# System 35
# Desktop Computer

## Microprocessor Development Software

HEWLETT
PACKARD

# WARRANTY STATEMENT

Hewlett-Packard (HP) warrants its computer software products against defects in materials and workmanship for a period of ninety (90) days. During the warranty period, HP will, at its option, repair or replace products which prove to be defective. Any defective materials must be returned directly to HP. Products that are installed are warranted from the installation date, all others from the ship date.

If the buyer schedules or delays the installation more than 30 days after delivery, then warranty period starts on the 31st day from the date of shipment.

HP warrants that its software and firmware designated by HP for use with a CPU will execute its programming instructions when properly installed on that CPU. HP does not warranty that the operation of the CPU, or software, or firmware will be uninterrupted or error free.

No other warranty is expressed or implied. The implied warranties of merchantability and fitness for a particular purpose are specifically disclaimed. There is no liability for any direct, indirect, special, incidental or consequential damages, however based.

# Microprocessor Development Software

Hewlett-Packard Desktop Computer Division
Postfach 1430, 7030 Böblingen, Federal Republic of Germany

# Preface

The Microprocessor Development Software provides additional capabilities to the HP Desktop Computer Models 9835A and 9845B to allow you to write Assembler language programs for one of three microprocessor types.

The instructions given in this manual consider that you have a working knowledge of the 9835A or 9845B Desktop Computer. Complete operating instructions for these instruments are given in the appropriate manuals supplied with each Desktop Computer. Program operation with either Desktop Computer is essentially the same. Where a difference in key definition occurs, both are given. When using a software program, the keys CONTINUE (9835A), CONT (9845B) and EXECUTE have the same function. The word CONTINUE is used through-out this manual.

The manual does not present any information which will help you to understand your chosen Microprocessor. We advise you to refer to the manufacturers documentation to achieve an understanding of their microprocessors before reading this manual.

Section VII provides information and descriptions of the Assembler language instructions for each microprocessor type. These instructions apply to each particular type of microprocessor and they are not interchangeable.

If any program which is described in this manual fails, please contact your local HP Sales and Service Office. Office locations are listed at the back of this manual.

At the front of this manual you will find a Software Registration card that enables you to receive information concerning revisions, additions or modifications to this software pack for a period of one year.

# Table of Contents

**Section One: Introductory Description**

**Section Two: Quick Reference Guide**

**Section Three: Generation of the Program Cartridge**

**Section Four: Obtaining the Main Menu**

**Section Five: Initialization Module**

**Section Six: The Editor Module**

**Section Seven: 8080/85 Assembler Language Instructions**

Section Seven: 6800 Assembler Language Instructions

Section Seven: Z80 Assembler Language Instructions

**Section Eight: Macros**

**Section Nine: Assembler Module**

**Section Ten: Console Module**

**Section Eleven: Examples on applying the pack**

# Section One
# Introductory Description

## Introduction

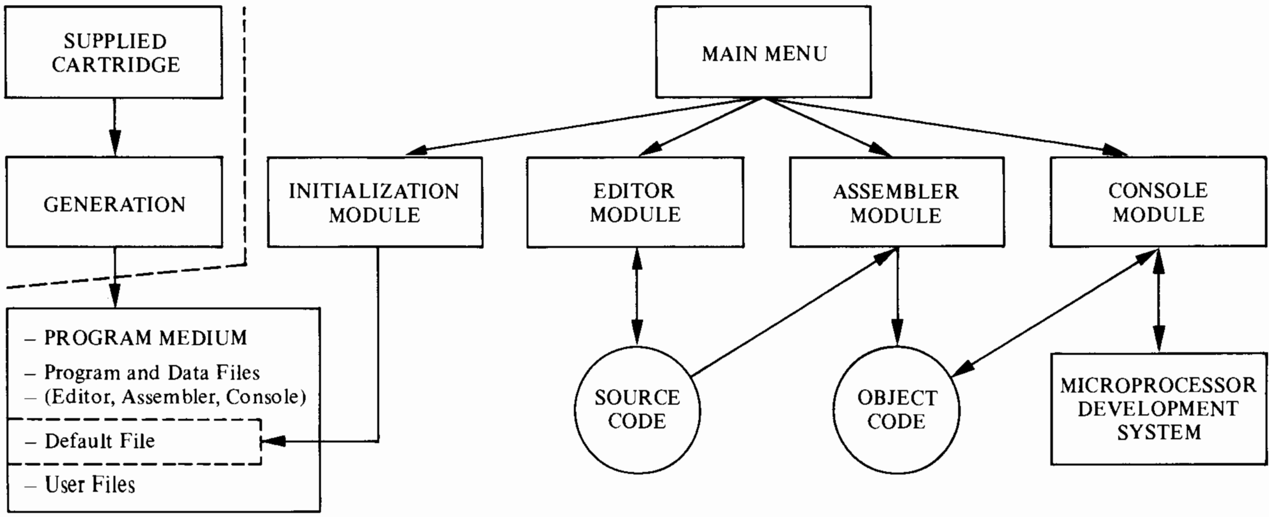
Your Microprocessor Development Software is a cartridge for use with the 9835A or 9845B Desktop Computer. The cartridge contains all the necessary Cross-Assembler programs and data files to generate Assembler language programs using the instructions of your chosen microprocessor type, namely the 8080/85, the 6800, or the Z80.

Capabilities of the program allow you to;

Create or modify Source code
Assemble Source code into Object code
Download the Object code into a compatible target microprocesser system under control of the microprocessor monitor operating system.
Communicate with the target microprocessor system.

The generation procedure loads the cartridge into the Desktop computer whereby a copy is made onto some chosen form of mass storage. When this phase is completed, the generated program asks you which of the four available modules you intend to work with.

The Initialization Module, for changing the peripheral default values.
The Editor Module, for writing your Source code.
The Assembler Module, that translates your Source code into Object code.
The Console Module, that allows you to transfer your final Object code to a Microprocessor system under control of the target processor operating system.



# Terms

During the course of this manual, phrases will be used which are in common circulation in the data processing industry. While the meaning of most are either well-known or deducible from the content, there are a few which may be new to you.

**Default Value.** Such parameters as printer width, paper length, tape cartridge selection code, printer select code etc., are stored on a file known as the default value file. These values are known as default values. Unless changes are made the default values are used during the operation of the program.

**Macro.** Defines a body of text that is automatically inserted into your Source code each time a Macro is called.

**Module.** In programming, a program segment which performs a specific, independent program task.

**msus** (mass storage unit specifier). This specifier tells the System which mass storage unit it is addressing and where it can be found. Refer to System 9835A/9845B Operating and Programming manual for further information.

**Object Code.** Is the result of putting the Source code through an Assembler program. It can also be called Machine code and is usually in hexadecimal code.

**Object file.** The file in which your absolute machine code is stored in the format of your particular microprocessor.

**Select code.** The computer accesses I/O devices with a select code. It is an expression in the range 0 through 16. Refer to the 9835A/9845B Operating and Programming manual for further information.

**SFK** (Special Function Keys). The keys to the top right of the keyboard have been predefined by the program. The definition is contained on the overlay. Some keys can be defined for use as typing aids for often used statements.

**Source Code.** This is the program written by you in Microprocessor Assembler language.

**Source file.** The file in which your written Source code is stored.

**Work files.** The files where any overflow of your Source code, from the Desktop Computer memory, is stored.

# Conventions

## Description and range of parameter values.

<LNA>, <LNB>, <LNC> , <LN> – Different line numbers from 1 to 99999.

<INCREMENT> – A number for the line increment.

<STR1>, <STR2>, <STR> – A given string with a maximum of 74 characters.

<SC> – Select Code. An integer in the range 0 through 16, or an HP-IB address, for example 7,1.

msus – Mass storage unit specifier. For example:

":T15" is the standard cartridge drive
":T14" is the optional cartridge drive
":F8" is the Flexible disc at select code 8.

## Syntax

The special rules of the command syntax are explained below.

< > – Items within angled brackets must be replaced by an actual value.

[ ] – Items within square brackets are optional.

. . . – Three dots indicate that the previous item can be repeated.

| – A vertical line between two parameters means “or“; only one of the parameters can be included.

For more information refer to the 9835A or 9845B Operating and Programming Manual.

## File Protection

Files can be read that are either unprotected or protected with the protect code “P”. Files protected with some other protect code cannot be used.

Only unprotected files can be overwritten. To prevent a file from being overwritten, always use the protect code “P”. For further information, refer to the Protect file statement in the Desktop Computer’s Operating and Programming manual.

# Specifications

LENGTH OF SOURCE FILE.

The length of the Source file depends upon:

- The size of the mass storage device.
- The maximum number of lines that can be input by the user. Maximum number 32767 lines.
- The line numbers generated by the Editor. Range 1 to 99999.

LENGTH OF SOURCE FILE LINES

The maximum line length is 80 characters. The line number requires 6 and the other 74 are available for Source code.

LENGTH OF OBJECT FILE.

The Object file must fit on one mass storage medium.

GENERAL

| | |
|---|---|
| Nesting levels for INCLUDE directive | 8 levels |
| Conditional ‘IF‘ nesting levels | 8 levels |
| Macro nesting levels | 8 levels |
| Serial Interface | RS 232C |
| Baud rate | 75–9600 |

# Equipment Requirements — Model 9835A

The Microprocessor Development Software package consists of:

09835–12531 Operating Manual.
7120–8903 An overlay illustrating the functions of the SFKs.
One, or all three of the following magnetic tape cartridges.
09835–12534 for the 6800 microprocessor type.
09835–12544 for the 8080/85 microprocessor type.
09835–12554 for the Z80 microprocessor type.

THE HARDWARE CONFIGURATION

To use the Microprocessor Development Software, you require:

HP 9835A Desktop Computer, Option 201. At least 115K bytes of memory.
HP 98331A Mass Storage ROM
HP 9885M Flexible Disk Drive or equivalent.
HP 9876A Thermal Printer, HP 2631A Hard Copy Printer or equivalent.
HP 98036A Serial Interface, Option 001 for use between the 9835A and a Microprocessor Development system.

A Microprocessor Development system may be connected for debugging purposes. It should:

- Contain the microprocessor,
- Contain a Read/Write memory that can hold the Object code you intend to debug,
- Have a monitor (mini operating system) driving RS 232C system console. (See Section IX, Console Module for further information).

The table below gives you the number of Source lines that may be entered into the Desktop Computer memory. As can be seen, a large amount of memory allows you to write more Source line without an overflow onto a mass storage device.

| Memory Size | Maximum number of Source lines that the Editor can hold before an overflow onto a mass storage device work file is necessary |
|---|---|
| 114 706 | 170 |
| 180 152 | 805 |
| 246 200 | 1440 |

# Equipment Requirements — Model 9845B

The Microprocessor Development Software package consists of:

09845–12531 Operating Manual.
7120–8904 An overlay illustrating the functions of the SFKs.
One, or all three of the following magnetic tape cartridges.
09845–12534 for the 6800 microprocessor type.
09845–12544 for the 8080/85 microprocessor type.
09845–12545 for the Z80 microprocessor type.

## THE HARDWARE CONFIGURATION

To use the Microprocessor Development Software, you need the following items:

HP 9845B Desktop Computer, option 204. At least 187K bytes of memory.

Optionally, any mass storage or printer supported by Hewlett-Packard for the HP 9845B Mainframe. Two mass storage channels are required. Recommended is one non-cartridge mass memory like the HP 9885M Flexible Disk Drive for reasons of speed. If the HP 9885M is used, then the HP 9845B must contain the 98413A Mass Storage ROM.

A Microprocessor Development System may be connected for debugging purposes. It should:

- Contain the microprocessor,
- Contain a Read/Write memory that can hold the Object code you intend to debug,
- Have a monitor (mini operating system) driving an RS 232C system console. (See Section Ten, Console Module for further information.
- Use the HP 98036A Serial Interface, Option 001, between the development system and the Desktop Computer.

The table below gives you the number of Source lines that may be entered into the Desktop Computer memory. As can be seen, a large amount of memory allows you to write more Source lines without an overflow onto a mass storage device.

| Memory Size | Maximum number of Source lines that the Editor can hold before an overflow onto a mass storage device workfile is necessary. |
|---|---|
| 187 146 | 805 |
| 318 026 | 2075 |
| 448 906 | 3345 |

# Section Two
# Quick Reference Guide

## Program Start

LOAD "AUTOST", 1 and press EXECUTE.

## Generation

Produces a copy that is optimised for your available memory size.

Symbol Reference Table.

A guide to the number of symbols you can enter in the Symbol table.

| Memory Size in Bytes | Answer to the question "Symbol Table Size" | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 500 | | 1000 | | 2000 | | 4094 | |
| | Symbol Refs | No of Macro Lines | Symbol Refs | No of Macro Lines | Symbol Refs | No of Macro Lines | Symbol Refs | No of Macro Lines |
| 9835A | | | | | | | | |
| 115 402 | 4839 | 100 | 2939 | 200 | – | – | – | – |
| 180 842 | 13019 | 100 | 11119 | 200 | 7319 | 400 | – | – |
| 246 282 | 21326 | 100 | 19426 | 200 | 15626 | 400 | 7677 | 818 |
| 9845B | | | | | | | | |
| 187 146 | 13830 | 100 | 11929 | 200 | 8130 | 400 | 181 | 818 |
| 318 026 | 30190 | 100 | 28290 | 200 | 24490 | 400 | 16541 | 818 |
| 448 906 | 32767 | 1430 | 32767 | 1345 | 32767 | 1174 | 32767 | 818 |

## Obtaining the Main Menu

Obtaining the Main Menu.

Type MASS STORAGE IS msus.
LOAD "AUTOST", 1

Press EXECUTE

MAIN MENU

```
Choose and then press one of the SFKs

     k0 for    INITIALIZATION
     k1 for    EDITOR
     k2 for    ASSEMBLER
     k3 for    CONSOLE

Press STOP to terminate the program
```

## Initialization

Allows you to change the default values already stored in the default file of your mass storage device.

## Editor

Allows you to write Source code.

### Special Function Keys (SFK)

For Syntax conventions see Section One.

**BREAK** Allows use of the Mainframe functions.

**CHANGE** "< STR > " **TO** "< STR2 > " [ **FROM** < LNA > [ , < LNB > ]] Allows String substitution.

**COPY** < LNA > , < LNB > **TO** < LNC > Copies one range of lines to another.

**DELETE** < LNA > , < LNB > Deletes all lines between and including those defined.

**EDIT** [ < LN > [ , < INCREMENT > ]] Enters Edit mode. Specifies the line number and the line increment.

**EDIT KEY** Permits the use of some SFKs for user-defined typing aids.

**EXIT** Returns you to the Main Menu.

**FIND** "< STR > " [ **FROM** < LNA > [ , < LNB > ]] Searches the Source code for a defined String.

**JOIN** "< FILE NAME > [ msus ] " [ < LN > ] [ @ ] Links programs from a mass storage device onto a program in memory.

**LIST** [ # < SC > ; ] [ < LNA > [ , < LNB > ]] [ @ ] Lists defined parts of the program. # < SC > ; specifies select code for a different printer. Symbol @ prints code without line numbers.

**LOAD** "< FILE NAME > [ msus ] " [ @ ] Transfers the Source code from a known device into memory. Symbol @ generates line numbers.

**RENUMBER** [ < LN > , [ < INCREMENT > ]] Modifies line numbering.

**SCRATCH** Deletes contents of Editor memory including working files.

**STORE** " < FILE NAME > [ msus ] " [ < LNA > [ , < LNB > ]] [ @ ] Stores the Source code as a data file on a given device. Symbol @ stores files without line numbers.

**SYNTAX** Allows Editor to make an automatic Syntax check.

## Mainframe Keys

Model 9845B

CLEAR → END
CLEAR LINE
DEL CHR
DEL LN
HOME
INS CHR
ROLL (up arrow)
ROLL (down arrow)
PRT ALL
TAB CLEAR
TAB SET
TYPWTR
Up arrow
Down arrow
Right arrow
Left arrow

Model 9835A

BACK
CLR → END
CLEAR LINE
DEL CHR
DEL LN
FWD
HOME
INS CHR
PRT ALL
TAB CLEAR
TAB SET
TO END
TYPWTR
Up arrow
Down arrow
Right arrow
Left arrow

Inactive keys in Edit mode.

STEP    RUN    RESULT    PAUSE    CLEAR    SPACE DEPENDENT

Keys that do the same operation.

CONTINUE and EXECUTE except when BREAK is pressed.

Keys whose function change between the Edit and Command mode.

CONTINUE/EXECUTE RECALL

Keys which have a different function to the mainframe

CLEAR LINE HOME INS LN STORE down arrow

HOME/SHIFT (9845B)

SHIFT down arrow (9835A) SHIFT up arrow (9835A)

## Assembler

Translates your Source code into an absolute machine code.

**SOURCE** "< FILE NAME > [ msus ] " [ , < CONTROL1 > ] [ , < CONTROL2 > . . . ]

Specifies the name of the Source file to be assembled.

**OBJECT** [ " < FILE NAME > [ msus ] " ] [ , < CONTROL1 > ] [ ,< CONTROL2 > . . . ]

Specifies the file name where the Object code is to be stored. If Object code need not be stored, press CONTINUE.

**BREAK** Allows a temporary use of the Mainframe functions.

**EXIT** Returns you to the Main Menu.

## Macros

Defines a body of text that is automatically inserted into the Source code each time a Macro is called.

< Macro name > **MACRO** [ = < Dummy Parameters > . . . ] [ < COMMENT > ] Defines Macro

[ < LABEL > ] < Macro Name > [ < Actual > ] , [ < Actual > . . . ] [ < COMMENT > ] Calls Macro

[ < LABEL > ] **ENDM** [ < COMMENT > ] Terminates a Macro expansion.

[ < LABEL > ] **EXITM** [ < COMMENT > ] Terminates the current expansion level

[ < LABEL > ] **IRP** = < DUMMY > , < ACTUAL > [ , < ACT > . . . ] [ < COMMENT > ]

Repeats the code sequence between IPR and the ENDM directive for every actual parameter.

[ < LABEL > ] **IRPC** = < DUMMY > , < ACTUAL > [ < COMMENT > ]

Repeats the code sequence between IRPC and the ENDM directive for every character in the actual parameter.

[ < LABEL > ] **REPT** < EXPRESSION > [ < COMMENT > ]

Repeats the code sequence between REPT and the ENDM directive by the number of times defined by the Expression

= Dummy parameter prefixes.

^ Pre-evaluation prefixes.

**=NUM** Dummy parameter that is replaced by a 4-digit number during Macro expansion.

**NOTE**
When using Macros, the syntax for the LABEL and COMMENT fields obey the rules of the particular microprocessor type concerned.

## Console Module

Provides the communication between the Desktop Computer and the Microprocessor system.

COMMANDS

**LOAD** " < FILE NAME > [ msus ] " Transfers file from the Desktop Computer to Microprocessor System

**STORE** " < FILE NAME > [ msus ] " Transfers data from Microprocessor System to Desktop Computer file.

**BREAK** Allows a temporary use of the Mainframe functions.

**EXIT** Returns you to the Main Menu.

Note. 8080/85 Conversion. A letter W in front of the Object file will convert the Object code format to that required by the Insert format.

Special Keys

| | ASCII char. | HEX. value |
|---|---|---|
| STORE | CR | 0D |
| SHIFT/STORE | LF | 0A |
| BACKSPACE | BS | 08 |
| TAB | HT | 09 |
| CLEAR LINE | DEL | 7F |

## Operators for the 8080/85

| | |
|---|---|
| + | Addition. |
| – | Subtraction. |
| * | Multiplication. |
| / | Integer Division. Any remainder is discarded (7/2=3). |
| MOD | Modulo. Result is the remainder caused by a division operation. |
| ( ) | Parenthesized expressions override any precedence. |
| SHR | Shift to the right. |
| SHL | Shift to the left. |
| HIGH | Isolate High Order 8 Bits of 16-bit value. |
| LOW | Isolate Low Order 8 Bits of 16-bit value. |
| NOT | Logical One's Complement. |
| AND | Logical AND. |
| OR | Logical OR. |
| XOR | Logical Exclusive OR. |
| EQ | Equal. |
| NE | Not Equal. |
| LT | Less Than. |
| LE | Less Than or Equal. |
| GT | Greater Than. |
| GE | Greater Than or Equal. |

## Operators for the 6800

| | |
|---|---|
| + | Addition |
| – | Subtraction |
| * | Multiplication |
| / | Integer Division |
| ( ) | Parenthesized expressions override any precedence. |

## Operators for the Z80

| | | |
|---|---|---|
| + | | Addition |
| – | | Subtraction |
| * | | Integer Multiplication |
| / | | Integer Division. Any Remainder is disregarded (7/2=3) |
| .MOD. | | Modulo. Result is the remainder caused by a div. operation. |
| ( ) | | Parentheized expression overrides any precedence |
| .SHR. | | Shift Operand to the right |
| .SHL. | | Shift Operand to the left |
| .NOT. | \ | Logical One's Complement |
| .AND. | & | Logical AND |
| .OR. | | Logical OR |
| .XOR. | | Logical Exclusive OR |
| .EQ. | | Equal |
| .LT. | < | Less Than |
| .GT. | > | Greater Than |
| .UGT. | | Unsigned Greater Than |
| .ULT. | | Unsigned Less Than |

## 8080/85 Assembler Language Instructions

### Assembler Directives

| | | | | |
|---|---|---|---|---|
| [ < LABEL >: ] | **DB** | < EXPR > \| < STRING > [ , < EXPR > \| < STRING > . . . ] [ ; < COMMENT > ] | | Define Byte. |
| [ < LABEL >: ] | **DS** | < EXPRESSION > | [ ; < COMMENT > ] | Define Storage. |
| [ < LABEL >: ] | **DW** | <EXPRESSION > [ , < EXPRESSION > . . . ] [ ; < COMMENT > ] | | Define word |
| [ < LABEL >: ] | **END** | [ < START ADDRESS > ] [ ; <COMMENT > ] | | Defines End of Source code. |
| < NAME > | **EQU** | < EXPRESSION > | [ ; < COMMENT > ] | Equates a symbol to an Operand. It may only be used once in a program with the same Name/Label. |
| <LABEL >: | **EQU** | < EXPRESSION > | [ ; < COMMENT > ] | |
| [ < LABEL >: ] | **FAIL** | 'USER DEFINABLE ERROR MESSAGE' | | Forces Assembler to print a user defined error message. |
| [ < LABEL >: ] | **IF** | < EXPRESSION > | [ ; < COMMENT > ] | Code sequence following IF directive is assembled if least sign. bit of expression equals 1. |
| [ < LABEL >: ] | **IFC** | < OPER1 >, < OPER2 > | [ ; < COMMENT > ] | Code sequence following IFC directive is assembled if Operands are equal. |
| [ < LABEL >: ] | **IFNC** | < OPER1 >, < OPER2 > | [ ; < COMMENT > ] | Code sequence following IFNC direcitve is assembled if Operands are not equal. |
| [ < LABEL >: ] | **ELSE** | | [ ; < COMMENT > ] | Separates the IF, IFC, or IFNC directive from the ENDIF directive in order that one of two sequences can be defined. |
| [ < LABEL >: ] | **ENDIF** | | [ ; < COMMENT > ] | Used with IF, IFC, IFNC directives to end a sequence. |
| [ < LABEL >: ] | **ORG** | < EXPRESSION > | [ ; < COMMENT > ] | Assigns origin of location counter. |
| < NAME > | **SET** | < EXPRESSION > | [ ; < COMMENT > ] | Sets a symbol to an Operand. SET is redefinable in the same program. |
| < LABEL >: | **SET** | < EXPRESSION > | [ ; < COMMENT > ] | |
| [ < LABEL >: ] | **SPC** | < EXPRESSION > | [ ; < COMMENT > ] | Program listing is spaced < expression > lines. |

## Assembler Controls

$[ CONTROL [ , CONTROL . . . ] ]
Gives optional control of the output format with one or more of the following:

**TITLE "ASCII String"**
Program heading

**EJECT**
Moves paper to the top of the next page.

**SOURCE "< FILE NAME > [ msus ] "**
A file on the mass storage, specified in the Operand field of this directive, is included in the assembly.

**INCLUDE "< FILE NAME > [ msus ] "**
Same as for SOURCE.

**OBJECT** [ "< FILE NAME > [ msus ] " ]
Specifies in which file the Object code is to be stored.

**SYMBOLS**
Prints the Symbol Table after the program listing.

***NOSYMBOLS**
Suppresses Symbol Table print.

***XREF**
Prints the cross reference of all user symbols. Only one list is printed if XREF and SYMBOLS have been specified.

**NOXREF**
Suppresses the cross reference.

**SAVE**
Current list control settings LIST, COND, and GEN are stored and remain valid until explicitly changed.

**RESTORE**
Recalls the list control settings.

***LIST**
Instructs the Assembler to generate a program listing.

**NOLIST**
Suppresses the printing of the program listing. Error messages appear on the CRT.

***COND**
Includes conditional skipped Source lines in the listing.

**NOCOND**
Does not include conditionally skipped lines in the listing.

***GEN**
Lists MACRO expansion.

**NOGEN**
Does not list MACRO expansion.

*Marked controls show default settings if others are not specified.

# 6800 — Assembler Language Instructions

## Assembly Directives

| | | | | |
|---|---|---|---|---|
| [ < LABEL > ] | **END** | | [ < COMMENT > ] | Defines the end of a Source code. |
| < LABEL > | **EQU** | < EXPRESSION > | [ < COMMENT > ] | Equates a symbol to an Operand. May only be used once in a program. |
| [ < LABEL > ] | **FAIL** | 'USER DEFINABLE ERROR MESSAGE' | | Forces the Assembler to print user defined error messages. |
| [ < LABEL > ] | **FCB** | < EXPR > [ , < EXPR > . . . ] | [ < COMMENT > ] | Form constant byte. |
| [ < LABEL > ] | **FCC** | /TEXT/ | [ < COMMENT > ] | Form constant character. |
| [ < LABEL > ] | **FCC** | COUNT, TEXT | | |
| [ < LABEL > ] | **FDB** | < EXPR > [ , < EXPR > . . . ] | [ < COMMENT > ] | Form double constant byte. |
| [ < LABEL > ] | **IF** | < EXPRESSION > | [ < COMMENT > ] | Code sequence following IF is assembled if expression evaluates to a value other than zero. |
| [ < LABEL > ] | **IFC** | < OPER1 > , < OPER2 > | [ < COMMENT > ] | Code sequence following IFC is assembled if Operands are equal. |
| [ < LABEL > ] | **IFNC** | < OPER1 > , < OPER2 > | [ < COMMENT > ] | Code sequence following IFNC is assembled if Operands are not equal. |
| [ < LABEL > ] | **ELSE** | | [ < COMMENT > ] | Separates the IF, IFC, or IFNC directive from the ENDIF directive in order that one of two sequences can be defined. |
| [ < LABEL > ] | **ENDIF** | | [ < COMMENT > ] | Used with the IF, IFC, and IFNC directives to end a sequence. |
| [ < LABEL > ] | **MON** | < START ADDRESS > | [ < COMMENT > ] | Used when the Microprocessor system requires the Start Address in Object code. |
| [ < LABEL > ] | **NAM** | PAGE HEADING | | Defines the page heading. |
| [ < LABEL > ] | **ORG** | < EXPRESSION > | [ < COMMENT > ] | Assigns origin of location counter. |
| [ < LABEL > ] | **PAGE** | | [ < COMMENT > ] | Moves paper to top of next page. |
| [ < LABEL > ] | **RMB** | < EXPRESSION > | [ < COMMENT > ] | Reserve memory bytes. |
| < LABEL > | **SET** | < EXPRESSION > | [ < COMMENT > ] | Sets a symbol to an Operand. SET is redefinable. |
| [ < LABEL > ] | **SPC** | < EXPRESSION > | [ < COMMENT > ] | Program listing is spaced < expression > lines. |

## Assembler Controls.

[ < LABEL > ]  **OPT**  [ < CONTROL > [ , < CONTROL > . . . ] ]

Gives optional control of output format with one or more of the following:

**SOURCE** " < FILE NAME > [ msus ] "  A file on the mass storage, specified in the Operand field with this directive, is included in the assembly.

**OBJECT** [ " < FILE NAME > [ msus ] " ]  Specifies in which file the Object code is to be stored.

**SYMBOL**  Prints the Symbol Table after the program listing.

***NOSYMBOL**  Suppress Symbol Table print.

***XREF**  Prints the cross reference of all user symbols. Only one list is printed if **XREF** and **SYMBOL** have been specified.

**NOXREF**  Suppresses the cross reference.

**SAVE**  Current list of control settings LIST, COND, GEN, and MEX are stored and remain valid until explicitly changed.

**RESTORE**  Recalls the list control settings.

***LIST**  Instructs the Assembler to generate a program listing.

**NOLIST**  Suppresses the printing of the program listing.

***COND**  Includes conditional skipped Source lines in the listing.

**NOCOND**  Does not include conditionally skipped lines.

***MEX**  Lists MACRO expansion.

**NOMEX**  Does not list MACRO expansion.

***GEN**  Lists all code generated by the FCC directive.

**NOGEN**  Lists only the first line generated by the FCC directive.

*Marked controls show default settings if others are not specified.

## Z80 — Assembler Language Instructions

### Assembler Directives

| | | | | |
|---|---|---|---|---|
| [<LABEL>:] | **COND** | <EXPRESSION> | [;<COMMENT>] | Code sequence following COND is assembled if the expression evaluates to a value other than zero. |
| [<LABEL>:] | **DEFB** | <EXPR>\|<STRING>[, <EXPR>\|<STRING>... ] | [;<COMMENT>] | Defines byte. |
| <LABEL>: | **DEFL** | <EXPRESSION> | [;<COMMENT>] | Defines a Label to an Operand. Is redefinable in same program. |
| [<LABEL>:] | **DEFM** | <EXPR>\|<STRING>[,<EXPR>\|<STRING>... ] | [;<COMMENT>] | Defines memory. |
| [<LABEL>:] | **DEFS** | <EXPRESSION> | [;<COMMENT>] | Defines storage |
| [<LABEL>:] | **DEFW** | <EXPRESSION>[,<EXPRESSION>... ] | [;<COMMENT>] | Defines word. |
| [<LABEL>:] | **ELSE** | | [<;COMMENT>] | Separates the COND, IFC, or IFNC directives from the ENDC directive in order that one of the two sequences can be defined. |
| [<LABEL>:] | **ENDC** | | [;<COMMENT>] | Used with COND, IFC and IFNC directives to end a sequence. |
| [<LABEL>:] | **END** | [<START ADDRESS>] | [;<COMMENT>] | Defines the end of a Source code. |
| <LABEL>: | **EQU** | <EXPRESSION> | [;<COMMENT>] | Equates a symbol to an Operand. May only be used once for the same Label. |
| [<LABEL>:] | **FAIL** | 'USER DEFINABLE ERROR MESSAGE' | | Forces the Assembler to print user-defined error messages. |
| [<LABEL>:] | **IFC** | <OPER1>,<OPER2> | [;<COMMENT>] | Code sequence following IFC assembled if Operands are equal. |
| [<LABEL>:] | **IFNC** | <OPER1>,<OPER2> | [;<COMMENT>] | Code sequence following IFNC if assembled if Operands are not equal. |
| [<LABEL>:] | **ORG** | <EXPRESSION> | [;<COMMENT>] | Assigns origin of location counter. |
| [<LABEL>:] | **SPC** | <EXPRESSION> | [;<COMMENT>] | Program listing is spaced <expression> lines. |

## Assembler Controls.

* [ CONTROL [ , CONTROL . . . ] ] Gives optional control of output format with one or more of the following:

**HEADING** "ASCII String" Program heading.

**EJECT** Advances paper to the top of the next page.

**SOURCE** " < FILE NAME > [ msus ] " A file on the mass storage, specified in the Operand file of this directive, is included in the assembly.

**INCLUDE** " < FILE NAME > [ msus ] " Same as for SOURCE.

**OBJECT** [ "FILE NAME [ msus ] " ] Specifies in which file the Object code is to be stored.

**SYMBOLS ON** Prints the Symbol Table after the program listing.

***SYMBOLS OFF** Suppresses Symbol Table print.

***XREF ON** Prints the cross reference of all user symbols. Only one list is printed if XREF and SYMBOLS ON have been specified.

**XREF OFF** Suppresses the cross reference.

***LIST ON** Instructs the Assembler to generate a program listing.

**LIST OFF** Suppresses the printing of the program listing.

**SAVE** Current list control settings LIST, CONDLIST, and MACLIST are stored and remain valid until explicitly changed.

**RESTORE** Recalls the list control settings.

***CONDLIST ON** Includes conditional skipped Source lines in the listing.

**CONDLIST OFF** Does not include conditionally skipped lines in the listing.

***MACLIST ON** Lists MACRO expansion.

**MACLIST OFF** Does not list MACRO expansion.

*Marked controls show default settings if others are not specified.

# Section Three
# Generation of the Program Cartridge

## Introduction

The tape cartridge supplied to you by Hewlett-Packard will not run on your machine in its existing condition. It must first of all be copied (Initialized) onto a second mass storage device. This could be either another cartridge or a flexible disc. This copy would run on the Desktop Computer. If desired, the copy can then be re-copied onto another empty cartridge in the primary tape drive (:T15).

Cartridge generation is made so that the copy produced is optimized for your available memory size. Hence, after generation the cartridge will only run correctly on a Desktop Computer that has the identical amount of memory as the machine on which it was made.

**CAUTION**

Ensure that the Desktop Computer and the selected mass storage device are each set up according to their installation procedures. This includes power switch settings, grounding requirements, and installation of the proper fuses. Refer to the appropriate manuals for installation instructions. The power must be off on both units before the interface cable is connected. Failure to comply with this caution could result in damage to the equipment.

## Conventions

If you have any difficulty with the conventions used in this section, refer to Section One for a complete explanation.

# Operating Procedure for Producing a Copy

**NOTES**

If, during the generation procedure an error occurs you must start the procedure again. Hewlett-Packard cannot guarentee that the program will work correctly if this is not done.

A generated cartridge will only work on a Desktop Computer that contains the identical amount of memory as the machine on which it was made.

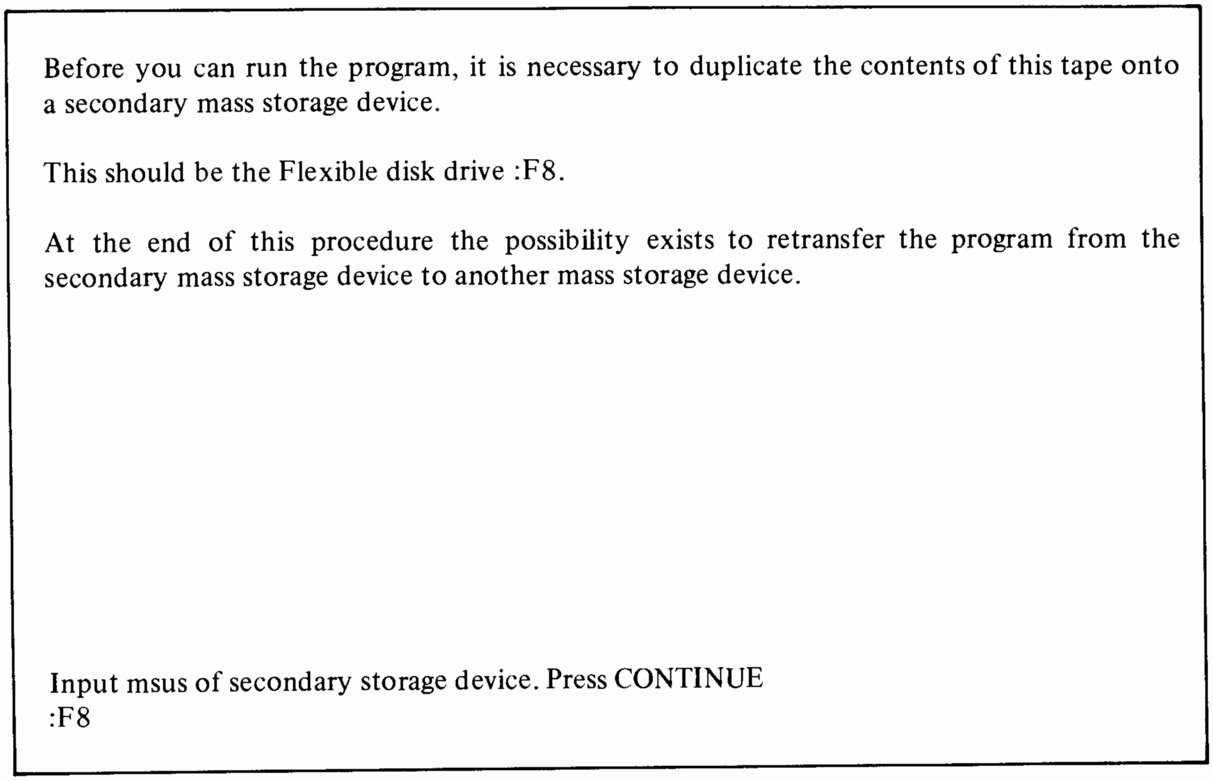
The following procedure is basically identical for both the 9835A and 9845B Desktop Computers.

To simplify the presentation, the presented menus have been taken from the 9835A CRT while the 9845B messages, though similar, are not shown. Minor differences between the Desktop Computer messages are noted.

Insert the supplied program cartridge into the standard tape drive of your Desktop Computer. Ensure that the secondary mass storage device (:T14, :F8, or other) contains the necessary storage medium.

Load the program with the automatic start capability of the Desktop Computer. Alternatively, type LOAD "AUTOST", 1 and press EXECUTE.

The following message is displayed on the 9835A CRT.



Before you can run the program, it is necessary to duplicate the contents of this tape onto a secondary mass storage device.

This should be the Flexible disk drive :F8.

At the end of this procedure the possibility exists to retransfer the program from the secondary mass storage device to another mass storage device.

Input msus of secondary storage device. Press CONTINUE
:F8

The 9845B Desktop Computer message reads:

```
Before you can run the program, it is necessary to duplicate the contents of this tape onto
a secondary mass storage device.

This can be either:            Optional tape cartridge :T14, or
                               Flexible disk drive :F8

At the end of this procedure the possibility exists to retransfer the program from the
secondary magnetic storage device to another mass storage device.

If :T14 is used, a blank Initialized cartridge must be installed.







Input msus of secondary storage device. Press CONTINUE
:T14
```

Pressing CONTINUE puts the entered value in the middle of the CRT and the bottom two lines display the next question.

```
Input max. symbol table size (Default is 1000). Press CONTINUE
1000
```

It is here that you must decide the number of symbols required for the symbol table and the number of entries needed in the X-ref table. The amount of memory available must be divided between the two. One more factor to bear in mind is the space required if you are going to make use of Macros.

Knowing the maximum number of lines in your Source code, divide by 5 to give the approximate number of symbols you require. This number will vary considerably from user to user. If, from experience, you know that you use a lot of symbols then enter a high number. Conversly, enter a low number when only a small amount of symbols are going to be used.

Depending upon the number you enter, the program allocates a number of Macro lines. In most cases, this is the Symbol table number divided by 5. Having done this, some of the available memory space is now occupied by the Symbol table and Macro storage. The remainder is used for symbol references.

The X-ref program must also store in memory the information as to where the symbols are defined and referenced. If there is insufficient space to store all the symbol references, this will not affect the operation of the Assembler program but will merely leave some empty entries in the X-ref list. An error message is given in this case.

You should always enter more than sufficient symbols to ensure correct operation of the Assembler program as missing entries in the X-ref table are not a serious handicap.

For example, if you know that your program will be 5000 Source lines long then 1000 symbols is the number you need. Entering 1000 gives you space for 200 Macro lines. The remaining amount of memory is used for symbol references.

The reference table below, gives you a guide line as to the number you should enter in the CRT.

Symbol Reference Table

| Memory Size in Bytes | Answer to the question "Symbol Table Size" | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 500 | | 1000 | | 2000 | | 4094 | |
| | Symbol Refs | No of Macro Lines | Symbol Refs | No of Macro Lines | Symbol Refs | No of Macro Lines | Symbol Refs | No of Macro Lines |
| 9835A | | | | | | | | |
| 115 402 | 4839 | 100 | 2939 | 200 | – | – | – | – |
| 180 842 | 13019 | 100 | 11119 | 200 | 7319 | 400 | – | – |
| 246 282 | 21326 | 100 | 19426 | 200 | 15626 | 400 | 7677 | 818 |
| 9845B | | | | | | | | |
| 187 146 | 13830 | 100 | 11929 | 200 | 8130 | 400 | 181 | 818 |
| 318 026 | 30190 | 100 | 28290 | 200 | 24490 | 400 | 1·6541 | 818 |
| 448 906 | 32767 | 1430 | 32767 | 1345 | 32767 | 1174 | 32767 | 818 |

**NOTE**

The figures in the above table may vary slightly depending upon your particular system configuration.

Pressing CONTINUE puts the answers to all previous questions on the CRT and another message is given.

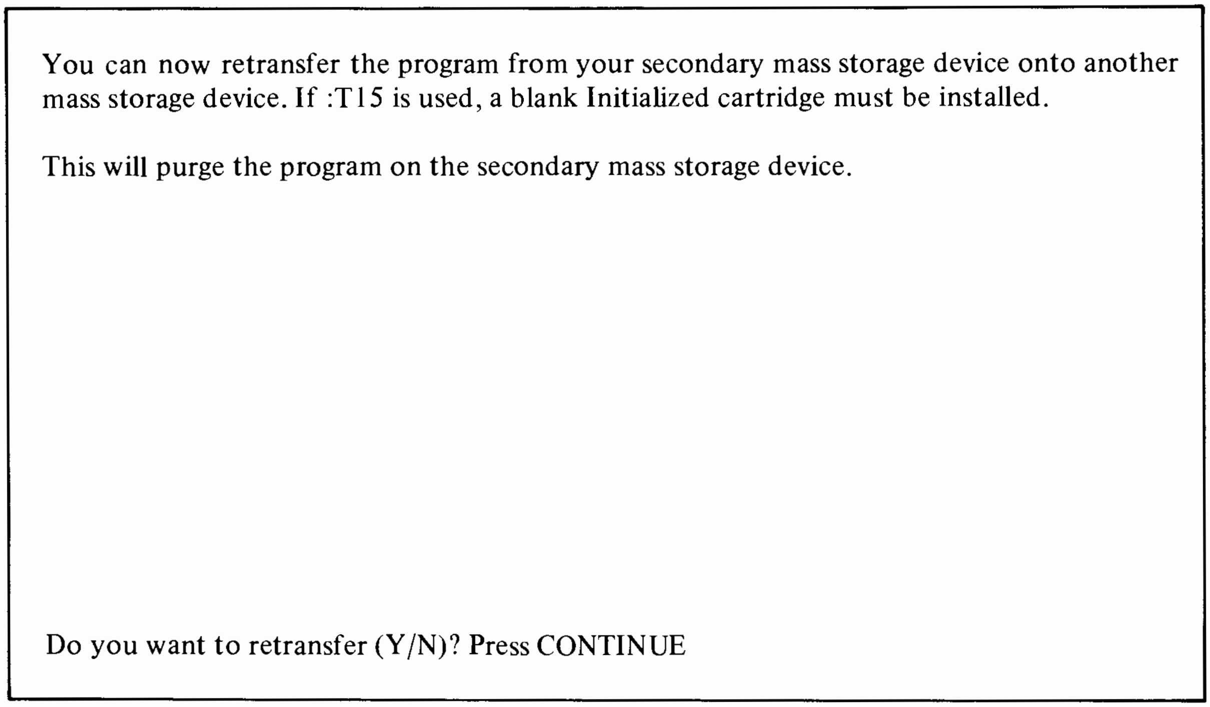
Ensure secondary storage device is ready. Press CONTINUE.

