8A	89	88	PB
97	96	95	94	93	92	91	90	PC
9F	9E	9D	9C	9B	9A	99	98	PD
AF	AE	AD	AC	AB	AA	A9	A8	PF
B7	B6	B5	B4	B3	B2	B1	B0	MKH
BF	BE	BD	BC	BB	BA	B9	B8	MKL
CF	CE	CD	CC	CB	CA	C9	C8	SMH
DF	DE	DD	DC	DB	DA	D9	D8	EOM
EF	EE	ED	EC	EB	EA	E9	E8	TMM
F7	F6	F5	F4	F3	F2	F1	F0	PT

Example: PA.4 --> Bit 4 of Port A --> 84H

- Limitations

The Operands of an Expression can either be Constants or Addresses. However, not all combinations of Operands are allowed at the calculation of an Expression.

The following Table gives Information about allowed Combinations:

Operator	X const Y const	X const Y address	X address Y const	X address Y address
X + Y	C	A	A	I
X - Y	C	I	A	C
X * Y	C	I	I	I
X / Y	C	I	I	I
X MOD Y	C	I	I	I
X SHL Y	C	I	I	I
X SHR Y	C	I	I	I
X EQ Y	C	I	I	C
X LT Y	C	I	I	C
X LE Y	C	I	I	C
X GT Y	C	I	I	C
X GE Y	C	I	I	C
X NE Y	C	I	I	C
X AND Y	C	I	I	I
X OR Y	C	I	I	I
X XOR Y	C	I	I	I
NOT X	C	-	I	-
HIGH X	C	-	I	-
LOW X	C	-	I	-
unary + X	C	-	A	-
unary - X	C	-	I	-

A = address
C = const
I = illegal

A maximum of one external Symbol may appear in an Expression but only in connection with Operator '+ '.

```
e.g.extrn ext, ext1
    dw ext + 3
    dw ext - 3      forbidden
    dw ext + ext1   forbidden
```

but:

```
    dw ext +(-3)    allowed
```

2.1.9.3.5. Location Counter

The Location Counter is represented by "$" Character. The Location Counter represents the value of current Address Pointer.

For multiple byte Instructions, the 1.Byte is relevant.

```
Example:    100   Loop:  MOV  PA, A
            102   JR $-2
```

The Location Counter $ in Expression ($-2) represents the value 102. JR $-2 is, therefore, the same in meaning as JR LOOP.

2.1.9.4. Comment Field

Comments will be limited via:

; <lf>

All Text, which occur in a Source line between ";" and "<lf>", will be handled as Comments.

Comments will not be considered by Assembler, but will be transferred to Assembler-Listfile.

Every printable character may be used in Comments. Comments can occur at any desired position in the Source text.

2.1.10. RA87/ RA310 /RA110 /RA210 Control Commands

Control Commands can influence the Assembler Run in a User specific way. The most Control Commands have Default values; these are valid when the Command is not explicitly executed.

Assembler Commands are optional. They must be arranged in Control Command lines or in Assembler Call.

2.1.10.1. Control Command Line

Syntax: $Control Command[Control Command] ... [;<Comments>]

Control Command
- Primary controls as well as general controls may be entered.
- multiple Control words, as well as a possible comment, must be separated via one FF, HT, CR or one Blank.
- Parameters attached to Control words are enclosed within brackets and can be separated via one Blank,FF,HT,CR characters.
- Entered Parameters always make reference to the last Control Command.
- The definition of Control Commands can occur in the Call itself or in a Control Command line in Primary Program.
- The content of a Control Command line is, depending on position, severely limited (See following paragraphs).
- The '$' should be written in the first column of the SRC Code.
- The respective control word shouldn't be separated form '$', the possible operand should be separated by one blank or tab.

Hint: Please also refer to the Call Convention in Chapter3.

2.1.10.2. Primary-controls

Primary Control assignments must be defined in a Call command or Control command line, which must lie before the first non command line of the Assembler Primary Program. Primary Control assignments may only occur once within a primary assembler module. Double entry of Primary Controls results in an overwriting of the last Primary control action. Moreover, Primary controls defined in a Call command have higher priority.

2.1.10.2.1. DEBUG/ NODEBUG

Syntax: DEBUG / DG
NODEBUG / NODG

Default value: NODEBUG

The control command DEBUG states that the Object Module shall contain the following:

- Name and length of each Symbol
- Indication to which Segment a Symbol is defined.
- Description about the area of validity for each Symbol.
- Address of each Symbol, as far as this is known at conversion time.

This information can be partly used for symbolic debugging of the program.

The Control command NODEBUG states, that no DEBUG information will be put in an Object Module.

2.1.10.2.2. OBJECT/ NOOBJECT

Syntax: OBJECT (filename) / OB (filename)
NOOBJECT / NOOB

Default value: OBJECT (primary filename.REL)

- The Control command OBJECT states, that the conversion will produce an Object Module.
- (filename) states, which Disc-File will contain the Object Module.
- If there is no Control command given, then the Object Module will be saved on the same Disc drive which is used for the Primary program input. The Object-code-file has the same name as the Primary Program file but with extension REL.
- OBJECT changes also the filename of the Listfile.

Example: $OBJECT (OTHER.OBJ)
A file other.obj is created for Objectfile and a file other.prn is created for Listfile.

The Control command NOOBJECT states, that no Object Module will be produced.

2.1.10.2.3. PAGING/ NOPAGING

Syntax: PAGING / PI
NOPAGING / NOPI

Default value: PAGING

- The Control command PAGING states that the LISTING printed output will be divided into pages.
- Every page contains a Header and a page number.
- If required the user defines a title, a date or on the 2nd line a Sub-title.

NOPAGING prevents action of the following Control commands:

- PAGELENGTH
- TITLE
- SUBTITLE
- EJECT

2.1.10.2.4. PAGELENGTH

Syntax: PAGELENGTH (length) / PL (length)

Default value: PAGELENGTH (60)

The Parameter (length) is a non zero, unsigned whole decimal number, which indicates the maximum number of lines per page of a listing. This number also contains the lines used for the listing header.

The minimum values for (length) is 13, the maximum value is 255.

2.1.10.2.5. PAGEWIDTH

Syntax: PAGEWIDTH (width) / PW (width)

Default value: PAGEWIDTH (132)

The Parameter (width) is a non zero, unsigned whole decimal number which gives the maximum line width (in characters including control characters LF) per page of a Listing.

The minimum value for (width) is 72, the maximum value is 132.

2.1.10.2.6. PRINT/ NOPRINT

Syntax: PRINT (filename) / PR (filename)
NOPRINT / NOPR

Default value: PRINT (Primary filename.PRN)

The Control command PRINT/NOPRINT states, whether a Listing will be output or not.
(filename) states on which Disc drive the list file will be saved or which equipment will be used as List output device. Any allowed output device may be used.
In case the PRINT/NOPRINT control command is not used, the default logical device for the List file will be the same as that for the Primary program input.
The output file has the same name as the Primary program but with the extension PRN.

Example: $PRINT (LST:)

This Command produces in a Listing on the Matrix-Printer.

The Control command NOPRINT states, that no printout will occur, even if the contrary has been selected by another Control Command e.g.LIST.
The OBJECT-Control command can influence the filename of the Listfile.

2.1.10.2.7. TITLE

Syntax: TITLE ('title') / TT ('title')

Default value: no Title

The Parameter ('titel') is a string of printable ASCII characters enclosed within brackets.

The title will appear in the first line of each page, if the title string is too long, it will be truncated on the right side.

The maximum allowed length for ('title') is 60 characters. A smaller (PAGEWIDTH) will naturally reduce this length.

Example: $TITLE ('TEST PROGRAM#4')

Until the occurence of TITLE Control command, the Default Title line or TITLE line from the Call command will be considered at the start of each page.

2.1.10.2.8. WORKFILE

Syntax: WORKFILE (path) / WF (path)

(path) means Floppy disc drive and path.

During the Assembly process workfiles are required, which are released after completion of conversion. If Control command WORKFILE is not entered, then all necessary Workfiles will be written to Default Path.

In the following example Floppy drive A is selected.

```
$WORKFILE (A:)
```

2.1.10.2.9. XREF/ NOXREF

Syntax: XREF / XR
NOXREF / NOXR

Default value: NOXREF

The Control command XREF states, that a Cross-reference-list, based on the line numbers within the Primary program, will be produced and appended at the end of the List file.

The Control command NOXREF states that no Cross-reference-List will be produced.

If NOPRINT has been set, then XREF is cancelled.

2.1.10.2.10. PROCESSOR

Syntax: PROCESSOR (type) / PC (type)

No default value is set; the PROCESSOR (type) must be specified.

PROCESSOR (type) defines the type of CPU, which will be taken into account by the Assembler.

"type" selects the following CPU's:

- n = 07 uPD 7807
- n = 09 uPD 7809
- n = 10 uPD 7810
- n = C10 uPD 78C10
- n = 11 uPD 7811
- n = C11 uPD 78C11
- n = C14 uPD 78C14
- n = 310 uPD 78310
- n = 312 uPD 78312
- n = 112 uPD 78112
- n = 210 uPD 78210
- n = 220 uPD 78220
- n = 224 uPD 78224

```
example:  $PROCESSOR (11)    ; The Assembler considers
                             ; Processor uPD 7811 in RA87
```

2.1.10.2.11. OPTIMIZE

Syntax: OPTIMIZE (n) / OP (n)

Default value: OPTIMIZE (0)

OPTIMIZE enables the optimization of executable code i.e. jump instructions are adapted to the jump destinations.

RA87:
OPTIMIZE tries to convert (where ever possible) all JMP/JRE instructions into 1 or 2 byte instructions.

RA310 / RA110 / RA210 :
OPTIMIZE tries to optimize the BR instruction without Dollar-sign.

n defines the maximum number of optimization runs.

Where n < 256

The optimization process will be stopped, if 2 consecutive optimization runs do not bring any code saving.

Optimize influences the time required for Assembling.

2.1.10.2.12. DATE

Syntax: DATE ('date')/ DA ('date')

Default value: No Date (Blanks)

The Parameter ('date') can be any string of maximum 8 printable characters. The date appears in the first line of all pages of a Listing under the heading ('date'). The information within ('date') is used as a Parameter. If no Control command DATE is given, then the Date-field is blanked.

2.1.10.2.13. ERRORPRINT

Syntax: ERRPRN (filename)/ EP (filename)

Default value: No Errorprint

The Control command ERRPRN produces, that error lines, Error Messages and following Source lines from list file are written to a file specified by ERRPRN (filename).

(filename) gives the name of the file and logical device, to which the error list will be copied.

Example: ERRPRN (CON:)

The Errors are listed on the Console.

2.1.10.2.14. TAB

Syntax: TAB (n) / TB (n)

Default value: TAB (8)

TAB is used to set the Tabulator distance.

Setting:		
	n = 0	No Tabulator.(Default)
	1<=n<=8	possible setting
	n > 8	not allowed

Example: TAB (5)

Tabulator jumps to positions 0, 5, 10, 15 ...

2.1.10.3. General Controls

General control commands can appear in a Call command or in a Control command line within the Primary program. General control commands can be changed within a Module.

General controls, that appear in the source file before the first non-command line can be overwritten by general controls from the Call command.

2.1.10.3.1. EJECT

Syntax: EJECT / EJ

With the control command EJECT, the printing of a page is interrupted, a new page starts. The line with the control command EJECT will be printed on the new page after the title line. Several EJECT Statements in a line, produce only a page feed. EJECT in Call command leads to an abort of Assembler run operation.

If NOPRINT, NOLIST or NOPAGING are active, control EJECT is ignored.

2.1.10.3.2. INCLUDE

Syntax: INCLUDE (filename) / IC (filename)

The Parameter (filename) is a file address stating which file on a Floppy disc shall be included within the Primary program text.

Example: $INCLUDE (SYSLIB.SRC)

With the control command INCLUDE, primary program lines from the named file are binded subsequently. The input from the named file continues until of End of File is

recognized. Now the input of further Primary program lines may continue from the original file.

An INCLUDE File can itself contain an INCLUDE Control command. N.B. INCLUDE Files can only be nested to a maximum depth of 1.

This Control command may only appear in a Control command line and may not be used in a Call command.

2.1.10.3.3. LIST/ NOLIST

Syntax: LIST / LI
NOLIST / NOLI

Default value: LIST

The Control command LIST states that the listing of a Primary program shall begin, as soon as the next Primary program line is read.

The Control Command NOLIST inhibits the listing of Primary program, until the possible appearance of LIST Control command. If NOLIST is given within Call command, then the listing will be inhibited from the first non-command line.

With the control command LIST, all program lines (of the primary program file or of an include file) inclusive the control lines are listed.

The Control Command NOLIST lists only the input ,lines, which convering error messages.
NOLIST does not inhibit the concluding information of a Listfile or a possible Cross reference list output. NOPRINT overrules LIST.

2.1.10.3.4. SUBTITLE

Syntax: SUBTITLE ('title') / ST ('title')

Default value: No Subtitle

The Parameter ('title') is a string of printable ASCII-Characters enclosed in brackets.
The string characters appear in the second title line of every page of a Listing. If the subtitle line is too long, it will be truncated at the right side.
The maximum length for ('title') is 60 Characters. A smaller page width (PAGEWIDTH) reduces this value.

Example: $SUBTITLE ('TEST PROGRAM#4')

2.1.10.3.5. SET

Syntax: SET (switch[:switch]...)

switch:
- A Variable conforming to the rules for constructing a Symbol but without underline.
- Switch exists only on Assembler level. If several switches are entered, they must be separated by colons.

The SET Command sets the given Symbol to TRUE (=0FFH)

```
Example:    $SET  (ALPHA:BETA)  ; ALPHA and BETA will be
                                ; set TRUE.
```

Note:
- A maximum of 5 switches are allowed per SET directive.
- SET-Directives may only set a maximum of 5 switches.
- RESET command enables resetting of a switch.
- Refer to IF, ELSE, ELSEIF, ENDIF;
- SET can appear anywhere in Program.

2.1.10.3.6. RESET

Syntax: RESET (switch[:switch]...)

switch:
- A Variable conforming to the rules for constructing a Symbol but without underline.
- A switch exists only on Assembler level. If several switches are given, they must be separated by colons.

RESET directive sets the entered Symbols to FALSE (=00H)

```
Example:    $RESET  (ALPHA:BETA)  ; ALPHA and BETA will
                                  ; be set to FALSE.
```

Note:
- A maximum of 5 switches per RESET directive is allowed.
- RESET-Directives may only reset a maximum of 5 switches.
- SET can set the switch TRUE again.
- Refer to IF, ELSE, ELSEIF, ENDIF;
- RESET can occur anywhere in Program.

2.1.10.3.7. IF/ ELSE/ ELSEIF/ ENDIF

```
Syntax:  IF (switch)
         sourcetext
         [ELSEIF (switch)
         sourcetext
         [ELSEIF (switch)
         sourcetext] ...]
         [ELSE
         sourcetext]
         ENDIF
```

switch: - A Variable conforming to the rules for constructing a Symbol but without underline.
- A switch exists only at Assembler level. If several switches are entered, they must be separated by colons.

sourcetext: Text, which can be processed by Assembler.

By using IF,ELSE,ELSEIF,ENDIF, it is possible to assemble parts of the program under the following conditions. The code will only be assembled when the switches condition under test is found to be true.
When using a combination of IF and ELSEIF,the branch that will be executed is the one, where the switch is TRUE.
If several switches in different branches are set TRUE, then the first branch will be sequentially executed.
If all the switches are at FALSE, then the ELSE branch will be executed, assuming that the ELSE branch is there.

Example:

```
1)  $IF (ALPHA)          ; Switch condition: If ALPHA
    sourcetext           ; TRUE then the Sourcetext
                         ; will be considered.

    $ENDIF               ; IF-Clause will be returned
                         ; to Reset condition.
2)  $IF (ALPHA:BETA)     ; This branch will be executed
    text 1               ; If ALPHA or BETA are in
                         ; the TRUE state.
    $ELSE                ; otherwise the ELSE branch
    text 2               ; will be executed.
    $ENDIF


3)  $IF (ALPHA)          ; This branch will be
    text 1               ; executed when ALPHA is TRUE,
                         ; then jump to first Statement
                         ; after $ENDIF.
    $ELSEIF (BETA)       ; This branch will be
    text 2               ; executed when BETA is TRUE,
                         ; then jump to first statement
                         ; after $ENDIF.

    $ELSEIF (DELTA)      ; This branch will be
    text 3               ; executed when DELTA is TRUE,
                         ; then jump to first statement
                         ; after $ENDIF.
    $ELSE                ; This branch will be
    text 4               ; executed when ALPHA,BETA and
                         ; DELTA are all FALSE.
    $ENDIF
```

Note: - Nesting of this structure is not possible.
- Compare SET and RESET Control commands.
- Structures of this type are allowed only after the first non-Command line.

2.1.10.4. Priorities

Slave \ Master	NOOBJECT	NOPAGING	NOPRINT	NOLIST
DEBUG	X			
OPTIMIZE	X			
PAGELENGTH		X	X	
PAGEWIDTH			X	
TITLE		X	X	
SUBTITLE		X	X	
EJECT		X	X	X
LIST			X	
XREF			X	
TAB			X	X

Interpret as:e.g.
NOOBJECT overrules -DEBUG
-OPTIMIZE

2.1.11. Assignment directives

2.1.11.1. Segment Directives

Remark by using of named segments:
If a module in a segment is specified by a name, the name must be consistent by repeated references onto this segment

```
MODUL 1:
B:    C1      CSEG
        .
        .
        .
      C1      CSEG
                      ;right

MODUL 2:
      C2      CSEG
        .
        .
        .
              CSEG
                      ;ERROR

MODUL 3:
      C3      CSEG
        .
        .
        .
      C4      CSEG
                      ;ERROR
```

2.1.11.1.1. DSEG/ ENDS-Directive

```
Syntax:   [name]   DSEG
          :
          :
          [ENDS]
```

DSEG: defines a Data Segment.

The directive DSEG assigns the Assembler to assemble the following Data relative to the start of the Segment. The Address level begins at 0. The Base Address can be changed by use of ORG assignment.

```
Example:DSEG                        ; defines a Data Segment

      Control value 1:  DB
      Control value 2:  DB

    Table:              DB (100)  ; A Table of 100 Bytes is
                                  ; reserved.

                        ENDS
```

Note:
- The Data Segment contains only Data declarations.
- The Data Segment may not contain Machine instructions.
- The defined Data will be stored in RAM and are capable of being changed.
- If DSEG appears several times in Program, then the address level will be incremented from the address reached in the last DSEG. A maximum of 1 Data-Segment per Source Module is allowed.
- Labels are not allowed before the DSEG-Directive.

2.1.11.1.2. VSEG/ ENDS-Directive (RA87 only)

```
Syntax:   [name]   VSEG   [(address)]
            :
            :
            :
          [ENDS]
```

address:
- Entry of a 8-Bit-Address (MSB)
- 8-Bit-Address will be given to the Linker as a 16-Bit-Address.
- If Address entry is missing, then a relocatable V-Segment will be produced.

- The user must load the 8-Bit-Address in V-Reg. If at Program-Run-time, the V-Reg has a different content, then false access to Data will occur.

The VSEG/ENDS-Directive defines an Address area of 256 Bytes relative to a Base Address in V-Reg.

Note: - Upto 5 absolute V-Segments and additionally one relocatable V-Segment can be defined.

- In the V-Seg only the following address influencing Directives are allowed.

  DBIT [nur uPD 7809]
  DB
  DW

- ORG Assignments are not allowed.

- Sequence: Firstly Bit size using DBIT [only uPD 7809], then Byte-/ Word size can be defined.

- Bit Arrays are not allowed.

- Addressing: - Bits will be addressed in sequence, from 0-127(16 Bytes) [uPD 7809 only].
- If no Bits are defined, then a full 256 Byte size within a V-Segment can be defined (--> Memory Re-organisation).
- If the Address level oversteps the number of 256 bytes, an Error Message occurs.
- V-Segments lying one upon another procedure a Warning when using Locator.
- Double definition of a memory location of a V-Segment Bit is not allowed.

- V-Segment-Definitions: - May only occur in instructions related to the V-Register (Operands: wa.bit)

  - will be signified as bits in the Symbol-List. (principally handled like EQU; however note the max. size of 8 bits). Therefore, they cannot be tested using ICE Systems.

  - Labels are not allowed before VSEG-Directive.

2.1.11.1.3. CSEG/ ENDS-Directive

```
Syntax:   [name]   CSEG
           :
           :
          [ENDS]
```

CSEG: defines a Code Segment

The Data and Code lines defined in this Segment will be calculated relative to the Segment start. The Address level begins at 0 but can be changed via the ORG-assignment.

```
Example:    CSEG                ; A Code Segment will be
                                ; defined.
             :
             :
            PUSH     A

     LOOP:  MOV      A,SYM
             :
             :
     SYM:    DB      5
            ENDS
```

Note:
- The defined Data cannot be re-initialized.
- This is the Segment containing executable Code.
- This Segment will be stored in ROM.
- If CSEG appears several times in Program, then the Address level will be incremented from the address reached in the last CSEG. A maximum of 1 Code Segment per Source Module is produced.
- Labels before CSEG-Directive are not allowed.

2.1.11.1.3.1. CSEG FIXED/ ENDS Directive

```
Syntax:   [name]   CSEG FIXED
          :
          :
          ENDS
```

CSEG FIXED: defines a Code-Segment with max. Size 800H.

The Data and Code lines defined in this Segment will be defined relative to the Segment start.

The Address level starts at 0 but can be changed via ORG assignment.

```
Example     FIXCOD   CSEG FIXED
              :
              :
            ENDS
```

Note:
- If the Address level oversteps the size 800H, an Error Message occurs.
- Otherwise see CSEG-Directive
- The exact position of CSEG FIXED within the Memory can be found by referring to the respective Product Description.
- All CSEG FIXED segments must be located to the memory area of 800H - FFFH finally (by LOC87, LC310/210/110).

2.1.11.1.3.2. CSEG CALLT0 (RA310 / RA110)
CSEG CALLT1 (RA310)
CSEG CALLT (RA87 / RA210)

Syntax:
```
[name]   CSEG  CALLT0  (RA310 / RA110 only)
[name]   CSEG  CALLT1  ( RA310 only)
[name]   CSEG  CALLT   ( RA87 / RA210 only)
```

- The Data and Code lines defined in above Segments will be calculated relative to the Segment start.
- The Address level begins at 0 but can be changed via ORG assignment.
- The segment CSEG CALLT defined in RA87 / RA210, corresponds to the segment CSEG CALLT0 defined in RA310 / RA110. The Segment CSEG CALLT1 is only valid for RA310 only.
- The Segments have a size of 040H respectively. If Address level oversteps 040H, an Error Message occurs.
- Otherwise see CSEG-Directive
- The exact position of the Segments within the Memory can be found in the respective Product Description.

2.1.11.2. Directives for Allocation of Constants

2.1.11.2.1. EQU Assignment

Syntax: name EQU well-expression

name: <name> signifies the Constant.
<name> may not be ended by a colon.
The Symbol defined by <name> may not be used as a label again.

well-expression:
- See paragraph 2.1.9.3.4. with following limitations.
- No external Symbols are allowed.
- Labels or Names in <expression> must have been defined before (no forward referencing).
- Bit values of the Form MKL.5, among

others, are allowed expressions with EQU assignment but only for uPD7809.

The assignment EQU allocates the value of the expression in Operand Field to the name given in Label Field.

```
Example:   ASYM    EQU   33D   ;  ASYM <--- 33
           CSYM    EQU   A+45D ;  CSYM <--- 78
           DSYM    EQU   PA.2  ;  DSYM <--- 082H
                                     as Bit value
                                              (PA.2)
```

Note:
- <name> may only be allocated a value with EQU once per Module
- EQU may appear anywhere in a program but must lie between assignments NAME and END.
- A Symbol, which has a bit value allocated to it, is only used as a bit-symbol.
- The symbol value cannot redefined with SET-Directive; an error occurs.

There is still another format of EQU-Assignment. Please refer to Chapter 2.1.11.7.9.

2.1.11.2.2. SET Assignment

Syntax: name SET expression

name: <name> signifies the Constant.
<name> may not be ended with a colon.
The Symbol defined by <name> may not be used as a label again.
<name> may not appear as Public or External.

expression: See paragraph 2.1.9.3.4. with following limitations.
- No external Symbols are allowed.
- labels and names (in <expression>) must be defined before (no forward references)

The assignment SET allocates the value of expression in the Operand Field.

```
Example:    AB   SET   33    ;  AB <-- 33
            CD   SET   AB+45 ;  CD <-- 78
            AB   SET    5    ;  AB <--  5
```

Note:
- <name> may be allocated a value by SET several times within a Module.
- The SET assignment, in contrast to the Control directive SET, does not lie in a Control command line.
- SET may appear anywhere in Program (between assignments NAME and END).
- SET in connection with Bit values (as for EQU) is not allowed.
- The symbol value cannot redefined with

EQU-Directive; an error occurs.

2.1.11.3. Directives for Memory reservation

2.1.11.3.1. Bit memory reservation

2.1.11.3.1.1 DBIT Assignment for RA87 (uPD7809 only)

Syntax: [name] DBIT [init]

name: Defines a logical Address, where the Bit resides

init: Is a constant value with Range 0...1;

The Assignment DBIT reserves sequential memory locations for the defined Data, starting at the present Address level.

If a Label occurs between two DBIT assignments, the Bit level will be switched forward to the next Byte limit. If a Bit is not initialized, it is given Default value of 0. After the reckoning of all DBIT assignments, the code produced by the initialized bits will be transferred to the Listing.

DBIT assignments may only appear directly after VSEG-Directive.

In a VSEG-Segment a max. of 128 -Bits may be reserved using DBIT assignment; i.e.The address lies between 0 and 127.

```
Example:   FLAG  DBIT     ; reserves a Bit that can be
                          ; referred to by Symbol FLAG.
           FLAG1 DBIT 1   ; reserves a Bit, set to value
                          ; 1 that can be referred to by
                          ; Symbol FLAG1.
```

2.1.11.3.1.2 DBIT Assignment for RA310 / RA110 / RA210

Syntax: [name] DBIT

name: defines a logical address in saddr-area

- Memory can only define bit by bit
- DBIT may only used in BSEG
- It is not allowed to initialize bits
- The logical address is a bit number beginning at 100H for RA310 / RA210 and at 200H for RA110
- The highest bit number is 900H
- It is not allowed to define labels between DBIT-Directives

Note:
- DBIT can only be used in V-Segment, B-Segment.
- No other statements (apart from comments and empty lines) may appear between different DBIT assignments

2.1.11.3.2. DB Assignment

Syntax: [name:] DB [(index)] [init[,init]...]

name: Defines a logical Address, where the Byte resides.

(index): defines the number of Bytes to be reserved. The Range lies between 1...65535.

init: Is a constant Number in Byte size, either Hex-, Decimal-, Octal- or Binary format, as well as String-Format.
For initialization a max. of (index) values can be given, as long as they fit in the Program line.
The individual init values must be separated by commas.

The Assignment DB stores (reserves Memory for) the fixed Data in directly sequential memory locations, starting with the current value of Address.

```
Example:TABLE   : DB (10)     ; Space for a 1-dimensional
                              ; field will be reserved.
                              ; The logical start ad-
                              ; dress will be allocated
                              ; the Symbol TABLE.

        TAB1 : DB   'ABCDE'   ; 5 Bytes will be in-
                              ; itialized with String
                              ; 'ABCDE'.
```

Note:
- The DB assignment can stand at any desired position.
- It is efficient to initialize Data defined in Code Segment.
- Reserved Memory plus Program Code must be less than 64kByte.
- The length entry is not required for initialized Data. If, however, the length is given, it will be checked.

2.1.11.3.3. DW Assignment

Syntax: [name:] DW[(index)] [init[,init]...]

name: defines a logical Address, where the Word resides.

(index): defines the number of Words to be reserved. The Range lies between 1...32767.

init: Is a constant Number of Word size, either Hex-, Decimal-, Octal- or Binary format. The individual Init values must be separated by commas. A maximum of upto (index) values can be initialized, as long as they fit in the Program line.

The assignment DW stores (reserves Memory for) the fixed Data in directly sequential Memory locations, starting at the current value of Address.

```
Example: TABLE : DW (10)    ; Space for a 1-dimensional
                            ; Field will be reserved, the
                            ; logical Start Address will
                            ; be allocated Symbol TABLE.

         LOOP  : DW  10H    ; A value with content 10Hex
                            ; will be reserved.
```

Note:
- The DW assignment can stand at any desired place.
- It is efficient to initialize the Data defined in Code Segment.
- Memory reservation plus Program Code must be smaller than 64kByte.
- The length entry is not required for initialized Data. If however, length is given, then it will be checked.

2.1.11.3.4. DS Assignment

Syntax: [name:] DS Expression

name: defines a logical Address, under which a memory area can be accessed.

Expression: arithmetic expression (see 2.1.9.3.4.) with following restrictions:

- Labels and Names, which appear in <Expression>, must have been already defined (no forward referencing).
- External Symbols are not allowed in <Expression>.
- The value of <Expression> must be 'absolute'

The Assignment DS reserves a Data field in memory. The size of the Field will be represented by <Expression>.

```
Example:  DSEG
          AB      EQU   15
          FELD    DS    AB+10 ; A Field of 25 Bytes will
                              ; be reserved.
```

Note:
- Data will be reserved in units of 1 Byte.
- The data field cannot be initialized by DS Assignment.
- No Op-code will be generated.

2.1.11.4. Directives for Describing End of Program

2.1.11.4.1. END Assignment

Syntax: END

The END assignment indicates to the end of a Program.

```
Example:      DSEG                    ; Data segment will
                                      ; be arranged.

          EINS :  DB
          ZWEI :  DW

              CSEG                    ; Code segment will be
                                      ; arranged.

         START:  MOV  A, EINS
                 MOV  H, ZWEI
                 ENDS
                 END                  ; Signifies the end of
                                      ; Program.
```

Note:
- The END assignment may only appear once. It must be the last primary program directive.
- The Source-Programm will be translated until the first END assignment appears.

2.1.11.5. Directives for Link Run

2.1.11.5.1. NAME Assignment

Syntax: NAME name

name: Must conform to the rules for Definition of Symbols.

The assignment NAME allocates a name to the Object Module produced during Assembling.

```
Example:    $PAGELENGTH (50)
            NAME        Main        ; The Object Module is
             :                      ; given the name "MAIN"
             :
            END
```

Note:
- NAME must appear in Primary Program before the first entry of Data or Instruction, it can be preceeded by comment or control command lines.
- NAME may only occur once per Module.

2.1.11.5.2. PUBLIC Assignment

Syntax: PUBLIC name[,name]...

name: gives Symbol name;
must conform to the rules for definition of Symbols.
Several names must be seperated by commas.

The PUBLIC assignment enables access by other Programs to all Symbols stated in the Operand Field.

```
Example:   NAME    MYPROG                ; ALPHA, BETA and
           PUBLIC  ALPHA, BETA, DELTA    ; DELTA are en-
                                         ; abled for access
                                         ; by other Programs.

            DSEG
    ALPHA : DB
    BETA  : DB
            CSEG

   DELTA:   MOV  A, ALPHA
             :
             :
            ENDS
            END
```

Note:
- Each name in the list may be declared only once as PUBLIC.
- The given name must be the name of a Data Element or the Label of an instruction.
- The sum of PUBLIC- and EXTRN Symbols may not exceed 255.
- It is allowed to use bits in saddr-area (0FE20H / 0FE40H - 0FF1FH) in connection with PUBLIC (only RA310 / RA110 / RA210).
- The PUBLIC-Declaration of a Symbol must conform to the rules for definition of Symbols.
- The PUBLIC-Directive may only appear in the Modul Header.

2.1.11.5.3. EXTRN/EXTBIT assignment

```
Syntax:   EXTRN   name[,name]...
          EXTBIT  name[,name]...     only RA87 (uPD7809)/
                                     RA310/RA110/RA210
```

name: gives the Symbolname;
must conform to the rules for definition of Symbols. Several names must be separated by commas.

The assignment EXTRN/EXTBIT delivers to the Assembler a List of Symbols, which are called in this Program, but which are defined in an other Program.

```
Example:    NAME   MYPROG
            EXTRN  ALPHA, BETA    ; ALPHA and BETA will be
              :                   ; signified, that they
              :                   ; are not declared in
              CSEG                ; this Program Module.

    DELTA:  MOV    A, ALPHA       ; ALPHA is a Data field
            CALL   BETA           ; BETA is a Label
              :
              :
            ENDS
            END
```

Note:
- Each Name in the List may be declared only once by EXTRN.
- A fixed Symbol in the EXTRN assignment may not be re-defined.
- The sum of PUBLIC- and EXTRN Symbols may not exceed 255 (OS dependant).
- Macro-Names may not be arranged with EXTRN.
- If the instructions CALF or CALT or the unary Operators HIGH or LOW are to be used in connection with EXTRN-Symbols, no check of the valid Address area occurs during Linking process.
  The same is valid for the corresponding instructions of the uPD78310/ uPD78312/ uPD78112/uPD78210/ uPD78220/ uPD78224.
- EXTBIT symbols can be interpreted as bit symbols in saddr-area (0FE20H-0FF1FH) (only RA310/RA210) or (0FE40H-0FF1FH) (only RA110).
- EXTRN/EXTBIT definitions may only occur in the module header.
- EXTRN/EXTBIT symbols used as 'saddr' or 'saddr.bit' operand are supposed to lay in the saddr area. If not the Locater will output the error message 'MISUSED SYMBOL'.

2.1.11.6. Directives for Locator Run

2.1.11.6.1. ORG Assignment

Syntax: name ORG Expression

Expression: arithmetic Expression (see 2.1.9.3.4.) with following restrictions:

- No bit symbols are allowed
- External Symbols are not allowed in <Expression>.
- No forward references are allowed; all names and symbols must defined before.
- <Expression> must be 'absolute'.

name: must conform to Symbol definition.

The assignment ORG can be used to change the current value of the current address level to be used.

```
Example:        DSEG
        ALPHA : DB (100)          ; 100 Bytes will be reserved
                                  ; relative to Segment start.
                ORG       1000H   ; The Address level will be
                                  ; set to 1000H.
         BETA : DB (100)          ; 100 Bytes will be reserved
                                  ; from Address 1000H.
                ORG       2000H   ; The Address level will be
                                  ; set to 2000H.
        DELTA : DB (100)          ; 100 Bytes will be reserved
                                  ; from Address 2000H.
                ENDS
```

Note
- The validity of an Address level reaches to the end of a Segment. Therefore, the validity of an ORG assignment is limited. The ORG assignment is valid, starting from the assignment itself, until the end of Segment is reached or upto a new ORG assignment or either CSEG,DSEG or VSEG directives.
- ORG assignments may only occur in Data or Code segments.
- The ORG assignment defines a new Absolute Segment.
- A maximum of 10 ORG assignments per Module is allowed.
- If the entered adress is smaller than:
  - 0C0H at RA87
  - 080H at RA310 / RA110 / RA210

  A Warning occurs in the Listing and on the Console.
- ORG defined in CSEG belongs to Code Segment.
- ORG defined in DSEG belongs to Data Segment.

2.1.11.7. Macro-Directives

2.1.11.7.1. MACRO Assignment

Syntax: name MACRO [parameter[,parameter]...]

name: defines a name for the Macro. It must conform to the rules for the construction of a Symbol.

parameter: Must also conform to the rules for construction of a Symbol. These formal Parameters will be substituted via Expressions in a Call. Several Parameters must be separated by commas.

The MACRO assignment signifies the start of an instruction sequence, which can be implemented in a Code

Segment by a call to MACRO. With the calling of the parameters, for these, macro values can be locally delivered.

```
Example:   TYP MACRO PARAM1, PARAM2 ; A Macro with the
                 :                  ; name "TYP" and the
                                    ; parameters PARAM1
                                    ; and PARAM2 is
                                    ; defined.
               MOV  A, PARAM1
               MOV  H, PARAM2
               :
               ENDM                 ; End Macro definition.

               The Call
               TYP 2, $-3
               results in
               .
               .
               .
                   MOV A, 2
                   MOV H, $-3
```

Note:
- The Maximum number of Parameters is five.
- Parameter Symbols are added to the maximum quantity of Symbols.
- Formal and current Parameters must agree in Type and quantity.
- A Module may not contain more than 10 Macros.
- The Macro must be ended via Direktive ENDM.
- It is not allowed to nest Macros by itself or with REPT and IRP.

2.1.11.7.2 LOCAL Assignment

Syntax: LOCAL label [,label]...

label: signifies a label; the Name must conform to the rules for construction of s Symbol.

The LOCAL assignment signifies the labels in the Label-List as local within the defined Macro. The labels must be separated by commas.
The same applies for symbols defined inside macros.

```
Example:  ABC MACRO PARAM1
          LOCAL ABCLAB
          LOCAL DEF
           .
          DEF SET PARAM1

           .
           .
          ABCLAB:
           .
           .
           .
          ENDM
```

Note: If a Label within a Macro definition is not specified as LOCAL, by repeated Macro Call an Error will result (Multiple Declaration).
- A label defined as LOCAL corresponds to exactly one entry in the Symbol Table, even by repeated Macro Calls.
- The LOCAL assignment may only be used within Macro definitions.

2.1.11.7.3. REPT Assignment

Syntax: [name:] REPT Expression

name: defines a logical Address

Expression: arithmetic Expression(2.1.9.3.4) with following restriction:

- Bit Symbols are not allowed.
- External Symbols are not allowed in <Expression>.
- Labels or names which appear in <Expression> must have already been defined (No forward referencing).
- The value of <expression> must be 'absolute'.

The REPT assignment repeats the instruction sequence (between the REPT assignment and the following ENDM assignment or through conditional assembly accessable EXITM assigment) n times, where n is the value of the Expression.

Example:
```
REPT 3
DCR  A
ENDM

will execute as

DCR A
DCR A
DCR A
```

Note:
- REPT may not be nested by itself or with IRP and MACRO.
- No Macro definition may exist within a REPT instruction sequence.

2.1.11.7.4 IRP assignment

Syntax: [name:] IRP param <Expression[Expression]...>

name: defines a logical Address

Expression: arithmetic Expression (2.1.9.3.4.) with following restriction:

- no external symbols
- no bit symbols

parameter: must conform to the rules for construction of a Symbol.

The IRP assignment repeats the instruction sequence between the IRP assignment and the following ENDM assignment, or through conditional assembly accessable EXITM assignment), n times, where n is the value enclosed within the brackets < >.
By the x-times repetition the Parameter param will be allocated the xth expression.

```
Example   N1 EQU 1
.
.
.
IRP PAR, <N1, 2,4>
ADI A, PAR
ENDM
```

will be executed as:

```
ADI A, 1
ADI A, 2
ADI A, 4
```

Example: - IRP may not be nested by itself or with REPT and MACRO.
- IRP may not be used within REPT assignments.
- No Macro definitions may exist within an IRP instruction sequence.

2.1.11.7.5. ENDM assignment

Syntax: ENDM

The assignment ENDM defines the physical and logical end of an instruction sequence defined by MACRO, REPT or IRP assignments.

Example: see Example under 2.1.11.7.1.

2.1.11.7.6. EXITM Assignment

Syntax: EXITM

The assignment EXITM defines the logical end of a sequence of instructions defined by MACRO, REPT or IRP assignments. EXITM doesn't have to be used but recommends itself, when conditional Assembly occurs within the Macro definition.

Example:

```
ABC MACRO
.
.
.
$IF (XYZ)
  MOV C, A
  EXITM
$ENDIF
  MOV A, C
  ENDM
  .
  .
$SET (XYZ)
  ABC
```

Note:

- EXITM is only relevant for the Macro Call. Here it operates exactly like ENDM. For the Macro definition only ENDM applies to Macro's physical end.
- If EXITM was called in an active $IF branch within a Macro, then $ENDIF Control command will be ignored; this doesn't appear in the Listing.
- EXITM may only be used in Macros in connection with conditional Assembling.
- At conditional Assembling within a Macro, the switches must have been defined, before the Macro definition using SET or RESET. The switch can be re-defined before each Macro Call.

2.1.11.7.7. EXTBIT-Directive (RA310,RA210,RA110)

Syntax: EXTBIT symbollist

symbollist ::= symbol[,symbol]

Using the EXTBIT-Directive, you can address bits - located in the saddr-area - out of module boundary. However, you must define a symbol attributed by the EXTBIT-Directive in another module using the PUBLIC-Directive.

Note: - Every symbol in the symbollist must conform to symbol definition.
- Every symbol in the symbollist may occur only once.
- Redefinition of an EXTBIT attributed symbol is not allowed.
- EXTBIT attributed symbols are added to the number of global symbols. There are only 255 global symbols allowed in RAxxx-Family-Assembler.

- Errors are noticed by linker or locater, if an EXTBIT attributed symbol is located outside the saddr-area (not RA87).

2.1.11.7.8. BSEG/ENDS-Directive (not for RA87)

Syntax: [name] BSEG
:
[ENDS]

name: must conform to symbol definition

The BSEG/ENDS-Directive defines an address area of at most

- 100H Byte from base address 0FE20 for RA310/RA210
- 100H Byte from base address 0FE40 for RA110

In BSEG you can define single bit-values.

Note:
- You can define only one B-Segment in a module.
- A B-Segment is a relocatable segment; B-Segments will be linked together by the LKxxx-Linker.
- The default address of the B-Segment (specified by the LCxxx-Locater) is the base address of the saddr-area:
  0FE20 for RA310/RA210
  0FE40 for RA110
- Address level of BSEG starts at
  100H for RA310/RA210/RA110
- The length of a BSEG will be counted in bits.
- The only directive in BSEG, that changed address level is - DBIT Directive
- Bits will be addressed in sequence.
- Bit Arrays are not allowed.
- Labels are not allowed before BSEG-Directive.
- ORG Assignments are not allowed within a BSEG

```
example:    BITSEG   BSEG
                     DBIT
                     DBIT
              ABC    DBIT
                     DBIT
            ENDS
```

A B-Segment named BITSEG is created. The size of the segment is 4 Bit. The symbol 'ABC'is a so called Bitsymbol.

- A bitsymbol may only used in assignments refering to the 'saddr.bit' operand. They are marked in the symbollist with the attribute 'BIT'.

2.1.11.7.9. Extended EQU-Assignment (RA310/RA210/RA110)

extended Syntax:

```
name EQU name1      ; name1 = Bitsymbol
name EQU value.bit
name EQU saddr.bit
name EQU sfr.bit
name EQU A.bit
name EQU X.bit
name EQU PSWH.bit
name EQU PSWL.bit
```

- Name must conform to symbol definition.
- Bit is in the range 0..bit..7.
- Symbols defined within the saddr could be exported using the PUBLIC-Directive.
- Symbols defined with the extended EQU-Assignment could be used in assignments, which have the operands:

```
-- saddr.bit
-- sfr.bit
-- A.bit
-- X.bit
-- PSWH.bit
-- PSWL.bit
```

2.1.11.8. Directives for use of Register banks (not for RA87)

2.1.11.8.1. USING Directive (only for RA310)

Syntax: USING RBxx[,RBxx..]

The use of Register banks can be defined via directive USING.

```
RBxx : --> RB0
           RB1
           :
           :
           RB7
```

Note: The double definition of a Register bank is forbidden.

Example:

```
USING RB0, RB5 : The use of Register banks 0 und 5
                 is defined.
```

2.1.11.9. Assignment Directives of Functional and Absolute Names

2.1.11.9.1. RSS-Directive (only RA310)

Syntax: RSS <well-defined expression>
<well-defined expression> is an absolute value and may only assume the values 0 and 1. With the RSS-directive, different absolute name registers can be attached to functional name

registers. The assignment ist evident out of the following table. No bit symbols are allowed.

table

functional	absolut name	
	RSS 0	RSS 1
X	R0	R4
A	R1	R5
C	R2	R6
B	R3	R7
VPL	R8	R8
VPH	R9	R9
UPL	R10	R10
UPH	R11	R11
E	R12	R12
D	R13	R13
L	R14	R14
H	R15	R15
AX	RP0	RP2
BC	RP1	RP3
VP	RP4	RP4
UP	RP5	RP5
DE	RP6	RP6
HL	RP7	RP7

```
Example: RSS   0
         MOV   b, a ; equivalent with
         MOV   r3, r1

         RSS   1
         MOV   b, a ; equivalent with
         MOV   r7, r5
```

2.1.12. RA87/ RA310 / RA110 / RA210 Key words (reserved Words)

All Assembler instructions and Register descriptions as given in paragraph 2.1.2.

DSEG	DEBUG	NODEBUG
CSEG	OBJECT	NOOBJECT
VSEG (7809 only)	PAGING	NOPAGING
ENDS	PAGELENGTH	PAGEWIDTH
NAME	PRINT	NOPRINT
PUBLIC	TITLE	WORKFILE
EXTRN	XREF	NOXREF
EQU	PROCESSOR	OPTIMIZE
DBIT	EJECT	INCLUDE
DB	LIST	NOLIST
DW	SUBTITLE	SET
ORG	RESET	IF
MACRO	ELSE	ELSEIF
ENDM	ENDIF	DG
END	NODG	OB

```
NOOB                PI                  NOPI
PL                  PW                  PR
NOPR                WF                  XR
NOXR                PC                  EJ
IC                  LI                  NOLI
ST                  TT                  DATE
OP
DS                  NOT                 DA
MOD                 OR                  XOR
AND                 NE                  GT
EQ                  GE                  LT
LE                  SHL                 HIGH
SHR                 TB
LOW                 TAB
LOCAL               USING (RA310 only)
REPT                EP
IRP                 ERRPRN
EXITM               EXTBIT
FIXED               BSEG (not for RA87)
CALLT0 (RA310 / RA110 only)
CALLT (RA87 / RA210 only)
CALLT1 (RA310 only)
```

2.1.13. Boundary Characters

B L A N K	- Control command<--> Control command - Opcode <--> Displacement - NAME/ MACRO/ DS/ CS/ BS <--> name - name <--> EQU/ SET <--> name - EXTRN/ PUBLIC <--> name - DBIT/ DB/ DW <--> const
,	Operand <--> Operand Separation Char.between List elements
;	Start of Comment
:	Label Field <--> Opcode Field
LF	Line end
HT FF CR	identical with Blank
+	Sign of addition,Autoincrement Register, unary Plus
-	Sign of subtraction,Autodecrement Register, unary Minus
(, <	Start of a Parameterlist/ Initialisation/ Indices

), >	Ends Parameterlist/ Initialisation/ Indices
.	Fullstop operator for direct Bit addressing of Sr5-Registers
* /	
$	- Location-Counter - For Address recognition in RA310 Syntax. - Separates Control command and Sourcecode, when in first column - in Symbols as non-significant character
[,], !	- as Syntax boundary in RA310

2.1.14. Segments

The Assembler RA87/ RA310 / RA110 / RA210 produces the following Segments:

Segment		created via Directive
Code-Segment		CSEG
Code-Segment Fixed		CSEG FIXED
Code-Segment Callt0/Callt (RA310/RA87/RA110/RA210)		CSEG CALLT0 or CSEG CALLT
Code-Segment Callt1 (RA310 only)		CSEG CALLT1
Data-Segment		DSEG
V-Segment relocatable(RA87 only)		VSEG
Bit-Segment (RA310/RA110/RA210 only)		BSEG
V-Segment 0	RA87	VSEG (address)
V-Segment 1	RA87	VSEG (address)
V-Segment 2	RA87	VSEG (address)
V-Segment 3	RA87	VSEG (address)
V-Segment 4	RA87	VSEG (address)
Absolute-Segment 0		ORG address
Absolute-Segment 1		ORG address
Absolute-Segment 2		ORG address
Absolute-Segment 3		ORG address
Absolute-Segment 4		ORG address
Absolute-Segment 5		ORG address
Absolute-Segment 6		ORG address
Absolute-Segment 7		ORG address
Absolute-Segment 8		ORG address
Absolute-Segment 9		ORG address

- The individual Segment can be characterized via entry of Names (--> name segments).

```
e.g.: Code CSEG          ; Name of Code-Segment        : Code
      CodFix CSEG Fixed  ; Name of Code-Segment-Fixed  : CodFix
      Abs1 ORG  1000H    ; Name of Absolute-Segment    : Abs1
```

Note: You may refer to these names when using the 'SEGMENT' directive of the Locator.

2.2. Stack

2.2.1. Stack Structure

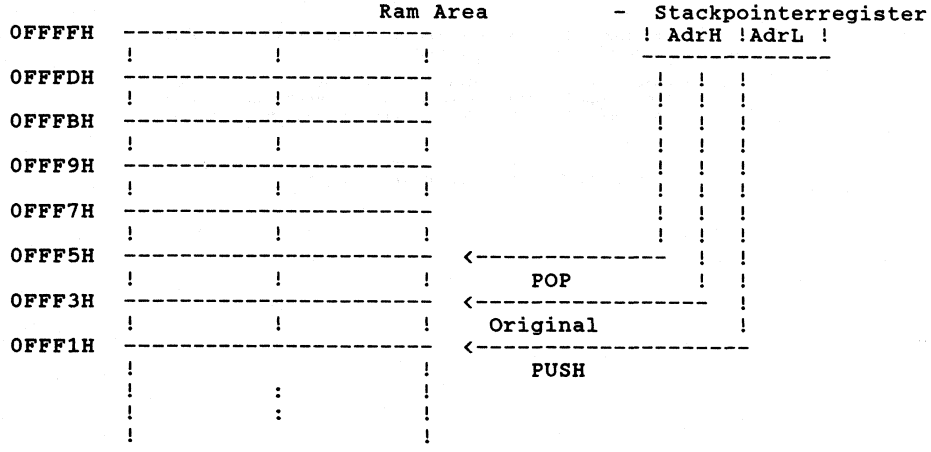
The Stack is an area in RAM, which can be addressed by the Stackpointer.
The Stackpointer is realized through a Hardware register, that must be initialized by the loading of a Program. By stack operations the stackpointer is incremented or decremented.

The following Operations are allowed:

PUSH: stores a 16Bit value in RAM addressed by the Stackpointer and finally decrements the Stackpointer.

POP: fetches a 16Bit value from RAM addressed by the Stackpointer and finally increments the Stackpointer.

Fundamental structure of the Stack:



Note:
- The Stack is constructed in LIFO fashion (last in - first out).
- In order to avoid any conflicts within the user specified RAM Area, it is efficient to define the Stack area at the highest logical Address.

2.2.2. Stack size

- The Stack size is user specific. The size depends on the Operations to be performed on the Stack.

- The Linker requires the maximum size of the Stack.

- A PUSH Operation sequentially adds 1 to the Stack size, the POP-Operation subtracts one from Stack size. The highest value of the Stack size gives, at the same time, the maximum Stack size.
  At a subroutine Call the Stack size will be incremented only once. If the calls are nested, then the Stack size will be incremented for each level of nesting.

		Stack size	
		Present	maximum
	PUSH H	1	1
	PUSH D	2	2
	:	:	:
	:	:	:
	PUSH C	3	3
	CALL BETA	4	4
	POP C	2	4
	:	:	:
	POP D	1	4
	POP H	0	4
	END	:	:
BETA:	PUSH H	5	5
	PUSH D	6	6
	:	:	:
	:	:	:
	POP D	5	6
	POP H	4	6
	RET	3	6

- PUSH, POP, RET, CALL are taken from RA87 instruction set.
  For RA310 the corresponding instructions are to be used.

2.3. Character Set

The following characters may appear in Sourcetext.

a) alphanumeric Characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z

b) numerical Characters:

0 1 2 3 4 5 6 7 8 9

c) Special Characters:

Blank ? @ , . + - * / ( ) $ = ; : ' & # < > ! " % [ ] ^ _\ < >

d) Other Control Characters:

CR LF HT FF

2.4. Limitations

- The number of Inputfiles		1
- The number of Characters per Symbol		6
- Nesting depth of Include Command		1
- Number of characters per line		102
- Segment size		64k -1
- Number of EXTRN-, EXTBIT- and PUBLIC-Symbols		256
- max. number of Characters in a Module		65535
- max.number of defined Macros		10
- max.nesting depth of Include/Macro		10
- Macro Transfer Parameter		5
- Nesting depth at conditional Assembly		0
- max.number of ORG assignments per Module		10
- max.number of Switches at conditional Assembly		5
- max.number of Errors		9999
- Number of Segments:	- Code	1
	- Code-Fixed	1
	- Code-Table0	1
	- Code-Table1 (RA310 only)	1
	- Data	1
	- 5 V-Segments (RA87 only	5
	- rel. V-Segment (RA87 only)	1
	- Absolute-Segments	10
- The number of Symbols and References for XREF List.		Depends on available Memory of respective Operating System.
- max. number of bits in BSEG	(RA310/RA210)	2048
	(RA110 only)	1792

3. Environment of Assemblers RA87/RA310/RA110/RA210

3.1. Assembling

3.1.1. File presence

The Software Packet Assembler RA87 contains the following Files:

```
RA87.EXE or RA87.CMD
RA87 or RA87.OM1
     -----
RA87.OV0 !
RA87.OV1  >  only RA87.V1.5
RA87.OV2 !
     -----
RA87.OM2
RA87.OM3
RA87.OM4
RA87.OM5
```

The Software Packet Assembler RA310 contains the following Files:

```
RA310.EXE or RA310.CMD
RA310 or RA310.OM1
RA310.OM2
RA310.OM3
```

The Software Packet Assembler RA110 contains the following Files:

```
RA110.EXE or RA110.CMD
RA110 or RA110.OM1
RA110.OM2
RA110.OM3
```

The Software Packet Assembler RA210 contains the following Files:

```
RA210.EXE or RA210.CMD
RA210 or RA210.OM1
RA210.OM2
RA210.OM3
RA210.OM4
```

A correct Assembler run can only occur if all above named Files are present.

3.1.2. Assembler Call

The Activation occurs at Operating System level according to following format:

```
[<path>] RA87 [<path>]<source-file-name>
[<RA87 Control commands>]<cr><lf>
```

Meaning:

<path> and path	: Floppy drive name
<source-file-name> assembled.	: Name of Source file to be
<RA87 Control commands>	: Control commands for the Assembler to influence the Assembler Output. (see 2.1.10.xx) for RA310 use RA310 for RA110 use RA110 for RA210 use RA210

The Assembler Call may not be longer than 122 Characters.
A successful assembler call results in writing the startup-message on the Console.

3.1.3. Control commands in Call

By Call the following Control commands (as well as their abbreviations) can be entered:

DEBUG	NODEBUG
OBJECT	NOOBJECT
PAGING	NOPAGING
PAGELENGTH	PAGEWIDTH
PRINT	NOPRINT
TITLE	WORKFILE
XREF	NOXREF
PROCESSOR	OPTIMIZE
SUBTITLE	SET
RESET	DATE
LIST	NOLIST
TAB	ERRPRN

The use of any other Control command at Assembler activation results in an Error Message.

3.2. Output

3.2.1. Output Equipment

3.2.1.1. Terminal Output

After the Assembler process the following message appears on the Terminal:

Part of Display:

```
-----------------------------------------
! ASSEMBLY COMPLETE  xxxx ERROR(S) FOUND !
-----------------------------------------
```

3.2.1.2. Printer Output

There is a direct Printer Output (PRINT (LST:)). The Listfile is prepared ready for printing and may be output to the printer by a corresponding System Call.

3.2.1.3. Floppy Disc Output

The Assembler produces the following Output files on Floppy Disc.

- Listfile (dependent on PRINT Control command) and possibly XREF-List.
- Object file (dependent on OBJECT Control command) and only after Assembler process, without errors.

3.2.2. Output Files

3.2.2.1. Listfile

Listfile output is optional; i.e. a Listfile is only executed, when the Control command PRINT is set.

The Listfile delivers the following Information:

1. Line : - Type of Processor
  - Assemblerversion
  - TITLE:
  - DATE beginning at PAGEWIDTH - 24
  - Page number starting at Position PAGEWIDTH -10

2. Line : - optional SUBTITLE

3. Line : - Source file name

4. Line : - Object file name

5. Line : - Call Command

6. Line to 11. Line - Header

Lines 3 to 11 appear only on page 1 .

From Line 13 a Line is splitted as follows:

Column 1 to Column 5 : Contain the Line number, a decimal number sequentially ordered for each Program and Control line.

Column 8 to Column 11 : Contain the relative/ absolute Address.
The Columns, which are unassigned, are filled with space.

```
                                Sequence of Address Input:
                                        Low-Byte,
                                        High-Byte

Column 13                   : Indicates whether relocatable
                              Code will be produced.
                              Blank  ==>  relocatable Code
                              A      ==>  absolute Code

Column 15 to Column 24:       Object Field: Printout of the
                              Hexadecimal
                              relocatable/absolute      Object
                              code, created by Assembler.
                              - Value of EQU-, SET-Direktive;
                              - The Bit number used in V-
                                Segment.

Column 26                   : M-Field: Signifies Macro defini-
                              tion/Macro-Call.
                              M  ==>  Macro definition
                              #  ==>  Macro expansion

Column 28                   : Signifies an Include File:
                              Default  ==>  Blank
                              1-9  Includes Nesting depth.

from Column 31              : Copy of Source code
```

The Assembler assignments DBIT, DB and DW, which contain a List of Operands, result in code for each Operand being written in the Object Field. If more than 5 Operands are defined, then a line break occurs.

Error Messages will be entered in the List file in accordance with paragraph 3.3.1. They appear from column 1 directly under the faulty Listfile line.

A Source file line which is longer than (PAGEWIDTH 30) shall be continued from column 31 on the next line. The Information Field stays empty.

Page Advance,Page numbering:

A Page feed is generated when the Page line counter (which includes Comment,Error,Heading,Empty lines and Error Message lines) reaches the value set by PAGELENGTH (n).

The Page numbering starts at 1 and stops at 65535. When the last limit is overstepped, the Page numbering starts again from 0 upwards.

The Page number is entered in column("PAGEWIDTH-10") on line 1 of the List page. The entry has the form:

"Page xxxxx"
with 1 < = xxxxx < = 65535

RA87 List file format

```
        10        20        30        40        50        60        70        80        90        100       110       120       132
++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!
01 +UPD7809 ASSEMBLER Vx.y       1 < = Title < = 60         DATE ddddddd PAGE XXXXX                                              +
02 +1 < = Subtitle < = 60                                                                                                       +
03 +SOURCE FILE: :LW:filename.asm                                                                                               +
04 +OBJECT FILE: :LW:filename.obj                                                                                               +
05 +COMMAND    : :LW:filename.asm <Complete call of  RA87 Assembler>                                                             +
06 +                                                                                                                            +
07 +                              /* The lines 3-11 appear only on Page1 */                                                     +
08 +                                                                                                                            +
09 +     ASSEMBLE LIST                                                                                                          +
10 +                                                                                                                            +
11 + STNO ADRS R OBJECT     M I  SOURCE STATEMENT                                                                               +
12 +                                                                                                                            +
13 +nnnnn aaaa R ooooooooo M I  TTTTTT                                                                                           +
   +  !     !  !        !   ! !    !------Source Text                                                                           +
   +  !     !  !        !   ! !-----------Includes Nesting Depth                                                                +
   +  !     !  !        !   !-------------Signifies Macro                                                                      +
   +  !     !  !        !-----------------Object code                                                                          +
   +  !     !  !-------------------------R/A ==> relocatable/ absolute                                                         +
   +  !     !----------------------------Address                                                                              +
   +  !-----------------------------------Line number                                                                          +
   +                                                                                                                            +
   +                                                                                                                            +
   +                                                                                                                            +
   +                                                                                                                            +
   +                                                                                                                            +
   +                                                                                                                            +
   +                                                                                                                            +
   +                                                                                                                            +
   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Final information in List File:

At End of a List File

- A Page feed occurs

- The following information will be output:

```
TARGET CHIP: UPD78 xx

ASSEMBLY COMPLETE    xxxxx Error(s) found

Maximum Stack-Size: xxxxH

Segment Information:
--------------------

ADRS  LEN     NAME
xxxx  xxxx  xxxxxx
  :     :      :
  :     :      :
```

For every valid Segment a line is output containing:

- The Segment Base Address

- The length of the Segments in Bytes

- The Segment name

3.2.2.2. Cross Reference List

The Cross-reference List is an alphabetical Listing of all Symbols with their Program position, (Line numbers relative to Program start) at which the Symbols shall be used.

3.2.2.2.1. Format of the Cross Reference List

By using of the Primary Control Assignment XREF, a list of all used Symbols is summarized at the end of Program Listing.

The Address of the given Program element is relative to the start of the Segment, to which the element belongs.

The XREF-List contains the following Information:

- Name — Name of Symbol

- Addr — Address of Symbol (relative to start of the segments, which are belonged) or value of Symbol (for EQU- and SET assignments) or Bit number (DBIT)

- R/A — relocatable/ absolute --> R/A

- Attributes — Attribute of Symbol:
  - - MOD - Module
  - - MAC - Macro
  - - EXT - External Variable
  - - LOC - Local Variable
  - - PUB - Global Variable
  - - EQU - EQU, SET
  - - BIT - Bit Variable
  - - ERR - false Symbol
  - - PAR - Macro-Parameter
  - - MLB - Macro-Label

- Seg Segment name
- References Line number of the assignments in which the Symbol appears (Suffix # by definition).

```
      10        20        30        40        50        60        70        80        90        100       110       120
++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++!++++++++++
01 +Cross-Reference Listing:                                                                                                 +
02 +                                                                                                                        +
03 +                                                                                                                        +
04 +                                                                                                                        +
05 +Name       Addr  R Attr Seg    References                                                                               +
06 +---------------------------------------                                                                                 +
07 +                                                                                                                        +
08 +NNNNNNNNNN XXXXH R AAA  SSSSSS rrrrr rrrrr                                                                            +
09 + :                                                                                                                      +
10 + :                                                                                                                      +
   + :                                                                                                                      +
   + :                                                                                                                      +
   +                                                                                                                        +
   +  /* No informations to Page number,Title,Sub-title etc., appear                                                        +
   +     in the Cross reference List.                                                                                       +
   +                                                                                                                        +
   +     The header-lines 1 - 7 appear at the start of every new side.                                                      +
   +     After a maximum of 66 printed lines (see Control command PAGELENGTH)                                              +
   +     an automatic Page feed occurs.                                                                                     +
   +                                                                                                                        +
   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

3.3. Assembler Error Messages

3.3.1. Error Recognition and Error Classes

The Assembler recognizes the Error Classes:
- Primary Program Error
  Errors are entered in List file. They relate to the form and content of the Source file.
- Time out Error
  Results in the premature ending of Assembling.
  The Error will be output to the Terminal.

3.3.2. Format of Error Messages

The Assembler Errors appear in the Listing at the position, where they occured.
They have the form:

*** ERROR *** message ***

3.3.3. List of Error Messages

```
*** ERROR --1--     contact NEC Europe ------------------------****
*** ERROR --2--     contact NEC Europe ------------------------****
*** ERROR *** insufficient memory   ****************************
*** ERROR *** attempt to open more than six files   ************
*** ERROR *** illegal filename   ******************************
*** ERROR *** illegal device specification   ******************
*** ERROR *** no more room in disk directory/on disk   *********
*** ERROR *** attempt to open a file already open   ************
*** ERROR *** no such file   **********************************
*** ERROR *** access to a write protected file   ***************
*** ERROR *** cannot delete an open file   ********************
*** ERROR *** overlay doesn't exist or invalid overlay   *******
*** ERROR *** illegal read/write operation   ******************
*** ERROR *** illegal attempt to open/close/delete a file   ****
*** ERROR ***    commandline exceeds   ***************************
*** ERROR ***    error in commandline   **************************
*** ERROR ***    sourcefile does not exist   *********************
*** ERROR ***    cannot create listfile   ************************
*** ERROR ***    cannot read sourcefile   ************************
*** ERROR ***    cannot create objectfile   **********************
*** ERROR ***    cannot open errorfile   *************************
*** ERROR ***    include-/macro-nesting too deep   ***************
*** ERROR ***    more than one command per line not allowed  ***
*** ERROR ***    too many switches or switch too long   ********
*** ERROR ***    no switch definition found   ********************
*** ERROR ***    too many switches defined   *********************
*** ERROR ***    illegal parameter in controlline   **************
*** ERROR ***    illegal switchsymbol   **************************
*** ERROR ***    IF-clauses not correct   ************************
*** ERROR ***    premature end of file   *************************
*** ERROR ***    inputfile too long   ****************************
*** ERROR ***    insufficient input-memory   *********************
*** ERROR ***    inputstring too long   **************************
*** ERROR ***    number greater than 0FFFFH   ********************
*** ERROR ***    illegal character or symbol too long  *********
*** ERROR ***    illegal character in symbol   *******************
*** ERROR ***    missing apostrophe (ill. string)   *************+
```

```
*** ERROR ***     illegal character in constant   ***************
*** ERROR ***     too many elements in source-line   *************
*** ERROR ***     division by zero   ***************************
*** ERROR ***     controlword not correct    **********************
*** ERROR ***     crossreference-table inconsistency   ***********
*** WARNING ***   occurrence-table-overflow   ********************
*** ERROR ***     illegal relocatable value **********************
*** ERROR ***     public bits are out of saddr-area   ************
*** ERROR ***     no byte-oriented directive allowed in BSEG   **
*** ERROR ***     too many bits defined in BSEG   ***************
*** ERROR ***     segmentlength greater than 0FFFFH   ************
*** ERROR ***     segment name not consistant ********************
*** WARNING *** 000H <= ORG-address < 080H   ********************
*** ERROR ***     location counter overflow   ********************
*** ERROR ***     illegal use of local symbol   ******************
*** ERROR ***     fix-code-segment exceeds ***********************
*** ERROR ***     table-code-segment exceeds *********************
*** ERROR ***     illegal use of externals ***********************
*** ERROR ***     illegal parenthesis use ************************
*** WARNING ***   double declaration of registerbank *************
*** ERROR ***     illegal memory access   ************************
*** ERROR ***     identifier expected   **************************
*** ERROR ***     symbol declared twice   ************************
*** ERROR ***     too many operands   ****************************
*** ERROR ***     illegal operand / separator   ******************
*** ERROR ***     illegal register type   ************************
*** ERROR ***     well-defined expression required   *************
*** ERROR ***     illegal separator / operand   ******************
*** ERROR ***     value >= 256   *********************************
*** ERROR ***     illegal absolute value   ***********************
*** ERROR ***     destination allowed only in same segment   ****
*** ERROR ***     address out of range   *************************
*** ERROR ***     illegal expression   ***************************
*** ERROR ***     undeclared identifier   ************************
*** ERROR ***     symbol-table-overflow   ************************
*** ERROR ***     too many global symbols   **********************
*** ERROR ***     too many absolute segments   *******************
*** ERROR ***     too many code-segments   ***********************
*** ERROR ***     too many data-segments   ***********************
*** ERROR ***     too many V-segments   **************************
*** ERROR ***     illegal value   ********************************
*** ERROR ***     named ORG-section not allowed  *****************
*** WARNING ***   000H <= ORG-address < 0C0H   ******************
*** ERROR ***     too many V-bits   ******************************
*** ERROR ***     illegal bit-number   ***************************
*** ERROR ***     no initialization in this segment   ************
*** ERROR ***     too many initialization-elements   *************
*** ERROR ***     illegal initialization-element   ***************
*** ERROR ***     string incomplete   ****************************
*** ERROR ***     opcodes allowed only in code-segment   *********
*** ERROR ***     no DBIT-directive allowed here   ***************
*** ERROR ***     V-segment too long   ***************************
*** ERROR ***     illegal start-address of V-segment   ***********
*** ERROR ***     globals in V-segments/equates not allowed   ***
*** ERROR ***     segment already closed   ***********************
*** ERROR ***     no module name defined   ***********************
*** ERROR ***     module name already defined   ******************
*** ERROR ***     ORG not allowed here   *************************
*** ERROR ***     illegal set-symbol   ***************************
*** ERROR ***     illegal macro-name   ***************************
*** ERROR ***     nested REPT/IRP/MACRO not allowed   ************
```

```
*** ERROR ***    illegal IRP-List   ****************************
*** ERROR ***    misused reserved symbol   ********************
*** ERROR ***    too many defined macros   ********************
*** ERROR ***    illegal number of macro-parameters   **********
*** ERROR ***    symbol declared before PUBLIC-directive   *****
```

4. Linker Function

The linker LK310/210/110 is used for the modular design of software for NEC's UPD78KIII/KII/KI family of single chip microprocessors. The lnk87 is used for the single chips of the uCOM87 family. The linker binds one or more relocatable input modules to one relocatable output module. Each module is attached to a file (input file resp. output file).

