

## Table of Contents

Preface . . . . .	iii
Glossary of terms . . . . .	v
<b>PART 1 ASSEMBLY LANGUAGE</b> . . . . .	1-1
Introduction . . . . .	1-3
Syntax description . . . . .	1-7
<b>Chapter 1 Format of source statements</b> . . . . .	1-9
Label field . . . . .	1-11
Operation field . . . . .	1-11
Operand field . . . . .	1-13
Comment field . . . . .	1-16
Input of source statements and corrections . . . . .	1-16
Addressing modes . . . . .	1-17
<b>Chapter 2 Functional operation of instructions</b> . . . . .	1-19
Load and Store instructions . . . . .	1-19
Arithmetic instructions . . . . .	1-19
Logical instructions . . . . .	1-19
Character handling instructions . . . . .	1-19
Branch instructions . . . . .	1-19
Shift instructions . . . . .	1-21
Control instructions . . . . .	1-22
I/O instructions . . . . .	1-22
External transfer instructions . . . . .	1-23
Move table instructions . . . . .	1-23
<b>Chapter 3 Assembly directives</b> . . . . .	1-25
Program framework . . . . .	1-26
IDENT . . . . .	1-27
END . . . . .	1-28
Linkage control . . . . .	1-29
ENTRY . . . . .	1-30
EXTRN . . . . .	1-31
COMM . . . . .	1-32
Assembly control . . . . .	1-34
IFT . . . . .	1-35
IFF . . . . .	1-35
XIF . . . . .	1-35
STAB . . . . .	1-36

AORG . . . . .	1-37
RORG . . . . .	1-37
Value definition . . . . .	1-38
DATA . . . . .	1-38
EQU . . . . .	1-40
Area reservation . . . . .	1-41
RES . . . . .	1-41
Listing control . . . . .	1-42
EJECT . . . . .	1-42
NLIST . . . . .	1-42
LIST . . . . .	1-42
Symbol generation. . . . .	1-43
FORM . . . . .	1-43
XFORM. . . . .	1-47
GEN . . . . .	1-48
List of predefined symbols. . . . .	1-49
<b>Chapter 4 Programming considerations . . . . .</b>	<b>1-51</b>
Stand Alone or Monitor controlled programming. . . . .	1-51
Interrupt system . . . . .	1-51
System stack . . . . .	1-51
User stack . . . . .	1-52
Trap action . . . . .	1-54
Simulation routine. . . . .	1-54
Adaptation of P855M software to P800M software . . . . .	1-54
Use of RTN instruction . . . . .	1-55
Stand Alone Input and Output Programming . . . . .	1-55
<b>PART 2 ASSEMBLER . . . . .</b>	<b>2-1</b>
<b>Introduction . . . . .</b>	<b>2-3</b>
<b>Chapter 1 Processing . . . . .</b>	<b>2-5</b>
Input . . . . .	2-5
<b>Chapter 2 Output . . . . .</b>	<b>2-9</b>
Chapter 3 Error messages . . . . .	2-13

<b>PART 3 LINKAGE EDITOR . . . . .</b>	<b>3-1</b>
<b>Introduction . . . . .</b>	<b>3-3</b>
<b>Chapter 1 Processing . . . . .</b>	<b>3-5</b>
Link-edit operation . . . . .	3-5
Link-load operation . . . . .	3-6
<b>Chapter 2 Input . . . . .</b>	<b>3-7</b>
Define entry points . . . . .	3-9
Define external reference names . . . . .	3-10
Define relative base address for relocatable program sections . . . . .	3-10
Define absolute base address for relocatable program sections. . . . .	3-10
Process input file up to end of file mark . . . . .	3-11
Select a specified object module in the input file . . . . .	3-11
Satisfy external reference from an object module library . . . . .	3-11
List the names of all unsatisfied external references . . . . .	3-11
Terminate processing. . . . .	3-12
<b>Chapter 3 Output . . . . .</b>	<b>3-13</b>
<b>Chapter 4 Linkage Editor generation . . . . .</b>	<b>3-19</b>