The CRT displays:

Your program cartridge is being duplicated.

Remember!
A generated cartridge will only work on a Desktop computer that contains the identical amount of memory as the machine on which it was made.

After duplication has been made a further message appears



You can now retransfer the program from your secondary mass storage device onto another mass storage device. If :T15 is used, a blank Initialized cartridge must be installed.

This will purge the program on the secondary mass storage device.

Do you want to retransfer (Y/N)? Press CONTINUE

To answer with a N (NO), a message informs you on how to run the program.

You may want to use a mass storage other than your present device. In this case, answering with a Y (YES) rewinds the original program cartridge for removal and you are asked if you would like to retransfer your program to some other mass storage device.

Answering YES brings-up the following message:

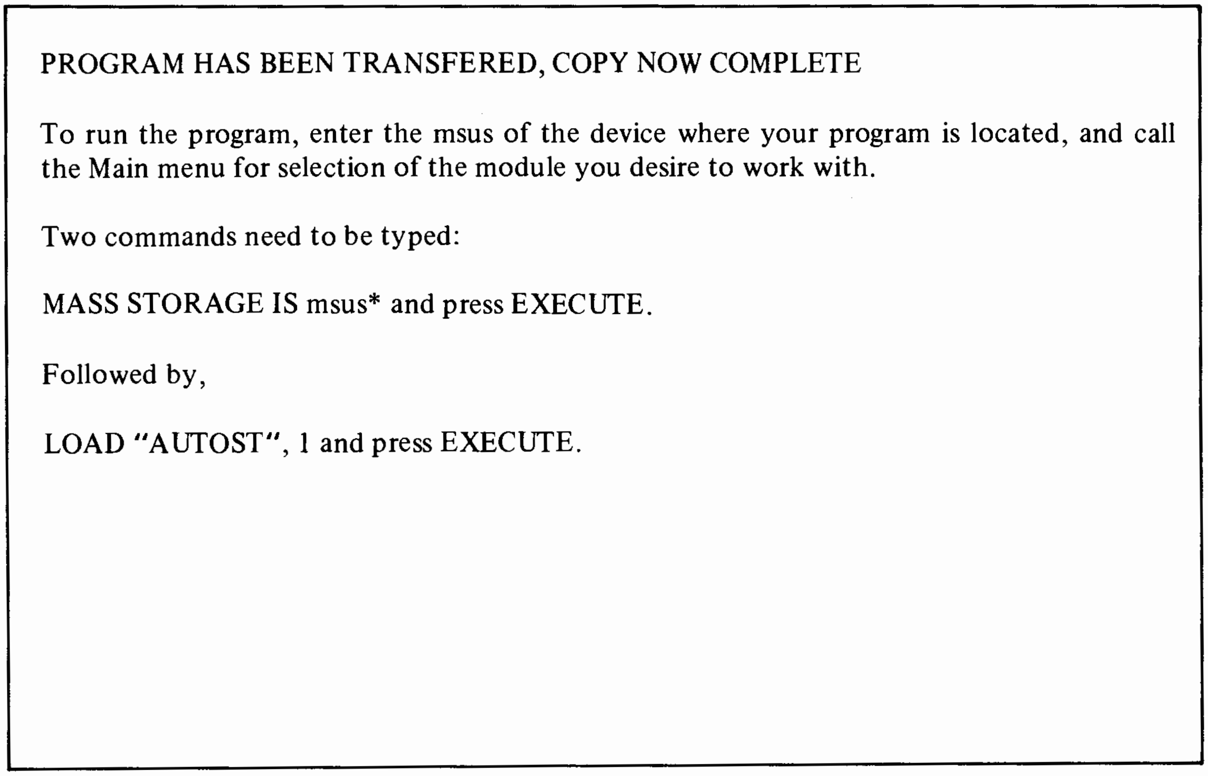
Input msus of the new storage device. Press CONTINUE.
:T15

Enter appropriate msus. Press CONTINUE.

The next message informs you that the retransfer is taking place.

Your program is being retransfered to the storage device :T15

When duplication is finished, or if you had previously answered NO to the retransfer question, the following message appears.



```
PROGRAM HAS BEEN TRANSFERED, COPY NOW COMPLETE

To run the program, enter the msus of the device where your program is located, and call
the Main menu for selection of the module you desire to work with.

Two commands need to be typed:

MASS STORAGE IS msus* and press EXECUTE.

Followed by,

LOAD "AUTOST", 1 and press EXECUTE.
```

**NOTE**

*The msus in the above menu is replaced, on your CRT, by the actual device select code where your program is located. This is the device select code you must type with the above commands.

The Main Menu appears

# Section Four
# Obtaining the Main Menu

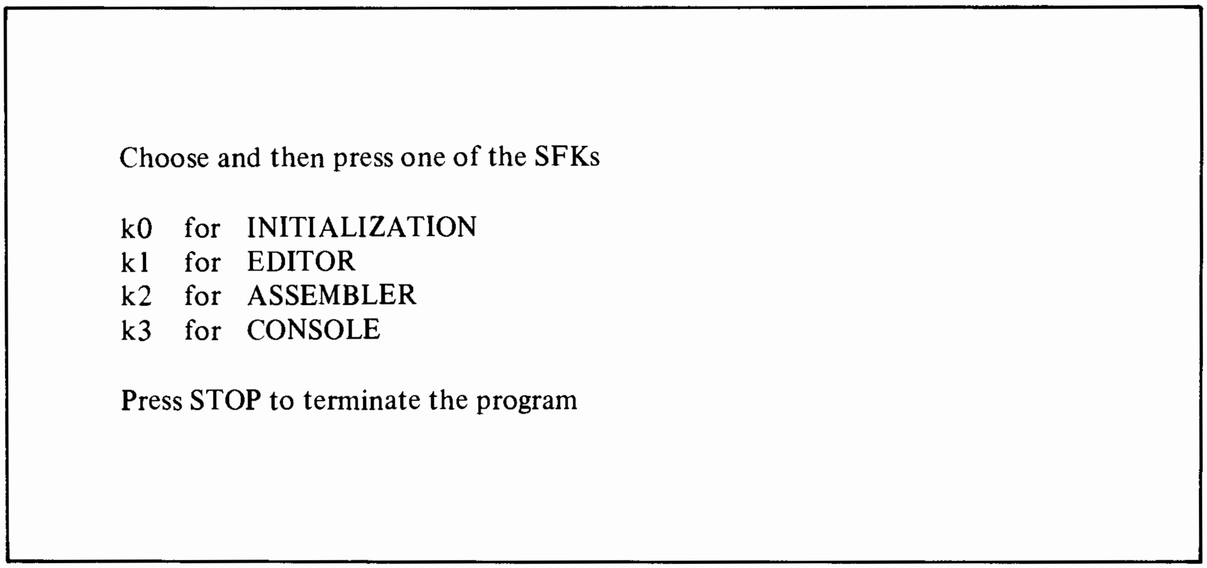
## Program Start

Type MASS STORAGE IS msus* and press EXECUTE.

Load the file with LOAD "AUTOST", 1 and press EXECUTE.

The Main Menu appears.

MAIN MENU



```
Choose and then press one of the SFKs

k0   for   INITIALIZATION
k1   for   EDITOR
k2   for   ASSEMBLER
k3   for   CONSOLE

Press STOP to terminate the program
```

The above four modules are further explained in the following manual Sections.

NOTES

The SFK BREAK can be used to leave any of the above modules to perform an operation with the Desktop Computer such as a catolog or a mathematical calculation. The key CONTINUE brings you back to the module you were working with before pressing SFK BREAK.

The msus in the above instruction should be replaced by the actual mass storage unit specifier where your program is located.

# Section Five
# Initialization Module

## Introduction

Special Function Keys (SFK) allow you to change the peripheral default values that are already stored in the default file. Each time a Menu appears on the CRT, the program waits until you press one of the SFK's.

Inputs, sent to the program by pressing CONTINUE are checked for syntax correctness and you are informed of any errors that may exist in your input.

The SFK BREAK can be used to leave this module to perform an operation with the Desktop Computer such as a catolog or a mathematical calculation. The CONTINUE key brings you back to the Initialization module.

### Conventions

If you have any difficulty with the conventions used in this section, refer to Section One for a complete explanation.

## The Menus

Selecting SFK k0 from the Main menu results in the Initialization Menu appearing on the CRT.

```
                 INITIALIZATION MENU

Choose and then press one of the SFKs

k0           for   PRINT-OUT of DEFAULTS
k1           for   PRINTER
k2           for   EDITOR and ASSEMBLER
k3           for   CONSOLE
k5 (EXIT)    for   INITIALIZATION completed
```

On the 9845B, press k7 (EXIT) when Initialization is complete.

Pressing SFK k0 produces a print-out of default values on the printer.

**When you run this module for the first time ensure that the select code for the printer has been set correctly.**

```
                      PRINT-OUT OF DEFAULT VALUES
                                              System 9835    System 9845
PRINTER

PRINTER SELECT CODE ......................... 7,1            ..0
WIDTH ....................................... 80             ..80
NUMBER OF LINES EACH PAGE.................... 70             ..70
PAPER IS PERFORATED ......................... Y              ..Y
FORMATTED LISTING............................ Y              ..Y

EDITOR AND ASSEMBLER

msus FOR EDITOR WORK FILES................... :F8            ..N
msus FOR SOURCE FILE ........................ :F8            ..:T14
msus FOR OBJECT FILE......................... :F8            ..:T14

CONSOLE

msus FOR LOAD FILE........................... :F8            ..:T14
msus FOR STORE FILE.......................... :F8            ..:T14
SELECT CODE FOR SERIAL INTERFACE ............ 11             ..11

SERIAL INTERFACE RS 232C/V24

     BAUD RATE .............................. 9600           ..9600
     DATA WORD LENGTH........................ 8              ..8
     PARITY CHECK............................ N              ..N
     STOP BIT ............................... 1              ..1
     *INTEL FORMAT CONVERSION................ Y              ..Y
```

The two right-hand columns list the default values for your system. Both columns are shown here however, only the column appropriate to your system will appear on the print-out. After the print-out has been made the program returns to the Initialization Menu.

**NOTE**

*The "INTEL FORMAT CONVERSION" only appears on the 8080/85 program (refer to the Console module for more information).

To make changes in the following Menus, select the appropriate SFK and the given default value appears in the display line of the CRT. Make any necessary changes before pressing CONTINUE to put the new value back in the menu.

Pressing SFK k1, on the Initialization Menu, results in the following Printer Menu appearing on the screen.

```
                              PRINTER

    To change default values, press appropriate SFK

         k0   for      PRINTER SELECT CODE                          7,1

         k1   for      WIDTH                                        80
                       (min 60 and max 260)

         k2   for      NUMBER of LINES each page                    70
                       (min 40 and max 999)

         k3   for      PAPER is PERFORATED                          Y
                       (possibilities Y=yes/N=no)

        *k4   for      FORMATTED LISTING                            Y
                       (possibilities Y=yes/N=no)

         k5   (EXIT)   for retrieving the Initialization Menu
```

If changes are necessary, select one of the above SFKs and the displayed default value appears at the bottom of the CRT.

**NOTE**

*Only answer Y (yes) to this question if your printer can execute form feeds.

After making the change, press CONTINUE and the correct statement is written in the menu. Further changes are made in the same way after pressing the appropriate SFK.

Press k5 EXIT (k7 on the 9845B) to retrieve the Initialization Menu.

To obtain the Editor and Assembler Menu, select SFK k2

```
                         EDITOR AND ASSEMBLER

    To change default values, press appropriate SFK.

        *k0   for      msus for EDITOR WORK FILES                  N
                       (possibilities msus/N=none)

         k1   for      msus for SOURCE FILE                        :F8

         k2   for      msus for OBJECT FILE                        :F8

         k5   (EXIT)   for retrieving the Initialization Menu
```

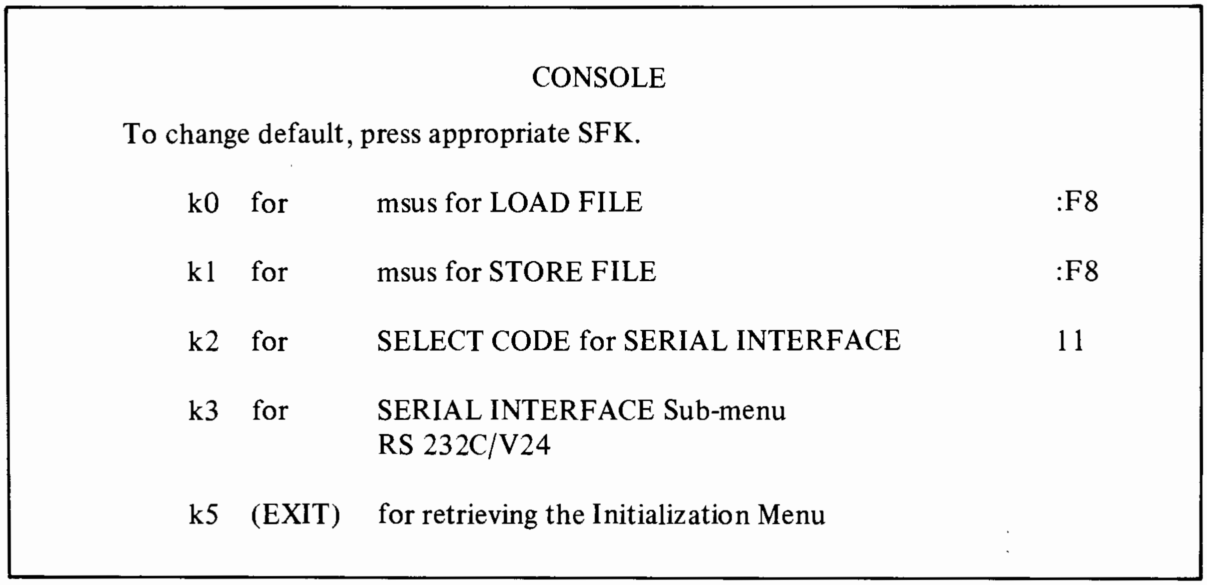
**NOTE**

*The msus for the "Editor work files" is the mass storage device that the Editor is working with if the Source lines cannot be held in memory. Enter N if you do not wish to use work files. The maximum number of Source lines is then limited to the amount shown in the table that lists Source lines with respect to memory size (Equipment requirement paragraph, Section One).

After selection of the appropriate SFKs, change the default values and press CONTINUE to enter the information on the menu.

Press k5 EXIT (k7 on the 9845B) to retrieve the Initialization Menu.

To obtain the CONSOLE Menu, select SFK k3



```
                              CONSOLE

    To change default, press appropriate SFK.

        k0   for      msus for LOAD FILE                         :F8

        k1   for      msus for STORE FILE                        :F8

        k2   for      SELECT CODE for SERIAL INTERFACE           11

        k3   for      SERIAL INTERFACE Sub-menu
                      RS 232C/V24

        k5   (EXIT)   for retrieving the Initialization Menu
```

After selection of the appropriate SFKs, change the default values and press CONTINUE to enter the information on the menu.

Press k5 EXIT (k7 on the 9845B) to retrieve the Initialization Menu.

Selecting the SFK k3 brings-up a further menu for the “Serial Interface RS 232C/V24“.

```
                SERIAL INTERFACE RS 232C/V24 SUB-MENU

    To change default values, press appropriate SFK.

    k0    for      BAUD RATE ............................... 9600
                   (possibilities 75/110/150/300/
                   600/1200/1800/2400/4800/9600)

    k1    for      DATA WORD LENGTH......................... 8
                   (possibilities 5/6/7/8)

    k2    for      PARITY CHECK............................. N
                   (possibilities E=even/O=odd/N–none)

    k3    for      STOP BIT ................................ 1
                   (possibilities 1/1.5/2)

   *k4    for      INTEL FORMAT CONVERSION.................. Y
                   (possibilities Y=yes/N=no)

    k5    (EXIT)   for retrieving the CONSOLE menu
```

After selection of the appropriate SFKs, change the default values to one of the possibilities given. Press CONTINUE to enter the information on the menu.

---

**NOTE**

*The k4 statement in the above menu only appears on the 8080/85 program menu.

---

At this stage, the Initialization is complete.

Press EXIT k5 (k7 on the 9845B) twice to return to the Initialization Menu.

# Section Six
# The Editor Module

## Introduction

The Editor allows you to generate, modify, and correct Source code. The working field of the Editor is the CRT. Writing and editing is performed by use of the cursor controlled by the display keys.

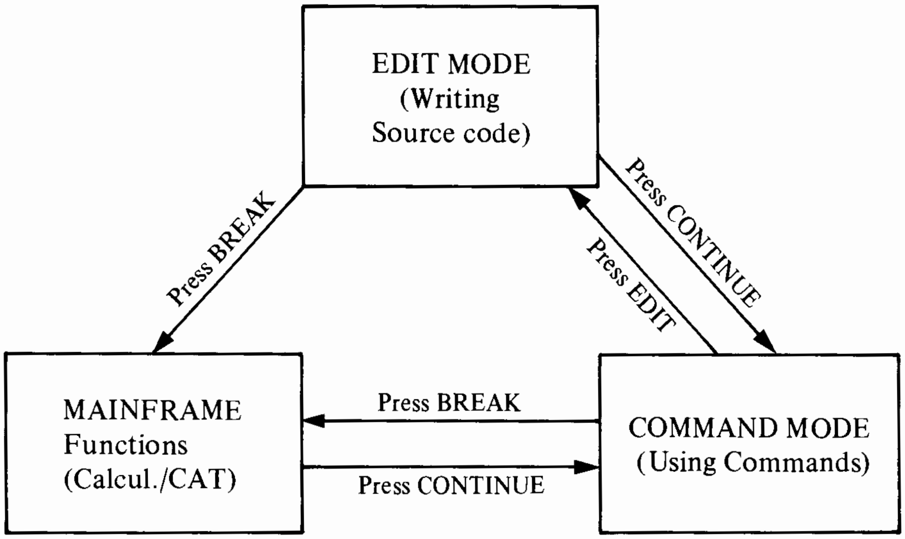
When using the Editor module, three terms will often occur. For a better understanding, a brief description of each term is given below.

BREAK. Use of the SFK BREAK, takes you out of the EDIT or COMMAND mode and all mainframe keys, except the SFKs, return to their original status. You may now perform such functions as calculations, catalog, etc. . . Pressing CONTINUE takes you to the Command mode.

COMMAND MODE. With this mode instructions can be given to the Desktop Computer that allow you to manipulate your Source code. Commands may either be typed on the CRT or called for by the various function keys.

EDIT MODE. This mode is entered by the PROGRAM key entitled EDIT on the 9835A or the SFK key entitled EDIT if you are using a 9845B. It allows you to write and edit your Source code. When in this mode, the operation of some mainframe keys change while others are inactive. Key descriptions are given later in this Section.

The following diagram gives you an overview of the Editor structure and how you can change between the various modes.



## Writing or Modifying Source Code

To use the Editor, call-up the Main Menu on the CRT and select SFK k1.

Before you start editing your Source code, you may wish to load a particular file from a mass storage device into memory. To begin a new Source code, enter the Edit mode by using the EDIT key.

Section VII of this manual gives a brief description of the Assembler Language instructions and differences that have been incorporated by Hewlett-Packard.

## File Handling

The Editor has a file handling system so that Source code can be handled that may be larger than the working space of the Desktop Computer memory.

A working file is created on the mass storage device whenever an overflow of the Editor memory occurs. The length of the file depends upon the size of the mainframe memory and the input Source code.

## Used as a Text Editor

Generally, the Editor works with Line numbers but by turning OFF the Syntax check and using the LIST statement followed by the @ sign, it can function as a text editor. Existing text already on a mass storage file, without line numbers, can be loaded.

When used for text editing, there is no provision in the program for vertical formatting of the listing (SPACE and PAGE commands are only recognized by the Assembler program).

You can put LF, FF characters into the text from the Desktop Computer keyboard, with the key CONTROL, that will effect the listing. However, it must be remembered that you have a default value, entered on the Printer Menu of the Initialization module, whereby the number of lines per page is given. Hence, everytime a print-out is made form feeds are automatically made when the respective number of lines is reached. It is possible to minimize the number of form feeds, executed by the Editor, by increasing the default value for the number of lines per page to 999.

# The Keyboard

A complete description of the Mainframe keyboard is given in your 9835A or 9845B Operating and Programming Manual.

The following is a list of keys which are the same as the Mainframe or very similar.

**Model 9845B**

CLEAR → END
CLEAR LINE
DEL CHR
DEL LN
HOME
INS CHR
ROLL (up arrow ↑)
ROLL (down arrow ↓)
PRT ALL
TAB CLEAR
TAB SET
TYPWTR
up arrow ↑
down arrow ↓
right arrow →
left arrow ←

**Model 9835A**

BACK
CLR → END
CLEAR LINE
DEL CHR
DEL LN
FWD
HOME
INS CHR
PRT ALL
TAB CLEAR
TAB SET
TO END
TYPWTR
up arrow ↑
down arrow ↓
right arrow →
left arrow ←

**Keys that are inactive**

STEP
RUN
RESULT
PAUSE
CLEAR
SPACE DEPENDENT

**Keys that have the same function as one another**

The CONTINUE and EXECUTE keys both have the same function except when the BREAK key has been pressed.

**Keys whose function change between Edit and Command Modes**

CONTINUE/EXECUTE

When in “EDIT MODE“ – Continue key takes you from the Edit mode to the Command mode.

When in “COMMAND MODE“ – Continue key executes a given command.

RECALL

When in “EDIT MODE“ – Using the STORE, DELETE LINE, CLEAR LINE or CLR→END key puts the line in which the cursor is to the top of a recall buffer. It returns on using the RECALL key.

When in “COMMAND MODE“ – The RECALL key has its Mainframe function.

**Keys with a different function to Mainframe.**

CLEAR LINE Clears the line of characters but the line numbers remain.

HOME Puts the cursor at the 7th character position on the same line that it was on before HOME was pressed.

INS LN Like its original mainframe function, the INS LN key inserts lines. However there is no restriction, as with the mainframe, where an insertion cannot be made if the difference between two line numbers is only one. With this key the Editor makes an insertion and then renumbers a minimum number of following lines.

STORE/ down arrow ↓ In addition to moving the cursor through the Source code it also allows you to append empty lines with line numbers at the end of the code.

TAB Has tab default positions across the CRT. The first setting is at character position 7 and then with increments of 10 until position 77. These positions may be deleted with the TAB CLR and/or new ones added with TAB SET.

9845B ONLY.

HOME/SHIFT In the Edit mode it puts the cursor to the first blank space at the end of the Source line.

9835A ONLY.

SHIFT down arrow ↓ Moves information on the CRT down 20 lines.

SHIFT up arrow ↑ Moves information on the CRT up 20 lines in the printout area.

**NOTE**

It is not possible to change the line number during editing. Line numbering is handled by the program.

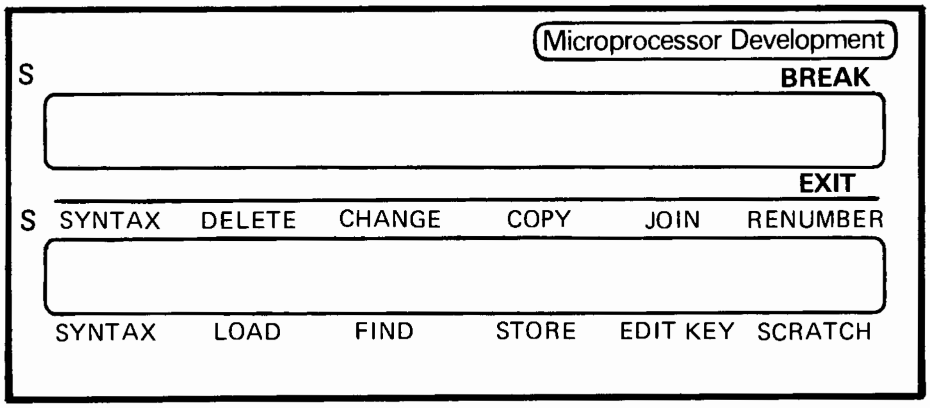
# SFK Overlay

The Editor module makes extensive use of the Special Function Keys. An SFK overlay is provided for the Hewlett-Packard Desktop computer and contains the pre-defined definitions that are implemented by the program.

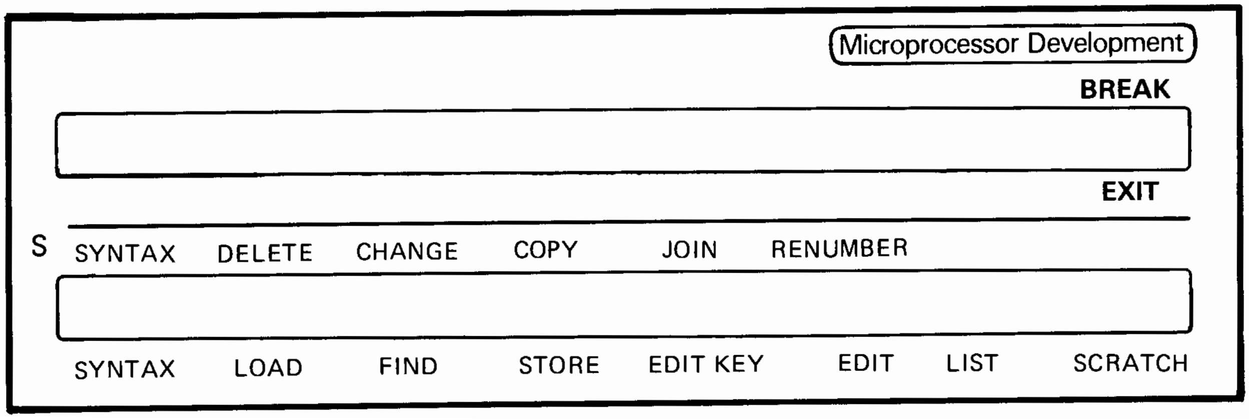
The keys in the upper row, 0 through 4 (0 through 6 for the HP 9845B) can be defined by you as typing aids for statements, variable names or other series of often-used keystrokes (see EDIT KEY statement).

The overlays for the 9845B and 9835A are shown below.

## 9835A Overlay



## 9845B Overlay



# Special Function keys

The following is a description of the Special Function Keys whose definition has been defined by the program.

## Conventions

If you have any difficulty with the conventions used in this section, refer to Section One for a complete explanation.

## SFK — BREAK

The BREAK key takes you out of the Editor mode and allows you to use the standard features of the Desktop Computer. This would be used, for example, when a CAT (Catalog) is required or when you need to make a calculation.

## SFK — CHANGE

Complete Syntax: **CHANGE** "<STR1>" **TO** "<STR2>" [ **FROM** <LNA>[ ,<LNB> ]]

**CHANGE**"<STR1>"**TO**"<STR2>"

The Editor substitutes <STR2> for <STR1> throughout the complete Source code. The maximum length of either string is 74 characters.

If <STR2> is longer than <STR1> the Editor inserts the additional characters. If <STR2> is shorter than <STR1> then the remaining characters of <STR1> are deleted.

**CHANGE** "<STR1>" **TO** "<STR2>" **FROM** <LNA>

String substitution takes place from <LNA> until the end of the program.

**CHANGE** "<STR1>" **TO** "<STR2>" **FROM** <LNA>, <LNB>

This instruction defines the range within which the substitution should be made. The range begins at line <LNA> and continues to line <LNB>, all lines inclusive.

## SFK key — COPY

**COPY** <LNA>,<LNB> **TO** <LNC>

The range <LNA> to <LNB> is copied into the Source code beginning with location line <LNC>. An automatic renumbering takes place if there is insufficient space to insert the specified range. The renumbering is similar to pressing the INS LN key.

COPY can only be used if the line <LNC> does not exist. The line <LNC> cannot be placed between <LNA> and <LNB>.

## SFK — DELETE

**DELETE** <LNA>,<LNB>

Deletes all lines between and including <LNA> and <LNB>.

## SFK — EDIT (9845B)
## PROGRAM key — EDIT (9835A)

Complete Syntax: **EDIT** [ <LN>[ , <INCREMENT> ] ]

**EDIT**

Press EDIT and CONTINUE, and the CRT is ready for your inputs. Line number 10 appears on the CRT. If the Source is already in the Editor memory it appears in the CRT with the first line at the top. Upper and lower case letters, control characters, as well as all other types of ASCII characters may be input.

A line may contain a maximum of 74 characters (plus 6 for the line number). A beep is sounded when the maximum is reached and no more characters can be entered.

After you have finished with each line, use the STORE key to move the cursor to the start of the next. A syntax check is made when the SYNTAX is ON (default value).

The maximum line number is 99999, however only a total of 32767 lines can be stored.

**EDIT** <LN>

It is possible to find a particular line in your program by using the Edit command followed by <LN>, whereby <LN> is the line number you require. When found the line is displayed together with 19 additional lines. If <LN> is greater than the last available line, then the last 10 lines are displayed.

Use this Command to start a Source code at a given line number <LN>, (default is 10).

**EDIT** <LN>,<INCREMENT>

Applies when you begin to write your Source code in the Edit mode. The command specifies the starting line number and the increment. If used with an existing program, the <INCREMENT> is only effective for lines added at the end of the Source code.

## SFK — EDIT KEY

Words or messages can be stored on the SFK's k0 through k4 (9835A) or k0 through k6 (9845B), and can be used afterwards as typing aids when entering your Source code. Text must be kept to less than 74 characters.

To program a SFK, follow the instructions in the display line on the CRT.

To recall your text, when in the Edit mode, press the SFK containing the programmed message and it will appear at the position of the cursor on the CRT.

## SFK — EXIT

SFK EXIT returns you to the Main Menu. If you have not stored your Source code, a CRT prompt asks you if you want to or not. All previous code, and the work file on your mass storage device, are purged.

## SFK — FIND

Complete Syntax: **FIND** "<STR>"[ **FROM**<LNA>[,<LNB>] ]

**FIND** "<STR>"

Using this command, the Source code is searched for a defined String. The maximum length of your defined String must not exceed 74 characters. The Source code is searched until the first string is found. This done, the cursor is located beneath the first character of the String. Modifications can now be made within the String.

Leaving the line with any other operation except CONTINUE causes the Command to end. The CONTINUE key continues the search for the next String.

**FIND** "<STR>" **FROM** <LNA>

The Source code is searched for the specified String <STR> beginning with <LNA>. The sequence ends by entering the Command mode.

**FIND** "<STR>" **FROM** <LNA>, <LNB>

The Source code is searched for the specified String <STR> within the range <LNA> to <LNB>. The sequence ends by entering the command mode.

## SFK — JOIN

Complete Syntax: **JOIN** "<FILE NAME> [msus]" [ <LN> ] [ @ ]

**JOIN** "<FILE NAME>"

Source code is loaded from the mass storage device and joined onto the Source code already in the Editor memory. When line numbers exist on the new code, and they are smaller or equal to the last line number on the existing code, and when no LN is specified, then the first line number of the new code will be the last line number of the already existing Source file plus the line increment.

**JOIN** "<FILE NAME> msus"

The msus defines the mass storage device from which the file is to be joined. If no msus specification is made, the default is the msus for the Source file stored in the Initialization module default file.

**JOIN** "<FILE NAME> msus" <LN>

If <LN> is specified then the file being joined has <LN> as the first file number. The <LN> may not be smaller than the last line number of the existing Source file.

**The optional @ will generate line numbers in front of the lines being loaded irrespective of whether line numbers are there or not. The @ is normally used for Source code that does not have line numbers. If < LN > is specified, the Source code to be joined begins with the specified number.**

## SFK — LIST (9845B)
## PROGRAM key — LIST (9835A)

Complete Syntax: **LIST** [ #<SC>; ] [ <LNA>[ ,<LNB> ]] [@]

| | |
|---|---|
| **LIST** | The whole Source code is listed |
| **LIST** <LNA> | Listing is made from <LNA> to the end of the code. |
| **LIST** <LNA>,<LNB> | Listing is made from <LNA> to <LNB> inclusive. |

The above three commands are listed on the printer whose Select Code is in the Initialization module default file.

Adding the option # <SC>; before an instruction specifies the Select Code of a different printer.

Additionally, to put the option @ after an instruction causes the Source code to be printed without line numbers.

## SFK — LOAD

Complete Syntax: **LOAD** "<FILE NAME> [msus]"[@]

**LOAD** "<FILE NAME>"

Type in the name of the file that is to be loaded from the default msus. The maximum length of a line should not exceed 74 characters plus another 6 for the line number. The maximum number of lines is 32767. No more than 6 characters are allowed in the file name. The Select Code is the msus for the Source file in the Initialization default file.

**LOAD** "<FILE NAME> msus"

Loads the Source code from a given mass storage device specified by the msus.

The optional @ will generate line numbers in front of the lines being loaded irrespective of whether line numbers are there or not. The @ is normally used for Source code that does not have line numbers.

## SFK — RENUMBER

Complete Syntax: **RENUMBER** [<LN> [,<INCREMENT> ]]

**RENUMBER**

This command allocates new line numbers to the existing Source code. If no specification is given, line numbers begin with 10, and increment by 10.

**RENUMBER** <LN>

Starts renumbering with <LN> as the first line of the Source code. Lines increment by 10.

**RENUMBER** <LN>,<INCREMENT>

Defines the first line number of the Source code and the increment.

## SFK — SCRATCH

This command deletes the contents of the Editor memory including work files.

## SFK — STORE

Complete Syntax: **STORE** "<FILE NAME> [ msus ]"[ <LNA> [,<LNB> ]] [@]

**STORE** "<FILE NAME>"

When you have finished with your Source code, it can be stored as a data file using the device given on the Initialization module default file.

**STORE** "<FILE NAME> msus"

Source code is stored on a specified mass storage device.

**STORE** "<FILE NAME> msus" <LNA>

It is also possible to store just a part of the Source code, beginning with <LNA> until the end of the code.

**STORE** "<FILE NAME> msus" <LNA>,<LNB>

When both specifiers are given, the Source code between <LNA> and <LNB> is stored.

The optional specification @ means that the file is stored without line numbers.

## SFK — SYNTAX

The Editor makes a syntax check of each line, however it does not consider the logical data flow. If a syntax error is found, an error message is displayed at the bottom of the CRT and the cursor is placed over the mistake so that it can be corrected. A syntax check is made whenever a line is left, regardless of the type of instruction used. A line is also checked when an insert is made.

When the syntax rejects a statement it can be overidden by using the STORE key in conjunction with the SHIFT key (press SHIFT/STORE). The Syntax accepts the instruction as valid and the next line may be entered.

There are two possible states that can be entered:

Syntax ON. This is the initial state of the Editor. A syntax check is made and the word "SYNTAX" appear on the display line.

Syntax OFF. No syntax check occurs.

# Points To Remember

1. You cannot enter more Source lines than your Desktop Computer memory will accept without using work files. The specification section of this manual (Section I) lists the amount of Source lines possible with respect to memory size. Trying to exceed the maximum causes an error message. To remedy this you must store your Source code and then specify that you need files by entering the correct information in the "Editor and Assembler" menu of the Initialization Section.

2. When using the COPY, LOAD, or JOIN commands it may happen that you exceed the maximum amount of line numbers or maximum number of lines. Your Source code is still available up to the moment the Error message occured. No more than 100 work files are possible.

# Section Seven
# 8080/85 Assembly Language Instructions

## Introduction

The following is a description of the Assembly Language instructions needed to write Source code for the 8080/8085 Microprocessor. We recommend that you make yourself acquainted with this explanation as it differs in some respects from the description given by 8080/85 Programming manuals.

## Source Line Format

A Program written in Assembler Language is called a Source code. It consists of symbolic commands called statements. Each statement is written on a single line and may consist of from one to four fields. From left to right the fields are:

(1) LABEL: (2) INSTRUCTION (3) OPERAND (4) ;COMMENT

The following pages describe each field individually. An illustration of the format pattern is given below.

```
[ <LABEL>:] [ <INSTRUCTION> ]   [ <OPER> [,<OPER> . . . ] ]   [; <COMMENT> ]

Space LABEL: space INSTRUCTION space OPERAND, OPERAND space ;COMMENT
  |             |                  |                         |
Optional     Optional          at least one               Optional
```

To assist you when writing your Source code, it is not necessary to ensure that all your characters are inputted as capital letters as all lower case characters are converted to upper case.

## Label Field

A Label is required for statements involved in the definition of symbols and occasionally at destinations of branch and jump instructions.

A Label is a symbol name whose value is the location where the instruction is assembled. Any number of characters (limited by the line length) may be used however, the assembler only recognises up to eight alphanumeric characters whereby the first character must be alphabetic. A Label may be used in any Source line but need not begin in the first character position of a statement. All Labels must end with a colon.

In the 8080/85 language Labels are sometimes called Names. They may be used instead of a Label for the SET, EQU, and MACRO directives. These directives are described later in this Section. Names follow the same coding rules as Labels but are not terminated with a colon.

The Assembler maintains a location counter to provide addresses for the symbols in the Label field. When a symbol is found in the Label field the Assembler places the symbol and the corresponding location counter value in a symbol table.

## Instruction Field

Instructions are machine or pseudo mnemonics and must be present in all statements except when the statement consists only of a Comment or Label. Pseudo mnemonics are described later in this Section, and a list of 8080/85 machine mnemonics is given at the back of this Section.

## Operand Field

The Operand field identifies the data to be acted upon by the specified instruction. Some instructions require no Operand, while others require one or more.

The Operand field is separated from the Instruction field by at least one blank. If more than one Operand is required they are separated from each other by a comma.

The type of information placed in the Operand field depends upon the particular Instruction. The following are recognized by the Assembler as Operands:

Numbers<br>
Symbols<br>
Expressions<br>
ASCII strings<br>
Location counter symbol ($)

The listed Operand types are now described in more detail.

## Numbers

The Assembler accepts numbers in the Operand field in several different bases: binary, octal, decimal, and hexadecimal. Numbers are accepted in the following formats:

| | | |
|---|---|---|
| <Number> | Specifies a Decimal number | 0–65536 |
| <Number>D | Specifies a Decimal number | 0D–65536D |
| <Number>H | Specifies a Hexadecimal number | 0H–0FFFFH |
| <Number>O | Specifies an Octal number | 0O–177777O |
| <Number>Q | Specifies an Octal number | 0Q–177777Q |
| <Number>B | Specifies a Binary number | 0B–1111111111111111B |

The <Number> must always begin with a numeric character. For example:

| | |
|---|---|
| A9FFH | Illegal (will be recognized as a Symbol) |
| 0A9FFH | Correct specification. |

## Symbols

Symbols can be divided into three separate identities, Reserved, User definable, and Assembler generated symbols.

Reserved symbols are those that already have a special meaning to the Assembler and therefore cannot appear as user-defined symbols. The mnemonic names for microprocessor instructions and the Assembler directives are all reserved symbols.

The following instruction Operand symbols are also reserved:

| | |
|---|---|
| A | Register A |
| B | Register B, or Register pair B & C |
| C | Register C |
| D | Register D, or Register pair D & E |
| E | Register E |
| H | Register H, or Register pair H & L |
| L | Register L |
| SP | Stack Pointer |
| PSW | Program Status Word |
| M | Memory Reference Identification. |
| $ | Location Counter. |

User-defined symbols are symbols you create to reference instructions and data addresses. These symbols are defined when they appear in the Label field of an instruction, EQU, SET, or MACRO.

# Expressions

An expression is an Operand entry consisting of either a single term or a combination of terms. It contains a valid series of numbers, and symbols that may be connected by operators.

An intermediate result, if obtained during the evaluation of expressions, will be truncated to a 16 bit value.

The Assembler includes five groups of Operators which permit the following Assembly-time operations:

Arithmetic
Shift
Byte
Logical
Relational

It is important to remember that these are assembly-time operations. Once the Assembler has evaluated an expression, it becomes a permanent part of your program.

Expressions are evaluated from left to right. Operators with higher precedence are evaluated before other operators that immediately preceed or follow them. When two operators have equal precedence, the left most is evaluated first.

The following list describes classes of operators in order of precedence.

| | |
|---|---|
| Parenthesized expressions | High |
| Unary Minus | ↓ |
| HIGH LOW | |
| * / MOD SHR SHL | |
| + – | |
| Relational Operators | |
| NOT | |
| AND | |
| OR XOR | Low |

The Assembler must complete the numeric evaluation of symbols and expressions in two passes. Only one level of forward addressing is permitted in the use of symbols or expressions in the Operand field of the Source statements.

For instance, in the example below P would not be evaluated.

```
P        EQU    R
R        EQU    Q
Q        EQU    5
```

## ARITHMETIC OPERATORS

| Operator | Meaning |
|---|---|
| + | Addition |
| – | Subtraction |
| * | Multiplication |
| / | Integer Division. Any remainder is discarded (7/2=3). |
| MOD | Modulo. Result is the remainder caused by a division operation. (7 MOD 3 = 1) |

Examples:

The following expressions generate the bit pattern for the ASCII character A, where A equals 65 decimal.

```
5+30*2
(25/5)+30*2
5+(–30*–2)
```

Example of MOD

The MOD operator must be separated from its Operands by spaces:

```
NUMBER    MOD    8
```

Assuming that NUMBER has the value 25, then the expression evaluates to 1.

## SHIFT OPERATORS.

| Operator | Meaning |
|---|---|
| y SHR x | Shift 'y' to the right 'x' bit positions. |
| y SHL x | Shift 'y' to the left 'x' bit positions. |

The shift operators do not wraparound any bits shifted out of the byte. Bits vacated by the shift operation are zero filled. Note that the shift operator must be separated from its Operands by spaces.

Example:

Assume that NUMBER has the value 0101 0101B. The effect of the shift operators is as follows:

```
NUMBER    SHR   2     0001 0101B
NUMBER    SHL   1     1010 1010B
```

## BYTE ISOLATION OPERATORS

| Operator | Meaning |
|---|---|
| HIGH | Isolate High Order 8 Bits of 16-bit value. |
| LOW | Isolate Low Order 8 Bits of 16-bit value. |

The Assembler regards expressions as 16-bit values. In certain cases, you need to deal with only part of an address, or you need to generate an 8-bit value. This is the function of the HIGH and LOW operators.