Besides of the module treatment, the linker may optionally create a mapfile.

In the following chapters only the LK310 will be named. But the same functions and operation rules are valid for the LK210/LK110 or LNK87.

Linker input files are:

- relocatable object files from translators as RA310/RA210/RA110/RA87 and from LK310/LK210/LK110 OR LNK87
- library files from LB310/LB210/LB110/LIB87

Linker output files are:

- a relocatable linked object file
- a optional formattable list file (mapping)

The locater LC310/210/110 or LOC87 enable the transformation of the relocatable output object file into an executable, absolute file.

The linker creates a temporary workfile during the linkage process. This workfile is given the name of the linker's output ROF with the extension "tmp".

At the end of a linkage process the workfile is deleted.

4.1. Linkage Process

4.1.1. Structure of a Translator Object File

A compiling or assembling process upon an user designed source module creates a relocatable object file (ROF). Therein all module specific symbols and data are assigned to relocatable or absolute segments.

The segments are defined as follows (the definition applies to segment types):

CSEG : Relocatable code segment; it contains all assembler statements and instructions which occur between the directives CSEG and ENDS or

another segment directive. Compilers allocate all code generating statements and all constant declarations to the CSeg.

CSFX : Relocatable code segment with a fixed base address of 800H and a length of 800H. The segment contains all assembler statements and instructions which are placed in the Fixed Area.

CST0 : Relocatable code segment with a processor dependent base address (e.g. 40H for uPD7811) and a length of 40H. This segment contains all Call Table assignments for processors with one Call Table and Call Table 0 assignments for processors with two Call Tables.

CST1 : As CST1. Used only for Call Table 1 assignments associated with processors having two Call Tables.

DSEG : Relocatable data segment; it contains all assembler statements and instructions which occur between the directives DSEG and ENDS or another segment directive. Compilers allocate all data, defined in a module, to DSeg (with the exception of dynamic data).

VSEG : Relocatable V - segment with a base address variable in steps of 256 bytes and with a length of 100H. The segment contains all data definitions which are refered to "Paging".

SSEG : Relocatable stack segment; it contains the stack requirement of each translator (= assembler or compiler) module. Stack requirement includes all PUSH commands from assemblers or compilers, as well as actual parameters and dynamic data from compilers.

ASEG : Absolute segment. The absolute segment contains assembler statements and instructions, which occur between the directives ORG and ENDS or another segment directive. The ASeg is seperated in code resp. data sections. The seperation depends on the class of the segment (CSeg or DSeg) preceeding the current org-directive. Compilers will allocate that symbols to an ASeg, which occur in compiler specific absolute memory allocation statements.

BSEG : Relocatable bit segment; it contains bit definitions of assemblers which occur between the directives BSEG and ENDS or another segment directive.

At the time of translation, each segment can get an user defined name with a length of up to 6 characters. If no name is assigned, default names will be given. Default names are the above segment identifiers. The absolute

segments may represent an exception. If the user assigns no name, a not visible "artificial name" will be created at translation time.

At the time of translation the relocatable segments don't have any fixed locations in the physical memory of the target system. All statements and symbols attached to segments do not have a real, existing address. They have only relative addresses dependent on a relocatable segment base address.

The absolute segments, on the other hand, have fixed addresses. Yet after translation they correspond to final addresses within the target system.

After a translation process the ROF contains, apart from segment allocations, any possible references to other modules. Among these references are all symbol declarations which have global (=EXTERNAL/ PUBLIC) attributes attached.

For a translator, it is possible to allocate a global symbol to a segment because of this symbols' use in current translator module. However the relative address within a segment cannot be defined when using EXTERNALS. The relative address will be given by the translator module, in which the symbol is defined as PUBLIC. The combination of PUBLIC and EXTERNAL must be done by the linker.

4.1.2. Combining Translator Modules to a Bound Module

In order to combine several translator modules into a bound module, they must be linked. To achieve this, all relocatable segments of a module are copied to segments of higher order. This in turn requires the conversion of the previous base addresses of the relocatable segments by adding the length of a segment to its base address. The result represents the base for a relocatable segment with the same name and type of the next module. The resulting new segment information is relocatable again.

Linking resolves EXTERNAL references, as well as segment relocation. Linking does not affect absolute segments of translator modules.

In general, only relocatable segments of same name and type will be be linked by a linkage process. Segments with the same name but of different type, absolute segments of the same name or segments with different names will not be linked.

Absolute segments without user defined names are treated in a special way. As described in section 5.1.1., the translators create for those segments so called artificial names. As the artificial names, created for different translator modules are equivalent and the linkage of absolute segments with the same name provides errors, the linker assigns to each artificially named

absolute segment a new, also not visible artificial name.

The modified segment and symbol information will be written to the output ROF, together with the revised, remaining data of the input ROFs. The data is modified, depending on the changed symbol/segment information.

Messages concerning the flow of a linkage process appear in the optional output mapfile.

To prevent operator errors, the linker only allows the linking of input modules pertaining to the same processor type. Thus linking of e.g. uPD7809 modules with uPD7811 modules, is not possible.

4.1.3. Handling of Library Modules during the Linkage Proces

In the case that an input ROF is identified as a library, the linker searches for PUBLIC symbols in the library, which can be used to satisfy unresolved EXTERNALS in already bound input modules.

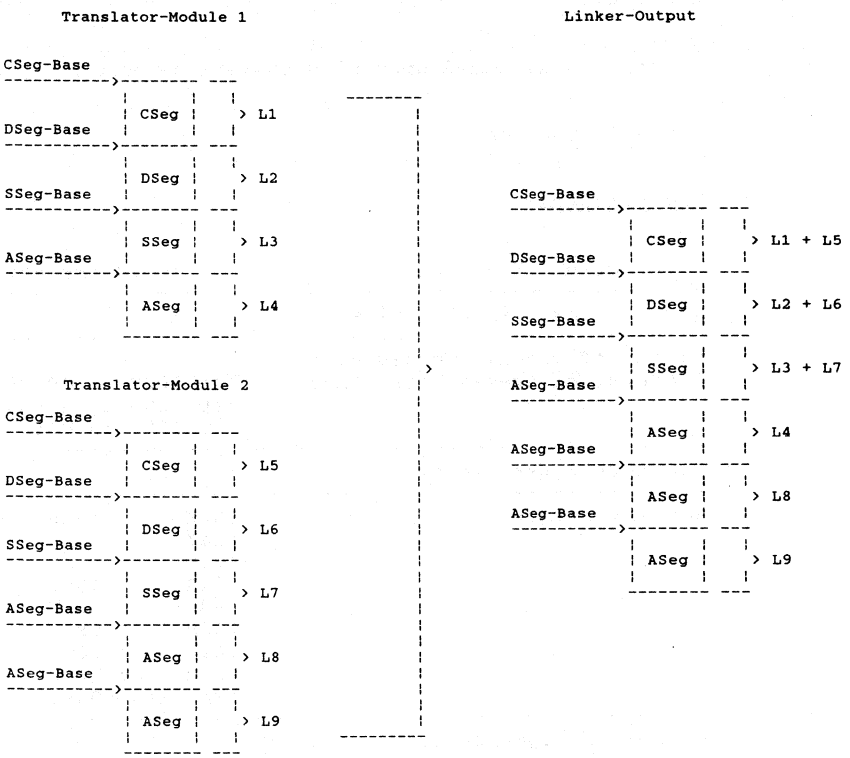
Thus the user can deduce that a library should be linked with input modules, which have references to the library, only after linking of the refering input modules.

In the case where a module within a library has EXTERNAL references, the library will be searched again. This procedure will be continued until no more unresolved EXTERNALS can be found or the current library cannot satisfy any more unresolved EXTERNAL.

The user should bear in mind, that the linker's run time can increase drastically when badly structured libraries are produced. A badly structured library is one, containing many EXTERNAL cross references within the library. During linking a library module will only be linked, when an EXTERNAL exists which refers to a PUBLIC symbol within the module.

A library as the first input file of a link list is nosensical and, therefore, is not accepted by the linker.

4.1.4. Example of the Combination of Segments by Linking



4.2. System Environment

The linker may run under MS-DOS and CP/M86 on IBM PC XT/AT personal computers or compatibles. Further it runs under UDI.

The minimum memory requirement is 128k byte on 16 bit operating systems (UDI, MS-DOS, CP/M86) . (VMS/ULTRIX: t. b. d.)

Common to all named operating systems is and must be:

- Single user os

4.3. Restrictions

Restricted Feature	Restriction
Number of input files	100
Number of global symbols with 6 chars. on average per symbol	ap.2000 on CP/M86 ap.2000 on MS-DOS ap.2000 on UDI
Characters per symbol	32
Segment lenght	64K : ASEG 64K : CSEG 64K : DSEG 64K : SSEG 800H : CSFX 40H : CST0 40H : CST1 FFH : VSEG 2K bits: BSEG
Number of segments with different names and 6 chars. on average per segment name	255

Note: for VAX/VMS or ULTRIX the restrictions have to be defined still.

5. Linker Operation

5.1. Linker Activation

5.1.1. Activation with an Explicit Command Line

Activation syntax:

```
[path]LK310 [path]inputfile <,[path]inputfile> &CR
[TO [path]outputfile] &CR
[MAP/ MA/ NOMAP/ NOMA] &CR
[PRINT ([path]mapfile)/ PR ([path]mapfile)/ NOPRINT/
NOPR] &CR
[NAME (outputmodule)/ NA (outputmodule)] CR
```

Semantics:

- Each separating space may occur one or more times
- The "path" specification depend on operatig system dependent drive or directory assignments
- The filenames "inputfile", "outputfile", "mapfile" must correspond to operating system conventions
- CR means Carriage Return

The activation with an explicit command line can be used alternatively with the activation with an implicit command line (see section 5.1.2.).

The sequence of the controls MAP ... NAME is as desired. All controls and names can be written in upper or lower case letters. Internally all descriptors will be seen as being written in upper case.

If the command line exceeds the max. length of an input line allowed by the current os, the command line can be continued onto the next line after typing "&" (ampersand). The "&" is only possible instead of a seperator and if the current os allows the prolongation. Otherwise the prolongation may cause an error resp., in case of os prevention, it must be done by the use of a command file (see section 5.1.2.).

Seperators are commas in the list of input files or spaces between the commands.

Characters in a line between the "&" and the terminating CR are not considered.

5.1.2. Activation with an Implicit Command Line (Command File)

Activation syntax:

[path]LK310 [path]cmdfile CR

Semantics:

- Each separating space may occur one or more times
- The "path" specification depend on operatig system dependent drive or directory assignments
- The filename "cmdfile" must correspond to operating system conventions and must have the extension ".JOB". "JOB" may be written in upper or lower case letters or any mixture of them.
- CR means Carriage Return

The activation with an implicit command line can be used alternatively with the activation with an explicit command line (see section 5.1.1.).

In using the command file, the linker can get its parameters as described in section 5.1.1. The command file must be created with ASCII based text editors which may not insert any control sequences. The command file must be terminated with a CR, otherwise errors are caused.

A combination of command file and explicit parameters is not allowed and will cause errors.

5.2. Linker Controls

List of Controls

Control	Abbrev.	Default	Meaning
MAP/NOMAP	MA/NOMA	MAP	MAP causes linker map to be created which is output to the print file. The linker map contains information abput segments (size,reloc addr.) error messages etc. The map file format is described in section 2.4. NOMAP inhibits creation of a map and causes error message output on the console.
NAME (modulename)	NA	Module name of the first input file	Definition of the module name of the bound output ROF. The name may have a length of up to 32 characters. The output ROF's file name is not affected.
PRINT (filename)	PR	PR(output-file.MP1)	Definition of the map file name. If NOMAP is given, then PRINT has no effect. The os dependant name of line or console causes output to these devices.
TO (outputfile)	-	TO input-file1.LNK	Definition of the name of the bound output ROF.

Control	Abbrev.	Default	Meaning
NOPRINT	NOPR	PR(output-file.MP1)	If NOPR and MAP occur together then the map file is output direct to console.
cmdfile.JOB	-	-	Definition of a command file.

Notes:

- If any output file name is the same as an input file name, an error message will be caused and linking will be aborted.
- The same happens if the name of the workfile created during the linkage process (name is: outputfile.TMP) is the same as an input file name.

5.3. Startup Message

After activating the linker, the following message is printed on the console:

```
COPYRIGHT (C) 1986 NEC CORPORATION
system-id  UPD78K    LINKER  V x.y                [dd Mmm yyyy]
```

with:
- system-id ..... Identifier of the current os
- V x.y ..... Actual linker version number. "V" may be replaced by "E" if the current version is a preliminary version.
- dd Mmm yyyy ..... Date of creation of the current version.
  - dd ..... Day of creation : 1 ... 31
  - Mmm .... Month of creation : Jan .. Dec
  - yyyy ... Year of creation : 1986 ...

If the linkage process runs correctly, no further message occurs on the console until the end of the process is marked with:

LINK COMPLETE

followed by the current operating system's command mode setup (e.g. > for MS-DOS).

If warnings occur, they will be printed on the console after the startup message.

If an error occurs, the message

PROGRAM ABORTED

followed by a corresponding message will be printed on the console and the linkage process will be terminated.

5.4. List File Format

```
+ COPYRIGHT (C)  1986  NEC CORPORATION                                  Page XX +
+ system-id  UPD78K    LINKER  V x.y              [dd Mmm yyyy]                  +
+                                                                              +
+        INPUT FILES:                    INPUT MODULES                         +
+        file0                           module0                               +
+        file1                           LIBRARY                               +
+        file2                           LIBRARY                               +
+        file3                           LIBRARY                               +
+        file4                           module4                               +
+                                                                              +
+                                                                              +
+        OUTPUT FILE:   fileX                                                  +
+
+
+        CONTROLS SPECIFIED IN INVOCATION LINE:
+
+
+
+               MAP        PRINT       (fileY)       NAME       (moduleZ)
+
+                                                                              +
+                                                                              +
+        LINK MAP OF MODULE  moduleZ                                           +
+                                                                              +
+        LIBRARY  MODULES  INCLUDED:                                           +
+                                                                              +
+        LIBRARY-FILE                    LIBRARY-MODULE                        +
+        file1                           module1                               +
+        file2                           module20                              +
+                                        module21                              +
+                                        module22                              +
+                                        module23                              +
+        file3                           module3                               +
+                                                                              +
+        LOGICAL SEGMENTS INCLUDED:                                            +
+                                                                              +
+        START     STOP        LENGTH    TYPE     SEGMENTNAME                  +
+                                                                              +
+        ------    --------    ------    CSEG     Seg1                         +
+        ------    --------    ------    DSEG     :                            +
+        ------    --------    ------    SSEG     :                            +
+        ------    --------    ------    CSFX     :                            +
+        ------    --------    ------     :       :                            +
+        ------    --------    ------     :       SegN                         +
+                                                                              +
+                                                                              +
+    *** WARNING: UNRESOLVED EXTERNAL NAMES:                                   +
+        symbol  IN    module                                                  +
+                                                                              +
+ LINK COMPLETE                                                                +
```

Notes:

- The version mark "V" may be replaced by "E" in case of linker prereleases
- START, STOP, LENGTH values are byte oriented excepted in combination with the segment BSEG; here the values are bit oriented
- In case of a not user named absolute segment, TYPE will be ASEG, SEGMENTNAME will be empty
- The bordering "+++" is only used for marking, it is not part of the real map file

5.5. Error Messages

5.5.1. Operation Error Messages

The linker recognises the following error classifications within the operation error messages:

- Source LK310 ERRORS (*1)
- Fatal command-tail and control ERRORS (*2)
- Fatal input / output ERRORS (*3)
- Fatal insufficient memory ERRORS (*4)
- Fatal linker failure ERRORS (*5).

In the following overview all error messages are shown, together with their meaning and possible removal.

Each error, independent of its class, causes termination of the current linkage process.

In case of error caused termination the linker must be activated again after the error source is removed by the user.

The operation error messages belong to the format:

```
PROGRAM ABORTED
*** ERROR #xxx, message
```

with xxx ..... error number

Error Nr.	Error class	Error Text	Meaning	Removal
1	*5	FATAL ERROR. CONTACT NEC	OS error message; disallowed file access	If necessary check file name
2	*5	FATAL ERROR. CONTACT NEC	As for 1	As for 1
3	*4	INSUFFICIENT MEMORY	Available system memory too small a) To load LK310 b) To load further GLOBALS in the internal symbol table c) To load ROF records into the corresp. buffer	Increase memory for a) and c) For b) - Link fewer GLOBALS - Increase memory
4	*3	ATTEMPT TO OPEN MORE THAN 6 FILES	ISIS allows only 6 files to be opende simultaneously. This can be overstepped by using multiple nested Submits	Reduce nesting of Submits.
5	*3	ILLEGAL FILENAME: filename	Given "filename" is not allowed under current os	Give allowed file name in a new call
6	*3	ILLEGAL DEVICE: filename	Directory/Drive defined in "filename" is not allowed	Give valid access path in a new call
7	*3	FULL DISK DIRECTORY	Disc directory is full	Insert a new disk or delete some files from the current disk
8	*3	FILE filename ALREADY OPEN	The file "filename" used by LK310, was at time of linker activation not closed correctly	Close the file
9	*3	filename NO SUCH FILE	The user defined file "filename" doesn't exist	Give correct filename or create a corresp. file
10	*3	ATTEMPT TO A WRITE PROTECTED FILE: filename	A file "filename" was, at the linker activation, write protected	Remove write protection on the affected file

Error Nr.	Error Class	Error Text	Meaning	Removal
11	*3	ATTEMPT TO DEL AN OPEN FILE: filename	LK310 tries to delete an opened file "filename"	Close the file by os
12	*3	OVERLAY DOES NOT EXIST	An overlay needed by LK310 is not available	Copy the overlay onto home directory of LK310
13	*3	SYSTEM ERROR	OS error	Reboot the system
14	-	-	-	-
15	-	-	-	-
16	-	-	-	-
17	-	-	-	-
18	-	-	-	-
19	-	-	-	-
20	*3	MODULENAME TOO LONG, FILE: filename	The module name of the file "filename" exceeds limit of 32 characters	Check the affected file
21	*2	INVALID SYNTAX	LK310 activation has been incorrect Syntax	Activate LK310 in the way described in section 2.1.
22	*2	UNRECOGNIZED CONTROL	Control directive in command line is at the wrong place or is not allowed	- " -
23	*2	MAP FILENAME TOO LONG	User given map file name exceeds os dependent limit	Give correct map file name
24	*2	MODULENAME TOO LONG	User given module name exceeds limit of 32 characters	Give correct module name
25	*2	LEFT PARANTHESIS MISSED	The "(" needed as a seperator of map file or module name is missed	Activate LK310 in the way described in section 2.1.
26	*2	LEFT PARANTHESIS MISSED	The ")" needed as a seperator of map file or module name is missed	Activate LK310 in the way described in section 2.1.

Error Nr.	Error Class	Error Text	Meaning	Removal
27	*2	OUTPUT FILENAME TOO LONG	Output file name given by the user exceeds os dependent limits	Give correct output file name
28	*2, *4	COMMAND LINE TOO LONG	Input file names and controls given by the user overflow a corr. buffer	Give fewer input file names or increase system memory
29	*2	DUPLICATE FILENAME	The user defined file name for any output file (ROF, mapfile or workfile) is same as an input file name	Change the name of the output file
30	*2	INPUT FILENAME TOO LONG	User given input file name exceeds os dependent limit	Give correct input file name
31	*2	FILENAME EXPECTED	User didn't input a file name after TO	Activate LK310 in the way described in section 2.1.
32	*3	PREMATURE EOF IN filename	The file "filename" contains false record length. It is shorter than expected	Check file
33	*3	CHKSUM ERROR (record) IN filename	False Checksum in record "record" of file "filename"	Check file
34	*3	RECLEN OF (record) OUT OF RANGE IN filename	The record length in "record" of file "filename" is out of range.	Check file
35	*3	BAD RECORD SEQUENCE (record) IN filename	The record sequence of file "filename" doesn't correspond to the ROF syntax	Check file
36	*3	UNEXPECTED PROCESSORTYPE IN filename	The file "filename" has not been translated for the same processor as the prelinked input files	Check file
37	*1	DUPLICATE ASEG segname IN filename	Multiple use of the same segment name for absolute segments	Change corresponding segment definition in the affected module

Error Nr.	Error Class	Error Text	Meaning	Removal
38	*1	LIMIT OF SEGTAB EXCEEDED	Overflow in memory used for higher order segment table ment Table.	Define fewer segments or give same name to relocatable segments if possible
39	*1	NUMBER OF SEGMENTS EXCEEDED	More than 256 segments with different names found	Reduce number of different named segments
40	*1	SEGMENT segname EXCEEDS LIMIT OF 0XXXXH	The reloc. segment "segname" needs more than the allowed space	Reduce contents of the affected segment by dividing it
41	*1	BAD SEGDEF IN filename	A segment of file "filename" contains invalid segment code	Reassemble/recompile the affected file
42	*1	DUPLICATE SEGMENT segname IN filename	The given segname already exists in file "filename"	Rename the segment
43	*1	NUMBER OF INPUT FILES EXCEEDED	More than 255 input files given for one single linkage process	Fewer Files pro Link Process.
44	*3	BAD RECORD DEFINITION IN filename	The first record of the input file "filename" is invalid	Check file
45				
46	*3	BAD lnrec IN filename	The line record of file "filename" is bad. The line base calculation has caused overflow.	Check file
47	*3	BAD mtrec IN filename	The modtail record of file "filename" is bad. Block nesting of the corresp. input module is incorrect	Check file
48	*3	(record) BAD SEGDEF IN filename	Invalid record "record" in file "filename"	Check file

Error Nr.	Error Class	Error Text	Meaning	Removal
49	*3	BAD glrec IN filename	globrec/ gglobrec is bad in file "filename" It is not at block level 0 of the affected input module	Check file
50	*3	BAD terec IN filename	Tendrec is bad. It is not at block level 0 of the affected input module	Check file
51	*3	BAD cdrec IN filename	Overflow has occured during relocation of the code base address or local references in file "filename"	Check file
52	*3	BAD lcrec IN filename	Overflow has occured during relocation of a local symbol in file "filename"	Check file
53	*3	NUMBER OF LIBRARY MODULES EXCEEDED	More than 256 modules in an input library	Check library
54	*3	ILLEGAL FIXUP IN filename	Invalid fixup in input file "filename"	Check file
55	*3	BAD LIBRARY SYMBOL IN filename	Invalid global symbol in the input library "filename"	Check library
56	*2	LIBRARY OUT OF PLACE	A library occurs in the first position of the input list	Library at earliest second file to be linked
57	*3	BAD LIBRARY RECORD IN filename	The content of LibHeadRec of the library "filename" is bad	Check library
58	-	-	-	-
59	-	-	-	-

5.5.2. System Error Messages

The linker can generate some so called "system error messages".
They only serve the error analysis of the linker software itself.

The sytem error messages have the format:

```
PROGRAM ABORTED
*** ERROR #xxx        with xxx .... system error number
```

Reserved system error messages are:

```
#60:  WORKFILE - PREMATURE EOF
      ... At reading of IROF name from WF, an EOF before
          end of the name was recognised.
#61:  WORKFILE - IROFNAME TOO LONG
      ... Length of IROF name is longer than FnLen
          characers.
#62:  IROF     - MODULENAME TOO LONG
      ... Length of module name is longer than ModLen
          characters.
#63:  IROF     - BAD SEGMENT
      ... Bad segment code of an IROF segment record
#64:  BAD PARSER TABLE
      ... ROF-Parser - Table is badly defined.
#65:  IROF     - ERROR IN RELOCATE CALCULATION
      ... CalcRelAdr: Error during calculation of reloc.
          addresses.
#66:  IROF     - BAD SEGMENT-CODE OF SYMBOL-ELEMENT
      ... CalcRelAdr:  Bad segment code of a global sym-
          bol.
#67:  RelPhase - ERROR IN LINKING SYMBOLS
      ... RelGlRecS : A global symbol is unknown.
#68:  GetVarTxtS - TextNr out of range
#69:  GetTxtS    - TextNr out of range
#70:  MsgPrnS    - MsgNr  out of range
#71:  RelPhase   - ERROR IN LINKING SEGMENTS
      ... RelSgRecS : A segment symbol is unknown.
#72:  RelPhase   - ERROR IN LINKING LIBRARY
      ... RelLhRecS : Library module multiple linked.
```

All system error messages belong to the group of "Fatal Linker Failures ERRORS".

5.6. Warnings

The linker can generate some so called "warnings" in case of errors occuring during the linkage process which don't cause a breaking. These errors belong to mismatches in symbol or module defining.

The user must not remove the warnings. In contrary to the occurence of "real" errors, warnings dont't prevent the creation of an output ROF. This ROF can be relinked or located and finally it can be laoded. But the user should hold in mind that linker warnings may cause an uncorrect program execution if they are not removed.

The warnings have the format:

*** WARNING: message

List of Warnings:

Warning Text	Meaning [output device]	Removal
TYPE MISMATCH OF OPERANDS symname IN: modname	The global symbol "symname" has in module "modname" a type definition different from that in a previous linked module. [Mapfile + Console]	Make equal the type of each symbol occurence.
MORE THAN ONE MAINMODULE. CONFLICT IN: modname	During linking of modules, created by a compiler, more than one main module has been recognised. [Mapfile + Console]	Correct the sources. Only one main module is allowed in a final link.
symname MULTIPLY DEFINED, DUPLICATE IN: modname	The global symbol "symname" is defined PUBLIC in more than one module. [Mapfile + Console]	Correct the sources. One symbol may have the definition PUBLIC only once.
MISSING MAINMODULE	During linking of modules, created by a compiler, no mainmodule has been recognised. [Mapfile + Console]	Correct the sources. One main module of a compiler or one or more "joker" modules (=assembler modules) must be bound.
UNRESOLVED EXTERNAL NAMES:	Header, if one ore more unresolved EXTERNALS occur. [Mapfile + Console]	Correct the sources or link a definition module.
-	-	-

6. LOCxx-Function

LOCxx is available for the translation of Relocatable Object Files (output files from RAxx or LNKxx) into Absolute Object Files. LOCxx locates one I/P file to two (optionally three) Output Files.

Locater-Input File is:

- relocatable Object File from RAxx or LNKxx.
  N.B.An Input File coming from RAxx may not contain any Global Symbols.

Locater-Output Files are:

- an absolute, executable Object File
- an absolute Symbol File
- a formatted Listfile (Mapping) (optional)

6.1. Method of operation of the Locater

6.1.1. Locating a Relocatable Object File

So that an In Circuit Emulator can load and execute a Relocatable Object File, the Relocatable Object File must first be located. The Relocatable Segments must be transformed into Segments with Absolute Addresses. Secondly, the Relocatable Object File must be converted into two Absolute Files; (Absolute Object File and Absolute Symbol File).

The relocatable addresses of Symbols will be added to the defined Segment start address.

The relocatable displacements in Code Record will be replaced by Absolute displacements.

Locating will change the Absolute Segments of a Relocatable Object File.

Messages concerned with Locater Process activity appear on Monitor or can be optionally sent to Output-Mapfile.

6.1.2. Structure of a Relocatable Object File

A Relocatable Object File is created during a Translater or Linker Run. This File contains Module specific Symbols and directives from certain Segments.

These Segments can be divided into two Groups:

a) Absolute Segments:
These Segments are, at translation time,existent at fixed Addresses and cannot, therefore, be further shifted within Physical Memory.

b) The relocatable Segments:
These Segments relate to Symbolic Segment Start Addresses and do not have any fixed position in physical memory before locating. These Segment Start Addresses can be deined at Location time, which results in a shifting of the Segments in Physical Memory.

The following Segment types can be handled differently:

Code-Segment:
It contains all Directives and arrangements, occuring after the Segment directive CSEG.

Data-Segment:
It contains all Directives and arrangements, occuring after the Segment Directive DSEG.

Stack-Segment:
It contains the Stack requirement of each and every Translation Module.

Absolute-Code-Segment:
It contains all Directives and arrangements, occuring within a CSeg after the ORG Directive.

Absolute-Data-Segment:
It contains all Directives and arrangements, occuring within a DSeg after the ORG Directive.

Code-Segment-Fixed:
See Assembler-Specification

Code-Segment-Table0:
See Assembler-Specification

Code-Segment-Table1:
See Assembler-Specification

V-Segment:
See Assembler-Specification

Bit-Segment:
It contains all symbols, occuring after the DBIT-Directive. The segment will be located in the range, which is reserved for bitsegments, by special address handling.

If a Segment, through definition of a Segment Start Address, is shifted outside of the internal memory area; an output occurs with corresponding message. This is only valid, however, for Relocatable Segments which can be freely shifted over the complete Memory Area.

6.1.3. Segment types

The individual Segment types can have the following Addresses and attributes allocated to them.

Proc Fam	Segment-code	Segment name	Segment-type	Valid Area	max. Length
upd87	ACSg = 0	Absolut-Code-Segment	absolute	0 - FFFFH	64K
upd87	ADSg = 9	Absolut-Data-Segment	absolute	0 - FFFFH	64K
upd87	CSeg = 1	Code-Segment	rel.	0 - FFFFH	64K
upd87	CSFx = 2	Code-Segment-Fixed	rel.	800H - 0FFFH	800H
upd87	CST0 = 3	Code-Segment-Table0	rel.	80H - 0BFH	40H
upd87	CST1 = 4	Code-Segment-Table1	rel.	not defined	---
upd87	DSeg = 5	Data-Segment	rel.	0 - FFFFH	64K
upd87	SSeg = 7	Stack-Segment	rel.	0 - FFFFH	64K
upd87	VSeg = 6	V-Segment	rel.	0 - FFFFH	FFH
upd87	BSeg = 8	Bit-Segment	rel.	not defined	---

Proc Fam	Segment-Code	Segment name	Segment-type	Valid Area	max. Length
upd110	ACSg = 0	Absolut-Code-Segment	absolute	0 - 1FFFH	2000H
				FE40H - FFFFH	1C0H
upd110	ADSg = 9	Absolut-Data-Segment	absolute	0 - 1FFFH	2000H
				FE40H - FFFFH	1C0H
upd110	CSeg = 1	Code-Segment	relo.	0 - 1FFFH	2000H
				FE40H - FFFFH	1C0H
upd110	CSFx = 2	Code-Segment-Fixed	relo.	800H - 0FFFH	800H
upd110	CST0 = 3	Code-Segment-Table0	relo.	40H - 07FH	40H
upd110	CST1 = 4	Code-Segment-Table1	relo.	not defined	---
upd110	DSeg = 5	Data-Segment	relo.	0 - 1FFFH	2000H
				FE40H - FFFFH	1C0H
upd110	SSeg = 7	Stack-Segment	relo.	0 - 1FFFH	2000H
				FE40H - FFFFH	1C0H
upd110	VSeg = 6	V-Segment	relo.	not defined	---
upd110	BSeg = 8	Bit-Segment	relo.	FE40H - FF1FH	E0H

Proc Fam	Segment-Code	Segment name	Segment-type	Valid Area	max. Length
upd210	ACSg = 0	Absolut-Code-Segment	absolute	0 - FFFFH	64K
upd210	ADSg = 9	Absolut-Data-Segment	absolute	0 - FFFFH	64K
upd210	CSeg = 1	Code-Segment	relo.	0 - FFFFH	64K
upd210	CSFx = 2	Code-Segment-Fixed	relo.	800H - 0FFFH	800H
upd210	CST0 = 3	Code-Segment-Table0	relo.	40H - 07FH	40H
upd210	CST1 = 4	Code-Segment-Table1	relo.	not defined	---
upd210	DSeg = 5	Data-Segment	relo.	0 - FFFFH	64K
upd210	SSeg = 7	Stack-Segment	relo.	0 - FFFFH	64K
upd210	VSeg = 6	V-Segment	relo.	not defined	---
upd210	BSeg = 8	Bit-Segment	relo.	FE20H - FF1FH	100H

Proc Fam	Segment-Code	Segment name	Segment-type	Valid Area	max. Length
upd310	ACSg = 0	Absolut-Code-Segment	absolute	0 - FFFFH	64K
upd310	ADSg = 9	Absolut-Data-Segment	absolute	0 - FFFFH	64K
upd310	CSeg = 1	Code-Segment	relo.	0 - FFFFH	64K
upd310	CSFx = 2	Code-Segment-Fixed	relo.	800H - 0FFFH	800H
upd310	CST0 = 3	Code-Segment-Table0	relo.	40H - 07FH	40H
upd310	CST1 = 4	Code-Segment-Table1	relo.	8040H - 807FH	40H
upd310	DSeg = 5	Data-Segment	relo.	0 - FFFFH	64K
upd310	SSeg = 7	Stack-Segment	relo.	0 - FFFFH	64K
upd310	VSeg = 6	V-Segment	relo.	not defined	---
upd310	BSeg = 8	Bit-Segment	relo.	FE20H - FF1FH	100H

If a Segment, after Location Process, lies outside the Valid Address Area or is longer than allowed, the Location Process will be aborted and a corresponding Error Message will be output.

Depending on Processor type, Individual Segments have extra Address Default Areas. These Default Areas arise principally from the physical position of the Processor Internal Memory.

Position of the Internal Memory Areas:

Processor-family	Processor-type	Processor-type	internal Memory Area	
upd87 = 0	0	7811 78C11	0000H - 0FFFH FF00H - FFFFH	ROM RAM
upd87 = 0	1	78C14	0000H - 3FFFH FF00H - FFFFH	ROM RAM
upd87 = 0	2	7809	0000H - 1FFFH FF00H - FFFFH	ROM RAM
upd87 = 0	3	7807	FF00H - FFFFH	RAM
upd87 = 0	4	7810 78C10	FF00h - FFFFH	RAM
upd87 = 0	5	frei		
upd110 = 2	2	78112	0000H - 1FFFH FE40H - FEFFH	ROM RAM
upd210 = 3	0	78210	0000H - 0FFFH FE00H - FFFFH	ROM RAM
upd210 = 3	4	78224	0000H - 3FFFH FC80H - FEFFH	ROM RAM
upd310 = 1	0	78310	0000H - 0FFFH FE00H - FFFFH	ROM RAM
upd310 = 1	2	78312	0000H - 0FFFH FE00H - FFFFH	ROM RAM

NEC

If no Segment Start Addresses are defined, then the following default addresses will apply for each Segment type.

Segment	Default Address ProcFam upd87	Default Address ProcFam upd310
Code-Segment	00C0H	80H
Code-Segment-Fixed	0800H	800H
Code-Segment-Table0	0080H	40H
Code-Segment-Table1	---	8040H
Data-Segment	FF00H	FE00H
Stack-Segment	FF00H	FE00H
V-Segment	FF00H	FE00H
Bit-Segment	---	FE20H

Segment	Default Address ProcFam upd110	Default Address ProcFam upd210
Code-Segment	0080H	80H
Code-Segment-Fixed	0800H	800H
Code-Segment-Table0	0040H	40H
Code-Segment-Table1	---	---
Data-Segment	FE40H	FC80H
Stack-Segment	FE40H	FC80H
V-Segment	FE40H	FC80H
Bit-Segment	FE40H	FE20H

6.1.4. Locating the Segments

6.1.4.1. Principles

The Locater can process a max. of 256 different Segments. The segments will be discriminated via Segment names (max. 32 ASCII Characters). At Locater level no Segments may appear with the same name, since Segment names may only be given once.

The Locater is told, via directives CODE, DATA and STACK, at which Addresses the Code-, Data- and Stack Segments are to be located. It is assumed that the corresponding Memory Areas for the Segment types correct physical memory type (RAM,ROM) are given.

From the Start Address defined via CODE, the following Segment types will be located:

- Code-Segment

From the Start Address defined via DATA, the following Segment types will be located:

- Data-Segment

From the Start Address defined via STACK, the following Segment types will be located:

- Stack-Segment

If these Start Addresses are not defined, then the Default-Start Addresses for RAM- and ROM-Memory will be used. If no Start Address is defined for STACK, then the Stack-Segment will be appended to the Data Segments.

The directive GAP in the Locater Call Line, enables the User to define upto 16 Memory Gaps. No Segments may be located by the Locater in these Gaps. A GAP can be a max. of FFFEH long.

Segments, explicity defined by a name and start address, may be located at a desired address via directive SEGMENT. For explicitly defined Segments a check is made to see if they are to be located in disallowed areas, such as Memory Gaps or Memory Areas which are disallowed for a particular Segment. If this is the case, then an Abort with Error Message occurs.

Generally, at location of the Segments, the following Strategy is complied with:

1. Structure of a Memory Map with Memory Gaps.
2. Location of Absolute Segments.
3. Location of explicitly defined Segments.

4. Location of the Segment types with limited allowed Memory Area (Code-Segment-Fixed, Code-Segment-Table0, etc.)
5. Location of the inexplicitly defined Segments into defined general Memory Area.

The sequence of operation within these Classes will be determined by the Link sequence. A memory optimisation is used for the inexplicitly defined Segments. This optimisation locates the current Segment into the smallest memory area available; assuming that this area is allowed for the current Segment. After a Segment has been located, then the Memory Map must be updated to take this into account.

As for explicity defined Segments, a check is made to see if the Segments are to be located in Memory Gaps or disallowed Memory Areas. If this is the case, then an Abort with Error Message occurs.

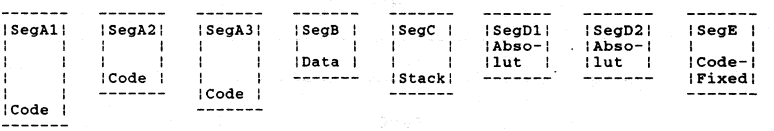
There is no optimisation of Memory positioning via shifting of Segments which have already been located.

6.1.4.2. Example

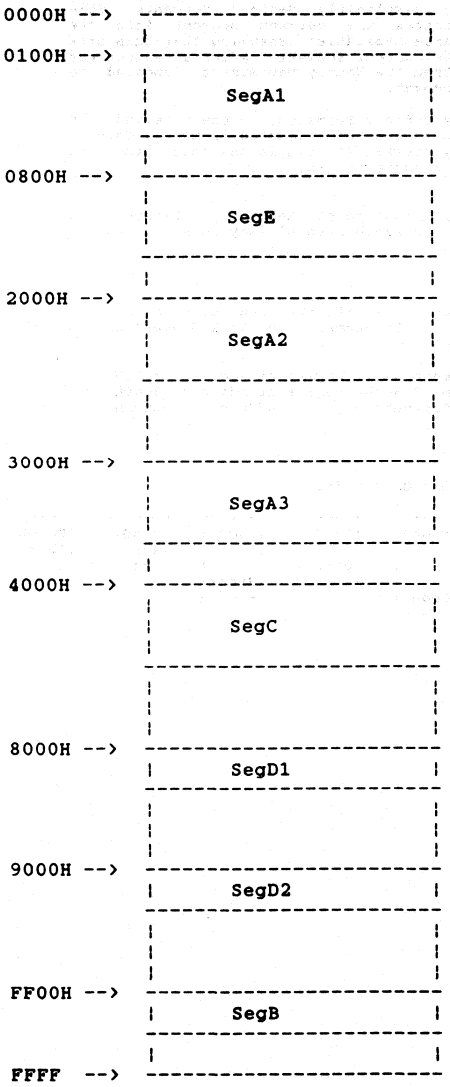
Relocatable Object File with two Absolute-Segments, 3 Code-Segments, a Data-Segment, a Stack-Segment and a Code-Segment-Fixed.

The Absolute-Segment SegD1 begins at Address 8000H,
The Absolute-Segment SegD2 begins at Address 9000H.
The relocatable Segments begin at address 0000H.

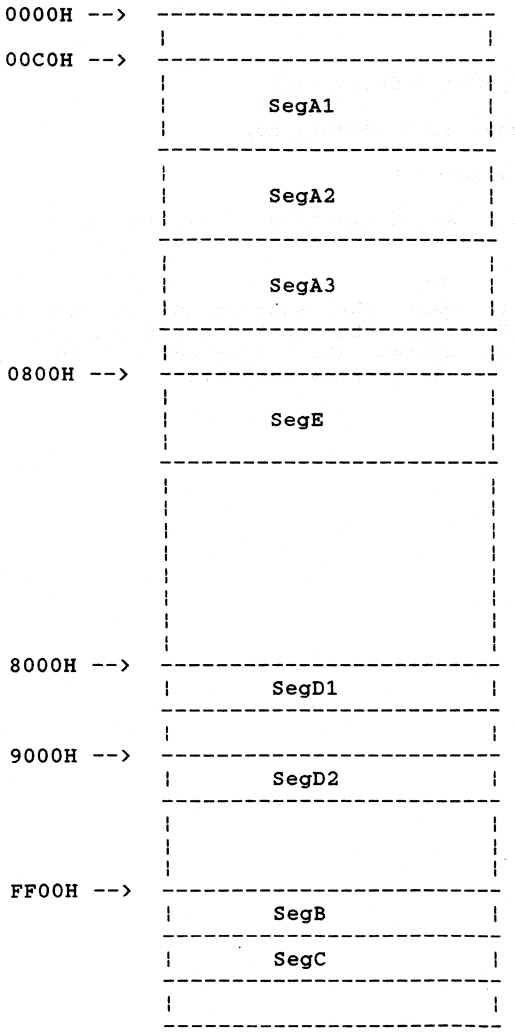
Segments in relocatable Object File:



If the Locater is called with the Call Line LOCxx inputfile SEGMENT (SegA2 2000H, SegA3 3000H, SegC 4000H) CODE (100H) then the Segments will be ordered in physical memory as follows:



If the Locater is called with the Call Line LOCxx inputfile, i.e. no defined Segment Start Addresses, then the following position of Segments in physical Memory will result.



6.2. ystem Environment

OCxx can be run on the system enviroment as the RAxxx.

Commonalities of the operating systems:

- Single-User-Systems

Operating System differences:

- Memory requirement/Memory Map
- Overlay-Management
- availability and definition of System Calls

In order to ease the adaptation to the mentioned Operating Systems, the uCOM87 Utilities will have a corresponding System Shell adapted to each Operating System and its own Overlay Manager.

6.3 Restrictions

Type	LOCxx
Library Process	no
Number of Input Modules/Files	1
Number of Global Symbols	512 pro Module
Number of Characters/Symbol	6
Max. length of a Segment	FFFEH
Number of relocatable Segments in Input-Object-File	256

7. LOCxx-Operation

7.1. LOCxx-Call

7.1.1. Activation with an explizite Command Line

Syntax of the Call Line in BNF:

```
[path] LOCxx [path] inputfile [TO [path] outputfile] &
[MAP/NOMAP] &
[PRINT ([path] mapfile) / NOPRINT] &
[SYMBOLCOLUMNS (nr)] &
[SEGMENT (segname blank segadr <,segname blank segadr>)]&
[GAP (strtadr blank endadr <,strtadr blank endadr>)] &
[STACKSIZE (value)] &
[CODE (strtadr)] &
[DATA (strtadr)] &
[STACK (strtadr)] &
```

Valid Abbreviations of Key Words:

MAP	MA
NOMAP	NOMA
PRINT	PR
NOPRINT	NOPR
SYMBOLCOLUMNS	SC
SEGMENT	SEG
GAP	GA
STACKSIZE	SS
TO	TO
CODE	CO
DATA	DA

input file: A valid Input-File is a relocatable Object-File, generated by Tools RA87 or LNK87.

If a inputfilename without a extension is defined, then the Locater generates automatically the Default Extension "LNK", and the Locater works with inputfile .LNK. If the user wants do supresse this Default Extension Generation, he must finish the inputfilename with ".".
Example:

```
xyz   ---> the Locater works with xyz.lnk
xyz.  ---> the Locater works with xyz
```

The following Filenames are influenced by Input-File

Inputfile	:	inputfilename.extension
absolute Object-File	:	inputfilename.HEX
absolute Symbol-File	:	inputfilename.SYM
Mapfile	:	inputfilename.MP2

output file: An Absolute Object File and an Absolute Symbol File constitute an Output-File. These Files are so structured, that they are considered as Input files for NEC Tools EVAKIT and several In-Circuit-Emulators.

If the output file is to be defined, it must follow the definition of Input-File in the Call Line.

The following Filenames are influenced via output file:

absolute Object-File	:	outputfilename.extension
absolute Symbol-File	:	outputfilename.HEX
Mapfile	:	outputfilename.MP2

For Pathnames, use the valid Pathnames for respective Operating System.

The Filenames inputfile, outputfile, mapfile must correspond to the filename conventions of the respective Operating System.

The Sequence of the Control Commands MAP ... STACKSIZE is as required. All Commands and Names may be written in Upper or Lower Case Characters. Internally all descriptors are seen as being Upper Case Characters.

If the Locater-Call is longer than an input line, then the entry of Symbol "&" enables the continuation of the Call onto the next line.

This continuation is, however, only possible after a complete control command and a valid dividing Character. Dividing Characters are blanks between the Commands.

If other Characters exist between "&" and the ending CR in the line, then an Error Message is output.

7.1.2. Activation with an implicit Command Line (Command File)

Activation syntax:

[path]LOCxx [path]cmdfile CR

Semantics:

- Each separating space may occur one or more times
- The "path" specification depend on operatig system dependent drive or directory assignments
- The filename "cmdfile" must correspond to operating system conventions and must have the extension ".JOB". "JOB" may be written in upper or lower case letters or any mixture of them.
- CR means Carriage Return

The activation with an implicit command line can be used

alternatively with the activation with an explicit command line (see section 7.1.1.).

In using the command file, the locater can get its parameters as described in section 7.1.1. The command file must be created with ASCII based text editors which may not insert any control sequences.
The command file must be terminated with a CR, otherwise errors are caused.

A combination of command file and explicit parameters is not allowed and will cause errors.

7.2. LOCxx Control Commands

List of the Control Commands:

Control directive	Abbrev.	Default	Description
MAP/NOMAP	MA/NOMA	MAP	Control MAP results in the creation of a Map. Control NOMAP inhibits this. unterdrueckt. The Map Filename can be defined via PRINT.
PRINT (filename)	PR	PR (output-file.MP2)/ outputfile-name	Definition of LOCxx-Mapfile. If PRINT is defined without following filename, then Mapfile is given the output-filename MP2. If PRINT (...)is defined in connection with NOMAP, then an Error Message occurs.
NOPRINT	NOPR	no Default	Diverts the LOCxx-Mapfile to the consol output. If NOPRINT is defined in connection with NOMAP, then an Error Message occurs.
SYMBOLCOLUMNS (1/2)	SC	SC (2)	The number of columns for Symbol O/P in Listfile. If SYMBOLCOLUMNS is defined in connection with NOMAP, then an Error Message occurs.
CODE (adr)	CO	Start Addr. of internal ROM.	Start Address of Code Segments
DATA (adr)	DA	Start Addr. of internal RAM.	Start Address of Data Segments
STACK (adr)	ST	Start Addr. of internal RAM.	Definition of lower Stack Limit. SS will be added to STACK. The resulting sum will be used as Stack Base. The Stack Base will be loaded at Program Run Time via Assembler command
STACKSIZE (value)	SS	-	Size of the Stack. If no entry is given, then a Size derived from Module information will be used. If a Stacksize is defined, which is smaller than the Stacksize derived from Module information, then a warning follows.
SEGMENT (segname blank segadr,...)	SEG	No Default	Start Address of Segment defined via Segment name.
GAP (strtadr blank endadr,...)	GA	No GAP	Start and End Addresses of all forbidden areas of memory within Target System.

The following is valid for all output file names (Object File, Map File):

If an O/P File has the same name as an I/P File, or different O/P Files have the same name, then the Location Process will be Aborted and a corresponding Error Message will occur.

7.3. List-File

A List-File is created by Locater, unless this is forbidden via directive NOMAP. The name of the List-File can be defined via PRINT directive. If no name is defined, then the name from the Output Filename, together with extension "MP2", will be used as a default. The Monitor Output may also be defined as List-File via :CO:.

The directives MAP and NOMAP, NOMAP and PRINT and NOMAP and NOPRINT operate exclusively.

7.4. Error Messages

Operation Error Messages

LOCxx recognises the following Error Classes within the Operation Error Messages:

- Source LOCxx-ERRORS (*1)	(Error in LOCxx-Source-Code)
- Fatal command-tail and control ERRORS (*2)	(Error in Call Line)
- Fatal input-/ output-ERRORS (*3)	(Error in Input-File)
- Fatal insufficient memory ERRORS (*4)	(Error due to insufficient memory)
- Fatal locater failure ERRORS (*5)	(Error during Locating)
- WARNINGS (*6)	(Error, which, under certain conditions, can lead to an abortive function.)

In the following Overview, all Error Messages are represented together with their meaning and possible remedy.

Error Number	Error Class	Error text	Meaning	Remedy
1	*5	---	OS-Error Message disallowed access to File.This Error may only occur during LOCxx-Development	If necessary Check Filename
2	*5	---	As in 1	As in 1
3	*4	INSUFFICIENT MEMORY	Available System Memory too small to: a) To load LOCxx b) To load linked GLOBALS in the internal Symbol-table. c) To load ROF-Records in the Record Buffer	For a) und c): - Increase Memory For b): - Link fewer Global Symbols - Increase Memory
4	*3	ATTEMPT TO OPEN MORE THAN 6 FILES	ISIS allows only 6 Files to be opened simultaneously. Files ist ueberschrit-ten. This can occur due to multiple nesting of Submits in LOCxx.	Reduce Nesting of Submits.
5	*3	ILLEGAL FILE-NAME	Filename is not allowed under OS.	Enter allowed file name in new Call.
6	*3	ILLEGAL DEVICE-SPECIFICATION	Directory/Drive defined in filename is not allowed.	Enter valid access path in new Call.
7	*3	NO MORE ROOM IN DISK DIRECTORY	Disc directory is full	Insert new Disc or delete some Files from current Disc.
8	*3	ATTEMPT TO OPEN FILE ALREADY OPEN	A File used by LOCxx was,at LOCxx activation incorrectly closed.	Close File.
9	*3	NO SUCH FILE	The user defined File doesn't exist.	Enter correct filename or create new File.

Error Number	Error Class	Error text	Meaning	Remedy
10	*3	ATTEMPT TO DELETE OR OPEN FOR WRITING A WRITEPROT. FILE	A File used by LOCxx was,at LOCxx activation write protected.	Remove write protection on File.
11	*3	CAN'T DELETE AN OPEN FILE	OS Error Message should only occur during development.	Close File per LOCxx Software.
12	-	-	-	-
13	-	-	-	-
14	-	-	-	-
15	-	-	-	-
16	-	-	-	-
17	-	-	-	-
18	-	-	-	-
19	-	-	-	-

Error Number	Error Class	Error text	Meaning	Remedy
20	*2	SYNTAX ERROR	Syntax Error in the Call Line.	Check the Call Line using BNF-Description.
21	*5	SEGMENT segname OUT OF RANGE	At location of Segment Addresses, the allowed segment-address-area has been overstepped	Renew Locater Run(or for Absolute Segments Translator-Run) with other start Addresses.
22	*2	SEGMENTNAME ERROR	Invalid Segment name entered in Call Line. e.g.Name too long.	Check Segment names after SEGMENT
23	*2	MORE THAN 256 SEGMENTS DEFINED	More than 256 Segments defined in Call Line.	Check Call Line. Link some Segments together using Link Process.
24	*2	INVALID GAP	Invalid GAP defined in Call Line. e.g. Start Address bigger than End Address	Check Call Line.
25	*4	AVAILABLE MEMORY TO SMALL	The Memory available is too small.	LOCxx cannot be run on this Computer.
26	*3	UNKNOWN PROCESSORTYPE	Invalid Processor type in Segment-Record.	Check Input File.
27	*6	SEGMENT segname OUT OF INTERNAL RAM/ROM	Segment is larger than internal Memory or Segment will be located outside the internal Memory.	Renew Locater Call with other Start Addresses.When external Memory is available; no operation error.
28	*2	DOUBLE CONTROL	A Command has been defined twice in Call Line.	Enter correct Call.
29	*5	CHECKSUM ERROR	Error during read of Input file.	Check File.
30	*5	CANNOT ALLOCATE SEGMENT segname AT SPECIFIED ADDRESS	Segment has been allocated to a general (or specific to this-Segment), forbidden physical Memory Area.	Allocate Segment to a valid Memory Area via a new Locater Run.

Error Number	Error Class	Error text	Meaning	Remedy
31	*5	INVALID MODCODE IN SEGMENTRECORD	Error at reading of the MODCODE-Coding in SEGREC	Check File
32	*6	SIZE OF SYMBOLRECORDS DECREASED	Due to a lack of available Memory space in working Memory, not all specified Symbols can be processed.	LOC can only be run in limited fashion on this Computer.
33	*5	MAX BLOCK-DEEPTH ECCEEDED	Error in structure of Input file: Nesting depth too large.	Check File, especially Translator Source-file.
34	*5	CODERECORD CONTAINS NO CODE	Erroroneous Coderecord in Input file	Check File
35	*5	BAD RECORD SEQUENCE	The Input File sequence doesn't correspond to the specified ROF-Syntax	Check File.
36	*1	PARSETAB ERROR	System Errorin Parser-Table	Please contact NEC (or company AiD)
37	*5	PREMATURE EOF	The Input file contains false Record Info. It is shorter than the given length.	Check File.
38	*5	RECORD OUT OF RANGE	A Record in Input file is out of range.	- " -
39	*3	BAD RECORD-DEFINITION	Invalid RECDEF- Coding in Input file	Check Input File.

Error Number	Error Class	Error text	Meaning	Remedy
40	*6	SEGMENT segname OVERLAP AT adr1, adr2	At location more than one Segment has been allocated to physical Memory Area from Addr 1 to Addr 2.	If this not desired, make new Locater Call with other Start Addresses.
41	*6	INPUTFILE INCLUDES NO MAINMODULE	An Input file containing no Main Program is to be located.This leads to,in normal case to in executable Code.	Renewed Translator Run or Linker Run.
42	*6	INSUFFICIENT STACKSIZE DEFINED	The Stacksize transfered from Module Info. is larger than the defined Stacksize.	This isn't necessarily an Error, since the user can generally asses the Stacksize more accurately than via Translator calculations.
43	*3	MIUSED SYMBOL	Erroneous Bit operation	Renewed Translator Run
44	*2	BASE NOT ALLOWED	Anumber has appeared in Locater Call, having an invalid number base.	Valid number bases: H : 16 D : 10 O : 8 B : 2 If no number base is defined, then the default number base is 10.
45	*2	INDIFFERENT FILENAMES DEFINED	Meaningless Filenames defined in Call.	Check Call. Default Filename extensions are HEX, SYM u. MP2

Error Number	Error Class	Error text	Meaning	Remedy
46	*3	SEGMENT NOT FOUND	Erroneous Segment-record in Input file.	Check File
47	*3	SYMBOLADRESS OUT OF RANGE	At location of Symbol Addresses an Overflow has been found.	Check File
48	*2	INVALID NUMBER DEFINED	In Call, a number has been defined, which is invalid for the defined number base.	Check Call and renewed Call.
49	*2	DEFINED ADDRESS OUT OF 64k	A Segment Base Address has been defined in Call, which is longer than FFFFH.	- " -
50	*5	INVALID CODERECORD	Erroneous Coderecord in Input file.	Check File
51	*5	INVALID FIXUPRECORD (ADR)	Erroneous Fixup-record in Input file.	- " -
52	*5	INVALID FIXUPRECORD (MODE)	Erroneous Fixup-record in Input file.	- " -
53	*5	INVALID FIXUPRECORD (SYM)	Erroneous Fixup-record in Input file.	- " -
54	*5	INVALID FIXUPRECORD (SEG)	Erroneous Fixup-record in Input file.	- " -
55	*2	DELIMITER NOT ALLOWED	Erroneous Delimiter after Call Command. e.g. Bracket missing.	Check Call and renew Call.
56	*2	UNKNOWN KEYWORD	Invalid Key Word defined in Call.	- " -
57	*2	CONTRARY CONTROL DEFINED	Contrary Controls defined in Call. e.g. MAP, NOMAP.	- " -
58	*2	FILENAME ERROR	A Filename has been defined in Call, which doesn't correspond to OS conventions.	- " -

Error Number	Error Class	Error text	Meaning	Remedy
59	*2	DEFINED SYMBOLCOLUMNS NOT ALLOWED	An invalid number of Symbol columns has been defined in Call.	Valid Symbol columns are: 1, 2 Check Call and renew Call.
93	*5	ILLEGAL CALLF-SYMBOLADDRESS	CallF-instruction with symbol, which is located outsides the CallF-area	Locate the symbol in the CallF-area
94	*5	ILLEGAL CALLF-OFFSET	CallF-instruction with Offset outsides the CallF-area	Locate the symbole, that the symbol-address in addition with the instruction-offset results a CallF-offset, which is located in the CallF-area
95	*5	ILLEGAL CALLT-SYMBOLADDRESS	CallT-instruction with symbol, which is located outsides the CallT-area or which has a odd address	Locate the symbol in the CallT-area at a even address
96	*5	ILLEGAL CALLT-OFFSET	CallT-instruction with Offset outsides the CallT-area	Locate the symbole, that the symbol-address in addition with the instruction-offset results a CallT-offset, which is located in the CallT-area
97	*3	INVALID SEGMENT-INDEX	Symbol refer to a not existing segment	Check input-file
98	*2	CANNOT RELOCATE ABSOLUTESEG	A start Address has been defined for an Absolute Segment.	Check Command Line Check Segment Type
99	*2	ILLEGAL CODE-DATA- STACK-CONFIGURATION	The same Start Address has been defined for CODE as for DATA- or STACK-Area.	Check Command Line

8. List-File-Format

The List-File (LMF) is a printable formatted ASCII-File. The generation of this dieses File can be inhibited by entry of NOMAP Command. With Command PRINT (Filename) the Filename can be defined.

The structure of the List-File is described as follows:

```
Copyright (C) 1986 NEC Copporation                                  Page xxx
procfamstring LOCATER  Vx.x  [xx  Xxx  86]

          INPUTFILE:      inputfilename
          OUTPUTFILES:    outputfilename1
                          outputfilename2
          CONTROLS SPECIFIED IN INVOCATIONLINE:
          PRINTFILE:      printfilename
          SYMBOLCOLUMNS:  number
          MAP:
          CODE:           adr
          DATA:           adr
          STACK:          adr
          STACKSIZE:      value
          SEGMENTE:       segmentname1, segmentaddress
                          segmentname2, segmentaddress
                              :
                              :

          GAPS:           strtadr1, endadr1
                          strtadr2, endadr2


          UNRESOLVED EXTERNALS:
          symbolname1
          symbolname2
             :
             :
```

```
SYMBOLTABLE OF MODULE: mainmodulname                              PAGE xxx

ADDRESS TYPE SEGIND SYMBOLNAME        ADDRESS TYPE SEGIND SYMBOLNAME
xxxxH.x xxx  xxx    xxxxxxxxxxx       xxxxH.x xxx  xxx    xxxxxxxxxxx


MODULE modulname                                                  PAGE xxx

ADDRESS TYPE SEGIND SYMBOLNAME        ADDRESS TYPE SEGIND SYMBOLNAME
xxxxH.x xxx  xxx    xxxxxxxxxxx       xxxxH.x xxx  xxx    xxxxxxxxxxx


      MEMORY-MAP OF MODULE: mainmodulename                        PAGE xxx

      START    STOP     LENGTH    SEGTYP    SEGNAME
      xxxx     xxxx     xxxx      x         xxxxxxxxxx
```

9. The Librarian

In this chapter the librarian utility is explained. The whole chapter refers only to the LB310. But the same function and operation rules are valid for the LB210, LB110 or LIB87 utilities.

9.1. LB310 - Function

Libraries are a tool for modular programming. In principal, a Library consists of a number of relocatable or absolute object files. Furthermore a Library contains an "Index" of the object files with their associated declared Global Symbols (PUBLICS).

For the creation and management of Libraries a Library-Manager is used. The Library-Manager copes with the task of bringing new relocatable or absolute object files into a new or existing Library.

LB310 is a Library-Manager which uses LK310 specified relocatable Object files,(abbreviated to ROF), as Input files and orders them in a Library as an Output file.

The Input files as well as the Library Output file correspond to the ROF Syntax Format.

The LB310 can optionally produce a List file for documentation of a Library. During an LB310 process a Work file is produced and finally deleted. The Work file is given the name of a Library Output with the extension "TMP".
User files with the same name and extension will be destroyed.

9.1.1. Library creation process

The result of a Compilation or Assembly process on a User source module is an ROF (Relocatable Object File). In the ROF all module specific entries for Symbols, Segments and Instructions are contained, as well as pointers between different entries.

An ROF of similar content is produced by a Link process. In the Link ROF the information from original module specific ROF's are gathered together in the form of an "Overall module".

As already described, LB310 can attach one or more ROF's to a Library. For the LB310, because of above mentioned reasons, it is irrelavant if the ROF to be attached to a Library is the result of a Compiler, Assembler or Link process. It is worth mentioning, that for later use of the Library the module concept will be dependent on the character of the ROF to be attached to the Library.

- A Compiler or Assembler ROF would be defined by the User as a self contained Module and would be handled as such in the Library.
- A number of Compiler or Assembler ROF's, which have been linked together via a Link process, lose their user defined self contained Module character.
  The linked ROF's will be handled as a Module in the Library.

For Library creation the Library Manager is told which ROF's are to be attached to which Library. The Library-Manager opens an existing Library File or, if necessary, generates a new Library file.

In the Library file every Input ROF will be completely accepted. Furthermore, the Module name of the Input-ROF and it's position in Library file will be entered in a "Library Module Index". All the Global Symbols(PUBLICS) belonging to Input ROF will be entered in a "Library Symbol Index".

After the transfer of the current Library Module/Symbol Indexes in the Library file, the Library creation is ended and the Library file is then closed.

If, during the creation of Library Module/Symbol Indexes two Modules/Symbols of the same name are recognised, then a corresponding Message occurs.

9.1.2. Using a Library

Compared with a Linker Output ROF a Library, at first glance, is very similar. In principal both files contain a collection of ROF's and higher order information.

There is, however, a significant difference:-

A Linker Output ROF contains higher order Module Information which masks individual modules; the Library file contains only higher order Index information,which does not mask the individual module.

This difference leads to a different handling of Linker/Library files. It is true that both types of files can be linked with other ROF's during a Link process. However, whereas a complete Linker file would be linked, only those modules from a Library file will be linked, whose Global Symbols are referenced by other, already linked, ROF's.
References to a Library, which occur after Library in the Link sequence, will not be taken into account; i.e.the user must ensure that for the Link call, that the Library is input after all ROF's which make reference to the Library.

References within a Library are resolved independently of the sequence of Library generation. However, for a more efficient Link process during Library creation,the user should minimise the number of Cross references occuring within a Library.

9.1.3. System Environment

The Librarian may run under MS-DOS and CP/M86 on IBM PC XT/AT personal coputers or compatibles. Further, it runs under UDI. The minimum memory requirement is 128k byte for 16bit operating systems (UDI, MS-DOS, CP/M86). (For VMS/ULTRIX: t. b. d.)

Commonalities for all named Operating systems are:

-Single user O.S.

9.1.4. Limitations

Type of limitation	Limitation
number of modules per Library with an average of 6 characters per module name.	256
number of exported Symbols (PUBLICS) with an average Symbol length of 6 Characters.	512

9.2. Using LB310

For the description of the elements of LB310 successive access to the Syntax representation of the BNF will be made. The following Syntax elements will be used:

[]	:	0- or 1-times repeat of the bracketed expressions.
<>	:	0- or n-times repeat of bracketed expressions
/	:	Alternative of expressions
,	:	Sequence of expressions

Upper Case letters sequence: Terminal-Symbols

Lower Case letters sequence: Non Terminal Symbols

9.2.1. LB310 Call

Call Syntax:

[path]LB310

LB310 gives message at top left of Terminal with:

```
sysid   uPD78K  LIBRARIAN  Vx.y                [dd  Mmm
yyyy]
>
```

dd Mmm yyyy means date of generation of Vx.y

After the Prompt sign >, the User can activate one of the functions CREATE, ADD, DELETE, LIST or EXIT with the specific Syntax. After execution of a function, LB310 outputs prompt > again. If the function activation is incorrect or if during the execution of the function an error occurs, then a corresponding message will be output.

9.2.2. LB310 Control Commands

The Control Commands can always be input, after LB310 activation, when the Prompt > appears.

9.2.2.1. CREATE

Syntax:

CREATE [path]libfilelname CR

with:
- CREATE = Control Command for generation of a new Library file
- path = Operating System dependant access path
- libfilename = Name of the new Library file. If a file with the same name already exists, then a corresponding message will be output and the command will not be executed.
- CR = Carriage Return, ends command.

Function: CREATE enables the generation of a new Library. The Library is empty. ADD command can be used to attach modules to a Library.

9.2.2.2. ADD

Syntax:

ADD [path]filename [(modname <,modname>)] TO [path]-libfilename CR

with:
- ADD = Control Command to append Modules to an existing Library
- path = Operating System dependant access path
- filename = Name of an Input file, which must contain at least one module. The Input file can be either Compiler, Assembler or Linker ROF or a Library file. In the case of an ROF the entry of a Module name is not allowed; a corresponding Error Message results. The ROF will be totally transferred to the Library under it's Module name. If the Input file is a Library file, then with modname the Modules from an existing Library are specified for transfer to the current Library. If a specified Module does not

exist, then a corresponding message is output. If no modname is given, then the complete existing Library will be appended to the current Library.

- modname = Name of the Module of an existing Library, which is to be appended to the current Library.
- TO = Control directive
- libfilename = Name of the current Library
- CR = Carriage Return, ends command.

Function: ADD is used to append new Modules to an existing Library.
The Library Indexes for Module names and Symbols will be updated.
If, with ADD command, modules are to be added to the Library, which have a module name or exported Symbol name already existing in the Library, then a corresponding message occurs and the command is aborted. A Message, with Abort, also occurs when the maximum allowed number of modules or exported symbols is overstepped.

When transferring the Modules of an existing Library into the current Library, only those Modules will be transferred which do not already exist in the current Library. Modules which already occur in the current Library will not be transferred and a message will occur to this effect.

9.2.2.3. DELETE

Syntax:

DELETE [path]libfilename (modname <,modname>) CR

with:
- DELETE = Control command for deletion of modules in an existing Library
- path = Operating System dependant access path
- libfilename = Name of the existing Library
- modname = Name of the module to be deleted
- CR = Carriage Return, ends command.

Function: DELETE is used to delete Modules from an existing Library and the Library Indexes for Module names and export Symbols will be updated.

If the user gives Module names which do not appear in the Library, then a corresponding message to this effect, will occur.

9.2.2.4. LIST

Syntax:

```
LIST [path]libfilename [(modname <,modname>] TO &
[path]mapfilename [PUBLICS]
```

- LIST = Control command for output of a Library-Index of an existing Library
- path = Operating System dependant access path
- libfilename = Name of the existing Library
- modname = Module names to be output. If no modname is entered, then all module names will be output.
- TO = Control directive
- mapfilename = Name of the output file
- PUBLICS = Directive, specifying that all export symbols of the named modules are to be output.

Function: LIST enables the contents of Libraries to be checked. The names of modules within a Libraryas well as the export Symbols of a respective Module can be output.
If the Control directive TO mapfilename is not entered, the output appears on the Console.

If the user enters Module names which do not appear in the Library, a corresponding message occurs.

Listformat:

```
COPYRIGHT 1985 NEC ELECTRONICS EUROPE GmbH                    page xx
       sysid  uPD78K      LIBRARIAN Vx.y      [dd mmm yyyy]


                MAPPING OF LIBRARY ABCDEF
                MODUL1
                          SYM10
                          SYM11
                MODUL2
                          SYM20
                          SYM21
                          SYM22
```

9.2.2.5. EXIT

Syntax:

EXIT

Function: Control command for exit from LB310. Control is handed back to the Operating System.

9.3.2. LB310 Error Messages

9.2.3.1. User Error Messages

Since LB310 is an interactive Program, user errors result only in an error message and a Function Abort but there is no LB310 Abort.

An exception are Error Messages referring to Memory requirement; they cause an LB310 Abort.