- Absolute addressing** addressing specific locations in memory (see also relocatable addressing)
- Assembler** a system program which translates programs written in Assembly Language into binary object code
- Bootstrap** a program provided for initial loading of the system
- Breakpoint** address at which execution of program stops to allow further debugging
- Character** eight bits, representing an integer, letter or other data item
- Cluster** a set of data in object code
- Common blank common** an area to which external references can be made from one or more modules
- labeled common** a predefined external reference which can be used in several modules
- Debugging Package** a processor which allows the programmer to insert breakpoints in a load module and call debugging functions before execution of a program
- Directive** an instruction used for providing a framework for a program or for guiding the assembly process
- Effective memory address** address in memory where the actual information can be found
- Entry point** a label to which an external reference is made
- External reference** a reference to an entry point in another program or module
- File code** one or two hexadecimal digits associated with an I/O device

**Identifier** a character or a combination of characters used to label an instruction or a value which is to be referred to by other instructions

**Internal symbol** identifier in a module

**IPL** Initial Program Loader. A program to load the monitor

**Label** identifier of max. six characters long, the first always being a letter

**Linkage Editor** a processor used to link independent object modules before execution

**Load Module** program output by the Linkage Editor containing no external references

**Location counter** counter used to assign a relative or absolute address to program elements

**Mnemonic** abbreviation for an instruction, as used in the operation code field of a source statement, to indicate a machine instruction or directive

**Module** a part of a program, enclosed by an IDENT and END directive, which can be treated independently of the rest of the program

**Monitor** a system program which supervises the loading, processing and execution of user programs, starts and supervises the operation of processors and initialises I/O operations

**Object code** program as translated by a language translator and suitable to be input to the Linkage Editor

**Operand** an expression indicating the address, value or register to be operated upon by the machine instruction

**Pass** one program run

**Real Time Clock** a mechanism by means of which the amount of computer time allocated to a program is measured and a signal is given when that period of time has ended

X

**Relocatable addressing** addressing in relation to the beginning of a program, not to specific locations in memory. The relocation of the addresses is then done by the machine

**Source statement** one line in a source program

**Stand Alone processor** processor not running under Monitor control. It contains its own I/O routines

**Symbol** an identifier, used as an address value in the operand field of other instructions

**Update Package** a processor which handles the additions and deletions in source or object programs

XI

**PART 1**

**ASSEMBLY LANGUAGE**

---

This part contains a description of the Assembly Language. In this description it is made clear how the programmer can write his programs using the instructions of the P800M Instruction Set as well as the directives which guide the assembly process when the program is input to the Assembler. The instruction sets of the P800M series computers are upward compatible.

Programs for the P800M computers are written in a symbolic language closely related to the machine code. Each statement (or line) of the program relates to a single machine instruction or to a data item to be taken into account by an instruction.

To write programs in the Assembly Language, the user should be familiar with the syntax of the instructions, which are divided in the following main groups:

- Load and Store instructions
- Arithmetic instructions
- Logical instructions
- Character handling instructions
- Branch instructions
- Shift instructions
- Control instructions
- Input/Output instructions
- External Transfer instructions
- Move Table instructions.