Example:

Assume that ADDRESS is an address manipulated at assembly time for building tables or lists of items that must all be below address 255 in memory. The following IF directive determines whether the HIGH-order byte of ADDRESS is zero, thus indicating that the address is still less than 256:

```
IF HIGH ADDRESS EQ 0
               ....
               ....
               ....
```

## LOGICAL OPERATORS.

| Operator | Meaning |
|---|---|
| NOT | Logical One's Complement |
| AND | Logical AND |
| OR | Logical OR |
| XOR | Logical Exclusive OR |

The logical operators act only upon the least significant bit of values involved in the operation (except for the NOT operator).

Example:

```
If         VALUE1 = 12
           VALUE2 = 15
           VALUE3 = 20

Then VALUE1 AND VALUE2 AND VALUE3 = 0
```

RELATIONAL OPERATORS.

| Operator | Meaning |
|---|---|
| EQ | Equal |
| NE | Not Equal |
| LT | Less Than |
| LE | Less Than or Equal |
| GT | Greater Than |
| GE | Greater Than or Equal |

The relational operators produce a yes-no result. Thus, if the evaluation of the relation is TRUE, the value of the result is all ones. If false, the value of the result is all zeros.

Relational operators compare unsigned binary values. Hence, a –3 will be evaluated as greater than 10.

Example:

The following IF directive tests the values of VALUE1 and VALUE2 for equality. If the result of the comparison is TRUE, the assembly language coding following the IF directive is assembled. Note that the relational operator must be separated from its Operand by spaces.

```
IF VALUE1    EQ    VALUE2
              .    . . .
              .    . . .
```

## ASCII Strings

All ASCII characters enclosed in single quotes define an ASCII String. Two successive quotes must be used to represent one single quote. For example:

'TODAY"S DATE' is written for TODAY'S DATE

Double quote marks can also be used to define an ASCII String. In this case, the double quotes determine the String delimiter while a single quote is regarded as a character within the String.

# Comment Field

Comments can be optionally used at the end of a Source line. If present, they are ignored by the Assembler during translation and appear only in the program listing.

A Comment must begin with a semicolon. Alternatively, a line can exist as only a comment.

# Assembler Directives

The Assembler directives, sometimes called pseudo mnemonics, are instructions entered into the Source code which direct the Assembler when it generates the Object code. They control the operation of the Assembler program and define the format of the Assembler output listing.

The following examples are taken from an assembled program. Both the Source code and Assembler program are listed at the end of the examples in their entirety.

**DB** – Define byte

[ <LABEL>: ]   **DB**   <EXPR> | <STRING> [ , <EXPR> | <STRING> . . . ] [;<COMMENT> ]

An 8-bit value corresponding to each Operand is stored in a byte of the Object program. This directive may have one or more Operands, separated by commas. The number of Operands is limited by the Source line length of 80 characters. The Operand can be any constant, symbol, an expression evaluating to an 8-bit value, or an ASCII String. When the Label is present the address of the first generated byte is assigned to the symbol represented by the Label.

Example

```
210 1217 24           DB      00100100B ;BINARY VALUE
220 1218 54455854     DB      'TEXT'    ;ASCII "TEXT" INTO MEMORY
```

**DS** – Define storage

[ <LABEL> : ]   **DS**   <EXPRESSION> [; <COMMENT> ]

The DS directive increases the location counter by the value of the expression. This reserves a block of memory equal to the size of the expression value. When a symbol is used, it must have been previously defined. A symbol that has been defined by a Label is assigned the address of the first byte reserved by this directive. The contents of the memory block are unchanged. The Assembler only advances the location counter and does not generate any Object code.

Example

```
240 121C        TEMP: DS   8          ;CREATE AREA 8 BYTES LONG
```

**DW** – Define word

[ <LABEL>: ]   **DW**   <EXPRESSION> [,<EXPRESSION> . . . ] [; <COMMENT> ]

The DW directive is similar to the Define Byte directive except that it generates a double byte Object from a 16-bit value. The least significant bits are stored in the first byte. Strings of one or two characters are allowed, longer Strings cause errors.

Example

```
200 1215 0C43   CONSTANT:DW   041414Q  ;OCTAL VALUE
```

**END** – End of program.

[ <LABEL> : ]     **END**     [ <START ADDRESS> ]  [; <COMMENT> ]

This directive stops the Assembler program. If the END directive is missing, the Assembler program is terminated by the end of the file. However, the Assembler does not stop if an END directive is part of a file called by the INCLUDE or SOURCE control from your mass storage device. The directive is not translated into Object code. The start address should be present when it is required by your Microprocessor system. Refer to the Console Section of this manual for further information.

Example

```
480       120F                     END       4623      ;STOPS THE ASSEMBLER
```

**EQU** – Equate

<LABEL>:     **EQU**     <EXPRESSION>  [;<COMMENT> ]

The EQU directive assigns a value, given by the Operand field, to the Label or Name specified in the Label field. However, this name cannot be redefined by a subsequent EQU nor a SET directive.

Example

```
160       00D7          VALUE     EQU       3270      ;OCTAL VALUE
```

**FAIL**

[ <LABEL> :]     **FAIL**     'USER DEFINABLE ERROR MESSAGE'

The FAIL directive forces the Assembler to print out a user-defined error message. This directive would be contained in your Source code as an indication that unwanted code has been Assembled. It is often used within the IF-ELSE-ENDIF directives to ensure that the correct directive has been Assembled.

Example

```
**'ASSEMBLING PART 2'**
   340                               FAIL      'ASSEMBLING PART 2'
```

### IF-ELSE-ENDIF

```
[ <LABEL> :]     IF      <EXPRESSION>  [; <COMMENT> ]
      .
      .                  CODE SEQUENCE 1
      .
[ <LABEL> : ]    ELSE                  [;<COMMENT> ]
      .
      .                  CODE SEQUENCE 2
      .
[ <LABEL> : ]    ENDIF                 [;<COMMENT> ]
```

Code sequences between the IF-ENDIF directives are controlled by the value after the IF directive.

A code sequence between the IF-ENDIF directive will be assembled when the least significant bit of the expression value evaluates to a 1.

When the ELSE directive is present, the code sequence between the IF and the ELSE directives is assembled when the least significant bit of the expression value evaluates to a 1. When the least significant bit of the expression evaluates to a 0, the code sequence between the ELSE and the ENDIF directives is assembled.

Example

```
   260                           IF     VALUE-3270    ;CONDITIONAL ASSEMBLY START
   270
   280                 START:    MVI        A,VALUE+2 ;THIS LINE IS NOT ASSEMBLED
   290
   300                           ELSE
   310
   320 1224 3E47       START:    MVI        A,'G'     ;THIS LINE IS ASSEMBLED
   330
**'ASSEMBLING PART 2'**
   340                           FAIL       'ASSEMBLING PART 2'
   350
   360                           ENDIF                ;CONDITIONAL ASSEMBLY END
```

### IFC-ELSE-ENDIF

```
[ <LABEL> : ]    IFC    <OPER1> ,<OPER2>     [;<COMMENT> ]
                 .
                 .      CODE SEQUENCE
                 .
[ <LABEL> : ]    ENDIF                       [;<COMMENT> ]
```

The IFC directive is similar to the IF directive. However, the code sequence is Assembled if the two Operands, following the IFC directive, are equal on a character for character basis. No evaluation is performed on the two Operands. Operands can be Symbols or Strings. You use the ELSE directive in the same manner as with the IF directive.

### IFNC-ELSE-ENDIF

[ <LABEL> : ]    **IFNC**   <OPER1> , <OPER2>    [ ; <COMMENT> ]
.
.   CODE SEQUENCE
.
[ <LABEL>: ]    **ENDIF**    [ ; <COMMENT> ]

The IFNC directive is similar to the IFC directive, whereby the code sequence is Assembled if the two Operands, following the IFNC directive, are not equal on a character for character basis.

### ORG – Origin

[ <LABEL> : ]    **ORG**   <EXPRESSION>    [ ; <COMMENT> ]

This directive defines the numerical address of the first byte of machine code which results from the assembly of the immediately subsequent section of a Source program. There may be any number of ORG statements in a program.

The ORG directive sets the location counter to the value expressed in the Operand field. This field may contain the actual value (decimal, hexadecimal, octal, or binary) to which the location counter is to be set. Alternatively, the Operand field may contain a previously defined symbol or expression which can be assigned a numerical value by the Assembler.

The location counter is initialized to 0000H before each assembly. If no ORG statement appears at the beginning of the program, the location counter begins as if an ORG zero had been entered.

Example

```
30 0000                         ORG        4623       ;DECIMAL ADDRESS
```

### SET

<LABEL> :    **SET**    <EXPRESSION>    [ ; <COMMENT> ]

The SET directive assigns the value given by the Operand field to the Label or Name specified in the Label field. This value remains unchanged until altered by a subsequent SET directive.

Example

```
150       00A0         LABEL     SET        00A0H      ;HEXADECIMAL ADDRESS
```

### SPC – Space

[ <LABEL> : ]    **SPC**    <EXPRESSION>    [ ; <COMMENT> ]

The SPC directive inserts a number of blank lines in the program listing. The number of lines to be left blank is stated in the Operand field. When the SPC directive causes the listing to cross page boundaries, only those blank lines required to get to the top of the next page are generated.

# Assembler Controls

All Assembler Controls are preceeded by the $ sign which must be placed in the first character position. As many Controls as required may be included on one line and each Control must be separated by a blank or a comma. If more controls are required that can fit on one line, other lines may be used as long as each line begins with the $ sign.

These Controls can also be entered after the SOURCE or OBJECT statement in the Assembler program. See Section IX.

$ [ <CONTROL1> [ ,<CONTROL2> . . . ] ]     [ ; COMMENT]

**TITLE** "ASCII String" Defines the heading for the top of the page. If entered in the first line of the Source code it appears at the top of the first page. Entered into the Source code at any other time causes it to appear on the next page following its definition.

**EJECT** Advances the paper to the top of the next page.

**SOURCE** "<FILE NAME> [msus]" A file on the mass storage device is included in the assembly if specified in the Operand field with this control. If a Source line contains two or more SOURCE controls, then the last Source file specified is assembled first. Eight levels of nesting are allowed. Enter the msus if the Source file is not stored on the mass storage device specified in the default file.

**INCLUDE** "<FILE NAME> [msus]" Same as for SOURCE.

**OBJECT** ["<FILE NAME> [msus]"] As long as the file in which the Object code is to be stored was not specified with the OBJECT statement of the Assembler program, it may be specified by inserting this control in the Source code. Refer to Assembler module section. The position of this control in the Source code is not important, as the Object code for the entire Source code will still be stored.

Entering the OBJECT Control without a file name results in no Object code being stored.

If the Object file is specified, no other Object control is accepted.

**SYMBOLS** Prints the Symbol Table after the program listing.

***NOSYMBOLS** Suppress Symbol Table print.

***XREF** When the Assembler is finished the XREF generator is called and prints the cross reference of all user symbols. If XREF and SYMBOLS have been specified, then only an XREF is made.

**NOXREF** Suppresses the cross reference.

**SAVE** The current settings of LIST, COND and GEN are stored but remain valid until explicitly changed. Nesting up to 8 levels is valid.

This Control is useful if you know that the Assembler program will encounter another set of Controls and it is necessary to restore the original values afterwards.

**RESTORE** Recalls the list Control settings. If more RESTORE commands are given than there are SAVE commands, then the extra RESTORE commands are ignored.

***LIST** Instructs the Assembler to generate a program listing.

**NOLIST** Suppresses the printing of the program listing. Error messages appear on the CRT.

***COND** Includes conditional skipped Source lines in the listing.

**NOCOND** Does not include conditionally skipped lines.

***GEN** Lists MACRO expansion.

**NOGEN** Does not list MACRO expansion.

*The Assembler uses these Controls unless otherwise specified.

Examples

To give your printed pages a heading, use the TITLE control followed by the heading enclosed in quotes

```
10                    $ TITLE 'ASSEMBLER DIRECTIVE DEMO PROGRAM'
```

To save list controls, and suppress the Macro listing, type the following controls.

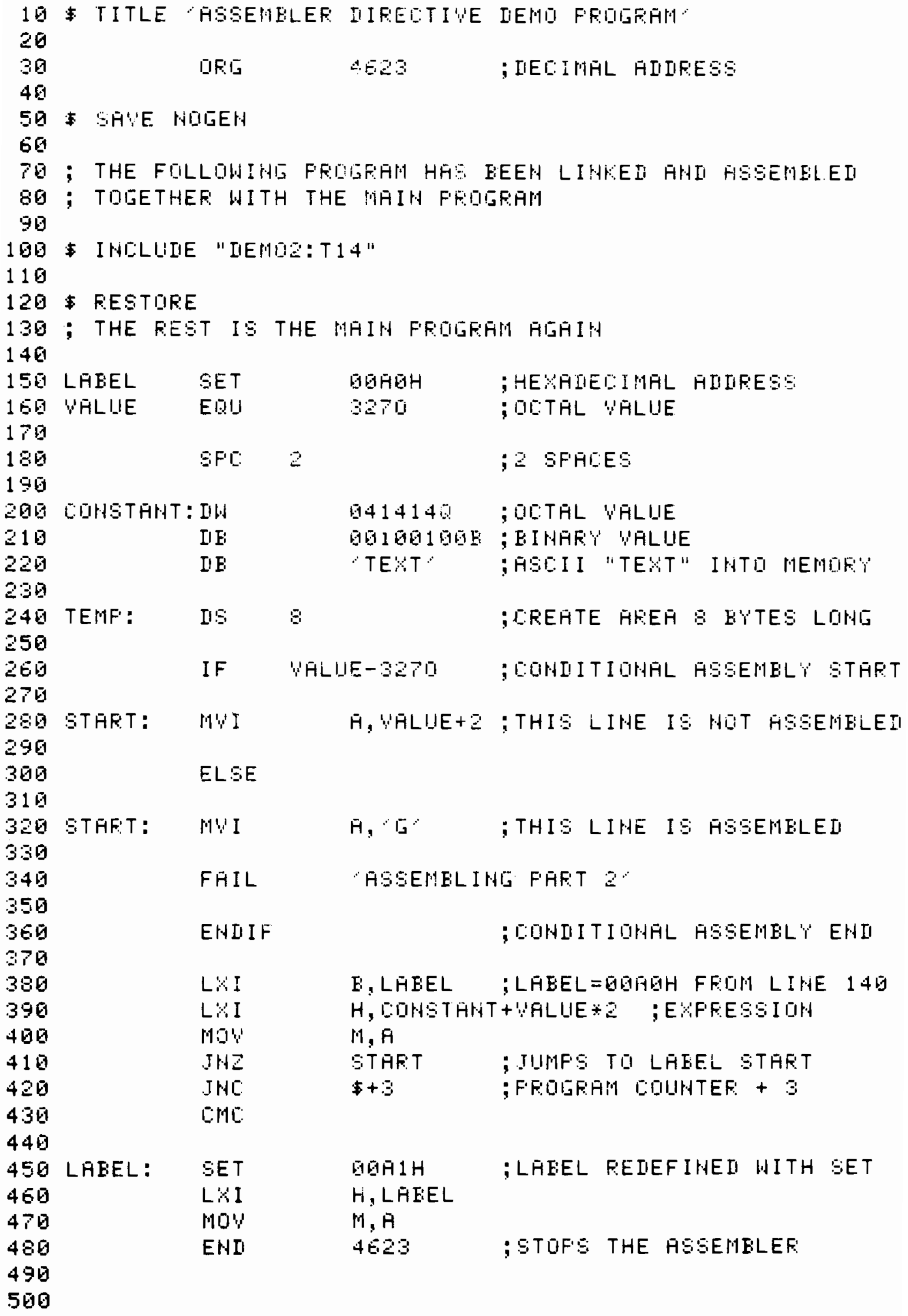
```
50                    $ SAVE NOGEN
```

You can use the INCLUDE control to assemble several Source files within the same assembly. It is possible to terminate the Source file with an END directive. Once the included file has been assembled, the Assembler continues with the file that contained the INCLUDE control. Eight levels of nesting are possible.

```
100                    $ INCLUDE "DEMO2:T14"
 10                    $ TITLE 'PROGRAM TO BE LINKED'
 20
 30 120F 3E3F                   MVI       A,63
 40 1211 3D            LOOP:    DCR   A
 50 1212 C21112                 JNZ       LOOP
 60                             END
110
120                    $ RESTORE
130                    ; THE REST IS THE MAIN PROGRAM AGAIN
```

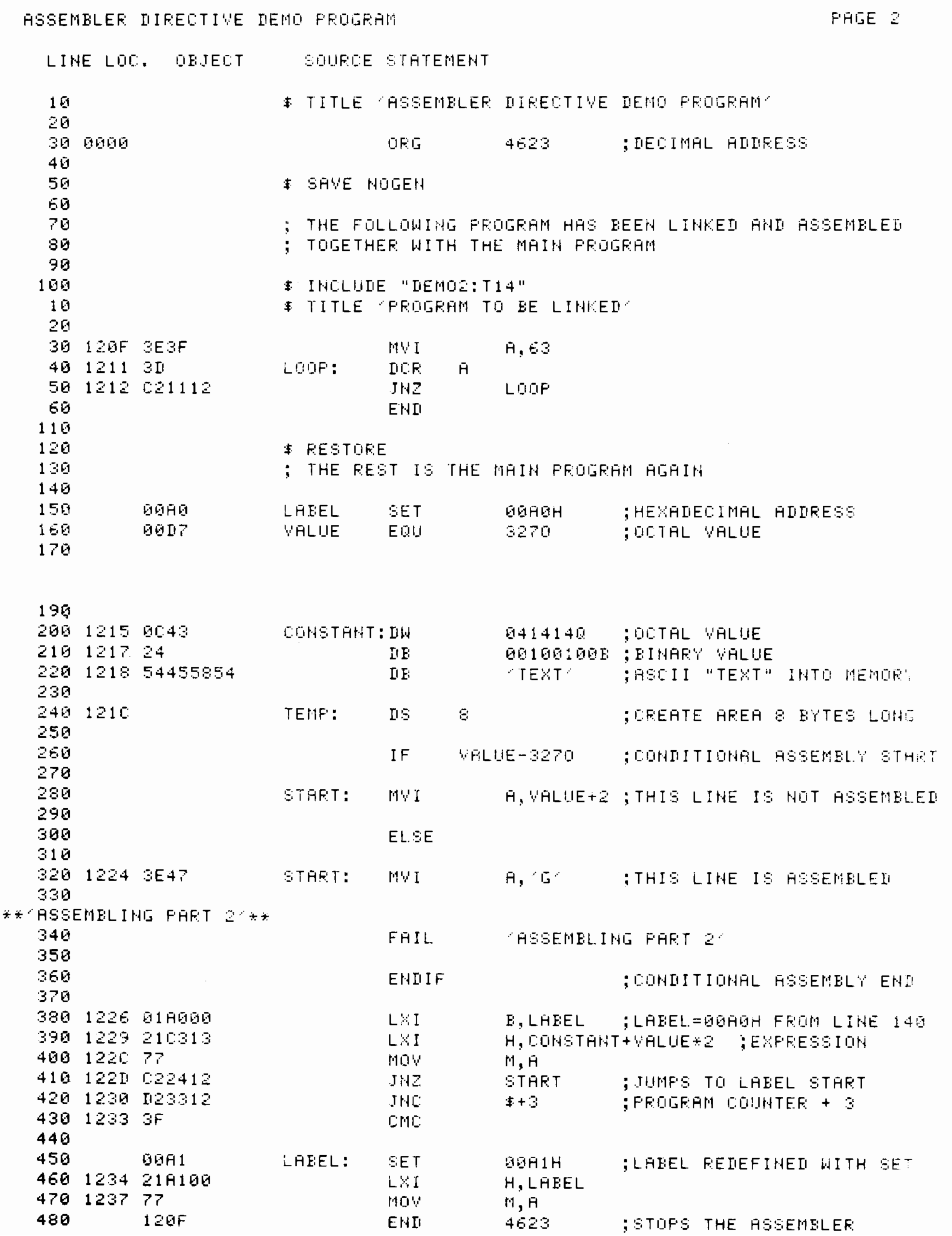
Note that on the original Source code the $ sign would be placed in the first character position.

# Example of Source Code



```
 10 $ TITLE 'ASSEMBLER DIRECTIVE DEMO PROGRAM'
 20
 30          ORG       4623      ;DECIMAL ADDRESS
 40
 50 $ SAVE NOGEN
 60
 70 ; THE FOLLOWING PROGRAM HAS BEEN LINKED AND ASSEMBLED
 80 ; TOGETHER WITH THE MAIN PROGRAM
 90
100 $ INCLUDE "DEMO2:T14"
110
120 $ RESTORE
130 ; THE REST IS THE MAIN PROGRAM AGAIN
140
150 LABEL    SET       00A0H     ;HEXADECIMAL ADDRESS
160 VALUE    EQU       3270      ;OCTAL VALUE
170
180          SPC   2             ;2 SPACES
190
200 CONSTANT:DW        041414Q   ;OCTAL VALUE
210          DB        00100100B ;BINARY VALUE
220          DB        'TEXT'    ;ASCII "TEXT" INTO MEMORY
230
240 TEMP:    DS    8             ;CREATE AREA 8 BYTES LONG
250
260          IF    VALUE-3270    ;CONDITIONAL ASSEMBLY START
270
280 START:   MVI       A,VALUE+2 ;THIS LINE IS NOT ASSEMBLED
290
300          ELSE
310
320 START:   MVI       A,'G'     ;THIS LINE IS ASSEMBLED
330
340          FAIL      'ASSEMBLING PART 2'
350
360          ENDIF               ;CONDITIONAL ASSEMBLY END
370
380          LXI       B,LABEL   ;LABEL=00A0H FROM LINE 140
390          LXI       H,CONSTANT+VALUE*2  ;EXPRESSION
400          MOV       M,A
410          JNZ       START     ;JUMPS TO LABEL START
420          JNC       $+3       ;PROGRAM COUNTER + 3
430          CMC
440
450 LABEL:   SET       00A1H     ;LABEL REDEFINED WITH SET
460          LXI       H,LABEL
470          MOV       M,A
480          END       4623      ;STOPS THE ASSEMBLER
490
500
```

# Example of Program Listing



```
  ASSEMBLER DIRECTIVE DEMO PROGRAM                                       PAGE 2

    LINE LOC.  OBJECT      SOURCE STATEMENT

    10                   $ TITLE 'ASSEMBLER DIRECTIVE DEMO PROGRAM'
    20
    30 0000                       ORG       4623      ;DECIMAL ADDRESS
    40
    50                   $ SAVE NOGEN
    60
    70                   ; THE FOLLOWING PROGRAM HAS BEEN LINKED AND ASSEMBLED
    80                   ; TOGETHER WITH THE MAIN PROGRAM
    90
   100                   $ INCLUDE "DEMO2:T14"
    10                   $ TITLE 'PROGRAM TO BE LINKED'
    20
    30 120F 3E3F                  MVI       A,63
    40 1211 3D           LOOP:    DCR   A
    50 1212 C21112                JNZ       LOOP
    60                            END
   110
   120                   $ RESTORE
   130                   ; THE REST IS THE MAIN PROGRAM AGAIN
   140
   150      00A0         LABEL    SET       00A0H     ;HEXADECIMAL ADDRESS
   160      00D7         VALUE    EQU       3270      ;OCTAL VALUE
   170


   190
   200 1215 0C43         CONSTANT:DW        0414140   ;OCTAL VALUE
   210 1217 24                    DB        00100100B ;BINARY VALUE
   220 1218 54455854              DB        'TEXT'    ;ASCII "TEXT" INTO MEMORY
   230
   240 121C              TEMP:    DS    8             ;CREATE AREA 8 BYTES LONG
   250
   260                            IF    VALUE-3270    ;CONDITIONAL ASSEMBLY START
   270
   280                   START:   MVI       A,VALUE+2 ;THIS LINE IS NOT ASSEMBLED
   290
   300                            ELSE
   310
   320 1224 3E47         START:   MVI       A,'G'     ;THIS LINE IS ASSEMBLED
   330
**'ASSEMBLING PART 2'**
   340                            FAIL      'ASSEMBLING PART 2'
   350
   360                            ENDIF               ;CONDITIONAL ASSEMBLY END
   370
   380 1226 01A000                LXI       B,LABEL   ;LABEL=00A0H FROM LINE 140
   390 1229 21C313                LXI       H,CONSTANT+VALUE*2  ;EXPRESSION
   400 122C 77                    MOV       M,A
   410 122D C22412                JNZ       START     ;JUMPS TO LABEL START
   420 1230 D23312                JNC       $+3       ;PROGRAM COUNTER + 3
   430 1233 3F                    CMC
   440
   450      00A1         LABEL:   SET       00A1H     ;LABEL REDEFINED WITH SET
   460 1234 21A100                LXI       H,LABEL
   470 1237 77                    MOV       M,A
   480      120F                  END       4623      ;STOPS THE ASSEMBLER
```

# Cross Reference List

```
SYMBOL      VALUE    DEFINED   REFERENCED

CONSTANT    1215       200       390
LABEL       00A1       150     RE-DEFINED:    450
                                 380   460

LOOP        1211        40        50
START       1224       320       410
TEMP        121C       240     * UNREFERENCED *
VALUE       00D7       160       260    390
```

# Character Set

The Assembler recognizes the complete ASCII character set. Generally all characters are considered to be alphabetic and lower case are converted to upper case. The following are exceptions, and they each have a special meaning.

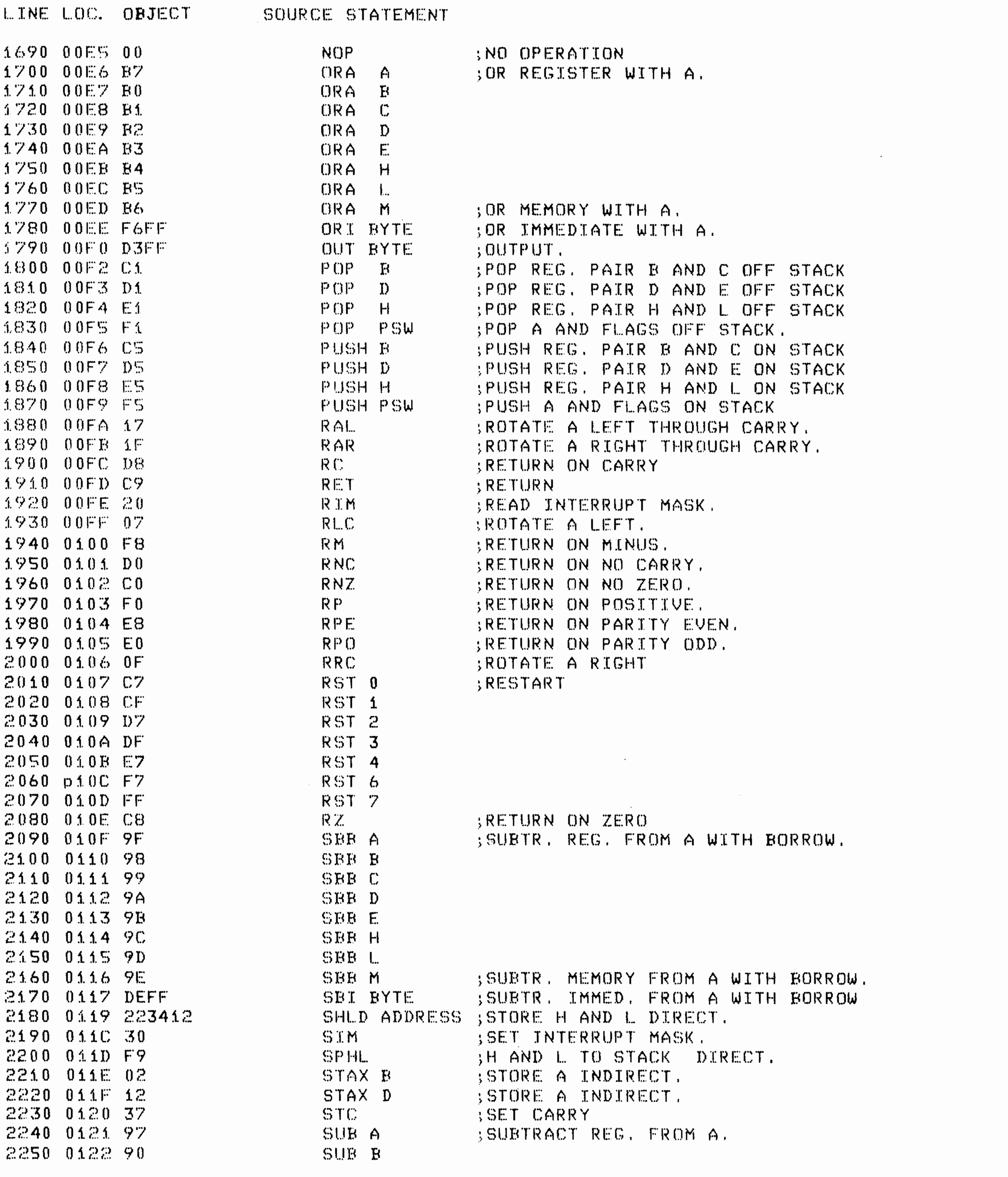
| | | | |
|---|---|---|---|
| + | Plus sign | ADDITION | |
| – | Minus sign | SUBTRACTION | |
| * | Asterisk | MULTIPLICATION | |
| / | Slash | DIVISION | |
| ( | Left Parenthesis | LEFT PARANTHESIS | |
| ) | Right Parenthesis | RIGHT PARENTHESIS | |
| $ | Dollar sign | PROGRAM COUNTER AND CONTROL LINE ID | |
| ' | Single quote | STRING DELIMITER | |
| " | Double quote | STRING DELIMITER | |
| : | Colon | LABEL DELIMITER SUFFIX | |
| ; | Semicolon | COMMENT FIELD DELIMITER | |
| B | | BINARY NUMBER SUFFIX | B, D, H, O, Q, are also alpha characters |
| D | | DECIMAL NUMBER SUFFIX | |
| H | | HEX NUMBER SUFFIX | |
| O | | OCTAL NUMBER SUFFIX | |
| Q | | OCTAL NUMBER SUFFIX | |
| SPACE | | FIELD SEPARATOR OR SYMBOL TERMINATOR | |
| HOR TAB | | FIELD SEPARATOR OR SYMBOL TERMINATOR | |
| , | Comma | OPERAND SEPARATOR | |
| = | Equals sign | MACRO DUMMY ARGUMENT PREFIX | |
| ^ | | MACRO PRE-EVALUATION PREFIX | |

# Instructions

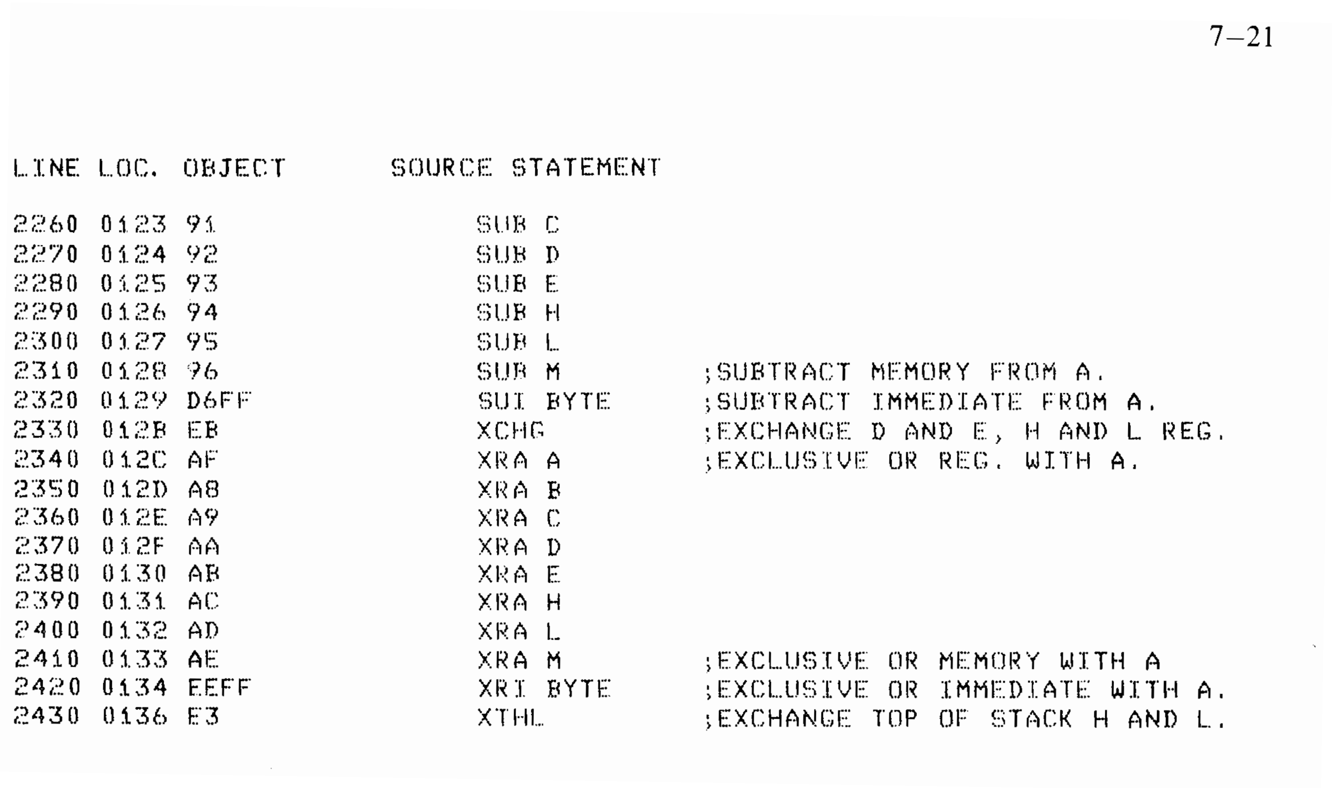
```
LINE LOC. OBJECT      SOURCE STATEMENT

  10 0000 CEFF            ACI  BYTE      ;ADD IMMEDIATE TO A WITH CARRY.
  20 0002 8F              ADC  A         ;ADD REGISTER TO A WITH CARRY.
  30 0003 88              ADC  B
  40 0004 89              ADC  C
  50 0005 8A              ADC  D
  60 0006 8B              ADC  E
  70 0007 8C              ADC  H
  80 0008 8D              ADC  L
  90 0009 8E              ADC  M         ;ADD MEMORY TO A WITH CARRY
 100 000A 87              ADD  A         ;ADD REGISTER TO A
 110 000B 80              ADD  B
 120 000C 81              ADD  C
 130 000D 82              ADD  D
 140 000E 83              ADD  E
 150 000F 84              ADD  H
 160 0010 85              ADD  L
 170 0011 86              ADD  M         ;ADD MEMORY TO A.
 180 0012 C6FF            ADI  BYTE      ;ADD IMMEDIATE TO A
 190 0014 A7              ANA  A         ;AND REGISTER WITH A
 200 0015 A0              ANA  B
 210 0016 A1              ANA  C
 220 0017 A2              ANA  D
 230 0018 A3              ANA  E
 240 0019 A4              ANA  H
 250 001A A5              ANA  L
 260 001B A6              ANA  M         ;ADD MEMORY WITH A.
 270 001C E6FF            ANI  BYTE      ;AND IMMEDIATE WITH A.
 280 001E CD3412          CALL ADDRESS   ;CALL UNCONDITIONAL.
 290 0021 DC3412          CC   ADDRESS   ;CALL ON CARRY.
 300 0024 FC3412          CM   ADDRESS   ;CALL ON MINUS.
 310 0027 2F              CMA            ;COMPLEMENT A
 320 0028 3F              CMC            ;COMPLEMENT CARRY.
 330 0029 BF              CMP   A        ;COMPARE REGISTER WITH A.
 340 002A B8              CMP   B
 350 002B B9              CMP   C
 360 002C BA              CMP   D
 370 002D BB              CMP   E
 380 002E BC              CMP   H
 390 002F BD              CMP   L
 400 0030 BE              CMP   M        ;COMPARE MEMORY WITH A.
 410 0031 D43412          CNC   ADDRESS  ;CALL ON NO CARRY.
 420 0034 C43412          CNZ   ADDRESS  ;CALL ON NO ZERO.
 430 0037 F43412          CP    ADDRESS  ;CALL ON POSITIVE.
 440 003A EC3412          CPE   ADDRESS  ;CALL ON PARITY EVEN.
 450 003D FEFF            CPI   BYTE     ;COMPARE IMMEDIATE WITH A.
 460 003F E43412          CPO   ADDRESS  ;CALL ON PARITY ODD.
 470 0042 CC3412          CZ    ADDRESS  ;CALL ON ZERO.
 480 0045 27              DAA            ;DECIMAL ADJUST A.
 490 0046 09              DAD B          ;ADD B AND C TO H AND L.
 500 0047 19              DAD D          ;ADD D AND E TO H AND L.
 510 0048 29              DAD H          ;ADD H AND L TO H AND L.
 520 0049 39              DAD SP         ;DECREMENT STACK POINTER.
 530 004A 3D              DCR A          ;DECREMENT REGISTER
 540 004B 05              DCR B
```

```
LINE LOC. OBJECT       SOURCE STATEMENT

 550 004C 0D                DCR C
 560 004D 15                DCR D
 570 004E 1D                DCR E
 580 004F 25                DCR H
 590 0050 2D                DCR L
 600 0051 35                DCR M         ;DECREMENT MEMORY.
 610 0052 0B                DCX B         ;DECREMENT B AND C.
 620 0053 1B                DCX D         ;DECREMENT D AND E.
 630 0054 2B                DCX H         ;DECREMENT H AND L.
 640 0055 3B                DCX SP        ;DECREMENT STACK POINTER
 650 0056 F3                DI            ;DISABLE INTERRUPT.
 660 0057 FB                EI            ;ENABLE INTERRUPT.
 670 0058 76                HLT           ;HALT.
 680 0059 DBFF              IN BYTE       ;INPUT.
 690 005B 3C                INR A         ;INCREMENT REGISTER.
 700 005C 04                INR B
 710 005D 0C                INR C
 720 005E 14                INR D
 730 005F 1C                INR E
 740 0060 24                INR H
 750 0061 2C                INR L
 760 0062 34                INR M         ;INCREMENT MEMORY.
 770 0063 03                INX B         ;INCREMENT B AND C REGISTERS.
 780 0064 13                INX D         ;INCREMENT D AND E REGISTERS.
 790 0065 23                INX H         ;INCREMENT H AND L REGISTERS.
 800 0066 33                INX SP        ;INCREMENT STACK POINTER
 810 0067 DA3412            JC   ADDRESS ;JUMP ON CARRY.
 820 006A FA3412            JM   ADDRESS ;JUMP ON MINUS.
 830 006D C33412            JMP  ADDRESS ;JUMP UNCONDITIONAL.
 840 0070 D23412            JNC  ADDRESS ;JUMP ON NO CARRY.
 850 0073 C23412            JNZ  ADDRESS ;JUMP ON NO ZERO.
 860 0076 F23412            JP   ADDRESS ;JUMP ON POSITIVE
 870 0079 EA3412            JPE  ADDRESS ;JUMP ON PARITY EVEN.
 880 007C E23412            JPO  ADDRESS ;JUMP ON PARITY IDD.
 890 007F CA3412            JZ   ADDRESS ;JUMP ON ZERO.
 900 0082 3A3412            LDA  ADDRESS ;LOAD A DIRECT.
 910 0085 0A                LDAX B        ;LOAD A INDIRECT.
 920 0086 1A                LDAX D        ;LOAD A INDIRECT.
 930 0087 2A3412            LHLD ADDRESS ;LOAD H AND L DIRECT
 940 008A 017856            LXI  B,WORD  ;LOAD IMMEDIATE REG. PAIR B AND C
 950 008D 117856            LXI  D,WORD  ;LOAD IMMEDIATE REG. PAIR D AND E
 960 0090 217856            LXI  H,WORD  ;LOAD IMMEDIATE REG. PAIR H AND L
 970 0093 317856            LXI  SP,WORD ;LOAD IMMEDIATE STACK POINTER.
 980 0096 7F                MOV  A,A      ;MOVE REG. TO REG.
 990 0097 78                MOV  A,B
1000 0098 79                MOV  A,C
1010 0099 7A                MOV  A,D
1020 009A 7B                MOV  A,E
1030 009B 7C                MOV  A,H
1040 009C 7D                MOV  A,L
1050 009D 7E                MOV  A,M      ;MOVE MEMORY TO REGISTER
1060 009E 47                MOV  B,A      ;MOVE REG. TO REG.
1070 009F 40                MOV  B,B
1080 00A0 41                MOV  B,C
1090 00A1 42                MOV  B,D
1100 00A2 43                MOV  B,E
1110 00A3 44                MOV  B,H
```

```
LINE LOC. OBJECT      SOURCE STATEMENT

1120 00A4 45                MOV  B,L
1130 00A5 46                MOV  B,M       ;MOVE MEMORY TO REGISTER
1140 00A6 4F                MOV  C,A       ;MOVE REG. TO REG.
1150 00A7 48                MOV  C,B
1160 00A8 49                MOV  C,C
1170 00A9 4A                MOV  C,D
1180 00AA 4B                MOV  C,E
1190 00AB 4C                MOV  C,H
1200 00AC 4D                MOV  C,L
1210 00AD 4E                MOV  C,M       ;MOVE MEMORY TO REGISTER
1220 00AE 57                MOV  D,A       ;MOVE REG. TO REG.
1230 00AF 50                MOV  D,B
1240 00B0 51                MOV  D,C
1250 00B1 52                MOV  D,D
1260 00B2 53                MOV  D,E
1270 00B3 54                MOV  D,H
1280 00B4 55                MOV  D,L
1290 00B5 56                MOV  D,M       ;MOVE MEMORY TO REGISTER
1300 00B6 5F                MOV  E,A       ;MOVE REG. TO REG.
1310 00B7 58                MOV  E,B
1320 00B8 59                MOV  E,C
1330 00B9 5A                MOV  E,D
1340 00BA 5B                MOV  E,E
1350 00BB 5C                MOV  E,H
1360 00BC 5D                MOV  E,L
1370 00BD 5E                MOV  E,M       ;MOVE MEMORY TO REGISTER
1380 00BE 67                MOV  H,A       ;MOVE REG. TO REG.
1390 00BF 60                MOV  H,B
1400 00C0 61                MOV  H,C
1410 00C1 62                MOV  H,D
1420 00C2 63                MOV  H,E
1430 00C3 64                MOV  H,H
1440 00C4 65                MOV  H,L
1450 00C5 66                MOV  H,M       ;MOVE MEMORY TO REGISTER
1460 00C6 6F                MOV  L,A       ;MOVE REG. TO REG.
1470 00C7 68                MOV  L,B
1480 00C8 69                MOV  L,C
1490 00C9 6A                MOV  L,D
1500 00CA 6B                MOV  L,E
1510 00CB 6C                MOV  L,H
1520 00CC 6D                MOV  L,L
1530 00CD 6E                MOV  L,M       ;MOVE MEMORY TO REGISTER
1540 00CE 77                MOV  M,A       ;MOVE REGISTER TO MEMORY
1550 00CF 70                MOV  M,B
1560 00D0 71                MOV  M,C
1570 00D1 72                MOV  M,D
1580 00D2 73                MOV  M,E
1590 00D3 74                MOV  M,H
1600 00D4 75                MOV  M,L
1610 00D5 3EFF              MVI  A,BYTE    ;MOVE IMMEDIATE REGISTER
1620 00D7 06FF              MVI  B,BYTE
1630 00D9 0EFF              MVI  C,BYTE
1640 00DB 16FF              MVI  D,BYTE
1650 00DD 1EFF              MVI  E,BYTE
1660 00DF 26FF              MVI  H,BYTE
1670 00E1 2EFF              MVI  L,BYTE
1680 00E3 36FF              MVI  M,BYTE    ;MOVE IMMEDIATE MEMORY.
```



```
LINE LOC. OBJECT       SOURCE STATEMENT

1690 00E5 00                   NOP             ;NO OPERATION
1700 00E6 B7                   ORA   A         ;OR REGISTER WITH A.
1710 00E7 B0                   ORA   B
1720 00E8 B1                   ORA   C
1730 00E9 B2                   ORA   D
1740 00EA B3                   ORA   E
1750 00EB B4                   ORA   H
1760 00EC B5                   ORA   L
1770 00ED B6                   ORA   M         ;OR MEMORY WITH A.
1780 00EE F6FF                 ORI  BYTE       ;OR IMMEDIATE WITH A.
1790 00F0 D3FF                 OUT  BYTE       ;OUTPUT.
1800 00F2 C1                   POP   B         ;POP REG. PAIR B AND C OFF STACK
1810 00F3 D1                   POP   D         ;POP REG. PAIR D AND E OFF STACK
1820 00F4 E1                   POP   H         ;POP REG. PAIR H AND L OFF STACK
1830 00F5 F1                   POP   PSW       ;POP A AND FLAGS OFF STACK.
1840 00F6 C5                   PUSH B          ;PUSH REG. PAIR B AND C ON STACK
1850 00F7 D5                   PUSH D          ;PUSH REG. PAIR D AND E ON STACK
1860 00F8 E5                   PUSH H          ;PUSH REG. PAIR H AND L ON STACK
1870 00F9 F5                   PUSH PSW        ;PUSH A AND FLAGS ON STACK
1880 00FA 17                   RAL             ;ROTATE A LEFT THROUGH CARRY.
1890 00FB 1F                   RAR             ;ROTATE A RIGHT THROUGH CARRY.
1900 00FC D8                   RC              ;RETURN ON CARRY
1910 00FD C9                   RET             ;RETURN
1920 00FE 20                   RIM             ;READ INTERRUPT MASK.
1930 00FF 07                   RLC             ;ROTATE A LEFT.
1940 0100 F8                   RM              ;RETURN ON MINUS.
1950 0101 D0                   RNC             ;RETURN ON NO CARRY.
1960 0102 C0                   RNZ             ;RETURN ON NO ZERO.
1970 0103 F0                   RP              ;RETURN ON POSITIVE.
1980 0104 E8                   RPE             ;RETURN ON PARITY EVEN.
1990 0105 E0                   RPO             ;RETURN ON PARITY ODD.
2000 0106 0F                   RRC             ;ROTATE A RIGHT
2010 0107 C7                   RST 0           ;RESTART
2020 0108 CF                   RST 1
2030 0109 D7                   RST 2
2040 010A DF                   RST 3
2050 010B E7                   RST 4
2060 010C F7                   RST 6
2070 010D FF                   RST 7
2080 010E C8                   RZ              ;RETURN ON ZERO
2090 010F 9F                   SBB A           ;SUBTR. REG. FROM A WITH BORROW.
2100 0110 98                   SBB B
2110 0111 99                   SBB C
2120 0112 9A                   SBB D
2130 0113 9B                   SBB E
2140 0114 9C                   SBB H
2150 0115 9D                   SBB L
2160 0116 9E                   SBB M           ;SUBTR. MEMORY FROM A WITH BORROW.
2170 0117 DEFF                 SBI BYTE        ;SUBTR. IMMED. FROM A WITH BORROW
2180 0119 223412               SHLD ADDRESS    ;STORE H AND L DIRECT.
2190 011C 30                   SIM             ;SET INTERRUPT MASK.
2200 011D F9                   SPHL            ;H AND L TO STACK  DIRECT.
2210 011E 02                   STAX B          ;STORE A INDIRECT.
2220 011F 12                   STAX D          ;STORE A INDIRECT.
2230 0120 37                   STC             ;SET CARRY
2240 0121 97                   SUB A           ;SUBTRACT REG. FROM A.
2250 0122 90                   SUB B
```



```
LINE LOC. OBJECT       SOURCE STATEMENT

2260 0123 91                 SUB C
2270 0124 92                 SUB D
2280 0125 93                 SUB E
2290 0126 94                 SUB H
2300 0127 95                 SUB L
2310 0128 96                 SUB M          ;SUBTRACT MEMORY FROM A.
2320 0129 D6FF               SUI BYTE       ;SUBTRACT IMMEDIATE FROM A.
2330 012B EB                 XCHG           ;EXCHANGE D AND E, H AND L REG.
2340 012C AF                 XRA A          ;EXCLUSIVE OR REG. WITH A.
2350 012D A8                 XRA B
2360 012E A9                 XRA C
2370 012F AA                 XRA D
2380 0130 AB                 XRA E
2390 0131 AC                 XRA H
2400 0132 AD                 XRA L
2410 0133 AE                 XRA M          ;EXCLUSIVE OR MEMORY WITH A
2420 0134 EEFF               XRI BYTE       ;EXCLUSIVE OR IMMEDIATE WITH A.
2430 0136 E3                 XTHL           ;EXCHANGE TOP OF STACK H AND L.
```