The following Error Classifications can occur:

- Source LB310-ERRORS (*1)
- Fatal command-tail and control ERRORS (*2)
- Fatal input-/ output-ERRORS (*3)
- Fatal insufficient memory ERRORS (*4)
- Fatal library failure ERRORS (*5).

In the following Overview all Error Messages, with hints to their meaning and possible remedy, occur:-

Error Number	Error Class	Error Text	Meaning	Remedy
1	*5	FATAL ERROR. CONTACT NEC!	O.S.Error Message; Invalid File access	Check Filename
2	*5	FATAL ERROR. CONTACT NEC!	as 1	as 1
3	*4	INSUFFICIENT MEMORY	Insufficient System memory for:- a) Loading LB310 b) To load the linked GLOBALS in the internal Symbol table c) To load ROF-Records in Recordbuffer.	For a) and c): - Increase memory For b): - Link fewer Globals -Increase memory
4	*3	ATTEMPT TO OPEN MORE THAN 6 FILES	ISIS allows a max.of 6 files to be open simultaneously. This can occur due to multiple nesting of LB310 in Submits.	Decrease Submit Nesting.
5	*3	ILLEGAL FILE-NAME: filename	Entered filename is Invalid.	Enter valid file name in a new Call.
6	*3	ILLEGAL DEVICE: filename	The Directory/Drive defined in filename is Invalid.	Enter valid access path in a new Call..
7	*3	FULL DISK DIRECTORY	Directory of Disc is full.	Insert new Disc or delete some files on current Disc.
8	*3	FILE filename ALREADY OPENED	A file used by LB310 was,at LB310 activation, not closed correctly.	Close File filename.
9	*3	filename NO SUCH FILE	The User defined File filename does not exist.	Enter correct filename or create File.

Error Number	Error class	Error text	Meaning	Remedy
10	*3	ATTEMPT TO A WRITE PROTECTED FILE: filename	A File used by LB310 was,at LB310 activation Write protected.	Remove Write protection from file.
11	*3	ATTEMPT TO DEL AN OPEN FILE: filename	O.S. Error Message; should only occur during development.	Close File via LB310 S/W.
12	*3	OVERLAY DOES NOT EXIST	An Overlay needed by LB310 is not at hand.	Copy overlay onto Home Dir. of LB310
13	*3	SYSTEM ERROR	O.S.Error	Re-boot
14	*3	FILE filename ALREADY EXISTS	The File filename to be generated,already exists.	Delete existing File or change filename.
15	-	-	-	-
16	-	-	-	-
17	-	-	-	-
18	-	-	-	-
19	-	-	-	-
20	*3	MODULENAME TOO LONG, FILE: filename	The module name given in File filename is longer than Modlen (= 32) Characters.	Check the tool used to create File filename
21	*2	INVALID SYNTAX	Command line from LB310 does not correspond to defined Syntax.	Call LB310 using valid Syntax.
22	*2	UNRECOGNIZED CONTROL	Control directive in the Command section is not in correct place or is invalid.	Call LB310 command using valid Syntax.
23	*2	MAP FILENAME TOO LONG	The Map Filename given by the User is too long.	Enter correct Map filename.
24	*2	MODULENAME TOO LONG	The module name given by the User is too long (> 32 Chars.)	Enter correct module name.
25	*2	LEFT PARANTHESIS MISSED	The "(" needed when giving module name is missing.	Call LB310 Command using correct Syntax.

26	!	*2	!RIGHT PARANTHESIS MISSED	! The ")" needed when giving module name is missing.	! Call LB310 Command using correct Syntax.

Error Number	Error class	Error text	Meaning	Remedy
27	*2	LIBRARYNAME TOO LONG	The Library file name given by user is too long.	Enter correct Library file name.
28	*2, *4	COMMAND LINE TOO LONG	The module names and control directives given by user do not fit in the Command line buffer.	Enter fewer module names or increase System memory.
29	*2	DUPLICATE FILENAME	The file names entered by the User are the same.	Change the file name.
30	*2	INPUT FILENAME TOO LONG	Input filename entered by User is too long.	Enter correct Input filename.
31	*2	FILENAME EXPECTED	User did not enter file name after TO Command.	Call LB310 command with correct Syntax.
32	*3	PREMATURE EOF IN filename	The File filename contains false Record length Info.It is shorter than given in the last length read.	Check File.
33	*3	CHKSUM ERROR (record) IN filename	In the Record record of File filename,the Checksum is false.	- " -
34	*3	RECLEN OF (record) OUT OF RANGE IN file-name	The length of Record record in File filename lies outside the defined Area.	- " -
35	*3	BAD RECORD SEQUENCE (record) IN filename	The Record sequence of File filename does not correspond to the Syntax defined in the ROF.	Check File
36	*3	UNEXPECTED PROCESSOR-TYPE IN filename	The File filename has not been translated for the same Processor as the previously linked Input-Files.	Check File

Error Number	Error class	Error text	Meaning	Remedy
37	*2	filename IS NOT A LIBRARY	The Library specified by the User via file name does'nt exist.	Create Library or correct file name
38	*2	DUPLICATE MODULENAME	The User gave repeated Module name at Command Call.	Call LB310 command with correct Syntax.
39				
40				
41	*1	MODULE modname ALREADY IN LIBRARY	The Module modname of Inputfile is already in the work Library.	Delete module from Library before new linkage.
42	*1	SYMBOL symname ALREADY IN LIBRARY	The Export Symbol sym-name is already in the work Library.	Possibly delete assoc-iated module from Library.
43	*2	CHARACTER "x" NOT ALLOWED	At Command Call the User entered an Invalid Character.	Call LB310 command with correct Syntax
44	*2	TO EXPECTED	The User forgot TO at Command Call.	Call LB310 Command with correct Syntax
45	*2	PUBLICS EXPECTED	At Command Call the User entered an invalid Input;PUBLICS was expected.	Call LB310 Command with correct Syntax
46	*2	MODULENAME EXPECTED	User failed to enter Module name after"(" or ",".	- " -
47	*3	lnarec OUT OF RANGE	LibNameRecord of Work Library is too long.	Check Library file.
48	*3	llcrec OUT OF RANGE	LibLoc Record of Work Library is too long.	Check Library file.
49	*3	ldrec OUT OF RANGE	LibDic Record of Work Library is too long.	Check Library file.

Error Number	Error class	Error text	Meaning	Remedy
50	*3	RECLEN OUT OF RANGE IN filename	The Record in File filename is too long or too short.	Check File
51	*3	BAD LIBRARY RECORD IN filename	The LibHeadRec in File filename is erroneous.	- " -
52	*4	NUMBER OF LIBRARY MODULES EXCEEDED	The max. number of Modules to be appended to the Work Library (max.=256)has been overstepped.	Delete unwanted modules from Library or create second Library.
53	*3	CHKSUM ERROR OF LIBRARY RECORD IN filename	The Checksum in Lib-HeadRec of File file-name is erroneous.	Check File.
54	*4	NUMBER OF SYMBOLS EXCEEDED EXCEEDED	At linkage of new Modules,the max. number of Symbols(=512)was overstepped.	Delete unwanted modules from Library or create second Library.
55	-	-	-	-
56	-	-	-	-
57	-	-	-	-
58	-	-	-	-
59	-	-	-	-

9.2.4. LB310 Warnings

The Librarian can generate some "Warnings"in case of Errors which do not cause an Abort.These Errors are due to mismatches in module definition.

The User doesn't have to remove Warnings. Contrary to the occurence of "Real" Errors, warnings don't prevent creation of Output Library. The user should keep in mind, however, that the warnings may cause an incomplete Library.

The Warnings have the Format:

Warning Text	Meaning (Output device)	Remedy
MODULE modname NOT IN LIBRARY	The module "modname" not in the current I/P Library.(Console).	Correct the module/ Library name or put module to I/P Library
MODULE modname ALREADY IN LIBRARY	The module "modname" is already in the current Library.(Console).	Correct the module name or stop Library creation with that module.
LIBRARY libname is EMPTY	The library"libname" doesn't contain any module.(Console).	Stop deleting of modules from current Library.

Appendix A An Assembler Example using RA310

Shown here is an example of RA310 assembler package operation using

RA310	assembler
LK310	linker
LC310	locator

The given example has been chosen for uPD78312 microcomputer of the ucom-78KIII family. The uPD78312 is a 16 bit micro,thus all types of segment definitions and instructions for a 16 bit microcomputer can be shown.

The example demonstrates:

a) Invoking RA310 with control commands embedded in the source module body and the assembler call.

b) Structure of assembler source modules.

c) List output of assembled source files with cross reference listing and segment information per module.

d) Invoking LK310 to link the different object modules created by RA310 with a bulk map.

e) Invoking the Locator LC310 to make an absolute module with Hex-format files and the symbol files.

   The Hex file [and the symbol] files can be downloaded to a hardware debugger like an EVAKIT or In-Circuit Emulator for debugging. For uPD78312 the In-Circuit-Emulator IE-78310-R is available as hardware debugger.

```
RA310 INITPC.ASM
RA310 PWM.ASM
RA310 DATA.ASM

LK310 INITPC.REL,PWM.REL,DATA.REL TO DEMO.LNK MAP NAME(MOD_DEMO)

LC310 DEMO.LNK TO DEMO.HEX GAP (2000H 0FDFFH) SEGMENT(BUFFER 0FE20H) CODE(80H) DATA(0FE00H) STACK(0FE00H) STACKSIZE(01FH)
```

```
COPYRIGHT NEC CORPORATION 1985, 1986
UPD78312 ASSEMBLER V2.3 INITIALIZATION OF COMPUTER                    DATE 23-05-86  PAGE     1
INITIALIZATION OF uPD78312
SOURCE FILE: INITPC.ASM
OBJECT FILE: INITPC.REL
COMMAND    : RA310 INITPC.ASM


     ASSEMBLE LIST

 STNO  ADRS R OBJECT     M I  SOURCE STATEMENT

    0                         $PROCESSOR (312)
                              $XREF
                              $DEBUG
                              $TAB(8)
                              $PAGELENGTH(60)
                              $PW(132)
                              $ERRPRN (CON:)
                              $OPTIMIZE (5)

                              $TITLE ('INITIALIZATION OF COMPUTER')
                              $SUBTITLE ('INITIALIZATION OF uPD78312')

                              $DATE ('23-05-86')


    1                                 NAME INITPC

                              ;AUTHOR: H.T. NEC EE APPLICATION

    2                                 PUBLIC LOOP, START, RESET       ;PUBLIC LABELS
    3                                 PUBLIC SPSTRT, SBINIT, INITRB   ;PUBLIC EQU SYMBOLS
    4                                 EXTRN  PWMSRT                   ;16 BIT SYMBOL OF EXT. PWM-ROUTINE
    5                                 EXTBIT LED                      ;BITNAME OF EXT. MODULE PWM-ROUTINE
                                                                      ;EXTBIT ONLY IN SADDR AREA POSSIBLE


                              ;COMMENT: 1. INPUT PRIMARY CONTROLS (XREF, DG, TITLE ETC. WITH NO COMMENTS)
                              ;         2. INPUT NAME OF MODULE (E.G.SAME AS FILE NAME)
                              ;         3. INPUT ALL EXTRN, PUBLIC AND EXTBIT SYMBOLS DIRECTLY AFTER
                              ;             'NAME' LINE.
```

```
UPD78312 ASSEMBLER V2.3 INITIALIZATION OF COMPUTER                               DATE 23-05-86  PAGE    2
INITIALIZATION OF uPD78312

                             $EJECT
                             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

   6                                  CSEG
   7                                  ORG 00H                ; CSEG FOLLOWED BY ORG WILL MAKE THE
*** WARNING *** 000H <= ORG-address < 080H   ********************

                                                             ; ABSOLUTE SEGMENT A CODESEGMENT
   8  0000 A 8000            RESET:   DW START               ; RESET VECTOR

   9                                  ENDS                   ;END OF FIRST SECTOR (MAY BE OMMITTED)

                             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  10         FE20            SPSTRT   EQU     0FE20H         ;STACKPOINTER START ADDRESS
  11         0008            SBINIT   EQU     00001000B      ;STAND-BY REG. INIT.
  12         0000            EXTMAP   EQU     00000000B      ;INIT OF MM REG.
  13         0030            INITRB   EQU     00110000B      ;SELECT RB3 AND IE=BIT=!0

  14         0FF5            RSSBIT   EQU     PSWL.5         ;NAME CONVENTION FOR RSS-BIT IN PSWL
  15         0A25            CK1      EQU     STBC.5         ;CK1 IS BIT TO CHANGE SCLK (1/8 OR 1/2 FOSC)

  16                         INIT     ORG 080H               ;THIS IS AN 'NAMED' ABS. SEGMENT CALLED
                                                             ;'INIT'
  17                                  RSS 0                  ;THIS IS A PSEUDO INSTR. TO RESET RSS FLAG
                                                             ;FOR THE TRANSLATION OF FUNCTIONAL REGISTER
                                                             ;NAMES BY THE RA310
                                                             ;IT'S VALUE ENDS AT END OF EACH SEGMENT
                                                             ;IT'S DEFAULT VALUE IS RSS = 0

  18  0080 A 0BFC20FE        START:   MOVW SP,#SPSTRT        ;STACKPOINTER INITIALIZATION
                                                             ;JUST BELOW THE SADDR AREA
                                                             ;MAX. STACK SIZE=16 BYTES FOR INT. RAM

  19  0084 A 0944F708                 MOV STBC,#SBINIT       ;INIT. OF STAND-BY CONTROL REGISTER
                                                             ;NO HALT/STOP MODE, SCLK=!1/2 FOSC
                                                             ;SET OF 'STAND-BY' CONTROL FLAG

  20  0088 A 05AB                     SEL RB3                ;SELECT REGISTERBANK 3
                             ;        MOV PSWH,#INITRB       ;SELECT REGSITER BANK 3 FOR NEXT OPERATIONS
                                                             ;AND RESET IE-BIT;
                                                             ;(RB0 AND RB1 MAY BE USED FOR MACRO SERVICE
                                                             ;CHANNELS!)

  21  008A A 2BFE00                   MOV PSWL,#00H          ;RESET RSS FLAG AND ALL OTHER ALU FLAGS
                                                             ; TO '0'
  22  008D A 0295                     CLR1 PSWL.5            ;ANOTHER WAY TO RESET RSS-FLAG
                                                             ;OR
  23  008F A 0295                     CLR1 RSSBIT            ;USE SYMBOL FOR RSS-BIT

  24  0091 A 2B4000                   MOV MM,#EXTMAP         ;INIT. OF MEMORY MAPPING REG.
                                                             ;NO EXT. BUS, P4=INPUT MODE, NO WAIT
                                                             ;STATES REQUIRED FOR EXT. MEMORY
```

```
   25  0094 A 2B4E00                MOV CCW,#00H          ;RESET EOS AND TABLE AREA FLAG

   26  0097 A 4A                    DI                    ;SAME AS RESETTING IE-BIT, RESET COND. IS 'DI'

   27  0098 A 2C0000                BR PWMSRT             ;BRANCH TO EXTERNAL ROUTINE
                                                          ;BR LABEL --> RA310 TRIES TO OPTIMIZE
                                                          ;             BRANCHES TO LOCAL LABELS
                                                          ;BR !LABEL --> 3 BYTE BRANCH
                                                          ;BR $LABEL --> 2 BYTE BRANCH (ERROR IF TOO FAR)

   28  009B A B000          LOOP:   SET1    LED           ;SIGNAL ON IF PWM RUNNING AND INITIALIZED

   29  009D A 14FE                  BR $$                 ;LOOP ON SAME LOCATION

   30                               ENDS
   31                               END
```

```
TARGET CHIP: UPD78312

ASSEMBLY COMPLETE        0 Error(s) found

Maximum Stack-Size  : 0000H

Optimize-Passes used:    1


  Segment Informations:
  ---------------------

    ADRS    LEN    NAME

    0000  0000H  CSEG
    0000  0002H  ASEG1
    0080  001FH  INIT
```

Cross-Reference Listing:

NAME	ADDR	R	ATTR	SEG	REFERENCES		
CK1	0A25H		EQU		15#		
EXTMAP	0000H		EQU		12#	24	
INIT			SEG		16#		
INITPC			MOD		1		
INITRB	0030H		EQU		3	13#	
LED			EXT		5#	28	
LOOP	009BH	A	PUB	INIT	2	28#	
PWMSRT			EXT		4#	27	
RESET	0000H	A	PUB	ASEG1	2	8#	
RSSBIT	0FF5H		EQU		14#	23	
SBINIT	0008H		EQU		3	11#	19
SPSTRT	FE20H		EQU		3	10#	18
START	0080H	A	PUB	INIT	2	8	18#

```
COPYRIGHT NEC CORPORATION 1985, 1986
UPD78312 ASSEMBLER V2.3 PWM ROUTINE                                    DATE 23-5-86   PAGE    1
OUTPUT OF PULSES TO BY PWM0 CHANNEL
SOURCE FILE: PWM.ASM
OBJECT FILE: PWM.REL
COMMAND    : RA310 PWM.ASM


     ASSEMBLE LIST

 STNO  ADRS R OBJECT     M I  SOURCE STATEMENT

    0                         $PC(312)
                              $XREF
                              $DG
                              $PL(60)
                              $PW(132)
                              $OP(5)
                              $TAB(8)
                              $EP (CON:)

                              $TITLE ('PWM ROUTINE')
                              $SUBTITLE ('OUTPUT OF PULSES TO BY PWM0 CHANNEL')

                              $DATE ('23-5-86')

    1                                 NAME PWM

                              ;AUTHOR:        H.T. NEC EE APPLICATION

    2                         PUBLIC  PWMSRT
    3                         PUBLIC  LED                          ;P2.7 FOR LED DRIVE (BIT IN SADDR AREA)
    4                         EXTRN   START, LOOP                  ;CODE LABELS OF EXT. CSEG
    5                         EXTRN   PWMBUF                       ;PWM-VALUE BUFFER IN EXT. DSEG
    6                         EXTRN   SPSTRT, SBINIT, INITRB       ;EXTRN EQU ARE ALWAYS 16 BIT


    7                                 CSEG                         ;RELOC. CODESEGMENT FOR PWM START
    8          0817           LED     EQU     P2.7                 ;DEFINE THE BITNAME 'LED'
    9          0FF5           RSSBIT  EQU     PSWL.5               ;DEFINE AGAIN RSS BIT NAME(NOT IN
                                                                    ;SADDR AREA)
   10   0000   0295                   CLR1    RSSBIT               ;RESET RSS BIT IF NOT YET RESET

   11   0002   2B3202         PWMSRT: MOV PMC2,#00000010B          ;PC.1 USED AS CONTROL PIN FOR INTE2
                                                                    ;ALL OTHER PINS USED AS PORT
   12   0005   2B22FF                 MOV PM2, #0FFH               ;ALL PINS OF P2 TO INPUT MODE
                              ;BUT:
   13   0008   089F02                 CLR1    LED                  ;INIT OF PORT 2.7 LATCH, LED=0
   14   000B   089F22                 CLR1    PM2.7                ;USE JUST P2.7 AS OUTPUT

   15   000E   2B3310                 MOV     PMC3,#00010000B      ;P3.4 USED AS CONTROL PIN FOR PWM0 OUT

   16   0011   2B23FF                 MOV     PM3, #0FFH           ;ALL PINS OF P3 TO INPUT MODE

   17   0014   2B6001                 MOV     FRCC, #00000001B     ;PRESET PWM0 OUTPUT TO 'HIGH'
```

```
 18  0017  016E6002              OR      FRCC, #00000010B        ;PWM0 OUTPUT ENABLE
                        ;BETTER:
 19  001B  088860                SET1    FRCC.0
 20  001E  088960                SET1    FRCC.1
                        ;OR
 21        0B00          ALV0    EQU     FRCC.0
 22        0B01          ENPWM0  EQU     FRCC.1

 23  0021  088860                SET1    ALV0
 24  0024  088960                SET1    ENPWM0


 25  0027  0C140002              MOVW    PWM0,#200H              ;DUTY CYCLE 200/256
 26  002B  1C14                  MOVW    AX, PWM0                ;STORE VALUE TO RAM-BUFFER
 27  002D  1A00                  MOVW    PWMBUF,AX


 28  002F  2B6602                MOV     PWMM, #00000010B                ;DISABLE PWM1, ENABLE PWM0 TO COUNT
                                                            ;SET INPUT CLOCK SCLK/256=PERIOD TIME
                        ;OR:
 29        0B31          MOD1    EQU     PWMM.1
 30        0B32          CT0     EQU     PWMM.2
 31        0B33          CT1     EQU     PWMM.3

 32  0032  088966                SET1    MOD1
 33  0035  089A66                CLR1    CT0
 34  0038  089B66                CLR1    CT1
                        ;OR:
 35  003B  016C66D2              AND     PWMM, #11010010B

 36  003F  2C0000                BR      LOOP                    ;GO BACK TO ETERNAL LOOP

 37                              ENDS
 38                              END
```

```
TARGET CHIP: UPD78312

ASSEMBLY COMPLETE          0 Error(s) found

Maximum Stack-Size   : 0000H

Optimize-Passes used:      1


  Segment Informations:
  ---------------------

    ADRS    LEN    NAME

    0000    0042H  CSEG
```

Cross-Reference Listing:

```
NAME        ADDR   R ATTR SEG     REFERENCES
-----------------------------------------------

ALV0        0B00H    EQU              21#     23
CT0         0B32H    EQU              30#     33
CT1         0B33H    EQU              31#     34
ENPWM0      0B01H    EQU              22#     24
INITRB               EXT               6#
LED         0817H    EQU               3       8#     13
LOOP                 EXT               4#     36
MOD1        0B31H    EQU              29#     32
PWM                  MOD               1
PWMBUF               EXT               5#     27
PWMSRT      0002H R  PUB   CSEG        2      11#
RSSBIT      0FF5H    EQU               9#     10
SBINIT               EXT               6#
SPSTRT               EXT               6#
START                EXT               4#
```

```
COPYRIGHT NEC CORPORATION 1985, 1986
UPD78312 ASSEMBLER V2.3 GENERATION OF DATASEGMENTS                    DATE 23-05-86  PAGE     1

SOURCE FILE: DATA.ASM
OBJECT FILE: DATA.REL
COMMAND    : RA310 DATA.ASM


      ASSEMBLE LIST

 STNO  ADRS R OBJECT     M I  SOURCE STATEMENT

    0                         $PC(312)
                              $XREF
                              $DG
                              $EP(CON:)

                              $TITLE('GENERATION OF DATASEGMENTS')

                              $DATE('23-05-86')

    1                                 NAME DATA
    2                         EXTRN   SPSTRT, INITRB
    3                         PUBLIC  PWMBUF


    4                                 ORG 01F00H              ;ABSOLUTE DATASEGMENT IN ROM
    5  1F00 A 31323334        PAT1:    DB '1234'
    6  1F04 A 27              PAT2:    DB ''''                 ;ASCII '
    7  1F05 A 00010203        PAT3:    DB 00,01,02,03H
    8  1F09 A 040505          PAT4:    DB(3) 04,05,06          ;CONTROL PATTERN LENGTH
    9                         PAT5:    DB(0)                   ;= DS 0
   10  1F0C A 0000            PAT6:    DW SPSTRT
   11  1F0E A 01              PAT7:    DB LOW (INITRB+1)       ;EXTRN SYMBOLS ARE ALWAYS 16 BIT!!
                                                               ;ONLY POSITVE OFFSETS ARE ALLOWED TO EXTRN
                                                               ;REFERENCES


   12                         BUFFER   DSEG                   ;RELOC DATASEGMENT FOR RAM AREA
                                                               ;NAMED 'BUFFER'
                                                               ;SHOULD BE LOCATED TO SADDR AREA BY LC310
   13  0000                   PWMBUF:  DS 2                   ;2 BYTE BUFFER FOR PWM0 VALUE
   14  0002                   STORE:   DS 10                  ;MORE RAM AREA E.G. FOR MACRO SERVICE

   15                                 ENDS
   16                                 END
```

```
TARGET CHIP: UPD78312

ASSEMBLY COMPLETE        0 Error(s) found

Maximum Stack-Size  : 0000H


  Segment Informations:
  ---------------------

    ADRS    LEN    NAME

    0000  0000H  CSEG
    0000  000CH  BUFFER
    1F00  000FH  ASEG1
```

Cross-Reference Listing:

NAME	ADDR	R	ATTR	SEG	REFERENCES
BUFFER			SEG		12#
DATA			MOD		1
INITRB			EXT		2# 11
PAT1	1F00H	A	LOC	ASEG1	5#
PAT2	1F04H	A	LOC	ASEG1	6#
PAT3	1F05H	A	LOC	ASEG1	7#
PAT4	1F09H	A	LOC	ASEG1	8#
PAT5	1F0CH	A	LOC	ASEG1	9#
PAT6	1F0CH	A	LOC	ASEG1	10#
PAT7	1F0EH	A	LOC	ASEG1	11#
PWMBUF	0000H	R	PUB	BUFFER	3 13#
SPSTRT			EXT		2# 10
STORE	0002H	R	LOC	BUFFER	14#

UPD78310 LINKER V2.3 [27 Jun 86]

```
        INPUT FILES                     INPUT MODULES:
        INITPC.REL                      INITPC
        PWM.REL                         PWM
        DATA.REL                        DATA

        OUTPUT FILE:   DEMO.LNK

        CONTROLS SPECIFIED IN INVOCATION LINE:
        MAP NAME(MOD_DEMO)

        LINK MAP OF MODULE:  MOD_DEMO


        LOGICAL SEGMENTS INCLUDED:

        START     STOP      LENGTH    TYPE    SEGMENTNAME
        00000H    00041H    00042H    CSEG    CSEG
        00000H    00001H    00002H    ASEG
        00080H    0009EH    0001FH    ASEG    INIT
        00000H    0000BH    0000CH    DSEG    BUFFER
        01F00H    01F0EH    0000FH    ASEG

LINK COMPLETE
```

```
COPYRIGHT (C)  1986  NEC CORPORATION                          PAGE   1

UPD78310 LOCATER V2.3  [ 6 Jun 86]

          INPUT FILE:    DEMO.LNK
          OUTPUT FILES:  DEMO.HEX
                         DEMO.SYM
          CONTROLS SPECIFIED IN INVOCATIONLINE:
          CODE:          00080H
          DATA:          0FE00H
          STACK:         0FE00H
          STACKSIZE:     0001FH
          SEGMENTE:      BUFFER    0FE20H
          GAPS:          02000H    0FDFFH
    WARNING: SEGMENT  ALLOCATED BELOW 00080H
```

PAGE 2

SYMBOLTABLE OF MODULE: MOD_DEMO

ADDRESS	TYPE	SEGIND	SYMBOLNAME	ADDRESS	TYPE	SEGIND	SYMBOLNAME
009BH	PUB	2	LOOP	0080H	PUB	2	START
0000H	PUB	1	RESET	FE20H	PUB		SPSTRT
0008H	PUB		SBINIT	0030H	PUB		INITRB
1F11H	PUB	0	PWMSRT	FF02H.7	PUB		LED
FE20H	PUB	3	PWMBUF				

PAGE 3

MODULE: INITPC

ADDRESS	TYPE	SEGIND	SYMBOLNAME	ADDRESS	TYPE	SEGIND	SYMBOLNAME
009BH	PUB	2	LOOP	0080H	PUB	2	START
0000H	PUB	1	RESET	FE20H	PUB		SPSTRT
0008H	PUB		SBINIT	0030H	PUB		INITRB
1F11H	RES	0	PWMSRT	FF02H.7	RES		LED
0000H	LOC		EXTMAP	FFFEH.5	LOC		RSSBIT
FF44H.5	LOC		CK1				

MODULE: PWM

ADDRESS	TYPE	SEGIND	SYMBOLNAME	ADDRESS	TYPE	SEGIND	SYMBOLNAME
1F11H	PUB	0	PWMSRT	FF02H.7	PUB		LED
0080H	RES	2	START	009BH	RES	2	LOOP
FE20H	RES	3	PWMBUF	FE20H	RES		SPSTRT
0008H	RES		SBINIT	0030H	RES		INITRB
FFFEH.5	LOC		RSSBIT	FF60H.0	LOC		ALV0
FF60H.1	LOC		ENPWM0	FF66H.1	LOC		MOD1
FF66H.2	LOC		CT0	FF66H.3	LOC		CT1

MODULE: DATA

ADDRESS	TYPE	SEGIND	SYMBOLNAME	ADDRESS	TYPE	SEGIND	SYMBOLNAME
FE20H	RES		SPSTRT	0030H	RES		INITRB
FE20H	PUB	3	PWMBUF				
1F00H	LOC	4	PAT1	1F04H	LOC	4	PAT2
1F05H	LOC	4	PAT3	1F09H	LOC	4	PAT4
1F0CH	LOC	4	PAT5	1F0CH	LOC	4	PAT6
1F0EH	LOC	4	PAT7	FE22H	LOC	3	STORE

```
          MEMORY-MAP OF MODULE: MOD_DEMO
          START     STOP      LENGTH    SEGTYP     SEGNAME
          00000H    00001H    00002H     ASEG
          00002H    0007FH    0007EH     GAP
          00080H    0009EH    0001FH     ASEG       INIT
          0009FH    01EFFH    01E61H     GAP
          01F00H    01F0EH    0000FH     ASEG
          01F0FH    01F50H    00042H     CSEG       CSEG
          01F51H    0FE1FH    0DECFH     GAP
          0FE20H    0FE2BH    0000CH     DSEG       BUFFER
          0FE2CH    0FFFFH    001D4H     GAP
LOCATE COMPLETE

:0200000080007E
:100080000BFC20FE0944F70805AB2BFE000295028D
:0F009000952B40002B4E004A2C111FB70214FE77
:101F0F0002952B32022B22FF089F02089F222B33B0
:101F1F00102B23FF2B6001016E6002088860088977
:101F2F006008886008896 00C1400021C141A202BAA
:101F3F006602088966089A66089B66016C66D22C51
:021F4F009B00F5
:0F1F000031323334270001020304050620FE317D
:00000001FF
```

APPENDIX B; References to uPD78112

DETAILS OF THE RA110 RELOCATABLE ASSEMBLER PACKAGE

In general: The RA110 relocatable assembler software package supports the NEC single chip microcomputer uPD78112. The processor type must be selected using the '$PC(112)' primary control.

The features of this software are basically the same as those of the RA310 software. Only microcomputer specific details are different.

In details:

A. The mnemonics are changed according to the specification of the microcomputer. The list of all mnemonics are to be found in the respective product descriptions.

B. The default start addresses of the DSEG area are different compared to RA310.

	78110	78220/78224	78310/78312
CSEG	80H	80H	80H
DSEG	0FE40H	0FC80H	0FE00H
BSEG	----	0FE20H	0FE20H

Of course you can change the start addresses of those segments using the 'SEGMENT' or 'CODE'/'DATA' directive of the locator.

The RA110 does not support bitsegment 'BSEG' as no 'SADDR.BIT' operands are available. The port bit names are well defined by 'SFR.BIT'. As well generation of bitsymbols is not meaningful.

For details study the map specification of the respective microcomputer product description.

C. As the 781XX family does not feature a 'RSS' flag in the PSW there is no 'RSS' pseudo instruction in the RA110 assembler.

But, nevertheless, the RA110 supports the use of functional register names like A, B, C, etc. Instead of the absolute register names R0, R1, R2, etc.

The correlation of absolute and functional register names is like this:

X	R0	AX	RP0
A	R1	BC	RP1
C	R2	DE	RP2
B	R3	HL	RP3
E	R4		
D	R5		
L	R6		
H	R7		

D. The uPD78112 does not have external memory. It is a pure single chip. Therefore, it does not have a 'CALLT1' area at 8040H - 8080H and addresses between 2000H and 0FE7FH are not available.

APPENDIX C: References to uPD 78210/20/224

DETAILS OF THE RA210 RELOCATABLE ASSEMBLER PACKAGE

In general: The RA210 relocatable assembler software package supports the NEC single chip microcomputers uPD78210, uPD78220 and uPD78224. The processor types must be selected using the '$PC(XXX)' primary control.

The features of this software are basically the same as those of the RA310 software. Only microcomputer specific details are different.

In details:

A. The mnemonics are changed according to the specification of the microcomputer. The list of all mnemonics is to be found in the respective product descriptions.

B. The default start addresses of the DSEG and BSEG areas are different compared to RA310.

	78210	78220/78224	78310/78312
CSEG	80H	80H	80H
DSEG	0FE80H	0FC80H	0FE00H
BSEG	0FE80H	0FE20H	0FE20H

Of course you can change the start addresses of those segments using the segment or code/data directive of the locator.
For details study the map specification of the respective microcomputer product description.

C. As the 782XX family does not feature a 'RSS' flag in the PSW there is no 'RSS' pseudo instruction in the RA210 assembler.
But, nevertheless, the RA210 supports the use of functional register names like A, B, C, etc. instead of the absolute register names R0, R1, R2, etc.

The correlation of absolute and functional register names is like this:

X	R0	AX	RP0
A	R1	BC	RP1
C	R2	DE	RP2
B	R3	HL	RP3
E	R4		
D	R5		
L	R6		
H	R7		

D. The 782XX family supports a 1 MByte data access area. 4 additional bits at port 60 - 63 are output if an instruction like 'MOV A,&MEM' or 'MOV A,&!ADDR16' is executed. '&' indicates the 1 MByte access. ADDR16 may be a well defined, i. e. backward referenced symbol.

The 1 MByte area is splitted to max 16 64 KByte banks. The selection of the bank is performed by the a. m. port/nibble.

The user must generate the data pattern of the 64K banks by SEPARATE Data modules. These modules have absolute start addresses (for example 010000H) and have a maximum size of 64 KByte. They are not linked to the code modules in bank 0, but only used for the creation of a hexadecimal file that is burned into an EPROM in the prototype hardware.

Use of symbols of data modules: As the data modules are not linked to the code module (bank 0) the user must generate common include files, in which the addresses of the symbols in the absolute data module are well defined.

Example:

CODE MODULE:

```
COPYRIGHT NEC CORPORATION 1985, 1986
UPD78224 ASSEMBLER E0.0 PROGRAM CODE                                    DATE          PAGE    1
DATA FETCH FROM BANK1
SOURCE FILE: CODE.ASM
OBJECT FILE: CODE.REL
COMMAND    : RA210 CODE.ASM


      ASSEMBLE LIST


 STNO  ADRS R OBJECT     M I  SOURCE STATEMENT

    0                          $PC(224)
                               $TITLE('PROGRAM CODE')
                               $SUBTITLE('DATA FETCH FROM BANK1')

    1                                  NAME COD1

                           1   $INCLUDE(DATAS.INC)
                           1   ;-----------------------------------------------------
                           1   ;INCLUDE FILE FOR DEFINITION OF DATA MODULE SYMBOLS
                           1   ;
    2         0020         1   PAT0    EQU     020H
    3         0040         1   PAT1    EQU     040H
                           1   ;

-----------------------------------------------
    4                          ORG 0H
*** WARNING *** 000H <= ORG-address < 080H   ********************

    5  0000 A 8000             RESET:  DW START

    6                          ORG 80H
    7  0080 A 662000           START:  MOVW HL,#PAT0          ;AIMING AT DATA PATTERN 'PAT0' OF BANK1
                                                              ;AT ADDRESSES 10000H - 1FFFFH BY [HL]
    8  0083 A 644000                   MOVW DE,#PAT1          ;AIMING AT 'PAT1' OF BANK1 BY [DE]


    9  0086 A 3A0601                   MOV P6,#01H            ;P6 LOWER NIBBLE OUTPUTS 4 ADDITIONAL BITS
                                                              ; FOR ADDRESSING 1 MBYTE
                                                              ;'01' SELECT BANK1 (10000H - 1FFFFH)
                               ;
                               ;       INITIALIZATION OF 1 MBYTE BUS MODE NOT SHOWN HERE
                               ;
                               ;INDIRECT ADDRESSING
   10  0089 A 011610                   MOV A,&[HL+]           ;LOADING CONTENTS OF 10000H+PAT0 INTO ACCU
   11  008C A 011680                   MOV &[DE+],A           ;LOADING ACCU INTO 10000H+PAT1

                               ;DIRECT ADDRESSING
   12  008F A 0109F00005               MOV A,&!0500H          ;LOADING CONTENTS OF 10500H INTO ACCU
   13  0094 A 0109F10205               MOV &!0502H,A          ;LOADING ACCU INTO 10501H

   14  0099 A 14FE                     BR $$

   15                              ENDS
   16                              END
```

```
TARGET CHIP: UPD78224

ASSEMBLY COMPLETE          0 Error(s) found

Maximum Stack-Size  : 0000H


  Segment Informations:
  ---------------------

    ADRS    LEN    NAME

    0000  0000H  CSEG
    0000  0002H  ASEG1
    0080  001BH  ASEG2

DATA MODULE:
```

```
COPYRIGHT NEC CORPORATION 1985, 1986
UPD78224 ASSEMBLER E0.0 DATA PATTERN FOR BANK1                  DATE          PAGE    1

SOURCE FILE: DATA.ASM
OBJECT FILE: DATA.REL
COMMAND    : RA210 DATA.ASM


      ASSEMBLE LIST

 STNO  ADRS R OBJECT      M I  SOURCE STATEMENT

    0                          $PC(224)
                               $TITLE('DATA PATTERN FOR BANK1')

    1                                 NAME BANK1

    2                          ORG 20H
*** WARNING *** 000H <= ORG-address < 080H   ********************

    3  0020 A 5448495320       PAT0:   DB 'THIS IS A TEXT IN A DATA PATTERN'
              4953204120
              5445585420
              494E204120
              4441544120
              5041545445
              524E

    4                          ORG 40H
*** WARNING *** 000H <= ORG-address < 080H   ********************

    5  0040 A 412042524F       PAT1:   DB 'A BROWN FOX JUMPS OVER THE FENCE'
              574E20464F
              58204A554D
              5053204F56
              4552205448
              452046454E
              4345

    6                               END
    7                               END
```

```
TARGET CHIP: UPD78224

ASSEMBLY COMPLETE          0 Error(s) found

Maximum Stack-Size  : 0000H


  Segment Informations:
  ---------------------

    ADRS    LEN    NAME

    0000   0000H   CSEG
    0020   0020H   ASEG1
    0040   0020H   ASEG2


INCLUDED FILE FOR SYMBOL EXCHANGE:
;-------------------------------------------------
;INCLUDE FILE FOR DEFINITION OF DATA MODULE SYMBOLS
;
PAT0     EQU      020H
PAT1     EQU      040H
;
;-------------------------------------------------
```

Book 6

µCOM 87 AD

FLOATING POINT

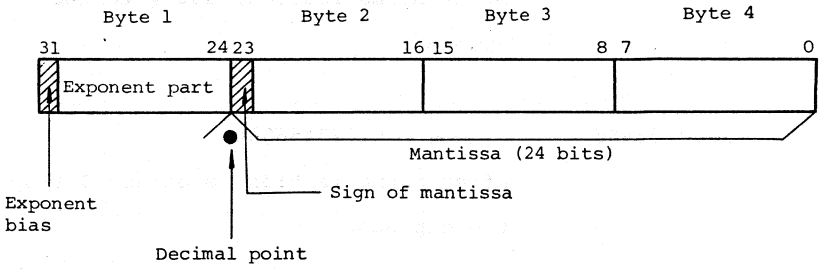
ARITHMETIC LIBRARY

APPLICATION NOTE

CHAPTER I FLOATING POINT OPERATION SUBROUTINE

1. Representation of Floating-point Binary Numbers

In this arithmetic operation program, a floating-point binary number consists of an 8-bit exponent and a 24-bit mantissa as shown in Figure 1.1. The decimal point is at the left of the MSB of the mantissa.



$$\text{Numeral} = (\text{sign}) \mid (\text{mantissa}) \mid \times 2^{(\text{exponent})}$$

$$\text{sign (bit 23)} = \begin{cases} 0; \text{ positive} \\ 1; \text{ negative} \end{cases}$$

Fig. 1-1 Representation of floating-point binary number

The following are more detailed explanations of each part.

1-1 The Mantissa

The mantissa consists of a positive 24-bit binary number. When the floating-point binary number is normalized the MSB (bit 23) of the mantissa is always 1. This bit therefore, can contain sign information of the mantissa.

Normalization means the adjustment of the exponent so that the most significant bit (MSB) of the mantissa (bit 23) becomes 1. This is illustrated below.

Examples

Before normalization		After normalization
1234.567890	⟶	0.123456789

After normalization, add 4 to the exponent.

0.000123456	⟶	0.123456000

After normalization, subtract 3 from the exponent.

1-2 The Exponent

The exponent consists of an 8-bit binary number (two's complement in the case of a negative number) plus 80H as the bias. The relationship between the actual exponent value and its representation in the exponent is as shown below:

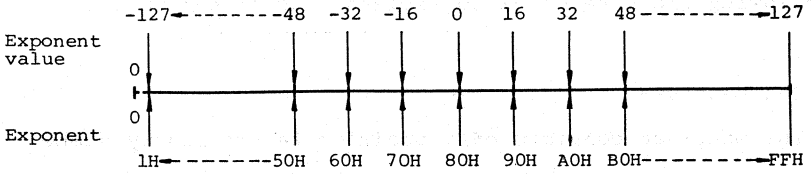


Fig. 1-2 Representation of the exponent

Figure 1-2 indicates the following:

MSB of exponent (Bit 31) = {'1' positive exponent / '0' negative exponent}

When the exponent is 0, it represents the number 0 and the mantissa is ignored. The number 0 can therefore be expressed simply by making the exponent 0.

2. Variables

This section explains the key variables in this arithmetic operation program. For other variables, refer to individual arithmetic operation subroutines.

2-1 Binary Floating-point Accumulator (B.F.P.ACC)

All arithmetic operations are performed using the binary floating-point accumulator (hereafter referred to as the B.F.P.ACC). The B.F.P.ACC consists of a contiguous five-byte memory area. (See Figure 2-1.)

The details of each part are explained below:

(1) Exponent (ACCE)

Refer to 1-2.

(2) Sign (ACCS)

Shows the sign of the mantissa.

ACCS = {80H ... positive / 00H ... negative}

(3) Mantissa (ACC1, ACC2 and ACC3)

ACC's 1 to 3 contain a three-byte absolute value. The MSB of ACC1 is, therefore, always 1 (normalized).

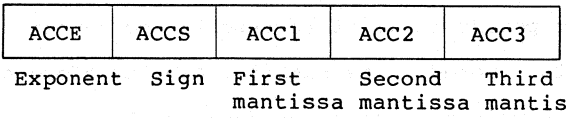


Fig. 2-1 B.F.P.ACC memory configuration

2-2 Operand (OPE.)

The operand (hereafter referred to as the OPE.), along with the B.F.P.ACC, plays an important role in arithmetic operations. It is used mainly to perform operations on two variables. The OPE. consists of a contiguous four-byte memory area (WSE, WS1, WS2 and WS3). (See Figure 2-2.)

For the details of each part, refer to 1-1 and 1-2.

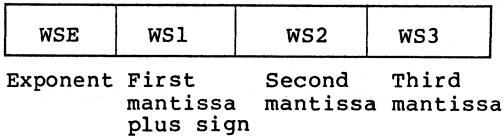


Fig. 2-2 Operand memory configuration

2-3 SF

This is the addition/subtraction flag between the B.F.P.ACC and the OPE.

$$SF = \begin{cases} 0 & : \text{Same signs indicate addition.} \\ 80H & : \text{Different signs indicate subtraction} \end{cases}$$

2-4 FSN

This is a sign flag used for checking the sign of the B.F.P.ACC:

FSN = { 0 : The B.F.P.ACC is 0.
1 : The B.F.P.ACC is positive.
OFFH: Different signs is negative.

3. Arithmetic Subroutines

3-1 Arithmetic Subroutines

The following four arithmetic subroutines are provided:

(1) BFADD: Floating-point addition subroutine
(2) BFSUB: floating-point subtraction subroutine
(3) BFMUL: Floating-point multiplication subroutine
(4) BFDIV: Floating-point division subroutine

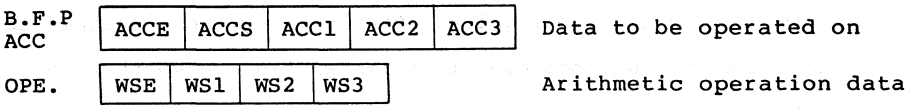


Fig. 3-1 Arithmetic operation memory configuration

These arithmetic operation subroutines perform arithmetic operations using the B.F.P.ACC data (five bytes) and the OPE data (four bytes). The results are stored in the B.F.P.ACC.

Given the following input condition, each subroutine moves the desired data from memory to the OPE and then performs its operation.

Input condition

"The starting address of the four-byte memory area storing the arithmetic operation data is indicated by the HL pair register. The data to be operated on is stored in the B.F.P.ACC."

Output condition

"RET: The result of the operation has overflowed
'1' is set in the CY flag.

RETS: A normal result is stored in the B.F.P.ACC.
'0' is set in the CY flag.
If the result is 0, '1' is set in the Z flag.
Otherwise, '0' is set in the Z flag."

A NOP instruction is required after calling these arithmetic operation subroutines even if it is already known that the result of the operation did not overflow.

3-1-1 BFADD (addition) subroutine

(1) Processing

Add the four-byte floating-point binary OPE data and the B.F.P.ACC data and store the result in the B.F.P.ACC.

(2) Input condition

Refer to the input condition for arithmetic operations described above.

(3) Output condition

Refer to the output condition for arithmetic operations described above.

(4) Subroutines used

Refer to Table 4-1.

(5) Stack depth

Max. 2

(6) Coding sequence

```
        〈
LXI   H, - ; OPE four-byte data starting address
CALL  BFADD; Addition processing
GJMP  ERROR; Overflow processing
        〉
```

(7) Processing time

Max. about 1.6 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC: 9BH, 80H, FFH, FFH, FFH

OPE : 82H, 7FH, FFH, FFH

(8) Processing procedure

The operation of adding B.F.P.ACC data and OPE data can be divided into the following patterns depending on the signs of B.F.P.ACC(X) and OPE(Y).

Table 3-1 Operation patterns by signs of X ans Y

Pattern	Sign of X	Sign of Y	Sign after operation	Operation flag SF	Operation (absolute value) processing
1	Positive	Positive	Positive	0	X + Y
2	Positive	Negative	-	80 H	X - Y
3	Negative	Positive	-	80 H	Y - X
4	Negative	Negative	Negative	0	X + Y

SF: As shown in Table 3-1, this is established as Sign of X $\forall$ $\overline{\text{Sign of Y}}$. Only bit 7 affects the result and bits 0 to 6 are zeroed. Thus :

$$SF = \begin{cases} 80\text{ H}: & \text{subtraction} \\ 0 \quad : & \text{addition} \end{cases}$$

The processing procedure is as follows:

a) The status of the B.F.P.ACC is checked. If it is 0, processing is terminated.

b) The status of the OPE is checked. If it is 0, processing is terminated.

c) The sign assigned to the greater exponent is used after the operation.

d) The mantissa with the greater exponent is stored in ACCs 1, 2 and 3, while the mantissa with the lesser exponent is stored in WSs 1, 2 and 3.

e) Addition/subtraction between the mantissas is performed depending on the SF flag.
At this time, it is necessary to shift the mantissa with the lesser exponent part right by the difference of the exponents for digit adjustment between the two exponents as illustrated in Example 1 below. (The result remains in WSs 1, 2, 3 and 4.)

Example 1: Adjustment of exponents

$0.123400000 \times 10^{4} + 0.567800000 \times 10^{-2}$
$\rightarrow 0.123400000 \times 10^{4} + 0.000000567 \times 10^{4}$
$\rightarrow (0.123400000 + 0.000000567) \times 10^{4}$

The digits overflowing after the right shift are truncated. If the difference between the exponents is 25 or more, the operation is terminated assuming that the lesser value is of no significance to the greater one.

f) If addition or subtraction in pattern 1 or 4 (Table 3-1) between mantissas after shifting results in an overflow, the mantissa of the result is shifted to the right by one bit and 1 is set in the most significant bit. At the same time, the exponent is increased by one (1).

Example 2:

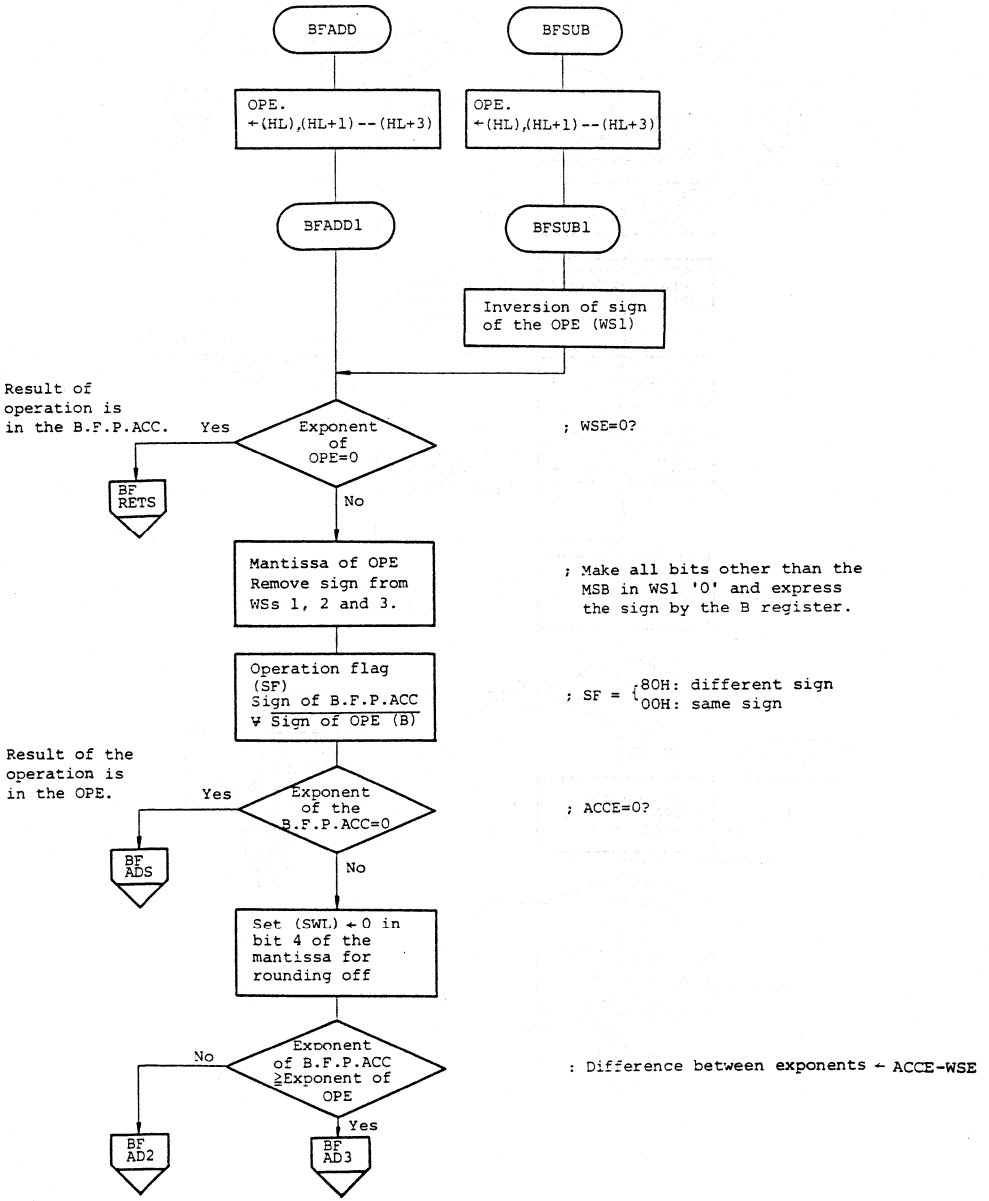
$0.999900000 \times 10^{1} + 0.001234000 \times 10^{1}$
$\rightarrow 1.001134000 \times 10^{1}$
$\rightarrow 0.100113400 \times 10^{2}$

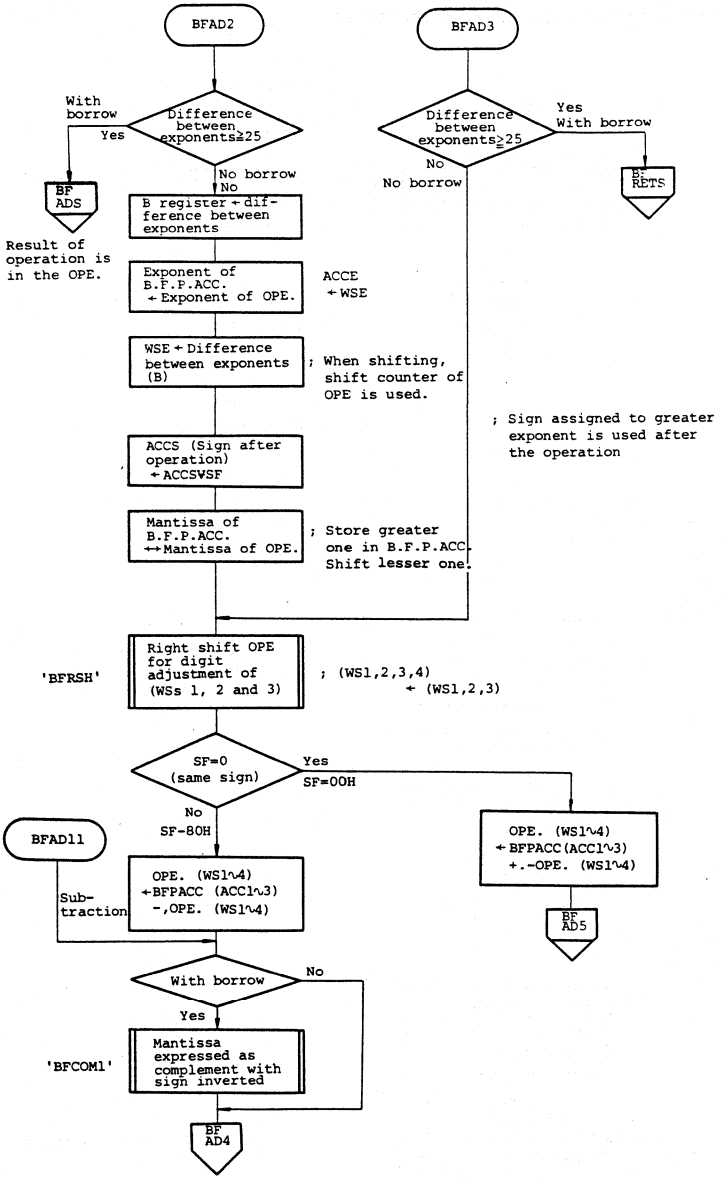
In patterns 2 and 3, if a borrow arises after subtracting the absolute value of the mantissa, it is represented in the form of the complement with the sign inverted.

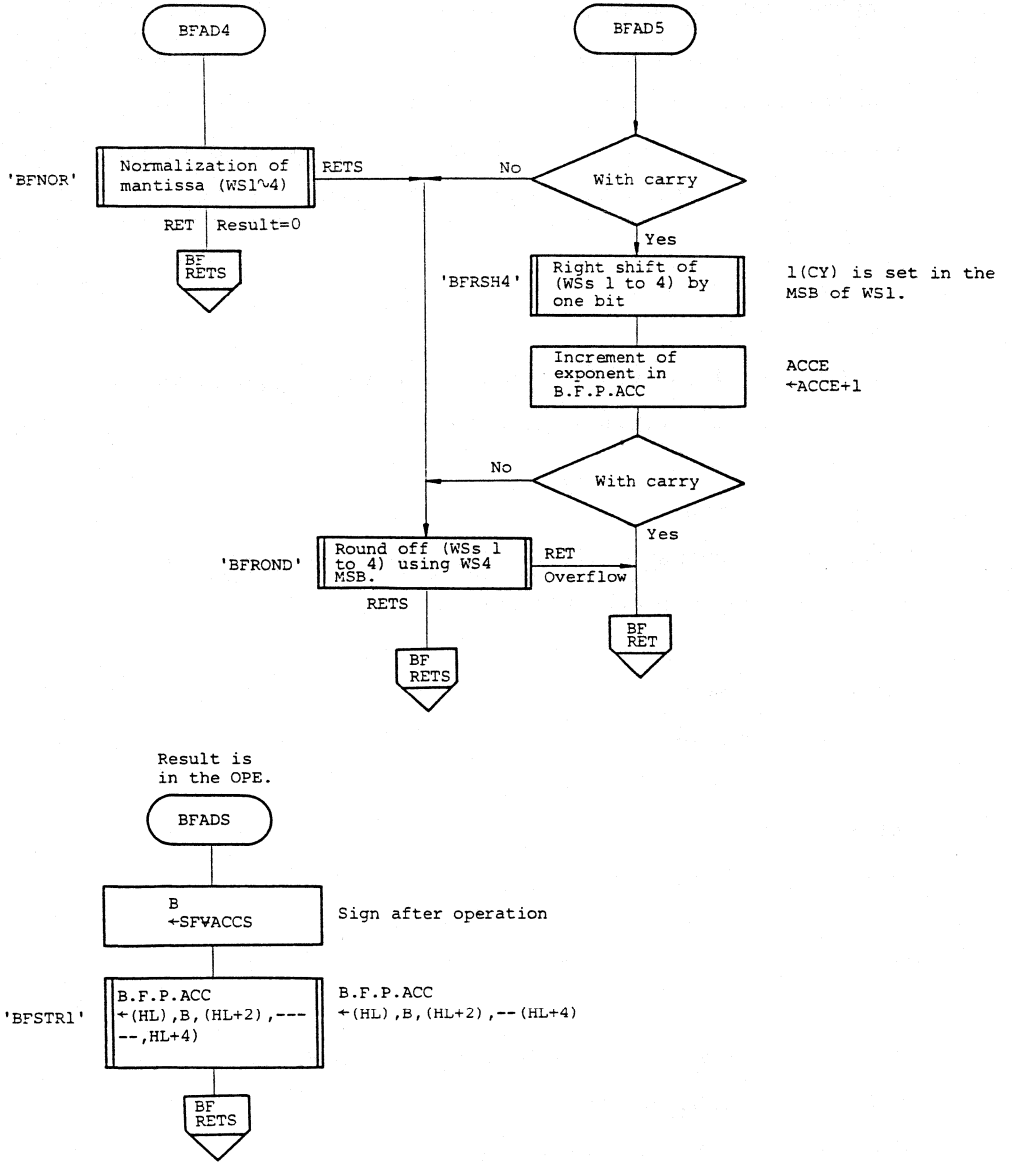
g) Normalization is performed after rounding off the most significant bit of WS4.

Below is a flowchart of the procedure:

Addition/subtraction subroutines




BFAD2
BFAD3
With borrow
Yes
Difference between exponents≧25
No borrow
No
Yes
With borrow
No borrow
BF ADS
BF RETS
Result of operation is in the OPE.
B register ← difference between exponents
Exponent of B.F.P.ACC. ← Exponent of OPE.
ACCE ← WSE
WSE ← Difference between exponents (B)
; When shifting, shift counter of OPE is used.
; Sign assigned to greater exponent is used after the operation
ACCS (Sign after operation) ← ACCS∨SF
Mantissa of B.F.P.ACC. ↔ Mantissa of OPE.
; Store greater one in B.F.P.ACC. Shift lesser one.
'BFRSH'
Right shift OPE for digit adjustment of (WSs 1, 2 and 3)
; (WS1,2,3,4) ← (WS1,2,3)
SF=0 (same sign)
Yes
SF=00H
No
SF-80H
BFADl1
OPE. (WS1∿4) ← BFPACC (ACC1∿3) +.-OPE. (WS1∿4)
BF AD5
Sub-traction
OPE. (WS1∿4) ← BFPACC (ACC1∿3) -,OPE. (WS1∿4)
With borrow
No
Yes
'BFCOM1'
Mantissa expressed as complement with sign inverted
BF AD4


BFAD4
BFAD5
'BFNOR'
Normalization of mantissa (WS1∿4)
RETS
No
With carry
RET
Result=0
BF RETS
Yes
'BFRSH4'
Right shift of (WSs 1 to 4) by one bit
1(CY) is set in the MSB of WS1.
Increment of exponent in B.F.P.ACC
ACCE ←ACCE+1
No
With carry
Yes
'BFROND'
Round off (WSs 1 to 4) using WS4 MSB.
RET
Overflow
RETS
BF RETS
BF RET
Result is in the OPE.
BFADS
B ←SFVACCS
Sign after operation
'BFSTR1'
B.F.P.ACC ←(HL),B,(HL+2),----,HL+4)
B.F.P.ACC ←(HL),B,(HL+2),--(HL+4)
BF RETS

3-1-2 BFSUB (Subtruction) subroutine

1) Processing

Subtract the contiguous four-byte floating-point OPE data from the B.F.P.ACC data and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input conditions for arithmetic operations described in the previous section.

3) Output condition

Refer to the output condition for arithmetic operations described in the previous section.

4) Subroutines used

Refer to Table 4-1.

5) Stack depth

Max. 2

6) Coding sequence

```
      )
      (
LXI   H,    ; OPE four-byte data starting address
CALL  BFSUB; Subtraction
GJMP  ERROR; Overflow processing
      )
      (
```

7) Processing time

Max. about 1.6 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC: 9BH, 80H, FFH, FFH, FFH

OPE : 82H, 7FH, FFH, FFH

8) Processing procedure

The subsequent processing is the same as that of the BFADD subroutine except that the sign of the OPE is inverted as shown in the following example.

Example:

B.F.P.ACC	-	OPE.	→	B.F.P.ACC	+	(-OPE.)
10	-	4	→	10	+	(-4)
10	-	(-4)	→	10	+	4
-10	-	4	→	-10	+	(-4)
-10	-	(-4)	→	-10	+	4

For a flowchart, refer to the FBADD subroutine.

3-1-3 BFMUL (Multiplication) subroutine

1) Processing

Multiply the four-byte OPE data and the B.F.P.ACC data and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for arithmetic operations.

3) Output condition

Refer to the output condition for arithmetic operations.

4) Subroutines used

Refer to Table 4-2.

5) Stack depth

Max. 4

6) Coding sequence

```
      (
LXI   H,    ; OPE four-byte data starting address
CALL  BFMUL; Multiplication
GJMP  ERROR; Overflow processing
      )
```

7) Processing time

Max. about 2.5 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC: 9BH, 80H, FFH, FFH, FFH

OPE : 82H, 7FH, FFH, FFH

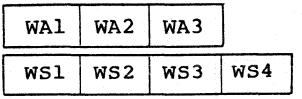
8) Processing procedure

a) Check the OPE data (multiplicand). If it is 0, the result of the operation is 0.

b) Add the exponent parts. At this time, the OPE data is converted to an unsigned mantissa.

c) Multiply the mantissas. (3 bytes x 3 bytes)

The multiplication subroutine (BFMX) has the memory configuration shown in Figure 3-2.



The operation of the one-byte multiplier x three-byte multiplicand is performed three times according to the following procedure:

c-1) Save the multiplicand (WS1, WS2, WS3) in (WA1, WA2, WA3), clearing the result area (WS1, WS2, WS3).

c-2) Perform the operation

(WA1, WA2, WA3) x one-byte multiiplier (ACC3)

+ previous result (WS1, WS2, WS3)

and store the result in the B register (WS1, WS2, WS3).

c-3) Shift the results of c-2 (B register, WS1, WS2, and WS3) right by one byte each and store the results in (WS1, WS2, WS3 and WS4).

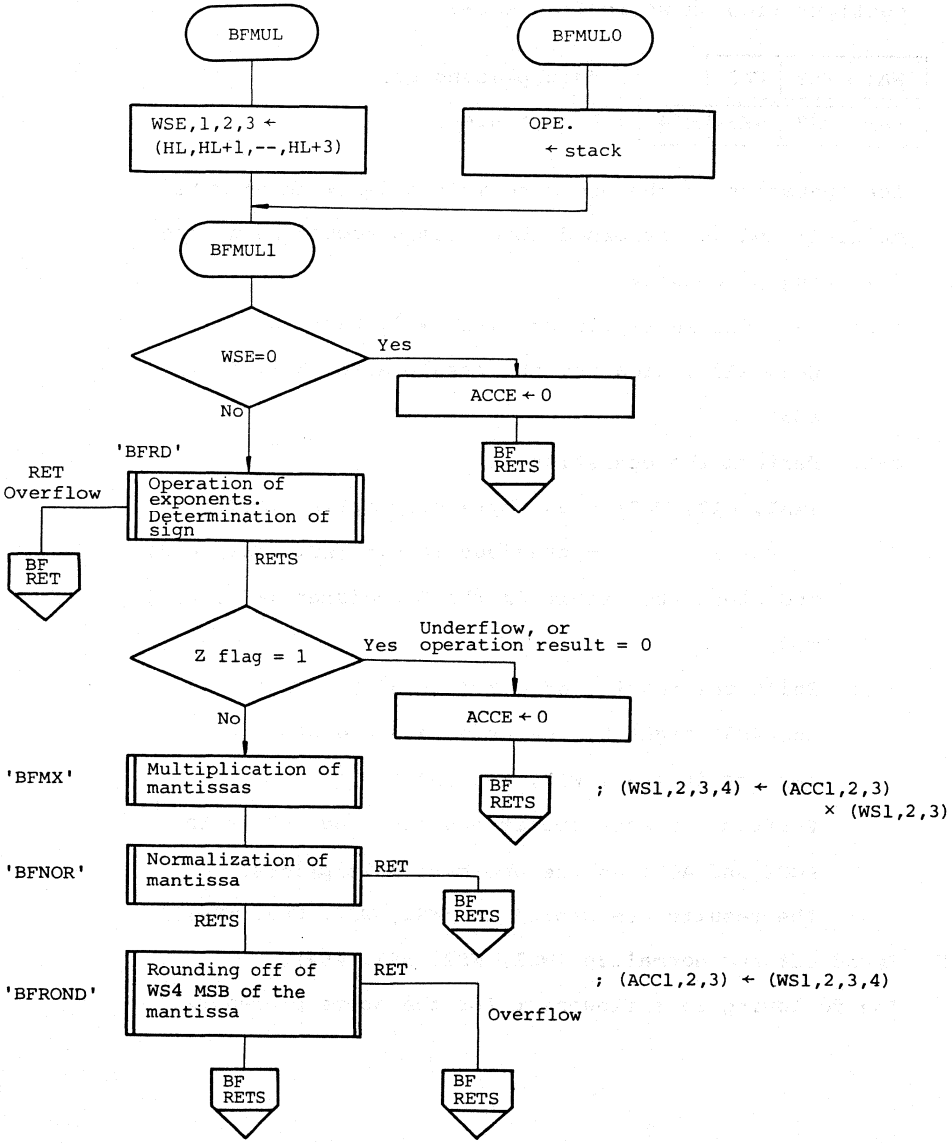
Perform the same operations (c-2 and c-3) with ACC2 and ACC1 as the one-byte multipliers.

The results are stored in (WS1, WS2, WS3, WS4).

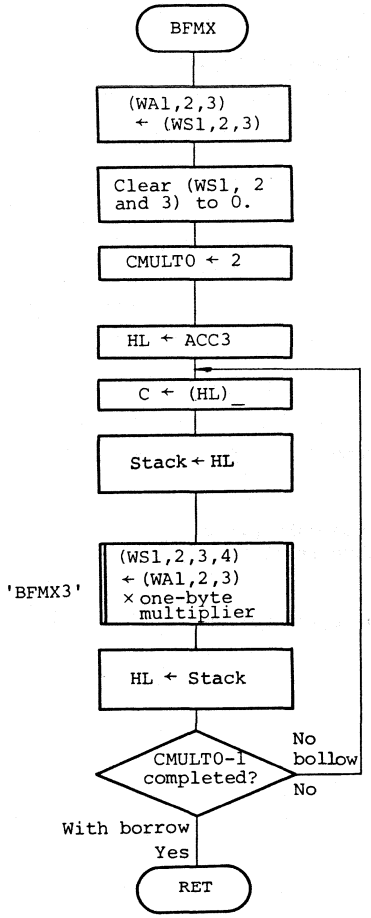
d) Round off and normalize (WS1, WS2, WS3, WS4).

The following is a flowchart for the above procedure.

Multiplication subroutine



Mantissa multiplication subroutine



; Save the multiplicand.

; Clear the result area to 0.

; Set the multiplier digit counter.