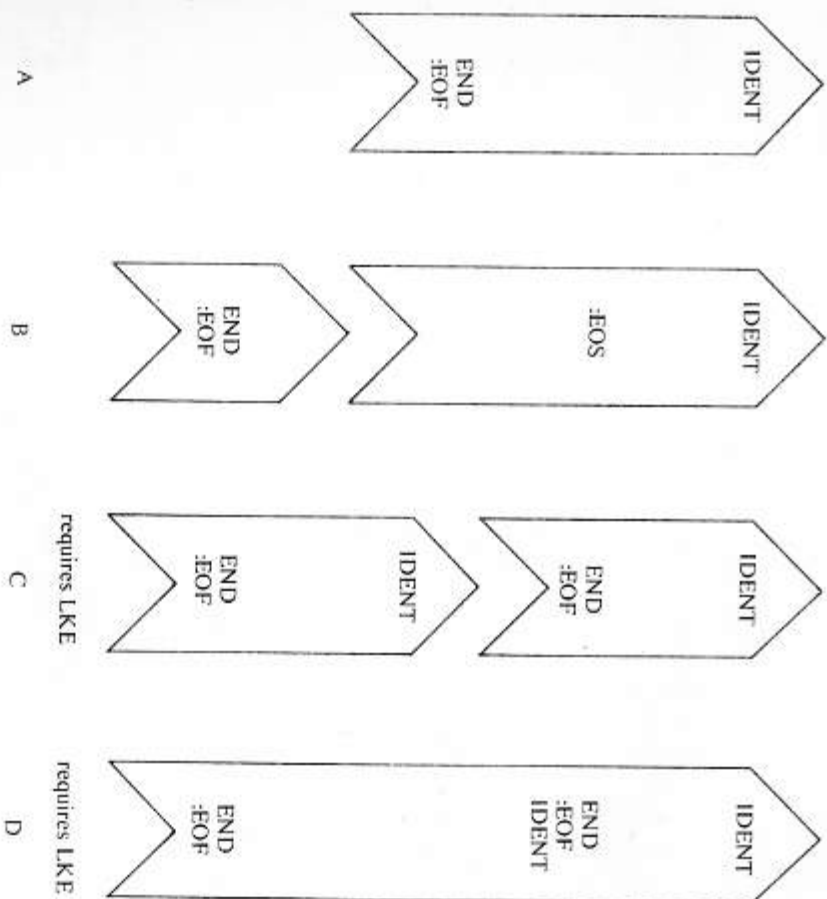
Programming in Assembly Language requires certain rules to be acceptable to the Assembler.

A source program may consist of one or more modules each of which starts with an identification IDENT and terminates with an END (see directives). The whole source program must be terminated by an "End Of File" mark (:EOF).

**NOTE:** If a source program consists of several modules the modules need not be separated by :EOF marks but by :EOS marks (End Of Segment). An :EOS mark at the end of a punched tape indicates the physical end of that tape when the program is punched on two tapes. The mark is not part of the Assembly Language.

The following figure shows various possibilities of how programs can be punched on tape.

In example A the program is contained on one punched tape. The program starts with an identification IDENT and is terminated by END which will cause an :EOS to be punched when the program is assembled, and is followed by an :EOF.



Example B is an example of a program punched on two tapes. The first tape starts with an IDENT and is terminated with :EOS which causes the Assembler to wait for operator action. The second tape does not contain an IDENT and is read immediately after the first one. The second tape is terminated by an END and :EOF.

Example C consists of two modules on two tapes both beginning with IDENT and ending with END and :EOF.

Example D consists of several modules punched on one tape. Each module begins with an IDENT and is terminated by END and either an :EOS mark if another module follows this one or by :EOF when it is the last module to be processed. This example requires the Linkage Editor to make of those modules one larger program which can be executed.

Each module of a program consists of a number of characters grouped into lines and each statement in a module is made up of the following characters:

- Letters:** A to Z inclusive
- Digits:** 0 to 9 inclusive
- Delimiters:**
  - + plus
  - minus
  - \* asterisk
  - = equal
  - apostrophe
  - comma
  - blank
  - / slash
  - ( left parenthesis
  - ) right parenthesis
  - period
  - colon

#### Location counter

The Assembler maintains a location counter which is a software counter used to assign a relative or absolute memory address to program elements. The location counter starts with a relative value equal to zero, or it starts at an absolute address defined by the AORG directive, at the beginning of an assembly. The value of the counter is incremented by 2 or a multiple of 2 depending on the kind of instruction given.

The current value of the location counter is referred to by an \* in the operand field (see below). In absolute program sections \* has an absolute value. In that case the value is incremented in the normal way and the value may be changed by a RES or RORG directive.

The location counter may take neither a negative relative value nor an odd value.

#### Symbols

A symbol is a character or a string of characters used to represent addresses or values. Symbols may appear in the label field as well as in the operand field of a statement.

Their syntax is the same as for the label (see under label field). Some symbols are predefined and have a special meaning for the Assembler e.g. \* indicates the current value of the location counter, P is the instruction counter etc.