# Differences between HP and the 8080/85

The following is a list of differences, known to us at the time of printing, that exist between the Hewlett-Packard software and other 8080/85 Assembler descriptions.

a) The character set has been expanded.

b) HP software accepts Labels in place of Names.

c) DB and DW pseudo mnemonics accept more than 8 operands.

d) The following pseudo mnemonics have been added:

FAIL
SPC
IFC
IFNC

e) The Macro parameter syntax has been changed. The Assembler generated symbol mechanism has been changed (no local directive). The nul operator is replaced by IFC and IFNC directives.

f) The Assembler does not have any relocation features.

g) Instructions in the Operand field are not allowed.

h) EQU and SET pseudo mnemonics do not allow register-type Operands in their Operand fields.

i) The following 8080/85 controls are not implemented:

| | |
|---|---|
| NOOBJECT | Replaced by OBJECT. |
| DEBUG | No debug facilities. |
| NODEBUG | |
| PRINT | List output on the mass storage is not provided. |
| NOPRINT | |
| PAGELENGTH | Printer specifications are set-up during Initialization. |
| PAGEWIDTH | |
| MACRODEBUG | No debug facilities. |
| NOMACRODEBUG | |

# Section Seven
# 6800 Assembly Language Instructions

## Introduction

The following is a description of the Assembly Language instructions needed to write Source code for the 6800 Microprocessor types. We recommend that you make yourself acquainted with this explanation as it differs in some respects from the description given by 6800 Programming manuals.

## Source Line Format

A program written in Assembler Language is called a Source code. It consists of symbolic commands called statements. Each statement is written on a single line and may consist of from one to four fields. From left to right the four fields are:

(1) LABEL (2) INSTRUCTION (3) OPERAND (4) COMMENT

The following pages describe each field individually. An illustration of the format pattern is given below.

```
[ <LABEL> ]   <INSTRUCTION>   [ <OPER> [, <OPER> . . . ] ] [ <COMMENT> ]

LABEL space INSTRUCTION space OPERAND, OPERAND space COMMENT
        |                 |                      |
   One or more       At least one           At least one
```

An asterisk (*) in the first character position indicates a Comment line and the entire line is ignored by the Assembler.

# Label Field

A Label is required for statements involved in the definition of symbols and occasionally at destinations of branch and jump instructions.

A Label is a symbol whose value is the location where the instruction is assembled. A Label must begin in the first character position of a statement. Any number of characters (limited by the line length) may be used. However, the Assembler only recognises up to eight alphanumeric characters whereby the first character must be alphabetic.

A Label may be used in any executable instruction at your option. However, it must be used in a statement which includes the Assembler directives SET, EQU, MACRO and it will be identical with the Symbol which the SET, EQU, MACRO directive is defining.

The Assembler maintains a location counter to provide addresses for the symbols in the Label field. When a symbol is found in the Label field, the Assembler places the symbol and the corresponding location counter value in a symbol table.

# Instruction Field

Instructions are machine or pseudo mnemonics and must be present in all statements except when the statement consists only of a Comment. Instructions must not start in the first character position.

Pseudo mnemonics are described later in this Section and a list of 6800 machine mnemonics is given at the back of this Section.

# Operand Field

The Operand field identifies the data to be acted upon by the specified instruction. Some instructions require no Operand, while others require one or more.

The Operand field is separated from the Instruction by one or more blanks. If more than one Operand is required, they are separated from each other by commas. Imbedded blanks are not allowed.

The type of information placed in the Operand field depends upon the particular instruction. The following are recognized by the Assembler as Operands:

Numbers
Symbols
Expressions
ASCII Strings
Location Counter Symbol (*)

The listed Operands are now described in more detail.

## Numbers

The Assembler accepts numbers in several different bases: binary, octal, decimal, and hexadecimal. Numbers (constants) are accepted in the following formats.

| | | |
|---|---|---|
| <Number> | Specifies a Decimal number | 0–65536 |
| <Number>D | Specifies a Decimal number | 0D–65536D |
| $<Number> | Specifies a Hexadecimal number | $0–$FFFF |
| <Number>H | Specifies a Hexadecimal number | 0H–0FFFFH |
| @<Number> | Specifies an Octal number | @0–@177777 |
| <Number>O | Specifies an Octal number | 0O–177777O |
| <Number>Q | Specifies an Octal number | 0Q–177777Q |
| %<Number> | Specifies a Binary number | %0–%1111111111111111 |
| <Number>B | Specifies a Binary number | 0B–1111111111111111B |

The number must begin with a numeric character. For example:

| | |
|---|---|
| A9FFH | Illegal (will be recognized as a symbol). |
| 0A9FFH | Correct specification. |

## Symbols

Symbols can be divided into three separate identities, Reserved, User definable, and Assembler generated symbols.

Reserved symbols are those that already have a special meaning to the Assembler and therefore cannot appear as user-defined symbols. The mnemonic names for microprocessor instructions and the Assembler directives are all reserved symbols.

A symbol may consist of up to 74 characters, however only the first eight characters are significant.

A symbol must not be any of the single characters A, B, or X as they define microprocessor registers for the programmer and are not allowed in expressions.

The Assembler recognises the special symbol * as a symbol for the location counter or as a multiplication depending on the context.

For example:

> SYMBOL***2 means that the SYMBOL is multiplied by the value of the location counter and the result is then multiplied by two.

## Expressions

An expression is a combination of symbols and numbers separated by one of the arithmetic operators +, –, *, /, ( ). The Assembler evaluates expressions algebraically from left to right. There is no hierarchy of precedence among the arithmetic operators. However, this can be influenced by the use of parentheses. Nesting is only limited by the length of the expression that can be inserted into the Source line. An intermediate result will be truncated to an integer value if it is obtained during the evaluation of an expression.

Note: The Assembler must complete the numeric evaluation of symbols and expressions in two passes. Only one level of forward addressing is permitted in the use of symbols or expressions in the Operand field of Source statements.

For instance, in the example below, P would not be evaluated.

```
P  EQU  R
R  EQU  Q
Q  EQU  5
```

## ASCII Strings

The apostrophe (') instructs the Assembler to translate the next character into the corresponding 7-bit ASCII code. Any ASCII character can be used. For example;

'S or '!

Double quote marks can also be used to define an ASCII String. In this case, the double quotes determine the String delimiter while a single apostrophe is regarded as a character within the String. For example:

"TODAY'S DATE"

ASCII Strings of length 1 or 2 characters can be used in expressions like any other number. A 1 character String is evaluated to the unsigned 8-bit value, whereas a 2 character String will result in a 16-bit number.

## Void Operands

Void Operands are missing Operands. They are separated by commas, and are evaluated as a zero. For example:

OPERAND , , X    is evaluated as    OPERAND , 0 , X

## Direct and Extended Addressing

For Direct addressing, the Source code is translated into two bytes of machine code. The second byte will contain the address in unsigned 8-bit binary form.

For Extended addressing, the Source code is translated into three bytes of machine code. The second of these bytes will contain the highest 8 bits of the address. The third byte will contain the lowest 8 bits of the address. The contents of the second and third bytes will both be coded in unsigned 8-bit binary form.

For instructions which may use the Direct mode of addressing as well as the Extended mode, the Assembler selects the mode according to the following rule:

The Assembler will select the Direct addressing if the numerical address is defined at this time and is in the range from zero to 255 (decimal) It will select Extended addressing if the numerical address exceeds 255 (decimal).

## Indirect Addressing

The data, for obtaining the numerical address, may be written in any of the following formats:

```
X
,X
Number, X
Symbol, X
Expression, X
```

The single character "X" informs the Assembler that the indexed mode is to be used, the character "X" being reserved to denote the index register. When used alone, the format "X" instructs the Assembler that the address of the Operand is identical with the contents of the index register. (This format has the same effect on the assembly as if 0, X had been written). If a symbol or an expression is used rather than a number, the Assembler will find or compute a numerical value of that symbol or expression. The Source code must then include other statements which define a numerical value for the symbol or which enable the Assembler to compute a numerical value for the symbol or expression.

## Immediate Addressing

The Assembler can be instructed to select the Immediate address mode for an Instruction by placing the # symbol in front of the Operand.

## Comment Field

Comments can be optionally used at the end of a Source line. If present they are ignored by the Assembler during translation and appear only in the program listing. Separation of a Comment and Operand field is made by at least one blank.

An asterisk (*) in the first character position indicates a Comment line and the entire line is ignored by the Assembler.

## Assembler Directives

The Assembler directives, sometimes called pseudo mnemonics, are instructions, entered into the Source code, which direct the Assembler when it generates the Object code. They control the generation of the Assembler program and define the format of the Assembler output listing.

The following examples are taken from an assembled program. Both the Source code and Assembler program are listed at the end of these examples in their entirety.

**END**

[ <LABEL> ]   **END**   [ <COMMENT> ]

The END directive will stop the Assembler program. If the END directive is missing, the Assembler program is terminated by the end of the file. However, the Assembler does not stop if an END directive is part of a file called by the SOURCE control. The directive is not translated into Object code.

**EQU** – Equate

<LABEL>   **EQU**   <EXPRESSION>   [ <COMMENT> ]

The EQU directive assigns a value, given by the Operand field, to the symbol specified in the Label field. However, this symbol cannot be redefined.

Example

```
160      00D7          VALUE    EQU      @327       OCTAL VALUE
```

## FAIL

[ <LABEL> ]    **FAIL**    'USER DEFINABLE ERROR MESSAGE'

The FAIL directive forces the Assembler to printout a user defined error message. This directive would be contained in your Source code as an indication that unwanted code has been Assembled. It is often used with the IF-ELSE-ENDIF directives to ensure that the correct directive has been Assembled.

Example

```
**      ASSEMBLING PART 2**
  360                          FAIL     ASSEMBLING PART 2
```

**FCB** – Form constant byte

[ <LABEL> ]    **FCB**    <EXPRESSION> [ , <EXPRESSION> . . . ] [ <COMMENT> ]

An 8-bit value corresponding to each Operand is stored in a byte of the Object program. This directive may have one or more Operands, separated by commas. The number of Operands is limited by the Source line length of 80 characters. The Operand can be any constant, symbol, or expression evaluating to an 8-bit value. Void Operands are allowed.

When the Label is present the address of the first generated byte is assigned to the symbol represented by the Label.

Example

```
210 0100 AC                 FCB      254Q      OCTAL VALUE
```

**FCC** – Form constant character.

[ <LABEL> ]    **FCC**    /TEXT/    [ <COMMENT> ]
[ <LABEL> ]    **FCC**    COUNT, TEXT

The FCC directive translates a string of ASCII characters into their 7-bit codes and stores them into successive bytes of the Object code. The Label is assigned to the first generated byte.

The Operand can be defined in two ways:

1) /TEXT/
As the text enclosed between identical delimiters. In this case, the delimiters can be any single character. The text must not begin with a comma if the delimiter is a number.

2) COUNT, TEXT
COUNT (a given number) specifies how many ASCII characters are to be generated and the text commences after the first comma following the COUNT. Should COUNT be longer than the text, spaces are inserted to fill the COUNT to a maximum of 80 characters.

Example

```
220 0101 54455854           FCC      /TEXT/    ASCII STRING INTO MEMORY
```

**FDB** – Form double byte

[ <LABEL> ]  **FDB**  <EXPRESSION> [ , <EXPRESSION> . . . ]  [ <COMMENT> ]

The FDB directive may have one or more Operands separated by commas. The 16-bit unsigned binary number, corresponding to the value of each Operand, is stored in two bytes of the Object program. If there is more than one Operand, they are stored in successive bytes. An FDB directive may be written with a Label.

Example

```
200 00FE 430C          CONSTANT FDB        0414140   OCTAL VALUE
```

**IF-ELSE-ENDIF**

[ <LABEL> ]  **IF**  <EXPRESSION>  [ <COMMENT> ]

. CODE SEQUENCE 1

[ <LABEL> ]  **ELSE**  [ <COMMENT> ]

. CODE SEQUENCE 2

[ <LABEL> ]  **ENDIF**  [ <COMMENT> ]

A code sequence between IF-ENDIF directive will be assembled when the expression evaluates to a value other than zero.

When the ELSE directive is included, the code sequence between the IF and the ELSE directive is assembled when the expression, following IF, evaluates to a value other than zero. When the expression equals zero, the code sequence between the ELSE and ENDIF directives is assembled.

Example

```
   280                          IF     VALUE-@327     CONDITIONAL ASSEMBLY START
   290
   300                START     LDA    A    #00100110B  THIS LINE NOT ASSEMBLED
   310
   320                          ELSE
   330
   340 010F 8647      START     LDA    A    #'G        THIS LINE IS ASSEMBLED
   350
**       ASSEMBLING PART 2**
   360                          FAIL        ASSEMBLING PART 2
   370
   380                          ENDIF                  CONDITIONAL ASSEMBLY END
```

**IFC-ELSE-ENDIF**

[ <LABEL> ]   **IFC**   <OPER1>,<OPER2>   [ <COMMENT> ]

CODE SEQUENCE

[ <LABEL> ]   **ENDIF**   [ <COMMENT> ]

The IFC directive is similar to the IF directive. However, the code sequence is assembled if the two Operands, following the IFC directive, are equal on a character for character basis. No evaluation is performed on the two Operands. Operands can be Symbols or Strings in double quotes. You use the ELSE directive in the same manner as with the IF directive.

**IFNC-ELSE-ENDIF**

[ <LABEL> ]   **IFNC**   <OPER1>,<OPER2>   [ <COMMENT> ]

CODE SEQUENCE

[ <LABEL> ]   **ENDIF**   [ <COMMENT> ]

The IFNC directive is similar to the IFC directive, whereby the code sequence is Assembled if the two Operands, following the IFNC directive are not equal on a character for character basis.

**MON**

[ <LABEL> ]   **MON**   <START ADDRESS>   [ <COMMENT> ]

The MON directive is used instead of the END directive when the microprocessor system requires the start address specification in Object code. For further information refer to the Console module.

**NAM** -- Name

[ <LABEL> ]   **NAM**   <PAGE HEADING>

This directive defines the heading for the top of the page. This directive may be used anywhere in the program. The page name appears after it has been defined and it may be changed at any time. The heading appears on page one if written in the first Source line

Example

```
10                 NAM     ASSEMBLER DIRECTIVE DEMO PROGRAM
```

**ORG** – Origin

[ <LABEL> ]  **ORG**  <EXPRESSION>  [ <COMMENT> ]

This directive defines the numerical address of the first byte of machine code which results from the assembly of the immediately subsequent section of a Source program. There may be any number of ORG statements in a program. The ORG directive sets the location counter to the value expressed in the Operand field.

The Operand field may contain the actual value (decimal, hexadecimal, octal, or binary) to which the location counter is to be set. Alternatively, the Operand field may contain a symbol or an expression which can be assigned a numeric value by the Assembler. Symbols must be previously defined.

The location counter is initialized to 0000H before each assembly. If no ORG statement appears at the beginning of the program, the location counter begins as if an ORG zero had been entered.

Example

```
30 0000                     ORG        000F9H     HEXADECIMAL ADDRESS
```

**PAGE**

[ <LABEL> ]  **PAGE**  [ <COMMENT> ]

The PAGE directive causes the Assembler to move the paper to the top of the next page. This directive does not appear on the program listing.

**RMB** – Reserve memory bytes

[ <LABEL> ]  **RMB**  <EXPRESSION>  [ <COMMENT> ]

The RMB directive causes the location counter to be increased by the value given in the Operand field. This reserves a block of memory equal to the size of the expression value. The Operand field may contain the actual number equal to the number of bytes to be reserved or the Operand may be a symbol or an expression which can be assigned a numerical value by the Assembler. When a symbol is used it must have been previously defined. A symbol that has been defined by a Label is assigned the address of the first byte reserved by this directive. The content of the memory block is unchanged. The Assembler only advances the location counter and does not generate any Object code.

Example

```
240 0105             TEMP      RMB   8               CREATE AREA 8 BYTES LONG
```

**SET**

<LABEL>      **SET**      <EXPRESSION>      [ <COMMENT> ]

The SET directive assigns the value, given by the expression to the symbol specified in the Label field. This value remains unchanged until altered by a subsequent SET directive.

Example

```
150       00A0         LABEL    SET       $00A0     HEXADECIMAL ADDRESS
```

**SPC** – Space

[ <LABEL> ]   **SPC**      <EXPRESSION>      [ <COMMENT> ]

The SPC directive inserts a number of blank lines, given by the expression in the program listing. The number of lines to be left blank is stated in the Operand field. When the SPC directive causes the listing to cross page boundaries only those blank lines required to get to the top of the next page are generated.

## Assembler Controls

All Assembler controls are preceeded by the OPT directive. As many Controls as required may be included on one line and each Control must be separated by a comma. If more controls are required that can fit on one line, other lines may be used as long as each line has the OPT directive.

These controls can also be entered after the SOURCE and OBJECT statements in the Assembler program. See Section IX.

[ <LABEL> ] **OPT** [ <CONTROL1> [,<CONTROL2> . . . ] ] [ <COMMENT> ]

Controls

**SOURCE** "<FILE NAME> [msus]" A file on the mass storage device is included in the assembly if specified in the Operand field with this control. If a Source line contains two or more SOURCE controls then the last Source file specified is assembled first. Eight levels of nesting are allowed. Enter the msus if the Source file is not stored on the mass storage device specified in the default file.

**OBJECT** ["<FILE NAME> [msus]"] As long as the file in which the Object code is to be stored was not specified with the OBJECT statement of the Assembler program, see Assembler Module Section IX, it may be specified by inserting this Control in the Source code. The position of this Control in the Source code is not important, as the Object code for the entire Source code will still be stored.

The Object code is not stored if the Object specification does not contain the file name.

If the Object file is specified, no other Object control is accepted.

**SYMBOL** Prints the Symbol Table after the program listing.

***NOSYMBOL** Suppress Symbol Table print.

***XREF** When the Assembler is finished the XREF generator is called and prints the cross reference of all user symbols. If XREF and SYMBOL have been specified, then only an XREF is made.

**NOXREF** Suppresses the cross reference.

**SAVE** The current list Control settings LIST, COND, GEN and MEX are saved but remain valid until explicitly changed. Nesting up to 8 levels is possible.

This Control is useful if you know that the Assembler program will encounter another set of Controls and it is necessary to restore the original value afterwards.

**RESTORE** Recalls the list control settings.

***LIST** Instructs the Assembler to generate a program listing.

**NOLIST** Suppresses the printing of the program listing.

***COND** Includes conditional skipped Source lines in the listing.

**NOCOND** Does not include conditionally skipped lines.

***GEN** Lists all code generated by the FCC directive.

**NOGEN** Lists only the first line generated by the FCC directive.

***MEX** Lists Macro expansion.

**NOMEX** Does not list Macro expansion.

*The Assembler uses these Controls unless otherwise specified.

Examples.

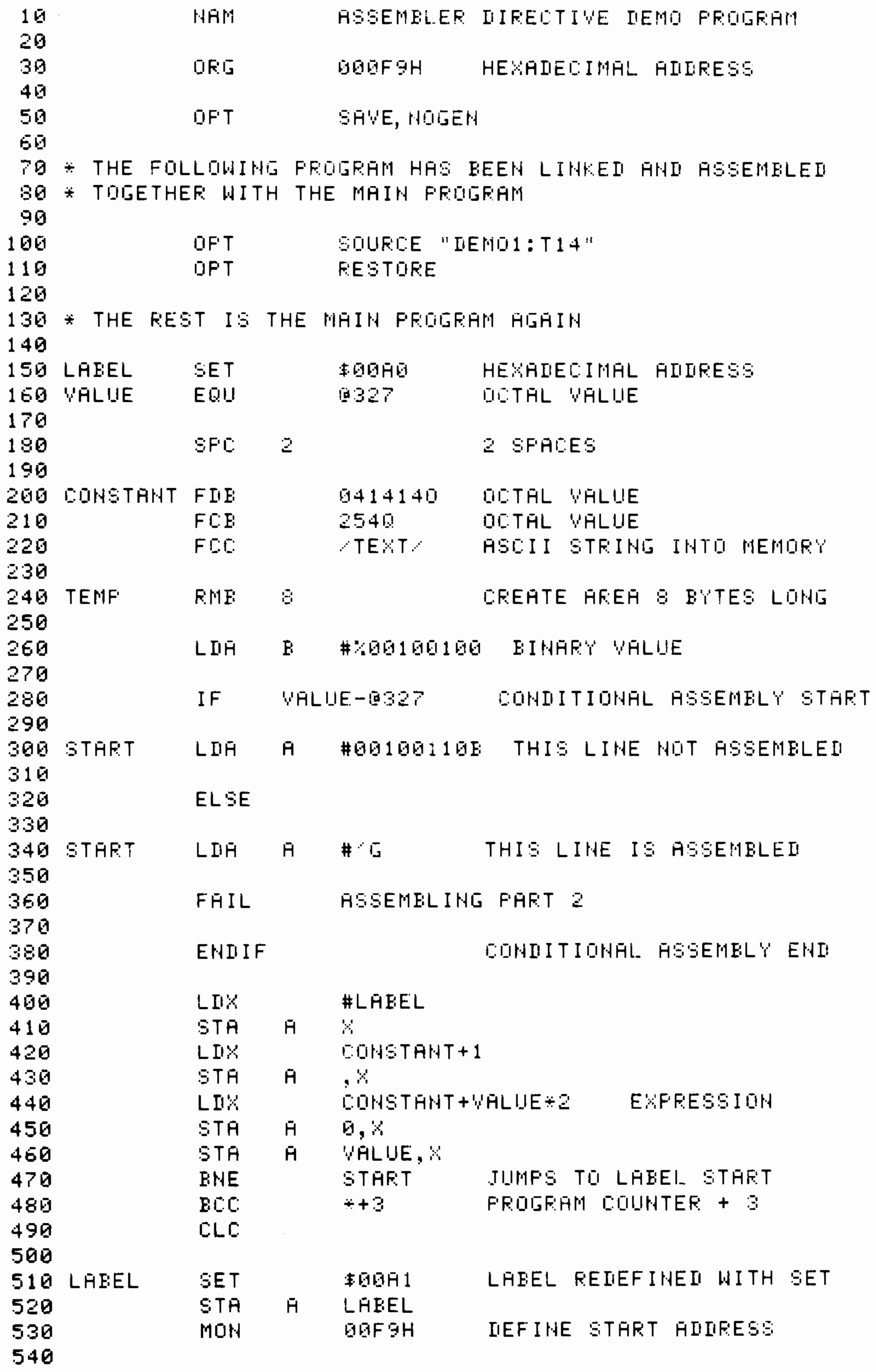
To save list controls, and print only one line generated by the FCC directive, type the following controls.

```
50                               OPT        SAVE, NOGEN
```

Several Source files can be assembled in the same assembly using the SOURCE control. The Source file may be terminated with an END directive. Once the file, called by the SOURCE control from a Source code, has been assembled, the Assembler continues with the file that contained the SOURCE control. Eight nesting levels are possible.

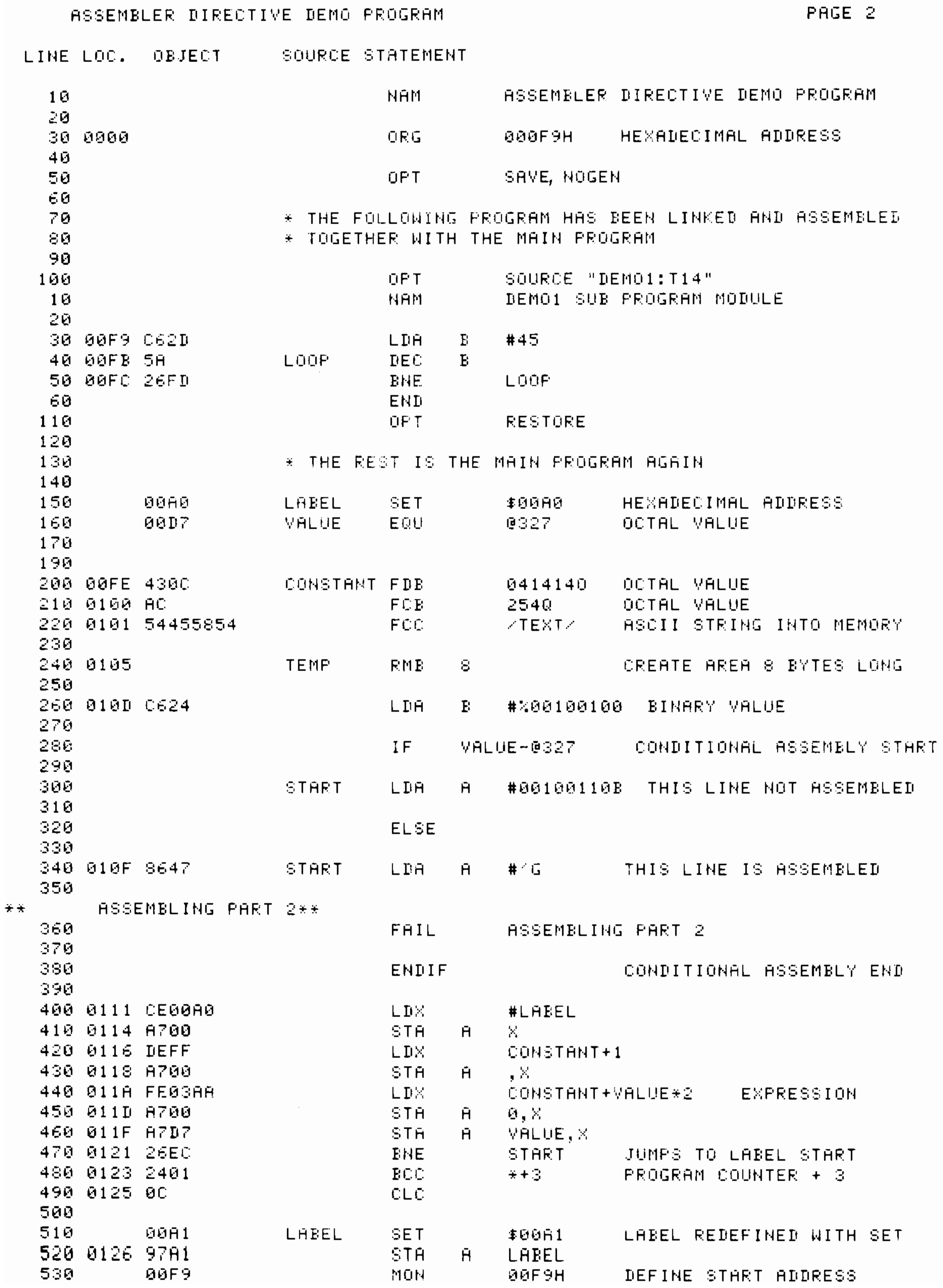
```
100                               OPT        SOURCE "DEMO1:T14"
 10                               NAM        DEMO1 SUB PROGRAM MODULE
 20
 30 00F9 C62D                     LDA   B    #45
 40 00FB 5A              LOOP     DEC   B
 50 00FC 26FD                     BNE        LOOP
 60                               END
110                               OPT        RESTORE
```

# Example of Source Code



```
10          NAM       ASSEMBLER DIRECTIVE DEMO PROGRAM
20
30          ORG       000F9H    HEXADECIMAL ADDRESS
40
50          OPT       SAVE,NOGEN
60
70 * THE FOLLOWING PROGRAM HAS BEEN LINKED AND ASSEMBLED
80 * TOGETHER WITH THE MAIN PROGRAM
90
100         OPT       SOURCE "DEMO1:T14"
110         OPT       RESTORE
120
130 * THE REST IS THE MAIN PROGRAM AGAIN
140
150 LABEL    SET      $00A0     HEXADECIMAL ADDRESS
160 VALUE    EQU      @327      OCTAL VALUE
170
180          SPC  2             2 SPACES
190
200 CONSTANT FDB      041414O   OCTAL VALUE
210          FCB      254Q      OCTAL VALUE
220          FCC      /TEXT/    ASCII STRING INTO MEMORY
230
240 TEMP     RMB  8             CREATE AREA 8 BYTES LONG
250
260          LDA  B   #%00100100  BINARY VALUE
270
280          IF   VALUE-@327      CONDITIONAL ASSEMBLY START
290
300 START    LDA  A   #00100110B  THIS LINE NOT ASSEMBLED
310
320          ELSE
330
340 START    LDA  A   #'G       THIS LINE IS ASSEMBLED
350
360          FAIL     ASSEMBLING PART 2
370
380          ENDIF              CONDITIONAL ASSEMBLY END
390
400          LDX      #LABEL
410          STA  A   X
420          LDX      CONSTANT+1
430          STA  A   ,X
440          LDX      CONSTANT+VALUE*2    EXPRESSION
450          STA  A   0,X
460          STA  A   VALUE,X
470          BNE      START     JUMPS TO LABEL START
480          BCC      *+3       PROGRAM COUNTER + 3
490          CLC
500
510 LABEL    SET      $00A1     LABEL REDEFINED WITH SET
520          STA  A   LABEL
530          MON      00F9H     DEFINE START ADDRESS
540
```

# Example of Program Listing



```
   ASSEMBLER DIRECTIVE DEMO PROGRAM                                      PAGE 2

 LINE LOC.  OBJECT     SOURCE STATEMENT

   10                            NAM       ASSEMBLER DIRECTIVE DEMO PROGRAM
   20
   30 0000                       ORG       000F9H    HEXADECIMAL ADDRESS
   40
   50                            OPT       SAVE, NOGEN
   60
   70                  * THE FOLLOWING PROGRAM HAS BEEN LINKED AND ASSEMBLED
   80                  * TOGETHER WITH THE MAIN PROGRAM
   90
  100                            OPT       SOURCE "DEMO1:T14"
   10                            NAM       DEMO1 SUB PROGRAM MODULE
   20
   30 00F9 C62D                  LDA   B   #45
   40 00FB 5A          LOOP      DEC   B
   50 00FC 26FD                  BNE       LOOP
   60                            END
  110                            OPT       RESTORE
  120
  130                  * THE REST IS THE MAIN PROGRAM AGAIN
  140
  150      00A0        LABEL     SET       $00A0     HEXADECIMAL ADDRESS
  160      00D7        VALUE     EQU       @327      OCTAL VALUE
  170
  190
  200 00FE 430C        CONSTANT FDB        0414140   OCTAL VALUE
  210 0100 AC                    FCB       254Q      OCTAL VALUE
  220 0101 54455854              FCC       /TEXT/    ASCII STRING INTO MEMORY
  230
  240 0105             TEMP      RMB   8             CREATE AREA 8 BYTES LONG
  250
  260 010D C624                  LDA   B   #%00100100  BINARY VALUE
  270
  280                            IF    VALUE-@327     CONDITIONAL ASSEMBLY START
  290
  300                  START     LDA   A   #00100110B  THIS LINE NOT ASSEMBLED
  310
  320                            ELSE
  330
  340 010F 8647        START     LDA   A   #'G       THIS LINE IS ASSEMBLED
  350
**       ASSEMBLING PART 2**
  360                            FAIL      ASSEMBLING PART 2
  370
  380                            ENDIF               CONDITIONAL ASSEMBLY END
  390
  400 0111 CE00A0                LDX       #LABEL
  410 0114 A700                  STA   A   X
  420 0116 DEFF                  LDX       CONSTANT+1
  430 0118 A700                  STA   A   ,X
  440 011A FE03AA                LDX       CONSTANT+VALUE*2    EXPRESSION
  450 011D A700                  STA   A   0,X
  460 011F A7D7                  STA   A   VALUE,X
  470 0121 26EC                  BNE       START     JUMPS TO LABEL START
  480 0123 2401                  BCC       *+3       PROGRAM COUNTER + 3
  490 0125 0C                    CLC
  500
  510      00A1        LABEL     SET       $00A1     LABEL REDEFINED WITH SET
  520 0126 97A1                  STA   A   LABEL
  530      00F9                  MON       00F9H     DEFINE START ADDRESS
```

## Sample Cross Reference

```
SYMBOL       VALUE    DEFINED   REFERENCED

CONSTANT     00FE       200       420    440
LABEL        00A1       150     RE-DEFINED:    510
                                  400    520
LOOP         00FB        40        50
START        010F       340       470
TEMP         0105       240     * UNREFERENCED *
VALUE        00D7       160       280    440    460
```

# Character Set

The Assembler recognizes the complete ASCII character set. Generally all characters are considered to be alphabetic.