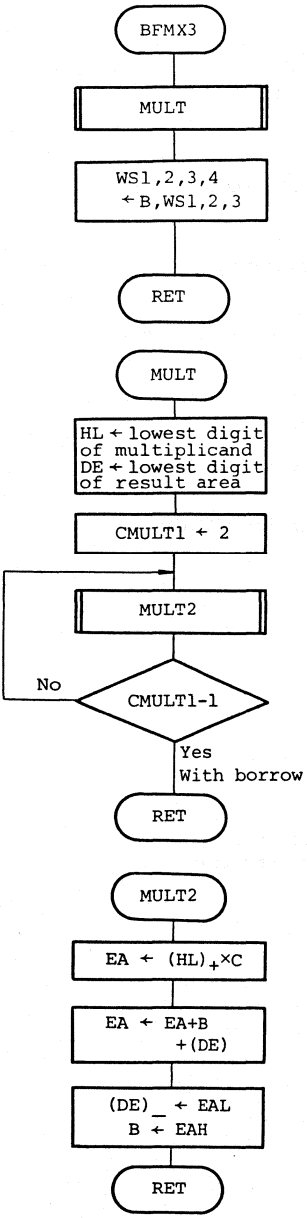
; Determine the lowest address of the multiplier.

; Determine the one-byte multiplier.

; Save the multiplier address in the stack.

; Decrease the counter by one. Check it after completion of the operation.

Mantissa (WA1, 2, 3) x one-byte multiplier + previous result



; B,WS1,WS2,WS3
← (WA1,2,3) × one-byte multiplier

'BFRSH0'

; Set the multiplicand digit counter

; (WA1,2,3) × one-byte multiplier + previous result

; (WA1,2,3) × one-byte multiplier completed?

B ; Carry from previous digit
; (DE); Previous result

3-1-4 BFDIV (Division) subroutine

1) Processing

Divide the B.F.P.ACC data by the four-byte OPE data and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for arithmetic operations.

3) Output condition

Refer to the output condition for arithmetic operations.

4) Subroutine used

Refer to Table 4-3.

5) Stack depth

Max. 2

6) Coding sequence

```
        )
        (
LXI   H,    ; OPE four-byte data starting address
CALL  BFDIV; Division
GJMP  ERROR; Overflow processing
        )
        (
```

7) Processing time:

Max. about 3.5 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC: 9BH, 80H, FFH, FFH, FFH

OPE : 82H, 7FH, FFH, FFH

(8) Processing procedure

The BFDIV subroutine performs division based on the non-recovery method. There are two methods of performing the operation of division. They are the recovery method and the non-recovery method. These are explained below using as an example of the division of decimal numbers.

(Recovery method)

The divisor is subtracted from the dividend repeatedly until the remainder becomes negative. Each time the divisor is subtracted the quotient is increased by one and the remainder is checked to see if it is negative or not. If the remainder is negative, it means that the divisor has been subtracted one time too many. Therefore, the previous remainder and quotient must be recovered, so the divisor is added to the negative remainder and the quotient is reduced by one. This processing is performed for each digit. See Example 1 for this method.

(Non-recovery method)

Unlike the recovery method, this method performs no recovery processing.

The divisor is subtracted from the dividend repeatedly until the remainder becomes negative. Then, the divisor is shifted to the right by one digit and added repeatedly until the remainder becomes positive. Then, the divisor is shifted to the right by one digit and subtracted repeatedly until the remainder becomes negative. The non-recovery method performs subtraction and addition operations alternately. In decimal notation, there is no difference in processing time between the recovery and non-recovery methods. In binary notation, however, the processing time with the non-recovery method is twice as fast as the recovery method because the former performs subtractions without recovery processing and determines whether the next digit is to undergo addition or subtraction depending on the existence of a borrow. See Example 2 for this method.

Example 1 Recovery method

184659 ÷ 789 = 234 + remainder 33

		ΔQ	Quotient	
184659	Dividend			
-) 78900	Divisor			
105759		+100	100	
-) 78900				
26859		+100	200	
-) 78900				
52041	Negative remainder	+100	300	
+) 78900				
26859		-100	200	Recovery operation
-) 7890				
18969		+10	210	
-) 7890				
11079		+10	220	
-) 7890				
3189		+10	230	
-) 7890				
-4701	Negative remainder	+10	240	
+) 7890				
3189		-10	230	Recovery operation
-) 789				
2400		+1	231	
-) 789				
1611		+1	232	
-) 789				
822		+1	233	
-) 789				
33		+1	234	
-) 789				
-756	Negative remainder	+1	235	
+) 789				
33	Remainder	-1	234	Recovery operation

Example 2 Non-recovery method

184659 ÷ 789 = 234 + remainder 33

		ΔQ	Quotient
184659	Dividend		
-) 78900	Divisor		
105759		+100	100
-) 78900			
26859		+100	200
-) 78900			
-52041	Negative remainder	+100	300
+) 7890			
-44151		-10	290
+) 7890			
-36261		-10	280
+) 7890			
-28371		-10	270
+) 7890			
-20481		-10	260
+) 7890			
-12591		-10	250
+) 7890			
-4701		-10	240
+) 7890			
3189	Positive remainder	-10	230
-) 789			
2400		+1	231
-) 789			
1611		+1	232
-) 789			
822		+1	233
-) 789			
33	Remainder	+1	234

Processing procedure

a) Check the OPE and B.F.P.ACC data. If the OPE data is zero, it is assumed that the operation has overflowed. The OPE data is converted into an unsigned mantissa.

b) Take the complement of the exponent of the OPE data and add the exponents.

c) Perform division of the mantissas.

c-1) Both dividend and divisor are 24 bits. It is possible that the division of the mantissa involves a carry from the decimal point. First, the OPE data is subtracted from the B.F.P.ACC data to obtain the most significant bit of the quotient. The operation on the next digit jumps to addition or subtraction depending on whether the quotient equals 0 or 1, respectively. The value of 1 is left in the quotient. The divisor is divided by two and stored in (WS1, WS2, WS3, WS4).

Example: 0.413 ÷ 0.211 → 1.957 Wrong
→ 0.1957 x 10 Correct

c-2) Perform a non-recovery operation based on the flowchart shown in Figure 3-3.

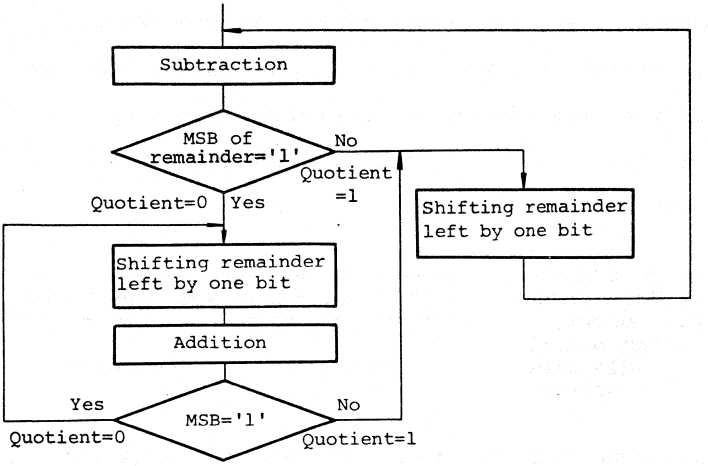


Fig. 3-3 Non-recovery method flowchart

As the result of addition/subtraction, the MSB of the remainder is cleared and shifted left by one bit to perform the operation on the next bit. At this time, the MSB is provided with the quotient.

Then:

1. If the remainder does not become negative after subtraction, the MSB equals 0. Therefore, the inverted value becomes the quotient. If the remainder is negative, the inverted value becomes the quotient and the MSB equals 1.

2. If the sum does not become positive after addition, the MSB equlas 1. Otherwise the MSB equals 0. In both cases, the inverted value becomes the quotient.

In both 1 and 2, the inverted value of the MSB becomes the quotient. It is, therefore, necessary by to set 1 in the MSB of WS1 when the divisor is divided by two in c-1. Figure 3-4 shows the memory configuration for the mantissa division subroutine.

WD1 Quotient (high order)
WD2 Quotient (middle order)
WD3 Quotient (low order)
WD4 Remainder (high order)
WD5 Remainder (middle order)
WD6 Remainder (low order)

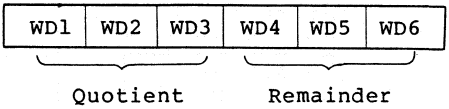


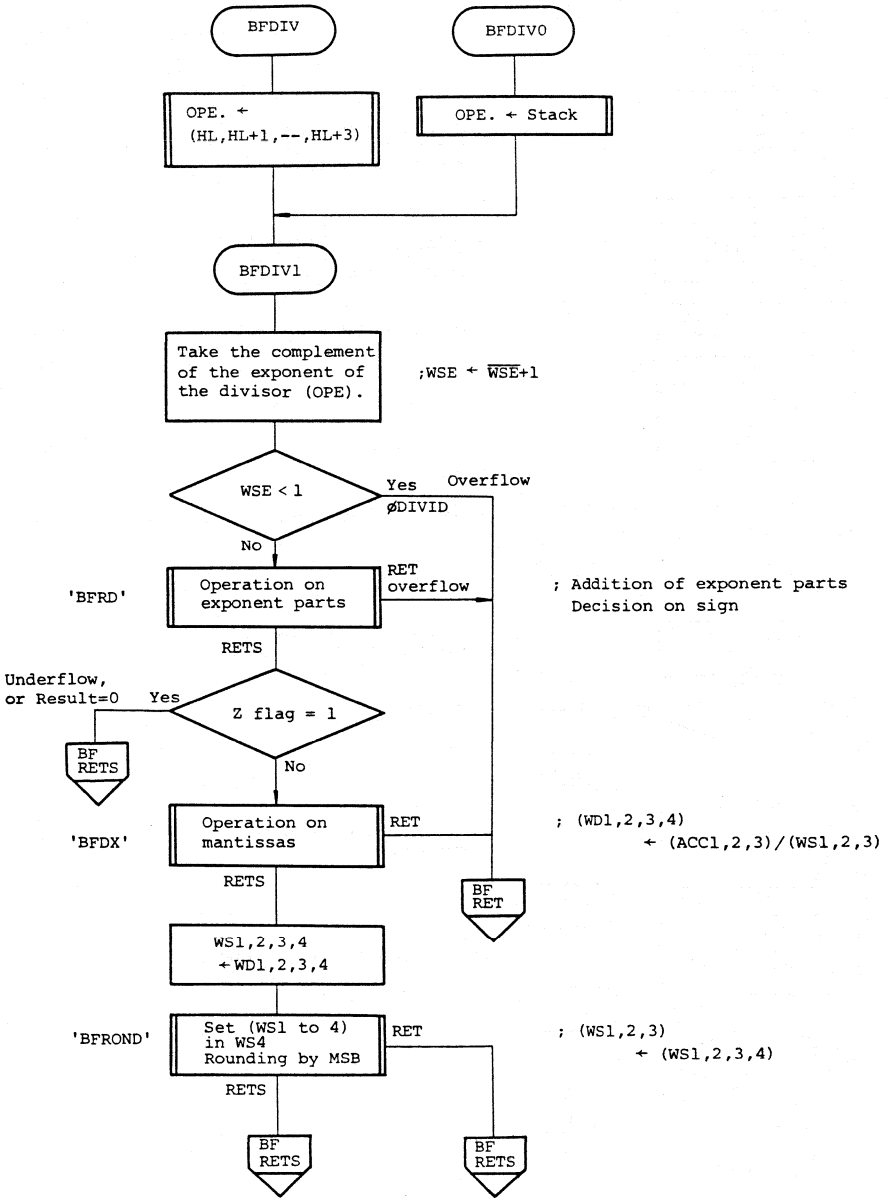
Fig. 3-4 Mantissa division subroutine memory configuration

As the result of the operation, the quotient is saved in the MSB of WD4. When the six bytes are sifted left by one bit, the quotient and the remainder are shifted left simultaneously.

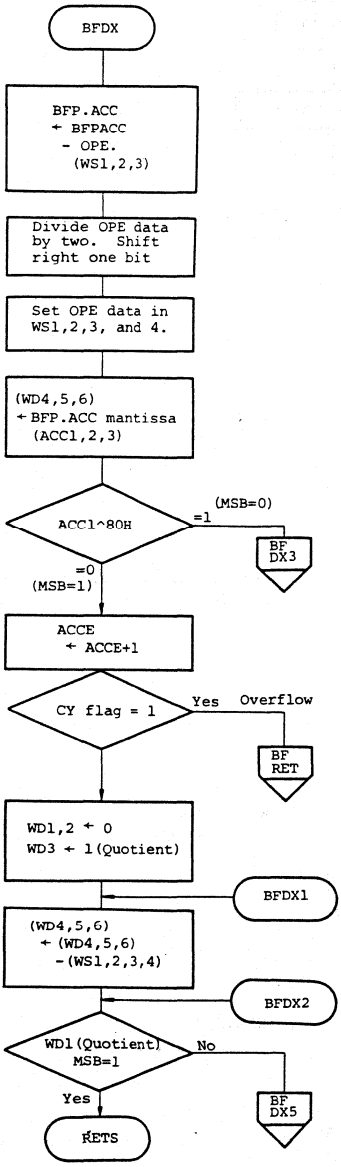
The division is complete when 1 is found in the MSB of WD1.

This subroutine is shown in the following flowchart.

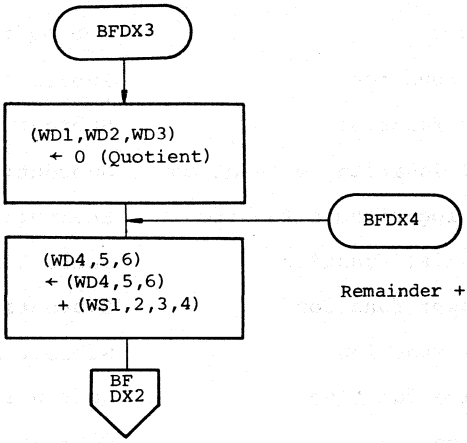
BFDIV subroutine (Division)

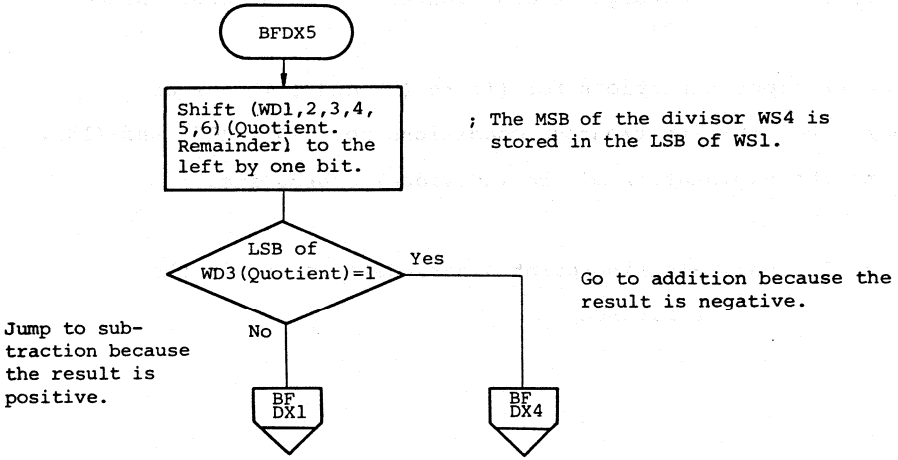


BFDX subroutine (Division of mantissas)



BFDX subroutine (Division of mantissas)





3-2 Function Subroutine

The following function subroutines are provided:

(1)	SIN:	Sine function	Subroutine
(2)	COS:	Cosine function	Subroutine
(3)	TAN:	Tangent function	Subroutine
(4)	LNX:	Natural logarithmic function	Subroutine
(5)	LOG:	Common logarithmic function	Subroutine
(6)	EXP:	Exponential function	Subroutine
(7)	ARCTAN:	Arctangent function	Subroutine
(8)	ARCSIN:	Arcsine function	Subroutine
(9)	ARCCOS:	Arccosine function	Subroutine
(10)	POWER:	Power function	Subroutine
(11)	SQR:	Square root	Subroutine
(12)	TRACA:	(Polar coordinates	Rectangular coordinates)
(13)	TRACB:	(Rectangular coordinates	Polar coordinates)

The input/output conditions for (1) to (9) and (11) are as follows. For the input/output conditions for (10), (12) and (13), refer to the explanation of the individual subroutine.

Input condition: Floating-point binary data is set in the B.F.P.ACC.

Output condition:

RET: The result of the operation has overflowed. '1' is set in the CY flag.

RETS: A normal result is stored in the B.F.P.ACC. '0' is set in the CY flag. If the result is 0, '1' is stored in the Z flag. Otherwise, '0' is stored in the Z flag. Note that the ARCTAN Subroutine in (7) above does not cause an overflow. The NOP instruction is required after calling this subroutine.

3-2-1 SIN (Sine Function) subroutine

1) Processing

Obtain the sine of the B.F.P.ACC data and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for function operations explained above.

3) Output condition

Refer to the output condition for function operations explained above.

4) Subroutines used

See Table 4-4.

5) Stack depth

Max. 13

6) Coding sequence

```
(
CALL  SIN   ; SIN (X)
GJMP  ERROR; Jump to overflow processing
)
```

7) Processing time

Max. About 30 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC 81H, 00H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

The series

$$SIN\ X = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \ldots \quad (3.1)$$

(Unit: Radian)

is used to obtain the value of a sine function. This subroutine calculates the series to the fifth term.

Processing procedure

a) SIN (-X) = -SIN (X), X ← -X

b) Scale the data between 0 and 1 in order to assign it to formula (3.1). For this procedure, the following is utilized: (See Table 3-2.)

$SIN\ (2n\pi X) = SIN\ (X),\ \theta = X/2\pi - INT\ (X/2\pi)$

Table 3-2 Scaling

θ		x
0 to 1/4	←	0 to π/2
1/4 to 1/2	←	π/2 to π
1/2 to 3/4	←	π to 3/2π
3/4 to 1	←	3/2π to 2π

c) Further scale the data between -1/4 and 1/4.

c-1 $\theta \leftarrow 1/4 -$

If θ is negative: $(X > \pi/2), \theta \leftarrow \theta + 1/2$

c-2 If θ is negative: $(X > 3/2\pi),\ \theta \leftarrow -\theta$

c-3 $\theta \leftarrow \theta + 1/4$

c-4 If $X > \pi/2$: $\theta \leftarrow -\theta$

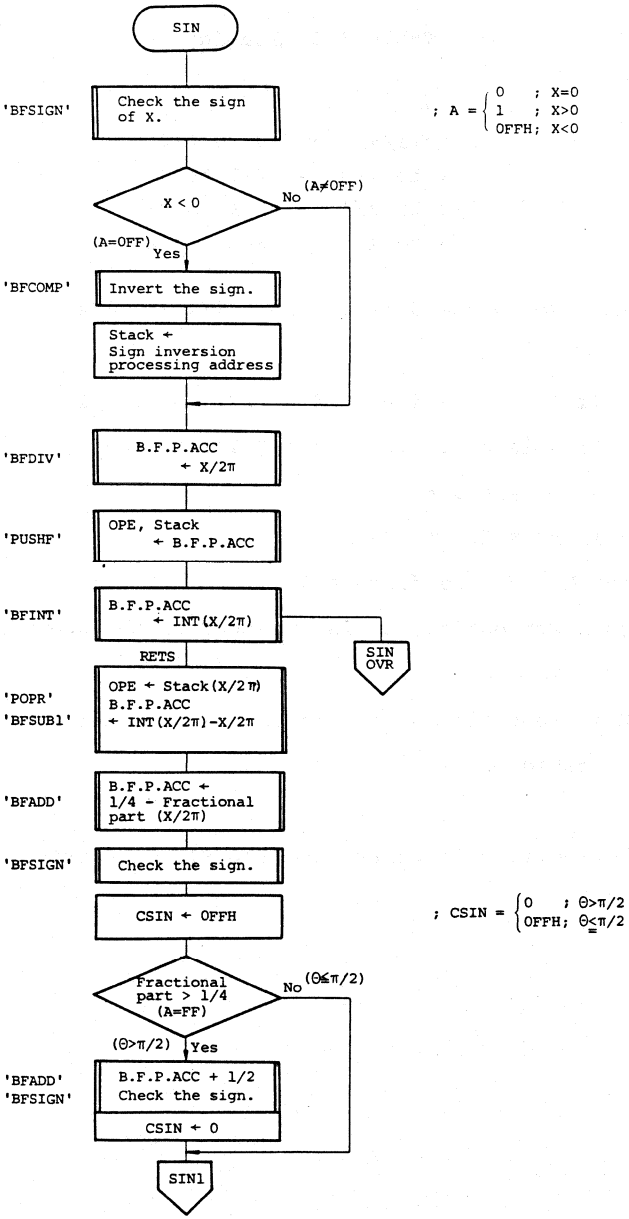
d) Assign $X = 2\pi\theta$ to formula 3.1.

In this subroutine, 2 is included in the coefficients of the series.

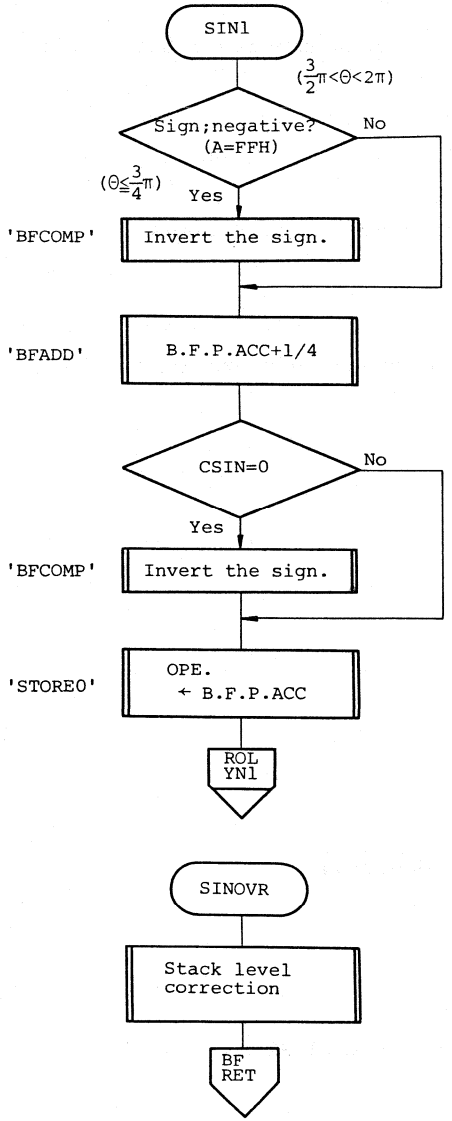
e) Ix X is less than 0, invert the sign of the result obtained in d) based on a).

The following is a flowchart for the SIN subroutine.

SIN subroutine



SIN subroutine



; To polynominal processing

3-2-2 COS (Consine Function) subroutine

1) Processing

Obtain the cosine of the B.F.P.ACC data and store the result in the B.F.P.ACC.

2) Input conditon

Refer to the input condition for function operations.

3) Output condition

Refer to the output condition for function operations.

4) Subroutines used

See Table 4-5.

5) Stack depth

Max. 13

6) Coding sequence

```
        ⟨
CALL  COS   ; COS (X)
GJMP  ERROR; Jump to overflow processing
        ⟩
```

7) Processing time

About 30 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC 81H, 00H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

The formula

$$\text{COS } X = \text{SIN } (\frac{\pi}{2} \pm X)$$

is used. (Unit: Radian)

Processing procedure

a) Calculate $X + \pi/2$.

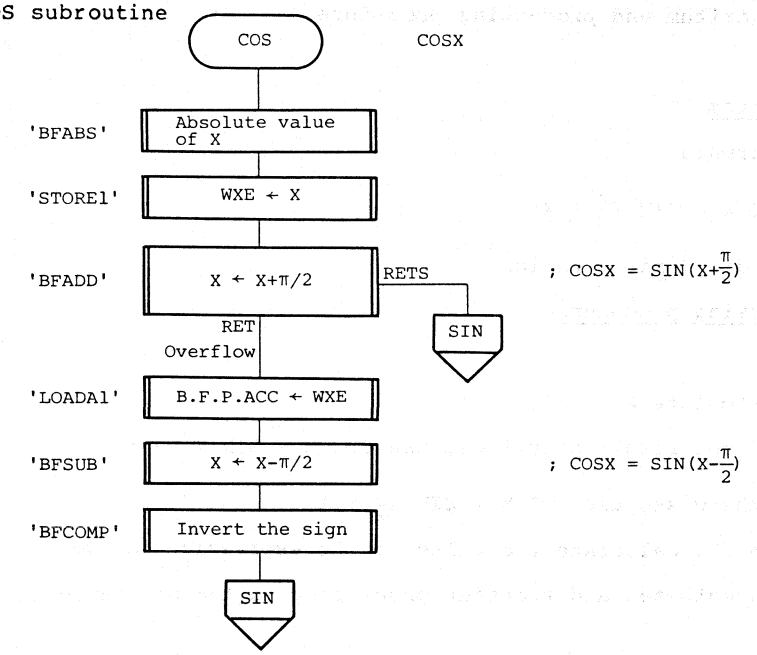
If the result overflows, use $\text{COS } X = \text{SIN } (\frac{\pi}{2} - X)$.

Otherwise, use $\text{COS } X = \text{SIN } (\frac{\pi}{2} + X)$.

b) In a), calculate the value of the expression in the parentheses and transfer processing to the SIN subroutine.

The following is a flowchart of the COS subroutine.

COS subroutine



3-2-3 TAN (Tangent Function) subroutine

1) Processing

Obtain the tangent of the B.F.P.ACC data and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for function operations.

3) Output condition

Refer to the output condition for function operations.

4) Subroutines used

See Table 4-6.

5) Stack depth

Max. 16

6) Coding sequence

```
        )
        (
CALL  TAN   ;  TAN (X)
GJMP  ERROR;  Jump to overflow processing.
        )
        (
```

7) Processing time

Max. About 65 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC 81H, 00H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

The formula

$$\text{TAN } X = \frac{\text{SIN } X}{\text{COS } X}$$

is used. (Unit: Radian)

Processing procedure

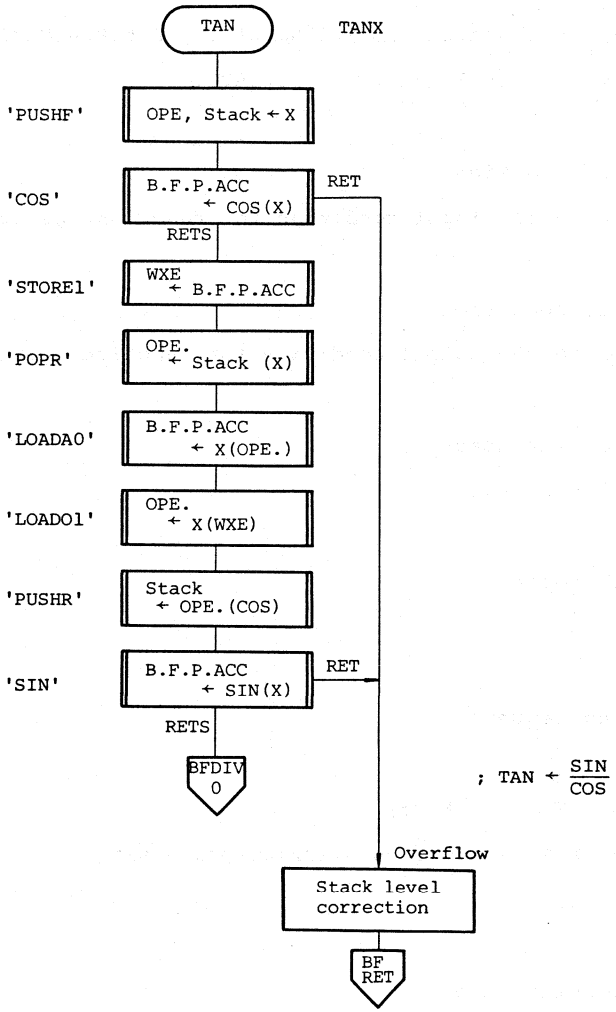
a) Obtain COS X. If the result overflows, terminate the processing.

b) Obtain SIN X. If the result overflows, terminate the processing.

c) Obtain $\frac{\text{SIN X}}{\text{COS X}}$

The following is a flowchart of the TAN subroutine.

TAN subroutine



3-2-4 LNX (Natural Logarithmic Function) subroutine

1) Processing

Obtain the natural logarithm of the B.F.P.ACC value and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for function operations.

3) Output condition

Refer to the output condition for function operations.

4) Subroutine used

See Table 4-7.

5) Stack depth

Max. 13

6) Coding sequence

```
        )
        (
CALL    LNX   ;  LN (X)
GJMP    ERROR;  Jump to overflow processing.
        )
        (
```

7) Processing time

Max. About 16 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC FFH, 80H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

The expansions

$$LN\ (1 + X) \cong X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \ldots \quad (3.2)$$

$$LN\ (1 - X) = -X - \frac{X^2}{2} - \frac{X^3}{3} - \frac{X^4}{4} + \ldots \quad (3.3)$$

can be used when -1 X 1.

Determine $Y = \frac{X - 1}{X + 1}$ ... (3.4)

Then, $-1 < Y < 1$ for $0 < X < \infty$, and

$$X = \frac{1 + Y}{1 - Y} \quad \ldots (3.5)$$

By (3.2), (3.3) and (3.5)

$$LNX = LN\ (1 + Y) - LN\ (1 - Y)$$

$$= 2\ (Y + \frac{Y^3}{3} + \frac{Y^5}{5} + \ldots) \quad \ldots (3.6)$$

Expression (3.6) is applicable to any X (> 0).

Processing procedure

a) In Expression (3.4), the rate of change of Y slows down as X becomes larger or smaller. Processing is performed by making the exponent of X (ACCE) equal to 80H (2^0).

b) If the B.F.P.ACC value is X_1, then:

i) When ACCE ≧ 80H

$$LNX = LN\ (X_1 \cdot 2^{ACC-80H})$$

$$= LNX_1 + (ACC - 80H) \cdot LN2 \quad \ldots (3.7)$$

ii) When ACCE < 80H

$$LNX = LN\ (X_1 \cdot 2^{80H-2ACCE})$$

$$= LNX_1 - (ACCE - 80H) \cdot LN2 \quad \ldots (3.8)$$

b-1) Obtain LNX_1 by (3.4) and (3.6).

b-2) Obtain ACCE-80H. That is, convert the difference between the exponent of X and 80H (2^0) (fixed-point binary number) into a B.F.P.ACC number (floating-point binary number).

b-3) Obtain (ACCE-80H) x LN2.

b-4) Compute the following:

i) When X ≧ 80H

Result of b-1) + Result of b-3) by (3.7).

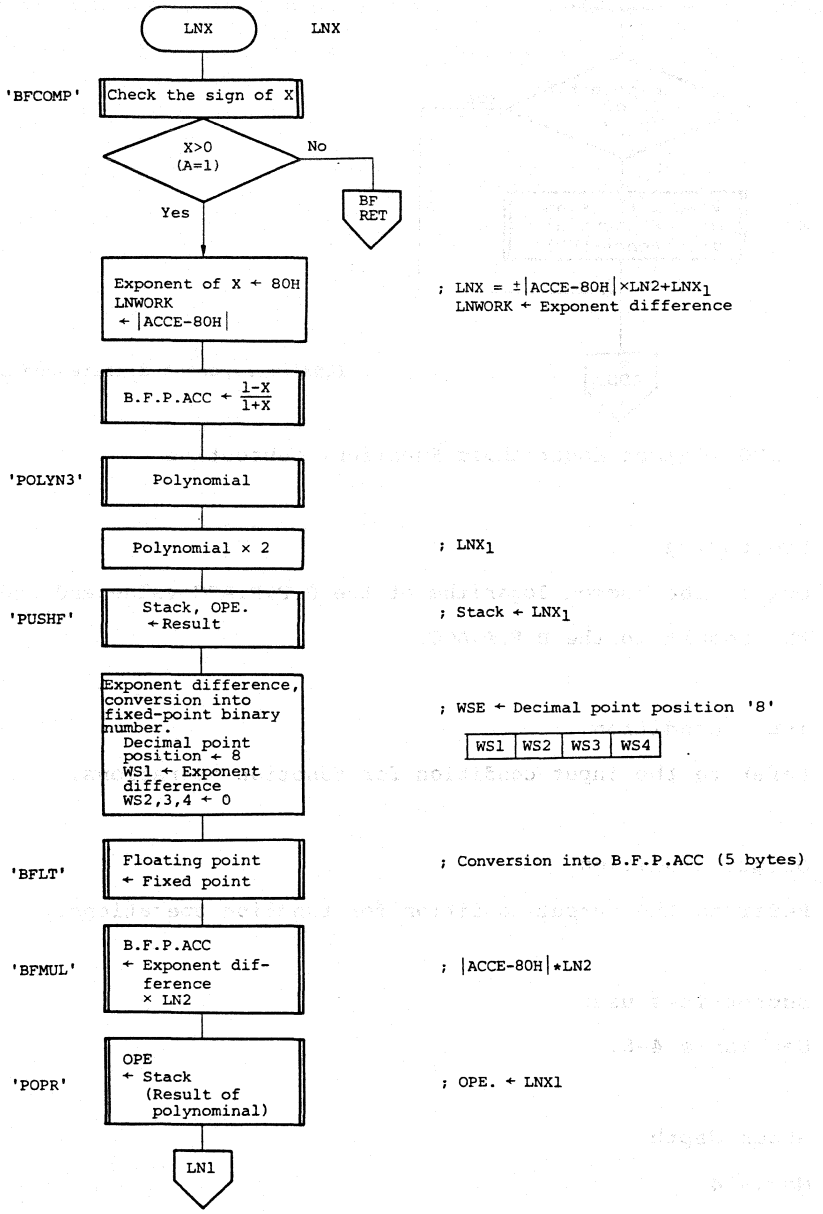
ii) When X < 80H

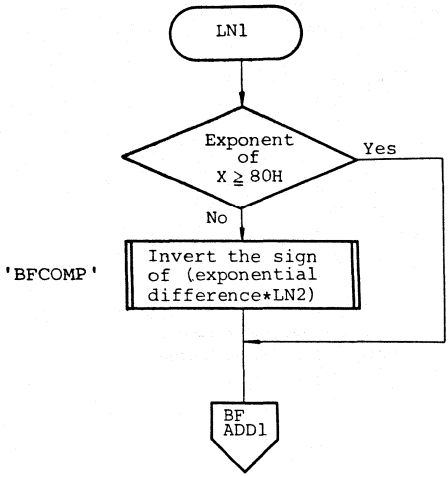
Result of b-1) - Result of b-2) by (3.8).

Store the result in the B.F.P.ACC.

The following is a flowchart for this subroutine.

LNX subroutine





3-2-5 LOG (Common Logarithmic Function) subroutine

1) Processing

Obtain the common logarithm of the B.F.P.ACC value and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for function operations.

3) Output condition

Refer to the output condition for function operations.

4) Subroutines used

See Table 4-8.

5) Stack depth

Max. 14

6) Coding sequence

```
CALL  LOG  ;  LOG (X)
GJMP  ERROR;  Jump to overflow processing.
```

7) Processing time

Max. About 20 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC FFH, 80H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

Use the following the formula:

$$\text{LOG } X = \frac{\text{LN } X}{\text{LN } 10}$$

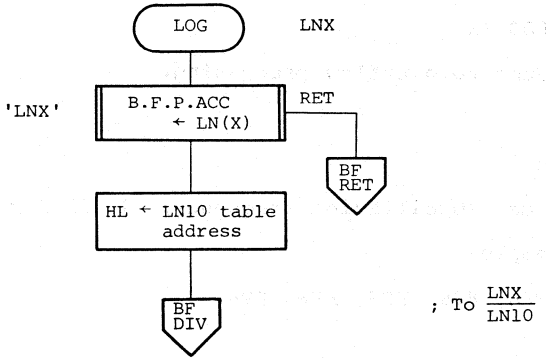
Processing procedure

a) Obtain LN X. If the result overflows, terminate the processing.

b) Compute $\frac{\text{LN } X}{\text{LN10}}$.

The following is a flowchart of this subroutine.

LOG subroutine



3-2-6 EXP (Exponential Function) subroutine

1) Processing

Obtain the exponential function of the B.F.P.ACC value and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for function operations.

3) Output condition

Refer to the output condition for function operations.

4) Subroutines used

See Table 4-9.

5) Stack depth

Max. 12

6) Coding sequence

```
CALL  EXP   ;  EXP (X)
GJMP  ERROR;  Jump to overflow processing.
```

7) Processing time

Max. About 28 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC 86H, 80H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

$EXP(X) = 2^{(X * Log2(E))}$ ... (3.8)

where, E is _____

For example, 2^Y is calculated by the following formula:

$2^Y = 2^{INT(Y)} \times 2^{(Y - INT(Y))}$ ... (3.9)

Processing procedure

a) Obtain X * Log2E. Overflow if exponent 89.

b) Obtain INT (X * Log2E).

c) Overflow if the integer part (TMPWSL) is 7E, 7FH.

d) Obtain the latter half of expression (3.9).

$2^{(Y - INT(Y))} = EXP(X - LN2 * (INT(X * Log2E)))$ ... (3.10)

Obtain the value by assigning the value in the EXP ( ) to the following formula:

$$EXP\ (Z) = 1 + \frac{Z}{1!} + \frac{Z^2}{2!} + \frac{Z^3}{3!} + \cdots$$

e) Multiply the result by $2^{INT\ (Y)}$

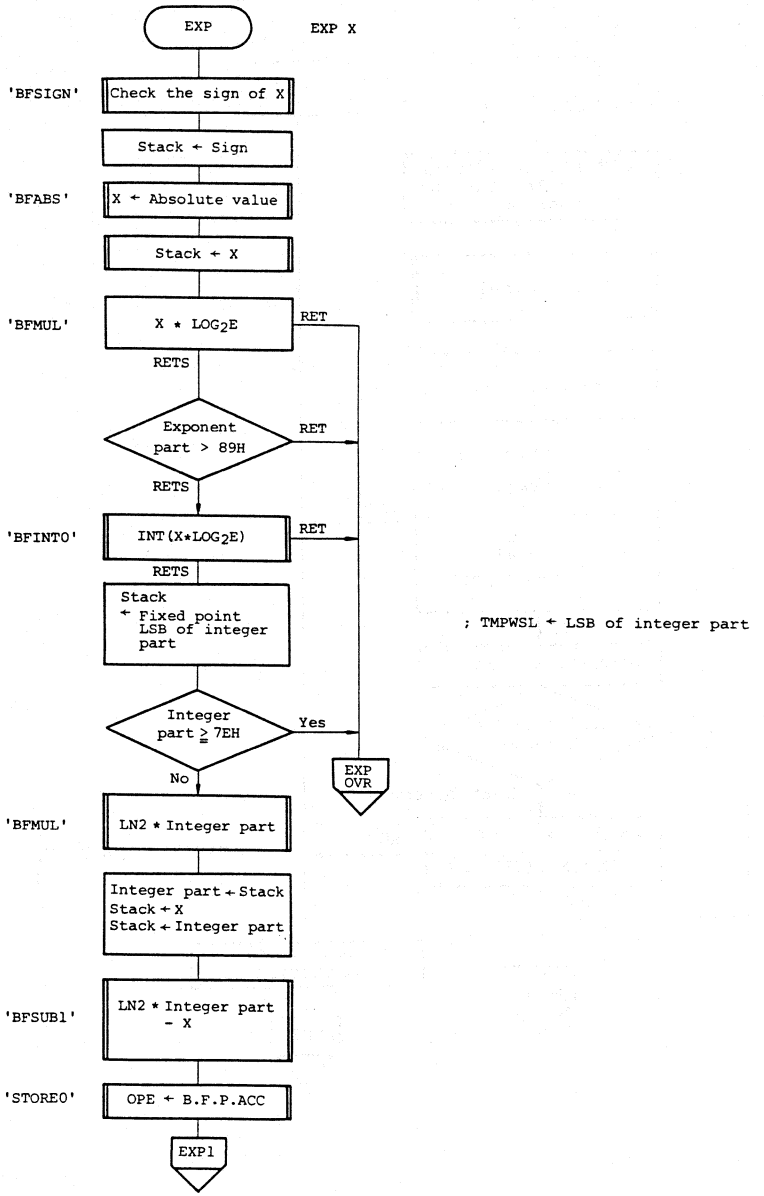
That is, add INT (Y) to the exponent part of the result obtained in d).

f) If X < 0, obtain the inverse number of the result by the following expression:

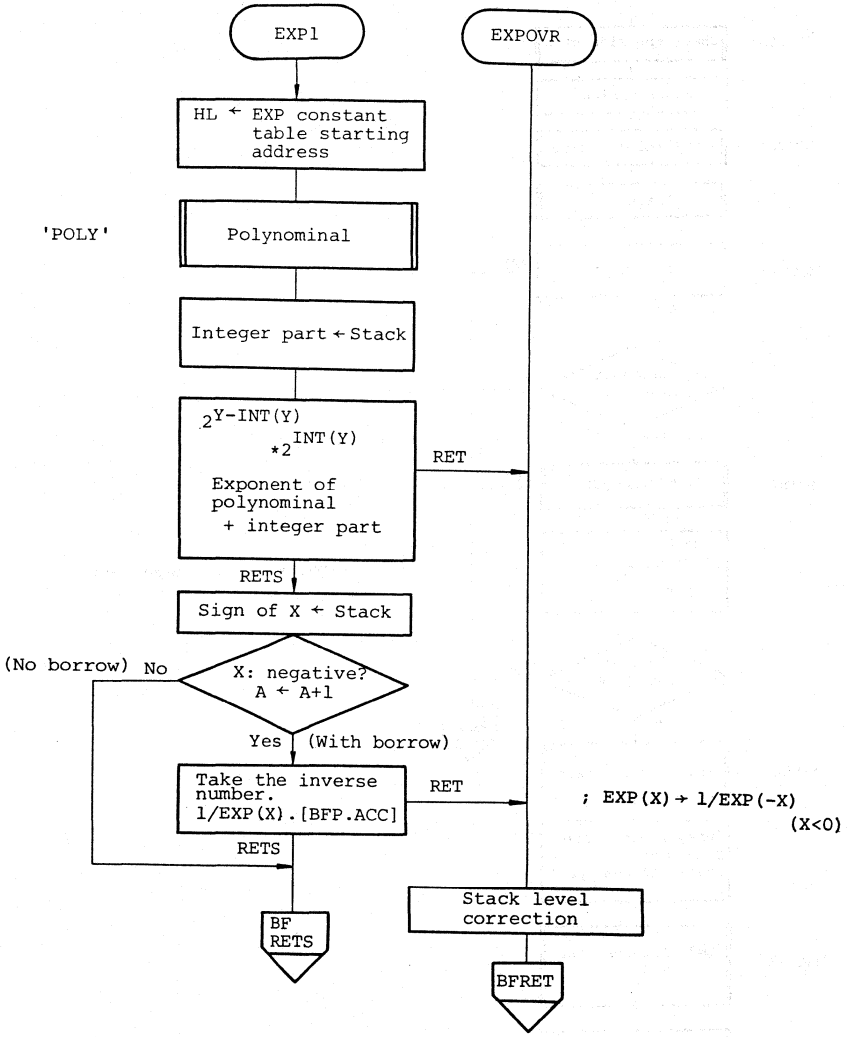
$$EXP\ (-|X|) = 1/EXP\ (|X|)$$

The following is a flowchart of this subroutine.

EXP subroutine



EXP subroutine



3-2-7 ARCTAN (Arc Tangent) subroutine

1) Processing

Obtain the arc tangent of the B.F.P.ACC value and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for functional operations.

3) Output condition

Refer to the output condition for functional operations.

4) Subroutines used

See Table 4-10.

5) Stack depth

Max. 16

6) Coding sequence

```
CALL  ARCTAN;  ARCTAN (X)
NOP          ;  Dummy for RETS
```

7) Processing time

Max. About 30 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC 90H, 80H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

For $|X| \leqq 1$: $\text{ARCTAN } X = X - \frac{X^3}{3} + \frac{X^5}{5} - \frac{X^7}{7} + \ldots$ (3.11)

For $|X| > 1$: $\text{ARCTAN } X = \pi/2 - \text{ARCTAN } \frac{1}{|X|}$ (3.12)

The result is expressed in radians.

Processing procedure

a) When $X < 0$, invert the sign of X by

$\text{ARCTAN } (X) = -\text{ARCTAN } (-X)$ (3.13)

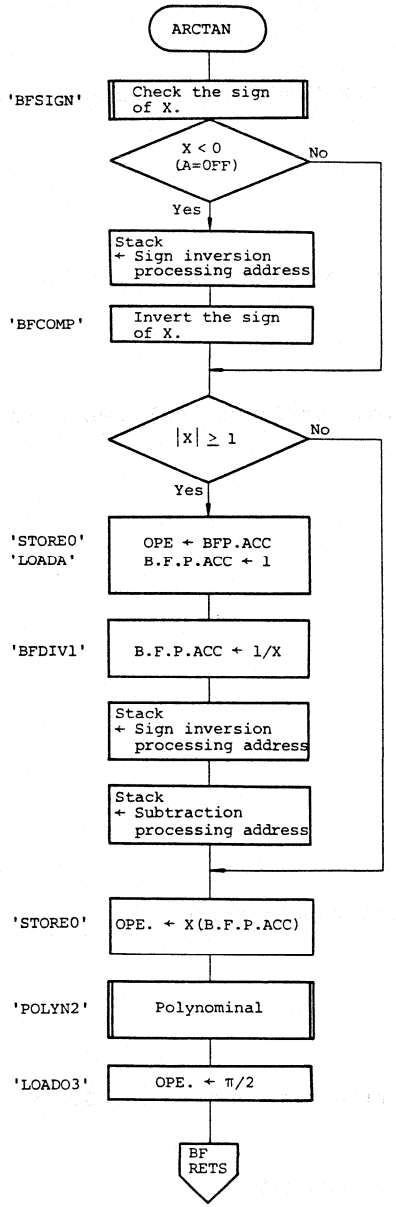
b) When $|X| \leqq 1$, use expression (3.11)

When $|X| > 1$, use expression (3.11) with $1/|X|$ and subtract the result from /2 by expression (3.12).

c) If X is less than 0 in a), invert the sign of the result obtained in b) by expression (3.13).

The following is a flowchart for this subroutine.

ARCTAN subroutine



; To invert ARCTAN 1/|X| - π/2

; After obtaining ARCTAN 1/|X|
; To calculate ARCTAN 1/|X| - π/2

; Return to the address in the stack by RETS

3-3-8 ARCSIN (Arc Sine) subroutine

1) Processing

Obtain the arc sine of the B.F.P.ACC value and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for functional operations.

3) Output condition

Refer to the output condition for functional operations.

4) Subroutines used

See Table 4-11.

5) Stack depth

Max. 21

6) Coding sequence

```
CALL  ARCSIN;  ARCSIN (X)
GJMP  ERROR ;  Jump to overflow processing
```

7) Processing time

Max. About 75 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC 80H, 80H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

$$\text{ARCSIN } X = \text{ARCTAN } \frac{X}{\sqrt{1 - X^2}} \quad \ldots \quad (3.14)$$

Provided:

When X = 1 $\pi/2$

When X = -1 $-\pi/2$

When $|X| > 1$ Overflow

In other cases, use expression (3.14). The result is expressed in radians.

Processing procedure

a) Obtain $\sqrt{1 = X^2}$. In the case of an overflow, terminate the processing.

b) Obtain $X/\sqrt{1 - X^2}$.

In the case of an overflow:

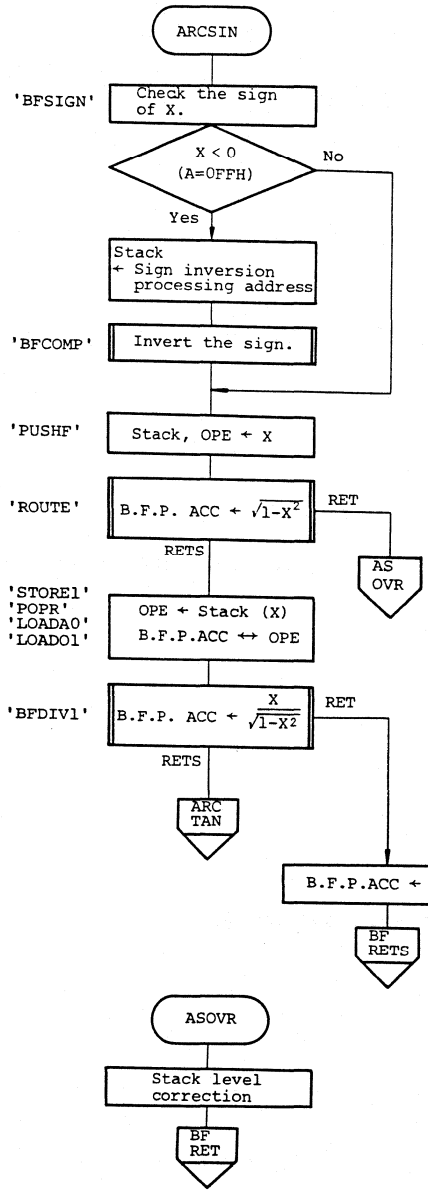
If X = 1 Result = $\pi/2$

If X = -1 Result = $-\pi/2$

c) Obtain $\text{ARCTAN } \frac{X}{\sqrt{1 = X^2}}$

The following is a flowchart for this subroutine.

ARCSIN subroutine



; ARCSIN(-X)=-ARCSIN(X)

; ARCTAN $\frac{X}{\sqrt{1-X^2}}$

; Jump to sign inversion processing by RETS depending on the sign of X.

X=1 ARCSIN ← π/2
X=-1 ARCSIN ← -π/2

3-2-9 ARCCOS (Arc Cosine) subroutine

1) Processing

Obtain the arc cosine of the B.F.P.ACC value and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for functional operations.

3) Output condition

Refer to the output condition for functional operations.

4) Subroutines used

See Table 4-12.

5) Stack depth

Max. 21

6) Coding sequence

```
CALL  ARCCOS;  ARCCOS (X)
GJMP  ERROR ;  Jump to overflow processing.
```

7) Processing time

Max. About 70 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC 80H, 80H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

$$\text{ARCCOS } X = \begin{cases} \text{ARCTAN } \dfrac{\sqrt{1 - X^2}}{X} & (X > 0) \\ \pi + \text{ARCTAN } \dfrac{\sqrt{1 - X^2}}{X} & (X < 0) \end{cases}$$

When X = 0, make the result $\pi/2$.

The above expression is used and the result is expressed in radians.

Processing procedure

a) Check and store the sign of X.

b) Obtain $\sqrt{1 - X^2}$. In the case of an overflow, terminate processing.

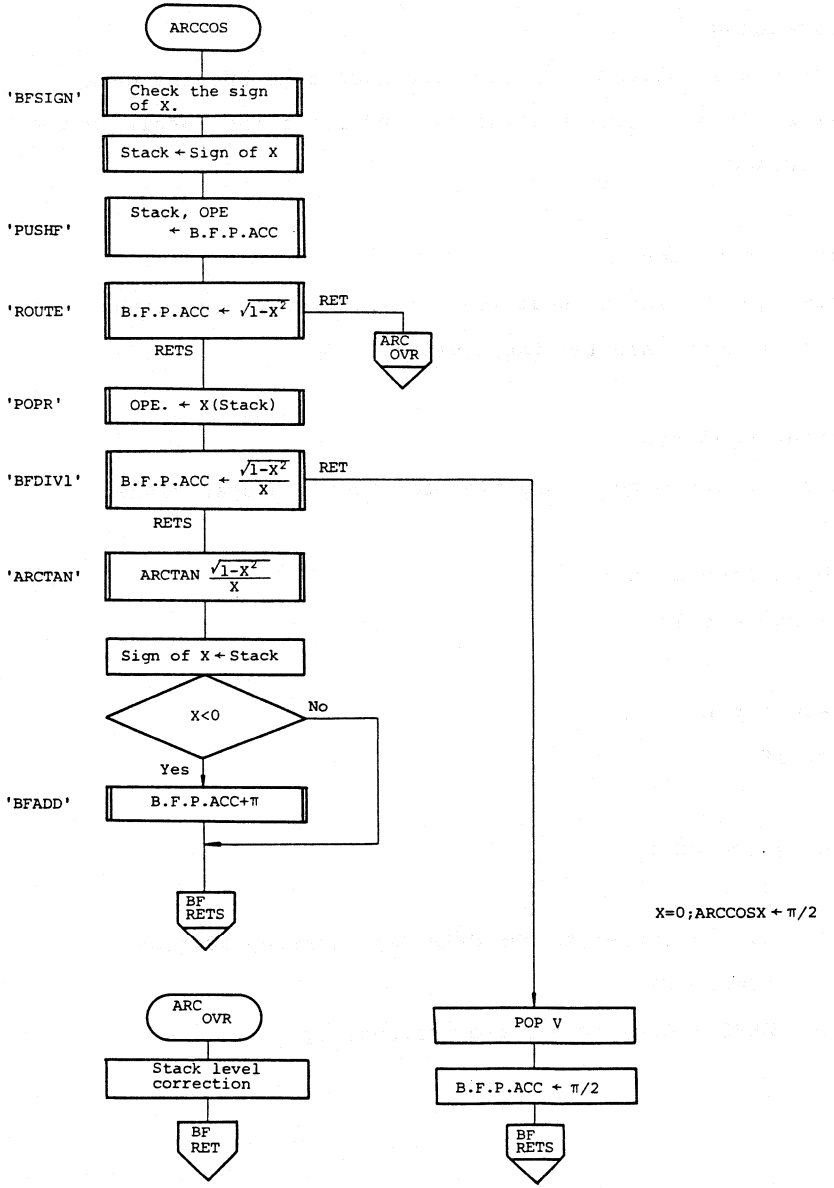
c) Obtain $\sqrt{1 - X^2}/X$. In the case of an overflow, make the result $\pi/2$.

d) Obtain $\text{ARCTAN } \dfrac{\sqrt{1 - X^2}}{X}$.

e) If X is less than 0, calculate
$\text{ARCTAN } \dfrac{\sqrt{1 - X^2}}{X} + \pi$

The following is a flowchart for this subroutine.

ARCCOS subroutine



3-2-10 POWER (Power Function) subroutine

1) Processing

Obtain the value of X^Y using the four-byte operand data (Y) and the B.F.P.ACC value (X) and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for arithmetic operations, because there are two input values.

3) Output condition

Refer to the output condition for function operations.

4) Subroutines used

See Table 4-13.

5) Stack depth

Max. 16

6) Coding sequence

```
      )
LXI   H, - ; Four-byte OPE data (Y) starting address
CALL  POWER; XY
GJMP  ERROR; Jump to overflow processing
      )
```

7) Processing time

Max. About 46 ms (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC FFH, 80H, FFH, FFH, FFH

OPE. 10H, 7FH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

When Y = 0 $X^Y = 1$

When X = 0 Overflow for Y < 0

$X^Y = 0$ for Y ≧ 0

When X = 0 and Y = 0 $X^Y = e^{Y * LN(X)}$ ... (3.15)

Processing procedure

a) If Y = 0; Jump to EXP. EXP (0) = 1

b) If X = 0 and Y < 0; Overflow.

c) If X = 0 and Y ≧ 0; 0

d) If X > 0; Obtain the value by using (3.15).

e) If X < 0;

When Y consists only of an integer part:

For $|X|^Y$, obtain the value using (3.15).

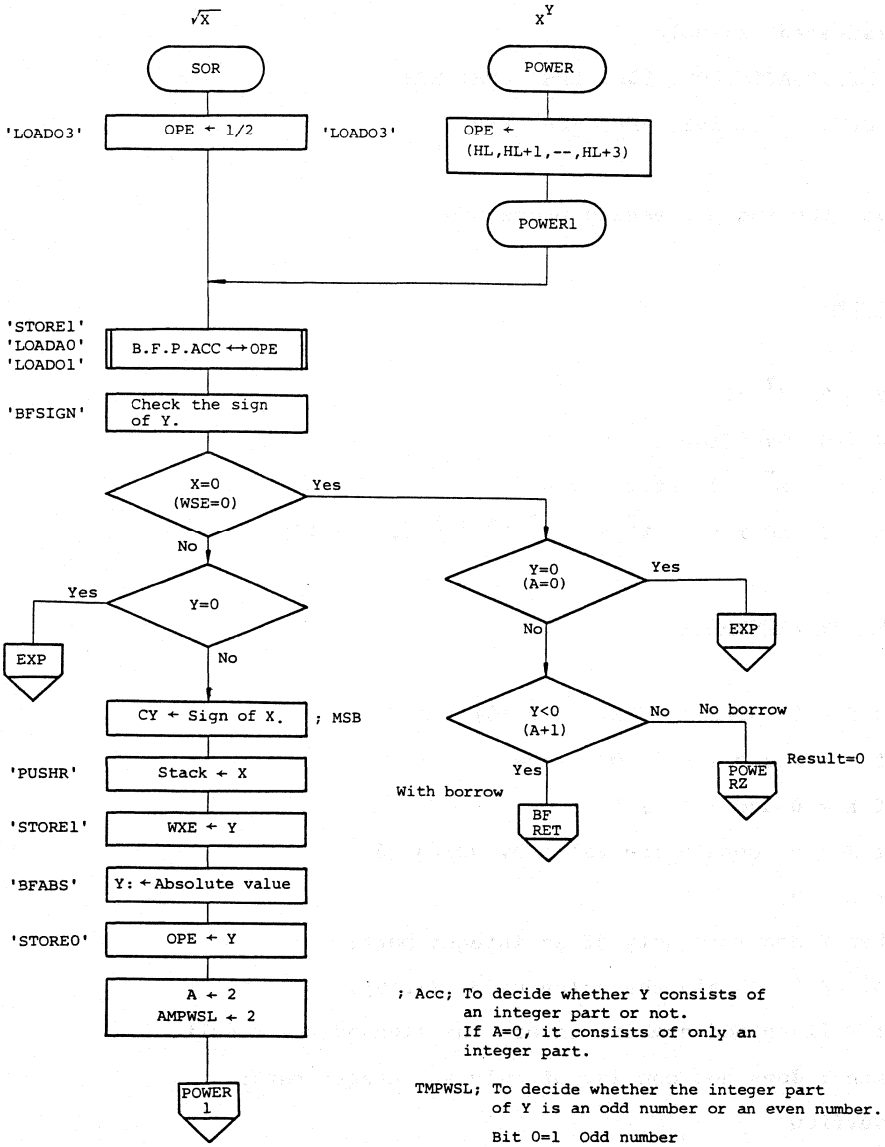
If Y if an odd number, invert the sign of the result.

When Y does not consist of only an integer part:

Overflow

The following is a flowchart for this subroutine.

POWER subroutine

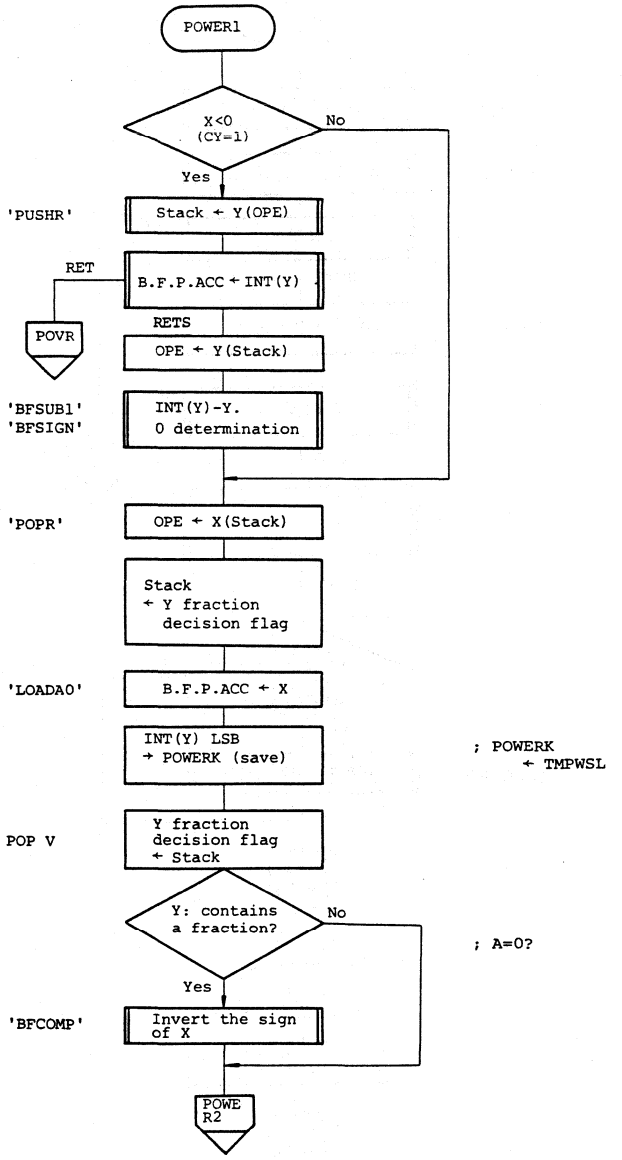


; Acc; To decide whether Y consists of an integer part or not.
If A=0, it consists of only an integer part.

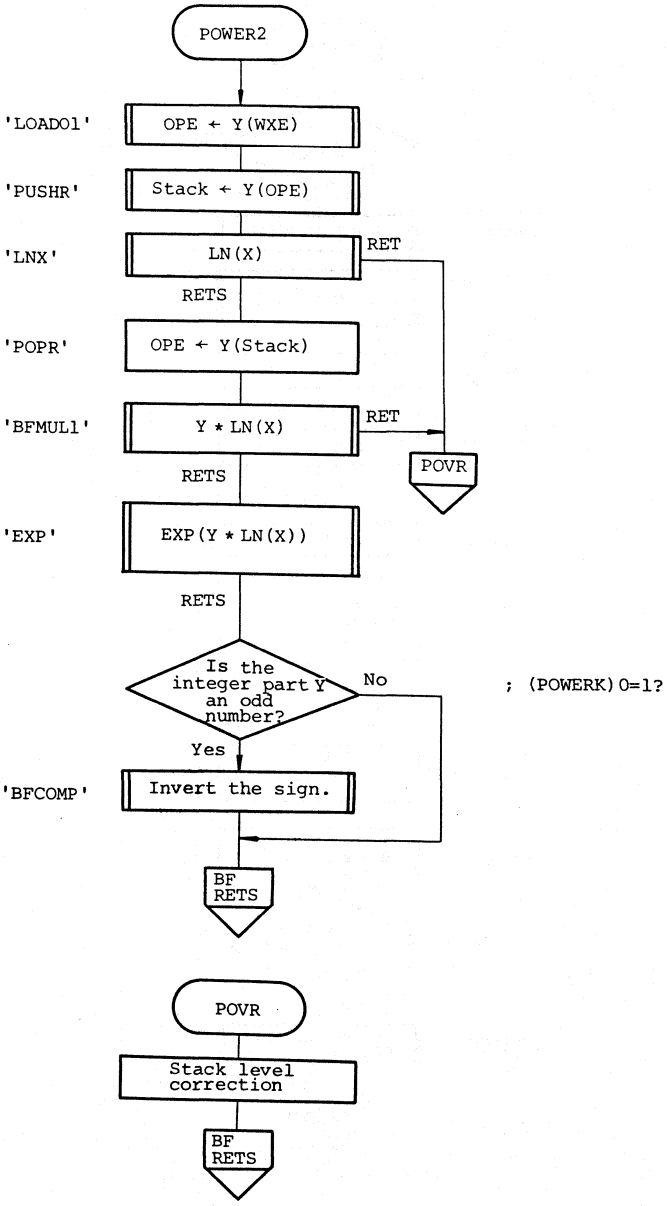
TMPWSL; To decide whether the integer part of Y is an odd number or an even number.

Bit 0=1 Odd number
Bit 0=0 Even number

POWER subroutine



POWER subroutine



3-2-11 SQR (Square Root) subroutine

1) Processing

Obtain the square root of the B.F.P.ACC data and store the result in the B.F.P.ACC.

2) Input condition

Refer to the input condition for function operations.

3) Output condition

Refer to the output condition for function operations.

4) Subroutines used

See Table 4-13.

5) Stack depth

Max. 16

6) Coding sequence

```
      )
      (
CALL  SQR   ; √X
GJMP  ERROR; Jump to overflow processing
      )
      (
```

7) Processing time

Max. About 46 ms. (Oscillation frequency: 11.0592 MHz)

Measurement example:

OPE. FFH, 80H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

Use the formula SQR (X) = $X^{1/2}$, (X > 0)

Processing procedure

a) The lefthand side of the above formula is a power function. Store 0.5 in the OPE.

b) The subsequent processing can be executed by the POWER subroutine.

3-2-12 TRACA (Polar coordinates ⟶ Rectangular coordinates) subroutine

1) Processing

Convert the polar coordinates (R, T) of the B.F.P.ACC data (R) and the four-byte operand data (T) into rectangular coordinates (X, Y) and store the resultant X and Y in the B.F.P.ACC and OPE, respectively.

2) Input condition

Refer to the input condition for arithmetic operations because there are two input values.

3) Output condition

RETS: After conversion, the coordinates X and Y are stored in the B.F.P.ACC and OPE, respectively.
'0' is stored in the CY flag.

RET : The conversion has resulted in an overflow.
'1' is stored in the CY flag.

4) Subroutines used

See Table 4-14.

5) Stack depth

Max. 16

6) Coding sequence

```
        )
        (
LXI   H, - ; Four-byte data (T) starting address
CALL  TRACA; Coordinate conversion processing
GJMP  ERROR; Jump to overflow processing
        )
        (
```

7) Processing time

Max. About 65 ms. (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC 81H, 80H, FFH, FFH, FFH

OPE. 80H, 7FH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

Use the following formula:

$$\begin{cases} X = R * \cos(T) \\ Y = R * \sin(T) \end{cases}$$

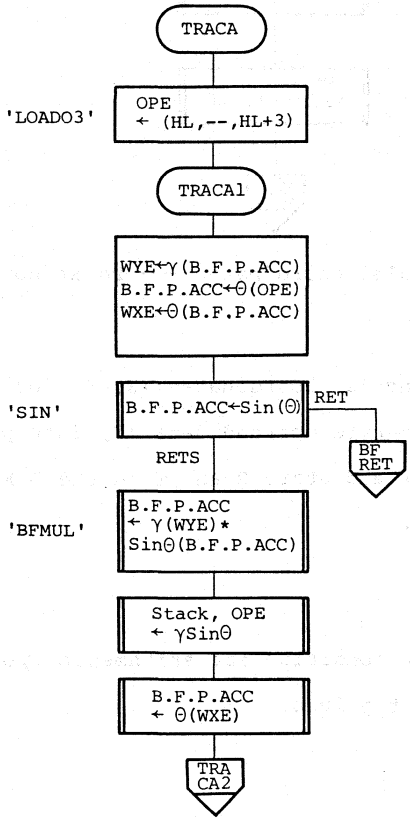
T: radians

The following is a flowchart for this subroutine.

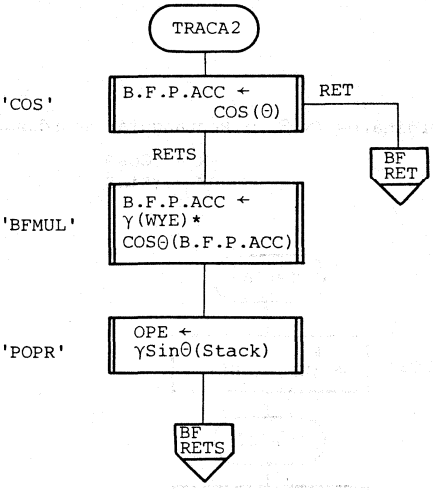
TRACA subroutine

Polar coordinates (γ,θ) → Rectangular coordinates (X,Y)

$$\begin{cases} X = \gamma \cos\theta \\ Y = \gamma \sin\theta \end{cases}$$



TRACA subroutine



3-2-13 TRACB (Rectangular coordinates ⟶ Polar coordinates) subroutine

1) Processing

Convert the rectangular coordinates (X, Y) for the B.F.P.ACC value (X) and four-byte operand data (Y) into polar coordinates (R, T) and store R and T in the B.F.P.ACC and OPE, respectively.