The following are exceptions, and the each have a special meaning:

| | | | |
|---|---|---|---|
| + | Plus sign | ADDITION | |
| – | Minus sign | SUBTRACTION | |
| * | Asterisk | MULTIPLICATION | |
| / | Slash | DIVISION | |
| # | | IMMEDIATE ADDRESSING MODE | |
| $ | Dollar | HEXADECIMAL NUMBER PREFIX | |
| @ | Commercial at | OCTAL NUMBER PREFIX | |
| % | Percent | BINARY NUMBER PREFIX | |
| ' | Single quote | SPECIFIES ASCII LITERAL CHARACTER | |
| " | Double quote | STRING DELIMITER | |
| B | | BINARY NUMBER SUFFIX | B, D, H, O, Q are also alpha characters |
| D | | DECIMAL NUMBER SUFFIX | |
| H | | HEXADECIMAL NUMBER SUFFIX | |
| O | | OCTAL NUMBER SUFFIX | |
| Q | | OCTAL NUMBER SUFFIX | |
| SPACE | | FIELD SEPARATOR | |
| HOR TAB | | FIELD SEPARATOR | |
| , | Comma | OPERAND SEPARATOR | |
| = | Equal sign | MACRO DUMMY ARGUMENT PREFIX | |
| ^ | | MACRO PRE-EVALUATION PREFIX | |

# Instructions

```
LINE LOC. OBJECT        SOURCE STATEMENT

 10                            NAM      6800/02 MACRO ASSEMBLER INSTRUCTION SET
 20           00FF      IMM    EQU      $FF     8 BIT IMMEDIATE DATA
 30           FFFF      IMM2   EQU      $FFFF   16 BIT IMMEDIATE DATA
 40           00FF      DIR    EQU      $FF     DIRECT ADDRESSING (BASE PAGE)
 50           FFFF      EXT    EQU      $FFFF   EXTENDED ADDRESSING
 60           0000      IND    EQU      $00     INDEX VAL. FOR INDEX. ADDRESSING
 70           0081      REL    EQU      $81     DISPL. VAL. FOR RELAT. ADDRESSING
 80      0000           START  ORG      $4000   BEGINNING ADDRESS FOR ASSEMBLY
 90      4000 1B               ABA              ADD ACCUMULATORS
100      4001 89FF             ADC A    #IMM    ADD ACCUMULATOR A WITH CARRY
110      4003 99FF             ADC A    DIR
120      4005 B9FFFF           ADC A    EXT
130      4008 A900             ADC A    IND,X
140      400A C9FF             ADC B    #IMM    ADD ACCUMULATOR B WITH CARRY
150      400C D9FF             ADC B    DIR
160      400E F9FFFF           ADC B    EXT
170      4011 E900             ADC B    IND,X
180      4013 8BFF             ADD A    #IMM    ADD ACCUMULATOR A
190      4015 9BFF             ADD A    DIR
200      4017 BBFFFF           ADD A    EXT
210      401A AB00             ADD A    IND,X
220      401C CBFF             ADD B    #IMM    ADD ACCUMULATOR B
230      401E DBFF             ADD B    DIR
240      4020 FBFFFF           ADD B    EXT
250      4023 EB00             ADD B    IND,X
260      4025 84FF             AND A    #IMM    LOGICAL AND ACCUMULATOR A
270      4027 94FF             AND A    DIR
280      4029 B4FFFF           AND A    EXT
290      402C A400             AND A    IND,X
300      402E C4FF             AND B    #IMM    LOGICAL AND ACCUMULATOR B
310      4030 D4FF             AND B    DIR
320      4032 F4FFFF           AND B    EXT
330      4035 E400             AND B    IND,X
340      4037 78FFFF           ASL      EXT     ARITHMETIC SHIFT LEFT OF MEMORY
350      403A 6800             ASL      IND,X
360      403C 48               ASL A            ARITHMETIC SHIFT LEFT OF ACCUM A
370      403D 58               ASL B            SAME AS A
380      403E 77FFFF           ASR      EXT     ARITHMETIC SHIFT RIGHT OF MEMORY
390      4041 6700             ASR      IND,X
400      4043 47               ASR A            ARITHMETIC SHIFT RIGHT OF ACCUM A
410      4044 57               ASR B            SAME AS A
420      4045 247F             BCC      *+REL   BRANCH IF CARRY CLEAR
430      4047 257F             BCS      *+REL   BRANCH IF CARRY SET
440      4049 277F             BEQ      *+REL   BRANCH IF EQUAL (ZERO)
450      404B 2C7F             BGE      *+REL   BRANCH IF >= (ZERO)
460      404D 2E7F             BGT      *+REL   BRANCH IF > (ZERO)
470      404F 227F             BHI      *+REL   BRANCH IF HIGHER (UNSIGNED)
480      4051 85FF             BIT A    #IMM    BIT TEST ACCUMULATOR A AND MEMORY
490      4053 95FF             BIT A    DIR
500      4055 B5FFFF           BIT A    EXT
510      4058 A500             BIT A    IND,X
520      405A C5FF             BIT B    #IMM    BIT TEST ACCUMULATOR B AND MEMORY
530      405C D5FF             BIT B    DIR
540      405E F5FFFF           BIT B    EXT
550      4061 E500             BIT B    IND,X
560      4063 2F7F             BLE      *+REL   BRANCH IF <= (ZERO)
570      4065 237F             BLS      *+REL   BRANCH IF LOWER OR SAME (UNSIGNED
LINE LOC. OBJECT       SOURCE STATEMENT

 580 4067 2D7F               BLT      *+REL    BRANCH IF < (ZERO)
 590 4069 2B7F               BMI      *+REL    BRANCH IF MINUS
 600 406B 267F               BNE      *+REL    BRANCH IF NOT EQUAL ZERO
 610 406D 2A7F               BPL      *+REL    BRANCH IF PLUS
 620 406F 207F               BRA      *+REL    BRANCH ALWAYS
 630 4071 8D7F               BSR      *+REL    BRANCH TO SUBROUTINE
 640 4073 287F               BVC      *+REL    BRANCH IF OVERFLOW CLEAR
 650 4075 297F               BVS      *+REL    BRANCH IF OVERFLOW SET
 660 4077 11                 CBA               COMPARE ACCUMULATORS
 670 4078 0C                 CLC               CLEAR CARRY
 680 4079 0E                 CLI               CLEAR INTERRUPT MASK
 690 407A 7FFFFF             CLR      EXT      CLEAR MEMORY
 700 407D 6F00               CLR      IND,X
 710 407F 4F                 CLR A             CLEAR ACCUMULATOR A
 720 4080 5F                 CLR B             SAME FOR B
 730 4081 0A                 CLV               CLEAR OVERFLOW
 740 4082 81FF               CMP A    #IMM     COMPARE ACCUM. A WITH MEMORY
 750 4084 91FF               CMP A    DIR
 760 4086 B1FFFF             CMP A    EXT
 770 4089 A100               CMP A    IND,X
 780 408B C1FF               CMP B    #IMM     COMPARE ACCUM. B WITH MEMORY
 790 408D D1FF               CMP B    DIR
 800 408F F1FFFF             CMP B    EXT
 810 4092 E100               CMP B    IND,X
 820 4094 73FFFF             COM      EXT      BIT COMPLEMENT MEMORY
 830 4097 6300               COM      IND,X
 840 4099 43                 COM A             BIT COMPLEMENT ACCUMULATOR A
 850 409A 53                 COM B             SAME AS B
 860 409B 8CFFFF             CPX      #IMM2    COMPARE INDEX REG. AND MEMORY
 870 409E 9CFF               CPX      DIR
 880 40A0 BCFFFF             CPX      EXT
 890 40A3 AC00               CPX      IND,X
 900 40A5 19                 DAA               DECIMAL ADJUST ACCUMULATOR A
 910 40A6 7AFFFF             DEC      EXT      DECREMENT MEMORY
 920 40A9 6A00               DEC      IND,X
 930 40AB 4A                 DEC A             DECREMENT ACCUMULATOR A
 940 40AC 5A                 DEC B             SAME FOR B
 950 40AD 34                 DES               DECREMENT STACK POINTER
 960 40AE 09                 DEX               DECREMENT INDEX REGISTER
 970 40AF 88FF               EOR A    #IMM     EXCL. OR ACCUM. A AND MEMORY
 980 40B1 98FF               EOR A    DIR
 990 40B3 B8FFFF             EOR A    EXT
1000 40B6 A800               EOR A    IND,X
1010 40B8 C8FF               EOR B    #IMM     EXCL. OR ACCUM. B AND MEMORY
1020 40BA D8FF               EOR B    DIR
1030 40BC F8FFFF             EOR B    EXT
1040 40BF E800               EOR B    IND,X
1050 40C1 7CFFFF             INC      EXT      INCREMENT MEMORY
1060 40C4 6C00               INC      IND,X
1070 40C6 4C                 INC A             INCREMENT ACCUMULATOR B
1080 40C7 5C                 INC B             SAME AS B
1090 40C8 31                 INS               INCREMENT STACK POINTER
1100 40C9 08                 INX               INCREMENT INDEX REGISTER
1110 40CA 7EFFFF             JMP      EXT      JUMP
1120 40CD 6E00               JMP      IND,X
1130 40CF BDFFFF             JSR      EXT      JUMP TO SUBROUTINE
1140 40D2 AD00               JSR      IND,X
1150 40D4 86FF               LDA A    #IMM     LOAD ACCUMULATOR A
1160 40D6 96FF               LDA A    DIR
1170 40D8 B6FFFF             LDA A    EXT
```

```
LINE LOC. OBJECT      SOURCE STATEMENT

1180 40DB A600               LDA A    IND,X
 190 40DD C6FF               LDA B    #IMM    LOAD ACCUMULATOR B
1200 40DF D6FF               LDA B    DIR
1210 40E1 F6FFFF             LDA B    EXT
1220 40E4 E600               LDA‘B    IND,X
1230 40E6 8EFFFF             LDS      #IMM2   LOAD STACK POINTER
1240 40E9 9EFF               LDS      DIR
1250 40EB BEFFFF             LDS      EXT
1260 40EE AE00               LDS      IND,X
1270 40F0 CEFFFF             LDX      #IMM2   LOAD INDEX REGISTER
1280 40F3 DEFF               LDX      DIR
1290 40F5 FEFFFF             LDX      EXT
1300 40F8 EE00               LDX      IND,X
1310 40FA 74FFFF             LSR      EXT     LOGICAL SHIFT RIGHT OF MEMORY
1320 40FD 6400               LSR      IND,X
1330 40FF 44                 LSR A            LOG. SHIFT RIGHT OF ACCUM. A
1340 4100 54                 LSR B            SAME AS B
1350 4101 70FFFF             NEG      EXT     NEGATE (2'S COMPLEMENT) MEMORY
1360 4104 6000               NEG      IND,X
1370 4106 40                 NEG A            NEGATE (2'S COMPL.) ACCUM. A
1380 4107 50                 NEG B            SAME AS B
1390 4108 01                 NOP              NO OPERATION
1400 4109 8AFF               ORA A    #IMM    INCL. OR ACCUM. A WITH MEMORY
1410 410B 9AFF               ORA A    DIR
1420 410D BAFFFF             ORA A    EXT
1430 4110 AA00               ORA A    IND,X
1440 4112 CAFF               ORA B    #IMM    INCL. OR ACCUM. B WITH MEMORY
1450 4114 DAFF               ORA B    DIR
1460 4116 FAFFFF             ORA B    EXT
1470 4119 EA00               ORA B    IND,X
1480 411B 36                 PSH A            PUSH ACCUMULATOR A ONTO STACK
1490 411C 37                 PSH B            PUSH ACCUMULATOR B ONTO STACK
1500 411D 32                 PUL A            PULL ACCUMULATOR A FROM STACK
1510 411E 33                 PUL B            PULL ACCUMULATOR B FROM STACK
1520 411F 79FFFF             ROL      EXT     ROTATE MEMORY LEFT
1530 4122 6900               ROL      IND,X
1540 4124 49                 ROL A            ROTATE ACCUMULATOR A LEFT
1550 4125 59                 ROL B            ROTATE ACCUMULATOR B LEFT
1560 4126 76FFFF             ROR      EXT     ROTATE MEMORY RIGHT
1570 4129 6600               ROR      IND,X
1580 412B 46                 ROR A            ROTATE ACCUMULATOR A RIGHT
1590 412C 56                 ROR B            SAME FOR B
1600 412D 3B                 RTI              RETURN FROM INTERRUPT
1610 412E 39                 RTS              RETURN FROM SUBROUTINE
1620 412F 10                 SBA              SUBTRACT ACCUMULATORS
1630 4130 82FF               SBC A    #IMM    SUBTR. FROM ACCUM. A WITH CARRY
1640 4132 92FF               SBC A    DIR
1650 4134 B2FFFF             SBC A    EXT
1660 4137 A200               SBC A    IND,X
1670 4139 C2FF               SBC B    #IMM    SUBTR. FROM ACCUM. B WITH CARRY
1680 413B D2FF               SBC B    DIR
1690 413D F2FFFF             SBC B    EXT
1700 4140 E200               SBC B    IND,X
1710 4142 0D                 SEC              SET CARRY
1720 4143 0F                 SEI              SET INTERRUPT MASK
1730 4144 0B                 SEV              SET OVERFLOW
1740 4145 97FF               STA A    DIR     STORE ACCUMULATOR A IN MEMORY
1741 4147 B7FFFF             STA A    EXT
1750 414A A700               STA A    IND,X
1760 414C D7FF               STA B    DIR     STORE ACCUMULATOR B IN MEMORY
```

```
LINE LOC. OBJECT       SOURCE STATEMENT

1761 414E F7FFFF              STA B    EXT
1770 4151 E700                STA B    IND,X
1780 4153 9FFF                STS      DIR      STORE STACK REGISTER
1790 4155 BFFFFF              STS      EXT
1800 4158 AF00                STS      IND,X
1810 415A DFFF                STX      DIR      STORE INDEX REGISTER
1820 415C FFFFFF              STX      EXT
1830 415F EF00                STX      IND,X
1840 4161 80FF                SUB A    #IMM     SUB FROM ACCUMULATOR A
1850 4163 90FF                SUB A    DIR
1860 4165 B0FFFF              SUB A    EXT
1870 4168 A000                SUB A    IND,X
1880 416A C0FF                SUB B    #IMM     SUB FROM ACCUMULATOR B
1890 416C D0FF                SUB B    DIR
1900 416E F0FFFF              SUB B    EXT
1910 4171 E000                SUB B    IND,X
1920 4173 3F                  SWI               SOFTWARE INTERRUPT
1930 4174 16                  TAB               TRANSFER ACCUM. A TO B
1940 4175 06                  TAP               TRANSFER ACCUM. A TO CC REGISTER
1950 4176 17                  TBA               TRANSFER ACCUMULATOR B TO A
1960 4177 07                  TPA               TRANSFER CC REGISTER TO ACCUM. A
1970 4178 7DFFFF              TST      EXT      TEST MEMORY
1980 417B 6D00                TST      IND,X
1990 417D 4D                  TST A             TEST ACCUMULATOR A
2000 417E 5D                  TST B             TEST ACCUMULATOR B
2010 417F 30                  TSX               TRANSFER STACK POINTER TO
                                                INDEX REGISTER
2020 4180 35                  TXS               TRANSFER INDEX REG TO
                                                STACK POINTER
2030 4181 3E                  WAI               WAIT FOR INTERRUPT
2040                          END
```

# Differences between HP and the 6800

The following is a list of differences, known to us at the time of printing, that exist between the Hewlett-Packard software and other 6800 Assembler descriptions.

a) The character set has been expanded.

b) The following pseudo mnemonics have been added:

FAIL<br>
SET<br>
IF<br>
ELSE<br>
ENDIF<br>
IFC<br>
IFNC

c) A complete set of Macros have been added.

d) NAM directive can appear anywhere in the program.

e) COUNT in the FCC directive is limited to 80 characters.

f) The following Assembler controls have been added:

MEX<br>
NOMEX<br>
SAVE<br>
RESTORE<br>
SOURCE

The following 6800 controls are not implemented:

| | |
|---|---|
| ERROR | |
| SERROR | |
| NOERROR | |
| SLIST | |
| MEMORY | |
| NOMEMORY | |
| OTAPE | Replaced by OBJECT "file name" |
| NOOTAPE | Replaced by OBJECT |
| PAGE | |
| NOPAGE | |
| GENERATE | Replaced by GEN |
| NOGENERATE | Replaced by NOGEN |
| TAB | |
| NOTAB | |

# Section Seven
# Z80 Assembly Language Instructions

## Introduction

The following is a description of the Assembly Language instructions needed to write Source code for the Z80 Microprocessor. We recommend that you make yourself acquainted with this explanation as it differs in some respects from the description given by Z80 Programming Manual.

## Source Line Format

A Program written in Assembler Language is called a Source code. It consists of symbolic commands called statements. Each statement is written on a single line and may consist of from one to four fields. From left to right the fields are:

(1) LABEL: (2) INSTRUCTION (3) OPERAND (4) COMMENT

The following pages describe each field individually. An illustration of the format pattern is given below.

```
[ <LABEL>:] [ <INSTRUCTION> ] [ <OPER> [,<OPER> . . . ] ] [;<COMMENT> ]

Space LABEL: space INSTRUCTION space OPERAND, OPERAND space ; COMMENT
  |            |                  |                         |
Optional     Optional          One or more               Optional
```

## Label Field

A Label is required for statements involved in the definition of Symbols and occasionally at destinations of branch and jump instructions.

A Label is a symbol name whose value is the location where the instruction is assembled. Any number of characters (limited by the line length) may be used. However, the Assembler only recognizes up to eight alphanumeric characters whereby the first character must be alphabetic.

All Labels must end with a colon, except if it starts in column one.

Label may consist of any permutation of upper and lower case. However, two names which differ in case construction will be considered as two different names. Thus, LABEL, label and LaBel will be considered as three different names.

The Assembler maintains a location counter to provide addresses for the symbols in the Label field. When a symbol is found in the Label field the Assembler places the symbol and the corresponding location counter value in a symbol table.

## Instruction Field

Instructions are machine or pseudo mnemonics and must be present in all statements except when the statement consists only of a Comment or Label. Instructions must not start in the first character position. Pseudo mnemonics are described later in this Section, and a list of Z80 machine mnemonics is given at the back of this Section.

## Operand Field

The Operand field identifies the data to be acted upon by the specified instruction. Some instructions require no Operand, while others require one or more.

The Operand field is separated from the Instruction field by at least one blank. If more than one Operand is required they are separated from each other by a comma.

The type of information placed in the Operand field depends upon the particular Instruction. The following are recognized by the Assembler as Operands:

Numbers
Symbols
Expressions
ASCII strings
Location counter symbol ($)

The listed Operand types are now described in more detail.

## Numbers

The Assembler accepts numbers in the Operand field in several different bases: binary, octal, decimal, and hexadecimal. Numbers are accepted in the following formats:

| | | |
|---|---|---|
| <Number> | Specifies a Decimal number | 0–65536 |
| <Number>D | Specifies a Decimal number | 0D–65536D |
| <Number>H | Specifies a Hexadecimal number | 0H–0FFFFH |
| <Number>O | Specifies an Octal number | 0O–177777O |
| <Number>Q | Specifies an Octal number | 0Q–177777Q |
| <Number>B | Specifies a Binary number | 0B–1111111111111111B |

The <Number> must always begin with a numeric character. For example:

| | |
|---|---|
| A9FFH | Illegal (will be recognized as a Symbol) |
| 0A9FFH | Correct specification. |

## Symbols

Symbols can be divided into three separate identities, Reserved, User definable, and Assembler generated symbols.

Reserved symbols are those that already have a special meaning to the Assembler and therefore cannot appear as user-defined symbols. The mnemonic names for microprocessor instructions and the Assembler directives are all reserved symbols.

The following instruction Operand symbols are also reserved:

| | |
|---|---|
| A | Register A |
| B | Register B |
| C | Register C |
| D | Register D |
| E | Register E |
| H | Register H |
| L | Register L |
| BC | Register Pair B and C |
| DE | Register Pair D and E |
| HL | Register Pair H and L |
| SP | Stack Pointer |
| IX | Index Register |
| IY | Index Register |
| AF | Register Pair A and F |
| AF. | Register Pair A' and F' (Zilog Assembler AF') |
| $ | Location Counter |

User-defined symbols are symbols you create to reference instruction and data addresses. These symbols are defined when they appear in the Label field of the instruction, the EQU, DEFL or MACRO.

## Expressions

An expression is an Operand entry consisting of either a single term or a combination of terms. It contains a valid series of Symbols and Numbers that can be connected by operators.

The Assembler includes five groups of operators which permit the following Assembly-time operations:

Arithmetic
Shift
Logical
Relational
Result

It is important to remember that these are assembly-time operations. Once the Assembler has evaluated an expression, it becomes a permanent part of your program.

Expressions are evaluated from left to right. Operators with higher precedence are evaluated before other operators that immediately preceed or follow them. When two operators have equal precedence, the left most is evaluated first.

The following list describes classes of operators in order of precedence.

| | |
|---|---|
| Parenthesized expressions | High |
| Unary Minus .NOT. \ .RES. | |
| * / .MOD. .SHR. .SHL. | |
| + – | |
| .AND. & | |
| .OR. .XOR. | ↓ |
| Relational Operators | Low |

The Assembler must complete the numeric evaluation of symbols and expressions in two passes. Only one level of forward addressing is permitted in the use of symbols or expressions in the Operand field of the Source statement.

For instance, in the example below P would not be evaluated.

```
P        EQU     R
R        EQU     Q
Q        EQU     5
```

## ARITHMETIC OPERATORS

| Operator | Meaning |
|---|---|
| + | Addition |
| – | Subtraction |
| * | Multiplication |
| / | Integer Division. Any remainder is discarded (7/2=3). |
| .MOD. | Modulo, Result is the remainder caused by a division operation. (7 .MOD. 3 = 1) |

Examples:

The following expressions generate the bit pattern for the ASCII character A, where A equals 65 decimal.

```
5+30*2
(25/5) +30*2
5+(–30*–2)
```

Example of .MOD.

The .MOD. operator must be separated from its Operands by spaces:

```
NUMBER    .MOD.   8
```

Assuming that NUMBER has the value 25, then the expression evaluates to 1.

## SHIFT OPERATORS.

| Operator | Meaning |
|---|---|
| y .SHR. x | Shift 'y' to the right 'x' bit positions. |
| y .SHL. x | Shift 'y' to the left 'x' bit positions. |

The shift operators do not wraparound any bits shifted out of the byte. Bits vacated by the shift operation are zero filled. Note that the shift operator must be separated from its Operands by spaces.

Example:

Assume that NUMBER has the value 0101 0101B. The effect of the shift operators is as follows:

```
NUMBER    .SHR.   2     0001 0101B
NUMBER    .SHL.   1     1010 1010B
```

Note that a shift one bit position to the left has the effect of multiplying a value by two; a shift one bit position to the right has the effect of dividing a value by two.

LOGICAL OPERATORS.

| Operator | | Meaning |
|---|---|---|
| .NOT. | \ | Logical One's Complement |
| .AND. | & | Logical AND |
| .OR. | | Logical OR |
| .XOR. | | Logical Exclusive OR |

Example:

If VALUE1 = 12
VALUE2 = 15
VALUE3 = 20

Then VALUE1 .AND. VALUE2 .AND. VALUE3 = 4

RELATIONAL OPERATORS.

| Operator | | Meaning |
|---|---|---|
| .EQ. | | Equal |
| .LT. | < | Less Than |
| .GT. | > | Greater Than |
| .UGT. | | Unsigned Greater Than |
| .ULT. | | Unsigned Less Than |

The relational operators produce a yes-no result. Thus, if the evaluation of the relation is TRUE, the value of the result is all ones. If FALSE, the value of the result is all zeros.

Relational operators .UGT. and .ULT. compare unsigned binary values. Hence, a –3 will be evaluated as greater than 10.

Example:

The following COND directive tests the values of VALUE1 and VALUE2 for equality. If the result of the comparison is TRUE, the assembly language coding following the COND directive is assembled. Note that the relational operator must be separated from its Operand by spaces.

```
COND VALUE1    .EQ.    VALUE2
```

RESULT OPERATOR

Assume, for example, that your Source code contains a statement that will put an intermediate calculation outside of the limits 32767 to –32767 that the program can work with. Assuming also that the end result will be in the range, then by putting the .RES. operator before the expression gives the correct answer to your problem. Without the .RES. operator, an error is given.

### ASCII Strings

All ASCII characters enclosed in single quotes define an ASCII String. Two successive quotes must be used to represent one single quote.

Double quote marks can also be used to define an ASCII String. In this case, the double quotes define the string delimiter while a single quote is regarded as a character within the string. For example:

'TODAY''S DATE' or "TODAY'S DATE"

### Special Operands

An expression or Register Pair completely enclosed in parenthesis indicates a memory address.

|  |  |  |
|---|---|---|
| For example: | 12+3 | Constant with value 15 |
|  | (12+3) | Memory address 15 |

## Comment Field

Comments can be optionally used at the end of a Source line. If present, they are ignored by the Assembler during translation and appear only in the program listing.

Separation of a Comment and Operand field is made by a semicolon.

# Assembler Directives

The Assembler directives, sometimes called pseudo mnemonics, are instructions entered into the Source code which direct the Assembler when it generates the Object code. They control the generation of the Assembler program and define the format of the Assembler output listing.

**COND-ELSE-ENDC**

```
[ <LABEL> :]    COND    <EXPRESSION>        [;<COMMENT> ]
                .
                .       CODE SEQUENCE 1
                .
[ <LABEL> : ]   ELSE                        [ ; <COMMENT> ]
                .
                .       CODE SEQUENCE 2
                .
[ <LABEL> :]    ENDC                        [ ; <COMMENT> ]
```

A code sequence between COND-ENDC directive will be assembled when the expression evaluates to a value other than zero.

When the ELSE directive is included, the code sequence between the COND and the ELSE directive is assembled when the expression after COND evaluates to a value other than zero. When the expression equals zero, the code sequence between the ELSE and the ENDC directives is assembled.

Example

```
   260                         COND  VALUE-3270     ;CONDITIONAL ASSEMBLY START
   270
   280                 START:  ADD      A,VALUE+2 ;THIS LINE IS NOT ASSEMBLED
   290
   300                         ELSE
   310
   320 1223 C647       START:  ADD      A,'G'     ;THIS LINE IS ASSEMBLED
   330
**'ASSEMBLING PART 2'**
   340                         FAIL     'ASSEMBLING PART 2'
   350
   360                         ENDC                 ;CONDITIONAL ASSEMBLY END
```

**DEFB** – Define byte
**DEFM** – Define Memory

[ <LABEL>:] **DEFB** <EXPR>|<STRING> [,<EXPR>|<STRING> . . . ] [;<COMMENT> ]

[ <LABEL>:] **DEFM** <EXPR>|<STRING> [,<EXPR>|<STRING> . . . ] [;<COMMENT> ]

An 8-bit value corresponding to each Operand is stored in a byte of the Object code. This directive may have one or more Operands, separated by commas. The number of Operands is limited by the Source line length of 80 characters. The Operand can be any constant, symbol, an expression evaluating to an 8-bit value, or an ASCII String.

The Label is optional. However, when it is present the address of the first generated byte is assigned to the symbol represented by the Label.

Example

```
210 1216 24                DEFB      00100100B ;BINARY VALUE
220 1217 54455854          DEFM      'TEXT'    ;ASCII "TEXT" INTO MEMORY
```

**DEFL** – Define label

<LABEL> : **DEFL** <EXPRESSION> [;<COMMENT> ]

This directive assigns the value given by the Operand field to the Symbol specified in the Label field. This value remains unchanged until altered by a subsequent DEFL directive.

Example

```
150      00A0        LABEL     DEFL      00A0H     ;HEXADECIMAL ADDRESS
```

**DEFS** – Define storage

[ <LABEL> : ] **DEFS** <EXPRESSION> [ ;<COMMENT> ]

The DEFS directive causes the location counter to be increased by the value given in the Operand field. This reserves a block of memory equal to the size of the expression value. The Operand field may contain the actual number equal to the number of bytes to be reserved or the Operand may be a symbol or an expression which can be assigned a numerical value by the Assembler. When a symbol is used it must have been previously defined. A symbol that has been defined by a Label is assigned the address of the first byte reserved by this directive. The content of the memory block is unchanged. The Assembler only advances the location counter and does not generate any Object code.

Example

```
240 121B             TEMP:     DEFS  8             ;CREATE AREA 8 BYTES LONG
```

**DEFW** – Define word

[ <LABEL> : ]  **DEFW**  <EXPR> [ , <EXPR> . . . ] [ ; <COMMENT> ]

The DEFW directive may have one or more Operands separated by commas. It is similar to the Define Byte directive except that it generates a double byte Object from a 16-bit value. The least significant bits are stored in the first byte. Strings of one or two characters are allowed however longer Strings cause errors.

Example

```
200 1214 0C43         CONSTANT:DEFW       041414Q    ;OCTAL VALUE
```

**END**

[ <LABEL> : ]  **END**  [<START ADDRESS>]  [; <COMMENT> ]

This directive stops the Assembler program. If the END directive is missing, the assembler program is terminated by the end of the file. However, the Assembler does not stop if an END directive is a part of a file called by the INCLUDE or SOURCE control from a Source code. The directive is not translated into Object code. The start address should be present when it is required by your Microprocessor system. Refer to the Console Section of this manual for further information.

Example

```
480      0000                 END                     ;STOPS THE ASSEMBLER
```

**EQU**

<LABEL> :  **EQU**  <EXPRESSION>  [ ; <COMMENT> ]

This directive assigns a value, given by the Operand field, to the symbol specified in the Label field. However, this directive cannot redefine the symbol by a subsequent EQU directive.

Example

```
160      00D7         VALUE    EQU      327O      ;OCTAL VALUE
```

**FAIL**

[ <LABEL> : ]  **FAIL**  'USER DEFINABLE ERROR MESSAGE'

The FAIL directive forces the Assembler to printout a user defined error message. This directive would be contained in your Source code as an indication that unwanted code has been assembled. It is often used within the COND-ELSE-ENDC directives to ensure that the correct code has been assembled.

Example

```
**'ASSEMBLING PART 2'**
   340                          FAIL      'ASSEMBLING PART 2'
```

**IFC-ELSE-ENDC**

```
[ <LABEL> : ]     IFC      <OPER1> ,<OPER2>     [ ;<COMMENT> ]
                  .
                  .        CODE SEQUENCE
                  .
[ <LABEL> : ]     ENDC                          [ ;<COMMENT> ]
```

The IFC directive is similar to the COND directive. However, the code sequence is Assembled if the two Operands, following the IFC directive, are equal on a character for character basis. No evaluation is performed on the two Operands. Operands can be Symbols or Strings. You use the ELSE pseudo in the same manner as with the COND directive.

**IFNC-ELSE-ENDC**

```
[ <LABEL> : ]     IFNC     <OPER1> ,<OPER2>     [ ;<COMMENT> ]
                  .
                  .        CODE SEQUENCE
                  .
[ <LABEL> : ]     ENDC                          [ ;<COMMENT> ]
```

The IFNC directive is similar to the IFC directive, whereby the code sequence is Assembled if the two Operands, following the IFNC directive, are not equal on a character for character basis.

**ORG** – Origin

```
[ <LABEL> : ]     ORG      <EXPRESSION>         [ ;<COMMENT> ]
```

This directive defines the numerical address of the first byte of machine code which results from the assembly of the immediately subsequent section of a Source code. There may be any number of ORG statements in a program. The ORG directive sets the location counter to the value expressed in the Operand field. This field may contain the actual value (decimal, hexadecimal, octal, or binary) to which the location counter is to be set. Alternatively, the Operand field may contain a symbol or an expression which can be assigned a numeric value by the Assembler. Symbols must be previously defined.

The location counter is initialized to 0000H before each assembly. If no ORG statement appears at the beginning of the program, the location counter begins as if an ORG zero had been entered.

Example

```
30 0000                     ORG       4623      ;DECIMAL ADDRESS
```

**SPC** – Space

```
[ <LABEL> : ]     SPC      <EXPRESSION>         [ ;<COMMENT> ]
```

The SPC directive inserts a number of blank lines in the program listing. The number of lines to be left blank is stated in the Operand field. When the SPC directive causes the listing to cross page boundaries only those blank lines required to get to the top of the next page are generated.

# Assembler Controls

All Assembler Controls are preceeded by the asterisk (✱) sign which must be placed at the first character position in the Source code. As many Controls as required may be included on one line and each Control must be separated by a blank or a comma. If more Controls are required that can fit on one line, other lines may be used as long as each line begins with the asterisk sign.

These Controls can also be entered after the SOURCE or OBJECT statement in the Assembler program. See Section IX.

✱[ <CONTROL1> [ , <CONTROL2> . . . ] ] [ ; COMMENT]

**HEADING** 'ASCII String' Defines the heading for the top of the page. If entered in the first line of the Source code it appears at the top of the first page. Entered into the Source code at any other time causes it to appear on the next page following its definition.

**EJECT** Advances the paper to the top of the next page.

**SOURCE** "<FILE NAME> [msus]" A file on the mass storage device is entered in the assembly if specified in the Operand field with this control. If a Source line contains two or more SOURCE controls, then the last Source file specified is assembled first. Eight levels of nesting are allowed. Enter the msus if the Source file is not stored on the mass storage device specified in the default file.

**INCLUDE** "<FILE NAME> [msus]" Same as for SOURCE.

**OBJECT** ["<FILE NAME> [msus]"] As long as the file on which the Object code is to be stored was not specified with the OBJECT statement of the Assembler program, see Assembler Module Section IX, it may be specified by inserting this Control in the Source code. The position of this Control in the Source code is not important, as the Object code for the entire Source code will still be stored.

Entering the OBJECT Control without a file name results in no Object code being stored. If the Object file is specified, no other Object control is accepted.

**SYMBOLS ON** Prints the Symbol Table after the program listing. The symbol table listing is generated by the cross reference generator. Only the cross reference list is generated if both the Symbol and cross reference control commands are selected.

***SYMBOLS OFF** Suppresses the Symbol Table print.

***XREF ON** When the Assembler is finished the XREF generator is called and prints the cross reference of all user symbols. If XREF ON SYMBOLS ON have been specified, then only an XREF is made.

**XREF OFF** Suppresses the cross reference.

**SAVE** The current list control settings LIST, CONDLIST and MACLIST are saved but remain valid until explicitly changed. Nesting up to eight levels is possible.

**RESTORE** Recalls the list Control settings. If more RESTORE commands are given than there are SAVE commands, then the extra RESTORE commands are ignored.

***LIST ON** Instructs the Assembler to generate a program listing.

**LIST OFF** Suppresses the printing of the program listing.

***CONDLIST ON** Includes conditional skipped Source lines in the listing.

**CONDLIST OFF** Does not include conditionally skipped lines.

***MACLIST ON** Lists MACRO expansion.

**MACLIST OFF** Does not list MACRO expansion.

*The Assembler uses these Controls unless otherwise specified.

Examples

To give your printed pages a title, use the HEADING Control followed by the title enclosed in quotes.

```
10                    * HEADING 'ASSEMBLER DIRECTIVE DEMO PROGRAM'
```

To save list controls, and suppress the Macro listing, type the following controls.

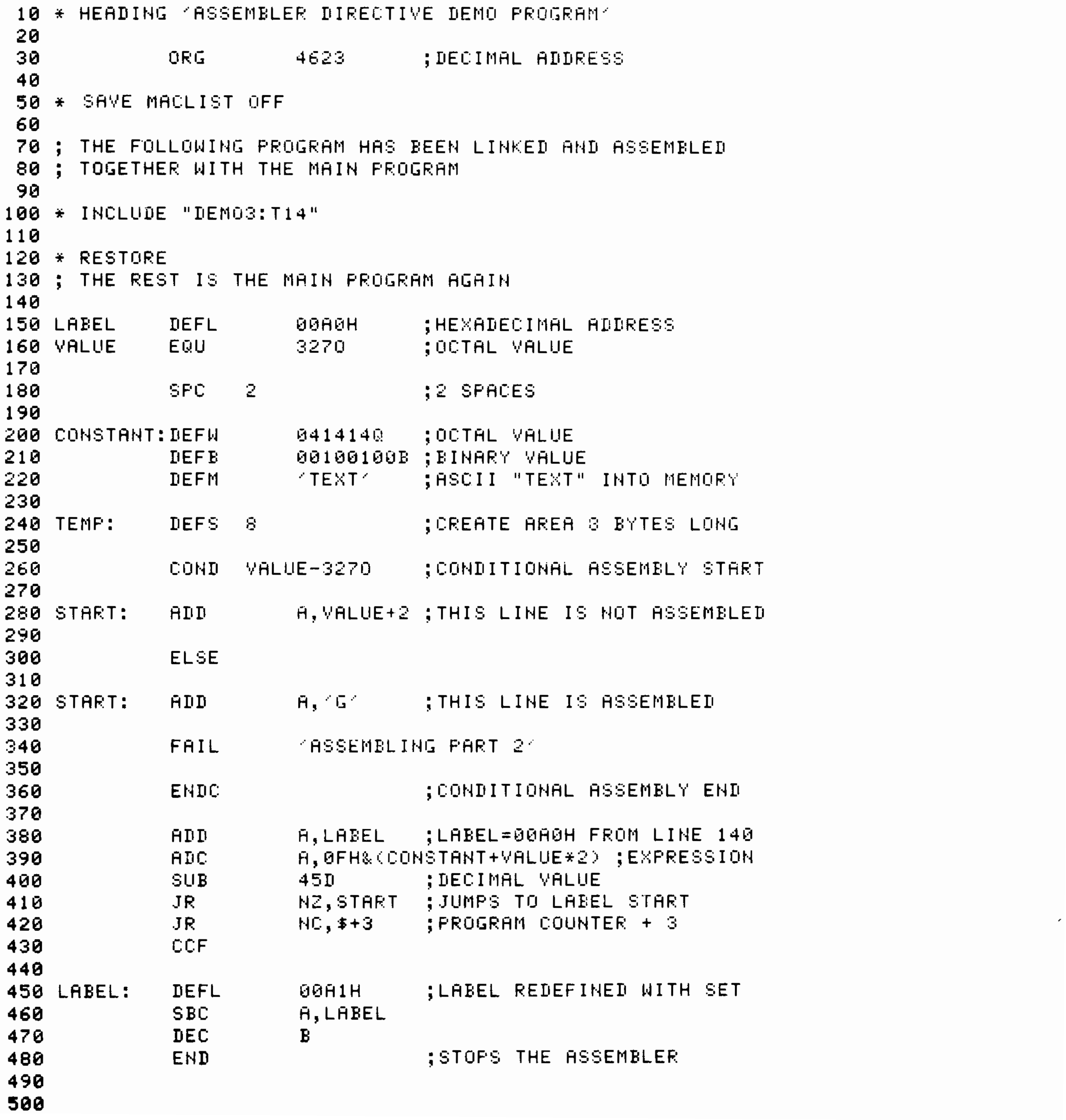
```
50                    * SAVE MACLIST OFF
```

You can use the INCLUDE Control to assemble several Source files within the same assembly. It is possible to terminate the Source file with an END directive. Once the included file has been assembled, the Assembler continues with the file that contained the INCLUDE Control. Eight levels of nesting are possible.

```
100                    * INCLUDE "DEMO3:T14"
 10                    * HEADING 'PROGRAM BEING LINKED'
 20 120F CB6F                   BIT      5,A
 30 1211 3D            LOOP:    DEC      A
 40 1212 20FD                   JR       NZ,LOOP
 50      0000                   END
110
120                    * RESTORE
130                    ; THE REST IS THE MAIN PROGRAM AGAIN
```

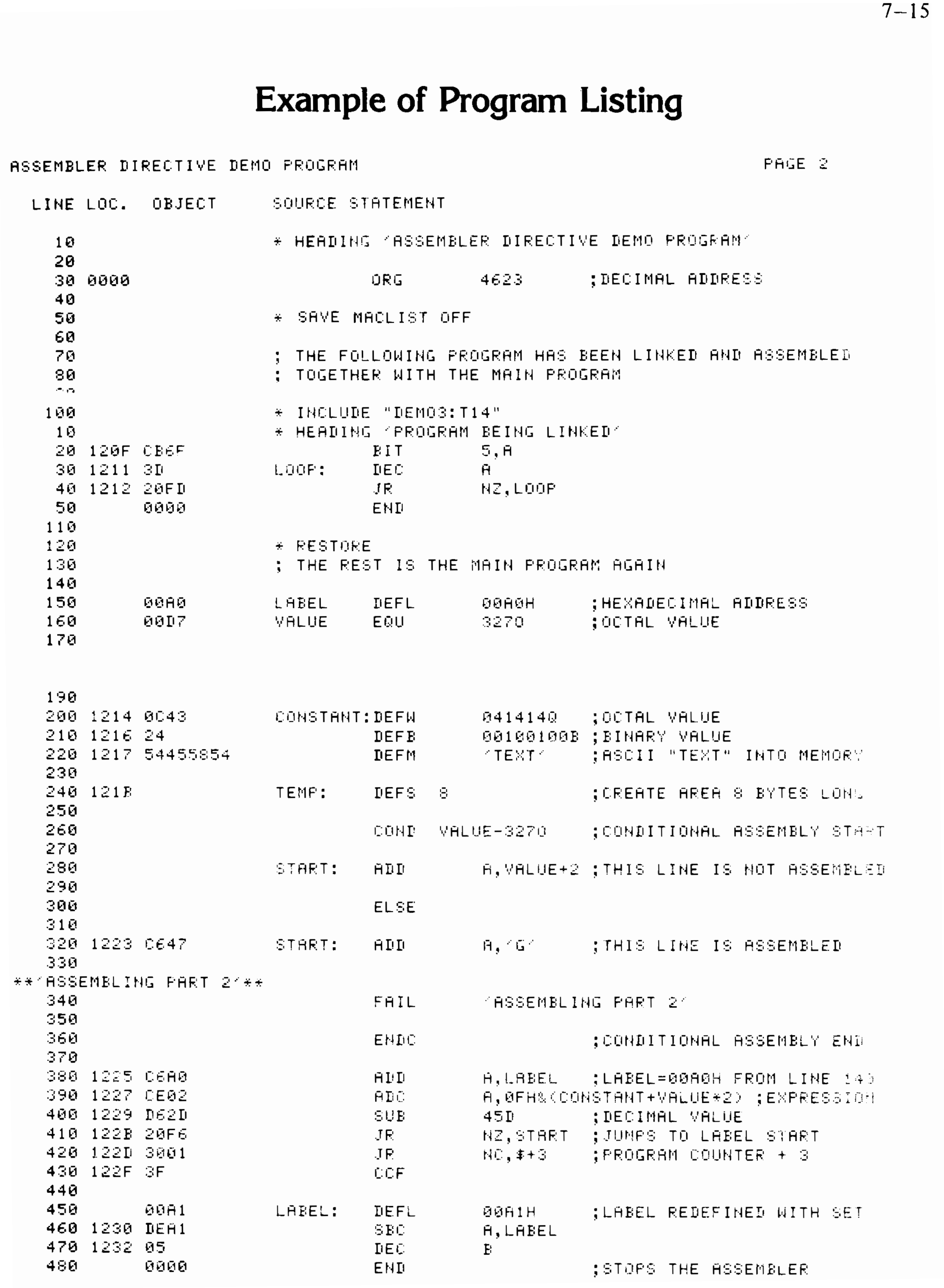
Note that on the original Source code the ✱ sign would be placed in the first character position.

# Example of Source Code



```
 10 * HEADING 'ASSEMBLER DIRECTIVE DEMO PROGRAM'
 20
 30          ORG       4623      ;DECIMAL ADDRESS
 40
 50 *  SAVE MACLIST OFF
 60
 70 ; THE FOLLOWING PROGRAM HAS BEEN LINKED AND ASSEMBLED
 80 ; TOGETHER WITH THE MAIN PROGRAM
 90
100 * INCLUDE "DEMO3:T14"
110
120 * RESTORE
130 ; THE REST IS THE MAIN PROGRAM AGAIN
140
150 LABEL    DEFL      00A0H     ;HEXADECIMAL ADDRESS
160 VALUE    EQU       3270      ;OCTAL VALUE
170
180          SPC   2             ;2 SPACES
190
200 CONSTANT:DEFW      041414Q   ;OCTAL VALUE
210          DEFB      00100100B ;BINARY VALUE
220          DEFM      'TEXT'    ;ASCII "TEXT" INTO MEMORY
230
240 TEMP:    DEFS  8             ;CREATE AREA 8 BYTES LONG
250
260          COND  VALUE-3270    ;CONDITIONAL ASSEMBLY START
270
280 START:   ADD       A,VALUE+2 ;THIS LINE IS NOT ASSEMBLED
290
300          ELSE
310
320 START:   ADD       A,'G'     ;THIS LINE IS ASSEMBLED
330
340          FAIL      'ASSEMBLING PART 2'
350
360          ENDC                ;CONDITIONAL ASSEMBLY END
370
380          ADD       A,LABEL   ;LABEL=00A0H FROM LINE 140
390          ADC       A,0FH&(CONSTANT+VALUE*2) ;EXPRESSION
400          SUB       45D       ;DECIMAL VALUE
410          JR        NZ,START  ;JUMPS TO LABEL START
420          JR        NC,$+3    ;PROGRAM COUNTER + 3
430          CCF
440
450 LABEL:   DEFL      00A1H     ;LABEL REDEFINED WITH SET
460          SBC       A,LABEL
470          DEC       B
480          END                 ;STOPS THE ASSEMBLER
490
500
```

# Example of Program Listing



```
ASSEMBLER DIRECTIVE DEMO PROGRAM                                    PAGE 2

  LINE LOC.  OBJECT      SOURCE STATEMENT

    10                   * HEADING 'ASSEMBLER DIRECTIVE DEMO PROGRAM'
    20
    30 0000                        ORG        4623       ;DECIMAL ADDRESS
    40
    50                   * SAVE MACLIST OFF
    60
    70                   ; THE FOLLOWING PROGRAM HAS BEEN LINKED AND ASSEMBLED
    80                   ; TOGETHER WITH THE MAIN PROGRAM
    90
   100                   * INCLUDE "DEMO3:T14"
    10                   * HEADING 'PROGRAM BEING LINKED'
    20 120F CB6F                   BIT        5,A
    30 1211 3D           LOOP:     DEC        A
    40 1212 20FD                   JR         NZ,LOOP
    50      0000                   END
   110
   120                   * RESTORE
   130                   ; THE REST IS THE MAIN PROGRAM AGAIN
   140
   150      00A0         LABEL     DEFL       00A0H      ;HEXADECIMAL ADDRESS
   160      00D7         VALUE     EQU        327O       ;OCTAL VALUE
   170


   190
   200 1214 0C43         CONSTANT:DEFW        041414Q    ;OCTAL VALUE
   210 1216 24                     DEFB       00100100B ;BINARY VALUE
   220 1217 54455854               DEFM       'TEXT'     ;ASCII "TEXT" INTO MEMORY
   230
   240 121B              TEMP:     DEFS  8               ;CREATE AREA 8 BYTES LONG
   250
   260                             COND  VALUE-327O      ;CONDITIONAL ASSEMBLY START
   270
   280                   START:    ADD        A,VALUE+2 ;THIS LINE IS NOT ASSEMBLED
   290
   300                             ELSE
   310
   320 1223 C647         START:    ADD        A,'G'      ;THIS LINE IS ASSEMBLED
   330
**'ASSEMBLING PART 2'**
   340                             FAIL       'ASSEMBLING PART 2'
   350
   360                             ENDC                  ;CONDITIONAL ASSEMBLY END
   370
   380 1225 C6A0                   ADD        A,LABEL    ;LABEL=00A0H FROM LINE 140
   390 1227 CE02                   ADC        A,0FH&(CONSTANT+VALUE*2) ;EXPRESSION
   400 1229 D62D                   SUB        45D        ;DECIMAL VALUE
   410 122B 20F6                   JR         NZ,START   ;JUMPS TO LABEL START
   420 122D 3001                   JR         NC,$+3     ;PROGRAM COUNTER + 3
   430 122F 3F                     CCF
   440
   450      00A1         LABEL:    DEFL       00A1H      ;LABEL REDEFINED WITH SET
   460 1230 DEA1                   SBC        A,LABEL
   470 1232 05                     DEC        B
   480      0000                   END                   ;STOPS THE ASSEMBLER
```

## Cross Reference List

```
SYMBOL      VALUE   DEFINED   REFERENCED

CONSTANT    1214      200       390
LABEL       00A1      150     RE-DEFINED:    450
                                380    460
LOOP        1211       30        40
START       1223      320       410
TEMP        121D      240     * UNREFERENCED *
VALUE       00D7      160       260    390
```

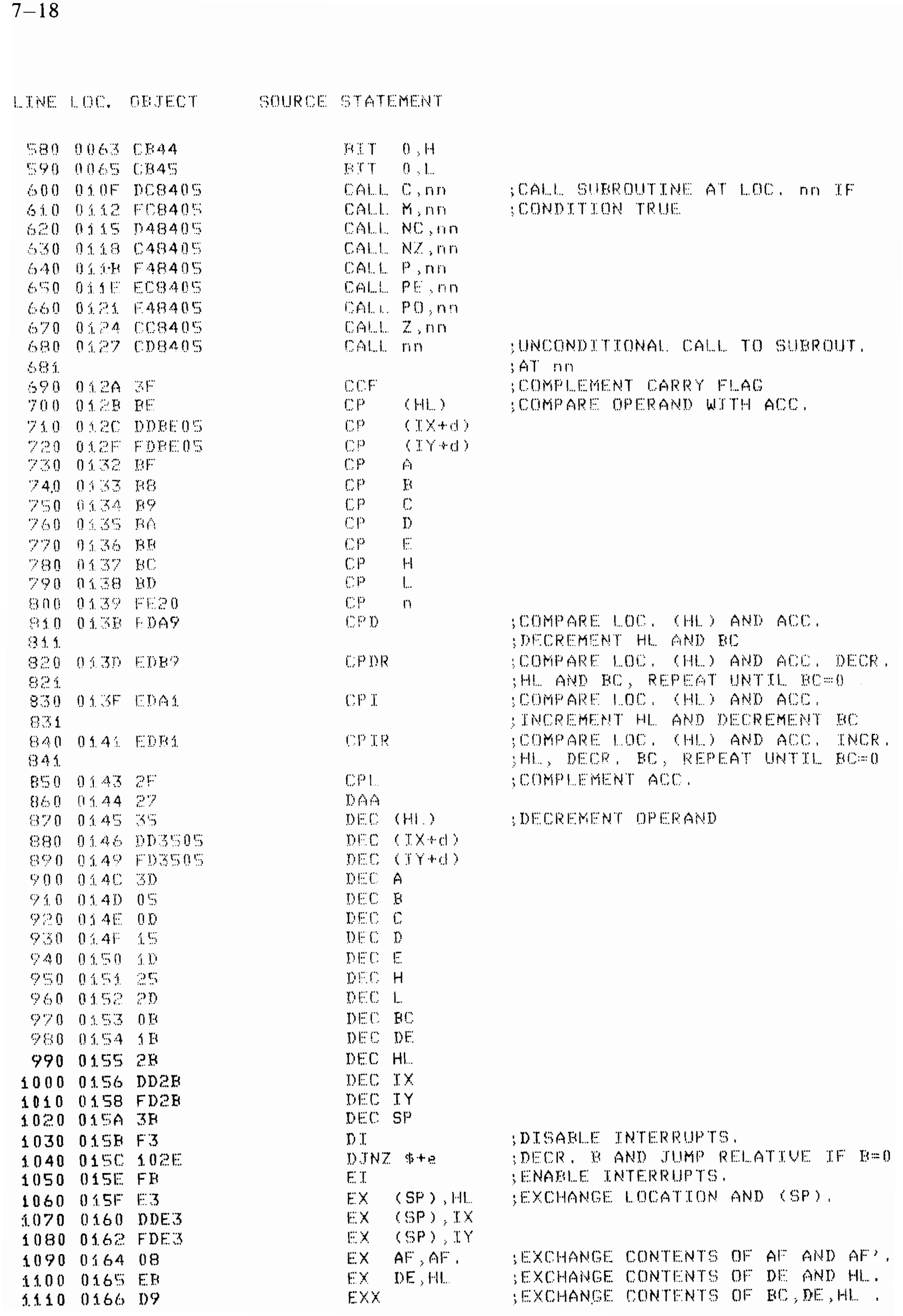
## Character Set

The Assembler recognizes the complete ASCII character set and generally all characters are considered to be alphabetic except the following who each have a special meaning.

| | | | |
|---|---|---|---|
| + | Plus sign | ADDITION | |
| – | Minus sign | SUBTRACTION | |
| * | Asterisk | MULTIPLICATION | |
| / | Slash | DIVISION | |
| ( | Left Parenthesis | LEFT PARENTHESIS | |
| ) | Right Parenthesis | RIGHT PARENTHESIS | |
| $ | Dollar sign | LOCATION COUNTER SYMBOL | |
| & | Ampersign | LOGICAL AND OPERATOR | |
| \ | Backslash | LOGICAL NOT OPERATOR | |
| > | Greater than sign | GREATER THAN OPERATOR | |
| < | Less than sign | LESS THAN OPERATOR | |
| ' | Apostroph | STRING DELIMITER | |
| " | Quotes | STRING DELIMITER | |
| : | Colon | LABEL DELIMITER | |
| ; | Semicolon | COMMENT FIELD DELIMITER | |
| B | | BINARY NUMBER SUFFIX | B, D, H, O, Q, are also alpha characters |
| D | | DECIMAL NUMBER SUFFIX | |
| H | | HEX NUMBER SUFFIX | |
| O | | OCTAL NUMBER SUFFIX | |
| Q | | OCTAL NUMBER SUFFIX | |
| SPACE | | FIELD SEPARATOR OR SYMBOL TERMINATOR | |
| HOR TAB | | FIELD SEPARATOR OR SYMBOL TERMINATOR | |
| . | Comma | OPERAND SEPARATOR | |
| = | Equals sign | MACRO DUMMY ARGUMENT PREFIX | |
| ^ | | MACRO PRE-EVALUATION PREFIX | |

# Instructions

```
LINE LOC. OBJECT      SOURCE STATEMENT

  10 0003 8E                 ADC A,(HL)     ;ADD WITH CARRY OPERAND TO ACC.
  20 0004 DD8E05             ADC A,(IX+d)
  30 0007 FD8E05             ADC A,(IY+d)
  40 000A 8F                 ADC A,A
  50 000B 88                 ADC A,B
  60 000C 89                 ADC A,C
  70 000D 8A                 ADC A,D
  80 000E 8B                 ADC A,E
  90 000F 8C                 ADC A,H
 100 0010 8D                 ADC A,L
 110 0011 CE20               ADC A,n
 120 0013 ED4A               ADC HL,BC      ;ADD WITH CARRY REG PAIR TO HL
 130 0015 ED5A               ADC HL,DE
 140 0017 ED6A               ADC HL,HL
 150 0019 ED7A               ADC HL,SP
 160 001B 86                 ADD A,(HL)     ;ADD OPERAND TO ACC.
 170 001C DD8605             ADD A,(IX+d)
 180 001F FD8605             ADD A,(IY+d)
 190 0022 87                 ADD A,A
 200 0023 80                 ADD A,B
 210 0024 81                 ADD A,C
 220 0025 82                 ADD A,D
 230 0026 83                 ADD A,E
 240 0027 84                 ADD A,H
 250 0028 85                 ADD A,L
 260 0029 C620               ADD A,n
 270 002B 09                 ADD HL,BC      ;ADD REG. PAIR TO HL
 280 002C 19                 ADD HL,DE
 290 002D 29                 ADD HL,HL
 300 002E 39                 ADD HL,SP
 310 002F DD09               ADD IX,BC      ;ADD REG. PAIR TO IX
 320 0031 DD19               ADD IX,DE
 330 0033 DD29               ADD IX,IX
 340 0035 DD39               ADD IX,SP
 350 0037 FD09               ADD IY,BC      ;ADD REG. PAIR TO IY
 360 0039 FD19               ADD IY,DE
 370 003B FD29               ADD IY,IY
 380 003D FD39               ADD IY,SP
 390 003F A6                 AND (HL)       ;LOGICAL AND OF OPERAND AND ACC.
 400 0040 DDA605             AND (IX+d)
 410 0043 FDA605             AND (IY+d)
 420 0046 A7                 AND A
 430 0047 A0                 AND B
 440 0048 A1                 AND C
 450 0049 A2                 AND D
 460 004A A3                 AND F
 470 004B A4                 AND H
 480 004C A5                 AND L
 490 004D E620               AND n
 500 004F CB46               BIT  0,(HL)    ;TEST BIT B OF LOCATION OR REG.
 510 0051 DDCB0546           BIT  0,(IX+d)
 520 0055 FDCB0546           BIT  0,(IY+d)
 530 0059 CB47               BIT  0,A
 540 005B CB40               BIT  0,B
 550 005D CB41               BIT  0,C
 560 005F CB42               BIT  0,D
 570 0061 CB43               BIT  0,E
```



```
LINE LOC. OBJECT      SOURCE STATEMENT

 580 0063 CB44                 BIT   0,H
 590 0065 CB45                 BIT   0,L
 600 010F DC8405               CALL C,nn      ;CALL SUBROUTINE AT LOC. nn IF
 610 0112 FC8405               CALL M,nn      ;CONDITION TRUE
 620 0115 D48405               CALL NC,nn
 630 0118 C48405               CALL NZ,nn
 640 011B F48405               CALL P,nn
 650 011E EC8405               CALL PE,nn
 660 0121 E48405               CALL PO,nn
 670 0124 CC8405               CALL Z,nn
 680 0127 CD8405               CALL nn        ;UNCONDITIONAL CALL TO SUBROUT.
 681                                          ;AT nn
 690 012A 3F                   CCF            ;COMPLEMENT CARRY FLAG
 700 012B BE                   CP    (HL)     ;COMPARE OPERAND WITH ACC.
 710 012C DDBE05               CP    (IX+d)
 720 012F FDBE05               CP    (IY+d)
 730 0132 BF                   CP    A
 740 0133 B8                   CP    B
 750 0134 B9                   CP    C
 760 0135 BA                   CP    D
 770 0136 BB                   CP    E
 780 0137 BC                   CP    H
 790 0138 BD                   CP    L
 800 0139 FE20                 CP    n
 810 013B EDA9                 CPD            ;COMPARE LOC. (HL) AND ACC.
 811                                          ;DECREMENT HL AND BC
 820 013D EDB9                 CPDR           ;COMPARE LOC. (HL) AND ACC. DECR.
 821                                          ;HL AND BC, REPEAT UNTIL BC=0
 830 013F EDA1                 CPI            ;COMPARE LOC. (HL) AND ACC.
 831                                          ;INCREMENT HL AND DECREMENT BC
 840 0141 EDB1                 CPIR           ;COMPARE LOC. (HL) AND ACC. INCR.
 841                                          ;HL, DECR. BC, REPEAT UNTIL BC=0
 850 0143 2F                   CPL            ;COMPLEMENT ACC.
 860 0144 27                   DAA
 870 0145 35                   DEC (HL)       ;DECREMENT OPERAND
 880 0146 DD3505               DEC (IX+d)
 890 0149 FD3505               DEC (IY+d)
 900 014C 3D                   DEC A
 910 014D 05                   DEC B
 920 014E 0D                   DEC C
 930 014F 15                   DEC D
 940 0150 1D                   DEC E
 950 0151 25                   DEC H
 960 0152 2D                   DEC L
 970 0153 0B                   DEC BC
 980 0154 1B                   DEC DE
 990 0155 2B                   DEC HL
1000 0156 DD2B                 DEC IX
1010 0158 FD2B                 DEC IY
1020 015A 3B                   DEC SP
1030 015B F3                   DI             ;DISABLE INTERRUPTS.
1040 015C 102E                 DJNZ $+e       ;DECR. B AND JUMP RELATIVE IF B=0
1050 015E FB                   EI             ;ENABLE INTERRUPTS.
1060 015F E3                   EX   (SP),HL   ;EXCHANGE LOCATION AND (SP).
1070 0160 DDE3                 EX   (SP),IX
1080 0162 FDE3                 EX   (SP),IY
1090 0164 08                   EX   AF,AF.    ;EXCHANGE CONTENTS OF AF AND AF'.
1100 0165 EB                   EX   DE,HL     ;EXCHANGE CONTENTS OF DE AND HL.
1110 0166 D9                   EXX            ;EXCHANGE CONTENTS OF BC,DE,HL .
```

```
LINE LOC. OBJECT      SOURCE STATEMENT

1111                                  ;WITH CONTENTS OF BC', DE', HL'
1112                                  ;RESPECTIVELY.
1120 0167 76                HALT      ;WAIT FOR INTERRUPT OF RESET.
1130 0168 ED46              IM   0    ;SET INTERRUPT MODE.
1140 016A ED56              IM   1
1150 016C ED5E              IM   2
1160 016E ED78              IN   A,(C)    ;LOAD REG. WITH INPUT FROM DEVICE.
1170 0170 ED40              IN   B,(C)
1180 0172 ED48              IN   C,(C)
1190 0174 ED50              IN   D,(C)
1200 0176 ED58              IN   E,(C)
1210 0178 ED60              IN   H,(C)
1220 017A ED68              IN   L,(C)
1230 017C 23                INC HL        ;INCREMENT OPERAND
1240 017D 03                INC BC
1250 017E 13                INC DE
1260 017F 33                INC SP
1270 0180 DD23              INC IX
1280 0182 FD23              INC IY
1290 0184 DB20              IN   A,(n)    ;LOAD ACC. WITH INPUT FROM
1291                                      ;DEVICE n.
1300 0186 34                INC (HL)
1310 0187 DD3405            INC (IX+d)
1320 018A FD3405            INC (IY+d)
1330 018D 3C                INC A
1340 018E 04                INC B
1350 018F 0C                INC C
1360 0190 14                INC D
1370 0191 1C                INC E
1380 0192 24                INC H
1390 0193 2C                INC L
1400 0194 EDAA              IND           ;LOAD LOC. (HL) WITH INPUT FROM
1401                                      ;PORT (C). DECREMENT HL AND B.
1410 0196 EDBA              INDR          ;LOAD LOC. (HL) WITH INPUT FROM
1411                                      ;PORT (C). DECREMENT HL AND B.
1412                                      ;REPEAT UNTIL B=0.
1420 0198 EDA2              INI           ;LOAD LOC. (HL) WITH INPUT FROM
1421                                      ;PORT (C). INCR. HL AND DECR. B
1430 019A EDB2              INIR          ;LOAD LOC. (HL) WITH INPUT FROM
1431                                      ;PORT (C). INCR. HL, DECR. B,
1432                                      ;REPEAT UNTIL B=0.
1440 019C E9                JP   (HL)     ;UNCONDITIONAL JUMP TO LOCATION.
1450 019D DDE9              JP   (IX)
1460 019F C38405            JP   nn
1470 01A2 FDE9              JP   (IY)
1480 01A4 DA8405            JP   C,nn     ;JUMP TO LOC. IF CONDITION TRUE
1490 01A7 FA8405            JP   M,nn
1500 01AA D28405            JP   NC,nn
1510 01AD C28405            JP   NZ,nn
1520 01B0 F28405            JP   P,nn
1530 01B3 EA8405            JP   PE,nn
1540 01B6 E28405            JP   PO,nn
1550 01B9 CA8405            JP   Z,nn
1560 01BC 382E              JR   C,$+e    ;JUMP RELAT. TO PC+e IF COND. TRUE
1570 01BE 302E              JR   NC,$+e
1580 01C0 202E              JR   NZ,$+e
1590 01C2 282E              JR   Z,$+e
1600 01C4 182E              JR   $+e      ;UNCONDITIONAL JUMP RELAT. TO PC+e
1610 01C6 02                LD   (BC),A   ;LOAD SOURCE TO DESTINATION
```

```
LINE LOC. OBJECT      SOURCE STATEMENT

1620 01C7 12                 LD   (DE),A
1630 01C8 77                 LD   (HL),A
1640 01C9 70                 LD   (HL),B
1650 01CA 71                 LD   (HL),C
1660 01CB 72                 LD   (HL),D
1670 01CC 73                 LD   (HL),E
1680 01CD 74                 LD   (HL),H
1690 01CE 75                 LD   (HL),L
1700 01CF 3620               LD   (HL),n
1710 01D1 DD7705             LD   (IX+d),A
1720 01D4 DD7005             LD   (IX+d),B
1730 01D7 DD7105             LD   (IX+d),C
1740 01DA DD7205             LD   (IX+d),D
1750 01DD DD7305             LD   (IX+d),E
1760 01E0 DD7405             LD   (IX+d),H
1770 01E3 DD7505             LD   (IX+d),L
1780 01E6 DD360520           LD   (IX+d),n
1790 01EA FD7705             LD   (IY+d),A
1800 01ED FD7005             LD   (IY+d),B
1810 01F0 FD7105             LD   (IY+d),C
1820 01F3 FD7205             LD   (IY+d),D
1830 01F6 FD7305             LD   (IY+d),E
1840 01F9 FD7405             LD   (IY+d),H
1850 01FC FD7505             LD   (IY+d),L
1860 01FF FD360520           LD   (IY+d),n
1870 0203 328405             LD   (nn),A
1880 0206 ED438405           LD   (nn),BC
1890 020A ED538405           LD   (nn),DE
1900 020E 228405             LD   (nn),HL
1910 0211 DD228405           LD   (nn),IX
1920 0215 FD228405           LD   (nn),IY
1930 0219 ED738405           LD   (nn),SP
1940 021D 0A                 LD   A,(BC)
1950 021E 1A                 LD   A,(DE)
1960 021F 7E                 LD   A,(HL)
1970 0220 DD7E05             LD   A,(IX+d)
1980 0223 FD7E05             LD   A,(IY+d)
1990 0226 7F                 LD   A,A
2000 0227 78                 LD   A,B
2010 0228 79                 LD   A,C
2020 0229 7A                 LD   A,D
2030 022A 7B                 LD   A,E
2040 022B 7C                 LD   A,H
2050 022C 7D                 LD   A,L
2060 022D 3E20               LD   A,n
2070 022F 46                 LD   B,(HL)
2080 0230 DD4605             LD   B,(IX+d)
2090 0233 FD4605             LD   B,(IY+d)
2100 0236 47                 LD   B,A
2110 0237 40                 LD   B,B
2120 0238 41                 LD   B,C
2130 0239 42                 LD   B,D
2140 023A 43                 LD   B,E
2150 023B 44                 LD   B,H
2160 023C 45                 LD   B,L
2170 023D 0620               LD   B,n
2180 023F 4E                 LD   C,(HL)
2190 0240 DD4E05             LD   C,(IX+d)
2200 0243 FD4E05             LD   C,(IY+d)
```

```
LINE LOC.  OBJECT      SOURCE STATEMENT

2210 0246 4F                  LD   C,A
2220 0247 48                  LD   C,B
2230 0248 49                  LD   C,C
2240 0249 4A                  LD   C,D
2250 024A 4B                  LD   C,E
2260 024B 4C                  LD   C,H
2270 024C 4D                  LD   C,L
2280 024D 0E20                LD   C,n
2290 024F 56                  LD   D,(HL)
2300 0250 DD5605              LD   D,(IX+d)
2310 0253 FD5605              LD   D,(IY+d)
2320 0256 57                  LD   D,A
2330 0257 50                  LD   D,B
2340 0258 51                  LD   D,C
2350 0259 52                  LD   D,D
2360 025A 53                  LD   D,E
2370 025B 54                  LD   D,H
2380 025C 55                  LD   D,L
2390 025D 1620                LD   D,n
2400 025F 5E                  LD   E,(HL)
2410 0260 DD5E05              LD   E,(IX+d)
2420 0263 FD5E05              LD   E,(IY+d)
2430 0266 5F                  LD   E,A
2440 0267 58                  LD   E,B
2450 0268 59                  LD   E,C
2460 0269 5A                  LD   E,D
2470 026A 5B                  LD   E,E
2480 026B 5C                  LD   E,H
2490 026C 5D                  LD   E,L
2500 026D 1E20                LD   E,n
2510 026F 66                  LD   H,(HL)
2520 0270 DD6605              LD   H,(IX+d)
2530 0273 FD6605              LD   H,(IY+d)
2540 0276 67                  LD   H,A
2550 0277 60                  LD   H,B
2560 0278 61                  LD   H,C
2570 0279 62                  LD   H,D
2580 027A 63                  LD   H,E
2590 027B 64                  LD   H,H
2600 027C 65                  LD   H,L
2610 027D 2620                LD   H,n
2620 027F 6E                  LD   L,(HL)
2630 0280 DD6E05              LD   L,(IX+d)
2640 0283 FD6E05              LD   L,(IY+d)
2650 0286 6F                  LD   L,A
2660 0287 68                  LD   L,B
2670 0288 69                  LD   L,C
2680 0289 6A                  LD   L,D
2690 028A 6B                  LD   L,E
2700 028B 6C                  LD   L,H
2710 028C 6D                  LD   L,L
2720 028D 2E20                LD   L,n
2730 028F 3A8405              LD   A,(nn)
2740 0292 ED5F                LD   A,R
2750 0294 ED4B8405            LD   BC,(nn)
2760 0298 018405              LD   BC,nn
2770 029B ED5B8405            LD   DE,(nn)
2780 029F 118405              LD   DE,nn
2790 02A2 2A8405              LD   HL,(nn)
```

```
LINE LOC. OBJECT     SOURCE STATEMENT

2800 02A5 218405           LD  HL,nn
2810 02A8 ED47             LD  I,A
2820 02AA DD2A8405         LD  IX,(nn)
2830 02AE DD218405         LD  IX,nn
2840 02B2 FD2A8405         LD  IY,(nn)
2850 02B6 FD218405         LD  IY,nn
2860 02BA ED4F             LD  R,A
2870 02BC ED7B8405         LD  SP,(nn)
2880 02C0 F9               LD  SP,HL
2890 02C1 DDF9             LD  SP,IX
2900 02C3 FDF9             LD  SP,IY
2910 02C5 318405           LD  SP,nn
2920 02C8 EDA8             LDD           ;LOAD LOC. (DE) WITH LOC. (HL).
2921                                     ;DECREMENT DE,HL, AND BC.
2930 02CA EDB8             LDDR          ;LOAD LOC. (DE) WITH LOC. (HL),
2931                                     ;REPEAT UNTIL BC=0.
2940 02CC EDA0             LDI           ;LOAD LOC. (DE) WITH LOC. (HL).
2941                                     ;INCREMENT DE,HL, DECREMENT BC.
2950 02CE EDB0             LDIR          ;LOAD LOC. (DE) WITH LOC. (HL).
2951                                     ;INCR. DE,HL, DECR. BC UNTIL BC=0.
2960 02D0 ED44             NEG           ;NEGATE ACC. (2's COMPLEMENT).
2970 02D2 00               NOP           ;NO OPERATION.
2980 02D3 B6               OR  (HL)      ;LOGICAL OR OF OPERAND AND ACC.
2990 02D4 DDB605           OR  (IX+d)
3000 02D7 FDB605           OR  (IY+d)
3010 02DA B7               OR  A
3020 02DB B0               OR  B
3030 02DC B1               OR  C
3040 02DD B2               OR  D
3050 02DE B3               OR  E
3060 02DF B4               OR  H
3070 02E0 B5               OR  L
3080 02E1 F620             OR  n
3090 02E3 EDBB             OTDR          ;LOAD OUTPUT PORT (C) WITH LOC.
3091                                     ;(HL), DECR. HL AND B UNTIL B=0.
3100 02E5 EDB3             OTIR          ;LOAD OUTPUT PORT (C) WITH LOC. (HL
3101                                     ;INCR. HL, DECR B, UNTIL B=0.
3110 02E7 ED79             OUT (C),A     ;LOAD OUTPUT PORT(C) WITH REG.
3120 02E9 ED41             OUT (C),B
3130 02EB ED49             OUT (C),C
3140 02ED ED51             OUT (C),D
3150 02EF ED59             OUT (C),E
3160 02F1 ED61             OUT (C),H
3170 02F3 ED69             OUT (C),L
3180 02F5 D320             OUT (n),A     ;LOAD OUTPUT PORT (n) WITH ACC.
3190 02F7 EDAB             OUTD          ;LOAD OUTPUT PORT (C) WITH LOC. (HL
3191                                     ;DECREMENT HL AND B.
3200 02F9 EDA3             OUTI          ;LOAD OUTPUT PORT (C) WITH LOC. (HL
3201                                     ;INCREMENT HL AND DECREMENT B.
3210 02FB F1               POP AF        ;LOAD DESTINATION WITH TOP OF STACK
3220 02FC C1               POP BC
3230 02FD D1               POP DE
3240 02FE E1               POP HL
3250 02FF DDE1             POP IX
3260 0301 FDE1             POP IY
3270 0303 F5               PUSH AF       ;LOAD SOURCE TO STACK
3280 0304 C5               PUSH BC
3290 0305 D5               PUSH DE
3300 0306 E5               PUSH HL
```

```
LINE LOC. OBJECT      SOURCE STATEMENT

3310 0307 DDE5                PUSH IX
3320 0309 FDE5                PUSH IY
3330 030B CB86                RES  0,(HL)    ;RESET BIT b OF OPERAND
3340 030D DDCB0586            RES  0,(IX+d)
3350 0311 FDCB0586            RES  0,(IY+d)
3360 0315 CB87                RES  0,A
3370 0317 CB80                RES  0,B
3380 0319 CB81                RES  0,C
3390 031B CB82                RES  0,D
3400 031D CB83                RES  0,E
3410 031F CB84                RES  0,H
3420 0321 CB85                RES  0,L
3430 03CB C9                  RET            ;RETURN FROM SUBROUTINE
3440 03CC D8                  RET  C         ;RETURN FROM SUBROUTINE IF COND.
3441                                         ;COND. TRUE
3450 03CD F8                  RET  M
3460 03CE D0                  RET  NC
3470 03CF C0                  RET  NZ
3480 03D0 F0                  RET  P
3490 03D1 E8                  RET  PE
3500 03D2 E0                  RET  PO
3510 03D3 C8                  RET  Z
3520 03D4 ED4D                RETI           ;RETURN FROM INTERRUPT
3530 03D6 ED45                RETN           ;RETURN FROM NON MASKABLE INTERRUPT
3540 03D8 CB16                RL   (HL)      ;ROTATE LEFT THROUGH CARRY
3550 03DA DDCB0516            RL   (IX+d)
3560 03DE FDCB0516            RL   (IY+d)
3570 03E2 CB17                RL   A
3580 03E4 CB10                RL   B
3590 03E6 CB11                RL   C
3600 03F8 CB12                RL   D
3610 03EA CB13                RL   E
3620 03EC CB14                RL   H
3630 03EE CB15                RL   pL
3640 03F0 17                  RLA            ;ROTATE LEFT ACC. THROUGH CARRY
3650 03F1 CB06                RLC  (HL)      ;ROTATE LEFT CIRCULAR
3660 03F3 DDCB0506            RLC  (IX+d)
3670 03F7 FDCB0506            RLC  (IY+d)
3680 03FB CB07                RLC  A
3690 03FD CB00                RLC  B
3700 03FF CB01                RLC  C
3710 0401 CB02                RLC  D
3720 0403 CB03                RLC  E
3730 0405 CB04                RLC  H
3740 0407 CB05                RLC  L
3750 0409 07                  RLCA           ;ROTATE LEFT CIRCULAR ACC.
3760 040A ED6F                RLD            ;ROTATE DIGIT LEFT AND RIGHT
3761                                         ;BETWEEN ACC. AND LOC. (HL).
3770 040C CB1E                RR   (HL)      ;ROTATE RIGHT THROUGH CARRY.
3780 040E DDCB051E            RR   (IX+d)
3790 0412 FDCB051E            RR   (IY+d)
3800 0416 CB1F                RR   A
3810 0418 CB18                RR   B
3820 041A CB19                RR   C
3830 041C CB1A                RR   D
3840 041E CB1B                RR   E
3850 0420 CB1C                RR   H
3860 0422 CB1D                RR   L
3870 0424 1F                  RRA            ;ROTATE RIGHT ACC. THROUGH CARRY
```

```
LINE LOC.  OBJECT       SOURCE STATEMENT

3880 0425 CB0E                RRC   (HL)        ;ROTATE RIGHT CIRCULAR
3890 0427 DDCB050E            RRC   (IX+d)
3900 042B FDCB050E            RRC   (IY+d)
3910 042F CB0F                RRC   A
3920 0431 CB08                RRC   B
3930 0433 CB09                RRC   C
3940 0435 CB0A                RRC   D
3950 0437 CB0B                RRC   E
3960 0439 CB0C                RRC   H
3970 043B CB0D                RRC   L
3980 043D 0F                  RRCA              ;ROTATE RIGHT CIRCULAR ACC.
3990 043E ED67                RRD               ;ROTATE DIGIT RIGHT AND LEFT
3991                                            ;BETWEEN ACC. AND LOC. (HL)
4000 0440 C7                  RST   00H         ;RESTART TO LOCATION
4010 0441 CF                  RST   08H
4020 0442 D7                  RST   10H
4030 0443 DF                  RST   18H
4040 0444 E7                  RST   20H
4050 0445 EF                  RST   28H
4060 0446 F7                  RST   30H
4070 0447 FF                  RST   38H
4080 0448 9E                  SBC   A,(HL)      ;SUBTRACT OPERAND FROM ACC.
4090 0449 DD9E05              SBC   A,(IX+d)    ;WITH CARRY.
4100 044C FD9E05              SBC   A,(IY+d)
4110 044F 9F                  SBC   A,A
4120 0450 98                  SBC   A,B
4130 0451 99                  SBC   A,C
4140 0452 9A                  SBC   A,D
4150 0453 9B                  SBC   A,E
4160 0454 9C                  SBC   A,H
4170 0455 9D                  SBC   A,L
4180 0456 DE20                SBC   A,n
4190 0458 ED42                SBC   HL,BC
4200 045A ED52                SBC   HL,DE
4210 045C ED62                SBC   HL,HL
4220 045E ED72                SBC   HL,SP
4230 0460 37                  SCF               ;SET CARRY FLAG (C=1)
4240 0461 CBC6                SET   0,(HL)      ;SET BIT b OF LOCATION
4250 0463 DDCB05C6            SET   0,(IX+d)
4260 0467 FDCB05C6            SET   0,(IY+d)
4270 046B CBC7                SET   0,A
4280 046D CBC0                SET   0,B
4290 046F CBC1                SET   0,C
4300 0471 CBC2                SET   0,D
4310 0473 CBC3                SET   0,E
4320 0475 CBC4                SET   0,H
4330 0477 CBC5                SET   0,L
4340 0521 CB26                SLA   (HL)        ;SHIFT OPERAND LEFT ARITHMETIC
4350 0523 DDCB0526            SLA   (IX+d)
4360 0527 FDCB0526            SLA   (IY+d)
4370 052B CB27                SLA   A
4380 052D CB20                SLA   B
4390 052F CB21                SLA   C
4400 0531 CB22                SLA   D
4410 0533 CB23                SLA   E
4420 0535 CB24                SLA   H
4430 0537 CB25                SLA   L
4440 0539 CB2E                SRA   (HL)        ;SHIFT OPERAND RIGHT ARITHMETIC
4450 053B DDCB052E            SRA   (IX+d)
```

```
LINE LOC.  OBJECT      SOURCE STATEMENT

4460 053F FDCB052E            SRA   (IY+d)
4470 0543 CB2F                SRA   A
4480 0545 CB28                SRA   B
4490 0547 CB29                SRA   C
4500 0549 CB2A                SRA   D
4510 054B CB2B                SRA   E
4520 054D CB2C                SRA   H
4530 054F CB2D                SRA   L
4540 0551 CB3E                SRL   (HL)       ;SHIFT OPERAND RIGHT LOGICAL
4550 0553 DDCB053E            SRL   (IX+d)
4560 0557 FDCB053E            SRL   (IY+d)
4570 055B CB3F                SRL   A
4580 055D CB38                SRL   B
4590 055F CB39                SRL   C
4600 0561 CB3A                SRL   D
4610 0563 CB3B                SRL   E
4620 0565 CB3C                SRL   H
4630 0567 CB3D                SRL   L
4640 0569 96                  SUB   (HL)       ;SUBTRACT OPERAND FROM ACC.
4650 056A DD9605              SUB   (IX+d)
4660 056D FD9605              SUB   (IY+d)
4670 0570 97                  SUB   A
4680 0571 90                  SUB   B
4690 0572 91                  SUB   C
4700 0573 92                  SUB   D
4710 0574 93                  SUB   E
4720 0575 94                  SUB   H
4730 0576 95                  SUB   L
4740 0577 D6FF                SUB   255
4750 0579 AE                  XOR   (HL)       EXCLUSIVE OR OPERAND AND ACC.
4760 057A DDAE05              XOR   (IX+d)
4770 057D FDAE05              XOR   (IY+d)
4780 0580 AF                  XOR   A
4790 0581 A8                  XOR   B
4800 0582 A9                  XOR   C
```

# Differences between HP and the Z80

The following is a list of differences, known to us at the time of printing, that exist between the Hewlett-Packard software and other Z80 Assembler descriptions.

a) The character set has been expanded.

b) Imbedded blanks are allowed in expressions.

c) DEFB and DEFM directives accept expressions and strings.

d) Conditional nesting is allowed.

e) The following pseudo mnemonics have been added:

```
FAIL       IFC
SPC        IFNC
ELSE
```

f) The Macro parameter syntax has been changed. Definition nesting is allowed and Repeat Macros have been added.

g) The following Assembler controls have been added:

```
CONDLIST ON/OFF
SOURCE “file name“
OBJECT “file name“
OBJECT
SYMBOLS ON/OFF
```

h) The Assembler predefined symbol AF’ has been changed to AF.

i) HP software does not have exponentiation **.

j) HP software will not recognize lower case mnemonics, operators, and predefined symbols. For example ’ld’ will not be recognized as ’LD’.

k) The operators = and ^ are not available as they conflict with Macro prefixes.

l) One or two character strings are processed like numbers (8 bit value or 16 bit value), thus the instruction

```
LD IX, ‘AB‘
```

results in the following Object code:

```
DD 214221
```

# Section Eight Macros

## Introduction

A Macro defines a body of text which is automatically inserted into your Source code each time a Macro is called.

Many instruction sequences are often repeated several times in a program, with only certain parameters changed. Instead of rewriting this code each time it occurs, it is useful to code the sequence once and call it with a single instruction whenever it is needed. The code sequence contains dummy parameters that can be replaced with actual values when it is called.

A Macro definition is initiated by the MACRO assembler directive, that lists the name by which the Macro can later be called. It also lists the dummy parameters that are to be replaced during macro expansion. Macros cannot be called before they have been defined. The definition is terminated by the ENDM directive. The instructions bounded by the Macro and the ENDM directives are called the Macro body. Each body, whether in the same program or on another mass storage device, must contain the ENDM directive.

A Macro may be contained in the same program section or on some other mass storage device. If another device is used, it must be called with the SOURCE directive.

Functionally, when the Assembler encounters a MACRO, REPT (Repeat Macro), IRP (Idefinite Repeat Macro), or an IRPC (Idefinite Repeat Character Macro) statement, it stops assembling the microprocessor instructions into Object code. All instructions, after one of the above statements, are stored in the Macro storage area of the Desktop Computer memory until an ENDM statement is seen. If the Assembler encounters another MACRO, REPT, IRP, or IRPC statement before the ENDM statement, then it must see the same number of ENDM statements before it will continue its normal Assembler operation.

The Macro Call statement starts the Macro expansion. It retrieves the lines of Source code from the Macro storage area, makes the necessary parameter replacements, inserts them into the main Source code and then they are assembled.

Duplicating a block of code several times can easily be made using the REPT, IRP and the IRPC directives. In the same way as a MACRO directive, these directives are terminated by ENDM.

The macros, described in this Section, apply to all three microprocessors types whether they be 6800, 8080/85 or the Z80.

## Macro Statement Format

The format of the Macro line is identical to the syntax that is required for the Source code.

```
<LABEL>       MACRO    [=<DUMMY1>  [,=<DUMMY2> . . . ] ]    [<COMMENT> ]
```

Example

```
              MAC1     MACRO    =VAR1,=VAR2
                       .
                       .
                       ENDM
```

The Label defines the Macro name given to the Macro body. Any valid user-defined symbol can be used.

The Instruction contains the MACRO directive that informs the Assembler of a Macro definition.

The Operand field consists of a list of dummy parameters for use in the Macro. A dummy parameter is any valid user-defined symbol, prefixed by an equals sign. If more than one dummy parameter is used, they must be separated by a comma. A maximum of 75 characters is allowed for dummy parameters. Each dummy has an overhead of two characters.

Dummy parameters are not only limited to the Operand field within a Macro body but may be placed in all fields.

A macro definition can be contained completely within the body of another macro definition. That is to say, macro definitions can be nested. No limit is imposed by the Assembler on the depth of macro definition nesting. The nested definitions or redefinitions of the same Macro are performed when the outer Macro is called. Thus the inner Macro cannot be called before the outer Macro has been expanded.

## Macro Call

Once a Macro has been defined, it can be called any number of times within a program. The call consists of the Macro name and the actual parameters to replace the dummy parameters. During assembly, each encountered call is replaced by the Macro definition code with actual parameters substituted for dummy parameters.

```
[<LABEL> ]    <MACRO NAME>            [<PAR1>[, <PAR2> . . . ] ]    [<COMMENT> ]
```

Example

```
     START      MAC1  "2" ,XYZ,^XYZ+2
```

The Label is optional. When used, the value of the current location counter is assigned to it.

The Instruction field contains the Macro name like any other machine or pseudo instruction.

The Operand field contains the actual parameters, separated by commas. Parameters are replaced on a one-to-one basis and in the same order as they are listed in the MACRO directive. If fewer parameters appear in the Macro call than in the definition, a null string is substituted for the remaining dummy parameters. Conversly, if more parameters appear in the Macro call than in the definition, the extras are ignored.

Any parameter within double quotes is passed. Quotes are not needed for anything that obeys the rules of Symbols and Numbers.

An expression can be prefixed with the pre-evaluation character (ˆ) if pre-evaluation is required. This character instructs the Assembler to first evaluate the expression and then pass the value into the Macro body.

---

NOTE

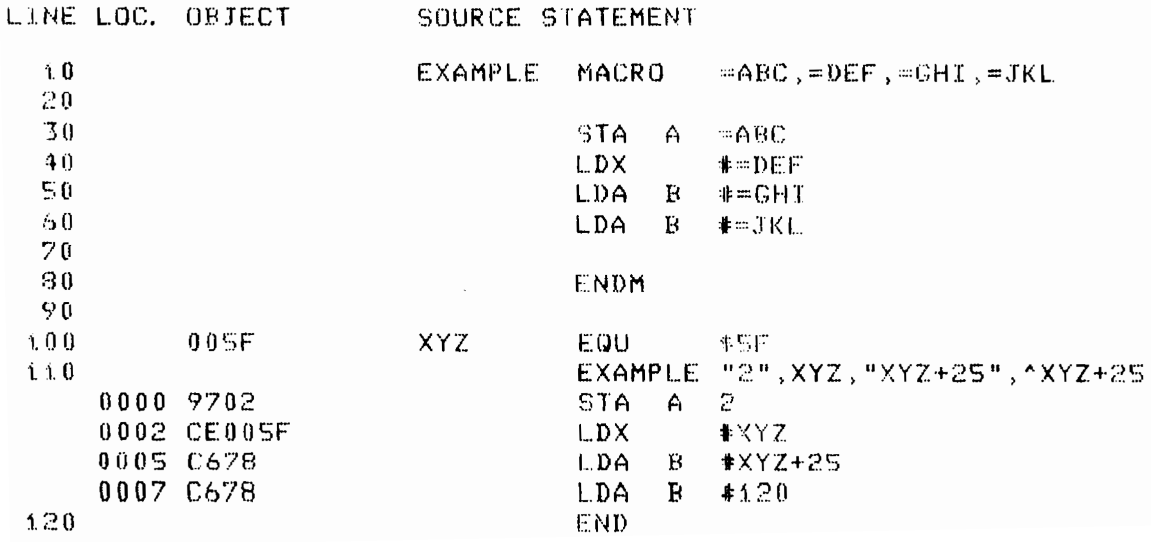
When writing Source code, the Editor's Syntax checker will reject the Macro call statement, and statements involving any Macro dummy parameter. This can be overwritten by using the STORE key in conjunction with the SHIFT key (press SHIFT/STORE). The Syntax accepts the statement as valid and the next line of text may be inserted.

---

# Nested Macro Calls

Macro calls can be nested within Macro definitions up to eight levels. A Macro definition can also contain nested calls to itself (recursive Macro calls) up to eight levels. This operation must be controlled using the conditional assembly directives, IF-ELSE-ENDIF, as described in Section Seven. If this is not done you will run into nesting level overflow.

Example



```
LINE LOC.  OBJECT           SOURCE STATEMENT

  10                        EXAMPLE  MACRO    =ABC,=DEF,=GHI,=JKL
  20
  30                                 STA  A   =ABC
  40                                 LDX      #=DEF
  50                                 LDA  B   #=GHI
  60                                 LDA  B   #=JKL
  70
  80                                 ENDM
  90
 100          005F          XYZ      EQU      #5F
 110                                 EXAMPLE  "2",XYZ,"XYZ+25",^XYZ+25
      0000 9702                      STA  A   2
      0002 CE005F                    LDX      #XYZ
      0005 C678                      LDA  B   #XYZ+25
      0007 C678                      LDA  B   #120
 120                                 END
```

# Macro Directives

## REPT Macro (Repeat Macro)

[ <LABEL> ]     REPT  <EXPRESSION>          [ <COMMENT> ]

This directive causes a sequence of Source code lines to be repeated as defined by the expression. A maximum of 255 times is allowed. All lines appearing between the REPT directive and a subsequent ENDM directive constitute the block to be repeated. If symbols are used in the expression, they must have been previously defined.