2) Input condition

Refer to the input condition for arithmetic operations since there are two input values.

3) Output condition

RET : The conversion has resulted in an overflow.
'1' is stored in the CY flag.

RETS: After conversion, the coordinates (X, Y) are stored in the B.F.P.ACC
and OPE, respectively.
'0' is stored in the CY flag.

4) Subroutines used

See Table 4-15.

5) Stack depth

Max. 19

6) Coding sequence

```
LXI   H, - ; Four-byte data (Y) starting address
CALL  TRACA; Coordinate conversion processing
GJMP  ERROR; Jump to overflow processing
```

7) Processing time

Max. About 10 ms. (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC	81H, 80H, FFH, FFH, FFH
OPE.	80H, 7FH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

Use the following formula:

$$\begin{cases} R = \sqrt{(X^2 + Y^2)} \\ T = \text{ARCTAN } (Y/X) \end{cases} \qquad T: \text{ radians}$$

Processing procedure

a) Obtain Y/X.

In the case of an overflow: If $Y < 0$, then $T = -\pi/2$

If $Y \geq 0$, then $T = \pi/2$

b) Obtain ARCTAN (Y/X).

If $X < 0$ and $Y < 0$, then T = ARCTAN (Y/X) $- \pi$

If $X < 0$ and $Y > 0$, then T = ARCTAN (Y/X) $+ \pi$

c) Obtain $X^2 + Y^2$.

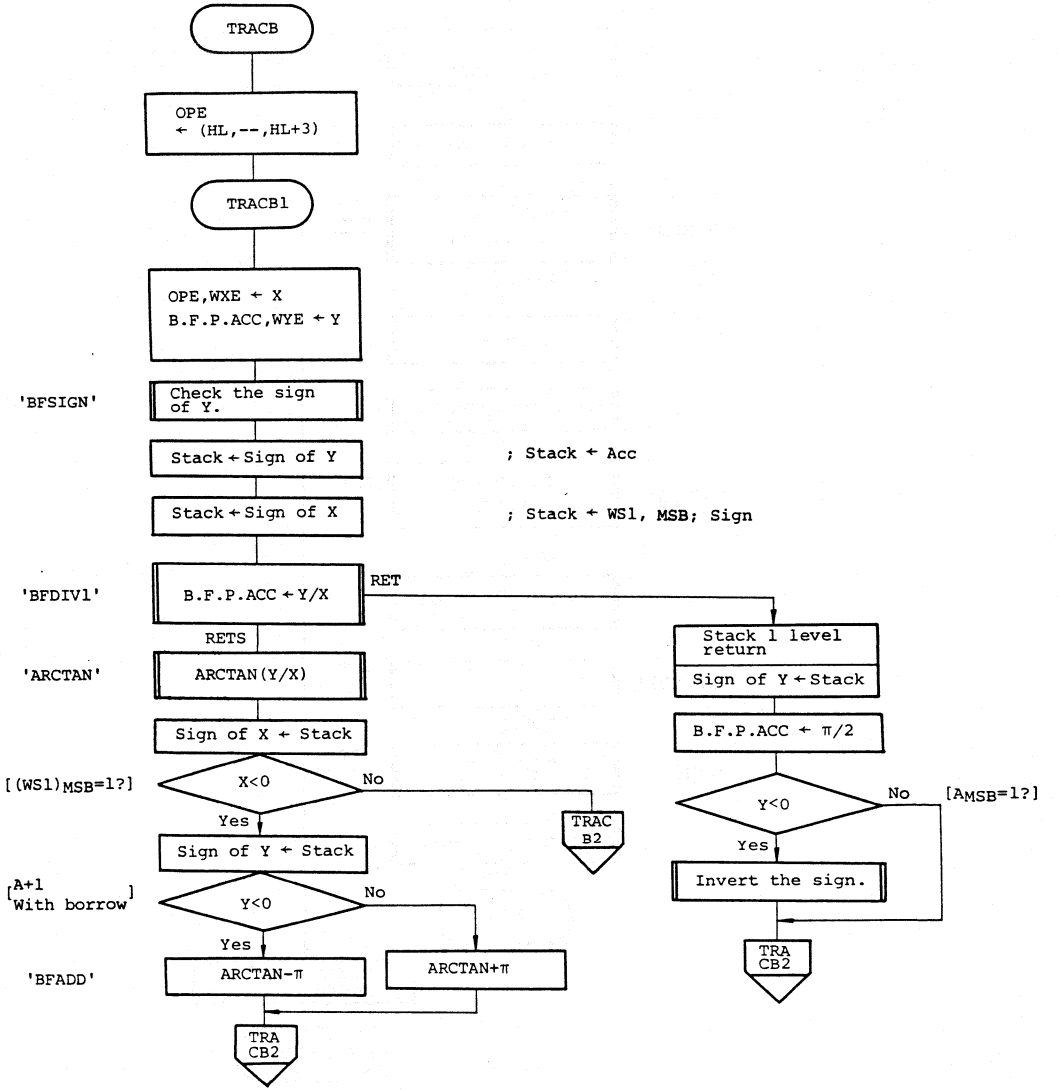
d) R = SQR $(X^2 + Y^2)$

The following is a flowchart for this subroutine.

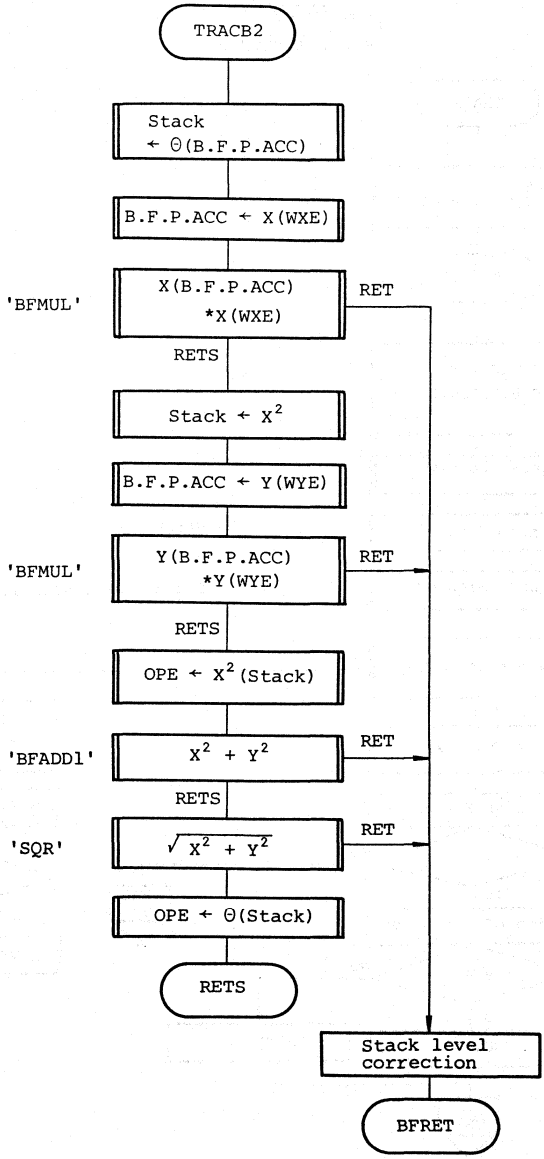
TRACB subroutine

Rectangular coordinates (X,Y) → Polar coordinates (γ,θ)

$$\begin{cases} \gamma = \sqrt{X^2 + Y^2} \\ \theta = \mathrm{ARCTAN}(Y/X) \end{cases}$$



TRACB subroutine



3-3 Description of Subroutines for Converting a Character Constant into a Floating-point Binary Number and Vice Versa

The following conversion subroutines are provided:

(1) BFINP (Character constant ——→ Floating-point binary number) subroutine

(2) BFOUT (Floating point binary number ——→ Character constant) subroutine

3-3-1 BFINP (Character constant ——→ Floating-point binary number) subroutine

1) Processing

Convert the character constant addressed by the contents of pair register HL into a floating-point binary number and store the result in the B.F.P.ACC.

2) Input condition

Set the starting address of the character constant in HL.

3) Output condition

RET : The value of the character constant is too large.
'1' is set in the CY flag.

RETS: The character constant is properly converted and the result has been stored in the B.F.P.ACC.
'0' is stored in the CY flag.
If the result of the conversion is 0, '1' is set in the Z flag.
Otherwise, '0' is stored in the Z flag.

4) Subroutines used

See Table 4-16.

5) Stack depth

Max. 5

6) Coding sequence

```
        )
        (
LXI     H,      ; Character constant starting address
CALL    BFINP; Conversion processing
GJMP    ERROR; Jump to overflow processing
        )
        (
```

7) Processing time

Max. About 388 ms. (Oscillation frequency: 11.0592 MHz)

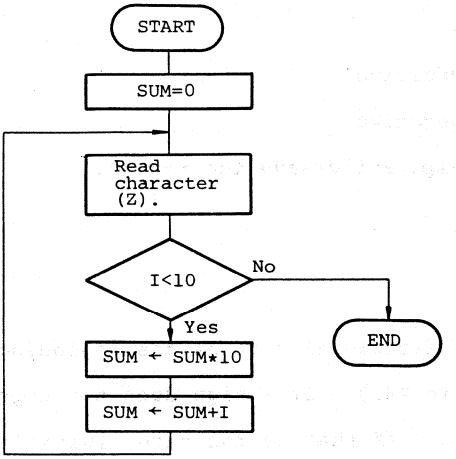
Measurement example:

2.9 ..... 9E - 38
   37

8) Algorithm and processing procedure

Algorithm

The flow during conversion of a decimal number into a binary number is as follows:



The operation of the fractional part is performed according to the above flow. The result can therefore be obtained simply by adjusting the position of the decimal point.
The meanings of the variables used in this subroutine are as follows:

ADDS	TMP1	TMP2	TMP3	VAE	VA1	VA2	VA3

ADDRS: Address in which the character constant is stored.

TMP1: Position of the decimal point in the character constant. With the determination of the decimal point flag, one is subtracted for each digit.

TMP2: Decimal point flag in the character constant

TMP2 = 0 No decimal point appears.

TMP2 ≠ 0 Decimal point appears.

TMP3: Sign flag

TMP3 = 80H Positive

TMP3 = 00h Negative

WXE, WX1, WX2, WX3: Temporarily save the result.

Processing procedure

a) Put the sign of the fractional part at the beginning of the character constant in TMP3. If a sign does not appear, the plus sign is implied. If that is the case, initialize the plus sign flag in TMP3.

b) Go to e) when there is an exponent sign (E). Set the decimal point flag (TMP2) when it is a character (.).

c) Multiply the B.F.P.ACC value by 10 and save the result in WXE through WX3.

d) Convert the character (0 to 9) into a B.F.P.ACC value. If the decimal point flag (TMP2) has been determined, decrease the decimal point position (TMP1) by one for each digit. For example, if the value is 12.3, calculate (12 x 10 + 3) and set 0FFH (-1) in TMP1.

e) Store WXE (previous result) + B.F.P.ACC value in the B.F.P.ACC.

f) Convert the exponent into a binary number.
If it is a negative value, express it in two's complement form.

g) Calculate the exponent obtained in f) + TMP1 and obtain the decimal point position.

h) If the result of g) is negative, divide the B.F.P.ACC value by 10 repeatedly until the result of g) becomes 0.
If the result obtained in g) is positive, multiply the B.F.P.ACC value repeatedly until the result of g) becomes 0.
An example is given below.

Example:

5 digits

$0.\widehat{12345}E4$

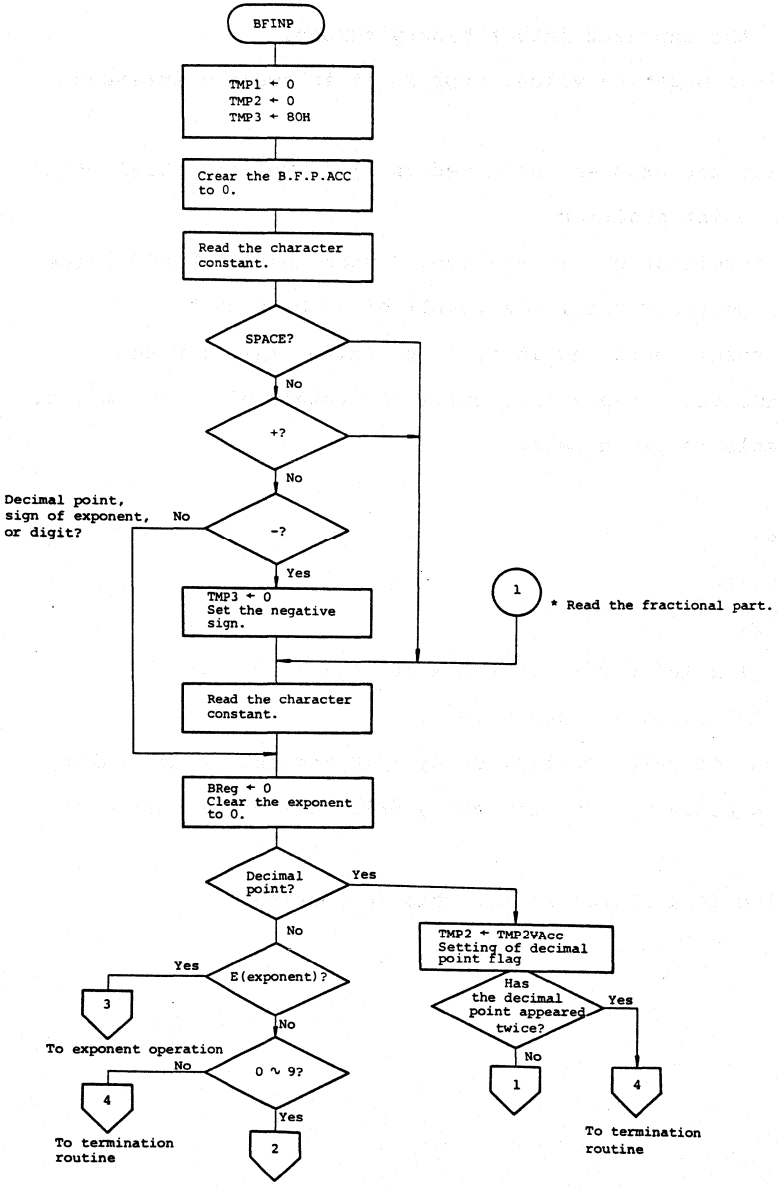
Obtain $(1 \times 10^4 + 2 \times 10^3 + 3 \times 10^2 + 4 \times 10 + 5)$.

Since $0.12345E4 = 12345 \times 10^{-1}$,

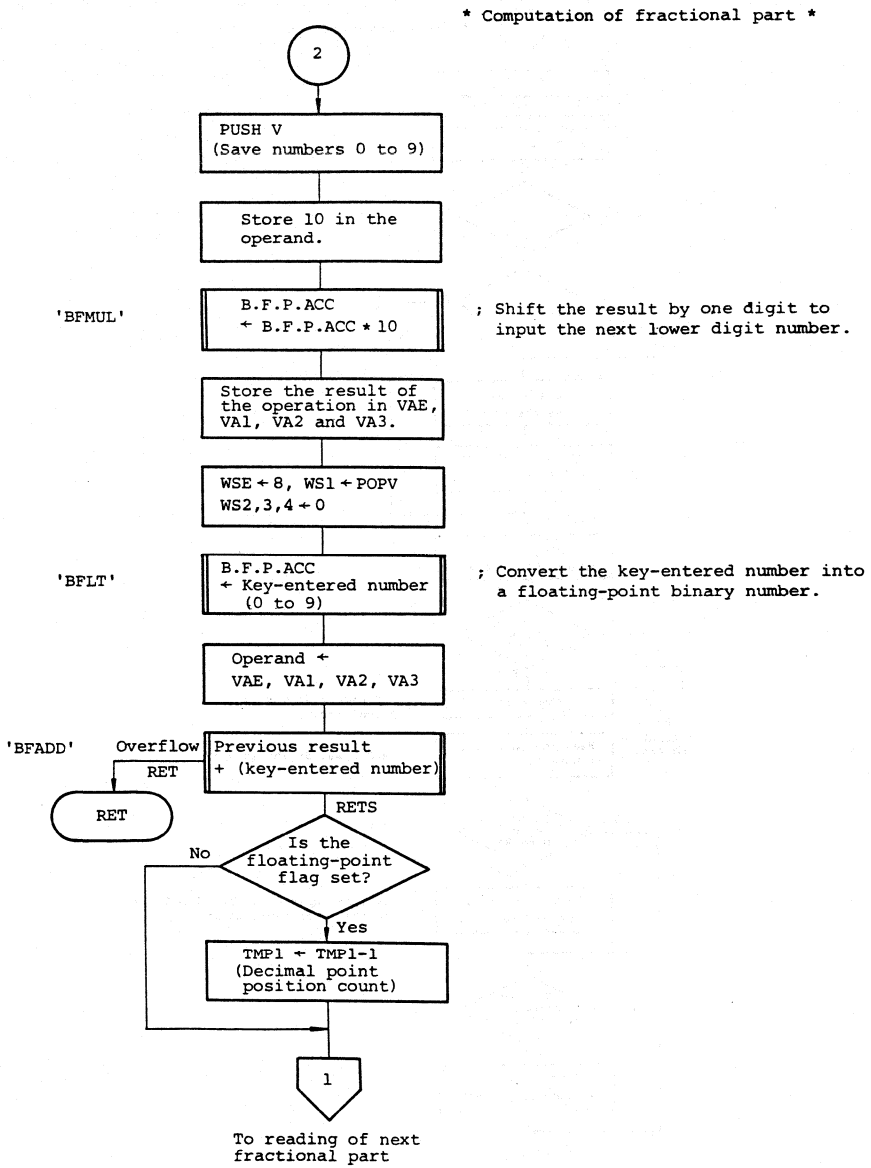
the obtained value multiplied by 1/10 remains as the value in the B.F.P.ACC. In this case, 0FFH (4 - 5) is stored in TMP1.

The following is a flowchart for this subroutine.

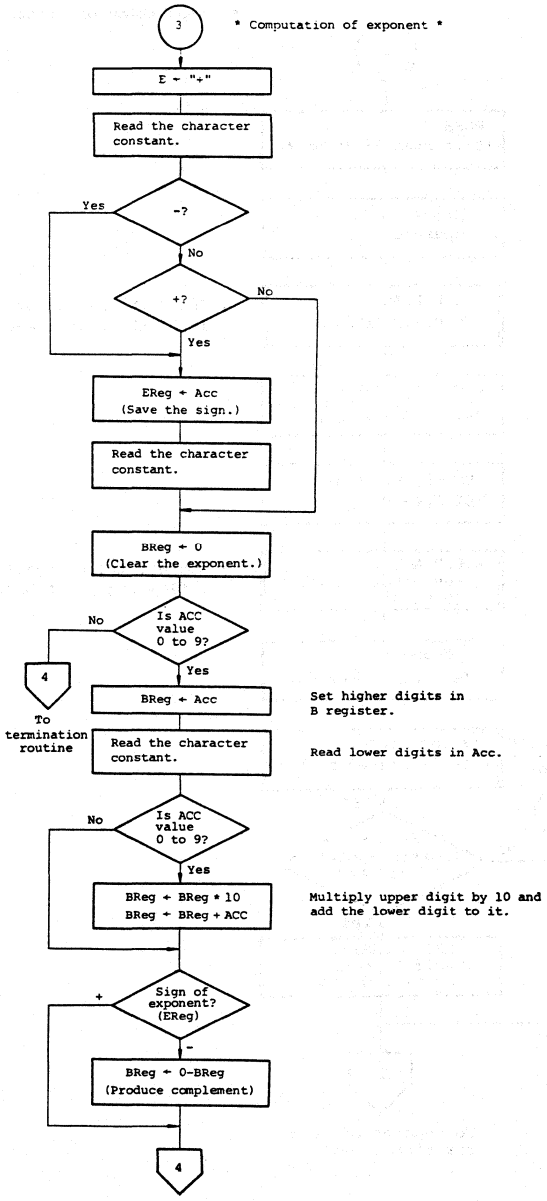
BFINP subroutine



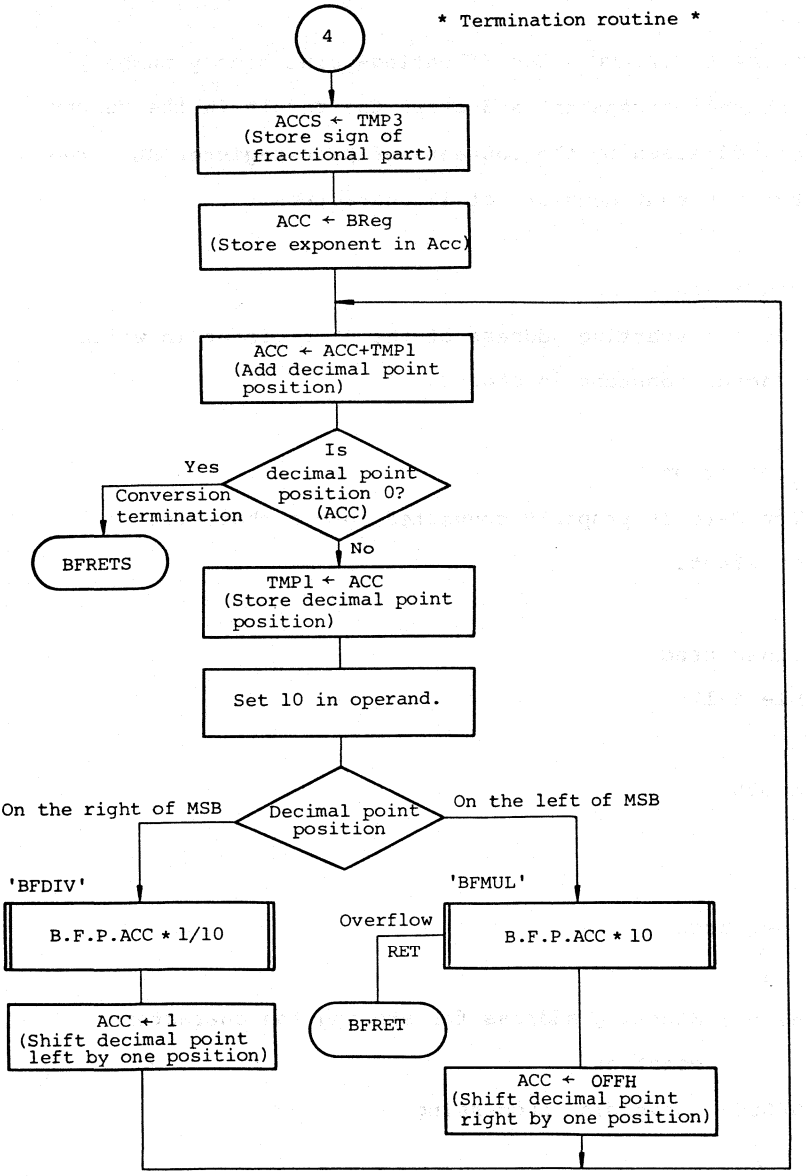
BFINP subroutine



BFINP subroutine



BFINP subroutine



3-3-2 BFOUT (Floating-point binary number ⟶ Character constant) subroutine

1) Processing

Convert the B.F.P.ACC value (floating-point binary number) into a character constant and store the result in the memory location addressed by the contents of pair register HL. The character constant consists of 13 characters.

2) Input condition

Determine the starting address of the HL register in which the character constant is stored.

3) Output condition

RET: The data is properly converted into a character constant.

4) Subroutines used

See Table 4-17.

5) Stack depth

Max. 6

6) Coding sequence

```
        )
        (
LXI   H, - ; Starting address for storing the character
             constant.
CALL  BFOUT; Conversion processing
        )
        (
```

7) Processing time

Max. About 160 ms. (Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC FFH, 80H, FFH, FFH, FFH

8) Algorithm and processing procedure

Algorithm

A floating-point binary number is converted into a decimal number in the following manner:

(1) Define the floating point such that it is in the range between 0.1 and 1 (exponent ACCE between 2^{-2} and 2^{1}).

Exponent (ACCE) $\geqq 2^{1}$: Multiply the value by 1/10.
Increase the decimal exponent by 1.

Exponent (ACCE) $< 2^{-1}$: Multiply the value by 10.
Decrease the decimal exponent by 1.

(2) Since the floating point obtained in (1) is within the range between 0.1 and 1, it is possible to obtain the first digit of the integer part of the decimal number by multiplying by 10.

The meanings of the variables used in this subroutine are as follows:

TMP1: The number of digits to the left of the decimal point is determined.

TMP2: The decimal exponent is expressed in binary notation. A negative number is expressed in the form of the complement.

When the floating point is fixed between 2^{-2} and 2^{1}, the exponent is increased or decreased by 1 depending on whether the value is multiplied by 1/10 or 10, respectively.

After this fixing of the floating point, the position of the decimal number is obtained.

If TMP2 is equal to or greater than 8, the decimal number than can be expressed contains seven characters (excluding the exponent part) as shown in Example 1 below. Since the range is exceeded, it is expressed by using an exponent.

Example 1: Expression when the decimal number exceeds seven characters.

98765432. (wrong) → 9.876543E7 (correct)

8 digits 1 6

If TMP2 is less than 8, the character constant contains a decimal number. Therefore, the decimal exponent is 0.

TMP3:

The number of digits to the right of the decimal point.

When 0FFH is determined, processing is terminated.

WXE, WX1, WX2, WX3:

The B.F.P.ACC value is saved in these areas.

Processing procedure

a) Save the B.F.P.ACC data in the four bytes beginning with WXE to prevent the data from being destroyed.

b) Set the B.F.P.ACC data between 0.1 and 1.
Store the decimal exponent in TMP2.

c) Determine TMP1, TMP2 and TMP3.
If TMP2 is equal to or greater than 8, the exponent (TMP2) is the current TMP2 value less 1, and 1 and 6 are set in TMP1 and TMP3, respectively.
If TMP2 is less than 8, the character constant contains a decimal point as shown in Example 2) below. Determine the position of the decimal point in the following manner: TMP3 ← 7 - TMP2 and TMP1 ← TMP2 - 1 and make the exponent 0 in this manner: TMP2 ← 0.

Example 2: Expression when the character constant contains a decimal point

3.141593 ... No exponent contained

-1234567

d) Multiply the B.F.P.ACC data by 10.

e) Since the integer part of the B.F.P.ACC data is one digit of the decimal number multiplied by d), specify the position of the decimal point (shift counter) as 8, make the B.F.P.ACC data a fixed-point number so that only the integer part remains, and use the remaining A register as one digit of the decimal number.

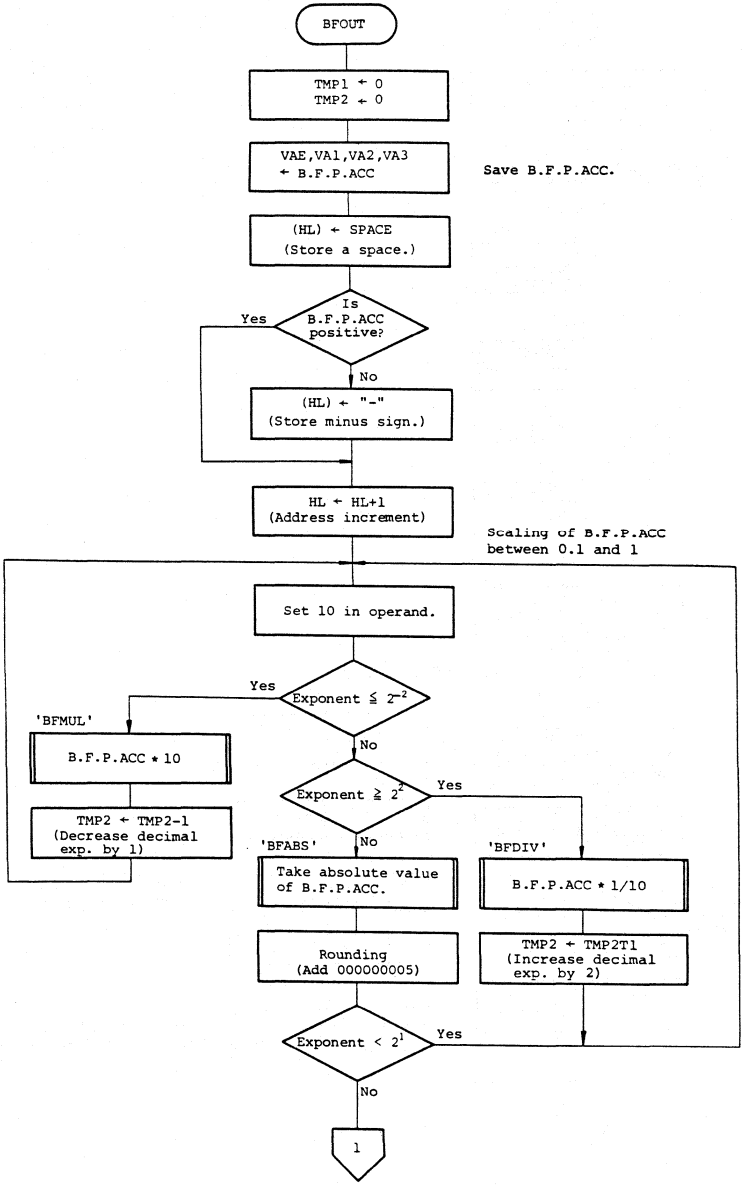
f) Store the decimal number in the designated address and make it a floating-point number.

Repeat d) to f) for all seven digits.

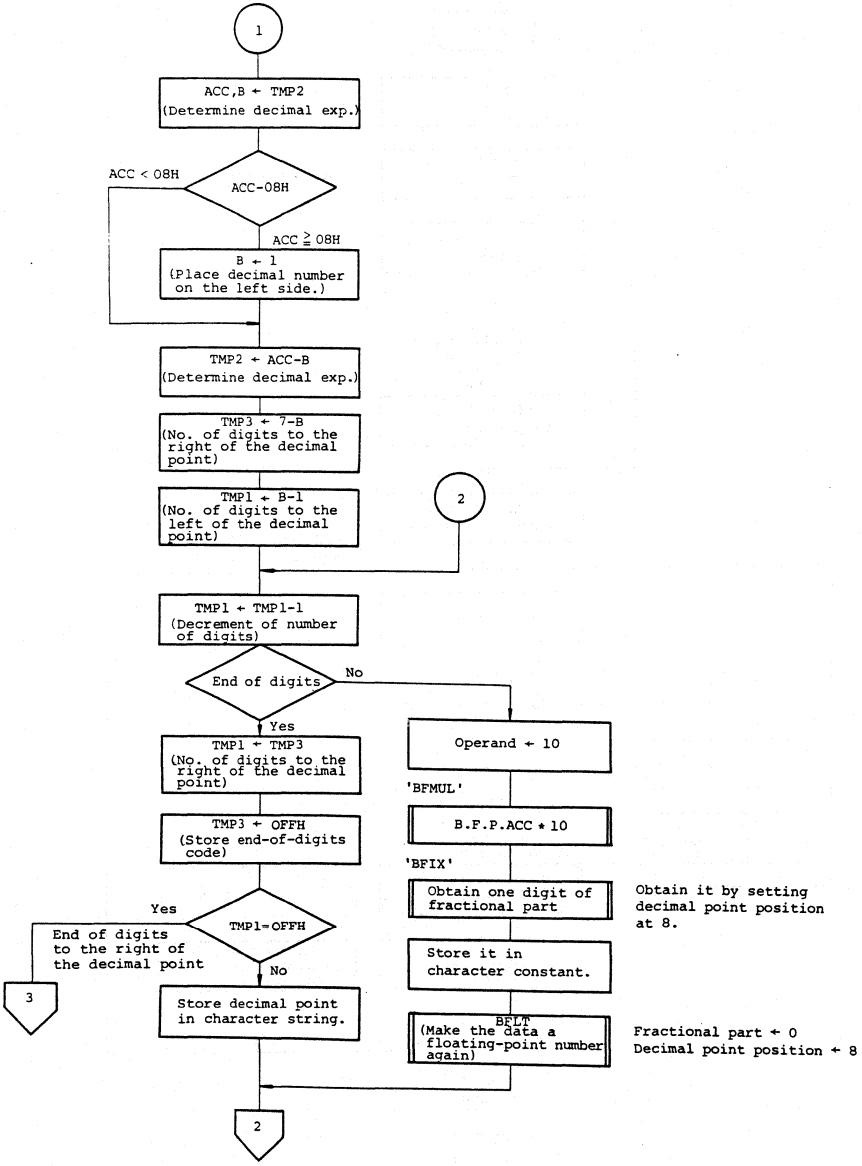
g) Make the exponent value (TMP2) a decimal number using the exponent (E). IF the value is negative, express it with the minus (-) sign.

The following is a flowchart for this subroutine.

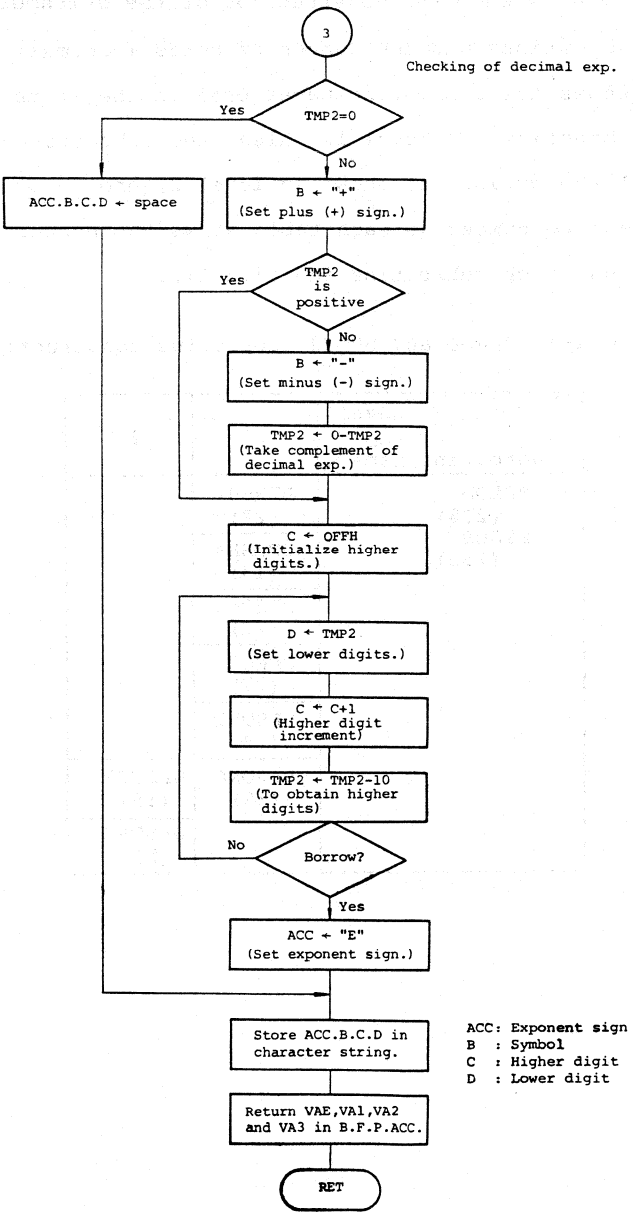
BFOUT subroutine



BFOUT subroutine



BFOUT subroutine



4. Construction of Subroutines for Arithmetic Operations

Tables 4-1 to 4-17 show the construction of the arithmetic operation subroutines and the number of bytes (decimal) used. Each table shows the subroutine names used in the arithmetic operation subroutines in level 1. Also, the subroutines used in each level 1 subroutine are shown in level 2, and so on. The parenthesized number in each table shows the number of bytes required to use each subroutine individually.

Table 4-1 BFADD and BFSUB subroutine construction

Level / Subroutine name	1	2
BFADD (278) BFSUB (285)	BFRSH (27)	
	BFRSH4 (13)	
	BFCOM1 (21)	
	BFNOR (27)	
	BFSTR1 (14)	
	BFROND (39)	BFRNDR (15)
		BFSTR2 (9)

Table 4-2 BFMUL subroutine construction

Subroutine name \ Level	1	2	3	4
BFMUL (266)	BFRD (48)			
	BFMX (99)	BFMX3 (66)	MULT (29)	MULT2 (12)
			BFRSH0 (27)	
	BFNOR (27)			
	BFROND (39)	BFRNDR (15)		
		BFSTR2 (9)		

Table 4-3 BFMUL subroutine construction

Subroutine name \ Level	1	2
BFDIV (281)	BFRD (48)	
	BFDX (136)	BFRSH4 (13)
		BFLSH1 (9)
	BFROND (39)	BFRNDR (15)
		BFSTR2 (9)

Table 4-4 SIN subroutine construction

Level / Subroutine name	1	2	3	
SIN (1047)	BFSIGN (15)			
	BFADD (263)	Same as BFADD subroutine		
	BFSUB (270)	Same as BFSUB subroutine		
	BFDIV (266)	Same as BFDIV subroutine		
	BFCOMP (6)			
	STORE0 (21)			
	PUSHF (39)			
	POPR (12)			
	BFINT (130)	BFLT (14)		
		BFIX2 (108)	BFCOM1 (21)	
			BFRSH (27)	
	BFCOMX (10)	BFCOMP (6)		
	POLYN1 (614)			

Table 4-5 COS subroutine construction

Level / Subroutine name	1	
COS (1076)	BFABS (4)	
	STORE1 (21)	
	BFADD (263)	Same as BFADD subroutine
	LOADA1 (25)	
	BFSUB (270)	Same as BFSUB subroutine
	BFCOMP (6)	
	SIN (1047)	Same as SIN subroutine

Table 4-6 TAN subroutine construction

Level / Subroutine name	1	
TAN (1103)	PUSHF (39)	
	POPR (12)	
	STORE1 (21)	
	LOADA0 (27)	
	LOADO1 (14)	
	SIN (1047)	Same as SIN subroutine
	COS (1076)	Same as COS subroutine
	BFDIV (266)	Same as BFDIV subroutine

Table 4-7 LNX subroutine construction

Level / Subroutine name	1	
LNX (880)	BFSIGN (15)	
	BFADD (263)	Same as BFADD subroutine
	BFSUB (270)	Same as BFSUB subroutine
	BFMUL (251)	Same as BFMUL subroutine
	BFDIV (266)	Same as BFDIV subroutine
	BFCOMP (6)	
	BFLT (14)	
	POLYN3 (612)	
	STORE0 (21)	
	LOADA (23)	
	PUSHF (39)	

Table 4-8 LOG subroutine construction

Level / Subroutine name	1	
LOG (889)	LNX (880)	Same as LNX subroutine
	BFDIV (266)	Same as BFDIV subroutine

Table 4-9 EXP subroutine construction

Level / Subroutine name	1	2	3
EXP (927)	BFSIGN (15)		
	BFABS (4)		
	PUSHF (39)		
	POPR (12)		
	STORE0 (21)		
	LOADA (23)		
	BFSUB (270)	Same as BFSUB subroutine	
	BFMUL (251)	Same as BFMUL subroutine	
	BFDIV (266)	Same as BFDIV subroutine	
	BFINT0 (429)	BFSIGN (15)	
		BFSUB (270)	
		BFLT (14)	
		BFIX2 (108)	BFCOM1 (21)
			BFRSH (27)
	POLY (566)		

Table 4-10 ARCTAN subroutine construction

Subroutine name \ Level	1	2	
ARCTAN (954)	STORE0 (21)		
	LOADA (23)		
	LOADO3 (14)		
	BFSIGN (15)		
	BFCOMP (6)		
	BFDIV (266)	Same as BFDIV subroutine	
	POLYN2 (608)		
	BFSUB (270)	Same as BFSUB subroutine	
	BFCOMX (10)	BFCOMP (6)	

Table 4-11 ARCSIN subroutine construction

Subroutine name \ Level	1	2	
ARCSIN (1747)	PUSHF (39)		
	POPR (12)		
	STORE1 (21)		
	LOADA0 (27)		
	LOADA (27)		
	LOAD01 (14)		
	BFSIGN (15)		
	ROUTE (1620)	BFMUL (251)	Same as BFMUL subroutine
		BFSUB (270)	Same as BFSUB subroutine
		SQR (1075)	Same as SQR subroutine
		BFCOMP (6)	
	BFDIV (266)	Same as BFDIV subroutine	
	BFCOMP (6)		
	ARCTAN (954)	Same as ARCTAN subroutine	

Table 4-12 ARCCOS subroutine construction

Level / Subroutine name	1	2	
ARCCOS (1762)	BFSIGN (15)		
	PUSHF (39)		
	ROUTE (1620)	BFMUL (251)	Same as BFMUL subroutine
		BFSUB (270)	Same as BFSUB subroutine
		SQR (1075)	Same as SQR subroutine
		BFCOMP (6)	
	POPR (12)		
	BFDIV (266)	Same as BFDIV subroutine	
	ARCTAN (954)	Same as ARCTAN subroutine	
	BFADD (263)	Same as BFADD subroutine	
	LOADA (27)		
	BFMUL (251)	Same as BFMUL subroutine	
	BFSUB (270)	Same as BFSUB subroutine	
	BFCOMP (6)		
	SQR (1075)	Same as SQR subroutine	

Table 4-13 SQR, POWER subroutine construction

Subroutine name \ Level	1	2	3	
SQR POWER (1075)	PUSHR (12)			
	POPR (21)			
	STORE0 (21)			
	STORE1 (21)			
	LOADA0 (24)			
	LOAD01 (11)			
	BFABS (4)			
	BFSIGN (15)			
	BFCOMP (6)			
	BFINT (130)	BFLT (14)		
		BFIX2 (108)	BFCOM1 (21)	
			BFRSH (27)	
	BFSUB (270)	Same as BFSUB subroutine		
	LNX (880)	Same as LNX subroutine		
	EXP (927)	Same as EXP subroutine		

Table 4-14 TRACA subroutine construction

Level / Subroutine name	1	
TRACA (1120)	PUSHF (39)	
	POPR (12)	
	LOADA0 (27)	
	STORE1 (21)	
	STORE2 (21)	
	SIN (1047)	Same as SIN subroutine
	COS (1076)	Same as COS subroutine
	BFMUL (251)	Same as BFMUL subroutine

Table 4-15 TRACB subroutine construction

Subroutine name \ Level	1	
TRACB (1186)	STORE1 (21)	
	STORE2 (21)	
	LOADA0 (27)	
	LOADA1 (27)	
	LOADA2 (27)	
	LOADA (21)	
	LOADO1 (14)	
	PUSHF (39)	
	POPR (12)	
	BFSIGN (15)	
	BFCOMP (6)	
	BFDIV (266)	Same as BFDIV subroutine
	BFMUL (251)	Same as BFMUL subroutine
	BFADD (263)	Same as BFADD subroutine
	ARCTAN (954)	Same as ARCTAN subroutine
	SQR (1075)	Same as SQR subroutine

Table 4-16 BFINP subroutine construction

Level / Subroutine name	1	
BFINP (848)	STORE1 (21)	
	CHARD (9)	
	ADADJ (9)	
	BFADD (263)	Same as BFADD subroutine
	BFMUL (251)	Same as BFMUL subroutine
	BFDIV (266)	Same as BFDIV subroutine
	BFLT (14)	

Table 4-17 BFOUT subroutine construction

Subroutine name \ Level	1	2
BFOUT (940)	STORE1 (21)	
	LOADA0 (27)	
	LOADA1 (27)	
	PUSHF (39)	
	POPR (12)	
	BFADD (263)	Same as BFADD subroutine
	BFSUB (270)	Same as BFSUB subroutine
	BFMUL (251)	Same as BFMUL subroutine
	BFDIV (266)	Same as BFDIV subroutine
	BFABS (4)	
	BFLT (14)	
	BFOX0 (108)	BFCOM1 (21)
		BFRSH (27)

5. Common Subroutine

The following are the common subroutines used in the arithmetic operation subroutines. The parenthesized number under each subroutine name shows the number of bytes used by the subroutine.

Table 5.1 Common subroutines

Subroutine name	Processing
POLYN1 POLYN2 POLYN3 (612)	Assign the square of the operand to polynomial (5.2) and multiply the result by the original operand. (Refer to expression 5.1) Store the result in the B.F.P.ACC. The HL register stores the address of the table that stores the number of terms of the polynomial and the coefficient of each term; Polynomial = $X (C_0 + C_1X^2 + C_2X^4 + C_3X^6 + \ldots)$ (5.1)
POLY (566)	Execute polynomial (5.2) and store the result in the B.F.P.ACC. The HL register stores the address of the table that contains the number of terms of the polynomial and the coefficient of each term. Polynomal = $C_0 + C_1X + C_2X^2 + \ldots$ (5.2)
BFNOR (27)	Normalize the fractional part of the operand (WS1, WS2, WS3, WS4). At this time, adjust the exponent (ACCE) to shift the fractional part to the left.
BFROND (39)	Round the fractional part (WS1, WS2, WS3, WS4) of the operand with respect to the MSB of WS4. Send the result (WS1, WS2, WS3) to the fractional part (ACC1, ACC2, ACC3) of the B.F.P.ACC.
BFCOM1 (21)	Express the fractional part (WS1, WS2, WS3, WS4) of the operand in two's complement and invert the sign (ACCS) of the B.F.P.ACC value.
BFCOMP (6)	Invert the sign (ACCS) of the B.F.P.ACC value.
BFSIGN (15)	Check the sign of the B.F.P.ACC value and store the information in the accumulator and FSN as follows: FSN, accumulator = 0 ; B.F.P.ACC = 0 1 ; B.F.P.ACC = positive 0FFH; B.F.P.ACC = negative

Subroutine name	Processing
BFSTR1 (14)	Send the operand to the five-byte area indicated by the HL register. (HL, HL+1, HL+2, HL+3, HL+4) ← Operand
BFSTR2 (9)	Send the fractional part (WS1, WS2, WS3) of the operand to the three-byte area indicated by the HL register. (HL, HL+1, HL+2) ← Operand (WS1, WS2, WS3)
BFABS (4)	Take the absolute value of the B.F.P.ACC value.
BFRSH0 BFRSH (27)	Shift the fractional part (WS1, WS2, WS3) to the right using the accumulator as a shift counter and store it in (WS1, WS2, WS3, WS4). 'BFRSH0' stores 8 in the accumulator.
BFRSH2 BFRSH4 (13)	Shift the fractional part (WS1, WS2, WS3) including the CY flag to the right by one bit and store the result in (WS1, WS2, WS3, WS4).
BFLSH1 (9)	Establish a counter in the B register and make a one-bit shift to the right, byte by byte including the CY flag to the higher digit from the area shown by the HL register.
BFIX0 BFIX2 (108)	Convert a floating-point binary number to a fixed-point binary number. Store the position of the fixed point in the accumulator. Since the fractional part consists of 32 bits, it is possible to handle up to 32 positions. An example of this conversion is given below: Example: Suppose the B.F.P.ACC contains the data: 80H, 80H, 82H, 00H, 00H; 0.5078125 If the fixed point position (accumulator) is 8, the point moves to the right by eight bits as shown below: 0.100000100 ⟶ 0; 0.5078125 8 16 It is, therefore, necessary to shift the fractional part to the right by eight bits. If the B.F.P.ACC value is negative, express it in two's complement. If it is 0, clear (WS 1 to 4).

Subroutine name	Processing
BFLT (14)	Convert a fixed-point number (WSE, WS1 to 4) into a floating-point number (ACC, ACCS, ACC1 to 3). At this time, determine the exponent (ACCE) by adding the bias 80H to WSE, make the sign (ACCS) positive, and normalize and round the fractional part (ACC1 to 3).
BFINT0 (429) BFINT (130)	Obtain the integer part of the B.F.P.ACC value and store the result in the B.F.P.ACC. This subroutine is explained below: a) If the B.F.P.ACC value is negative, subtract 1 from it. This is because the integer part of X is I when $I \leqq X < I + 1$. (I: Integer) b) Make the fixed-point position 32 and put it on the right of the fractional part. Convert the floating-point binary number (ACCE, ACCS, ACC1 to 3) into a fixed-point binary number and leave the integer alone as the fractional part. c) Convert the fixed-point binary number into a B.F.P.ACC value.
BFRD (48)	In multiplication/division between the P.F.P.ACC and operand values, perform an operation (addition) on the exponents, store the sign obtained from their respective signs, and make the fractional part of the operand unsigned.
STORE0 STORE1 STORE2 (21)	Send the B.F.P.ACC value to the four-byte memory area indicated by the HL register.
LOADA0 LOADA1 LOADA2 (27) LOADA (21)	Send the contents of the four-byte area indicated by the HL register to the B.F.P.ACC.
LOADO1 LOADO2 LOADO3 (13)	Send the contents of the four-byte memory area indicated by the HL register to the operand.
PUSHF (39)	Store the B.F.P.ACC in the stack after sending it to the operand.
PUSHR (12)	Store the operand in the stack.
POPR (12)	Return the four-byte data in the stack to the operand.

CHAPTER II APPLICATION

This chapter gives a program that uses the COM-87AD Function Package described in Chapter I to connect the COM87AD Serial Interface and the Console. The program accepts input "expressions" from the console and displays the result on the console.

1. System Configuration

Figure 2-1 shows the system configuration to connect the μCOM-87AD serial interface to the console in asynchronous mode for data transmission.

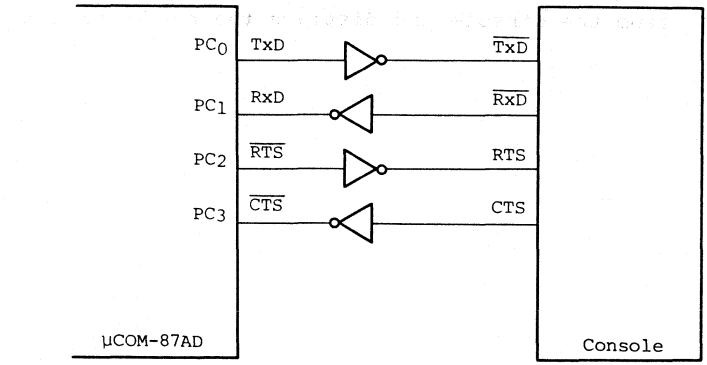


Fig. 2-1 System configuration

The interface conditions are as shown below:

(1)	Start bit	1 bit
(2)	Data length	8 bits
(3)	Parity	None
(4)	Stop bit	2 bits
(5)	Baud rate	4800 bps

Of the four lines necessary to perform serial data transmission, two are input/output lines TxD and RxD and the remaining two are control lines RST and $\overline{CTS}$. $\overline{RTS}$ is the data request signal from the μCOM-87AD. It is always active (low level) in this system.

When sending data, the μCOM-87AD checks the CTS signal from the console.

2. Evaluation of Expressions

This program treats a character string input from the console as an expression. The concept of this expression is explained in the following section.

The following are definitions of the terms used in this section:

(1) factor : number, function, expression in parentheses

(2) term : <factors> after multiplication/division

(3) expression : <terms> after addition/subtraction

Since the expression is in the hierarchic structure as shown in Fig. 2-2, the term and the factor are called in the order when evaluating the expression.

Fig. 2-2 Hierarchical structure of expression

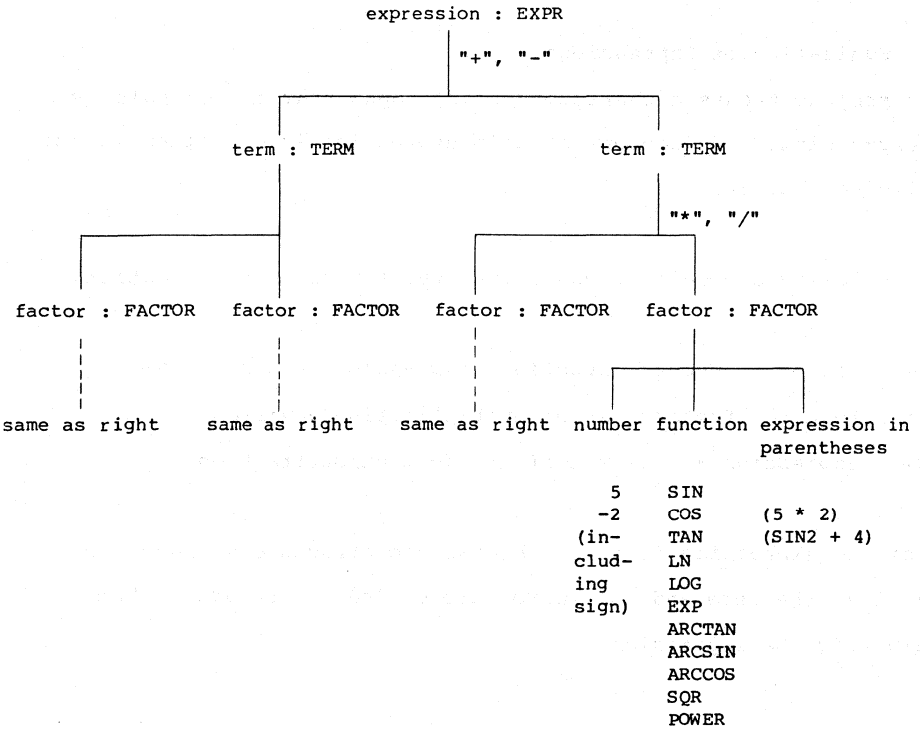


Figure 2-3 gives the flow of the evaluation of an expression.

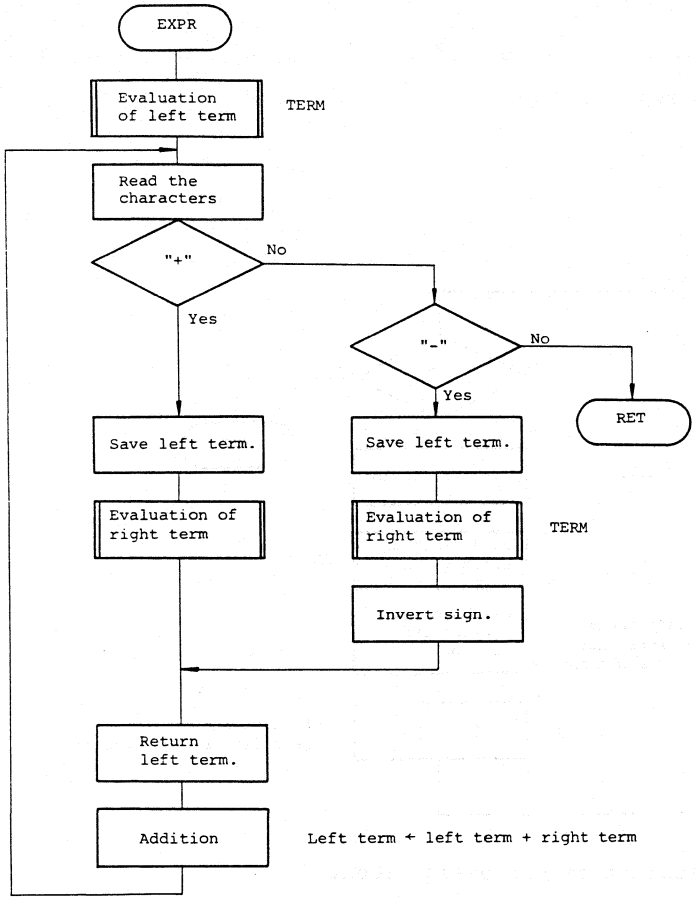


Fig. 2-3 Evaluation of expression

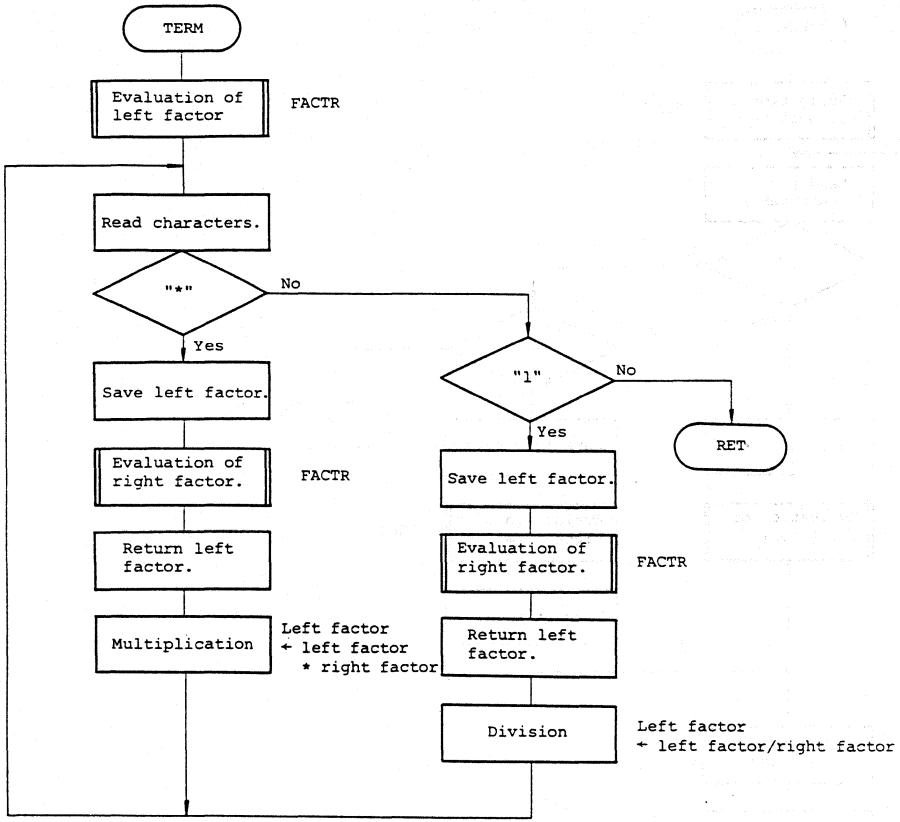


Fig. 2-3 Evaluation of expression (Cont'd)

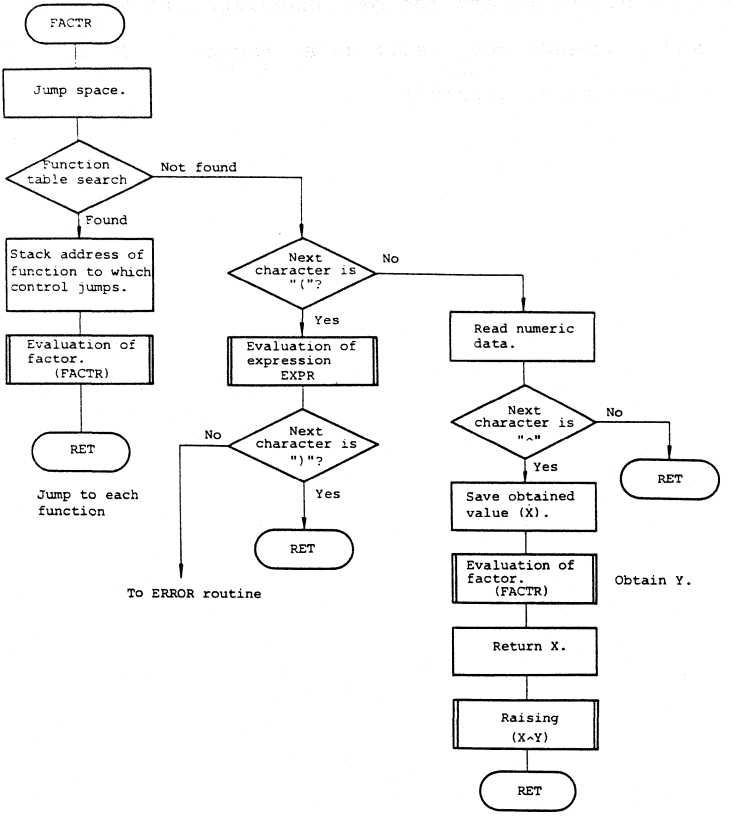
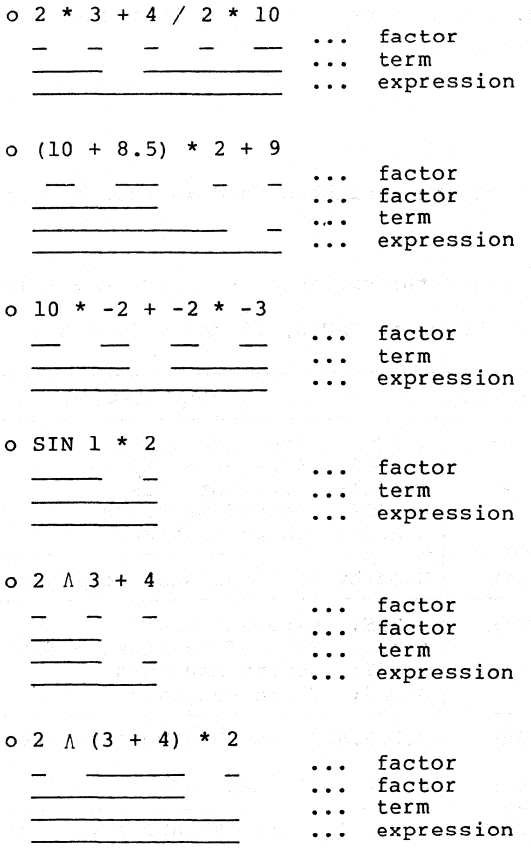


Fig. 2-3 Evaluation of expression (Cont'd)

The POWER function performs the raising operation between factors and treats the result as a factor again. If there are factors in parentheses, the subroutine treats the parenthesized factors as an expression and processes the result as a factor . This makes the nesting of parentheses possible.

The following are examples of how expressions are evaluated:

Examples:



The method of evaluating an expression explained above does not require a large capacity of program. But the processing speed is relatively low due to frequent use of a stack. Moreover, the stack nesting level increases each time parentheses are used. It is, therefore, necessary to be careful so that the stack range is not exceeded.

3. Variables

This section explains the variables used in this program.

This program determines the variable and stack areas in the internal RAM areas (FF00H to FFFFH) of the μCOM-87AD.

Variable name	Address	Explanation
B.F.P.ACC	FF00H to FF04H	o ACCE: Exponent part o ACCS: Sign of mantissa o ACC1: First mantissa o ACC2: Second mantissa o ACC3: Third mantissa
SF	FF05H	Used by Addition/subtraction subroutine
OPERAND	FF06H to FF0AH	o WSE: Exponent part o WS1: Sign of mantissa + first mantissa o WS2: Second mantissa o WS3: Third mantissa
WORK X REGISTER	FF0BH to FF0EH	B.F.P.ACC and OPERAND save area o WXE, WX1, WX2, WX3
WORK Y REGISTER	FF0FH to FF12H	B.F.P.ACC and OPERAND save area o WYE, WY1, WY2, WY3
WORK D REGISTER	FF13H to FF16H	BFDX subroutine operation area o WD1, WD2, WD3, WD4, WD5, WD6 Quotient Remainder

Variable name	Address	Explanation
WORK A REGISTER	FF19H to FF1BH	BFMX subroutine operation area o WA1, WA2, WA3
WORK REGISTER	FF1CH	o POWERK: Used by POWER subroutine Store least significant digit of INT(Y)
	FF1DH	o LNWORK: Used by LNX subroutine Store exponent of X - 80H
	FF1EH	o CMULT0: Used by BFMX subroutine Digit counter (three digits)
	FF1FH	o CMULT1: Used by MULT subroutine Byte counter (three bytes)
	FF20H	o CSIN: Used by SIN subroutine Work area for scaling
	FF21H	o TMPWSL: Temporary WS4 save area
	FF22H	o FASIN: Used by ARCSIN subroutine Store sign of X (Used at the time of overflow)
FLAG OF SIGN	FF23H	o FSN: Store the sign of BFPACC. 0: BFPACC = 0 1: BFPACC > 0 0FFH: BFPACC < 0
BFINP & BFOUT WORK	FF24H FF25H	o ADDRS: Character storing address
	FF26H	o TMP1: Temporary 1
	FF27H	o TMP2: Temporary 2
	FF28H	o TMP3: Temporary 3
	FF29H	o TMP4: Temporary 4
OUT AREA	FF2AH	o OUTADR: Output character string storing area
CHARACTER STORE AREA	FF37H to FF87H	o CHARST: Input character storing area (Max. 80 characters)
STACK AREA	FF88H to FFFFH	o Stack area

4. Explanation of Program

This program reads characters from the console through the serial interface of the μCOM-87AD into the character storing area (80 bytes from CHARST) until "=" is input. The characters are evaluated as an expression as explained in 2-2, and the result is output to the console.

Since, however, the coordinate conversion subroutines (TRACA and TRACB subroutines) each involve output of two arithmetic operation values, it is necessary to call the evaluation of the expression again after evaluating the expression. This calls the coordinate conversion subroutine using the values specified in the expression.

The evaluation of an expression starts with the current character string pointer and ends when an illegal character is encountered. After this, the character string obtained by the evaluation of the expression is output to the console. Next, the program examines whether the next character is "=" or not. If it is not "=", the program outputs "?" indicating a format error. Even in the case of a format error, however, the value of the expression evaluated up to the point at which the error occurs is output.

Figure 2-4 shows the general flow of this program.

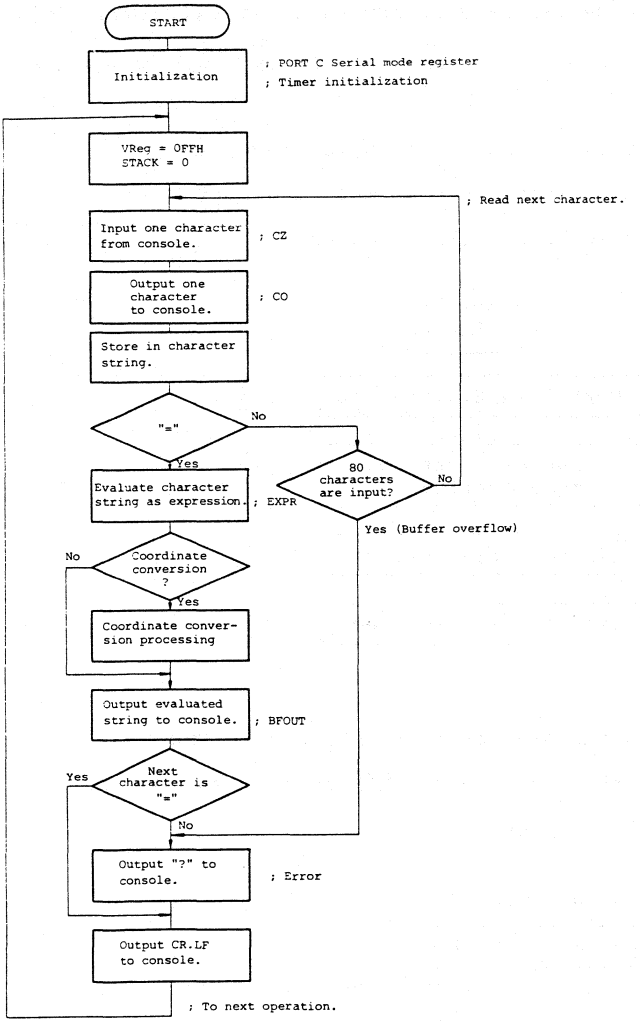
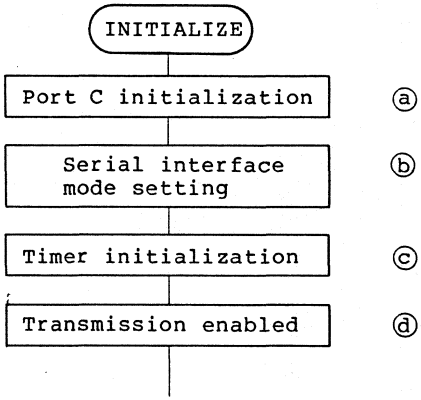


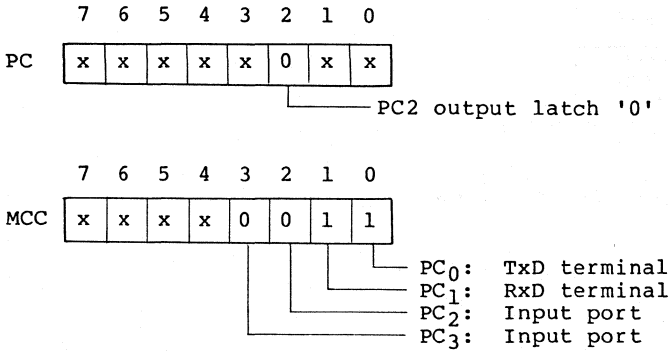
Fig. 2-4 General flow

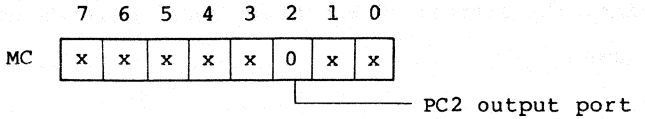
The method of μCOM-87AD serial data transmission is explained below.

To generate μCOM-87AD serial data transmission requires initializing the serial mode register, timer, and port C in advance as shown by the following flow diagram.

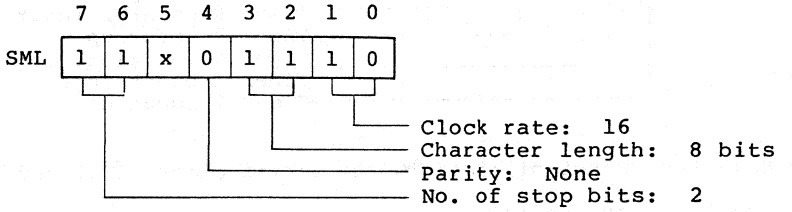


ⓐ Port C has PC_0 as the TxD terminal, PC_1 as the RxD terminal, PC_2 as the output port ($\overline{RTX}$ output) and PC_3 as the input port ($\overline{CTS}$ input).





ⓑ Set the parameters for serial data transmission in the serial mode register.



ⓒ Since the TO output is used as a serial clock ($\overline{SCK}$), the timer value C is given by the following formula:

$$C = \frac{f}{0.024 \times N \times B}$$

f: Clock oscillation frequency (MHz)

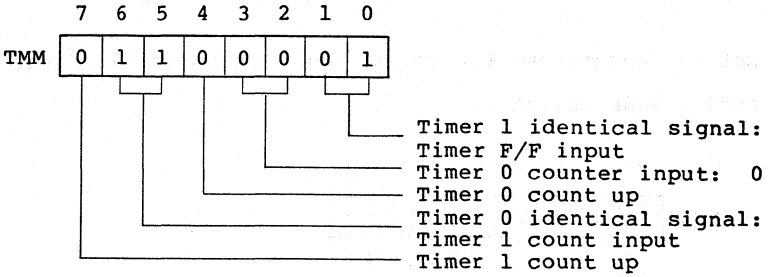
N: Clock rate

B: Data transfer rate (kbpx)

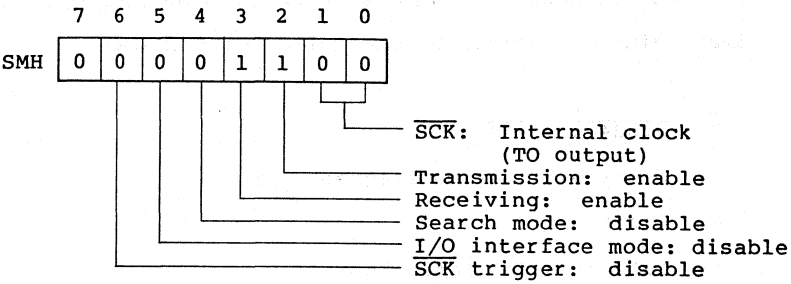
In this system, f is 11.0592 MHz with N = 16 and B = 48. If Timer 0 and Timer 1 are used in a cascade connection, these may be set as follows:

TM0 = 6, and TM1 = 1

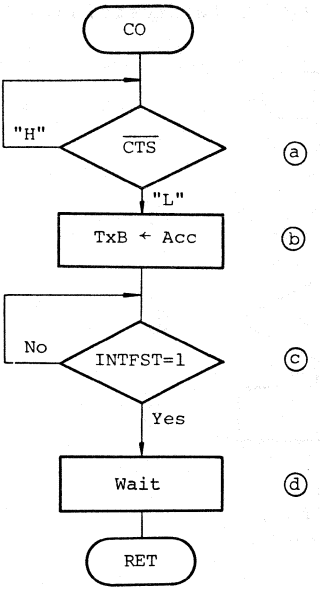
At this time, the setting of the timer mode register (TMM) is as follows:



ⓓ Make the output of timer TO the serial clock ($\overline{SCK}$) and enable transmission.



The following is the subroutine that transmits one byte of the content of the accumulator (ACC).



ⓐ Loop until the receiving side transfer request signal ($\overline{CTS}$) is received.

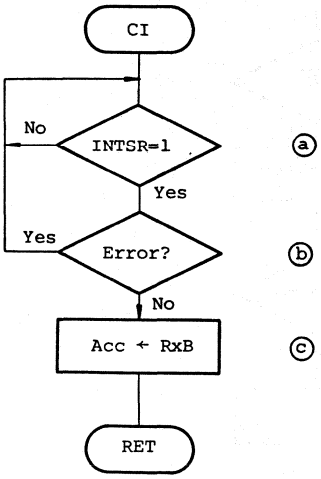
ⓑ Transfer the contents of ACC to the transmission buffer.

ⓒ Wait until the transmission buffer becomes empty.

ⓓ Wait until the serial register becomes empty.

When the CO subroutine is called during operations ⓒ and ⓓ, both the transmission buffer and serial register are always empty. In this state, it is possible to check the $\overline{CTS}$ signal, and the data transmitted is not cancelled by the receiving side.

The following is the follow of data reception.



ⓐ Loop until INTSR becomes equal to 1.

ⓑ Check the data received for any errors. If it contains an error, return to ⓐ.

ⓒ Transfer the data received to the ACC.

CHAPTER III RESULT OF EVALUATION

The COM-87AD function package has been evaluated by the program described in II.

The functions other than addition, subtraction, multiplication and division were compared with the equivalent functions of the PC-8001. It was found that for all these functions, the precision in terms of the number of significant figures was 5 to 6.

It can be said that the desired level of performance has been achieved.

The following are the results of evaluation for each function.

(1) BFADD subroutine

```
1+0= 1.000000                    ; Check for 0
0+1= 1.000000


1E38+1= 1.000000E+38             ; Neighbourhood of MAX
1E38+9E38=?
1E38+1E38=?
0.9E38+0.01E38= 9.100004E+37


9.9E-39+0.1E-39= 9.900000E-39 ; Neighbourhood of MIN
9.9E-39+0.2E-39= 9.900000E-39
9.9E-39+1E-39= 9.900000E-39
9.9E-39+1E39=?
9.9E-39+1E30= 1.000000E+30


1345+98765= 100110.0             ; Random
455675211368643+214447796= 4.556755E+14
1.3E20+4E-30= 1.300000E+20
223+0.11111111= 223.1111
5465678627R45TFG

134465463+478487687685= 0.000000    ?
3246754564+233= 3.246755E+09

1E-30+1E-10= 1.000000E-10      ;
122344.E-10+345667E-9= 3.579014E-04
1232343.6787654E-27+0= 1.232344E-21
98765E-25+111111111E-3= 111111.1
```

(2) BFSUB subroutine

```
1-0= 1.000000                    ; Check for 0
0-1=-1.000000


2133E-29-12.45E-10=-1.245000E-09
 Neighborhood of MAX.
23352E-9-0.999999E-20= 2.335200E-05
3763467E-30-12E-1=-1.200000


1E-39-8E-39=-8.000004E-39        ; Neighbourhood of MIN.
9.9E-39-0.1E-19=-1.000000E-20
9.9E-39-0.1E-39= 9.900000E-39
9.9E-39-1.1E-39= 9.900000E-39
9.9E-39-2=-2.000000
9.9E-39-2.2E-38= 0.000000    ?
9.9E-39-2.2E-36=-2.190100E-36



2.2E-38= 2.200000E-38            ; Random
9.9E-38-2.2E-38= 7.700001E-38
2.2E-39= 0.000000


12356565-45876= 1.231069E+07
123456789-123456789= 0.000000
125654-988656=-863002.2
235655656-787788778=-5.521332E+08
1E20-23454.3344E10= 9.999979E+19
```

(3) BFMUL subroutine

```
1*0= 0.000000                          ; Check for 0
0*1= 0.000000

9.9E37*0.1= 9.900001E+36               ; Neighbourhood of MAX
1E38*0.1= 1.000000E+37
1E38*0.99999999=?
1E38*0.9= 9.000002E+37
12E37*2=?

1E-38*2= 2.000001E-38                  ; Neighbourhood of MIN
9.9E-39*2= 1.980000E-38
9.9E-39*1E-1= 0.000000
9.9E-39*1E2= 9.900000E-37

6543665534*12344= 8.077501E+13         ; Random
123456789*123456789= 1.524158E+16
1E30*1E-39= 0.000000
1E30*1E-10= 1.000000E+20
1.234E20*234= 2.887560E+22

1E-20*100= 1.000000E-18
1.234E-30*1.2E-5= 1.480801E-35
12344E-35*99= 1.222056E-29
1.23456R= 1.234560    ?
1.2345E-30*2E-7= 2.469000E-37
```

(4) BFDIV subroutine

```
0/1= 0.000000                          ; Check for 0
1/0=?

1E38/2= 5.000001E+37                   ; Neighbourhood of MAX
1E38/0.1=?
1E38/1E-10=?
1E38/1E10= 1.000000E+28

1E-38/1E-10= 1.000000E-28              ; Nighbourhood of MIN
1E38-= 1.000000E+38
1E-38/2= 0.000000
9.9E-39/9.9E-2= 1.000000E-37

13346546534/4675765= 2854.410          ; Random
123456789/123456789= 1.000000
987654321/123456789= 8.000000
12/21= .5714286
1E10/2.3E20= 4.347826E-11
1.23E10/-1.2E10=-1.025000

1.23E-20/.99E-10= 1.242425E-10
456E-10/32432E-9= 1.406019E-03
122212E-10/123E-2= 9.935935E-06
```

(5) COS subroutine

μ COM-87AD	PC-8001
COS0= 1.000000	cos (0)= 1
COS3.141592653=-1.000000	cos (3.14159)=-1
COS1.570796326= 0.000000	cos (1.5708)= 0
COS500=-.8838571	cos (500)=-.883857
COS-10000=-.9523750	cos (-10000)=-.952141
COS-999999= .195093	cos (-999999)= .19509
COS8.377580409=-.4999996	cos (8.37758)=-.5
COS15707.96326= .9999988	cos (15708)= .999999
COS-9424.77796= .9999998	cos (-9424.78)= 1
COS162.3156204= .5000035	cos (162.316)= .499993

Comment

As the number becomes larger, the error increases in significance.

Example: COS-10000=-0.9523750 ... -0.85282 (True value)

This is because, the integer part of the value obtained by dividing the value by 2 is cut off and the fractional part is used. For this reason, as the number becomes larger, the significant figures in the fractional part decrease.

The same can be said for the SIN and TAN subroutines.

(6) SIN subroutine

μCOM-87AD	PC-8001
SIN0= 0.000000	sin (0)= 0
SIN0.2= .1986692	sin (.2)= .198669
SIN0.5= .4794254	sin (.5)= .479426
SIN0.6= .5646424	sin (.6)= .564643
SIN0.8= .7173561	sin (.8)= .717356
SIN1.1= .8912075	sin (1.1)= .891207
SIN1.45= .9927130	sin (1.45)= .992713
SIN1.88= .9525762	sin (1.88)= .952576
SIN2.22= .7965655	sin (2.22)= .796566
SIN2.77= .3631000	sin (2.77)= .3631
SIN-2.77=-.3631000	sin (-2.77)=-.3631
SIN3.111= 3.058796E-02	sin (3.111)= .030588
SIN3.333=-.1902402	sin (3.333)=-.19024
SIN-3.333= .1902402	sin (-3.333)= .19024
SIN3.141592653= 1.872535E-07	sin (3.14159)= 0
SIN6.283185307= 1.498028E-06	sin (6.28319)= 0
SIN9.42477796=-7.490141E-07	sin (9.42478)= 0
SIN1.570796326= 1.000000	sin (1.5798)= .999959
SIN10.99557428=-1.000000	sin (10.9956)=-1
SIN-32.98672286=-1.000000	sin (-32.9867)=-1
SIN33.3333333= .9405295	sin (33.3333)= .940531
SIN111.11=-.9144961	sin (111.11)=-.914496
SIN31415.92653=-3.067957E-03	sin (31415.9)=-6.13589E-03
SIN157079.6326=-1.227154E-02	sin (157080)=-.0122715
SIN5.55555555=-.6651012	sin (5.55556)=-.665102
SIN77E-1= .9881681	sin (7.7)= .988168
SIN100=-.5063675	sin (100)=-.506368
SIN500=-.4677570	sin (500)=-.467757
SIN1000= .8268607	sin (1000)=.826861
SIN10000=-.3056596	sin (10000)=-.30566
SIN50000=-.9998306	sin (50000)=-.999831
SIN90000=-.3426607	sin (90000)=-.342661
SIN999999=-.9807852	sin (999999)=-.980785
SIN9999999= 1.000000	sin (1E+07)= 1
SIN-50000= .9998306	sin (-50000)= .999831
SIN-10000= .3056596	sin (-10000)= .30566
SIN-159.435827=-.7071153	sin (-159.436)=-.707115
SIN46.60029102= .5000035	sin (46,6003)= .500001
SIN-8.377580409=-.8660253	sin (-8.37758)=-.866025

(7) TAN subroutine

μCOM-87AD	PC-8001
TAN0= 0.000000	tan (0)= 0
TAN-1.047197551=-1.732051	tan (-1.0472)=-1.73205
TAN2.09438=-1.732110	tan (2.09438)=-1.73211
TAN-2.0943951:= 1.732050	tan (-2.0944)= 1.73205
TAN2.1=-1.709847	tan (2.1)=-1.70985
TAN2.094395102=-1.732050	tan (2.0944)=-1.73205
TAN.523598775= .5773501	tan (.523599)= .57735
TAN10471.45191= .5756471	tan (10471.5)= .577436
TAN157080.1562= .5541788	tan (157080)= .570502
TAN1000= 1.470218	tan (1000)= 1.47022
TAN10000= .3209446	tan (10000)= .320945
TAN999999=-5.027339	tan (999999)=-5.02734
TAN9999999=?	tan (1E+07)=
TAN500= .5292224	tan (500)= .529222

(8) LN subroutine

μCOM-87AD	PC-8001
LN1E36= 82.89307	ln (1E+36)= 82.8931
LN1.6E30= 69.54757	ln (1.6E+30)= 69.5476
LN2.7182919= 1.000004	ln (2.71829)= 1
LN98765.4321= 11.50050	ln (98765.4)= 11.5005
LN12345.6789= 9.421061	ln (12345.7)= 9.42106
LN3.3333333= 1.203973	ln (3.33333)= 1.20397
LN1= 1.966953E-06	ln (1.96695E-06)=-13.139
LN0.000000001=-20.72326	ln (1E-09)=-20.7233
LN0.9999999E-30=-69.07755	ln (1E-30)=-69.0776
LN2.22E-35=-79.79298	ln (2.22E-35)= -79.793
LN9.11E-40=?	ln (0)=
LN4.539992976E-5=-10.00000	ln (4.53999E-05)=-10
LN1318815734= 21.00000	ln (1.31882E+09)= 21
LN20.08553692= 3.000000	ln (20.0855)= 3
LN2.590448618= .9518313	ln (2.59045)= .951831
LN59874.14171= 11.00000	ln (59874.1)= 11
LN0=?	
LN-0.00000001=?	

(9) LOG subroutine

μCOM-87AD	PC-8001
LOG0=?	log (0)=
LOG-0.0000000001=?	log (-1E-10)=
LOG1= 8.542369E-07	log (1)= 0
LOG0.1111111E-20=-20.95424	log (1.11111E-21)=-20.9542
LOG0.3333333E-30=-30.47712	log (3.33333E-31)=-30.4771
LOG0.7777777E-40=?	log (0)=
LOG0.9999999E-36=-36.00000	log (1E-36)=-36
LOG987654321= 8.994606	log (9.87654E+03)= 8.99461
LOG1234567890= 9.091516	
LOG9.999999999= 1.000000	log (10)= 1
LOG66.666666666= 1.823909	log (66.6667)= 1.82391
LOG0.4444444444=-.3521825	log (.444445)=-.352183
LOG10= 1.000000	log (10)= 1
LOG1000= 3.000000	log (1000)= 3
LOG100000= 5.000000	log (100000)= 5
LOG1E30= 30.00000	log (1E+30)= 30
LOG1E37= 37.00001	log (1E+37)= 37
LOG1E36= 36.00001	log (1E+36)= 36
LOG1E38= 38.00000	log (1E+38)= 38

(10) EXP subroutine

μCOM-87AD	PC-8001
EXP80= 5.540605E+34	exp (80)= 5.54061E+34
EXP85= 8.222998E+36	exp (85)= 8.22302E+36
EXP88=?	exp (88)=
EXP63= 2.293768E+27	exp (63)= 2.29378E+27
EXP60= 1.142008E+26	exp (60)= 1.14201E+26
EXP52= 3.831011E+22	exp (52)= 3.831E+22
EXP40= 2.353852E+17	exp (40)= 2.35385E+17
EXP20= 4.851631E+08	exp (20)= 4.85165E+08
EXP11= 59873.83	exp (11)= 59874.1
EXP10= 22026.47	exp (10)= 22026.5
EXP5= 148.4132	exp (5)= 148.413
EXP1= 2.718282	exp (1)= 2.71828
EXP0= 1.000000	exp (0)= 1
EXP0.3333333= 1.395612	exp (.333333)= 1.39561
EXP0.987654321= 2.684930	exp (.987654)= 2.68493
EXP-60= 8.756509E-27	exp (-60)= 8.75651E-27
EXP-80= 1.804858E-35	exp (-80)= 1.80486E-35
EXP-87= 1.645807E-38	exp (-87)= 1.64581E-38
EXP-88= 0.000000	exp (-88)= 6.05462E-39
EXP-88.5= 0.000000	exp (-88.5)= 3.67231E-39
EXP1.609437912= 5.000001	exp (1.60944)= 5
EXP2.995732273= 20.00000	exp (2.99573)= 20
EXP-3.688879454= 2,499999E-02	exp (-3.68888)= .025
EXP4.465908118= 87.00001	exp (4.46591)= 87
EXP4.605170186= 99.99999	exp (4.60517)= 100
EXP198.022318=?	exp (198.022)=
EXP46.05170186= 1.000005E+20	exp (46.0517)= 9.99999E+19

(11) SQR subroutine

μCOM-87AD	PC-8001
SQR1= 1.000001	sqr (1)= 1
SQR4= 2.000003	sqr (4)= 2
SQR121= 11.00000	sqr (121)= 11
SQR11.11110888= 3.333327	sqr (11.1111)= 3.33333
SQR60.49381506= 7.777717	sqr (60.4938)= 7.77778
SQR30.86358025= 5.555501	sqr (30.8636)= 5.5555
SQR.3333333333= .5773519	sqr (.333333)= .57735
SQR.8888888888= .9428087	sqr (.888889)= .942809
SQR.00000000001= 3.162277E-06	sqr (1E-11)= 3.16228E-06
SQR0.99999E-40= 0.000000	sqr (0)= 0
SQR0.7777777E-35= 2.788863E-18	sqr (7.77778E-36)= 2.78887E-18
SQR0.3333333E-20= 5.773494E-11	sqr (3.33333E-21)= 5.77352E-11
SQR-0.000000001=?	sqr (-1E-08)=

(12) POWER subroutine

μCOM-87AD	PC-8001
0Λ0= 1.000000	0 Λ 0 = 1
0Λ1= 0.000000	0 Λ 1 = 0
1Λ0= 1.000000	1 Λ 0 = 1
1Λ1= 1.000002	1 Λ 1 = 1
1Λ-1= .9999975	1Λ -1 = 1
2Λ-2= .2499989	2Λ -2 = .25
-1Λ2.567=?	-1 Λ 2.567 =
-1Λ-4= .9999917	-1 Λ-4 = 1
0Λ-9.8765=?	0 Λ-9.8765 =
-3.456789Λ4= 142.7880	-3.45679 Λ 4 = 142.788
-9.99999999Λ0= 1.000000	-10 Λ 0 = 1
-543231Λ-99= 0.000000	-54321 Λ -99 = 0
-2.222222222Λ-3=-9.112471E-02	-2.22222 Λ -3 =-.091125
5.555555555Λ-1.111111111= .1487735	5.55556 Λ -1.11111 = .148773
9.87654321Λ1.23456789= 16.90081	9.87654 Λ 1.23457 = 16.9008
-8Λ100=?	-8 Λ 100 =
8.88888888Λ10.00001= 3.079557E+09	8.88889 Λ 10 = 3.07953E+09
9Λ1.2345= 15.06642	9 Λ 1.2345 = 15.0665
2.19Λ-9.12= 7.855002E-04	2.19Λ -9.12 = 7.85505E-04
7.89Λ10.123= 1.205343E+09	7.89 Λ 10.123 = 1.20535E+09
90.1Λ10.00001= 3.525869E+19	90.1 Λ 10 = 3.52588E+19
30Λ30=?	30 Λ 30 =
19.876Λ-19.876= 1.564628E-26	19.876 Λ -19.876 = 1.56464E-26
9Λ9= 3.874231E+08	9 Λ 9 = 3.87421E+08
9.1Λ9.1= 5.336737E+08	9.1 Λ 9.1 = 5.33674E+08
9.999999Λ10.0000001= 1.000003E+10	10 Λ = 1E+10

(13) ARCTAN subroutine

μCOM-87AD	PC-8001
ARCTAN0= 0.000000	arctan (0)= 0
ARCTAN1= .7854021	arctan (1)= .785398
ARCTAN1.732050807= 1.047198	arctan (1.73205)= 1.0472
ARCTAN-0.577350269=-.5235988	arctan (-.57735)=-.523599
ARCTAN10= 1.471128	arctan (10)= 1.47113
ARCTAN-10=-1.471128	arctan (-10)=-1.47113
ARCTAN1000= 1.569796	arctan (1000)= 1.5698
ARCTAN1000000= 1.570795	arctan (1E+06)= 1.5708
ARCTAN1000000000= 1.570796	arctan (1E+09)= 1.5708
ARCTAN9999999999= 1.570796	arctan (1E+10)= 1.5708
ARCTAN3.333333= 1.279340	arctan (3.33333)= 1.27934
ARCTAN-3.333333=-1.279340	arctan (-3.33333)=-1.27934

(14) ARCSIN subroutine

μCOM-87AD	PC-8001
ARCSIN1= 1.570796 ARCSIN-1=-1.570796 ARCSIN0= 0.000000 ARCSIN-0.5=-.5235991 ARCSIN0.707106781= .7853938 ARCSIN-0.866025403=-1.047193 ARCSIN-1.00000001=-1.570796 ARCSIN-1.000001=? ARCSIN1.0000001=? ARCSIN1.00000001= 1.570796	arcsin (0) = 0 arcsin (-.5)=-.523599 arcsin (.707107)= .785398 arcsin (-.866025)=-1.0472

(15) ARCCOS subroutine

μCOM-87AD	PC-8001
ARCCOS1= 0.000000	arccos (1)= 0
ARCCOS-1= 3.141593	arccos (-1)= 3.14159
ARCCOS0= 1.570796	
ARCCOS0.523598775= 1.019727	arccos (.523599)= 1.01973
ARCCOS0.866025403= .5236036	arccos (.866025)= .523599
ARCCOS-0.5= 2.094395	arccos (-.5)= 2.09439
ARCCOS-0.866025403= 2.617989	arccos (-.866025)= 2.61799
ARCCOS0.707106781= .7854026	arccos (.707107)= .785398
ARCCOS1.00000001= 0.000000	arccos (1)= 0
ARCCOS-0.1= 1.670964	arccos (-.1)= 1.67096
ARCCOS0.1= 1.470629	arccos (.1)= 1.47063
ARCCOS0.7777777= .6796740	arccos (.777778)= .679674
ARCCOS-0.99999999= 3.141593	arccos (-1)= 3.14159
ARCCOS4.0=?	
ARCCOS-1.00000001= 3.141593	arccos (-1)= 3.14159
ARCCOS0.3333333= 1.230960	arccos (.333333)= 1.23096
ARCCOS-0.88888888= 2.665711	arccos (-.888889)= 2.66571

(16) TRACA subroutine

```
1<-1.570796326= 0.000000       : -1.000000
1<0.523598775= .8660254        :  .4999999
7<2.617993878=-6.062178        :  3.500001
99<-2.094395102=-49.50002      : -85.73652
8.8888888<2.35619449=-6.285396       :  6.285392
7.7777777<-57.52359877= 4.365061     : -6.437396
5.6666666 3145.519644=-4.008475      : -4.005402
2.8284217,-31416.71193= 31416.54     < -1.570706
2.888888888 -31416.71193= 2.036476   : -2.049010
-0.9999.00= .9998995         < 3.141593     ?
-0.999<3.14= .9989987        , -1.591372E-03
1.777777,2.094395102= 2.747178       < .8669852
1.777777<2.094395102=-.8888888       :  1.539600
9.999999<-2.617993878=-8.660253      : -5.000000
4.444444<-3.141592653=-4.444444      : -8.322378E-07
99999999<-1.570796326= 0.000000      : -9.999999E+07
```

(17) TRACB subroutine

```
1,1= 1.414215      < .7854021
1,-1= 1.414215     < -.7854021
-1,1= 1,414215     <  2.356191
-1,-1= 1.414215    <  -2.356191
7.777777,0.101010= 7.778374      < 1.298627E-02
-3.3333333,-0.785398163= 3.424603     < -2.910194
6.66666,-6.66666= 9.428085     < -.7854021
0.5,-0.866025403= .9999994     < -1.047198
-0.707106871,-0.707106781= 1.000002      < -2.356199
-8.8888888,-9.9999999= 13.37953     < -2.297437
-0.0001E-7,77.77777= 77.77783         < 1.570796
9.9E15,-2.22222222= 9.900027E+15  < -2.244668E-16
-6.66666E15,-0.012345E-15= 6.666662E+15 < -3.141593
1,-9= 9.055392     < -1.460139
12345.67898,0.987654321= 12345.68      < 8.000000E-05
0.5,0.866025403= .9999994      < 1.047198
-0.707106781,0.707106781= 1.000000      < 2.356191
-0.8660254,-0.5= .9999994     < -2.617994

-7.071067811,-7.07810= 10.00498     < -2.355694
11111111,11111111= 1.571339E+07   < .7854021
-999999999,-999999999= 1.414215E+09  < -2.356191
666666666,-666666666= 9.428083E+08  < -.7854021
```

```
;
;       ;       ;       ;       ;       ;       ;
;*************************************************
;*                                               *
;*      UCOM87AD B.F.P.ARITHMETIC PACKAGE        *
;*                      TEST PROGRAME            *
;*                                               *
;*************************************************
;
;
;*************************************************
;*      POWER ON INITIALIZE ROUTINE              *
;*************************************************
;
        MVI     PC,0H
;
        MVI     A,00000011B
        MOV     MCC,A   ;SET MCC REG.
;
        MVI     A,11111011B
        MOV     MC,A    ;SET PC2 TO OUTPUT MODE
;
        MVI     A,11001110B
        MOV     SML,A   ;SET SML REG.
                        ;CLOCK LATE = *16
                        ;CHAR.LENGTH = 8 BITS
                        ;PARITY DISEBLE
                        ;STOP BITS = 2 BITS
;
        MVI     A,06H   ;BOW LATE = 4800
        MOV     TM0,A
        MVI     A,1
        MOV     TM1,A
;
        MVI     TMM,01100001B
;
        MVI     SMH,00001100B
;
;       ;       ;       ;       ;       ;       ;
;*************************************************
;*      MAIN ROUTINE                             *
;*************************************************
;
MAIN1:
        MVI     V,0FFH
        LXI     SP,STACK        ;SET STACK
;
        LXI     H,CHARST        ;SET CHAR. START ADDR.
        MVI     C,80-1  ;SET BUFFER SIZE
;
MAIN2:
        CALL    CI      ;GET 1 CHAR. FROM CONSOLE
;
        CALL    CO      ;PRINT IT
;
```

```
        STAX    H+          ;SET IT TO BUFFER
        NEI     A,'='
        GJMP    MAIN3       ;IF CHAR.= '='
        DCR     C           ;DECREMENT BUFFER SIZE
        GJMP    MAIN2       ;LOOP UNTILL '='
        GJMP    ERROR       ;IF BUFFER OVER
;
MAIN3:
        LXI     H,CHARST
        SHLD    ADDRS
        CALL    EXPR        ; GET ! / LEFT EXPR
        ;
        CALL    CHARD
        EQI     A,','
        GJMP    MAIN31
        ;
        MVI     A,'<'
        GJMP    MAIN33
        ;
MAIN31: NEI     A,'<'
        GJMP    MAIN32
        ;
        CALL    ADADJ
        GJMP    MAIN4
MAIN32:
        MVI     A,','
        LXI     H,TRACA
MAIN33:
        LXI     H,TRACB
        PUSH    V
        LXI     B,MAIN40
        PUSH    B
        PUSH    H           ; PUSH JUMP ADDR
        ;
        CALL    PUSHF       ; SAVE LEFT EXPR
        CALL    EXPR        ; GET ! / RIGHT EXPR
        CALL    POPR        ; RESTORE LEFT EXPR
        CALL    STORE1      ; SAVE RIGHT EXPR
        CALL    LOADA0      ; GET ! / LEFT EXPR TO B.F.P.ACC
        ;
        LXI     H,WXE
        RET                 ; JUMP TRACA OR TRACB
        ;
MAIN40: GJMP    ERROR
        ;
        CALL    PUSHR
        CALL    OUT
        CALL    POPR
        CALL    LOADA0      ; GET !
        ;
        MVI     A,' '
        CALL    CO
        POP     V
        CALL    CO
        MVI     A,' '
```

```
        CALL    CO
        ;
MAIN4:  CALL    OUT
;
        CALL    CHARD   ;1CHAR.READ
        NEI     A,'='
        GJMP    MAIN6   ;IF '='
;
MAIN5:
ERROR:
        MVI     A,'?'
        CALL    CO      ;
;
MAIN6:
        MVI     A,0DH
        CALL    CO      ;PRINT CR
        MVI     A,0AH
        CALL    CO      ;PRINT LF
;
        GJMP    MAIN1   ;LOOP NEXT ARITHMETIC
;
;  OUT  SUBROUTINE
;
OUT:    LXI     H,OUTADR
        CALL    BFOUT
        ;
        LXI     H,OUTADR
        MVI     C,13-1
OUT1:
        LDAX    H+
        CALL    CO
        DCR     C
        GJMP    OUT1
        RET
;
        EJECT
;
;=============================================
;=       EXPR  ROUTINE                       =
;=============================================
;
EXPR:
        CALL    TERM    ;GET LEFT TERM
;
EXPR1:
        CALL    CHARD   ;GET CHAR.
        EQI     A,'+'
        GJMP    EXPR2   ;IF NOT PLUS
;
;       IF PLUS
;
        CALL    PUSHF   ;PUSH LEFT TERM
        CALL    TERM    ;GET RIGHT TERM
        GJMP    EXPR3
;
```

```
EXPR2:
        EQI     A,'-'
        GJMP    ADADJ   ;GOTO ADDR.ADJUST AND RETURN
;
;       IF MINUS
;
        CALL    PUSHF   ;PUSH LEFT TERM
        CALL    TERM    ;GET RIGHT TERM
        CALL    BFCOMP  ;COMPRIMENT IT,S SIGN
;
EXPR3:
        CALL    POPR    ;POP LEFT TERM
        CALL    BFADD1  ;LEFT TERM + RIGHT TERM
        GJMP    ERROR   ;IF OVER FLOW
        GJMP    EXPR1   ;GOTO NEXT EXPR
;
;       ;       ;       ;       ;       ;
;===============================================
;=      TERM ROUTINE                           =
;===============================================
;
TERM:
        CALL    FACTR   ;GET LEFT FACTOR
        NOP
;
TERM1:
        CALL    CHARD   ;READ CHAR.
        EQI     A,'*'
        GJMP    TERM3   ;IF NOT '*'
;
;       IF MULTIPLICATION
;
TERM2:
        CALL    PUSHF   ;PUSH LEFT FACTOR
        CALL    FACTR   ;GET RIGHT FACTOR
        NOP
        ;
        CALL    POPR    ;POP  LEFT FACTOR
        CALL    BFMUL1  ;LEFT FACTOR * RIGHT FACTOR
        GJMP    ERROR   ;IF OVER FLOW
        GJMP    TERM1   ;GOTO NEXT CHARACTER
;
TERM3:
        EQI     A,'/'
        GJMP    ADADJ   ;GOTO ADDR.ADJUST AND RETURN
;
;       IF DIVISION
;
        CALL    PUSHF   ;PUSH LEFT FACTOR
        CALL    FACTR   ;GET RIGHT FACTOR
        NOP
        ;
        CALL    STORE1  ;STORE RIGHT FACTOR
        CALL    POPR
        CALL    LOADAO  ;RESTOE BFP ACC
```

```
        LXI     H,WXE
        CALL    BFDIV
        GJMP    ERROR   ;IF OVER FLOW
        GJMP    TERM1   ;GOTO NEXT TERM
;
;       ;       ;       ;       ;       ;       ;
;===============================================
;=      FACTOR  ROUTINE                         =
;===============================================
;
FACTR:
        CALL    CHARD   ;CHAR. READ
        NEI     A,' '
        GJMP    FACTR   ;READ UNTILL NOT SPACE
;
        CALL    ADADJ   ;DECR. ADDRESS
;
        LXI     D,FTABL
FACTR1:
        PUSH    H
;
FACT11:
        CALL    CHARD   ;CHAR.READ FROM THE STRING
        MOV     B,A
        LDAX    D
        ANI     A,7FH
        ;
        EQA     A,B
        GJMP    FACTR2  ;IF NOT MUCH
;
        LDAX    D+
        ONI     A,80H
        GJMP    FACT11  ;SEACH UNTILL TABLE END
;
;       IF FOUND FUNCTION
;
        POP     H       ;DUMMY POP
        ;
        LDAX    D+
        MOV     L,A
        LDAX    D+
        MOV     H,A
        LXI     B,FACT31
        PUSH    B
        PUSH    H
        ;
        CALL    FACTR
        RET             ; JUMP THE FUNCTION
        ;
;
FACTR2:
        MVI     A,80H
        ONAX    D+
        GJMP    FACTR2  ;LOOP UNTILL THE TABLE END
        INX     D
```

```
          INX     D         ;
          POP     H         ;POP CHAR.STRING ADDR.
          SHLD    ADDRS
          LDAX    D         ;GET NEXT TABLE
          DCR     A
          GJMP    FACTR1    ;SEACH NEXT FANCTION TABLE
;                           ;IF END FUNCTION TABLE
;
FACTR3:
          CALL    CHARD     ;CHAR. READ FROM STRING
          EQI     A,'('
          GJMP    FACTR4
          CALL    EXPR
          CALL    CHARD     ;CHAR.READ FROM STRING
          EQI     A,')'
FACT31:   GJMP    ERROR     ;IF NESTING ERROR
          GJMP    FACTR5
;
FACTR4:
          CALL    ADADJ     ;DECR. STRING ADDR.
          CALL    BFINP
          GJMP    ERROR     ;IF OVER FLOW
FACTR5:   CALL    CHARD     ;CHAR. READ FROM STRING
          EQI     A,5EH     ; POWER FUNCTION ?
          GJMP    ADADJ     ;IF NOT POWER THEN
;
;         IF POWER FUNCTION
;
          CALL    PUSHF     ;
          CALL    FACTR     ;
          NOP
          ;
          CALL    STORE1    ;
          CALL    POPR
          CALL    LOADAQ    ;RESTORE BFP ACC
          LXI     H,WXE
          CALL    POWER     ;
          GJMP    ERROR     ;IF OVER FLOW
          RET
;
          EJECT
;
;=================================================
;=       ADDRESS  ADJUST  ROUTINE                =
;=================================================
;
ADADJ:
          LHLD    ADDRS
          DCX     H
          SHLD    ADDRS
          RET
;
;         ;       ;       ;       ;       ;       ;
;=================================================
;=       1 CHAR.  INPUT FROM CONSOLE             =
```

```
;================================================
;
CI:
        SKIT    FSR
        GJMP    CI          ;LOOP UNTILL FSR
;
        SKNIT   ER
        GJMP    CI          ;RETRY IF ERROR
;
        MOV     A,RXB       ;GET 1 CHAR.
        ANI     A,7FH
        RET
;
;       ;       ;       ;       ;       ;
;================================================
;=      1 CHAR. OUTPUT TO CONSOLE                =
;================================================
;
CO:
        OFFI    PC,CTS
        GJMP    CO          ;LOOP UNTILL /CTS = 0
;
        MOV     TXB,A
;
CO1:
        SKIT    FST
        GJMP    CO1         ;LOOP UNTILL FST = 1
;
        PUSH    D
        PUSH    V
        LXI     D,800
CO2:
        DCX     D
        MOV     A,D
        ORA     A,E
        SK      Z
        GJMP    CO2
;
        POP     V
        POP     D
;
        RET
;
        EJECT
;
;************************************************
;*                                              *
;*      BFINP  SUBROUTINE                       *
;*                                              *
;*       IN: HL REG <-- STRING START ADDR.      *
;*                                              *
;*       OUT:  BFP. ACC                         *
;*             RETS:   NORMAL   CY=0            *
;*             RET:  OVERFLOW   CY=1            *
;*                                              *
```

```
;***********************************************
;
BFINP:
        SHLD    ADDRS   ;SAVE STRING ADDR.
;
        MVI     A,0
        STAW    ACCE MOD 100H   ;CLEAR BFP.ACC
        LXI     D,TMP1
        STAX    D+      ;TMP1 = 0
        STAX    D+      ;TMP2 = 0
        MVIX    D,80H   ;TMP3 <-- 80H
;
        CALL    CHARD   ;GET CHAR.TO ACC
        NEI     A,' '
        GJMP    BFINP1  ;IF SPACE
        NEI     A,'+'
        GJMP    BFINP1  ;IF PLUS
        EQI     A,'-'
        GJMP    BFIN11
        MVIX    D,0     ;IF MINUS THEN TMP3 = 0
;
BFINP1:
        CALL    CHARD   ;CHAR.READ
;
BFIN11: MVI     B,0     ;CLEAR EXP
;
        EQI     A,'.'
        GJMP    BFIN13  ;IF NOT DECIMAL POINT
;
BFIN12:                 ;IF DECIMAL POINT FOUND
        XRAW    TMP2 MOD 100H
        STAW    TMP2 MOD 100H
        SK      Z
        GJMP    BFINP1  ;GOTO NEXT STRING
        GJMP    BFINP4  ;IF DUBLE DECIMAL POINT
;
BFIN13:
        NEI     A,'E'
        GJMP    BFINP3  ;IF EXP
;
BFIN14:
        SUI     A,'0'
        LTI     A,10
        GJMP    BFINP4  ;IF NOT DECIMAL DEGIT
;
BFINP2:
        STAW    TMP4 MOD 100H   ;SAVE DIGIT
        LXI     H,BFTEN
        CALL    BFMUL   ;PRE.RESULT * 10
        GJMP    BFRET   ;IF OVER FLOW
        CALL    STORE1  ;STORE PRE.RESULT
        LXI     H,WSE
        MVI     A,8
        STAX    H+      ;WSE = 8
        LDAW    TMP4 MOD 100H
```

```
        STAX    H+          ;WS1 = DIGIT
        MVI     A,0
        STAX    H+          ;WS2 = 0
        STAX    H+          ;WS3 = 0
        STAX    H+          ;WS4 = 0
        CALL    BFLT        ;GET NEW DIGIT TO BFP.ACC
        NOP
        LXI     H,WXE
        CALL    BFADD       ;ADD PRE RESULT TO NEW DIGIT
        GJMP    BFRET       ;IF OVER FLOW
        NEIW    TMP2 MOD 100H,0
        GJMP    BFINP1      ;GOTO NEXT STRING
        DCRW    TMP1 MOD 100H
        NOP
        GJMP    BFINP1      ;GOTO NEXT STRING
;
BFINP3:
        MVI     E,'+'
        CALL    CHARD       ;CHAR. READ
        NEI     A,'-'
        GJMP    BFIN31      ;IF MINUS
        EQI     A,'+'
        GJMP    BFIN32      ;IF NOT PLUS
BFIN31:
        MOV     E,A         ;SEVE EXP SIGN
        CALL    CHARD       ;MORE 1 CHAR. READ
BFIN32:
        MVI     B,0         ;CLEAR EXP DIGIT
        SUI     A,'0'       ;DIGIT IS 0 -- 9 ?
        LTI     A,10
        GJMP    BFINP4      ;IF ILEAGAL CHAR.
;
        MOV     B,A
        CALL    CHARD       ;GET LOW DIGIT
        SUI     A,'0'       ;IT IS 0 -- 9 ?
        LTI     A,10
        GJMP    BFIN33      ;IF ILLEGAL CHAR.
        PUSH    V           ; PUSH IT
        MVI     A,10
        MUL     B           ;B= B * 10
        MOV     A,EAL
        MOV     B,A         ;B = B * 10
        POP     V           ;POP LOW DIGIT
        ADD     B,A         ;B = B + ACC
        GJMP    BFIN30
;
BFIN33:
        CALL    ADADJ       ;ADDRESS ADJUST
BFIN30:
        EQI     E,'-'
        GJMP    BFIN40      ;IF NOT MINUS
        MVI     A,0         ;IF MINUS EXP SIGN
        SUB     A,B
        MOV     B,A         ;ADJUST EXP SIGN
        GJMP    BFIN40
```

```
;
BFINP4:
        CALL    ADADJ   ;ADDRESS ADJUST
BFIN40:
        LDAW    TMP3 MOD 100H
        STAW    ACCS MOD 100H   ;ACCS <-- TMP3
        MOV     A,B             ;GET EXP SIGN
BFIN41:
        ADDW    TMP1 MOD 100H
        SKN     Z
        GJMP    BFRETS  ;IF END
;
        STAW    TMP1 MOD 100H
        LXI     H,BFTEN
        ONI     A,80H
        GJMP    BFIN43
;
BFIN42:
        CALL    BFDIV
        NOP
        MVI     A,1     ;INCREMENT TMP1
        GJMP    BFIN41
;
BFIN43:
        CALL    BFMUL
        GJMP    BFRET   ;IF OVER FLOW
        MVI     A,0FFH  ;DECREMENT TMP1
        GJMP    BFIN41
;
;       ;       ;       ;       ;       ;       ;
;=============================================
;=      STRING CHARACTER READ ROUTINE        =
;=============================================
;
CHARD:
        LHLD    ADDRS
        LDAX    H+
        SHLD    ADDRS
        RET
;
        EJECT
;
;*********************************************
;*                                           *
;*      BFOUT ROUTINE                        *
;*                                           *
;*              IN : STORE START ADDRESS     *
;*                                           *
;*              OUT : NONE                   *
;*                                           *
;*********************************************
;
BFOUT:
        MVI     A,' '
        STAX    H+      ;SET PLUS SIGN
```

```
;
        EQIW    ACCE MOD 100H,0
        GJMP    BFOU01  ;IF BFP ACC <> 0
;
        MVI     A,'0'
        STAX    H+
        MVIX    H,'.'
        INX     H
        MVI     C,6-1
BFOU00:
        STAX    H+
        DCR     C
        GJMP    BFOU00
;
        PUSH    H
        GJMP    BFOU30
;
BFOU01:
        MVI     A,0
        STAW    TMP1 MOD 100H
        STAW    TMP2 MOD 100H
        EQIW    ACCS MOD 100H , 0
        GJMP    BFOU02  ;IF BFP ACC > 0
;
        MVI     A,'-'   ;IF BFP ACC < 0
        DCX     H
        STAX    H+
;
BFOU02:
        PUSH    H       ;SAVE STRING ADDR.
        CALL    BFABS   ;ABSOLUTE BFP ACC
        CALL    STORE1  ;SAVE BFP. ACC
;
;       SCALING 0.1 -- 1
;
BFOUT1:
        LTIW    ACCE MOD 100H,81H
        GJMP    BFOU12          ;IF BFP ACC >= 1 GOTO DIV
;
        CALL    PUSHF   ;SAVE BFP ACC
        LXI     H,BFN01 ;SET 0.1
        CALL    BFSUB   ;BFP ACC - 0.1
        NOP
        CALL    POPR    ;RESTORE BFP ACC TO OPERAND
        NEIW    ACCE MOD 100H , 0
        GJMP    BFOU03  ;IF BFP ACC = 0.1
        EQIW    ACCS MOD 100H ,80H
        GJMP    BFOU11  ;IF BFP ACC < 0.1 GOTO MULT
BFOU03:
        CALL    LOADA0  ;RESTORE BFP ACC
        LXI     H,BFDV58
        CALL    BFADD   ;ROUNDINDG
        NOP
        LTIW    ACCE MOD 100H , 81H
        GJMP    BFOUT1  ;ONE MORE TRY SCALING
```

```
        GJMP    BFOUT2
;                       ;IF 0.1 > BFP.ACC
BFOU11:
        CALL    LOADAO  ;LOAD BFP. ACC
        LXI     H,BFTEN
        CALL    BFMUL
        NOP
        DCRW    TMP2 MOD 100H   ;TMP2 <-- TMP2 -1
        NOP
        GJMP    BFOUT1
;                       ;IF 1 <= BFP. ACC
BFOU12:
        LXI     H,BFTEN
        CALL    BFDIV
        NOP
        INRW    TMP2 MOD 100H   ;TMP2 <-- TMP2+1
        NOP
        GJMP    BFOUT1
;
BFOUT2:
        LDAW    TMP2 MOD 100H
        MOV     B,A
        LTI     A,8
        MVI     B,1
        SUB     A,B     ;GET DECIMAL EXP.
        STAW    TMP2 MOD 100H   ;STORE IT
        MVI     A,7
        SUB     A,B     ;GET RIGHT DIGIT
        STAW    TMP3 MOD 100H   ;STORE IT
        MOV     A,B     ;GET LEFT DIGIT NUMBER
        STAW    TMP1 MOD 100H   ;STORE IT
;
BFOU21:
        DCRW    TMP1 MOD 100H
        GJMP    BFOU22
        LDAW    TMP3 MOD 100H
        STAW    TMP1 MOD 100H   ;TMP1 <-- TMP3
        MVIW    TMP3 MOD 100H ,0FFH ;TMP3 <-- OFFH
        NEIW    TMP1 MOD 100H ,0FFH
        GJMP    BFOUT3  ;IF MANTISSA END
        POP     H       ;GET STORE ADDRESS
        MVI     A,'.'
        STAX    H+
        PUSH    H
        GJMP    BFOU21  ;LOOP UNTILL MANTISSA END
;
;       GET 1 DECIMAL DIGIT
;
BFOU22:
        LXI     H,BFTEN
        CALL    BFMUL
        NOP
        CALL    BFIX0   ;CONVERT FIXED POINT NUMBER
        NOP
        LDAW    WS1 MOD 100H    ;GET 1 DIGIT
```

```
        ADI     A,'0'
        POP     H
        STAX    H+      ;STORE DECIMAL DIGIT
        PUSH    H
        MVIW    WSE MOD 100H , 8
        MVIW    WS1 MOD 100H , 0
        CALL    BFLT    ;CONVERT BFP.NUMBER
        NOP
        GJMP    BFOU21  ;GOTO NEXT DIGIT
;
BFOUT3:
        LDAW    TMP2 MOD 100H
        EQI     A,0
        GJMP    BFOU31
BFOU30:
        MVI     A,' '
        MOV     B,A
        MOV     C,A
        MOV     D,A
        GJMP    BFOU34
;
BFOU31:
        MVI     B,'+'
        ONI     A,80H
        GJMP    BFOU32
        MVI     B,'-'
        MVI     A,0
        SUBW    TMP2 MOD 100H
        STAW    TMP2 MOD 100H
BFOU32:
        MVI     C,0FFH
;
        LDAW    TMP2 MOD 100H
BFOU33:
        MOV     D,A
        INR     C
        NOP
        SUI     A,10
        SK      CY
        GJMP    BFOU33
        ADI     C,'0'   ;PLUS BIAS
        ADI     D,'0'
        MVI     A,'E'
BFOU34:
        POP     H
        STAX    H+      ;SET 'E'
        MOV     A,B
        STAX    H+      ;SET EXP.SIGN
        MOV     A,C
        STAX    H+      ;SET EXP. HIGH DIGIT
        MOV     A,D
        STAX    H+      ;SET EXP. LOW DIGIT
;
        CALL    LOADA1  ;RESTORE BFP. ACC
;
```

```
          RET
;
          EJECT
;
;  ***********************************************
;  *                                             *
;  *     BINARY FLOATING POINT SUBTRACT SUBROUTINE *
;  *                                             *
;  *        B.F.P.ACC < B.F.P.ACC - OPERAND      *
;  *                                             *
;  *                                             *
;  *     INPUT:                                  *
;  *             V REG -- OFFH                   *
;  *                                             *
;  *             BFSUB  - HL ( OPERAND TOP ADDR.)*
;  *                                             *
;  *     OUTPUT: RETS   -  NORMAL     CY=0       *
;  *             RET    -  OVERFLOW, CY =1       *
;  *                                             *
;  ***********************************************
;
          NOP               ; DUMMY FOR RET
BFSUB:    CALL     LOADO3   ; OPE <- (HL,---,HL+3)
          ;
          NOP
BFSUB1:   MVI      A,80H    ; MASK TO INVERT
          XRAW     WS1 MOD 100H
          STAW     WS1 MOD 100H
                            ; SIGN STORE
          GJMP     BFADD1   ;FALL INTO BFADD SUBROUTINE
;
          EJECT
;
;  ***********************************************
;  *                                             *
;  *    BINARY FLOATING POINT ADDITION SUBROUTINE*
;  *                                             *
;  *     B.F.P.ACC <- B.F.P.ACC + OPERAND        *
;  *                                             *
;  *    INPUT:                                   *
;  *             V REG  --  OFFH                 *
;  *                                             *
;  *             BFADD     HL  ( OPERAND TOP ADDR)*
;  *                                             *
;  *    OUTPUT: RETS   -  NORMAL     CY= 0       *
;  *             RET    -  OVERFLOW, CY= 1       *
;  *                                             *
;  ***********************************************
;
          NOP               ; DUMMY FOR RETS
BFADD:    CALL     LOADO3   ; OPE <- (HL,HL+1,---,HL+3)
          GJMP     BFADD1
          ;
          NOP
BFADDO:   CALL     POPR     ; OPE <- STACK
```

```
;
BFADD1: NEIW    WSE MOD 100H,0
        GJMP    BFRETS  ; IF ADDEND=0
;
; OPE 1ST FRCTN !NON-SIGNED! DISPLAY
;
        LXI     D,WS1   ; DE= OPE 1ST FRCTN ADDR
        LDAX    D
        MOV     B,A
        ORI     A,80H   ; OPE 1ST FRCTN !NON-SIGNED!
        STAX    D-      ; NON-SIGN STORE
        ;
        XRA     A,B
        LXI     H,ACCS
        XRAX    H-
                ; HL= BFP ACC EXP ADDR
        STAW    SF MOD 100H
                ; OPERATION FLAG SET
        LDAX    H
        NEI     A,0     ; JUDGE  BFP.ACC=0
        GJMP    BFADS   ; IF AUGEND=0
        ;
        ANIW    WS4 MOD 100H,0
                ; 4TH FRCTN  INITIAL OCLEAR
;
;  CHECK FOR / B.F.P.ACC < OPERAND  BY EXPONENT
;
        SUBNBX  D       ; DIFFERENCE OF EXPONENT
        GJMP    BFAD2   ; IF  ACCE < OPE.WSE
;
; CHECK FOR / DIFFERENCE OF EXP, TOO LAGE
;
        GTI     A,24+1
        GJMP    BFAD3   ; IF DIFFERENCE < 25
        GJMP    BFRETS  ; RESULT = BFP.ACC
;
; CHECK FOR / BFP.ACC = NEGLIGIBLE
;
BFAD2:  GTI     A,100H-25
        GJMP    BFADS   ; BFP.ACC NEGLIGIBLE !
;
; SET !SIGN OF RESULT!,!DIFER OF EXP!
;
        NEGA            ; EXP COMPLEMENT
        PUSH    V       ; SAVE EXP
        ;
        LDAX    D+
                ; DE= WS1 ADDR
        STAX    H+
                ; HL= ACCS ADDR
        LDAW    SF MOD 100H
        XRAX    H
        STAX    H+      ; RESULT.SIGN STORE
                        ; HL = ACC1 ADDR
```

```
;  EXCHANGE / BFP.ACC AND OPERAND /
;
          MVI       C,3-1
EXACOP:   LDAX      H
          MOV       B,A
          LDAX      D
          STAX      H+
          MOV       A,B
          STAX      D+
          ;
          DCR       C
          GJMP      EXACOP
;
;  (WS1,2,3,4) RIGHT SHIFT
;                FOR EQUALLY OF EXP
;
          POP       V
BFAD3:    CALL      BFRSH
;
;   BRANCHING   OF OPERATION (+,-)
;
          LXI       H,ACC3
          LXI       D,WS3
          ;
          OFFIW     SF MOD 100H,80H
          GJMP      BFAD9     ; IF !SUBTRACT!
;
;   ADDITION / OPE <- OPE + BFP.ACC
;                     (WS1,2,3) + (ACC1,2,3)
;
          CLC
          MVI       C,3-1
BFAD4:    LDAX      H-
          ADCX      D
          STAX      D-
          ;
          DCR       C
          GJMP      BFAD4
          ;
          SK        CY        ; OVER FLOW ?
          GJMP      BFADR     ; NO CARRY
;
;  RIGHT SFIFT FOR / END AROUND CARRY
;
          CALL      BFRSH4    ; (WS1,2,3,4) SHIFT
          ;
          INRW      ACCE MOD 100H
                    ; EXP INCREMENT
          GJMP      BFADR     ; GO TO ROUNDING
          GJMP      BFRET     ; IF OVER-FLOW
;
;         OPE. <-- BFP.ACC - OPE.
;                  (ACC1,2,3)-(WS1,2,3,4)
;
BFAD9:    INX       D
```

```
        LDAX    D-
                ; DE= WS3 ADDR
        RAL             ; CARRY FROM MSB OF WS4
        MVI     C,3-1
        ;
BFAD10: LDAX    H-
        SBBX    D
        STAX    D-
        ;
        DCR     C
        GJMP    BFAD10
        ;
BFAD11: SKN     CY
        CALL    BFCOM1  ; (WS1,2,3,4) COMPLEMENT
;
; NORMALIZE !
;
        CALL    BFNOR
        GJMP    BFRETS  ; IF ACCE=0
;
; ROUNDING !
;
BFADR:  CALL    BFROND
        GJMP    BFRET   ; IF OVERFLOW
        GJMP    BFRETS
;
; TRANS / (ACCE,S,1,2,3) <- (WSE,A,1,2,3)
;
BFADS:  LDAW    SF MOD 100H
        LXI     H,ACCS
        XRAX    H-
                ; HL= ACCE ADDR
        MOV     B,A
        CALL    BFSTR1
        GJMP    BFRETS
        EJECT
;
; ************************************************
; *                                              *
; *    BINARY FLOATING POINT MULTIPLY SUBROUTINE *
; *                                              *
; *     B.F.P.ACC <- B.F.P.ACC * OPERAND         *
; *                                              *
; *    INPUT:                                    *
; *                V REG  --  OFFH               *
; *                                              *
; *            BFMUL    HL <- OPERAND TOP ADDR   *
; *                                              *
; *    OUTPUT: RETS  - NORMAL    CY=0            *
; *            RET   - OVERFLOW, CY= 1           *
; *                                              *
; ************************************************
;
        NOP             ; DUMMY FOR RETS
BFMUL:  CALL    LOAD03  ; OPE <- (HL,HL+1,---,HL+3)
```

```
        GJMP    BFMUL1
        ;
        NOP
BFMULO: CALL    POPR    ; OPE <- STACK
        NOP
BFMUL1: ONIW    WSE MOD 100H,0FFH
        GJMP    BFMULZ  ; IF OPERAND=0
        ;
;
;   EXPONENT PART OPERATION
;
        CALL    BFRD    ; OPERATION OF EXP
        GJMP    BFRET   ; IF OVERFLOW
        ;
        SKN     Z
        GJMP    BFRETS  ; IF UNDERFLOW OR RESULT=0
        ;
;
;   (ACC1,2,3) * (WS1,2,3)
;
        CALL    BFMX
;
;   NORMALIZE
;
        CALL    BFNOR
        GJMP    BFRETS  ; RESULT= 0
;
;   ROUNDING
;
        CALL    BFROND
        GJMP    BFRET   ; IF OVERFLOW
        GJMP    BFRETS  ; NORMAL RESULT
;
;   RESULT=0 /  PROCESS
;
BFMULZ: ANIW    ACCE MOD 100H,0
        GJMP    BFRETS
;
;   MULTIPLY  (WS1,2,3) <- (WS1,2,3) * (ACC1,2,3)
;
BFMX:   MVI     C,3-1
        LXI     H,WS1
        LXI     D,WA1
        ;   (WA1,2,3) <- (WS1,2,3)
        ;   (WS1,2,3)  OCLEAR
BFMX1:  LDAX    H
        MOV     B,A
        XRA     A,A
        STAX    H+
        MOV     A,B
        STAX    D+
        ;
        DCR     C
        GJMP    BFMX1
;
```

```
;   (WS1,2,3) * B.F.P.ACC -1BYTE / 3TH OPERATE
;
        MVIW    CMULTO MOD 100H,3-1
        LXI     H,ACC3
BFMX2:  LDAX    H-
        MOV     C,A
                ; C: B.F.P.ACC 1BYTE
        PUSH    H
        CALL    BFMX3
        POP     H
        ;
        DCRW    CMULTO MOD 100H
        GJMP    BFMX2
        RET
        ;
;
;   (3BYTE/ WA1,2,3) * (1BYTE/ B.F.P.ACC)
;
BFMX3:  CALL    MULT
        PUSH    V
;
;   RIGHT SHIFT / (WS2,3,4) <- (WS1,2,3)
;
        CALL    BFRSHO
        POP     V
        STAW    WS1 MOD 100H
        RET
;
; MULTIPLY /  (3BYTE:WA1,2,3) * 1BYTE
;
MULT:   MVI     B,0     ; INITIAL OCLEAR
        MVIW    CMULT1 MOD 100H,3-1
                ; MULT COUNTER
        LXI     H,WA3
        LXI     D,WS3
        ;
MULT1:  CALL    MULT2
        ;
        DCRW    CMULT1 MOD 100H
        GJMP    MULT1
        RET             ; MULTIPLY END !
        ;
MULT2:  LDAX    H-
        MUL     C
        EADD    EA,B
;
        LDAX    D
        EADD    EA,A
;
        MOV     A,EAL
        STAX    D-
        MOV     A,EAH
        MOV     B,A
        RET
;
```

```
        EJECT
;
; ************************************************
; *                                              *
; *    BINARY FLOATING POINT DIVISION SUBROUTINE *
; *                                              *
; *     B.F.P.ACC <- B.F.P.ACC / OPERAND         *
; *                                              *
; *                                              *
; *     INPUT:                                   *
; *                V REG  --  OFFH               *
; *                                              *
; *               BFDIV    HL ( OPERAND TOP ADDR.)  *
; *                                              *
; *     OUTPUT    RETS     NORMAL    CY=0       *
; *               RET      OVERFLOW, CY=1       *
; *                                              *
; ************************************************
;
        NOP
BFDIV:  CALL    LOAD03
        GJMP    BFDIV1
;
        NOP
BFDIV0:
        CALL    POPR
;
        NOP
BFDIV1: LDAW    WSE MOD 100H
        NEGA            ; OPE EXP PART COMPLEMENT
        NEI     A,0
        GJMP    BFRET   ; IF DIVISOR=0
        ;
        STAW    WSE MOD 100H
;
; EXPONENT PART OPERATION
;
        CALL    BFRD
        GJMP    BFRET   ; IF OVERFLOW
        ;
        SKN     Z
        GJMP    BFRETS  ; IF RESULT=0 OR UNDERFLOW
        ;
;
; DIVISION / (ACC1,2,3) / (WS1,2,3)
;
        MVIW    WS4 MOD 100H,0
        CALL    BFDX
        GJMP    BFRET   ; IF OVERFLOW
;
; TRANS / (WS1,2,3,4) <- (WD1,2,3,4)
;
        LXI     H,WS4
        LXI     D,WD4
        ;
```

```
        LDAX    D-
        STAX    H-          ; WS4 <- 0
                            ; STORE 4TH MANTISSA
        LDAX    D-
        STAX    H-          ; WS3 <- WD3
        LDAX    D-
        STAX    H-          ; WS2 <- WD2
        LDAX    D-
        STAX    H-          ; WS1 <- WD1
        ;
;
; ROUNDING !
;
        CALL    BFROND
        GJMP    BFRET       ; IF OVERFLOW
        GJMP    BFRETS      ; NORMAL
;
;  DIVISION SUBROUTINE FOR MANTISSA
;
;           (WD1,2,3) <- (ACC1,2,3) / (WS1,2,3)
;
; B.F.P.ACC <- B.F.P.ACC - OPERAND
;
BFDX:   LXI     H,ACC3
        LXI     D,WS3
;
        CLC
        MVI     C,3-1
BFDX1:  LDAX    H
        SBBX    D-
        STAX    H-
;
        DCR     C
        GJMP    BFDX1
                            ; HL=ACCS,DE=WSE
;
; HALVES DIVISOR, AND STORE
;                   FOR NEXT BIT (QUOTIENT)
;
        STC                 ;SET MSB OFF WS1
        CALL    BFRSH4
;
; TRANS / (WD4,5,6) <- (ACC1,2,3)
;
        LXI     H,ACC3
        LXI     D,WD6
        ;
        LDAX    H-
        STAX    D-
        ;
        LDAX    H-
        STAX    D-
        ;
        LDAX    H-
        STAX    D-
```

```
;
;
; CHECK FOR NEXT OPERATION / (+,-)
;                       BY QUOTIENT.MSB
;
        OFFI    A,80H
        GJMP    BFDX9   ; IF REMAINDER NEGATIVE
;
; ADJUST EXPONENT AND SET QUOTIENT.LSB
;
        INRW    ACCE MOD 100H
        GJMP    BFDX3
        ;
        RET             ; OVERFLOW
        ;
BFDX3:
                        ; DE= 1TH QUOTIENT
        MVI     A,1
        STAX    D-
                        ; DE= 2TH QUOTIENT
        MVI     A,0
        STAX    D-
        STAX    D       ; DE = 3TH QUOTIENT
;
;  SUBTRACT DIVISOR FROM REMAINDER
;               IF IT IS POSSIBLE !
;
;   (WD4,5,6) <- (WD4,5,6) - (WS1,2,3,4)
;
BFDX4:
        LXI     H,WS4
        LXI     D,WD6
        MVI     A,0
        SUBX    H-
                        ;SET CARRY BY WS4
        MVI     C,3-1
        ;
BFDX5:  LDAX    D
        SBBX    H-
        STAX    D-
        ;
        DCR     C
        GJMP    BFDX5
;
;  CHECK FOR / DIVISION SUBROUTINE END ?
;
BFDX6:  OFFIW   WD1 MOD 100H,80H
        RETS
                ; IF QUOTIENT.MSB=1
;
;  (WD1,2,3,4,5,6) 6BYTE LEFT SHIFT BY BIT
;
        LDAW    WS4 MOD 100H
        RLL     A       ; CARRY FROM WS4.MSB
        :
```

```
        MVI     B,6-1
        LXI     D,WD6
        CALL    BFLSH1
;
;   JUDGE  NEXT OPERATION / (+,-) BY WD3.LSB
;
        OFFIW   WD3 MOD 100H,01H
        GJMP    BFDX4   ; IF REMAINDER: POSITIVE
        GJMP    BFDX7   ; IF REMAINDER: NEGATIVE
;
;   ADD DIVISOR / IF REMAINDER IS NEGATIVE
;
;       (WD4,5,6) <- (WD4,5,6) + (WS1,2,3)
;
BFDX9:
                        ; QUOTIENT 0 SET
        MVI     A,0
        STAX    D-
        STAX    D-
        STAX    D
                        ; ADDITION
BFDX7:  LXI     H,WS3
        LXI     D,WD6
        MVI     C,3-1
;
        CLC
BFDX8:  LDAX    D
        ADCX    H-
        STAX    D-
;
        DCR     C
        GJMP    BFDX8
        GJMP    BFDX6   ; JUMP TO / JUDGE DIV END
        ;
        EJECT
;
; ************************************************
; *                                              *
; *       COS (X) FUNCTION  SUBROUTINE           *
; *           INPUT:                             *
; *                   V REG  --  OFFH            *
; *           OUTPUT: RETS -- NORMAL    CY=0     *
; *                   RET  -- OVERFLOW, CY=1     *
; *                                              *
; ************************************************
;
COS:
        CALL    BFABS
        CALL    STORE1  ; WXE <- B.F.P.ACC
        LXI     H,BF2PDV        ; PI/2 STORE TOP ADDR
        CALL    BFADD   ; X + PI/2
        GJMP    COS1    ; IF OVERFLOW
        GJMP    SIN
        ;
```

```
COS1:   CALL    LOADA1  ; B.F.P.ACC <- WXE
        LXI     H,BF2PDV
        CALL    BFSUB   ; X - PI/2
        NOP             ; DUMMY FOR RETS
        CALL    BFCOMP  ; -(X-PI/2)
        EJECT
;
; ************************************************
; *                                              *
; *       SIN (X) FUNCTION  SUBROUTINE           *
; *           INPUT:                             *
; *                   V REG -- OFFH              *
; *                                              *
; *           OUTPUT:  RETS --- NORMAL    CY=0   *
; *                    RET  --- OVERFLOW, CY=1   *
; *                                              *
; ************************************************
;
SIN:    CALL    BFSIGN  ; CHECK SIGN
        EQI     A,OFFH
        GJMP    SINO
        ;
        CALL    BFCOMP  ; X <- (-X)
        LXI     H,BFCOMX-1
        PUSH    H
                        ; SIN (-X) = -SIN (X)
SINO:   LXI     H,BF2PI ; 2PI STORE TOP ADDR
        CALL    BFDIV   ; X/(2*PI)
        NOP
        CALL    PUSHF
        CALL    BFINT
                        ; INTEGER(X/(2*PI))
        GJMP    SINOVR  ; IF OVERFLOW
        ;
        CALL    POPR
        CALL    BFSUB1  ; INT(X/(2*PI)) - X/(2*PI)
        NOP
        LXI     H,BF4DV ; 1/4 STORE TOP ADDR
        CALL    BFADD   ; 1/4 - X
        NOP             ; DUMMY FOR RETS
        CALL    BFSIGN
        MVIW    CSIN MOD 100H,OFFH
                        ; SIGN INVERT / JUDGE FLAG
        EQI     A,OFFH
        GJMP    SIN1    ; IF X > 1/4
        ;
        LXI     H,BF2DV ; 1/2 STORE TOP ADDR
        CALL    BFADD   ; X <- X + 1/2
        NOP
        CALL    BFSIGN
        MVIW    CSIN MOD 100H,0
        ;
        ;   CSIN=0 : 0 < X < 1/4
        ;   CSIN=OFFH : X > 1/4
        ;
```

```
SIN1:   EQI     A,0FFH
        CALL    BFCOMP
        ;
        LXI     H,BF4DV
        CALL    BFADD   ; X <- X + 1/4
        NOP
        ;
        NEIW    CSIN MOD 100H,0
        CALL    BFCOMP
        ;
        CALL    STOREO  ; OPE <- X (B.F.P.ACC)
        GJMP    POLYN1
;
;  SIN  FUNCTION  OVERFLOW PROCESS
;
SINOVR: CALL    POPR
        NEIW    FSN MOD 100H,0FFH
        POP     H
        GJMP    BFRET
        ;
        EJECT
;
; ************************************************
; *                                              *
; *     TAN (X)  FUNCTION  SUBROUTINE            *
; *                                              *
; *          TAN(X) = SIN(X) / COS(X)            *
; *                                              *
; *           INPUT:                             *
; *                   V REG :  0FFH              *
; *                                              *
; *           OUTPUT : RETS    NORMAL  CY=0      *
; *                    RET     OVERFLOW CY=1     *
; *                                              *
; ************************************************
;
TAN:    CALL    PUSHF   ; STACK <- X(B.F.P.ACC)
        CALL    COS     ; COS (X)
        GJMP    TANOVR  ; IF OVERFLOW
        ;
        CALL    STORE1  ; WXE <- COS (X)
        CALL    POPR    ; B.F.P.ACC <- X(STACK)
        CALL    LOADAO
        CALL    LOADO1  ; STACK <- OPE(COS)
        CALL    PUSHR
        ;
        CALL    SIN     ; SIN (X)
        GJMP    TANOVR  ; IF OVERFLOW
        ;
        GJMP    BFDIVO  ; TAN <- SIN/COS
        ;
        ;  OVER FLOW PROCESS
        ;
TANOVR: CALL    POPR
```

```
        GJMP    BFRET
        EJECT
;
; ************************************************
; *                                              *
; *    LN (X) FUNCTION  SUBROUTINE               *
; *       INPUT:                                 *
; *                  V REG  --  OFFH             *
; *                                              *
; *       OUTPUT:    RETS - NORMAL    CY=0       *
; *                  RET -  OVERFLOW  CY=1       *
; *                                              *
; ************************************************
;
LNX:    CALL    BFSIGN  ; CHECK X.SIGN
        EQI     A,1
                        ; SKIP, IF X > 0
        GJMP    BFRET
        ;
        LDAW    ACCE MOD 100H
        SUI     A,80H
        STAW    LNWORK MOD 100H
        ;
        ; GET / POLYNOMIAL
        ;
        MVIW    ACCE MOD 100H,80H
                        ; EXPONENT CORRECT !
        LXI     H,BFN1
        CALL    BFADD   ; X + 1
        NOP             ; DUMMY FOR RETS
        CALL    STOREO  ; OPE <- (X+1)
        ;
        LXI     H,BFN1
        CALL    LOADA   ; B.F.P.ACC <- 1
        CALL    BFDIV1  ;  1 / (X + 1)
        NOP             ; DUMMY FOR RETS
        ;
        INRW    ACCE MOD 100H
        NOP             ;2 / (X+2)
        ;
        LXI     H,BFN1
        CALL    BFSUB   ; 2/(X+1) - 1
        NOP
        CALL    BFCOMP  ; SIGN INVERT
        CALL    STOREO  ; OPE <- B.F.P.ACC
;
;   POLY / X*(1 + X**2/3 + X**4/5 + -----)
;
        CALL    POLYN3
        NOP
        INRW    ACCE MOD 100H
        NOP
        CALL    PUSHF
                        ; POLYNOMIAL STORE
;
```

```
;  K(COUNTER)*LN2 + POLY (X) / IXI<1
;
        LXI     D,WSE
        MVI     A,8         ; FRAC POINT ADDR
        STAX    D+
                            ; WS1 ADDR
        LDAW    LNWORK MOD 100H
        OFFI    A,80H       ; EXPONENT < 0
        NEGA                ; IF < 0
        STAX    D+
                            ; WS2 ADDR
        MVI     A,0         ; WS2,3,4 = 0
        STAX    D+
        STAX    D+
        STAX    D+
        ;
        CALL    BFLT        ; K -> B.F.P.ACC
        NOP                 ; DUMMY FOR RETS
        ;
        LXI     H,BFLN2
        CALL    BFMUL       ; K * LN2
        NOP                 ; DUMMY FOR RETS

        OFFIW   LNWORK MOD 100H,80H
                            ; SUB,ADD ?
        CALL    BFCOMP      ; IF SUB  K*LN2
        GJMP    BFADDO      ; K*LN2 + POLYNOM
        ;
        EJECT
;
; ***********************************************
; *                                             *
; *     LOG (X) FUNCTION SUBROUTINE             *
; *                                             *
; *             LOG (X) =LN (X) / LN (10)       *
; *                                             *
; *        INPUT:                               *
; *                V REG. =  OFFH               *
; *                                             *
; *        OUTPUT:                              *
; *                RETS    NORMAL  CY=0         *
; *                RET     OVERFLOW CY=1        *
; *                                             *
; ***********************************************
;
LOG:    CALL    LNX
        GJMP    BFRET       ; IF OVERFLOW
        ;
        LXI     H,BFLN10
        GJMP    BFDIV
                ; LN(X) / LN(10)
        EJECT
;
; ***********************************************
; *                                             *
```

```
;  *          EXP (X)    FUNCTION  SUBROUTINE          *
;  *                                                   *
;  *              INPUT:                               *
;  *                       V REG. = OFFH               *
;  *                                                   *
;  *              OUTPUT:                              *
;  *                       RETS   NORMAL  CY=0         *
;  *                       RET    OVERFLOW CY=1        *
;  *                                                   *
;  *****************************************************
;
EXP:
        CALL    BFSIGN  ; CHECK X.SIGN
        PUSH    V       ; STORE X.SIGN
                        ; EXP(-X) = 1  / EXP(X)
        CALL    BFABS   ; X <- -X
        CALL    PUSHF   ; X STORE
        ;
        LXI     H,BFLGE2
                        ; LOG(E) BASE 2 TABLE TOP ADDR
        CALL    BFMUL   ; X * LOG(E)
        GJMP    EXP1    ; IF OVERFLOW
        ;
        LTIW    ACCE MOD 100H,80H+8
        GJMP    EXP1
                        ; IF OVERFLOW / > 2**127
        CALL    BFINTO  ; INT( X * LOG(E))
        GJMP    EXP1
                        ; IF NOT FIXED <- FLOAT
        LDAW    TMPWSL MOD 100H
        PUSH    V       ; STORE  INT(X * LOG(E))
        ;
        ADI     A,80H   ; ADD BIAS
        ADINC   A,80H-126
        GJMP    EXPO    ; IF OVERFLOW / EXP > 126
        ;
        LXI     H,BFLN2
        CALL    BFMUL   ; LN2 * INT(X*LOG(E))
        NOP             ; DUMMY FOR RETS
        ;
        POP     V
        CALL    POPR    ; RESTORE.X
        PUSH    V
        CALL    BFSUB1  ; LN2*(INT(X*LOG(E))) - X
        NOP
        CALL    STOREO  ; OPE <- B.F.P.ACC
        ;
        ; POLY / 1 + X/1 + X**2/2*1 + X**3/3*2*1 + ---
        ;
        LXI     H,EXPCON        ; EXP CONSTANT TOP ADDR
        CALL    POLY
        ;
        ; * POLYNOMIAL END *
        ;
        POP     V       ; RESTORE INT(X*LOG(E))
```

```
        ADDW    ACCE MOD 100H
        SKN     CY
        GJMP    EXP2    ; IF OVERFLOW
        ;
        STAW    ACCE MOD 100H
                        ; STORE B.F.P.ACC-EXP
        POP     V       ; X.SIGN RESTORE
        INR     A
        GJMP    BFRETS
        ;
        ;  EXP(-X) = 1 / EXP(X)
        ;
        CALL    STOREO  ; OPE <- B.F.P.ACC
        LXI     H,BFN1
        CALL    LOADA   ; B.F.P.ACC
        CALL    BFDIV1  ; 1 / EXP(X)
        GJMP    BFRET   ; IF OVERFLOW
        GJMP    BFRETS  ; IF NORMAL
        ;
        ;  OVER FLOW  RROCESS
        ;
EXPO:   POP     V
EXP1:   CALL    POPR
EXP2:   POP     V
        INR     A
        GJMP    BFRET
        ;
        ;  IF X < O , EXP = O
        ;
        ANIW    ACCE MOD 100H,0
        GJMP    BFRETS
        ;
        EJECT
;
; ***********************************************
; *                                             *
; *      POWER  FUNCTION  SUBROUTINE            *
; *                                             *
; *      INPUT:                                 *
; *               VREG. = OFFH                  *
; *               X = B.F.P.ACC                 *
; *               Y = OPERAND                   *
; *                                             *
; *      OUTPUT: RETS    NORMAL     CY = 0      *
; *              RET     OVERFLOW , CY = 1      *
; *                                             *
; ***********************************************
        ;
        ;  SQR --- OPERAND <- 1/2
        ;
SQR:    LXI     H,BF2DV
POWER:  CALL    LOADO3  ; OPE <- (HL,HL+1,HL+2,HL+3)
        ;
        ;  B.F.P.ACC <---> OPERAND
        ;
```

```
POWER1: CALL    STORE1  ; WXE <- B.F.P.ACC
        CALL    LOADA0  ; B.F.P.ACC <- OPE
        CALL    LOADO1  ; OPE <- WXE
;
        CALL    BFSIGN  ; CHECK ! Y.SIGN
        EQIW    WSE MOD 100H,0
        GJMP    POWER2
                        ; IF X= NOT.0
        NEI     A,0
        GJMP    EXP
                        ; IF X=Y=0 , X**Y = 1
        INR     A
        GJMP    POWERZ  ; IF X=0,Y>0
        GJMP    BFRET
                        ; IF X=0,Y<0
POWER2: NEI     A,0
        GJMP    EXP
                        ; IF X= NOT.0 ,Y=0
        LDAW    WS1 MOD 100H
        RLL     A
                        ; CY = X.SIGN
        CALL    PUSHR
        CALL    STORE1  ; WXE <- Y(B.F.P.ACC)
        CALL    BFABS
        CALL    STORE0
        ;
        MVI     A,2
        STAW    TMPWSL MOD 100H
                        ; INITIAL SSET
        SK      CY
                        ; CHECK X.SIGN
        GJMP    POWER3
        ;
        CALL    PUSHR
        CALL    BFINT   ; INTEGER  (Y)
        GJMP    POWER5  ; IF OVERFLOW
        ;
        CALL    POPR
        CALL    BFSUB1  ; INTEGER (Y) -Y
        NOP             ; DUMMY
        CALL    BFSIGN  ; IF INT(Y) = Y ,A =0
                        ; SIGN  RESTORE
;
;   CHECK ! / INT (Y) = ODD
;
POWER3: CALL    POPR
        PUSH    V       ; SIGN --> STACK
        CALL    LOADA0
        ;
        LDAW    TMPWSL MOD 100H
        STAW    POWERK MOD 100H  ; STORE "ODD" JUDGE AREA
        POP     V       ; SIGN RESTORE
;
;   X ** Y = EXP ( Y * LN(X) )
;
```

```
POWER4:
        NEI     A,0
        CALL    BFCOMP
        ;
        CALL    LOADO1  ; OPE <- Y(WXE)
        CALL    PUSHR
        CALL    LNX     ; LN (X)
        GJMP    POWER6  ; IF OVERFLOW
        ;
        CALL    POPR
        CALL    BFMUL1  ; Y * LN (X)
        GJMP    BFRET   ; IF OVERFLOW
        ;
        CALL    EXP     ; EXPONENT (Y*LN(X))
        GJMP    BFRET   ; IF OVERFLOW
        ;
        OFFIW   POWERK MOD 100H,00000001B
        CALL    BFCOMP  ; INVERT SIGN
        GJMP    BFRETS
;
;  OVER FLOW  PROCESS
;
POWER5: CALL    POPR
        CALL    POPR
        GJMP    BFRET
        ;
POWER6: CALL    POPR
        GJMP    BFRET
;
;   RESULT = 0 / PROCESS
;
POWERZ: ANIW    ACCE MOD 100H,0
        GJMP    BFRETS
        EJECT
;
; ************************************************
; *                                              *
; *        ARCTAN (X)  FUNCTION  SUBROUTINE      *
; *                                              *
; *         INPUT:                               *
; *                 V REG. = OFFH                *
; *                                              *
; *         OUTPUT:                              *
; *                 RETS:    NORMAL   CY=0       *
; *                 RET:    OVERFLOW  CY=1       *
; *                                              *
; ************************************************
;
ARCTAN:
        CALL    BFSIGN  ; CHECK XC.SIGN
        EQI     A,OFFH  ; IF X>0, X=0
        GJMP    ATAN1   ; IF X>0,X=0
                        ; ARCTAN(-X) = -ARCTAN(X)
        LXI     H,BFCOMX-1
        PUSH    H
```

```
        CALL    BFCOMP
        ;
ATAN1:  GTIW    ACCE MOD 100H,80H
        GJMP    ATAN2   ; IF  IXI < 1
        ;
        CALL    STOREO
        LXI     H,BFN1
        CALL    LOADA   ; B.F.P.ACC <- 1
        ;
        CALL    BFDIV1  ; 1 / IXI
        NOP
        LXI     H,BFCOMX-1
        PUSH    H
        LXI     H,BFSUB1-1
        PUSH    H
                        ; PI/2 - ARCTAN(1/IXI)
;
;  POLY / CO - C1*X**3/3 + C2*X**5/5 - -----
;
ATAN2:  CALL    STOREO
        CALL    POLYN2
        NOP
        LXI     H,BF2PDV
        CALL    LOAD03
                        ; OPERAND <- PI/2
        GJMP    BFRETS
        EJECT
;
; ************************************************
; *                                              *
; *       ARCSIN (X)  FUNCTION  SUBROUTINE       *
; *                                              *
; *          INPUT:                              *
; *                  V REG. = OFFH               *
; *                                              *
; *          OUTPUT:                             *
; *                  RETS:   NORMAL   CY=0       *
; *                  RET:    OVERFLOW CY=1       *
; *                                              *
; ************************************************
;
ARCSIN:
        CALL    BFSIGN  ; CHECK X.SIGN
        ;
        STAW    FASIN MOD 100H
        EQI     A,0FFH
        GJMP    ASIN1
                        ; IF X < 0
        LXI     H,BFCOMX-1
        PUSH    H
        CALL    BFCOMP
        ;
ASIN1:  CALL    PUSHF   ; STACK,OPE <- X
        CALL    ROUTE
        GJMP    ASIN2
```

```
        ;
        CALL    STORE1
        CALL    POPR
        CALL    LOADA0
        CALL    LOADO1
        CALL    BFDIV1  ; X / (1-X**X)**1/2
        GJMP    ASIN3   ; IF OVERFLOW
        GJMP    ARCTAN
        ;
        ; ARCSIN (X) = ARCTAN (X / (1-X**X)**1/2 )
        ;
;
;  ARCSIN  OVERFLOW
;
ASIN2:  CALL    POPR
        NEIW    FASIN MOD 100H,0FFH
        ;
        POP     H
        GJMP    BFRET
;
;   IF X = 1 , ARCSIN = PI/2
;
ASIN3:  LXI     H,BF2PDV
        CALL    LOADA
        GJMP    BFRETS
        EJECT
;
; ***********************************************
; *                                             *
; *       ARCCOS (X)  FUNCTION  SUBROUTINE      *
; *                                             *
; *          INPUT:                             *
; *                 V REG. = 0FFH               *
; *                                             *
; *          OUTPUT:                            *
; *                   RETS:   NORMAL  CY=0      *
; *                   RET:   OVERFLOW CY=1      *
; *                                             *
; ***********************************************
;
;   ARCCOS = ARCTAN ( ((1-X**2)**1/2)/X)  X>0
;   ARCCOS = PI + ARCTAN                  X<0
;   IF  X = 0 , ARCCOS = PI/2
;
ARCCOS:
        CALL    BFSIGN  ; CHECK X.SIGN
        PUSH    V       ; SIGN STORE
        ;
        CALL    PUSHF   ; STACK,OPERAND <- B.F.P.ACC
        CALL    ROUTE   ; ( 1-X**2) ** 1/2
        GJMP    ACOS1   ; IF  OVERFLOW
        ;
        CALL    POPR    ; OPERAND <- X
        CALL    BFDIV1  ; ( (1-X**2)**1/2 ) / X
        GJMP    ACOS2   ; IF OVERFLOW
```

```
        ;
        CALL    ARCTAN  ; ARCTAN( ((1-X**2)**1/2)/X )
        NOP             ; DUMMY FOR RETS
        ;
        POP     V       ; RESTORE X.SIGN
        INR     A
        GJMP    BFRETS  ; IF X > 0
        ;
        LXI     H,BFPI
        CALL    BFADD   ; PI + ARCTAN
        NOP             ; DUMMY FOR RETS
        GJMP    BFRETS
;
;  OVERFLOW  PROCESS
;
ACOS1:
        CALL    POPR
        POP     V
        GJMP    BFRET
;
;  IF X = 0 , ARCCOS = PI/2
;
ACOS2:
        POP     V
        LXI     H,BF2PDV
        CALL    LOADA   ; B.F.P.ACC <- PI/2
        GJMP    BFRETS
;
;  GET / ( 1 - X*X ) ** 1/2
;
ROUTE:  CALL    BFMUL1  ; X * X
        RET             ; IF OVERFLOW
        ;
        LXI     H,BFN1
        CALL    BFSUB   ; X * X - 1
        RET             ; IF  OVERFLOW
        ;
        CALL    BFCOMP
        ;
        CALL    SQR     ; ( 1 - X*X ) ** 1/2
        RET
        RETS
        ;
        EJECT
;
;  ***********************************************
;  *                                             *
;  *       POLAR ->  ORTHOGONAL -- COORDINATES   *
;  *                                             *
;  *                    X = R * COS(T)           *
;  *                    Y = R * SIN(T)           *
;  *         INPUT:                              *
;  *                 V REG. = OFFH               *
;  *                 BFP.ACC = R                 *
;  *                 HL = T START ADDR.          *
```

```
;  *                                                    *
;  *         OUTPUT:                                    *
;  *                  BFP.ACC = X                       *
;  *                  OPERAND = Y                       *
;  *                                                    *
;  *                   RETS    NORMAL    CY=0           *
;  *                   RET     OVERFLOW  CY=1           *
;  *                                                    *
;  ******************************************************
;
TRACA:  CALL    LOAD03
TRACA1: CALL    STORE2
                        ; WYE <- R
        CALL    LOADA0  ; B.F.P.ACC <- T
        CALL    STORE1  ; WXE <- T
        ;
        CALL    SIN     ; SIN (T)
        GJMP    BFRET   ; OVERFLOW
        ;
        LXI     H,WYE
        CALL    BFMUL   ; R * T
        NOP             ; DUMMY
        CALL    PUSHF
        CALL    LOADA1
        CALL    COS     ; B.F.P.ACC <- COS (T)
        GJMP    TRACA2
        ;
        LXI     H,WYE
        CALL    BFMUL   ; R * SIN (T)
        NOP
        CALL    POPR    ; OPE <- R * SIN(T)
        GJMP    BFRETS
;
;  OVER FLOW  PROCESS
;
TRACA2: CALL    POPR
        GJMP    BFRET
        EJECT
;
;  ******************************************************
;  *                                                    *
;  *       ORTHOGONAL -> POLAR  --  COORDINATES         *
;  *                                                    *
;  *              R = ( X*X + Y*Y ) **1/2               *
;  *              T = ARCTAN (Y/X)                      *
;  *                                                    *
;  *       INPUT:                                       *
;  *              V REG. = OFFH                         *
;  *              BFP.ACC = X                           *
;  *              HL = Y START ADDR.                    *
;  *                                                    *
;  *       OUTPUT:                                      *
;  *              BFP.ACC = R                           *
;  *              OPERAND = T                           *
;  *                                                    *
```

```
;  *                RETS:     NORMAL   CY=0                *
;  *                RET:    OVERFLOW   CY=1                *
;  *                                                       *
;  *******************************************************
;
;   T = ARCTAN (Y/X)
;
TRACB:   CALL     LOAD03   ;OPE <-- Y
;
;   B.F.P.ACC   <-->   OPERAND
;
TRACB1:  CALL     STORE1   ;SAVE X TO WXE
         CALL     LOADA0   ;GET Y TO BFP.ACC
         CALL     STORE2   ;SAVE IT TO WYE
         ;
         CALL     LOAD01
         ;
         CALL     BFSIGN   ; Y.SIGN CHECK
         PUSH     V        ; SIGN RESTORE
         ;
         LDAW     WS1 MOD 100H
         PUSH     V        ; MSB = SIGN
         ;
         CALL     BFDIV1   ; Y / X
         GJMP     TRACB2
                           ; IF Y/X TOO LAGE , ARCTAN <- PI/2
         CALL     ARCTAN   ; ARCTAN (Y/X)
         NOP
         ;
         POP      V
         ADI      A,80H    ; CY = X.SIGN
         POP      V
                           ; STORE Y.SIGN
         SK       CY
         GJMP     TRACB3
                           ; IF 1TH,OR,4TH QUADRANT
;
         INR      A
         LXI      H,BFPI   ;PI
         LXI      H,BFMPI  ;-PI
;
         CALL     BFADD
         NOP               ; DUMMY FOR RETS
         GJMP     TRACB3
         ;
TRACB3:  CALL     PUSHF
;
;   R = ( X*X + Y*Y ) ** 1/2
;
         CALL     LOADA1
         LXI      H,WXE
         CALL     BFMUL    ;  X * X
         GJMP     TRACB5   ; OVERFLOW
         ;
         CALL     PUSHF    ; STACN <-  X*X
```

```
        CALL    LOADA2
        LXI     H,WYE
        CALL    BFMUL   ; Y*Y
        GJMP    TRACB4  ; IF OVERFLOW
        ;
        CALL    POPR    ; OPE <- X*X(STACK)
        CALL    BFADD1  ; X*X + Y*Y
        GJMP    TRACB5  ; OVERFLOW
        ;
        CALL    SQR
        GJMP    TRACB5  ; ( X*X + Y*Y ) ** 1/2
        ;
        CALL    POPR    ; OPE <- T
        RETS
;
; Y / X  --  TOO LAGE , ARCTAN (Y/X)  =  PI/2
;
TRACB2: POP     V
        LXI     H,BF2PDV
        CALL    LOADA
                        ; B.F.P.ACC <-  PI/2
        POP     V
        OFFI    A,80H
                        ; SKIP , IF Y > 0
        CALL    BFCOMP
                        ; IF Y<0 , ARCTAN = - PI/2
        GJMP    TRACB3
;
; OVFRFLOE  PROCESS
;
TRACB4: CALL    POPR
TRACB5: CALL    POPR
        GJMP    BFRET
        EJECT
;
; ********************************
; *   DATA  TABLE                *
; ********************************
;
BFTEN:
        DB      84H,20H,0,0
BFDV58:                 ;0.00000005
        DB      68H,56H,0BFH,0ADH
BFNO1:
        DB      7DH,4CH,0CCH,0CCH       ;0.1
BFN1:
        DB      81H,00H,00H,00H         ;1
BF2DV:
        DB      80H,00H,00H,00H       ; 1 / 2
BF4DV:
        DB      7FH,00H,00H,00H       ; 1 / 4
BFPI:
        DB      82H,49H,0FH,0DBH      ; PI
BFMPI:
        DB      82H,0C9H,0FH,0DBH       ;-PI
```

```
BF2PI:
        DB      83H,49H,0FH,0DBH        ; 2PI
BF2PDV:
        DB      81H,49H,0FH,0DBH        ;  PI / 2
BFLGE2:
        DB      81H,38H,0AAH,3BH        ; LOG (E) BASE 2
BFLN2:
        DB      80H,31H,72H,18H         ; LN2
BFLN10:
        DB      82H,13H,5DH,8EH         ; LN (10)
;
; ******************************
; *    CONST  TABLE            *
; ******************************
;
SINCOS: DB      5-1
;
        DB      86H,1EH,0D7H,0BAH
        DB      87H,99H,26H,64H
        DB      87H,23H,34H,58H
        DB      86H,0A5H,5DH,0E0H
        DB      83H,49H,0FH,0DAH
;
EXPCON: DB      8-1
;
        DB      74H,94H,2EH,40H
        DB      77H,2EH,4FH,70H
        DB      7AH,88H,02H,6EH
        DB      7CH,2AH,0A0H,0E6H
        DB      7EH,0AAH,0AAH,50H
        DB      7FH,7FH,0FFH,0FFH
        DB      81H,80H,00H,00H
        DB      81H,00H,00H,04H
;
ATNCON: DB      9-1
;
        DB      78H,3BH,0D7H,4AH
        DB      7BH,84H,6EH,02H
        DB      7CH,2FH,0C1H,0FEH
        DB      7DH,9AH,31H,74H
        DB      7DH,5AH,3BH,84H
        DB      7EH,91H,7FH,0C8H
        DB      7EH,4CH,0BBH,0E4H
        DB      7FH,0AAH,0AAH,6CH
        DB      81H,00H,00H,00H
;
LNCON:  DB      9-1
;
        DB      78H,3BH,0D7H,4AH
        DB      7BH,04H,6EH,02H
        DB      7CH,2FH,0C1H,0FEH
        DB      7DH,1AH,31H,74H
        DB      7DH,5AH,3BH,84H
        DB      7EH,11H,7FH,0C8H
        DB      7EH,4CH,0BBH,0E4H
```

```
        DB      7FH,2AH,0AAH,6CH
        DB      81H,00H,00H,00H
;
        EJECT
;
;
;  ================================================
;  =                                              =
;  =              COMMON  SUBROUTINE              =
;  =                                              =
;  ================================================
;
;  ================================================
;  =                                              =
;  =          POLYNOMIAL  SUBROUTINE              =
;  =                                              =
;  ================================================
;
POLYN1: LXI     H,SINCOS        ; SIN CONST TOP ADDR
POLYN2: LXI     H,ATNCON        ; ARCTAN
POLYN3: LXI     H,LNCON         ; LN
        ;
POLYX:  CALL    PUSHR
        LXI     D,BFMULO
        PUSH    D
        ;
        PUSH    H
        CALL    LOADAO
        ;
        CALL    BFMUL1  ; X * X
        NOP
        CALL    STOREO
        POP     H
;
;  POLYNOMIAL / C0 + C1*X + C2*X**2 + -----
;
POLY:   CALL    PUSHR
        LDAX    H+
        PUSH    V
        CALL    LOADA   ; B.F.P.ACC <- CONST
        ;
POLY1:  POP     V
        CALL    POPR
        SUINB   A,1
        RET
        ;
        CALL    PUSHR
        PUSH    V
        PUSH    H
        ;
        CALL    BFMUL1  ; CONST * X
        NOP
        POP     H
        CALL    LOADO3  ; OPE <- NEXT CONST
        PUSH    H
```

```
        CALL    BFADD1  ; NEXT CONST + CONST * X(OPE)
        NOP
        POP     H
        GJMP    POLY1
;
;
;     ++  (WS1,2,3,4)  NORMALIZE   ++
BFNOR:
        NEIW    ACCE MOD 100H , 0H
        RET                     ;IF BFP ACC = 0
BFNORO:
        LDAW    WS1 MOD 100H
        OFFI    A,10000000B
        RETS
        ;
        ;  WS1,2,3,4  LEFT SHIFT
        ;
        CLC
        LXI     D,WS4
        MVI     C,4-1
BFNOR1: LDAX    D
        RAL
        STAX    D-
        DCR     C
        GJMP    BFNOR1
        ;
        DCRW    ACCE MOD 100H
        SK      Z
        GJMP    BFNORO
        RET
;
;     ++  (WS1,2,3,4)  ROUNDING  ++
;
BFROND: BIT     7,WS4 MOD 100H
        GJMP    BFRND1  ;  IF  WS4.MSB = 1
        ;
        CALL    BFRNDR
        RET
                        ; OVERFLOW
BFRND1: LXI     H,ACC1
        CALL    BFSTR2
                        ; B.F.P.ACC <-  WS1,2,3
        RETS
;
;     --  (WS1,2,3,4)  END-AROUND CARRY  --
;
BFRNDR: INRW    WS3 MOD 100H
        RETS
        ;
        INRW    WS2 MOD 100H
        RETS
        ;
        INRW    WS1 MOD 100H
        RETS
        ;
```

```
        MVIW    WS1 MOD 100H,80H
        INRW    ACCE MOD 100H
        RETS
        RET
;
;    ++  WS1,2,3,4  COMPLEMENT  ++
;
BFCOM1: CLC
        MVI     C,3
        LXI     D,WS4
BFCOM2: MVI     A,0
        SBBX    D
        STAX    D-
        DCR     C
        GJMP    BFCOM2
;
;    ++  SIGN (ACCS) INVERT  ++
;
        NOP
BFCOMP: LDAW    ACCS MOD 100H
        XRI     A,80H
        STAW    ACCS MOD 100H
        RET
;
;    ++  SIGN INVERT  SUBROUTINE  ++
;
        NOP
BFCOMX: CALL    BFCOMP
        GJMP    BFRETS
;
;    ++  CHECK, SIGN OF B.F.P.ACC  ++
;
BFSIGN: LDAW    ACCE MOD 100H
        STAW    FSN MOD 100H
        NEI     A,0
        RET
        ;
        BIT     7,ACCS MOD 100H
                        ; B.F.P.ACC < 0
        MVI     A,0FFH
        MVI     A,1
        STAW    FSN MOD 100H
        RET
;
;    ++  (HL,HL+1,HL+2,---,HL+4)  <-  OPERAND
;
BFSTR1: LDAW    WSE MOD 100H
        STAX    H+
        MOV     A,B
        STAX    H+
BFSTR2: LDAW    WS1 MOD 100H
        STAX    H+
        LDAW    WS2 MOD 100H
        STAX    H+
        LDAW    WS3 MOD 100H
```

```
          STAX      H+
          RET
;
;     ++   B.F.P.ACC   : ABSOLUTE   ++
;
BFABS:    MVI       A,80H
          STAW      ACCS MOD 100H
          RET
;
;     ++   (WS1,2,3,4)   RIGHT SHIFT   ++
;
BFRSHO:   MVI       A,8
BFRSH:    ANIW      WS4 MOD 100H,0
          MOV       B,A
                    ; B -- SHIFT COUNTER
          GJMP      BFRS11
;
BFRSH1:
          CALL      BFRSH2
BFRS11:   DCR       B
          GJMP      BFRSH1
          RET
          ;
BFRSH2:   CLC
BFRSH4:   LXI       D,WS1
          MVI       C,3
          ;
BFRSH3:   LDAX      D
          RLR       A
          STAX      D+
          DCR       C
          GJMP      BFRSH3
          RET
;
;     ++   (WS1,2,3,4)   LEFT   SHIFT   ++
;
          ; B -- SHIFT COUNTER
;
          LXI       D,WS4
BFLSH1:   LDAX      D
          RLL       A
          STAX      D-
          ;
          DCR       B
          GJMP      BFLSH1
          RET
;
;     ++   BINARY FLOATING   ->   FIXED   ++
;          (ACC1,2,3)              (WS1,2,3,4)
;
BFIXO:    MVI       A,8
BFIX2:    MVI       A,32
          ;
          LXI       D,WSE
          STAX      D+
```

```
                        ; DE= WS1 ADDR
BFIX:   LXI     H,ACCE
        NEIW    ACCE MOD 100H,0
        GJMP    BFIXED  ; IF B.F.P.ACC = 0
        ;
        ADI     A,80H-1 ; APPLY BIAS = 1  FOR SIGN
        SUBNBX  H+      ; DIFFERENCE OF EXP
        RET
        ;
        LTI     A,32-1
        GJMP    BFIXED  ; IF B.F.P.ACC  TOO SMALL
        ;
        ADI     A,1
        MOV     B,A     ; B - SHIFT COUNTER
                        ; DE =  WS1 ADDR
        LXI     H,ACC1
        LDAX    H+
        STAX    D+
        LDAX    H+
        STAX    D+
        LDAX    H
        STAX    D
        ;
        MOV     A,B
        CALL    BFRSH
        ;
        BIT     7,ACCS MOD 100H
                        ; IF < 0, COMPLEMENT
        CALL    BFCOM1
        GJMP    BFIXEN
        ;
BFIXED: MVI     A,0     ; WS1,2,3,4  OCLEAR
        LXI     H,WS1
        ;
        STAX    H+
        STAX    H+
        STAX    H+
        STAX    H+
        ;
BFIXEN: LDAW    WS4 MOD 100H
        STAW    TMPWSL MOD 100H
        RETS
;
;     ++  BINARY FIXED  ->  FLOATING  ++
;       (WS1,2,3,4)         (ACC1,2,3)
;
BFLT:   LXI     D,WSE
        LDAX    D+
                        ; DE= WS1
        XRI     A,80H
                        ; APPLY EXP BIAS
        LXI     H,ACCE
        STAX    H+
                        ; HL= ACCS ADDR
        MVI     A,80H
```

```
        STAX    H+
                        ; HL= ACC1 ADDR
        LDAX    D
        RLL     A               ; CY FROM MSB
        GJMP    BFAD11
                        ; JUMP TO NORMAL,ROUNDING
;
;    ++  B.F.P.ACC  INTEGER PART  ++
;
BFINTO:
        CALL    BFSIGN  ; GET X.SIGN
        INR     A
        GJMP    BFINT   ; NOT / X<0
        ;
        LXI     H,BFN1
        CALL    BFSUB   ; X-1
        RET             ; IF OVERFLOW
        ;
BFINT:  CALL    BFIX2
        RET             ; NOT DISPLAY
        CALL    BFLT
        NOP
        RETS
;
;    ++  EXPONENT  PART  OPERATION  ++
;                        MULTIPLY,DIVISION
;
BFRD:   LXI     D,WSE
        LXI     H,ACCE
        ;
        LDAX    H
        NEI     A,0
        GJMP    BFRETS  ; IF B.F.P.ACC = 0
        ;
        ADDX    D
        STAX    D
        RLR     A
        ;
        ANI     A,11000000B
        NEI     A,00000000B
        GJMP    BFRDZ   ; IF UNDER FLOW
        ;
        NEI     A,11000000B
        RET             ;IF OVER FLOW
        LDAX    D+
        XRI     A,80H
        STAX    H+
        ;
        SKN     Z
        RETS            ;IF ZERO
        ;
        LDAX    H       ;GET ACCS
        XRAX    D       ;ACCS EXR WS1
        ANI     A,80H
        STAX    H       ; RESULT SIGN
```

```
        LDAX    D
        ORI     A,80H   ; WS1 -- NON SIGNED
        STAX    D
        RETS
        ;
        ;   IF  UNDER FLOW
        ;
BFRDZ:  ANIW    ACCE MOD 100H,0
        RETS
;
;
;================================================
;=      STORE BFP. ACCUMULATOR SUBROUTINE       =
;================================================
;
STORE0: LXI     H,WSE
STORE1: LXI     H,WXE
STORE2: LXI     H,WYE
;
        PUSH    D       ;PUSH DE REG.
;
        LXI     D,ACCE
        LDAX    D+
        STAX    H+      ;STORE ACCE
        LDAX    D+
        XRAX    D+
        STAX    H+      ;STORE ACCS+ACC1
        LDAX    D+
        STAX    H+      ;STORE ACC2
        LDAX    D+
        STAX    H+      ;STORE ACC3
;
        POP     D       ;POP DE REG.
;
        RET
;
;       ;       ;       ;       ;       ;
;================================================
;=      LOAD BFP.ACCUMULATOR SUBROUTINE         =
;================================================
;
LOADA0: LXI     H,WSE
LOADA1: LXI     H,WXE
LOADA2: LXI     H,WYE
;
LOADA:  PUSH    D       ;PUSH DE REG.
;
        LXI     D,ACCE
;
        LDAX    H+
        STAX    D+      ;LOAD ACCE
        LDAX    H
        XRI     A,80H
        ANI     A,80H
        STAX    D+      ;LOAD ACCS
```

```
        LDAX    H+
        ORI     A,80H
        STAX    D+          ;LOAD ACC1
        LDAX    H+
        STAX    D+          ;LOAD ACC2
        LDAX    H+
        STAX    D+          ;LOAD ACC3
;
        POP     D           ;POP DE REG.
;
        RET
;
;       ;       ;       ;       ;       ;       ;
;=======================================================
;=      LOAD OPERAND SUBROUTINE                        =
;=======================================================
;
LOADO1: LXI     H,WXE
LOADO2: LXI     H,WYE
;
LOADO3: PUSH    D           ;PUSH DE REG.
;
        LXI     D,WSE
;
        LDAX    H+
        STAX    D+          ;LOAD WSE
        LDAX    H+
        STAX    D+          ;LOAD WS1
        LDAX    H+
        STAX    D+          ;LOAD WS2
        LDAX    H+
        STAX    D+          ;LOAD WS3
;
        POP     D           ;POP DE REG.
;
        RET
;
;=================================================
;=      PUSH BFP.ACCUMULATOR                     =
;=================================================
;
PUSHF:
        CALL    STOREO
;
;       ;       ;       ;       ;       ;       ;
;=================================================
;=      PUSH OPERAND                             =
;=================================================
;
PUSHR:
        POP     B
;
        LDED    WSE
        PUSH    D
        LDED    WS2
```

```
        PUSH    D
;
        PUSH    B
;
        RET
;
;
;=================================================
;=      POP  OPERAND                             =
;=================================================
;
POPR:
        POP     B
;
        POP     D
        SDED    WS2
        POP     D
        SDED    WSE
;
        PUSH    B
        RET
;
;    ++  RET  PROCESS  ++
;
BFRET:  STC
        RET
;
;    ++  RETS  PROCESS  ++
;
BFRETS: EQIW    ACCE MOD 100H,0
        NOP
        RETS
;
;       ;       ;       ;       ;       ;       ;
;
;       FUNCTION  TABLE
;
FTABL:
        DB      'S','I','N'+K
        DW      SIN
;
        DB      'C','O','S'+K
        DW      COS
;
        DB      'T','A','N'+K
        DW      TAN
;
        DB      'L','N'+K
        DW      LNX
;
        DB      'L','O','G'+K
        DW      LOG
;
        DB      'E','X','P'+K
        DW      EXP
```

```
;
        DB      'A','R','C','T','A','N'+K
        DW      ARCTAN
;
        DB      'A','R','C','S','I','N'+K
        DW      ARCSIN
;
        DB      'A','R','C','C','O','S'+K
        DW      ARCCOS
;
        DB      'S','Q','R'+K
        DW      SQR
;
        DB      0       ;TABLE END
;
        STACK   EQU     0
        CTS     EQU     00001000B
        K       EQU     80H
;
        ORG     0FF00H
;
        ACCE:   DS      1
        ACCS:   DS      1
        ACC1:   DS      1
        ACC2:   DS      1
        ACC3:   DS      1
;
        SF:     DS      1
;
        WSE:    DS      1
        WS1:    DS      1
        WS2:    DS      1
        WS3:    DS      1
        WS4:    DS      1
;
;
        WXE:    DS      1
        WX1:    DS      1
        WX2:    DS      1
        WX3:    DS      1
;
        WYE:    DS      1
        WY1:    DS      1
        WY2:    DS      1
        WY3:    DS      1
;
        WD1:    DS      1
        WD2:    DS      1
        WD3:    DS      1
        WD4:    DS      1
        WD5:    DS      1
        WD6:    DS      1
;
        WA1:    DS      1
        WA2:    DS      1
```

```
        WA3:    DS      1
;
        POWERK: DS      1
        LNWORK: DS      1
        CMULT0: DS      1
        CMULT1: DS      1
        CSIN:   DS      1
;
        TMPWSL: DS      1
        FASIN:  DS      1
        FSN:    DS      1
;
        ADDRS:  DS      2
        TMP1:   DS      1
        TMP2:   DS      1
        TMP3:   DS      1
        TMP4:   DS      1
;
        OUTADR: DS      13
        CHARST: DS      80
;
        END
```