For example, Complement Accumulator four times

```
REPT 4
CMA A
ENDM
```

The Source code generated would be:

```
CMA A
CMA A
CMA A
CMA A
```

## IRP (Indefinite Repeat Macro)

[ <LABEL> ]     IRP     =<DUMMY> , <PAR1> [ , <PAR2> . . . ]     [ <COMMENT> ]

The Operand field for this directive must contain one dummy parameter followed by a list of actual parameters. The code sequence between the IRP directive and the subsequent ENDM directive is repeated for every actual parameter. For example:

```
IRP         =BOB, "1", "2", "3"
LDA A       #=BOB
ENDM
```

The above example would generate the following Source code.

```
LDA A       #1
LDA A       #2
LDA A       #3
```

## IRPC (Indefinite Repeat Character Macro)

[ <LABEL> ]     **IRPC**   =<DUMMY> ,<PAR1>     [ <COMMENT> ]

This directive provides the capability to treat each character of PAR1 individually. The code sequence between the IRPC directive and the subsequent ENDM directive is repeated for every character of PAR1. Characters should not be separated by commas. For example:

```
IRPC        =BOB, "ABCDE"
MVI         =BOB, 10
ENDM
```

The Source code generated would be:

```
MVI   A,10
MVI   B,10
MVI   C,10
MVI   D,10
MVI   E,10
```

Note that the difference between the IRP and the IPRC directives is that the IPRC directive only passes one character at a time while the IRP directive passes the whole actual parameter. It is also useful to know that the above three directives may be used as standard Source code and not necessary be inside a MACRO body. They can be seen as a Macro definition and implied Macro call.

## = NUM (Number Macro Parameter)

Using Labels in Macros can cause problems if a Macro is called more than once. In a case like this, the Label would have to be different for each Macro. One way of doing this is to let the Assembler generate different Labels for each Macro called. Using the dummy parameter =NUM causes the Macro to replace the parameter with a four digit number which, in turn, will be incremented by each Macro call. The =NUM parameter is predefined by the Assembler and has a starting value of 0000. A maximum of 9999 numbers can be used.

For example

```
LINE LOC.  OBJECT       SOURCE STATEMENT

  10                    MIKE     MACRO      =VALUE
  20                    L=NUM    LDA A      $=NUM
  30                             =VALUE     L=NUM     THIS IS THE =NUM TIME
  40                             ENDM
  50
  60                    START    MIKE       "BRA"
     0000 9600          L0000    LDA A      $0000
     0002 20FC                   BRA    L0000         THIS IS THE 0000 TIME
  70                             MIKE       "JMP"
     0004 9601          L0001    LDA A      $0001
     0006 7E0004                 JMP    L0001         THIS IS THE 0001 TIME
  80                             MIKE       "LDX"
     0009 9602          L0002    LDA A      $0002
     000B DE09                   LDX    L0002         THIS IS THE 0002 TIME
                                 END
```

## EXITM (Exit Macro)

The EXITM directive provides an alternative way to terminate a Macro expansion or REPT/IRP/IRPC repetition. This instruction terminates only the current expansion level. It can be used anywhere within a Macro body but will not terminate the Macro definition. It can be used to prematurely terminate a macro expansion as, for example, when using the IF-ELSE-ENDIF directive. It cannot be used instead of the ENDM directive. For example:

```
10                  MODUL    MACRO     =STAT
20
30                           LD        (BC),A
40                           =STAT
50                           SBC       A,B
60                           RRC       L
70                           ENDM
80
90                           MODUL     "RST      30H"
    0000  02                 LD        (BC),A
    0001  F7                 RST       30H
    0002  98                 SBC       A,B
    0003  CB0D               RRC       L
100                          MODUL     "RL       D"
    0005  02                 LD        (BC),A
    0006  CB12               RL        D
    0008  98                 SBC       A,B
    0009  CB0D               RRC       L
120                          MODUL     EXITM
    000B  02                 LD        (BC),A
130       0000               END
```

# Section Nine
# Assembler Module

## Introduction

The Assembler module translates your written Source code into an Absolute machine code. This code is then stored in a specified Object file in the format of your particular microprocessor manufacturer. The Assembler performs the translation in two passes. Pass 1 builds the Symbol table, while pass 2 makes the actual translation, outputs the program listing and the Object code before finally producing a cross-reference listing.

## Source File

The input for the Assembler is the Source code stored on your mass storage device. A file can be with or without line numbers. Whether the first Source line contains line numbers or not will decide the format for the entire file. When line numbers are present the Source file begins with the character position 7 (5 positions are required for the line number) and the line numbers are left unchanged. If no line numbers are present then the Assembler assigns them beginning with 10 and increments by 10.

Example of a Source code.

```
10         LDX     $A000
20         JMP     X
30   LOOP  CMP A   #'S
40         BNE     LOOP
```

## Object Files

The Object file is the file on the mass storage device where the Assembler program stores the Object code during pass 2. It is stored in the format used for transferring to your microprocessor system. Refer to the Console Module Section for further information.

# List/Cross Reference Format

For an example of the List and Cross Reference formats, refer to Section Seven – Assembler Instructions.

# Using The Assembler Module

## SOURCE Instruction

**SOURCE** " <FILE NAME> [msus] " [ , <CONTROL1> [ , <CONTROL2> . . . ] ]

Once the Assembler module has been selected from the Main Menu, the CRT displays

```
COMMAND MODE
SOURCE
```

Enter the Source file name. The msus is also required if it is different from the default value contained in the default file. Only one Source file can be specified to start the assembly.

At this stage it is also possible to add Assembler Controls in the SOURCE line. For example:

```
COMMAND MODE
SOURCE "MIKE:T14", NOLIST
```

Press CONTINUE

The Source code stored in file MIKE is assembled and no listing is provided by the printer.

## OBJECT Instruction

**OBJECT** [ " <FILE NAME> [msus] " ] [ , <CONTROL1> [ , <CONTROL2> . . . ] ]

The CRT displays,

```
COMMAND MODE
OBJECT
```

If the Object code is not to be stored, do not enter a file name, just press CONTINUE.

Example with Object code stored.

```
COMMAND MODE
OBJECT "EDI:T14"
```

Your Source code will be Assembled and the Object code is stored in the file named EDI.

Assembler Controls can also be added in this line as with the Source line. The last seen Controls are taken as valid. For instance, Controls written in the Source instruction can be corrected in the Object instruction. However, any Controls contained in the Source code will override those listed in the instructions above.

## BREAK Key

Temporarily interrupts the operation of the Assembler and allows use of the Mainframe functions. For instance, when you want to change printer paper, make a calculation etc. etc.

## EXIT Key

Takes you from the Assembler module and returns you to the Main Menu.

# Section Ten
# Console Module

## Introduction

This module provides the communication between your Desktop Computer and a Microprocessor system. It permits you to transfer your final Object code to the Microprocessor system for testing and debugging. It also allows you to control the functions of the Monitor ROM during the testing/debugging process. Data is transferable in both directions.

**Listed below are the Microprocessor systems that Hewlett-Packard have had experience with and on which the information in this Section is based. To the best of our knowledge the facts stated in this Section are correct however, we accept no responsibility for malfunction of the software due to any changes that may have been made by the Microprocessor manufacturer since the time of writing.**

Systems complete with a power supply and a housing can be directly connected to a Hewlett-Packard Desktop Computer by use of the RS232C Interface cable, HP 98036A Option 001. A female connector, compatible with the Hewlett-Packard's Interface cable, must be connected to individual microprocessor boards.

| 6800 | 8080/85 | Z80 |
|---|---|---|
| Motorola | Intel | Zilog |
| Evaluation Module II with Minibug II or Minibug III | SBC 80/10<br>80/20<br>80/30 | Z80 – MCB/4<br>– MCB/16 |
| *Exorcisor with Exbug Monitor ROM | With 1K Monitor ROM. | With M01 1K Monitor ROM. |
|  | These OEM boards are also available in a case with a power supply, under the system numbers; | A case, power supply, etc. . for these boards are available under the number; |
|  | SYS 80/10<br>80/20<br>80/30 | Z80 – SCE 4 |

* Note: The Motorola Exorcisor has the CTS line connected to ground. It is necessary to cut this and connect it to the RS232C socket via an RS232C/TTL level converter. The MC1489AL/U29B may be used for this purpose.

# Using Other Microprocessor Systems

Generally, this is possible providing that their RS232C interface and Monitor ROM have the same specifications and characteristics as the above mentioned systems and also work in half-duplex mode.

Other systems perhaps not using the RS232C interface can be interfaced provided that you write your own version of the Console module.

The format with which the Object codes are stored on your mass storage device are included later in this section.

# Microprocessor System Modifications

Three Microprocessor systems have been used extensively by Hewlett-Packard and the following wiring information is that which was required to use the Microprocessor System with the Hewlett-Packard hardware.

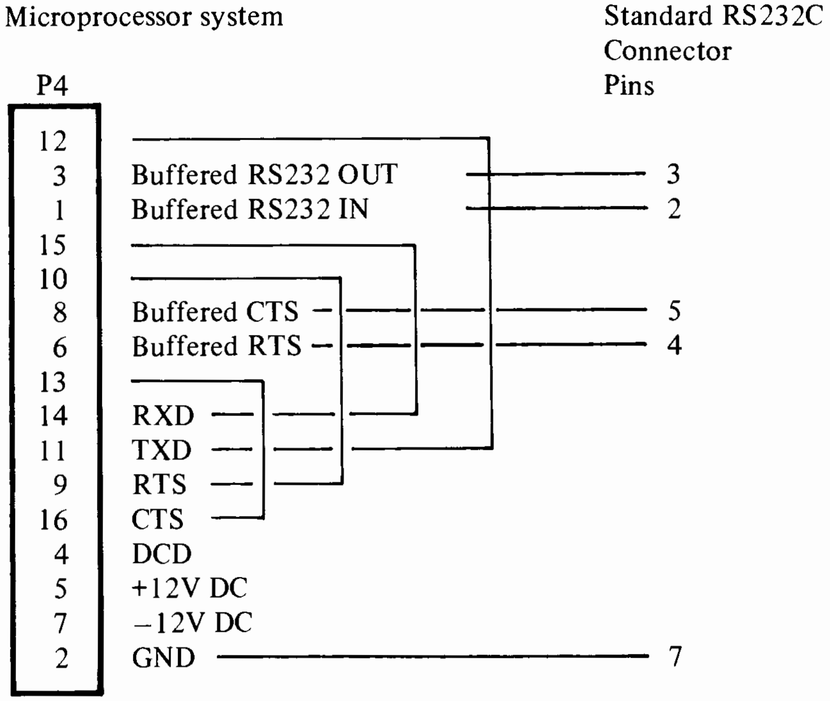
## Motorola Evaluation Module II

a) A wiring change was made to the circuit board to set the baud rate to 9600 bauds.

Modify the socket designated U24, in the bottom left-hand corner of the board, such that pin 5 is connected to pin 16. Above the socket are holes designated with the letter E. Connect E7A to E14.

b) Connections for an interface socket.

The following diagram illustrates the connections for a standard 25-pin RS232C socket.



## Intel SBC 80/30 mounted in the SYS 80/30 case

The following changes were made to the circuit board:

a) The USART 8251A was found not to work with the Hewlett-Packard Console module because after stopping the USART transmitting data by removing the CTS line it repeats the least sent character on moving the CTS line back to its original condition. The 8251 operates correctly. This information was true as of September 1979.

b) Break the connection between circuit board pins 67 and 68 in order to activate the CTS line.

## Zilog Z80 — MCB/16

The following changes were made to the circuit board:

Jumpers on connector J4.

Break the connection between pins 5 and 6. This breaks the short circuit between CTS and RTS.

Connect the pin 6 to pin 13. This connects the CTS line to the outside RS232C connector.

# RS232C Lines

The following interface lines are used for communication between the Desktop Computer and the Microprocessor system.

**Transmitted Data, (TD), pin** 2. This line outputs data from the Console Module to the Microprocessor system.

**Received Data (RD), pin** 3. This line inputs data from the Microprocessor system to the Console Module.

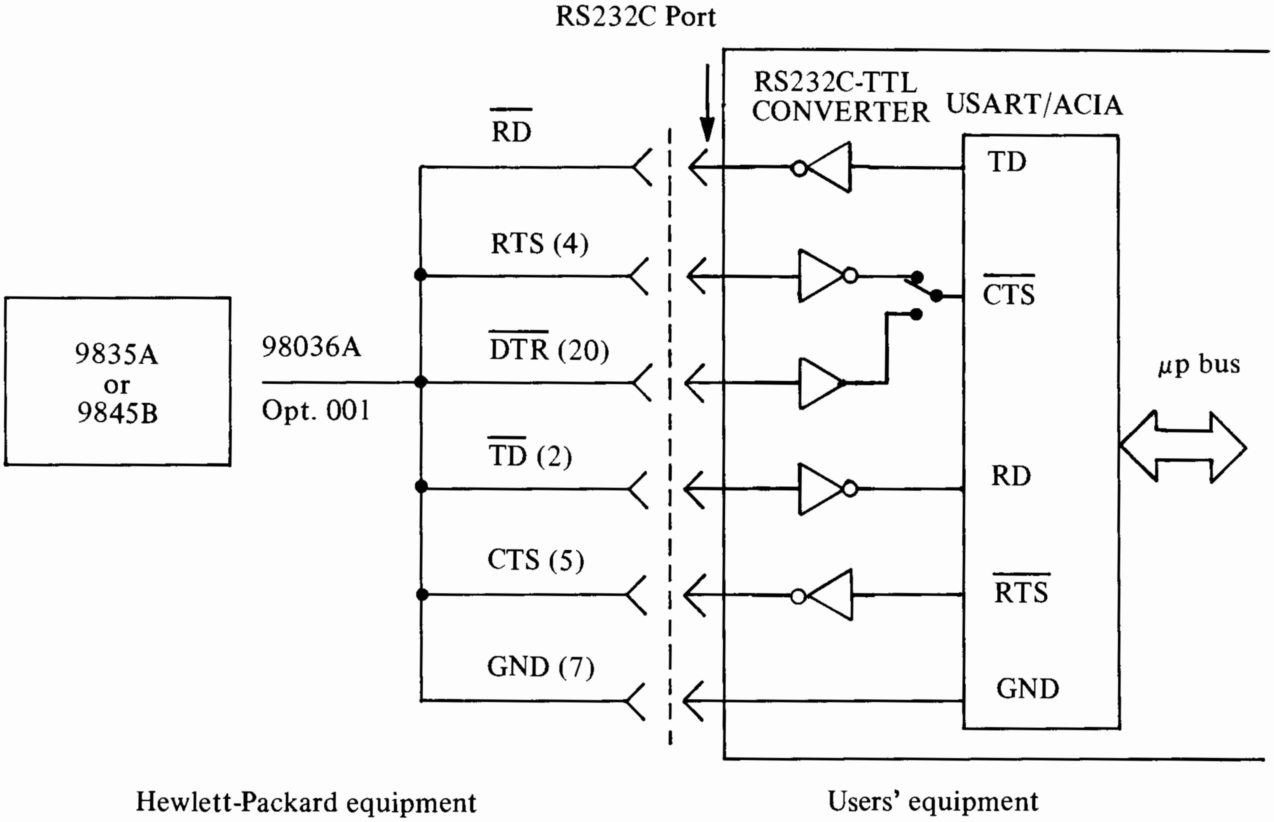
**Request to Send (RTS), pin 4.** Enables the Microprocessor system so that data can be transmitted. The moment this line falls the Microprocessor system must stop transmitting after the entire character has been sent. With some Microprocessor systems this line is connected to ground. This connection must be broken and the CTS line connected to the RS232C socket.

**Clear to Send Data (CTS), pin** 5. Enables the Console Module so that data can be transmitted. If this line falls, the Console module stops transmission after the entire character has been sent.

**Data Terminal Ready (DTR), pin 20.** Line goes high to prevent the Microprocessor system from sending data. Indicates a busy state of the Console module. This line has always the complement state of the RTS line. Ony one or the other is used.

**GND, pin** 7. Circuit ground

The diagram below illustrates the connection between Hewlett-Packard equipment and a Microprocessor Development System.



# HP 98036A Interface

## System and Cable Compatibility

Peripheral default values, contained in the "Serial Interface RS232C Sub-menu" of the Initialization Module, and the various switch settings on the Interface card must be the same.

Most Microprocessor Systems use an 8-bit, no parity, 1 Stop bit data. Check the Operating manual for your Microprocessor system that this is correct. If it is correct, the following values are recommended and should be set accordingly:

| | |
|---|---|
| Baud Rate . . . . . . . . . . . | 9600 |
| Data Word Length. . . . . . | 8 |
| Parity Check . . . . . . . . . . | None |
| Stop Bit. . . . . . . . . . . . . . | 1 |

## Interface Dismantling

Before installing the Interface cable in your Desktop Computer, check the configuration switch settings on the Interface.

To access the switches it is necessary to separate the Interface housing. Refer to the figure when using the procedure below.

1) Remove the four screws that hold the rear housing to the front house.

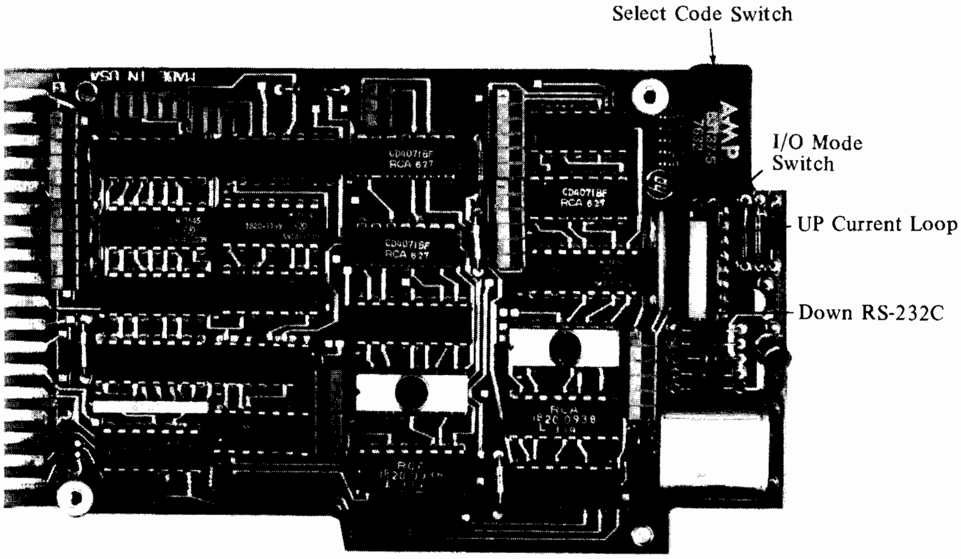
2) Pull the rear housing away from the front slightly, disconnect the cable connector from the PC assembly and remove the rear housing.

3) Remove the remaining four screws in the front housing and separate the front housing cases.

4) Carefully separate the printed circuit assemblies.

5) The various interface switches are shown in the figure below.

6) After setting the switches, reverse this procedure to assemble the interface. Be sure that the pins on the A2 assembly are properly seated in the connectors on the A1 board assembly.



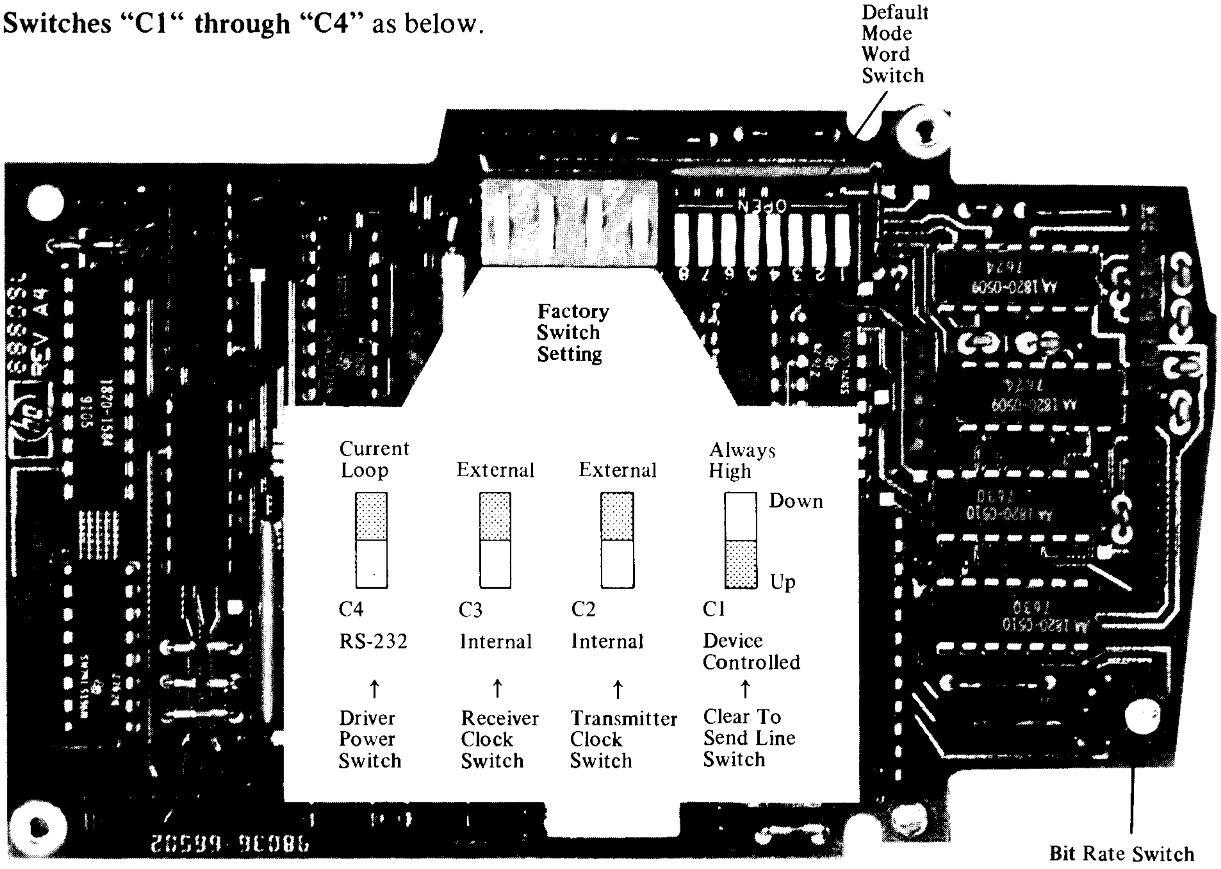
## Switch Settings

Once the printed circuit boards have been parted, set the switches as follows:

**Select Code switch** to 11.

**I/O Mode switch** to "Down RS232C"

**Switches "C1" through "C4"** as below.



**Default Mode Switch**

The switch setting are set automatically by Desktop Computer software.

**Baud Rate switch.** Position 1. (9600 baud)

If needed, other baud rate settings are:

| Switch position | Baud Rate | Switch position | Baud Rate |
|---|---|---|---|
| 2 | 4800 | 6 | 600 |
| 3 | 2400 | 7 | 300 |
| 4 | 1800 | 8 | 150 |
| 5 | 1200 | 9 | 110 |
|  |  | 0 | 75 |

## Interface Installation

After the Interface switches have been set, assemble the interface housing by reversing the disassembly procedure. Make sure that the pins on the A2 assembly are properly seated in the connectors on the A1 assembly. With the Desktop Computer switched off, install the interface housing into any one of the I/O slots on the back of the Computer. Connect the other end of the cable to your microprocessor system.

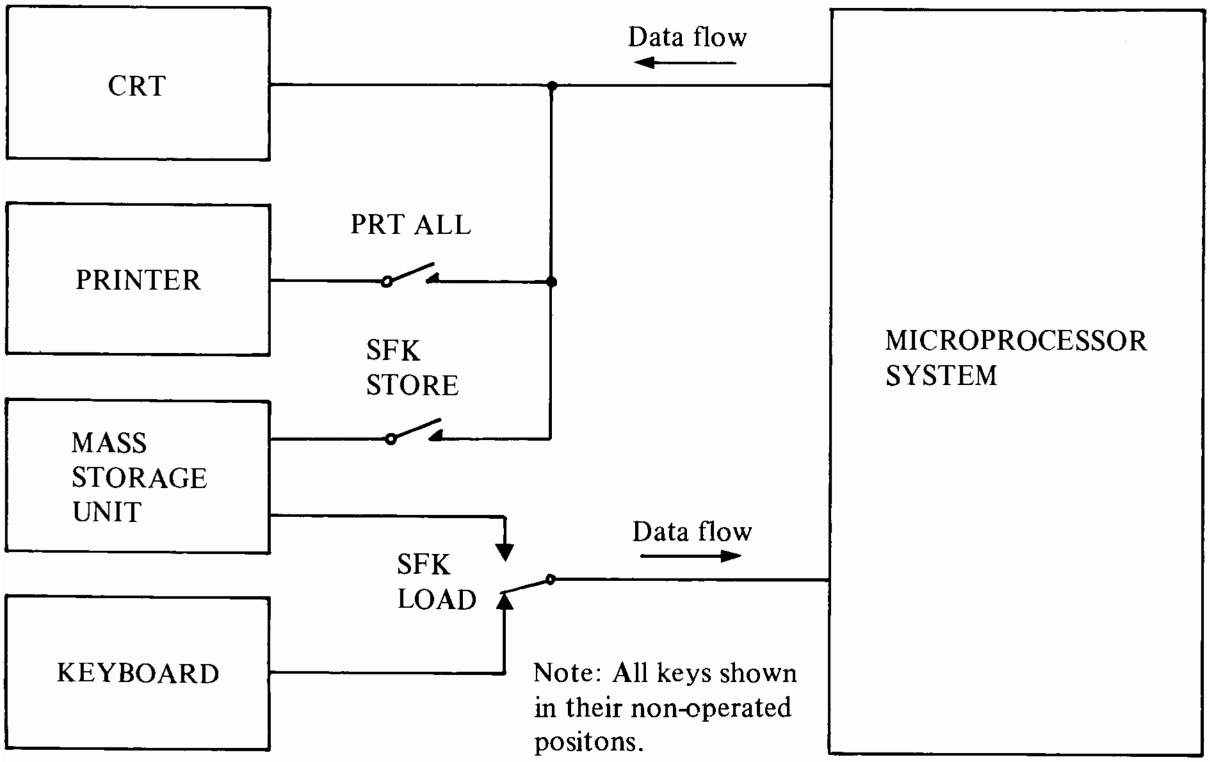
**CAUTION**
Before making any connections to the Desktop Computer, ensure that it is turned off.

# Running the System

Call-up the Main Menu on the Desktop Computer.

With the Main Menu present, select SFK k3 for the Console Module. The module is loaded.

A functional diagram of the Console and an associated Microprocessor System is shown below.



Note: All keys shown in their non-operated positons.

## Special Function Keys

The following commands may only be called by pressing the SFKs. They may not be typed.

**LOAD**

**LOAD** " <FILE NAME> [msus] "

This command allows you to transfer you data (e.g. Object code) from your mass storage device to the Microprocessor System.

**STORE**

**STORE** " <FILE NAME> [msus] "

This Command allows you to store data, for example, Object code sent by the Microprocessor system to the mass storage device.

To terminate the transfer, this key must be pressed once more at the end of data transfer.

If a file must be created, you are asked for the file capacity in bytes after the Mainframe STORE key has been pressed. You must enter the number of bytes which are to be input. Remember, when you are dumping the microprocessor memory in the Object code format, the microprocessor sends about three times the amount of bytes that you are dumping because of the formatting.

**BREAK**

This SFK takes you out of the Console module and allows you to use the standard features of the Desktop Computer. This would be used, for example, when a CAT (Catolog) is required or when you need to make a calculation. Use CONTINUE to return.

**EXIT**

Allows you to leave the Console Module. Main Menu SFK k3 must be selected to return.

---

**NOTE**

When using either the LOAD or the STORE command, it is possible to interrupt the operation by pressing the appropriate key, LOAD or STORE.

---

## Mainframe Keys

PRINT ALL — All commands and data that appear on the CRT are duplicated on the printer.

STORE — Sends the ASCII character CR to the Microprocessor system.
Hexadecimal value – 0D

SHIFT/STORE — Sends the ASCII character LF to the Microprocessor system.
Hexadecimal value – 0A

BACKSPACE/
Left arrow — Sends the ASCII character BS to the Microprocessor system.
Hexadecimal value – 08.

TAB — Sends the ASCII character HT to the Microprocessor system.
Hexadecimal value – 09.

CLEAR LINE — Sends the ASCII character DEL to the Microprocessor system.
Hexadecimal value – 7F.

FWD/
Right arrow — Sends the character, where the cursor is positioned on the CRT, to the Microprocessor system.

---

**CAUTION**

When either the Load or Store operation is taking place do not operate any of the keyboard keys as any interruption could cause loss of information.

---

# Object Code Formats

The Assembler program stores the Object code in this format on the mass storage device. It is transferred in this format by the Console to the Microprocessor system. An exception is the Intel Microprocessor system which requires a converted format. Refer to the paragraph, Intel Conversion, given below.

## Intel

```
Number of bytes    2           4            2                             2

            : (LENGTH)    (ADDRESS)     (TYPE)    (OBJECT CODE . . . )   (CHECK SUM)
            : (LENGTH)    (ADDRESS)     (TYPE)    (OBJECT CODE . . . )   (CHECK SUM)
            : (LENGTH)    (ADDRESS)     (TYPE)    (OBJECT CODE . . . )   (CHECK SUM)
               .
               .
               .
            : (LENGTH)    (ADDRESS)     (TYPE)    (OBJECT CODE . . . )   (CHECK SUM)
```

Where LENGTH is number of Object code bytes in the data string. An end-of-file string has 00 in this field.

ADDRESS specifies where the first Object code byte is to be stored. Successive bytes are stored in successive memory locations.

If you put an <address> after the END directive in your Source code then this address will appear in this field if it is the end-of-file string. If you put no <address> after the END statement then the end-of-file string contains only a colon followed by two zeros.

TYPE. Data strings have 00 in this position, end-of-file strings have 01.

OBJECT CODE is your Object code program that is transfered to the Microprocessor System.

CHECK SUM is two's complement of the 8-bit sum of the data that results from converting each pair of ASCII hexadecimal digits to one byte of binary from the LENGTH to, and including, the last byte of the Data string.

## Intel Conversion

The 1K Monitor ROM, used in the Intel Microprocessor systems SBC 10/20/30, accepts data in the format given opposite when the Object code is input with the R Command.

As the protocol of this command is designed for use with a paper tape reader it will not work satisfactorily with the Console module in its existing form. The Console module converts this code into a format which can be transmitted to the Microprocessor system. The code is converted to the following format:

```
Iyyyy CR
<data . . . > ESC
```

| | |
|---|---|
| where I | is the Insert command to the Monitor ROM |
| yyyy | is the start address |
| CR | is the ASCII character carriage return |
| <data> | the Object code in hexadecimal code. |
| ESC | is the ASCII character Escape |

**NOTE**

The I (Insert) command is included in the data sent form the Desktop Computer. You do not need to input it.

When you are using the Intel 1K Monitor ROM it is necessary to activate the conversion process by giving a positive answer (Y) to the question contained on the Initialization module RS232C submenu that asks if an "Intel Format Conversion" should be made.

If "Intel Format Conversion" has been selected then the Assembler program puts a W in front of the Object code file. Only files with a W at the beginning of the Object code are converted by the Console module.

This permits you to use the conversion process for Object code. It also allows you to transfer non-Object code data between the Microprocessor system and the mass storage device by not having this W at the beginning of the data file.

An automatic 'read after write' comparision is made by the Console. That is to say, each character is echoed from the Microprocessor system and checked by the Console that it was the correct character sent.

## Motorola

| | | | | | |
|---|---|---|---|---|---|
| Number of bytes | | 2 | 4 | | 2 |
| | S0 | (LENGTH) | (ADDRESS) | (OBJECT CODE . . . ) | (CHECK SUM) |
| | S1 | (LENGTH) | (ADDRESS) | (OBJECT CODE . . . ) | (CHECK SUM) |
| | S1 | (LENGTH) | (ADDRESS) | (OBJECT CODE . . . ) | (CHECK SUM) |
| | | | . | | |
| | | | . | | |
| | | | . | | |
| | S9 | (LENGTH) | (ADDRESS) | (OBJECT CODE . . . ) | (CHECK SUM) |

Where S0 indicates the start of the transmission and that the data file will follow. The Object code in this string is not stored in memory.

S1 preceeds the data strings containing your Object code.

S9 indicates the finish of the transmission. If you have not put the MON directive in your Source code, then the EOF string consists of S9. If you have used the MON directive, then this ADDRESS will be the <address> that you entered in your Source code.

After receiving the Object code transmission some Microprocessor Systems automatically commence running the program at the address specified.

LENGTH is the number of bytes in the data string from (LENGTH) until, and including, the CHECK SUM. An end-of-file string has 03H in this field.

ADDRESS specifies where the first Object code byte is to be stored. Successive bytes are stored in successive memory locations.

OBJECT CODE is your Object code program that is transfered to the Microprocessor System.

CHECK SUM is one's complement of the 8-bit sum of the data that result from converting each pair of ASCII hexadecimal digits to one byte of binary from the LENGTH to, and including, the last byte of the Data string.

## Zilog

```
Number of bytes        2          4          2                              2

                 : (LENGTH) (ADDRESS) (TYPE) (OBJECT CODE . . . ) (CHECK SUM)
                 : (LENGTH) (ADDRESS) (TYPE) (OBJECT CODE . . . ) (CHECK SUM)
                 : (LENGTH) (ADDRESS) (TYPE) (OBJECT CODE . . . ) (CHECK SUM)
                 .
                 .
                 : (LENGTH) (ADDRESS) (TYPE) (OBJECT CODE . . . ) (CHECK SUM)
```

Where LENGTH is number of Object code bytes in the data string. An end-of-file record has 00 in this field.

ADDRESS specifies where the first Object code byte is to be stored. Successive bytes are stored in successive memory locations.

TYPE. Data records have 00 in this position, end-of-file records have 01.

OBJECT CODE is your Object code program that is transfered to the Microprocessor System.

CHECK SUM is two's complement of the 8-bit sum of the datas that result from converting each pair of ASCII hexadecimal digits to one byte of binary from the LENGTH to, and including, the last byte of the Data string.

# Useful Operating Hints

The following information is based on Hewlett-Packard's experience with the previously mentioned Microprocessor systems. Due to changes that microprocessor manufacturers make from time to time it could happen that some instructions do not function as given in this manual. If this is the case, we recommend that you refer to the manufacturer's documentation.

## Intel Microprocessor Systems

The following procedure downloads your Object code into the Microprocessor System

a) Press **SFK k3** to select the Console module.

b) Press the Microprocessor system **RESET** button.

c) Press the capital letter U on the Desktop Computer keyboard.

A decimal point (.) appears on the Desktop Computer CRT. This indicates that the Microprocessor system is waiting for further commands.

d) Press **SFK LOAD**

e) Enter the **FILE NAME**

**LOAD** " <FILE NAME> [msus]"

The file name is where you stored your Object code that was generated by the Assembler.

f) Press **CONTINUE.**

After loading is complete a decimal point (.) will appear in the CRT indicating the Microprocessor system is waiting for further commands.

Refer now to the instruction manual of your Microprocessor system for the commands necessary to run and debug your Object code.

When you have finished testing and debugging you may want to retransfer your modified Object code back to the mass storage unit of your Desktop Computer.

a) Press **SFK STORE.**

Type the name you want to give to the Object code. For example.

**STORE** " <FILE NAME> [msus]"

b) Press **CONTINUE.**

c) Press the capital letter W on the Desktop Computer keyboard.

d) Type the start and finish address. For example:

**WAAAA, BBBB**

Where **AAAA** is the start address and **BBBB** is the finish address

e) Press **STORE** to transmit the ASCII character CR to the Microprocessor system.

f) When the Object code has been transfered (this can be seen from the CRT), press the SFK STORE once more.

## Motorola Microprocessor Systems

The following procedure downloads your Object code into the Microprocessor System.

a) Press **SFK k3** to select the Console module.

b) Press the Microprocessor system **RESET** button.

An asterisk (*) appears on the Desktop Computer CRT. This indicates that the Microprocessor system is waiting for further commands.

c) Press the capital letter **L** on the Desktop Computer keyboard.

d) Press **SFK LOAD**

e) Enter the **FILE NAME**

**LOAD** " <FILE NAME> [msus]"

The file name is where you stored your Object code that was generated by the Assembler.

f) Press **CONTINUE.**

After loading is complete an asterisk (*) will appear in the CRT indicating the Microprocessor system is waiting for further commands.

Refer now to the instruction manual of your Microprocessor system for the commands necessary to run and debug your Object code.

When you have finished testing and debugging you may want to retransfer your modified Object code back to the mass storage unit of your Desktop Computer.

a) Press **SFK STORE.**

Type the name you want to give to the Object code. For example:

**STORE** "<FILE NAME> [msus]"

b) Press **CONTINUE.**

c) Press the capital letter **P** on the Desktop Computer keyboard.

d) Type the start and finish address. For example:

**P AAAA BBBB**

Where **AAAA** is the start address and **BBBB** is the finish address

e) When the Object code has been transfered (this can be seen from the CRT), press the **SFK STORE** once more.

## Zilog Microprocessor Systems

The following procedure downloads your Object code into the Microprocessor System

a) Press **SFK k3** to select the Console module.

b) Press the Microprocessor system **RESET** button.

The sign ( > ) appears on the Desktop Computer CRT. This indicates that the Microprocessor system is waiting for further commands.

c) Press the capital letter **L** on the Desktop Computer keyboard.

d) Press the **STORE** key.

This sends the ASCII character CR to the Microprocessor system.

e) Press **SFK LOAD**

f) Enter the **FILE NAME**

```
LOAD " <FILE NAME> [msus]"
```

The file name is where you stored your Object code that was generated by the Assembler.

g) Press **CONTINUE.**

After loading is complete the sign ( > ) will appear in the CRT indicating the the Microprocessor system is waiting for further commands.

Refer now to the instruction manual of your Microprocessor system for the commands necessary to run and debug your Object code.

When you have finished testing and debugging you may want to retransfer your modified Object code back to the mass storage unit of your Desktop Computer.

a) Press **SFK STORE.**

Type the name you want to give to the Object code. For example.

```
STORE "<FILE NAME> [msus]"
```

b) Press **CONTINUE.**

c) Press the capital letter **P** on the Desktop Computer keyboard.

d) Type the start and finish address. For example:

```
P AAAA,BBBB
```

Where **AAAA** is the start address and **BBBB** is the finish address

e) Press **STORE** to transmit the ASCII character CR to the Microprocessor system.

f) When the Object code has been transferred (this can be seen from the CRT), press the **SFK STORE** once more.

## CRT Driver Used By The Console

All received characters are immediately shown on the CRT except for most of the ASCII Control characters (code less than 20H) which are ignored by the CRT driver. Only the following ASCII Control characters are interpreted:

| Hex. Value | Name |
|---|---|
| 07 | BEL (beep) |
| 08 | BS |
| 09 | HT (Interpreted as space) |
| 0A | LF |
| 0C | FF (Interpreted as LF) |
| 0D | CR |

Characters with values in the range 80H through 87H are interpreted as CRT Control characters:

| Hex. Value | Name |
|---|---|
| 80 | CLR (Clear all special features) |
| 81 | IV (Inverse video) |
| 82 | BL (Blinking) |
| 83 | IV + BL |
| 84 | UL (Underline) |
| 85 | IV + UL |
| 86 | BL + UL |
| 87 | IV + BL + UL |

The CRT Control characters are used by the Console CRT driver in the same way that the Mainframe uses them.

The characters with value 88H through FFH are interpreted as foreign characters.

Many microprocessor programs buffer the Console inputs in the Microprocessor system. When doing this it is useful to define “BS“ or “Left arrow“ to cancel the last buffered character and use the “Right arrow“ to advance in the microprocessor input buffer.

## Error Handling

If an error occurs, the appropriate message is given on the CRT. To continue program execution it is necessary to press the CONTINUE key.

# Section Eleven
# Examples on Applying the Pack

## Introduction

This section takes a look at some of the refinements of Source code writing. It is a collection of examples that illustrate ways of making the best use of commands and directives.

## The Editor

### Change

It could well be that you have some Source code from another Development System that you want to assemble on your Hewlett-Packard Desktop Computer. It is possible that the Source code, from the other System, contains some slight differences in the mnemonics to your –hp– software and you would like to change these.

Alternatively, if your have entered a lot of Source code with the Editor program and have incorrectly spelt a word from beginning to end, it would be useful to be able to change the spelling automatically.

In both these cases, you should use the CHANGE statement.

Example

The following source code has the statement CAL spelt wrongly, it should have been written **CALL**.

```
10            MOV     E,A       ;CALCULATE RECORD LENGTH
20            MVI     D,0       ;INITIALIZE D FOR HOLDING CHECKSUM
30            MVI     C,':'     ;
40            CAL     PO        ;PUNCH RECORD MARK CHARACTER
50            MOV     A,E       ;PUT RECORD LENGTH IN A
60            CAL     PBYTE     ;PUNCH RECORD LENGTH
70            CAL     PADR      ;PUNCH STARTING ADDRESS
80            XRA     A         ;ZERO A
90            CAL     PBYTE     ;RECALL RECORD TYPE
```

Using the statement **CHANGE "CAL" TO "CALL"** produces the following:

```
10         MOV     E,A      ;CALLCULATE RECORD LENGTH
20         MVI     D,0      ;INITIALIZE D FOR HOLDING CHECKSUM
30         MVI     C,':'    ;
40         CALL     PO       ;PUNCH RECORD MARK CHARACTER
50         MOV     A,E      ;PUT RECORD LENGTH IN A
60         CALL     PBYTE    ;PUNCH RECORD LENGTH
70         CALL     PADR     ;PUNCH STARTING ADDRESS
80         XRA     A        ;ZERO A
90         CALL     PBYTE    ;RECALLL RECORD TYPE
```

Notice that not only the CAL statements in lines 40, 60, 70 and 90 have been changed but that the word CALCULATE in line 10 and RECALL in line 90 have also been changed.

To avoid this, use the statement in a different way. Namely

**CHANGE "    CAL    " TO "    CALL "** (Note the spaces before and after the quotes).

```
10         MOV     E,A      ;CALCULATE RECORD LENGTH
20         MVI     D,0      ;INITIALIZE D FOR HOLDING CHECKSUM
30         MVI     C,':'    ;
40         CALL    PO       ;PUNCH RECORD MARK CHARACTER
50         MOV     A,E      ;PUT RECORD LENGTH IN A
60         CALL    PBYTE    ;PUNCH RECORD LENGTH
70         CALL    PADR     ;PUNCH STARTING ADDRESS
80         XRA     A        ;ZERO A
90         CALL    PBYTE    ;RECALL RECORD TYPE
```

It is now clear that the comments have not been affected and that the listing is in the correct position.