Book 7

μCOM-87 AD

EASY ARITHMETIC

SOFTWARE (PART 1)

APPLICATION NOTE

INTRODUCTION

The μCOM-87 family of microcomputers is based around NEC's original 8-bit single-chip microcomputer. The series contains a built-in 16-bit ALU having a large 16-bit arithmetic instruction set such as multiplication/division instructions to fully exploit the 16-bit ALU. This manual introduces the series' basic software functions such as fixed point format addition/subtraction/multiplication/division programs so that these powerful instructions can be utilized more effectively.

In addition, application notes (II) and (III) introduce floating point computation programs and hardware features, respectively.

Note: This application note applies to the following products:
μPD7811/7810/78PG11, μPD7811H/7810H/78PG11-15,
μPD78C14/78C11/78C10, μPD7809/7808/78P09

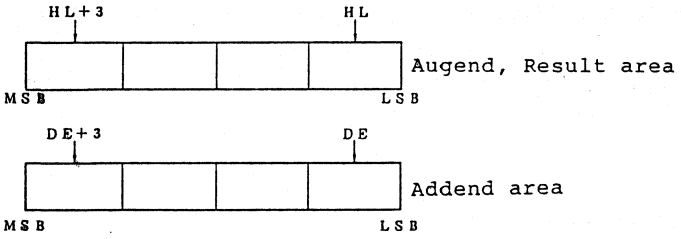
CHAPTER 1 ADDITION/SUBTRACTION/MULTIPLICATION/DIVISION

1.1 Binary Operation

1.1.1 Binary addition

32-bit + 32-bit → 32-bit

a) Memories used



b) Registers used

A, C, D, E, H, L

c) Input conditions

As shown in a) above, set as follows:

HL pair register ← The starting address of the area in which a 32-bit augend is stored

DE pair register ← The starting address of the area in which a 32-bit addend is stored.

d) Output conditions

RET ; The result of the operation overflows.

RETS ; The result of a normal operation is stored in the 4-byte area (HL, HL+1, ..., HL+3) as indicated in a).

• Program statement

```
;  ++++++++++++++++++++++++++++++++++++++++++++++
;  +                                            +
;  +          BINARY  ADDITION                  +
;  +            32BIT <- 32BIT + 32BIT          +
;  +                                            +
;  +        INPUT : HL <= AUGEND TOP ADDRS      +
;  +                DE <= ADDIT TOP ADDRS       +
;  +                                            +
;  +        OUTPUT : RET .. OVERFLOW            +
;  +                 RETS .. NORMAL             +
;  +                                            +
;  ++++++++++++++++++++++++++++++++++++++++++++++
;
BFADD:
        CLC                                     ①
        MVI     C,4-1                           ②
        ;
BFAD1:  LDAX    H                               ③
        ADCX    D+                              ④
        STAX    H+                              ⑤
        ;
        DCR     C                               ⑥
        GJMP    BFAD1                           ⑦
        ;
        SKN     CY          ; CHECK / OVERFLOW ?
        RET                 ; OVERFLOW          ⑧
        RETS                ; NORMAL            ⑨
```

(1) Clears the carry in advance.

(2) Sets the byte counter to (4-1).

(3) Loads 1 byte of augend in the accumulator.

(4) Adds 1 byte of addend to the contents of the accumulator and increments (+1) the addend address.

(5) Stores the operation result in the augend area and increments (+1) the augend address.

(6) Decrements (-1) the byte counter; this is skiped when the operation is completed.

(7) Loops until the end of the operations.

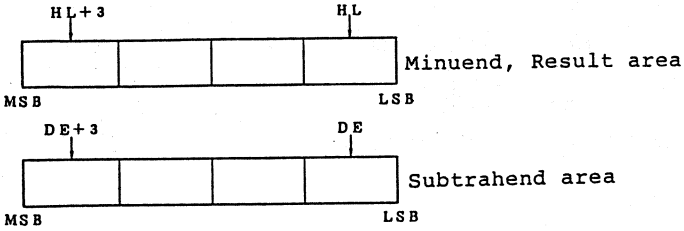
(8) The operation result overflows.

(9) The operation result is normal.

1.1.2 Binary subtraction

32-bit - 32-bit → 32-bit

a) Memories used



b) Registers used

A, C, D, E, H, L

c) Input conditions

As shown in a), set as follows:

HL pair register ← The starting address of the area in which a 32-bit minuend is stored

DE pair register ← The starting address of the area in which a 32-bit subtrahend is stored.

d) Output conditions

RET ; A borrow is generated because the minuend is smaller than the subtrahend.

RETS ; The result of a normal operation is stored in the 4-byte area (HL, HL+1, ..., HL+3) as indicated in a).

• Program statements

```
;  ++++++++++++++++++++++++++++++++++++++++++++++++
;  +                                              +
;  +          BINARY  SUBTRACTION                 +
;  +            32BIT <- 32BIT - 32BIT            +
;  +                                              +
;  +         INPUT : HL <= MINUEND TOP ADDRS      +
;  +                 DE <= SUBTRC TOP ADDRS       +
;  +                                              +
;  +         OUTPUT : RET .. OVERFLOW             +
;  +                  RETS .. NORMAL              +
;  +                                              +
;  ++++++++++++++++++++++++++++++++++++++++++++++++
;
BFSUB:
        CLC                                     ①
        MVI     C,4-1                           ②
        ;
BFSB1:  LDAX    H                               ③
        SBBX    D+                              ④
        STAX    H+                              ⑤
        ;
        DCR     C                               ⑥
        GJMP    BFSB1                           ⑦
        ;
        SKN     CY          ; CHECK / OVERFLOW ?
        RET                 ; BORROW            ⑧
        RETS                ; NORMAL            ⑨
```

(1) Clears the borrow in advance.

(2) Sets the byte counter to (4-1).

(3) Loads 1 byte of minuend in the accumulator.

(4) Subtracts 1 byte of subtrahend including the carry from the accumulator and increments (+1) the subtrahend address.

(5) Stores the operation result in the minuend area and increments (+1) the minuend address.

(6) Decrements (-1) the byte counter; this is skiped when the operation is completed.

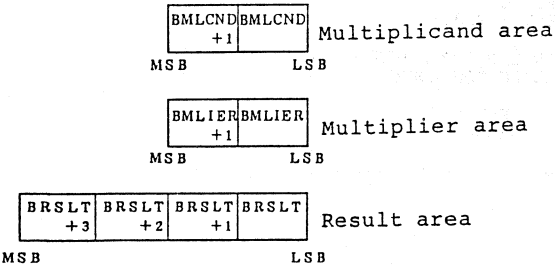
(7) Loops until the end of the operation.

(8) A borrow is generated in the operation result.

(9) The operation result is normal.

1.1.3 Binary multiplication
16-bit x 16-bit → 32-bit

a) Memories used



b) Registers used

A, B, C, D, E, H, L, EA

c) Input conditions

16-bit multiplicand and 16-bit multiplier are stored in the multiplicand are (BMLCND, BMLCND+1) and multiplier area (BMLIER, BMLIER+1) as shown in a).

d) Output conditions

RET ; The result of normal operation is stored in the result area (BRSLT, BRSLT+1, ..., BRSLT+3) as shown in a).

e) Processing procedure

Since the μCOM-87's unique multiplication instruction is used in this arithmetic operation, the 16-bit multiplier can be divided into an upper and lower 8 bits, and multiplication of 16 bits (multiplicand) x 8 bits (multiplier) is performed twice.

(1) Initialize by clearing the result area (BRSLT, BRSLT+1, BRSLT+2, BRSLT+3) to 0.

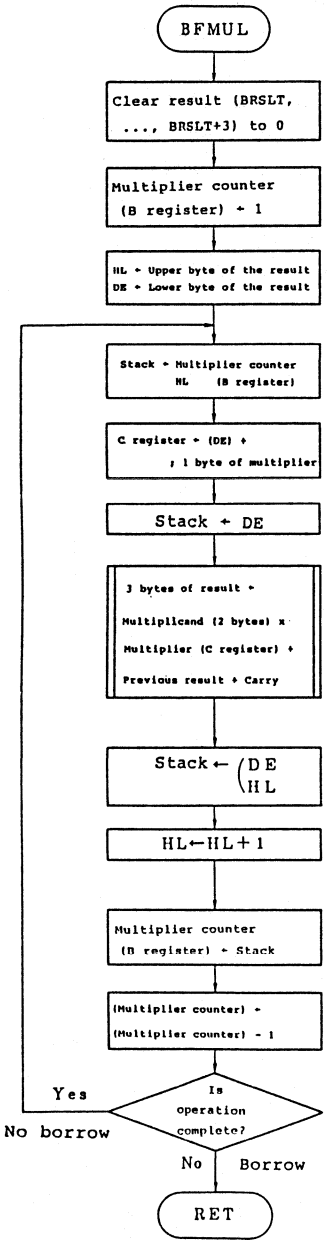
(2) Set the starting address (BRSLT) of the result area by using the HL register to store it to the stack in order to store the result of the operation in performed (3) in the result area.

(3) Perform the operation of the multiplicand (BMLCND, BMLCND+1) X 1-byte multiplier (BMLIER) + previous result (HL, HL+1, HL+2), and store the result in (HL, HL+1, HL+2).

(4) The multiplier becomes the upper digits (BMLIER+1) so that the starting address of the result area to which the result of (3) is stored can be recovered from the stack and incremented.

(5) Perform (3) and (4) again using 1 byte of multiplier as BMLIER. Store the result in (BRSLT, BRSLT+1, BRSLT+2, BRSLT+3).

Flow chart

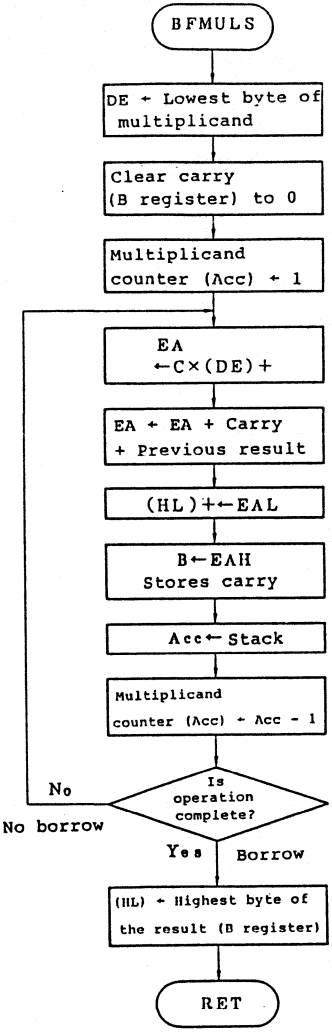


; Subroutine 'RSTCLR'

; Loads 1 byte of multiplier and increments the multiplier address using the DE register.

Subroutine 'BFMULS' (HL, HL+1, HL+2) ← 2 bytes of multiplicand x 1 byte of multiplier + Carry (B register) + Previous result (HL, HL+1, HL+2)

; The upper 1 digit is the multiplier; therefore, the address to which the result is stored is set to a 1 word higher.



; 1 byte x 1 byte

; EA←
EA+B+(HL)

; Stores 1 byte of the result

* Program statements

```
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +       BINARY  MULTIPLY                   +
;  +            32BIT <- 16BIT * 16BIT        +
;  +                                          +
;  +   MULTIPLICAND .. ( BMLCND+1,BMLCND )    +
;  +   MULTIPLIER   .. ( BMLIER+1,BMLIER )    +
;  +                                          +
;  +   RESULT ..( BRSLT+3,BRSLT+2,...,BRSLT )  +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
BFMUL:
;
;  **  RESULT  0CLEAR  **
;
        CALL    RSTCLR
;
;  **  MULTIPLICAND  COUNTER SET  **
;
        MVI     B,2-1
        LXI     H,BRSLT  ; RESULT TOP ADDRS
        LXI     D,BMLIER ; MULTIPLIER TOP ADDR
;
BFMUL1:
        PUSH    B
        PUSH    H
        LDAX    D+
        MOV     C,A      ;  MULTIPLIER SET
        PUSH    D
;
;  --  16BIT * 8BIT ->  24BIT --
;
        CALL    BFMULS
;
        POP     D
        POP     H
        INX     H
;
;  **  CHECK /  MULTIPLY END
;
        POP     B
        DCR     B
        GJMP    BFMUL1
        RET
```

* Program statements

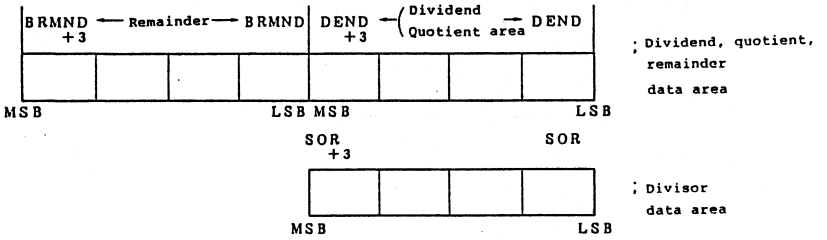
```
;SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;S                                 S
;S --  24BIT <- 16BIT * 8BIT --    S
;S                                 S
;SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BFMULS:
        LXI     D,BMLCND
        MVI     B,0
                ;  CY 0CLEAR
        MVI     A,2-1
;
;  --  8BIT * 8BIT + CY + RESULT
;
MULSS:
        PUSH    V
        LDAX    D+
        MUL     C
        EADD    EA,B
        LDAX    H
        EADD    EA,A
        MOV     A,EAL
        STAX    H+
        ;
        MOV     A,EAH
        MOV     B,A     ; CY STORE !
;
;  **  CHECK / 16BIT*8BIT  END ? '  **
;
        POP     V
        DCR     A
        GJMP    MULSS
        ;
        MOV     A,B
        STAX    H
        RET
```

```
;  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;  S                                                S
;  S            COMMON  SUBROUTINE                  S
;  S                                                S
;  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
;  **  0-CLEAR  PROCESS  **
;
HIMCLR: LXI     H,DMLCND+4
RMNDCL: LXI     H,RMIND
RSTCLR: LXI     H,BRSLT
DIVCLR: LXI     H,BRMND
        ;
        MVI     C,4-1
        ;
RCLR0:  MVI     A,0
RCLR1:  STAX    H+
        DCR     C
        GJMP    RCLR1
        RET
```

1.1.4 Binary division

32-bit ÷ 32-bit → Quotient 32-bit
Remainder 32-bit

a) Memories used



b) Registers used

A, B, C, D, E, H, L, EA

c) Input condition

A 32-bit dividend and a 32-bit divisor are stored in the areas indicated in a) as follows:

Dividend (DEND, ..., DEND+3)
Divisor (SOR, ..., SOR+3)

d) Output condition

RET ; When divisor = 0
RETS ; The result of a normal operation (quotient, remainer) is stored in the areas indicated in a) as follows:
Quotient (DEND, ..., DEND+3)
Remainder (BRMND, ..., BRMND+3)

e) Processing procedure

In arithmetic operation, a dividend (DEND, DEND+1, ..., DEND+3) and remainder (BRMND, BRMND+1, ..., BRMND+3) are placed in 8 consecutive bytes. The dividend and remainder are shifted 4 bits left and the highest 4 bits of the dividend are transferred to the lowest area of the remainder. The quotient is then

entered in the lowest area of the divisor, 4 bits at a time from the smallest value.

The quotient (remainder - divisor) of each digit is the number of repetitions required for a borrow to occur.

(1) Check if the divisor (SOR, SOR+1, ..., SOR+3) is 0; if 0, terminate the operation.

(2) Clear the remainder area (BRMND, ..., BRMND+3) to 0.

(3) Set the binary counter (B register) of the quotient to 7.

(4) Shift the 8-byte quotient/remainder 4 bits to the left.

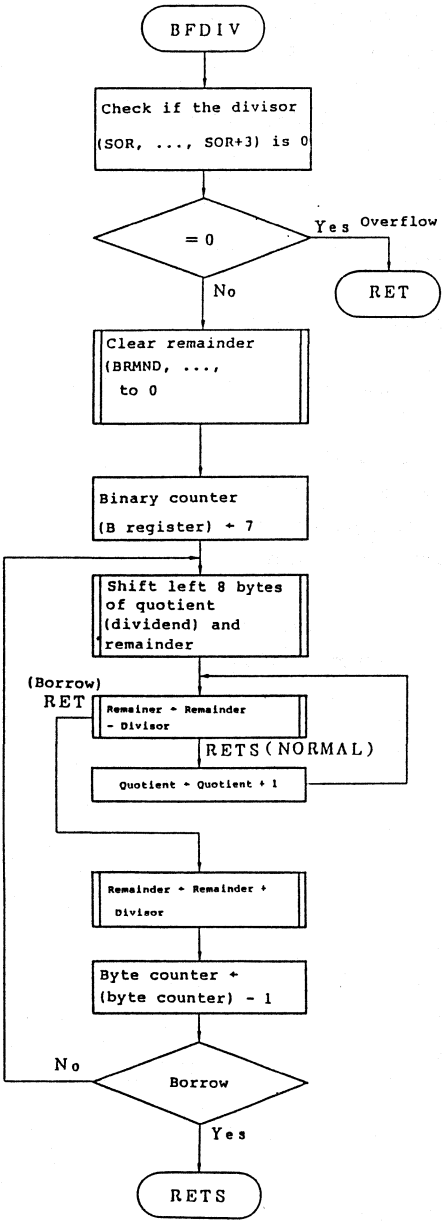
(5) Remainder ← Remainder - Divisor, jump to (7) if a borrow occurs.

(6) Increment (+1) the quotient. Jump to (5).

(7) The result of (5) is negative due to over-withdrawn. Therefore, perform the operation of Remainer ← Remainer + Divisor, in order to return to the previous state.

(8) Decrement the binary counter of the quotient and terminate the operation when a borrow occurs. If not, jump to (4).

Flow chart



; Subroutine 'DIVCLR'

; Subroutine 'BCDLS'
The highest 4 bits of the dividend are entered to the lowest 4 bits of the remainder.

; Subroutine 'BFSUB'

; Subroutine 'BFADD'
Over-withdrawn by 1 time; therefore, return to the previous state.

;Is the operation complete?

* Program statements

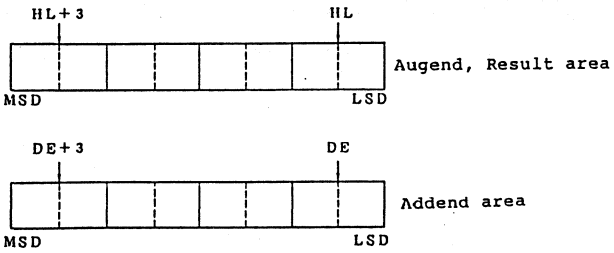
```
;  ++++++++++++++++++++++++++++++++++++++++++++++
;  +                                            +
;  +          BINARY  DIVISION                  +
;  +            32BIT <- 32BIT / 32BIT          +
;  +                                            +
;  +                                            +
;  +  DIVIDEND .. ( DEND+3,DEND+2,...,DEND )    +
;  +  SOR      .. ( SOR+3,SOR+2,...,SOR )       +
;  +  QUOTIENT .. ( DEND+3,DEND+2,...,DEND )    +
;  +  REMIND   .. ( BRMND+3,BRMND+2,..,BRMND )  +
;  +                                            +
;  ++++++++++++++++++++++++++++++++++++++++++++++
;
BFDIV:
;
;  **  CHECK FOR  / DIVISOR=0
;
        LXI     H,SOR
        LDEAX   H++
        DMOV    D,EA
        LDEAX   H
        DOR     EA,D
        SKN     Z
        RET             ; IF  OVERFLOW
;
;  **  QUOTIENT  0CLEAR  **
;
        CALL    DIVCLR  ; (See page 10)
;
;  **  BYTE-COUNTER  SET  **
;
        MVI     B,8-1
;
;  **  DIVIDEND,RMIND  1BYTE LEFT-SHIFT  **
;
BFDIV2:
        LXI     H,DEND
        MVI     C,8-1   ; SHIFT, BYTE-COUNTER SET
        CALL    BCDLS   ; SHIFT ! ; (See page 34)
;
;  **  SUBTRACT DIVISOR  FROM DIVIDEND  **
;
BFDIV3:
        LXI     H,BRMND
        LXI     D,SOR
        ;
        CALL    BFSUB  ; (See page 4)
        GJMP    BFDIV4          ; IF BORROW
                                ; IF NO BORROW
        INRW    DEND MOD 100H   ; 1-DIGIT QUOT (+1)
        GJMP    BFDIV3
;
;  **  IF BORROW , DIVISOR + DIVIDEND  **
;
BFDIV4:
        LXI     H,BRMND
        LXI     D,SOR
        CALL    BFADD  ; (See page 2)
        NOP
;
;  **  BIT COUNTER DECREMENT  **
;
        DCR     B       ; SKIP, IF END
        GJMP    BFDIV2
        ;
        RETS
```

1.2 Decimal Operation

1.2.1 Decimal addition

8-digit + 8-digit → 8-digit

a) Memories used



b) Registers used

A, C, D, E, H, L

c) Input conditions

As shown in a), set as follows:

HL Pair register ← The starting address of the area in which the 8-digit augend (4 bytes) is stored.

DE pair register ← The start address of the area in which the 8-digit addend (4 bytes) is stored.

d) Output conditions

RET ; The result of the operation overflows.

RETS ; The result of a normal operation is stored in the consecutive 4-byte area (HL, HL+1, ..., HL+3) as indicated in a).

· Program statements

```
;
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +          DECIMAL  ADDITION               +
;  +      8-PLACE <- 8-PLACE + 8-PLACE        +
;  +                                          +
;  +   AUGEND .. ( HL+3,...,HL )              +
;  +   ADDEND .. ( DE+3,...,DE )              +
;  +   RESULT .. ( HL+3,...,HL )              +
;  +                                          +
;  +    OUTPUT : RET .. OVERFLOW              +
;  +             RETS .. NORMAL               +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
BCDADD:
        MVI     C,4-1
        ;
BCDAD1: CLC                                     ①
BCDAD2: LDAX    H                               ②
        ADCX    D+                              ③
        DAA             ; DECIMAL ADJUST        ④
        STAX    H+                              ⑤
        ;
        DCR     C                               ⑥
        GJMP    BCDAD2                          ⑦
        ;
        SKN     CY      ; CHECK / OVERFLOW ?    ⑧
        RET             ; OVERFLOW              ⑨
        RETS            ; NORMAL                ⑩
```

(1) Clears the carry in advance.

(2) Loads 1 byte of augend in the accumulator.

(3) Adds 1 byte of addend in the accumulator and increments (+1) the addend address.

(4) Decimal-adjusts the result of the addition. Sets carry if a carry occurs.

(5) Stores the operation result to the augend area and increments (+1) the augend address.

(6) Decrements (-1) the digit counter; this is skiped when the operation is completed.

(7) Loops until the end of the operation.

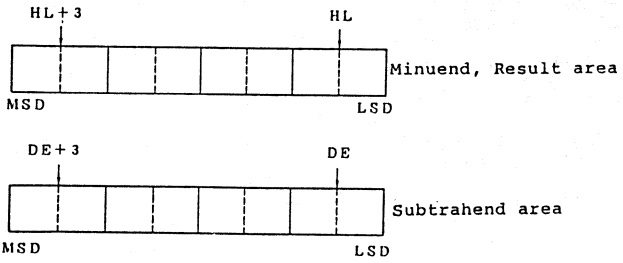
(8) The operation result overflows.

(9) The result of the operation is normal.

1.2.2 Decimal subtraction

8-digit - 8-digit → 8-digit

a) Memories used



b) Registers used

A, C, D, E, H, L

c) Input conditions

As shown in a), set as follows:

HL pair register ← The starting address of the area in which an 8-digit minuend (4 bytes) is stored.

DE pair register ← The starting address of the area in which an 8-digit subtrahend (4 bytes) is stored.

d) Output conditions

RET ; A borrow is generated because the minuend is smaller than the subtrahend.

RETS ; The result of a normal operation is stored in a consecutive 4-byte area (HL, HL+1, ..., HL+3) as indicated in a).

· Program statements

```
;
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +          DECIMAL  SUBTRACTION            +
;  +       8-PLACE <- 8-PLACE - 8-PLACE       +
;  +                                          +
;  +     MINUEND .. ( HL+3,..,HL )            +
;  +     SUBTRACTER .. ( DE+3,...,DE )        +
;  +     RESULT  .. ( HL+3,..,HL )            +
;  +                                          +
;  +      OUTPUT : RET .. OVERFLOW            +
;  +               RETS .. NORMAL             +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
BCDSUB:
        MVI     C,4-1
        ;
BCDSB1: STC                                        ①
BCDSB2: MVI     A,99H    ; FOR ADJUST              ②
        ACI     A,0
        SBBX    D+                                 ③
        ADDX    H                                  ④
        DAA              ; DECIMAL ADJUST          ⑤
        STAX    H+                                 ⑥
        ;
        DCR     C                                  ⑦
        GJMP    BCDSB2                             ⑧
        ;
        SK      CY    ; CHECK / OVERFLOW ?
        RET           ; BORROW                     ⑨
        RETS          ; NORMAL                     ⑩
```

(1) Sets carry in advance for the preparation of subtraction.

(2) Preparation for expressing the subtrahend in complement format.

(3) Converts the subtrahend to 100's complement.

(4) Adds the subtrahend's complement to the minuend (addition execution).

(5) Performs decimal adjustment.

(6) Stores the operation result in the minuend area and increments (+1) the minuend address.

(7) Decrements (-1) the digit counter.

(8) Loops until the end of the operation.

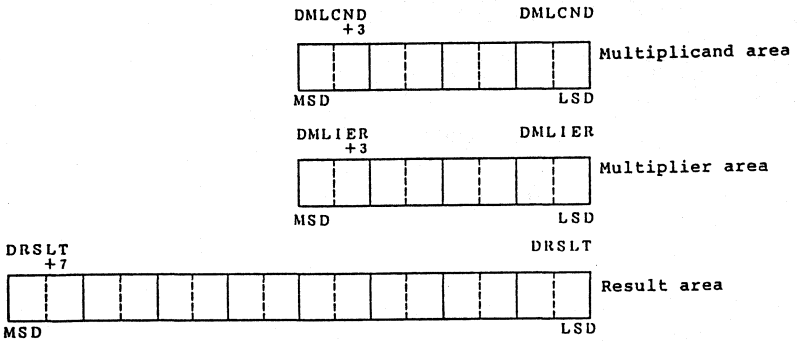
(9) A borrow is generated in the operation result.

(10) The operation result is normal.

1.2.3 Decimal multiplication

16-bit x 16-bit → 32-bit

a) Memories used



b) Registers used

A, B, C, D, E, HL

c) Input conditions

An 8-digit multiplicand and an 8-digit multiplier are stored in the multiplicand area (DMLCND, DMLCND+3) and multiplier area (DMLIER, DMLIER+3) as shown in a).

d) Output conditions

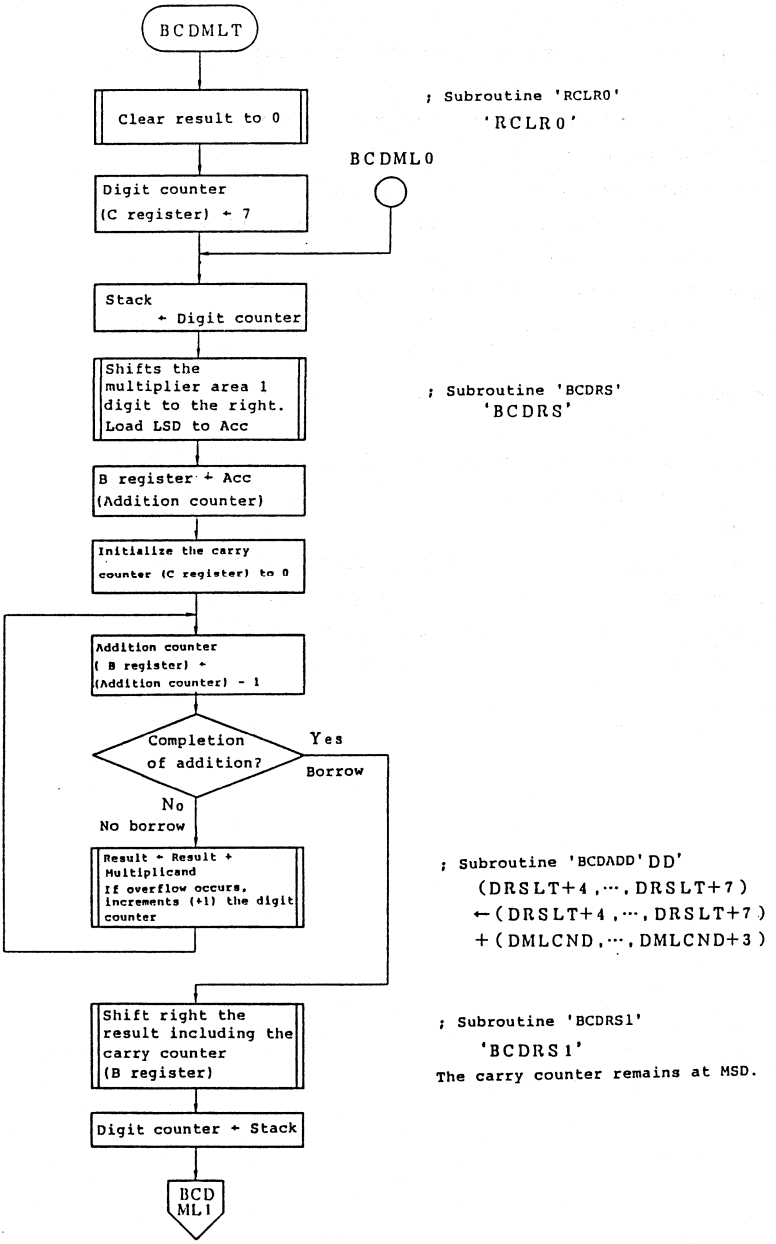
RET ; The result of a normal operation is stored in the result area (DRSLT, ..., DRSLT+7) as shown in a).

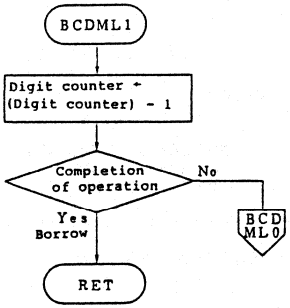
e) Processing procedure

In this arithmetic operation, the multiplier is shifted left and loaded 1 digit at a time beginning with the LSD. Using this as a counter, the "Result ← Result + Multiplicand" operation is repeated. In addition, after 1 digit has been added to the multiplier (counter), the result is shifted 1 digit to the right so that an addition can be made with the multiplier which is 1 word higher.

(1) Clear the result area to 0.
(2) Set the digit counter (C register) to 7.
(3) Store the contents of the digit counter in the stack.
(4) Shift the multiplier to the right, and load the LSD (counter) in the B register.
(5) Decrement the counter (B register)
(6) If a borrow occurs, jump to (9) after the completion of the addition.
(7) Clear the carry counter (C register) to 0.
(8) Perform the operation of "Result (upper 4 bytes) ← Result (upper 4 bytes) + Multiplicand". Increments the carry counter if an overflow occurs. Jump to (5).
(9) Shift the result, including the contents of the carry counter (B register), 1 digit to the right.
(10) Recover the contents of the digit counter from the stack and decrement the digit counter. If a borrow has occurred, complete the operation. If no borrow has occurred, jump to (3).

Flow chart


BCDMLT
Clear result to 0
; Subroutine 'RCLR0'
'RCLR0'
BCDML0
Digit counter
(C register) ← 7
Stack
← Digit counter
Shifts the
multiplier area 1
digit to the right.
Load LSD to Acc
; Subroutine 'BCDRS'
'BCDRS'
B register ← Acc
(Addition counter)
Initialize the carry
counter (C register) to 0
Addition counter
(B register) ←
(Addition counter) - 1
Completion
of addition?
Yes
Borrow
No
No borrow
Result ← Result +
Multiplicand
If overflow occurs,
increments (+1) the digit
counter
; Subroutine 'BCDADD'DD'
(DRSLT+4,…,DRSLT+7)
←(DRSLT+4,…,DRSLT+7)
+(DMLCND,…,DMLCND+3)
Shift right the
result including the
carry counter
(B register)
; Subroutine 'BCDRS1'
'BCDRS1'
The carry counter remains at MSD.
Digit counter ← Stack
BCD
ML1


BCDML1
Digit counter ← (Digit counter) - 1
Completion of operation
No
Yes
Borrow
BCD ML0
RET

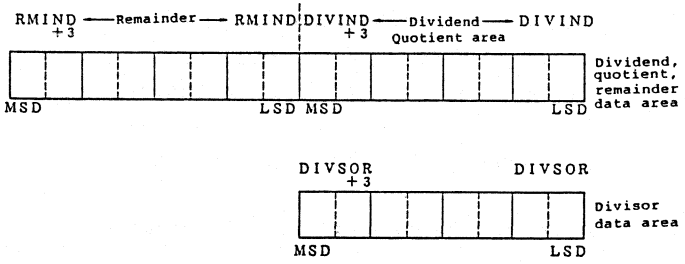
* Program statement

```
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +        DECIMAL  MULTIPLY                 +
;  +      16 PLACE <- 8 PLACE * 8PLACE        +
;  +                                          +
;  +   MULTIPLICAND .. (DMLCND+3,...,DMLCND)  +
;  +   MULTIPLIER   .. (DMLIER+3,...,DMLIER)  +
;  +   RESULT       .. (DRSLT+7,...,DRSLT )    +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
;  **  RESULT,MULTIPLICAND  0CLEAR  **
;
BCDMLT:
        LXI     H,DRSLT
        MVI     C,8-1
        CALL    RCLR0  ; (See page 9)
;
;  **  PLACE COUNTER SET  **
;
        MVI     C,16/2-1
;
;  **  MULTIPLIER  RIGHT-SHIFT  **
;
BCDML1:
        PUSH    B       ; PLACE-COUNTER STORE
        ;
        LXI     H,DMLIER+3
        MVI     C,8/2-1
        CALL    BCDRS  ; (See page 29)
        PUSH    V
;
;  **  CHECK / MULTIPLIER=0'  **
;
        MVI     B,0
BCDML2:
        POP     V
        SUINB   A,1
        GJMP    BCDML4  ; IF  =0
;
;  **  RESULT + MULCND -> RESULT  **
;
        PUSH    V
        LXI     H,DRSLT+4
        LXI     D,DMLCND
        CALL    BCDADD ; SKIP ,IF NORMAL    ; (See page 14)
        INR     B      ; IF OVERFLOW
                       ; DIGIT ADDITION / NOT END
        GJMP    BCDML2
;
;  **  RESULT  RIGHT-SHIFT  WITH CARRY  **
;
BCDML4:
        MOV     A,B        ; LOAD  RESULT-MSD
        LXI     H,DRSLT+7
        MVI     C,16/2-1   ; SHIFT-COUNTER SET
        CALL    BCDRS1     ; (See page 29)
;
;  **  CHECK  / MULTIPLY END ?'  **
;
        POP     B
        DCR     C
        GJMP    BCDML1
        RET             ; END !
```

1.2.4 Decimal division

8-digit ÷ 8-digit → Quotient 8-digit
Remainder 8-digit

a) Memories used



b) Registers used

A, B, C, D, E, H, L, EA

c) Input conditions

An 8-digit dividend and an 8-digit divisor are stored in the areas indicated in a) as follows:

Dividend (DIVIND, ..., DIVIND+3)
Divisor (DIVSOR, ..., DIVSOR+3)

d) Output conditions

RET ; An overflow has occurred due to divisor = 0
RETS ; The result of a normal operation (quotient, remainder) is stored in the areas indicated in a) as follows:
Quotient (DIVIND, ..., DIVIND+3)
Remainder (RMIND, ..., RMIND+3)

e) Processing procedure

In this arithmetic operation, a dividend (DIVIND, DIVIND+1, ..., DIVIND+3) and remainder (RMIND, RMIND+1, ..., RMIND+3) are placed in consecutive 8 bytes, and the dividend and remainder are shifted 1 digit to the left. The highest digit of the dividend is then

transferred to the lowest area of the remainder. The quotient is entered in the lowest area of the divisor 1 digit at a time from the smallest value.

The quotient (remainder - divisor) of each digit is the number of repetitions required for a borrow to occur.

(1) Check if the divisor (DIVSOR, DIVSOR+1, ..., DIVSOR+3) is 0. If 0, terminate the operation.

(2) Clear the remainder area (RMIND, ..., RMIND+3) to 0.

(3) Set the binary counter (B register) of the quotient to 7.

(4) Shift the 8-byte quotient/remainder 1 digit (4 bits) to the left.

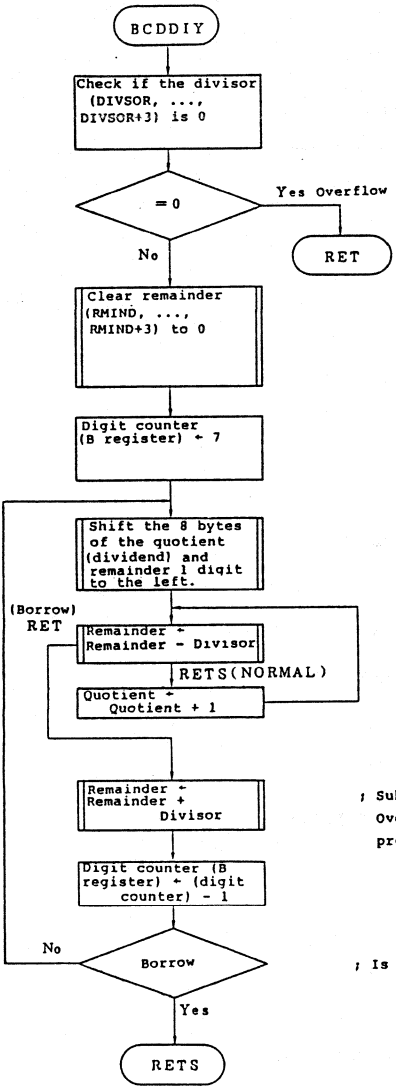
(5) Remainder ← Remainder - Divisor. Jump to (7) if a borrow occurs.

(6) Increment (+1) the quotient (DIVIND). Jump to (5).

(7) The result of (5) is negative due to over-withdrawn. Therefore, perform the Remainder ← Remainder + Divisor operation, in order to return to the previous state.

(8) Decrement the digit counter of the quotient and terminate the operation when a borrow occurs. If not, jump to (4).

Flow chart


BCDDIY
Check if the divisor (DIVSOR, ..., DIVSOR+3) is 0
= 0
Yes Overflow
RET
No
Clear remainder (RMIND, ..., RMIND+3) to 0
; Subroutine 'RMNDCL'
Digit counter (B register) ← 7
Shift the 8 bytes of the quotient (dividend) and remainder 1 digit to the left.
; Subroutine 'BCDLS'
MSD of the dividend is entered to the remainder LSD area
(Borrow) RET
Remainder ← Remainder - Divisor
RETS(NORMAL)
Quotient ← Quotient + 1
; Subroutine 'BCDSUB'
Remainder ← Remainder + Divisor
; Subroutine 'BCDADD'
Over-withdrawn by 1 time; therefore, return to the previous state.
Digit counter (B register) ← (digit counter) - 1
No
Borrow
; Is the operation complete?
Yes
RETS

* Program statement

```
;  ++++++++++++++++++++++++++++++++++++++++
;  +                                      +
;  +          DECIMAL  DIVISION           +
;  +       8-PLACE <- 8-PLACE / 8-PLACE   +
;  +                                      +
;  +     DIVIDEND .. (DIVIND+3,....,DIVIND )  +
;  +     DIVISOR  .. (DIVSOR+3,....,DIVSOR )  +
;  +     RSLT     .. (DIVIND+3,....,DIVIND )  +
;  +     REMIND   .. (RMIND+3,....,RMIND )    +
;  +                                      +
;  ++++++++++++++++++++++++++++++++++++++++
;
BCDDIV:
;
;  **  CHECK / DIVISOR=0'  **
;
        LXI     H,DIVSOR
        LDEAX   H++
        DMOV    D,EA
        LDEAX   H
        DOR     EA,D
        SKN     Z
        RET             ; OVERFLOW
;
;  **  RESULT,REMIND  0CLEAR  **
;
        CALL    RMNDCL ; (See page 10)
;
;  **  DIGIT-COUNTER SET  **
;
        MVI     B,8-1
;
;  **  QUOTIENT,REMIND  LEFT-SHIFT  **
;
BCDDV1:
        LXI     H,DIVIND
        MVI     C,16/2-1
        CALL    BCDLS  ; (See page 34)
;
;  **  SUBTRACT DIVISOR  FROM  DIVIDEND  **
;
BCDDV2:
        LXI     H,RMIND
        LXI     D,DIVSOR
        CALL    BCDSUB   ; (See page 18)
        GJMP    BCDDV3
        ;
        INRW    DIVIND MOD 100H ; 1-DIGIT.QUOT (+1)
        GJMP    BCDDV2
;
;  **  IF BORROW , DIVISOR + DIVIDEND
;
BCDDV3:
        LXI     H,RMIND
        LXI     D,DIVSOR
        CALL    BCDADD   ; (See page 16)
        NOP
;
;  **  CHECK / DIVISION END ?  **
;
        DCR     B       ; SKIP , IF DIVISION END
        GJMP    BCDDV1  ; NOT END !
        RETS            ; END !
```

CHAPTER 2 DATA TRANSFER

This chapter describes data transfer between areas (from an arbitrary area to an arbitrary area) specified in the memory.

Example: Transfer 7 bytes of data addresses 1000 to 1006H to addresses 1100 to 1106H in the same memory, as shown in Fig. 2-1:

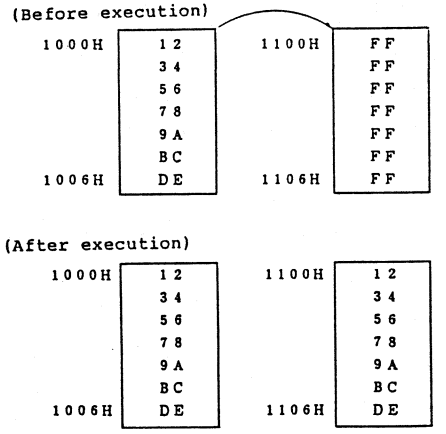


Fig. 2-1 Data Transfer

For the μCOM-87AD, N bytes of data can be transferred within the memory by using the BLOCK instruction.

When executing the BLOCK instruction, pair registers HL and DE, and C register are automatically selected for the transfer source address, transfer destination address, and number of transfer bytes, respectively. Therefore, the transfer source address, transfer destination address, and number of transfer bytes must

be specified by pair registers HL and DE, and the C register before the BLOCK instruction is executed.

In this example, 1000H, 1100H, and (7 to 1) are set in the HL pair register, DE pair register, and C register, respectively.

The following shows the program list.

```
ADRS1   EQU     1000H
ADRS2   EQU     1100H
        ;
        LXI     H,ADRS1
        LXI     D,ADRS2
        MVI     C,7-1
        BLOCK
```

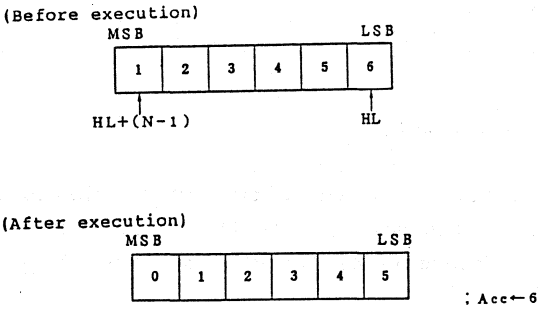
CHAPTER 3 SHIFT PROCESSING

1-bit shift instructions (RLL, RLR, etc.) and 4-bit shift instructions (RLD, RRD) for registers (Acc, B register, C register) are provided for the μCOM-87AD. This chapter describes the following two shift instructions:

- Byte shift
- 4-bit shift

3.1 N-Byte Data Shift Right

The following describes a subroutine which shifts N-byte data to the right in the memory. In the shift right, 0 is stored in the MSB and the contents of the LSB is output to Acc as shown in the figure below.



Before calling this subroutine (shown below), the address of the LSB and number of bytes (>1) of N-byte data must be set in the HL pair register and the C register, respectively.

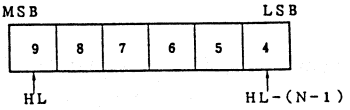
• Program statement

```
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                               S
; S      N-BYTE DATA  RIGHT SHIFT                 S
; S            HL - LSB                           S
; S            C-REG  <-  BYTE COUNTER            S
; S                                               S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BYTRST:
        DCR     C
        DCR     C
        ;
        LDAX    H
        PUSH    V
        ;
BYTRS1: INX     H
        LDAX    H-
        STAX    H+
        ;
        DCR     C
        GJMP    BYTRS1
        ;
        MVI     A,0
        STAX    H
        POP     V
        RET
```

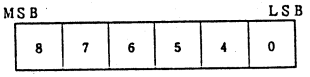
3.2 N-Byte Data Shift Left

The following describes a subroutine which shifts N-bytes of data to the left in the memory. In the shift left, 0 is stored in the LSB and the contents of the MSB is output to Acc as shown in the figure below.

(Before execution)



(After execution)



; Acc←9

Before calling this subroutine (shown below), the address of the MSB and number of bytes (>1) of N-byte data must be set in the HL pair register and the C register, respectively.

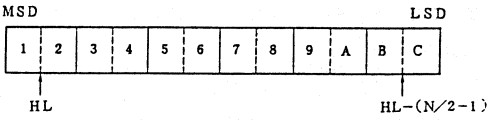
· Program statement

```
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                           S
; S     N-BYTE DATA  LEFT SHIFT               S
; S                  HL <-  MSB               S
; S                  C-REG <-  BYTE COUNTER   S
; S                                           S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BYTLST:
        DCR     C
        DCR     C
        ;
        LDAX    H
        PUSH    V
BYTLS1:
        DCX     H
        LDAX    H+
        STAX    H-
        ;
        DCR     C
        GJMP    BYTLS1
        ;
        MVI     A,0
        STAX    H
        POP     V
        RET
```

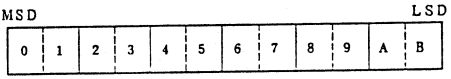
3.3 N-Digit Data Shift Right

The following describes a subroutine which shifts 4 bits the N-digit data to the right in the memory. In the shift right, 0 is stored in the MSD and the contents of the LSD is output to Acc as shown in the figure below.

(Before execution)



(After execution)



; Acc←0CH

Before calling this subroutine (shown below), the address of the MSD and number of bytes (=N/2-1) of the N-digit BCD must be set to the HL pair register and the C register, respectively. However, N=2N' (N'=1, 2, ...).

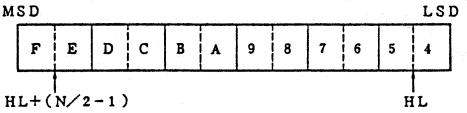
. Program statement

```
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                           S
; S     N-DIGIT DATA  RIGHT-SHIFT             S
; S           HL <- MSD                       S
; S           C-REG <- DIGIT COUNTER          S
; S                                           S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BCDRS:  MVI     A,0
;
BCDRS1: RRD
        DCX     H
        DCR     C
        GJMP    BCDRS1
        RET
```

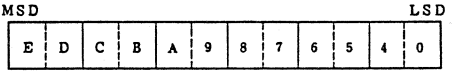
3.4 N-Digit Data Left Shift

The following describes a subroutine which shifts left 4 bits the N-digit data in the memory. In the shift left, 0 is stored in the LSD and the contents of the MSD is output to Acc as shown in the figure below.

(Before execution)



(After execution)



Before calling this subroutine (shown below), the address of the LSD and number of bytes (=N/2-1) of the N-digit BCD must be set to the HL pair register and the C register, respectively. However, N=2N' (N'=1, 2, ...).

* Program statement

```
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                           S
; S     N-DIGIT DATA  LEFT-SHIFT              S
; S           HL <- LSD                       S
; S           C-REG <- DIGIT COUNTER          S
; S                                           S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BCDLS:  MVI     A,0
;
BCDLS1: RLD
        INX     H
        DCR     C
        GJMP    BCDLS1
        RET
```

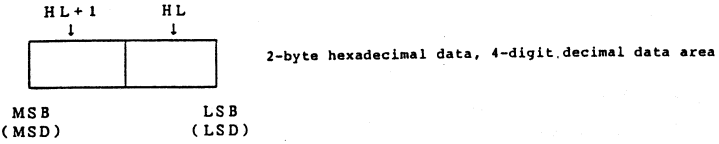
CHAPTER 4 DATA CONVERSION PROCESSING

This chapter describes processing which converts the numeric data expression format between hexadecimal format and decimal format, and hexadecimal format and ASCII format.

4.1 Converting HEX to BCD

Converts 2-byte hexadecimal data to 4-digit decimal data.

a) Memories used



b) Registers used

A, B, C, D, E, H, L, EA

c) Input conditions

The input condition is set as follows as shown in a):

HL Pair register ← The LSB address of the area in which 2-byte hexadecimal input data is stored.

d) Output conditions

RETS ; Converted 4-digit decimal data is stored in the area shown in (HL, HL+1), as indicated in a),

RET ; The hexadecimal data is greater than 270FH (=9999). Therefore, the data cannot be converted.

e) Processing procedure

In this conversion subroutine, a 4-digit decimal conversion value (2 bytes) is obtained 1 digit at a time from the MSD.

In this case, the conversion value of 9876 (=9 x 1000 + 8 x 100 + 7 x 10 +6) is obtained 1 digit at a time

from the MSD in such a manner that subtrahends are set such that MSD is 1000, the 3rd digit is 100, the 2nd is 10, and 1st digit is 1 and the number of subtractions of the hexadecimal input data required until a borrow occurs is 1 digit.

The following is the processing procedure.

(1) Set the subtrahend to 10000.

(2) Perform the operation (hexadecimal input data - subtrahend). If a borrow occurs, it cannot be expressed in 2 types; therefore, terminate as impossible to convert.

(3) Set the digit counter to 4.

(4) Decrement the digit counter and terminate the conversion if a borrow has occurred.

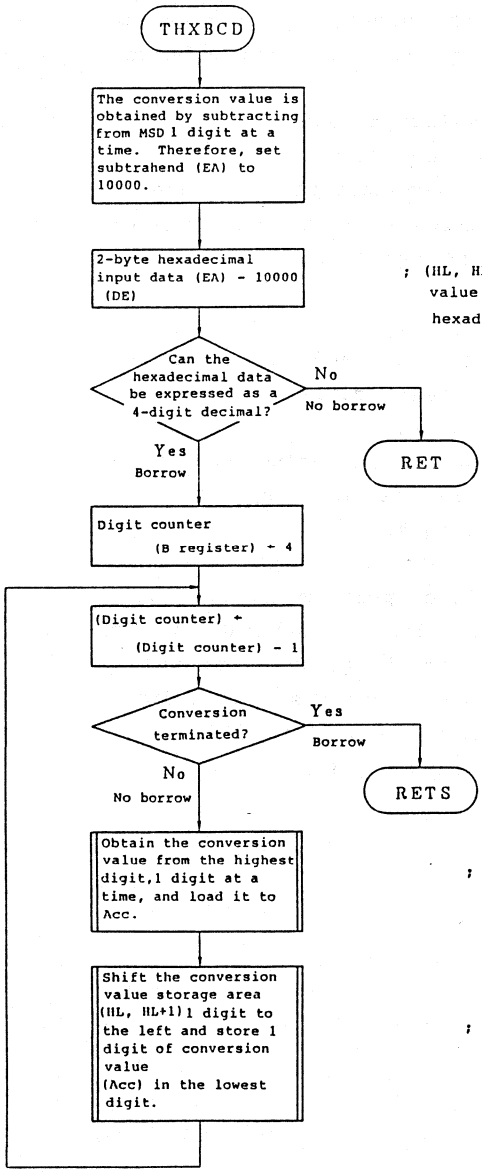
(5) Divide the subtrahend by 10.

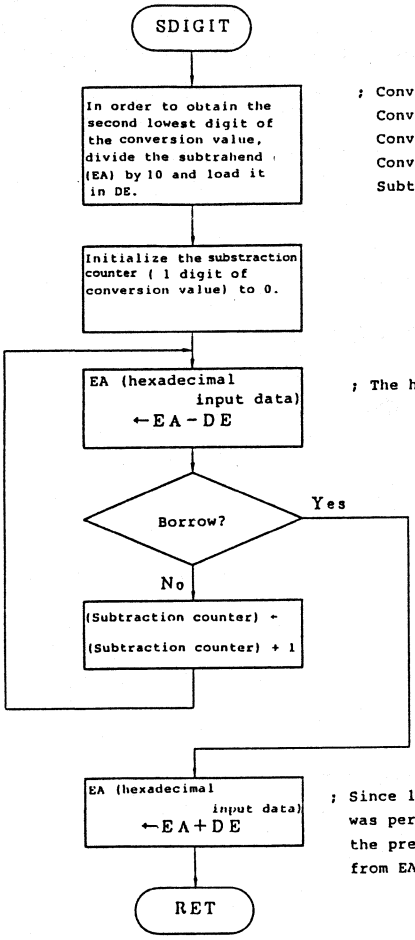
(6) Repeat until a borrow occurs using Acc as a subtraction counter (hexadecimal input data - subtrahend). Count using Acc.

(7) The hexadecimal input data is over-withdrawn in (6) by 1 time. Therefore, perform the operation of (hexadecimal input data + subtrahend) in order to bring it back to the previous state.

(8) Shift the conversion value storage area 1 digit to the left. (HL, HL+1), and store 1 digit of conversion value obtained in (6) in LSD, then jump to (4).

Flow chart


THXBCD
The conversion value is obtained by subtracting from MSD 1 digit at a time. Therefore, set subtrahend (EA) to 10000.
2-byte hexadecimal input data (EA) - 10000 (DE)
; (HL, HL+1) stores the conversion value in order to store the hexadecimal input data to EA.
Can the hexadecimal data be expressed as a 4-digit decimal?
No
No borrow
RET
Yes
Borrow
Digit counter (B register) ← 4
(Digit counter) ← (Digit counter) - 1
Conversion terminated?
Yes
Borrow
RETS
No
No borrow
Obtain the conversion value from the highest digit,1 digit at a time, and load it to Acc.
; Subroutine 'SDIGIT'
Shift the conversion value storage area (HL, HL+1) 1 digit to the left and store 1 digit of conversion value (Acc) in the lowest digit.
; Subroutine 'BCDLS1'



; Convert the 4th digit of the hexadecimal data : EA=1000
Convert the 3rd digit of the hexadecimal data : EA=100
Convert the 2nd digit of the hexadecimal data : EA=10
Convert the 1st digit of the hexadecimal data : EA=1
Subtrahends are set in this manner.

; The hexadecimal input data (HL, HL+1) is stored in EA.

; When exiting from this subroutine,
the subtraction counter is output as Acc.

; Since 1 extra subtraction (hexadecimal input data - DE) was performed, an addition is performed to return to the previous state. The hexadecimal data is recovered from EA to (HL, HL+1).

• Program statement

```
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +     TRANSFER / BCD  <-  HEX               +
;  +              (HL,HL+1)   (HL,HL+1)       +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
THXBCD:
        LXI     EA,10000
        DMOV    D,EA
        ;
        LDEAX   H
        DLT     EA,D     ; CHECK / HEX > 10000
        RET              ; HEX > 10000 !
        ;
        MVI     B,4      ; DIGIT-COUNTER  SET !
;
;  **  CHECK / TRANS END  ?  **
;
THXBD1: SUINB   B,1      ; COUNTER DECREMENT
        RETS             ; END !
;
;  **  SUBROUTINE / HEX -> BCD  **
;
        CALL    SDIGIT   ; 1-DIGIT BCD -> ACC
;
;  **  RESULT * 10 + 1-HEX DATA  **
;
        PUSH    H
        MVI     C,4/2-1
        CALL    BCDLS1   ; ACC -> LSD
        POP     H
        GJMP    THXBD1
```

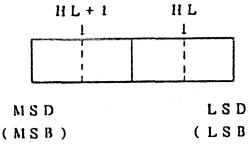
• Program statement

```
;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTST
;  T                                            T
;  T     TRANSFORM   SUBROUTINE                 T
;  T                                            T
;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTS
;
SDIGIT:
        PUSH    EA
        DMOV    EA,D
        MVI     A,10
        DIV     A
        DMOV    D,EA
        POP     EA
        ;
        MVI     A,0      ; RESULT 1-DIGIT 0 INITIAL
SDGT0:  DSUBNB  EA,D
        GJMP    SDGT1
        ;
        INR     A
        GJMP    SDGT0
        ;
SDGT1:  DADD    EA,D     ; FOR ADJUST
        RET
```

4.2 Converting BCD to HEX

Converts 4-digit decimal data to 2-byte hexadecimal data.

a) Memories used



; 2-byte hexadecimal,
4-digit decimal data area

b) Registers used

A, B, C, H, L, EA

c) Input conditions

As shown in a), set as follows:

HL pair register ← The address of LSD of the area in which 4-digit decimal input data is stored.

d) Output conditions

RETS ; Converted 2-byte hexadecimal data is stored in (HL, HL+1) which is shown in a).

RET ; Input data cannot be expressed in decimal format.

e) Processing procedure

The conversion value storage area is cleared to 0. Decimal input data is loaded to Acc 1 digit at a time by shifting the decimal input data from the MSD to the left. The operation of "(Storage area) ← (Storing area) x 10 + Acc" is repeated four times to complete the conversion.

The following is the processing procedure:

(1) Clear the conversion value storage register (EA) to 0.

(2) Set the digit counter to (4-1).

(3) Shift the decimal input data to the left and load the MSD in Acc.

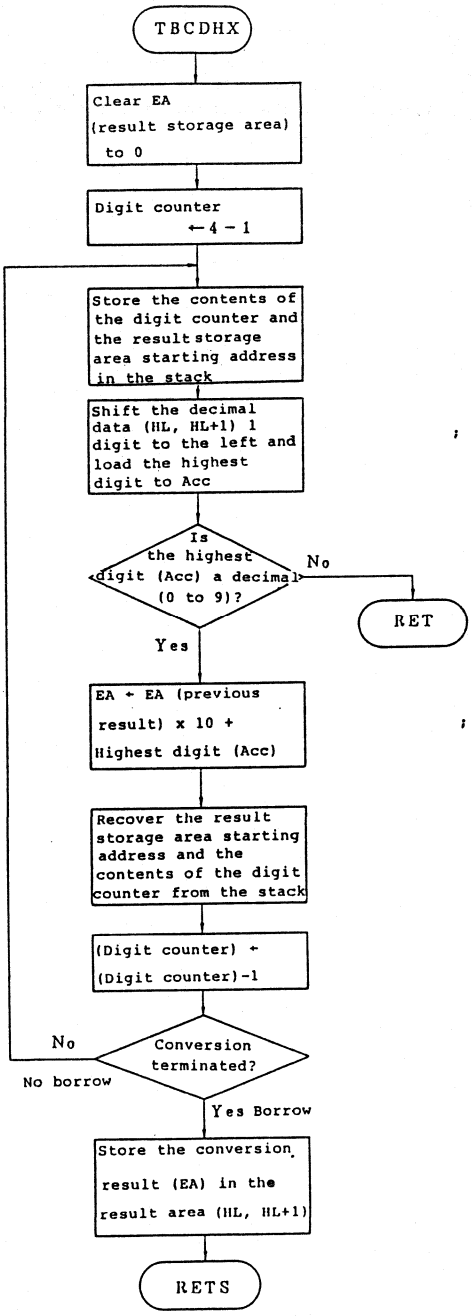
(4) Check whether or not the MSD (Acc) is decimal data (0 to 9).
If not decimal data, terminate the as conversion impossible.

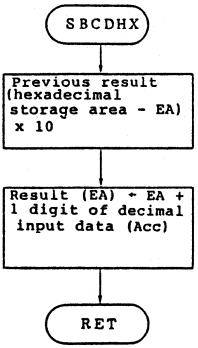
(5) Perform the operation "EA ← EA (previous result) x 10 + Acc".

(6) Decrement the digit counter. If no borrow occurs, jump to (3).

(7) Conversion is terminated when EA is stored in the storing area.

Flow chart


TBCDHX
Clear EA (result storage area) to 0
Digit counter ← 4 - 1
Store the contents of the digit counter and the result storage area starting address in the stack
Shift the decimal data (HL, HL+1) 1 digit to the left and load the highest digit to Acc
; Subroutine "BCDLS"
Is the highest digit (Acc) a decimal (0 to 9)?
No
RET
Yes
EA ← EA (previous result) x 10 + Highest digit (Acc)
; Subroutine "SBCDHX"
Recover the result storage area starting address and the contents of the digit counter from the stack
(Digit counter) ← (Digit counter)-1
No
Conversion terminated?
No borrow
Yes Borrow
Store the conversion result (EA) in the result area (HL, HL+1)
RETS




```
; The result is multiplied by 10 in order to add the
  lower digit of the next decimal input data to the
  previous result.
```

* Program statement

```
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +    TRANSFER * HEX <- BCD    *            +
;  +           (HL,HL+1)  (HL,HL+1)           +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
TBCDHX:
        LXI     EA,0    ; RESULT  0CLEAR
        MVI     A,2*2-1 ; DIGIT COUNTER SET
TBDHX1:
        PUSH    V
        PUSH    H
;
;  **  OUTPUT  1-DIGIT FROM MSD  **
;
        MVI     C,4/2-1
        CALL    BCDLS   ;(See page 30)
        ;
;
;  **  CHECK / BCD DISPLAY ?
;
        LTI     A,9+1
        RET             ; NOT BCD
;
;  *S*  SUBROUTINE / BCD -> HEX  *S*
;
        CALL    SBCDHX
;
;  *S*  SUBROUTINE  END  *S*
;
        POP     H
        POP     V
        DCR     A       ; SKIP  IF TRANSFER END
        GJMP    TBDHX1  ; NOT END
        ;
        STEAX   H       ; STORE RESULT
        RETS
```

• Program statement

```
;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTS
;  T                                             S
;  T     TRANSFER  SUBROUTINE                    S
;  T                                             S
;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTS
;
SBCDHX:
;
;  **  RESULT * 10  **
;
        DMOV    B,EA
        PUSH    V
        MVI     A,10
        MUL     C
        PUSH    EA
        ;
        MUL     B
        POP     B
        EADD    EA,B
        MOV     A,EAL
        MOV     B,A
;
;  **  RESULT *10 + 1-DIGIT  **
;
        POP     V
        DMOV    EA,B
        EADD    EA,A
        RET
```

4.3 Converting ASCII to HEX

The following example describes a process which converts ASCII 2-code (30 to 39H, 41 to 46H) to HEX 2-code (0 to FFH).

Example: Converting ASCII 2-code stored in the BC pair register to HEX 2-code (0 to FFH), and storing the result area to 1 indicated by the HL register. The following describes the program statements and the program.

* Program statement

```
; ++++++++++++++++++++++++++++++++++++++++++++
; +                                          +
; +     TRANSFER : HEX  <=  ASCII            +
; +                (2CODE)   (2CODE)         +
; +                                          +
; +          INPUT : BC REG <= ASCII         +
; +          OUTPUT:  (HL)   <= HEX          +
; +                                          +
; ++++++++++++++++++++++++++++++++++++++++++++
;
GETHEX:
        MOV     A,B     ; ASCII UPPER-CODE LOAD   ①
        CALL    SGTHEX  ; GET ! HEX 1CODE         ②
        RET             ; ERREGULAR ASCII !       ③
        ;
        RLD                                       ④
        MOV     A,C     ; ASCII LOWER-CODE LOAD   ⑤
        CALL    SGTHEX  ; GET ! HEX  2TH CODE     ⑥
        RET             ; ERREGULAR ASCII !       ⑦
        RLD                                       ⑧
        RETS            ; POSSIBLE


; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                            S
; S     SUBROUTINE / GET  HEX 1-CODE(ACC)      S
; S                                            S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
SGTHEX:
        SUINB   A,'0'   ; CHECK / ASCII > 30H     ⑨
        RET             ; ERROR , SO < 30H
        ;
        GTI     A,'9'-'0'                         ⑩
                        ; CHECK / 30H < ASCII < 39H
        RETS
        SUINB   A,'A'-'0'                         ⑪
                        ; CHECK / ASCII > 41H
        RET
        ADINC   A,0AH   ; CHECK / ASCII < 46H     ⑫
        RET             ;
        RETS
```

(1) Loads the upper 1-code of ASCII to the accumulator.

(2) Converts the upper 1-code of ASCII to a HEX 1-code.

(3) ASCII 1-code is not a numeric code (0 to F); therefore, it is handled as conversion impossible.

(4) Stores HEX 1-code in the lower 4 bits of (HL).

(5) Loads the lower 1-code of ASCII in the accumulator.

(6) Converts the lower 1-code of ASCII to a HEX 1-code.

(7) ASCII 1-code is not a numeric code (0 to F); therefore, it is handled as converison impossible.

(8) Shifts the upper 1-code of HEX 4 bits to the left, and stores the lower 1 code of HEX in the lower 4 bits of (HL).

(9) Subtract 30H (0 in HEX) from ASCII code, and check if ASCII 1-code is 30H or higher. If higher than 30H, skip. If not higher than 30H, it is not a numeric code (0 to F); therefore, handle it as conversion impossible.

(10) Check if the ASCII code is within the range of 30H to 39H (0 to 9 in HEX). If it is within this range, terminate the conversion. If not, skip.

(11) Check if the ASCII code is 41H (A in HEX) or higher. If higher than 41H, skip. If not, it is not a numeric code (0 to F); therefore, handle it as conversion impossible.

(12) Check if the ASCII code is 46H (F in HEX) or smaller. If smaller than 46H, convert A to F and skip. If not, handle it as conversion impossible.

4.4 Converting HEX to ASCII

The following shows an example of converting HEX 2-code (0 to FF) to ASCII 2-code (30 to 39H, 41 to 46H).

Example: Converting HEX 2-code (0 to FF) of the area indicated by the HL pair register to ASCII 2-code, and storing the result to the BC pair register.

· Program statement

```
; ++++++++++++++++++++++++++++++++++++++++++++
; +                                          +
; +     TRANSFER / ASCII <=  HEX             +
; +                                          +
; +        INPUT : (HL) <= HEX 2-CODE        +
; +        OUTPUT : BC REG <= ASCII 2-CODE   +
; +                                          +
; ++++++++++++++++++++++++++++++++++++++++++++
;
GETASC:
        MVI     A,0
        RLD                 ; HEX UPPER CODE  LOAD  ①
        CALL    SGTASC                              ②
        MOV     B,A         ; STORE RESULT          ③
        ;
        MVI     A,0
        RLD                 ; HEX LOWER CODE  LOAD  ④
        CALL    SGTASC                              ⑤
        MOV     C,A         ; STORE RESULT          ⑥
        RET


; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                            S
; S     GET /  ASCII 1-CODE ( BC REG )         S
; S                                            S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
SGTASC:
        LTI     A,0AH       ; CHECK / HEX > 9       ⑦
        ADI     A,41H-3AH   ; BIAS (+7)             ⑧
        ADI     A,30H       ; BIAS (+30H)           ⑨
        RET
```

(1) Shifts the HEX code memory 4 bits, and loads the upper partof the HEX code to the accumulator.

(2) Converts the upper 1-code of HEX to ASCII 1-code.

(3) Loads the upper 1 code of ASCII to the B register.

(4) Shifts the HEX code memory 4 bits, and loads the lower HEX code in the accumulator.

(5) Converts the lower 1-code of HEX to ASCII 1-code.

(6) Loads the lower 1-code of ASCII to the C register.

(7) Check if HEX 1-code is 9 or smaller. If 9 or smaller, skip.

(8) Since a bias of 30H converts HEX code (0 to F) to 30 to 3F, ASCII codes A to F would be converted to 41 to 46; as a result, 7 must be added prior to adding 30H.

(9) Adds bias 30H to HEX code.

CHAPTER 5 COMPARISON PROCESSING

This chapter describes the following processing:

- Processing in which two numeric data are compared for branching.
- Data retrieval processing
- Setting/resetting and testing of software flags provided in the memory, per bit

5.1 16-bit (2-byte) Data Comparison

In this processing, the following three different comparison processings are performed:

EQUAL : =
GREATER THAN : >
LESS THAN : <

In order to realize these three processings, the μCOM-87AD has the following 16-bit arithmetic operation instructions so that a comparison can be made between the contents of the expansion accumulator and a pair register. For these instructions, when a condition is satisfied, the next instruction is skipped.

```
DGT EA,H        (1)
DLT EA,H        (2)
DNE EA,H        (3)
DEQ EA,H        (4)
```

(1) Compares the contents of EA and the HL pair register, and skips the next instruction when EA>HL.
(2) Compares the contents of EA and the HL pair register, and skips the next instruction when EA<HL.

(3) Compares the contents of EA and the HL pair register, and skips the next instruction when EA≠HL.

(4) Compares the contents of EA and the HL pair register, and skips the next instruction when EA=HL.

5.2 Data Retrieval

The following describes the process by which data is retrieved from an area in the memory using the specified table shown in the figure below.

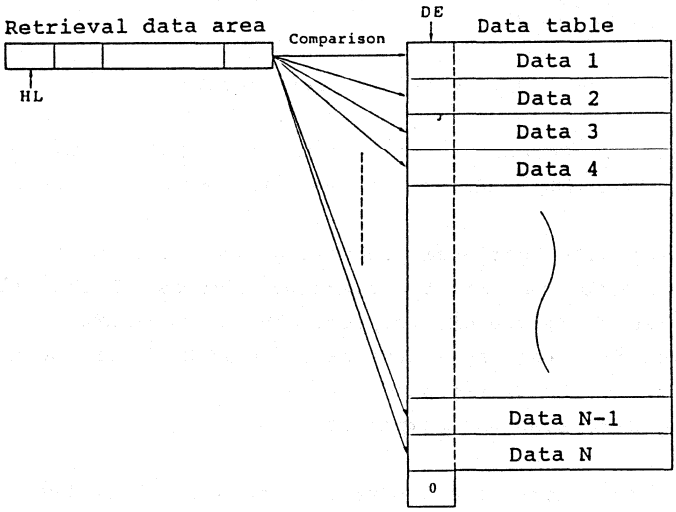


Fig. 5-1 Data Retrieval

The data table and the retrieve data area are formatted in this manner; the data length might or might not be uniformed between the retrieve data and the data table. In the present example, the data length of each data is assumed to be undefined. Therefore, the data format is set as shown in Fig. 5-2, so that checking can be performed during retrieval in order according to data length and the contents of data.

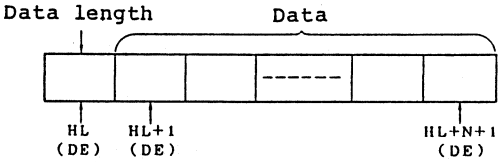


Fig. 5-2 Data Retrieval-Data Format

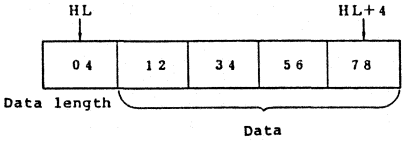
The data table is terminated when 0 is specified as the data length in the starting address of the data area as shown in Fig. 5-2.

When performing data retrieval processing, the starting address of the retrieve data area and the starting address of the table must be specified by HL and DE, respectively, in advance.

When retrieved from the table, the starting address of the data area is output from the DE register.

The following describes an example of this:

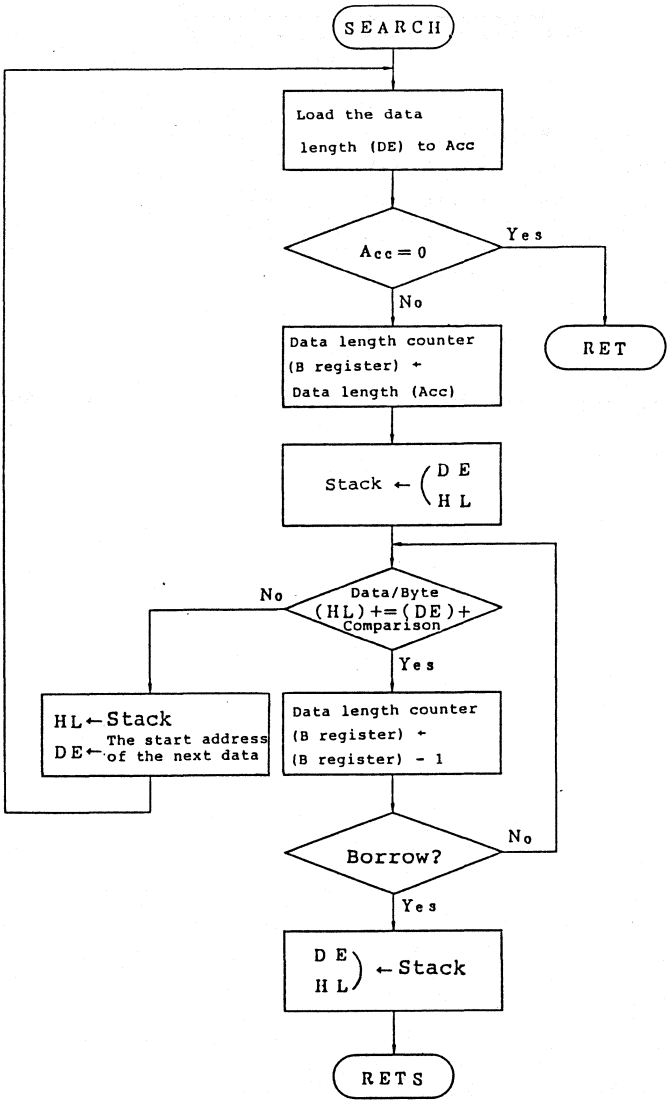
Example: When there is a 5-byte data area of which the starting address is indicated by the HL register, as shown in the figure below, and the starting address of the data table specified in advance is indicated by DE, 6-byte data indicated by the HL register is retrieved at address (LOOK+9) in the table; therefore, this address (LOOK+9) is output from the DE register.




```
                Data length
                    ↓
LOOK :  DB    03,12H,34H,78H
        DB    04,55H,66H,77H,88H
        DB    04,12H,34H,56H,78H
        DB    01,ABH
        DB
```

The following shows the flow chart and processing routine for this example.

Flow chart


SEARCH
Load the data length (DE) to Acc
Acc = 0
Yes
No
RET
Data length counter (B register) ← Data length (Acc)
Stack ← (D E / H L)
No
Data/Byte (HL) += (DE) + Comparison
Yes
HL ← Stack
DE ← The start address of the next data
Data length counter (B register) ← (B register) - 1
Borrow?
No
Yes
(D E / H L) ← Stack
RETS

· Program statement

```
; ++++++++++++++++++++++++++++++++++++++++++++
; +                                          +
; +     DATA  SEARCHING  ( HL ) ( DE )       +
; +                                          +
; +       INPUT : HL <= SERCH DATA ADDRS     +
; +               DE <= TABLE TOP ADDRS      +
; +       OUTPUT : DE <= SEARCH TABLE ADDRS  +
; +                                          +
; ++++++++++++++++++++++++++++++++++++++++++++
;
SEARCH:
        LDAX    D       ; LOAD ! DATA-LENGTH
        GTI     A,1-1   ; CHECK / LENGTH = 0
        RET             ; = 0
        ;
        MOV     B,A     ; STORE LENGTH
        PUSH    D
        PUSH    H       ; STORE SEARCH DATA ADDRS
SERCH1:
        LDAX    D+
        EQAX    H+      ; CHECK / SEARCH DATA = TABLE
        GJMP    SERCH2  ; NOT =
        ;
        DCR     B       ; DATA-LENGTH (-1)
        GJMP    SERCH1  ; NOT END-DATA
        ;
        POP     H
        POP     D
        RETS            ; SEARCH SUCCESS !
;
;  **  FOR NEXT-TABLE DATA CHECK  **
;
SERCH2:
        POP     H
        DMOV    EA,D
        POP     D
        EADD    EA,B
        DMOV    D,EA    ; DE <= NEXT-TABLE DATA ADDRS
        GJMP    SEARCH
```

5.3 Bit Test and Set/Reset in the Memory

a) Memory bit test

The contents of the memory and the immediate data are compared (AND) and memory bits are tested by the Z flag.

Therefore, an instruction (OFFIW) which skips the next instruction when the Z flag is 1 and an instruction (ONIW) which skips the next instruction when the Z flag is 0 are provided.

Each bit of the immediate data of ONIW and OFFIW corresponding to the memory bit to be checked must be set to 1 and 0, respectively.

- ONIW ADRS, 00000011B (1-1)
  Next instruction
  The instruction which exists when the skip condition is satisfied

For (1-1), the next instruction is skipped when bits 0 and 1 of the memory are "1" (Result of AND≠0).

- OFFIW ADRS, 11000000B (1-2)
  Next instruction
  The instruction which exists when the skip condition is satisfied

For (1-2), the next instruction is skipped when bits 6 and 7 of the memory are "0" (Result of AND=0).

b) Memory bit test (When one bit is tested)

To test one bit, the BIT instruction, which is a unique instruction of the μCOM-87AD, can be used.

```
BIT 1, ADRS                (1-3)
Next instruction
The instruction which exists when the skip condition is
satisfied
```

For (1-3), the next instruction is skipped when bit 1 of the memory is set to 1.

c) Memory bit set/reset

The memory can be set/reset per bit by ORing (ORIW) or ANDing (ANIW) the immediate data.

```
· ORIW ADRS, 00000011B    (1-4)
· ANIW ADRS, 00011111B    (1-5)
```

For (1-4), bits 0 and 1 of ADRS are set by ORing ADRS and 00000011B.

For (1-5), bits 7, 6, and 5 of ADRS are reset by ANDing ADRS and 00011111B.

CHAPTER 6 CONDITIONAL BRANCH PROCESSING

The following describes an example of multi-branch processing using the jump instruction. This set the jump destination address by using a pair register or expansion accumulator.

a) JEA instruction

The jump destination address is determined by the expansion accumulator.

The following is an example of multi-branch processing realized by the string effect of the HL pair register.

```
JHL1:    LXI    H,ADRS1
JHL2:    LXI    H,ADRS2
                ;                                  String effect
                ;
                ;
JHL10;   LXI    H,ADRS10
         DMOV   EA,H          ; EA <- HL     ; EA ← HL
         JEA                                 ; Branch by EA
```

b) JB instruction

The jump destination address is determined by the BC pair register.

The following is an example of multi-branching realized by setting address in the BC pair register by using the TABLE instruction.

```
SLL       A          (1)
TABLE                (2)
JB                   (3)
DW        CG00       (4)
DW        CG01
DW        CG02
DW        CG03
;
;
;
DW        CG7F
```

(1) The contents of the accumulator is doubled by shifting left. This is to prevent the address loaded to the BC register when the TABLE instruction is executed from being overlapped.

(2) The contents of the accumulator is added to the current program counter value. Then 2 is additionally added to it. The contents of the address is then set in the C register, and the contents of the address plus 1 is set in the B register.

```
ADRS ← PC+A+2
   C ← (ADRS)
   B ← (ADRS+1)
```

(3) Jumps to the address indicated by the contents of the BC register which has been set by the TABLE instruction.

CHAPTER 7 TABLE REFERENCE PROCESSING

This chapter describes interpolation as an application example of table reference.

As shown in Fig. 7-1, when a point between two points (0, 260) is evenly divided into 26 intervals by point X (X=0, 10, 20, f260), the value of fx at an arbitrary point N within these points (0, 260) can be obtained by interpolation.

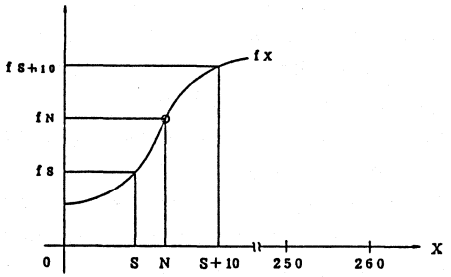


Fig. 7-1 Function fx

The following describes the interpolation procedure for the above problem.

(1) Calculate S=INT(N/10)*10, and determine in which of the 26 intervals N is located by S, S+10 .

(2) Using S obtained in (1), calculate the following formula to obtain the value of fN.

$$f_N = f_S + \frac{(f_{S+10} - f_S)*(N-S)}{10}$$

The following presents an example of an interpolation program.

Example Interpolation program

In this example, interpolation is performed by the value (2-byte) of f0, f10, f20, ..., and f260 in the table in the memory upon point N is given by the accumulator. In this example, as shown in Fig. 1-8-2, the starting address of the table is set by HL in advance, and S is obtained from operation procedure (1) above. The obtained value of S is then doubled (this is because each piece of data (f0, f10, f20, ..., f260) on the table is a 2-byte value) and added to the starting address so that it is possible to determine the data address in which f50 is stored.

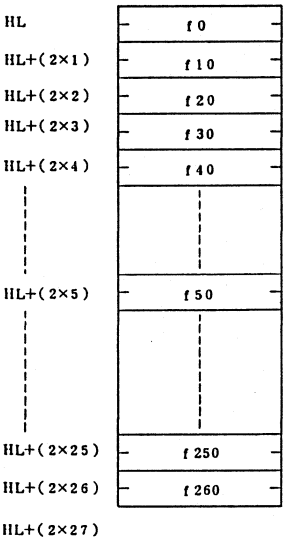
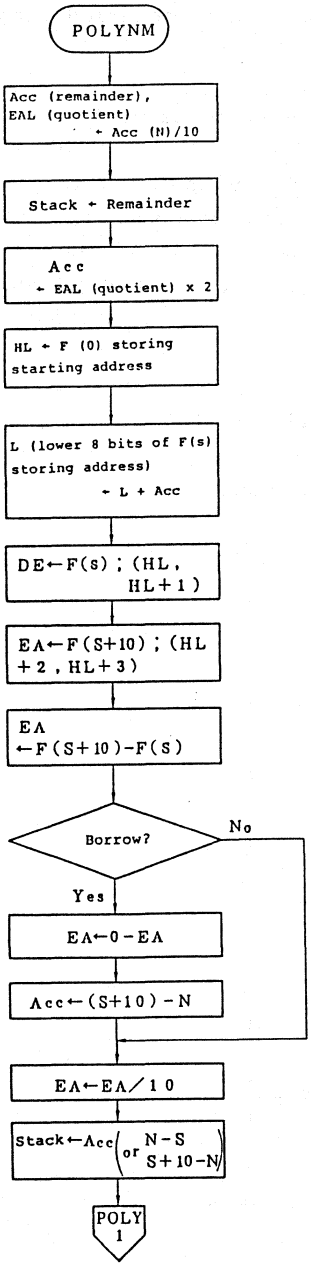
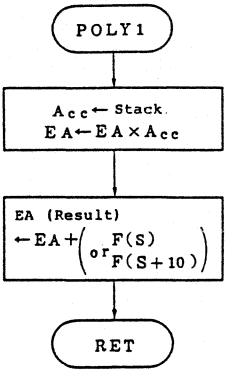


Fig. 7-2 Table Data RAM MAP

Flow chart


POLYNM
Acc (remainder), EAL (quotient) ← Acc (N)/10
Stack ← Remainder
; (N-S)←Remainder
Acc ← EAL (quotient) x 2
; The lower address of F(S) storing address ← Quotient x 2
HL ← F (0) storing starting address
L (lower 8 bits of F(s) storing address) ← L + Acc
DE←F(s) ; (HL, HL+1)
EA←F(S+10) ; (HL+2, HL+3)
EA ←F(S+10)-F(S)
Borrow?
No
Yes
EA←0-EA
Acc←(S+10)-N
; S+10-N←N-S
EA←EA/10
; |F(S+10)-F(S)|/10
Stack←Acc (or N-S, S+10-N)
POLY 1



; $F(S)+\frac{(F(S+10)-F(S))(N-S)}{10}$

or

$F(S+10)+\frac{F(S)-F(S+10)(S+10-N)}{10}$

· Program statements

```
;++++++++++++++++++++++++++++++++++++++++++++++++
;+                                              +
;+          INTERPOLATION  POLYNOMIAL           +
;+                                              +
;+        INPUT      ACC <- X                   +
;+                   HL <= DATA TABLE TOP ADDRS  +
;+        OUTPUT     EA - REG                   +
;+                                              +
;+  F(N)= F(S) + ( F(S+10)-F(S) ) * (N-S) /10   +
;+                                              +
;++++++++++++++++++++++++++++++++++++++++++++++++
;
POLYNM:
        LXI     EA,0
        MOV     EAL,A
        MVI     A,10
        DIV     A
        PUSH    V           ; STACK <- N-S
        ;
        MOV     A,EAL
        SLL     A           ; ACC <- 2BYTE DATA TOP ADDRS
        ;
        DMOV    EA,H
        EADD    EA,A
        DMOV    H,EA
                            ; HL <-  F(S) STORE ADDRS
        LDEAX   H
        DMOV    D,EA        ; DE <-  F(S)
        LDEAX   H+2         ; EA <-  F(S+10)
                            ; HL <- F(S) STORE ADDRS
        POP     V
        DSUB    EA,D        ; F(S+10) - F(S)
        SK      CY
        GJMP    POLYN1      ; IF F(S+10) > F(S)
;
        INX     H
        INX     H
        SUI     A,10
        NEGA                ; 10- ((S+10)-N)
        ;
        DMOV    D,EA
        LXI     EA,0
        DSUB    EA,D        ; F(S) - F(S+10)
POLYN1:
        PUSH    V
        MVI     A,10
        DIV     A           ; ( F(S+10) - F(S) ) / 10
        POP     V
        DMOV    B,EA
;
;    ( F(S+10)-F(S) ) * (N-S) / 10
;
        MUL     C
        PUSH    EA
        ;
        MUL     B
        MOV     A,EAL
        POP     B
        ADD     B,A
;
        LDEAX   H
        DADD    EA,B
        RET
```

NEC Customer Services

NEC's Commitment to Information

Our offices throughout Europe are always at your service for comprehensive support. Here are some of the technical services we provide:

- INSECT
- Seminars
- Update service
- Hotline
- Mailing list
- University program

INSECT

INformation **S**ystem and **E**lectronic **C**a**T**alog.

This is an on-line information service. Via a telephone link you can call up the latest data on all VLSI devices available from NEC. This includes enhancements, news and the most recent application know-how.

The service is free to our customers and other interested parties. For a menu-driven guest session, you can dial in to INSECT via the international packet switching network – in Germany this is DATEX-P – using of these numbers
45 21 10 13 020/030
and responding to the request for USERNAME and PASSWORD simply with "customer".

Seminars

No-one is more aware than NEC of the difference that a brief intensive training course can make to your mastery of advanced and often complex devices. We hold regular workshops and seminars at local NEC offices, in our Düsseldorf headquarters, or on customer premises. For information on NEC workshops and seminars, please contact your nearest office.

Update Service

When you buy an evaluation package from NEC, you become automatically entitled to one year's free updates for both hardware and software. All updates reach you fast and reliably via a courier service. In a field where rapid changes are the norm, you can be thus be sure of working with the most up-to-date development tools.

Hotlines

NEC's offices located throughout Europe are responsible for technical support and customer services. On the back cover of this brochure you can check which office is most convenient for you to contact. You are also welcome to contact our European headquarters in Düsseldorf directly.

Mailing list

Our engineering staff produces frequent additions to the available technical documentation in the form of application notes, product news and technical letters. If you would like to be included on our mailing list for this documentation, please inform us.

University Program

Many of our products, because of their complexity and dedicated application support through EB tools, are interesting subjects for graduate studies. NEC is always ready to discuss this possibility.

EUROPEAN DISTRIBUTORS

AUSTRIA
A & D
ABRAHAMCZIK & DEMEL
GES. MBH. & CO KG
EICHENSTRASSE 58-64/1
1120 WIEN
TEL.: (222) 85 76 61
TLX.: 134 273

BELGIUM
INTRA ELECTRONICS P.V.B.A.
BOSUIL 80, BUS 1
2100 DEURNE
TEL.: (03) 3 25 23 20
TLX.: 31102

MALCHUS ELECTRONICS B.V.B.A.
PLANTIN EN MORETUSLEI 172
2018 ANTWERPEN
TEL.: (03) 2 35 32 72
TLX.: 33 637

INNOCIRCUIT BENELUX B.V.B.A.
PLANTIN EN MORETUSLEI 172
2018 ANTWERPEN
TEL.: (03) 2 35 32 72 (B)
TEL.: (00 31) 10-4 27 77 80
TLX.: 33 637

DENMARK
MER-EL A/S
VED KLAEDEBO 18
2970 HOERSHOLM
TEL.: (2) 57 10 00
TLX.: 37 360

FINLAND
OY COMDAX AB
ITAELAHDENKATU 23A
PI 1
00210 HELSINKI
TEL.: (0) 67 02 77
TLX.: 57 125 876

OY FERRADO A/B
P.O. BOX 54
VALIMONTIE 1
00380 HELSINKI 38
TEL.: (90) 55 00 02
TLX.: 122 214

FRANCE
ASAP
2 AVENUE DES CHAUMES
78180 MONTIGNY LE BRETONNEUX
TÉL.: (1) 30 43 82 33
TLX.: 698 887

C.C.I.
5, RUE MARCELIN BERTHELOT
B.P. 92
92164 ANTONY
TÉL.: (1) 46 66 21 82
TLX.: 203 881

CGE COMPOSANTS
32 RUE GRANGE DAME ROSE
92360 MEUDON
TÉL.: (1) 46 30 24 25
TLX.: 632 118

PANTEC (GROUPE SCIENTECH)
29 RUE MARCEL DASSAULT
92100 BOULOGNE
TEL.: (1) 46 20 06 74
TLX.: 633 048 F

GERMANY
BIT-ELECTRONIC AG
DINGOLFINGER STRASSE 6
8000 MÜNCHEN 80
TEL.: (0 89) 41 80 07-0
TLX.: 5 212 931

GLEICHMANN + CO ELECTRONICS GMBH
WORMSER STRASSE 34
6710 FRANKENTHAL
TEL.: (0 62 33) 2 50 56
TLX.: 4 65 270

GLYN GMBH
SCHÖNE AUSSICHT 30
6272 NIEDERNHAUSEN
TEL.: (0 61 27) 80 77
TLX.: 4 186 911

H3W ELEKTRONIK VERTRIEBS GMBH
STAHLGRUBERRING 12
8000 MÜNCHEN 82
TEL.: (0 89) 42 92 71
TLX.: 5 214 514

MICROSCAN GMBH
ÜBERSEERING 31
2000 HAMBURG 60
TEL.: (0 40) 6 32 00 30
TLX.: 2 13 288

REIN ELEKTRONIK GMBH
LÖTSCHERWEG 66
4054 NETTETAL 1
TEL.: (0 21 53) 73 31 11
TLX.: 8 54 251

SYSTEM ELEKTRONIK VERTRIEBS GMBH
HEESFELD 4
3300 BRAUNSCHWEIG
TEL.: (05 31) 31 40 95
TLX.: 9 52 351

ULTRATRONIK GMBH
GEWERBESTRASSE 4
8036 HERRSCHING
TEL.: (0 81 52) 37 09-0
TLX.: 5 26 459

UNIELECTRONIC VERTRIEBS GMBH
LISE-MEITNER-STRASSE 8
6072 DREIEICH 1 B. FRANKFURT
TEL.: (0 61 03) 3 51 75
TLX.: 4 11 213

ITALY
ADELSY S.R.L.
VIA DEL FONDITORE, 5
LOCALITA ROVERI
40127 BOLOGNA
TEL.: (51) 532119
TLX.: 510 226

CLAITRON S.P.A.
VIA GALLARATE, 211
20151 MILANO
TEL.: (2) 3010091
TLX.: 313 843

DIS.EL. S.P.A.
VIA ALA DI STURA 71
10148 TORINO
TEL.: (011) 2201522
TLX.: 215 118 DISEL I

MELCHIONI S.P.A.
VIA COLLETTA, 37
20135 MILANO
TEL.: (2) 57941
TLX.: 315 293

NETHERLANDS
MALCHUS B.V.
FOKKERSTRAAT 511
3125 BD SCHIEDAM
TEL.: (10) 4 27 77 77
TLX.: 21 598 MALCH

INNOCIRCUIT BENELUX B.V.
POSTBUS 48
3100 AA SCHIEDAM
TEL.: (10) 4 27 77 80
TLX.: 21 598 MALCH

INTRA ELECTRONICS B.V.
GULBERG 33
5674 TE NUENEN
TEL.: (40) 83 80 09
TLX.: 59 418

NORWAY
JAKOB HATTELAND ELECTRONIC A/S
P.B. 25
5578 NEDRE VATS
TEL.: (47) 63 111
TLX.: 428 50

PORTUGAL
AMPEREL S.A.
AV. FONTES PEREIRA DE MELO 47, 4D
1000 LISBOA
TEL.: (1) 53 26 98
TLX.: 18588

SPAIN
COMELTA SA
EMILIO MUNOZ 41. NAVE 1-1-2
28037 MADRID
TEL.: (1) 7 54 30 77
TLX.: 42 007

VENCO
CALLE GALILEO 249
08028 BARCELONA
TEL.: (3) 330 97 51
TLX.: 98 266

SWEDEN
NAX AB KOMPONENTBOLAGET
BOX 4115
17104 SOLNA
TEL.: (8) 98 51 40
TLX.: 17912

NORDQVIST & BERG
BOX 1458
17128 SOLNA
TEL.: (8) 7 64 67 10
TLX.: 10407

TH'S ELEKTRONIK
BOX 3027
16303 SPAANGA
TEL.: (8) 36 29 70
TLX.: 11145

SWITZERLAND
MEMOTEC AG
GASWERKSTRASSE 32
4901 LANGENTHAL
TEL.: (63) 28 11 22
TLX.: 9 82 550

UNITED KINGDOM
2001 ELECTRONIC COMPONENTS
WOOLMERS WAY
STEVENAGE
HERTS
SG1 3AJ
TEL.: (4 38) 74 20 01
TLX.: 827 701

ANZAC COMPONENTS LIMITED
822, YEOVIL ROAD
SLOUGH TRADING ESTATE
SLOUGH
BERKSHIRE
SL1 4JA
TEL.: (62 86) 44 11
TLX.: 847 949

CELDIS LIMITED
37, LOVEROCK ROAD
READING
BERKSHIRE
RG3 1EL
TEL.: (7 34) 58 51 71
TLX.: 848 370

IMPULSE ELECTRONICS LIMITED
HAMMOND HOUSE
CATERHAM
SURREY
CR3 6XG
TEL.: (8 83) 4 70 11
TLX.: 291 496

S.T.C. MULTICOMPONENTS LIMITED
EDINBURGH WAY
HARLOW
ESSEX
CM20 2DF
TEL.: (2 79) 44 11 44
TLX.: 818 763

V.S.I. ELECTRONICS (UK) LIMITED
ROYDONBURY INDUSTRIAL PARK
HORSECROFT ROAD
HARLOW
ESSEX
CM19 5BY
TEL.: (2 79) 2 96 66
TLX.: 81 387

NEC ELECTRONICS (UK) LTD
CYGNUS HOUSE
LINFORD WOOD BUSINESS CENTRE
SUNRISE PARKWAY
LINFORD WOOD
MILTON KEYNES
MK14 6NP
TEL.: (9 08) 69 11 33
TLX.: 826 791

DUBLIN OFFICE
34/35 SOUTH WILLIAM STREET
DUBLIN 2
IRELAND
TEL.: (0 01) 79 42 00
TLX.: 90 847

SCOTLAND OFFICE
BLOCK 3
CARFIN INDUSTRIAL ESTATE
MOTHERWELL
SCOTLAND
ML1 4UL
TEL.: (6 98) 73 22 21
TLX.: 777 565

NEC OFFICES

NEC Electronics (Europe) GmbH, Oberrather Str. 4, 4000 Düsseldorf 30, W. Germany,
Tel. (02 11) 65 03 01, Telex 8 58 996-0

NEC Electronics (Germany) GmbH, Oberrather Str. 4, 4000 Düsseldorf 30,
Tel. (02 11) 65 03 02, Telex 8 58 996-0
- Königstr. 12, 3000 Hannover 1, Tel. (05 11) 31 60 91, Telex 9 230 109
- Arabellastr. 17, 8000 München 81, Tel. (0 89) 92 10 03-0, Telex 5 22 971
- Heilbronner Str. 314, 7000 Stuttgart 30, Tel. (07 11) 89 09 10, Telex 7 252 220

NEC Electronics (BNL) - Boschdijk 187a, NL-5612 HB Eindhoven, Tel. (0 40) 44 58 45,
Telex 51 923

NEC Electronics (Scandinavia) - Svärdvägen 25 B, S-18233 Danderyd,
Tel. (08) 75 36 020, Telex 13 839

NEC Electronics (France) S.A., 9, rue Paul Dautier, B.P. 187,
F-78142 Velizy Villacoublay Cedex, Tel. (1) 39 46 96 17, Telex 699 499

NEC Electronics (France) S.A., Representacion en Espana, Edificio «La Caixa»,
Paseo de la Castellana 51, E-28046 Madrid, Tel. (1) 41 94 150, Telex 41 316

NEC Electronics Italiana S.R.L., Via Fabio Filzi, 25A, I-20124 Milano, Tel. (02) 67 09 108,
Telex 315 355
- Rome Office, Via Monte Cervialto, 131, I-00139 Rome,
Tel. (06) 8 11 12 91, Telex 623 323

NEC Electronics (UK) Ltd., Cygnus House, Sunrise Park Way, Milton Keynes, MK14 6NP,
Tel. (09 08) 69 11 33, Telex 826 791
- Dublin Office, 34/35 South William Street, Dublin 2, Ireland, Tel. (00 01) 79 42 00
- Scotland Office, Block 3, Carfin Industrial Estate Motherwell ML1 4UL, Scotland,
Tel. (6 98) 73 22 21, Telex 777 565

UM87TOOLS.079V50