The reason for the difference is that space is also a valid character for the change algorithm and is included in the conversion.

## Find

Normally Source code can be changed automatically by using the CHANGE statement as above. However sometimes you do not want to change all of the items. It can sometimes happen that despite putting spaces or other characters in the CHANGE statement your still cannot automatically change the items you want and leave the others intact.

In a case like this, use the FIND statement. Items are found and you can modify them or not as you wish.

In the following example it is not possible to change only the instructions in the instruction field from JUMP to JMP with the CHANGE statement without also changing the words JUMP in the comment area.

```
10        MOV      E,A       ;CALCULATE RECORD LENGTH
20        MVI      D,0       ;INITIALIZE D FOR HOLDING CHECKSUM
30        MVI      C,':'     ;
40 LABEL:JUMP■     PO        ;JUMP TO MARK CHARACTER
50        MOV      A,E       ;PUT RECORD LENGTH IN A
60        JUMP■    PBYTE     ;PUNCH RECORD LENGTH
70        JUMP■    PADR      ;PUNCH STARTING ADDRESS
80        XRA      A         ;ZERO A
90 START:JUMP■     PBYTE     ;JUMP TO RECORD
```

However, by inputting,

**FIND "JUMP"**

the cursor will stop at the points market with a■ (in the example) and you can change those that you want to.

## Copy

This statement is very useful for moving sections of Source code from one position in the code to another. If the original code is not deleted afterwards it can be used as a means of duplicating routines in the Source code. In the following example it is used to duplicate code.

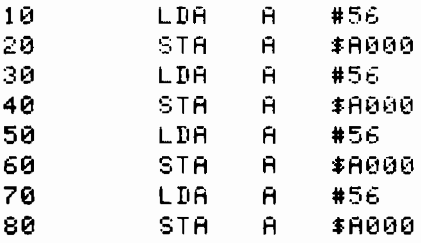
The following statements must first be entered by hand.

```
10        LDA   A   #56
20        STA   A   $A000
```

If you now execute **COPY 10 , 20 TO 30** you get

```
10        LDA   A   #56
20        STA   A   $A000
30        LDA   A   #56
40        STA   A   $A000
```

If you now enter **COPY 10 , 40 TO 50** you get



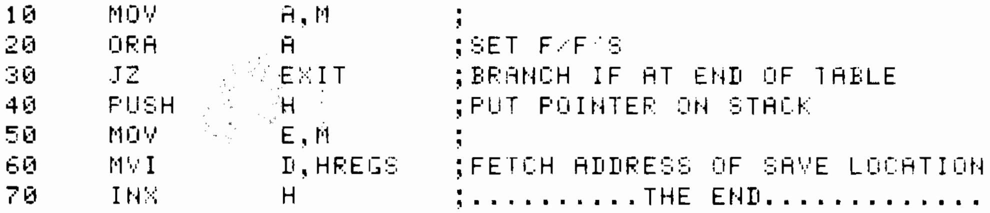
```
10       LDA   A   #56
20       STA   A   $A000
30       LDA   A   #56
40       STA   A   $A000
50       LDA   A   #56
60       STA   A   $A000
70       LDA   A   #56
80       STA   A   $A000
```

In this manner, code can be duplicated as often as desired.

As long as the destination line number does not exist and that it is smaller than the lowest line number to be copied or, higher than the highest, then there are no limits as to what can be done with this statement.

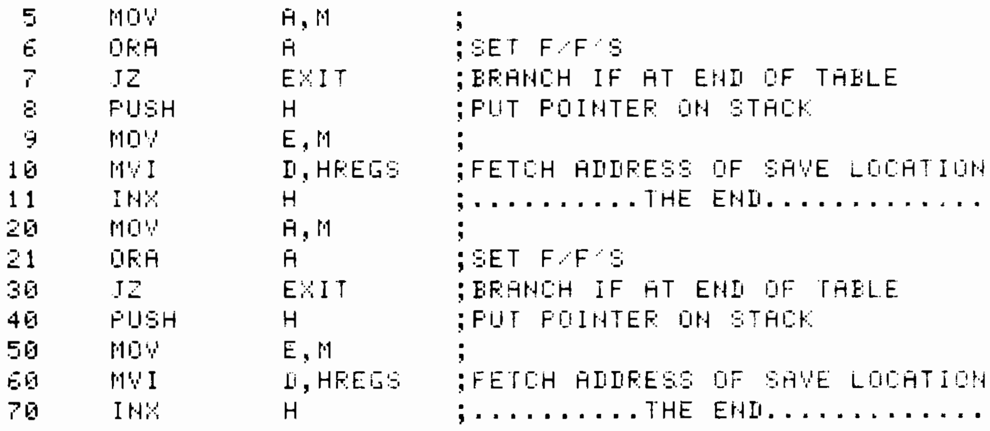
If you try to copy into an area where there is insufficient space between the line numbers, then the program will automatically renumber those line numbers at the high end of this area such that this insert works. The whole Source code from this point on will not be renumbered, only those lines necessary to enable the insertion.

This example demonstrates what this command can do.



```
10    MOV      A,M       ;
20    ORA      A         ;SET F/F'S
30    JZ       EXIT      ;BRANCH IF AT END OF TABLE
40    PUSH     H         ;PUT POINTER ON STACK
50    MOV      E,M       ;
60    MVI      D,HREGS   ;FETCH ADDRESS OF SAVE LOCATION
70    INX      H         ;..........THE END.............
```

Using the command **COPY 10 , 70 TO 5** results in the following:



```
 5    MOV      A,M       ;
 6    ORA      A         ;SET F/F'S
 7    JZ       EXIT      ;BRANCH IF AT END OF TABLE
 8    PUSH     H         ;PUT POINTER ON STACK
 9    MOV      E,M       ;
10    MVI      D,HREGS   ;FETCH ADDRESS OF SAVE LOCATION
11    INX      H         ;..........THE END.............
20    MOV      A,M       ;
21    ORA      A         ;SET F/F'S
30    JZ       EXIT      ;BRANCH IF AT END OF TABLE
40    PUSH     H         ;PUT POINTER ON STACK
50    MOV      E,M       ;
60    MVI      D,HREGS   ;FETCH ADDRESS OF SAVE LOCATION
70    INX      H         ;..........THE END.............
```

Notice that the original source code has been duplicated as from line 5 and that lines 10 and 20 have been renumbered to 20 and 21 to allow this. Line numbers 30 to 70 have not been affected.

# Loading and Storing Source Code without Line numbers

As a user you might want to input some Source code into your Desktop Computer that was originated by some other system.

You can do this by interfacing the other system or by reading the Source code from paper tape with the HP 9883A Paper Tape Reader.

The Source code should then be stored on your magnetic storage device so that it can either be modified by the Editor or assembled.

In the eventuality that your Source code does not have line numbers or that the line numbers are not complete or are not accepted by the Editor you can load them into the Editor with the statement.

**LOAD** "<FILENAME>" @

The code will then be loaded with possibly 2 sets of line numbers if your original code had line numbers.

Example

The following Source code is stored on the mass storage device under the file name RECORD

```
        MOV         E,A
        MVI         D,0
30      MVI         C,':'
40      CALL        PO
50      MOV         A,E
60      XRA         A
70      CALL        PADR
```

This code would give an error message if you try to load it because the first two lines have no line numbers.

By using **LOAD "RECORD"** @ the file can be loaded.

```
10            MOV         E,A
20            MVI         D,0
30   30       MVI         C,':'
40   40       CALL        PO
50   50       MOV         A,E
60   60       XRA         A
70   70       CALL        PADR
```

The first two lines can now be modified with the Editor functions such that they have the line numbers 10 and 20. The Source code is restored onto the mass storage device with,

**STORE "RECORD" @**

It is then restored without the first column of line numbers and with the first two line numbers corrected. The Source code may now be loaded into the Editor by using the ordinary LOAD statement as the line numbers in the file are now correct.

The procedure described in the above section is not necessary if the Source code has an understandable line number in each line. These line numbers need not necessarily even be in numerical order. Provided they exist and are understandable, the LOAD statement without the @ will replace the faulty line numbers by correctly sequenced ones.

# Macro Examples

## Macro Redefinition

After it has been defined, a Macro may be redefined to have the same name but a different body.

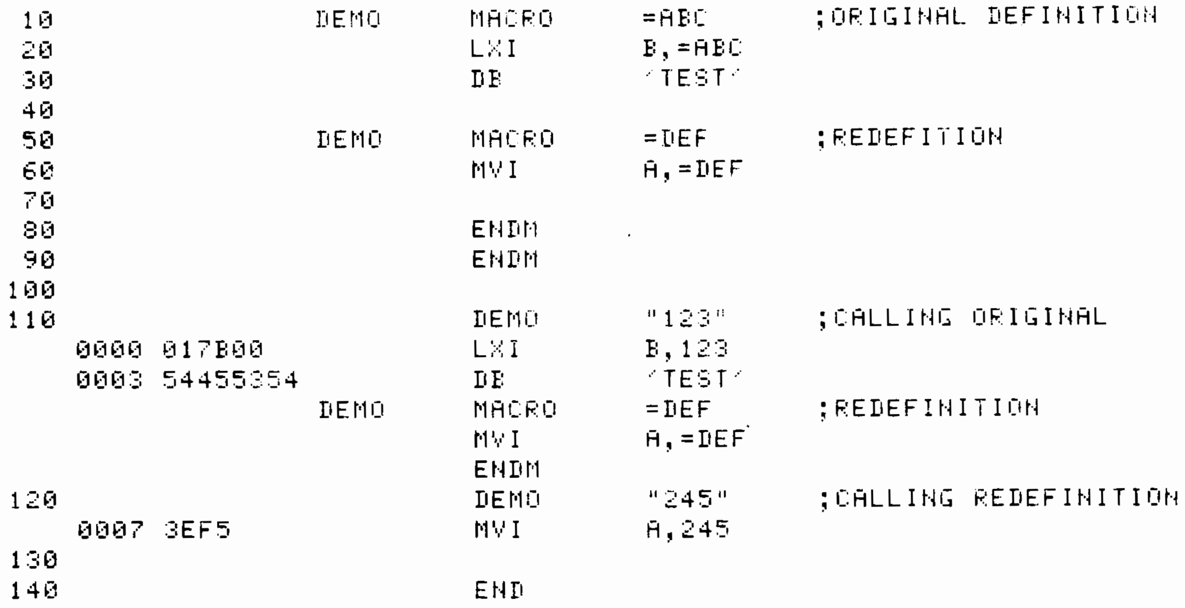
The Macro definitions should all be at the beginning of your Source code or at least before they are called. To have a redefined Macro in this list does not achieve anything as the least definition is the one that is used. The original definition will never be used.

A Macro redefinition is therefore normally put into the body of another Macro. This could be either the Macro defined with the same name or some other. Alternatively. a SOURCE or INCLUDE statement can be inserted into the body of a Macro such that the redefinition is loaded for the mass storage device. The redefinition is carried out when the Macro is called.

The following Source code is an example of redefining a Macro.

```
 10 DEMO      MACRO     =ABC       ;ORIGINAL DEFINITION
 20           LXI       B,=ABC
 30           DB        'TEST'
 40
 50 DEMO      MACRO     =DEF       ;REDEFITION
 60           MVI       A,=DEF
 70
 80           ENDM
 90           ENDM
100
110           DEMO      "123"      ;CALLING ORIGINAL
120           DEMO      "245"      ;CALLING REDEFINITION
130
140           END
```

The following is the result of assembling the Source code.



```
 10                     DEMO      MACRO     =ABC       ;ORIGINAL DEFINITION
 20                               LXI       B,=ABC
 30                               DB        'TEST'
 40
 50                     DEMO      MACRO     =DEF       ;REDEFITION
 60                               MVI       A,=DEF
 70
 80                               ENDM
 90                               ENDM
100
110                               DEMO      "123"      ;CALLING ORIGINAL
    0000 017B00                   LXI       B,123
    0003 54455354                 DB        'TEST'
                        DEMO      MACRO     =DEF       ;REDEFINITION
                                  MVI       A,=DEF
                                  ENDM
120                               DEMO      "245"      ;CALLING REDEFINITION
    0007 3EF5                     MVI       A,245
130
140                               END
```

The Macro called DEMO is first defined in line 10 and then redefined in line 50.

From line 10 to line 90 the assembler program lists the Macro definitions while, at the same time, stores them in memory.

In line 110 the Macro is called with a value of 123 to be passed into the dummy parameter = ABC. The Assembler listing then shows the expansion of the macro with this value.

The Assembler now recognises the new definition for DEMO and lists this. DEMO is now called again, in line 120, with a value of 245 for the dummy parameter = DEF. The listing then shows the expansion of the redefined Macro with this value.

# Macro Nesting

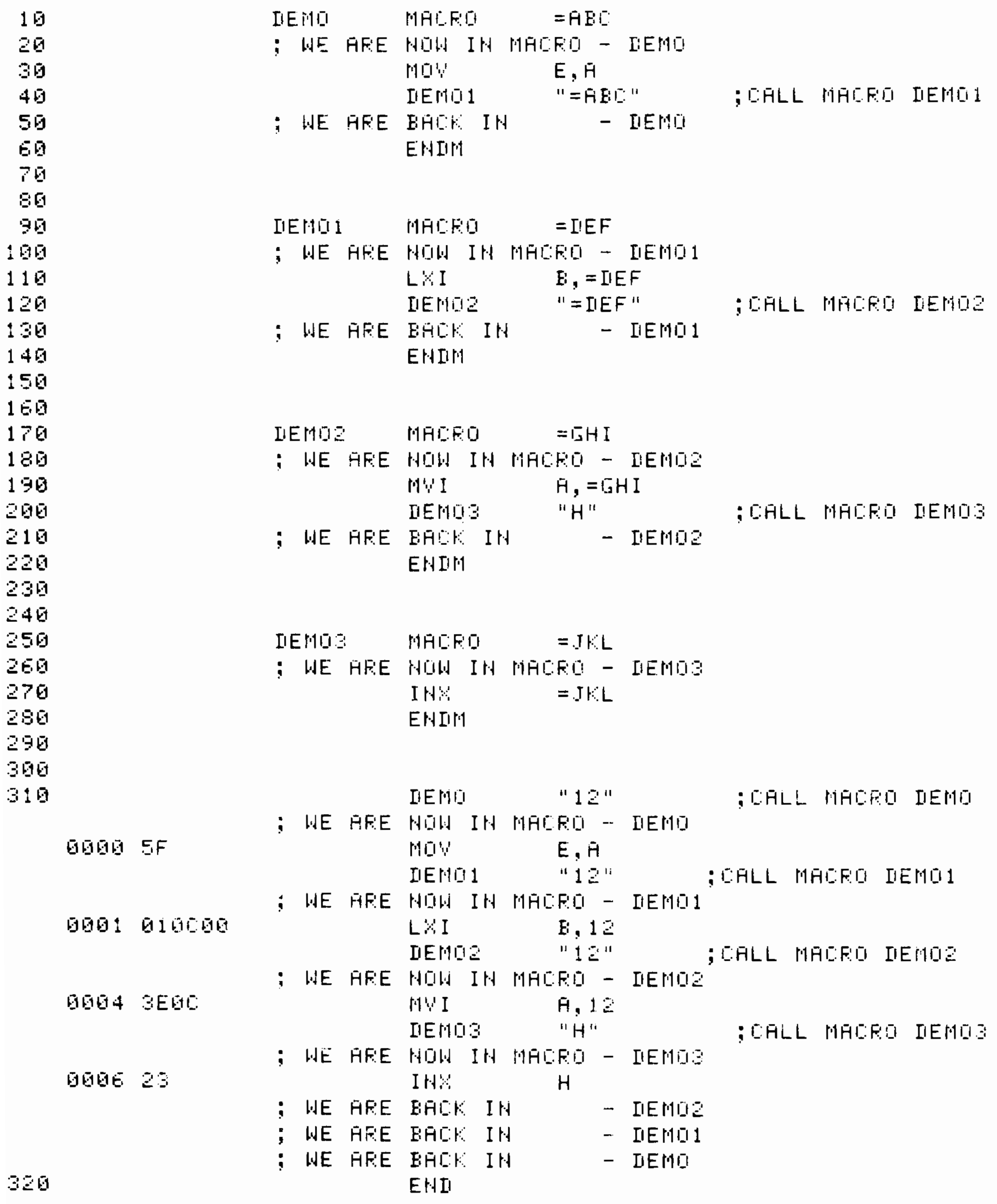
When a Macro is called from the main Source code, the Macro body is expanded and inserted into the main Source code at the point where it was called.

It is possible, in the Macro body, to have a call to another Macro and in this macro a call to yet another one, and so on. This process is called Macro nesting and can be repeated (nested) up to 8 times.

The following Source code is an example of Macros nested 4 deep.

```
 10 DEMO     MACRO     =ABC
 20 ; WE ARE NOW IN MACRO - DEMO
 30          MOV       E,A
 40          DEMO1     "=ABC"      ;CALL MACRO DEMO1
 50 ; WE ARE BACK IN      - DEMO
 60          ENDM
 70
 80
 90 DEMO1    MACRO     =DEF
100 ; WE ARE NOW IN MACRO - DEMO1
110          LXI       B,=DEF
120          DEMO2     "=DEF"      ;CALL MACRO DEMO2
130 ; WE ARE BACK IN      - DEMO1
140          ENDM
150
160
170 DEMO2    MACRO     =GHI
180 ; WE ARE NOW IN MACRO - DEMO2
190          MVI       A,=GHI
200          DEMO3     "H"         ;CALL MACRO DEMO3
210 ; WE ARE BACK IN      - DEMO2
220          ENDM
230
240
250 DEMO3    MACRO     =JKL
260 ; WE ARE NOW IN MACRO - DEMO3
270          INX       =JKL
280          ENDM
290
300
310          DEMO      "12"        ;CALL MACRO DEMO
320          END
```

The following is the result of assembling this Source code.



```
 10                       DEMO      MACRO     =ABC
 20                       ; WE ARE  NOW IN MACRO - DEMO
 30                                 MOV       E,A
 40                                 DEMO1     "=ABC"      ;CALL MACRO DEMO1
 50                       ; WE ARE  BACK IN      - DEMO
 60                                 ENDM
 70
 80
 90                       DEMO1     MACRO     =DEF
100                       ; WE ARE  NOW IN MACRO - DEMO1
110                                 LXI       B,=DEF
120                                 DEMO2     "=DEF"      ;CALL MACRO DEMO2
130                       ; WE ARE  BACK IN      - DEMO1
140                                 ENDM
150
160
170                       DEMO2     MACRO     =GHI
180                       ; WE ARE  NOW IN MACRO - DEMO2
190                                 MVI       A,=GHI
200                                 DEMO3     "H"         ;CALL MACRO DEMO3
210                       ; WE ARE  BACK IN      - DEMO2
220                                 ENDM
230
240
250                       DEMO3     MACRO     =JKL
260                       ; WE ARE  NOW IN MACRO - DEMO3
270                                 INX       =JKL
280                                 ENDM
290
300
310                                 DEMO      "12"        ;CALL MACRO DEMO
                          ; WE ARE  NOW IN MACRO - DEMO
     0000 5F                        MOV       E,A
                                    DEMO1     "12"      ;CALL MACRO DEMO1
                          ; WE ARE  NOW IN MACRO - DEMO1
     0001 010C00                    LXI       B,12
                                    DEMO2     "12"      ;CALL MACRO DEMO2
                          ; WE ARE  NOW IN MACRO - DEMO2
     0004 3E0C                      MVI       A,12
                                    DEMO3     "H"         ;CALL MACRO DEMO3
                          ; WE ARE  NOW IN MACRO - DEMO3
     0006 23                        INX       H
                          ; WE ARE  BACK IN      - DEMO2
                          ; WE ARE  BACK IN      - DEMO1
                          ; WE ARE  BACK IN      - DEMO
320                                 END
```

First of all, the assembler lists the Macros. This could have been suppressed by putting the NOLIST control before line 10 and the LIST control in line 290. The first Macro called DEMO is called, replacing = ABC by 12.

In line 40, the Macro DEMO calls the Macro DEMO1 passing the = ABC for = DEF as 12 was substituted for = DEF.

Line 120, of the second macro, calls DEMO2 (the 2nd macro) substituting = DEF, which is still equal to 12, for = GHI. This means that = GHI also becomes 12.

The last call is in line 200 where the macro DEMO3 is called and substitutes H for = JKL.

The ENDM statement in line 280 terminates the expansion of DEMO3 and the Assembler returns to expanding DEMO2 where an ENDM statement is also encountered (line 220). This terminates the expansion of DEMO2 and the Assembler returns to DEMO1. This too has an ENDM statement and the assembler returns in a similiar fashion to DEMO. Here the ENDM statement returns the Assembler back to the main program in line 320, where the assembly finishes.

## Calling Macros from a Mass Storage Device

As well as being put at the beginning of the Source code, a Macro can also be stored on a mass storage device. For example, the left-hand tape drive (: T14) of the HP 9845B Desktop Computer or an HP 9885M Flexible Disc Drive.

They can then be called into the main program by use of the Assembler Controls.

```
  OPT    SOURCE "<FILE NAME> [msus]"  – 6800
$ INCLUDE "<FILE NAME> [msus]"        – 8080/85
* INCLUDE " <FILE NAME>  [msus]"      – Z80
```

There are two different ways of doing this.

METHOD 1

The main Source code has the statements:

```
<macro name>   MACRO    <dummy parameters> ........
               OPT    SOURCE "<FILE NAME>"
               ENDM
               .
               .
               .
               <macro name>       ; MACRO CALL
               .
               .
```

In this case, on assembling, the first three lines of Source code (the Macro definitions) are stored in the Desktop Computer memory. However, the Macro body, which is stored on the mass storage device under <FILE NAME> , is left where it is. On calling the Macro from the main Source code, the OPT SOURCE statement has an effect for the first time and the Macro body is inserted into the main source code.

METHOD 2

The main Source code has the statements

```
                    OPT SOURCE "<FILE NAME>"
                    .
                    .
                    .
                    <macro name>     ; MACRO CALL
                    .
                    .
```

and the Macro stored on the mass storage device under <FILE NAME> contains the statements

```
   <macro name>     MACRO
                    .
                    .
                    . <macro body>
                    .
                    .
                    ENDM
```

In this case, on running the Assembler, the OPT SOURCE statement will load the entire Macro into the Desktop Computer memory together with the MACRO and ENDM statements. On reaching the Macro call statement the Macro will then be inserted into the main Source code.

## General

Both these methods achieve the same results except that Method1 will allow you, in general, to assemble programs with more Macros lines than Method 2 because it uses less memory. On the other hand, Method 2 will assemble faster if the same Macro is used several times. This is because the Macro code, already in memory, does not need to be loaded from your mass storage device every time it is called.

## Examples

### Method 1

The following Source code is on the mass storage device with the name SUBMAC

```
10          MOV       E,A
20          MVI       D,0
30          MVI       C,':'
40          XRA       A
```

The following is the main Source code

```
10 DEMO     MACRO
20 $ INCLUDE "SUBMAC"
30          ENDM
40
50
60
70          DEMO      ; MACRO CALL STATEMENT
80          END
```

As a result of assembling the above program we have:

```
10                DEMO     MACRO
20                $ INCLUDE "SUBMAC"
30                         ENDM
40
50
60
70                         DEMO      ; MACRO CALL STATEMENT
                  $ INCLUDE "SUBMAC"
   0000 5F                 MOV       E,A
   0001 1600               MVI       D,0
   0003 0E3A               MVI       C,':'
   0005 AF                 XRA       A
80                         END
```

### Method 2

The following is on the mass storage device with the name SUBMAC.

```
10 DEMO     MACRO
20          MOV       E,A
30          MVI       D,0
40          MVI       C,':'
50          XRA       A
60          ENDM
```

The following is the main Source code.

```
10 $ INCLUDE "SUBMAC"
20
30
40
50           DEMO       ; MACRO CALL STATEMENT
60           END
```

As a result of assembling the above program we have.

```
10                $ INCLUDE "SUBMAC"
10                DEMO     MACRO
20                         MOV      E,A
30                         MVI      D,0
40                         MVI      C,':'
50                         XRA      A
60                         ENDM
20
30
40
50                         DEMO     ; MACRO CALL STATEMENT
   0000 5F                 MOV      E,A
   0001 1600               MVI      D,0
   0003 0E3A               MVI      C,':'
   0005 AF                 XRA      A
70                         END
```

# ENDM Statements

As described at the beginning of Section 9 (Macros) the Macro Assembler program ceases the assembly process as soon as it encounters a MACRO statement. It puts the Source lines that follow the MACRO statement into the Macro storage area in the Desktop Computer memory. This process continues until the Macro assembler encounters an ENDM statement.

If you make certain that the ENDM statement is in the same file as the MACRO statement then you will have no problems.

Using Method 1, try the following.

| Macro body on the mass storage device | Main Source code |
|---|---|
| (macro body) | <macro name> **MACRO** |
| . | **OPT SOURCE "<filename>"** |
| . | . |
| . | . |
| **ENDM** | **MACRO CALL** |

The above will not work because the Assembler program will start to assemble the main Source code, will see the MACRO statement, and then will store everything after it into the Macro storage area.

The OPT SOURCE statement will be stored and not executed, and hence, the Macro body with the ENDM statement will not be loaded.

Consequently the Assembler will not see the ENDM statement and will store the Macro call plus the rest of the main Source code into the Macro storage area.

Using method 2, try the following.

| Macro body on the Mass storage device | Main Source code |
|---|---|
| <Macro name> **MACRO** | **OPT SOURCE "<file name>"** |
| . | **ENDM** |
| . | . |
| . (macro body) | . |
| . | . |
| . | . |
| . | **MACRO CALL** |
| | . |
| | . (main Source code) |

This example will work, as the OPT SOURCE statement will load the file from the mass storage device into the Desktop Computer and the Assembler will encounter the MACRO statement.

The entire Macro body will then be stored in the Desktop Computer Macro storage area until the end of the file on the mass storage device is reached.

The Assembler will now encounter the ENDM statement, will cease storing the Macro body and will continue assembling the Source code.

NOTE

The Macro body stored on the mass storage device should not contain an END statement.

## General

The above section on the ENDM statement, used together with Macro bodies stored on mass storage files, illustrates what points should be considered.

A good rule to follow in order to avoid problems is:

THE MACRO AND ENDM STATEMENTS SHOULD BE IN THE SAME FILE

HEWLETT
PACKARD

# SALES & SERVICE OFFICES

## AFRICA, ASIA, AUSTRALIA

**ANGOLA**
Telectra
Empresa Técnica de
Equipamentos
Eléctricos, S.A.R.L.
R. Barbosa Rodrigues, 42-I°DT.°
Caixa Postal, 6487
**Luanda**
Tel: 35515/6
Cable: TELECTRA Luanda

**AUSTRALIA**
Hewlett-Packard Australia
Pty. Ltd.
31-41 Joseph Street
**Blackburn**, Victoria 3130
P.O. Box 36
**Doncaster East**, Victoria 3109
Tel: 896351
Telex: 31-024
Cable: HEWPARO Melbourne

Hewlett-Packard Australia
Pty. Ltd.
31 Bridge Street
**Pymble**
New South Wales, 2073
Tel: 4496566
Telex: 21561
Cable: HEWPARO Sydney

Hewlett-Packard Australia
Pty. Ltd.
153 Greenhill Road
**Parkside**, S.A., 5063
Tel: 2725911
Telex: 82536
Cable: HEWPARD Adelaide

Hewlett-Packard Australia
Pty.Ltd.
141 Stirling Highway
**Nedlands**, W.A. 6009
Tel: 3865455
Telex: 93859
Cable: HEWPARO Perth

Hewlett-Packard Australia
Pty. Ltd.
121 Wollongong Street
**Fyshwick**, A.C.T. 2609
Tel: 804244
Telex: 62650
Cable: HEWPARO Canberra

Hewlett Packard Australia
Pty. Ltd.
5th Floor
Teachers Union Building
495-499 Boundary Street
**Spring Hill**, Queensland 4000
Tel: 2291544
Cable: HEWPARD Brisbane

**BANGLADESH**
The General Electric Co.
of Bangladesh Ltd.
Magnet House 72
Oilkusha Commercial Area
**Motijheel**, Oacca 2
Tel: 252415, 252419
Telex: 734
Cable: GECDAC Oacca

**ETHIOPIA**
Abdella Abdulmalik
P.O. Box 2635
**Addis Ababa**
Tel: 11 93 40

**GUAM**
Medical Only
Guam Medical Supply, Inc.
Suite C, Airport Plaza
P.O. Box 8947
**Tamuning** 96911
Tel: 646-4513
Cable: EARMED Guam

**HONG KONG**
Schmidt & Co. (Hong Kong) Ltd.
Wing On Centre, 26th Floor
Connaught Road, C.
**Hong Kong**
Tel: 5-455644
Telex: 74766 SCHMC HX

**INDIA**
Blue Star Ltd.
Kasturi Buildings
Jamshedji Tata Rd
**Bombay** 400 020
Tel: 29 50 21
Telex: 011-2156
Cable: BLUEFROST

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
**Bombay** 400 025
Tel: 45 78 87
Telex: 011-4093
Cable: FROSTBLUE

Blue Star Ltd.
Band Box House
Prabhadevi
**Bombay** 400 025
Tel: 45 73 01
Telex: 011-3751
Cable: BLUESTAR

Blue Star Ltd.
Bhavdeep
Stadium Road
**Ahmedabad** 380 014
Tel: 42880
Telex: 234
Cable: BLUEFROST

Blue Star Ltd.
7 Hara Street
P.O. Box 506
**Calcutta** 700 001
Tel: 23-0131
Telex: 021-7655
Cable: BLUESTAR

Blue Star Ltd.
Bhandari House
7th & 8th Floor
91 Nehru Place
**New Delhi** 110 024
Tel: 634770 & 635166
Telex: 031-2463
Cabla: BLUESTAR

Blue Star Ltd.
Blue Star House
11/11A Magarath Road
**Bangalore** 560 025
Tel: 55668
Telex: 043-430
Cable: BLUESTAR

Blue Star Ltd.
Meeakshi Mandiram
xxx/1678 Mahatma Gandhi Rd.
**Cochin** 682 016
Tel: 32069,32161,32282
Telex: 0885-514
Cable: BLUESTAR

Blue Star Ltd.
1-1-117/1
Sarojini Oevi Road
**Secunderabad** 500 003
Tel: 70126, 70127
Telex: 015-459
Cable: BLUEFROST

Blue Star Ltd.
2/34 Kodambakkam High Road
**Madras** 600 034
Tel: 82056
Telex: 041-379
Cable: BLUESTAR

**INDONESIA**
BERCA Indonesia P.T.
P.O. Box 496/Jkt.
Jln.Abdul Muis 62
**Jakarta**
Tel: 349255, 349886
Telex: 46748 BERSIL IA
Cable: BERSAL

BERCA Indonesia P.T.
P.O. Box 174/Sby.
23 Jln. Jimerto
**Surabaya**
Tel: 42027
Cable: BErcacon

**ISRAEL**
Electronics Engineering Oiv.
of Motorola Israel Ltd.
16, Kremenelski Street
P.O.Box 25016
**Tel-Aviv**
Tel: 38973
Telex: 33569, 34164
Cable: BASTEL Tel-Aviv

**JAPAN**
Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg., 4th Floor
4-20, Nishinakajima 5-chome
Yodogawa-ku, Osaka-shi
**Osaka**,532
Tel: 06-304-6021
Telex: 523-3624

Yokogawa-Hewlett-Packard Ltd.
29-21, Takaido-Higashi 3-chome
Suginami-ku, **Tokyo 166**
Tel: 03-331-6111
Telex: 232-2024 YHP-Tokyo
Cable: YHPMARKET TOK 23 724

Yokogawa-Hewlett-Packard Ltd.
Nakamo Building
24 Kami Sasajima-cho
Nakamura-ku, **Nagoya**, 450
Tel: 052 571-5171

Yokogawa-Hewlett-Packard Ltd.
Tanigawa Building
2-24-1 Tsuruya-cho
Kanagawa-ku
**Yokohama**, 221
Tel: 045-312-1252
Telex: 382-3204 YHP YOK

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Building
105, 1-chome, San-no-maru
**Mito**, Ibaragi 310
Tel: 0292-25-7470

Yokogawa-Hewlett-Packard Ltd.
Inoue Building
1348-3, Asahi-cho, 1-chome
**Atsugi**, Kanagawa 243
Tel: 0462-24-0452

Yokogawa-Hewlett-Packard Ltd.
Kumagaya Asahi
Hachijuni Building
4th Floor
3-4, Tsukuba
**Kumagaya**, Saitama 360
Tel: 0485-24-6563

**KENYA**
Advanced Communications Ltd.
P.O. Box 30070
**Nairobi**
Tel: 331955
Telex: 22639

Medical Only
International Aeradio (E.A.) Ltd.
P.O. Box 19012
Nairobi Airport
**Nairobi**
Tel: 336055/56
Telex: 22201/22301
Cable: INTAERIO Nairobi

Medical Only
International Aeradio (E.A.) Ltd.
P.O. Box 95221
**Mombasa**

**KOREA**
Samsung Electronics Co., Ltd.
15th Floor, Daeyongak Bldg.,
25-5, 1-KA
Choong Moo-Ro, Chung-Ku,
**Seoul**
Tel: (23) 6811, 778-3401/2/3/4
Telex: 2257S

**MALAYSIA**
Hewlett-Packard Sales SON BHO
Suite 2.21/2.22
Bangunan Angkasa Raya
Jalan Ampang
**Kuala Lumper**
Tel: 23320/27491

Protel Engineering
P.O. Box 1917
Lot 259, Satok Road
Kuching, **Sarawak**
Tel: 53544
Cable: PROTELENG

**MOZAMBIQUE**
A.N. Goncalves, Ltd.
162, 1° Apt. 14 Av. O. Luis
Caixa Postal 107
**Maputo**
Tel: 27091, 27114
Telex: 6-203 NEGON Mo
Cable: NEGON

**NEW GUINEA**
Hewlett-Packard Australia
Pty. Ltd.
Development Bank Building
Ground Floor
Ward Strip
**Port Moresby**, Paupua
Tel: 258933

**NEW ZEALAND**
Hewlett-Packard (N.Z.) Ltd.
4-12 Cruickshank Street
Kilbirnie, Wellington 3
P.O. Box 9443
Courtney Place
**Wellington**
Tel: 877-199
Cable: HEWPACK Wellington

Hewlett-Packard (N.Z.) Ltd.
Pakuranga Professional Centre
267 Pakuranga Highway
Box 51092
**Pakuranga**
Tel: 569-651
Cable: HEWPACK Auckland

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
Scientific Division
79 Carlton Gore Road, Newmarket
P.O. Box 1234
**Auckland**
Tel: 75-289
Cable: DENTAL Auckland

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
Norrie and Parumoana Streets
**Porirua**
Tel. 75-098
Telex: 3858

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
P.O. Box 309
239 Stanmore Road
**Christchurch**
Tel: 892-019
Cable. OENTAL Christchurch

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
303 Great King Street
P.O. Box 233
**Dunedin**
Tel: 88-817
Cable: DENTAL Ounedin

**NIGERIA**
The Electronics
Instrumentations Ltd.
N68/770 Oyo Road
Oluseun House
P.M.B. 5402
**Ibadan**
Tel: 461577
Telex: 31231 TEIL NG
Cable: THETIEL Ibadan

The Electronics Instrumentations Ltd.
144 Agege Motor Road, Mushin
P.O. Box 6645
**Lagos**
Cable: THETEIL Lagos

**PAKISTAN**
Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
**Karachi-3**
Tel: 511027, 512927
Telex: 2894
Cable: COOPERATOR Karachi

Mushko & Company, Ltd.
388, Satellite Town
**Rawalpindi**
Tel: 41924
Cable: FEMUS Rawalpindi

**PHILIPPINES**
The Online Advanced
Systems Corporation
Rico House
Amorsolo cor. Herrera Str.
Legaspi Village, Makati
P.O. Box 1510
Metro **Manila**
Tel: 85-35-81, 85-34-91,85-32-21
Telex: 3274 ONLINE

**RHODESIA**
Field Technical Sales
45 Kelvin Road North
P.O. Box 3458
**Salisbury**
Tel: 705231 (5 lines)
Telex: RH 4122

**SINGAPORE**
Hewlett-Packard Singapore
(Pte.) Ltd.
1150 Oepot Road
Alexandra P.O. Box 58
**Singapore 4**
Tel: 270-2355
Telex: HPSG RS 21486
Cable: HEWPACK, Singapore

**SOUTH AFRICA**
Hewlett-Packard South Africa
(Pty.), Ltd.
Private Bag Wendywood,
Sandton, Transvaal, 2144
Hewlett-Packard Centre
Daphne Street, Wendywood,
**Sandton**, 2144
Tel: 802-1040/6
Telex: 8-4782
Cable: HEWPACK Johannesburg

Hewlett-Packard South Africa
(Pty.), Ltd.
P.O. Box 120
Howard Place, Cape Province, 7450
Pine Park Centre, Forest Orive,
**Pinelands**, Cape Province, 7405
Tel: 53-7955 thu 9
Telex: 57-0006

**SRI LANKA**
Metropolitan Agencies Ltd.
209/9 Union Place
**Colombo 2**
Tel: 35947
Telex: 1377METROLTO CE
Çable: METROLTO

**SUDAN**
Radison Trade
P.O. Box 921
**Khartoum**
Tel: 44048
Telex: 375

**TAIWAN**
Hewlett-Packard Far East Ltd.
Taiwan Branch
39 Chung Hsiao West Road
Section 1, 7th Floor
**Taipei**
Tel: 3819160-9,3141010
Cable: HEWPACK TAIPEI

Hewlett-Packard Far East Ltd.
Taiwan Branch
68-2, Chung Cheng 3rd. Road
**Kaohsiung**
tel: (07) 242316-Kaohsiung

Analytical Only
San Kwang Instruments Co., Ltd.
20 Yung Sui Road
**Taipei**
Tel: 3615446-9 (4 lines)
Telex: 22894 SANKWANG
Cable: SANKWANG Taipei

**TANZANIA**
Medical Only
International Aeradio (E.A.), Ltd.
P.D. Box 861
**Dar es Salaam**
Tel: 21251 Ext. 265
Telex: 41030

**THAILAND**
UNIMESA Co. Ltd.
Elcom Research Building
2538 Sukumvit Ave.
**Bangchak,Bangkok**
Tel: 3932387, 3930338
Cable: UNIMESA Bangkok

**UGANDA**
Medical Only
International Aeradio (E.A.), Ltd.
P.O. Box 2577
**Kampala**
Tel: 54388
Cable: INTAERIO Kampala

**ZAMBIA**
R.J. Tilbury (Zambia) Ltd.
P.O. Box 2792
**Lusaka**
Tel: 73793
Cable: ARJAYTEE, Lusaka

**OTHER AREAS NOT LISTED, CONTACT:**
Hewlett-Packard Intercontinental
3200 Hillview Ave.
Palo Alto, California 94304
Tel: (415) 856-1501
TWX: 910-373-1267
Cable: HEWPACK Palo Alto
Telex: 034-8300, 034-8493

## CANADA

**ALBERTA**
Hewlett-Packard (Canada) Ltd.
11620A - 168th Street
**Edmonton** T5M 3T9
Tel: (403) 452-3670
TWX: 610-831-2431

Hewlett-Packard (Canada) Ltd.
210,7220 Fisher St. S.E.
**Calgary** T2H 2H8
Tel: (403) 253-2713
Twx: 610-821-6141

**BRITISH COLUMBIA**
Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
**Richmond** V6X 2W7
Tel: (604) 270-2277
TWX: 610-925-5059

**MANITOBA**
Hewlett-Packard (Canada) Ltd.
380-550 Century St.
**Winnipeg** R3H 0Y1
Tel: (204) 786-6701
TWX: 610-671-3531

**NOVA SCOTIA**
Hewlett-Packard (Canada) Ltd.
800 Windmill Road
**Dartmouth** B3B 1L1
Tel: (902) 469-7820
TWX: 610-271-4482

**ONTARIO**
Hewlett-Packard (Canada) Ltd.
1020 Morrison Dr.
**Ottawa** K2H 8K7
Tel: (613) 820-6483
TWX: 610-563-1636

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
**Mississauga** L4V 1M8
Tel: (416) 678-9430
TWX: 610-492-4246

Hewlett-Packard (Canada) Ltd.
552 Newbold Street
**London N6E 2S5**
Tel: (519) 686-9181

**QUEBEC**
Hewlett-Packard (Canada) Ltd.
275 Hymus Blvd.
**Pointe Claire** H9R 1G7
Tel: (514) 697-4232
TWX: 610-422-3022
TLX: 05-821-521 HPCL

**FOR CANADIAN AREAS NOT LISTED:**
Contact Hewlett-Packard (Canada)
Ltd. in Mississauga.

2/79

## CENTRAL AND SOUTH AMERICA

**ARGENTINA**
Hewlett-Packard Argentina
S.A
Av. Leandro N. Alem 822 - 12°
1001 **Buenos Aires**
Tel: 31-6063,4,5,6
Telex: 122443 AR CIGY
Cable: HEWPACKARG

Biotron S.A.C.I.y M.
Bolivar 177
1066 **Buenos Aires**
Tel: 30-4846, 34-9356, 34-0460,
33-2863
Telex: 011-7595
Cable: Biotron Baries

**BOLIVIA**
Casa Kavlin S.A.
Calle Potosi' 1130
P.O. Box 500
**La Paz**
Tel: 41530,53221
Telex: CWC BX 5298,ITT 3560082
Cable: KAVLIN

**BRAZIL**
Hewlett-Packard do Brasil
I.e.C. Ltda.
Alameda Rio Negro, 750
Alphaville
06400 **Barueri** SP
Tel: 429-3222
Cable: HEWPACK Sao Paulo

Hewlett-Packard do Brasil
I.e.C. Ltda.
Rua Padre Chagas, 32
90000-**Pôrto Alegre**-RS
Tel: (0512) 22-2998, 22-5621
Cable: HEWPACK Potto Alegre

Hewlett-Packard do Brasil
I.e.C. Ltda.
Av. Epitacio Pessoa, 4664
Lagoa
20000-**Rio de Janeiro**-RJ
Tel:
Telex: 021-21905 HPBR-BR
Cable: HEWPACK
Rio de Janeiro

**CHILE**
Jorge Calcagni y Cia. Ltda
Vicuna MacKenna 3, Ofic. 1204
Casilla 16475
Correo 9, **Santiago**
Tel: 34152
Telex: JCALCAGNI

**COLOMBIA**
Instrumentacidn
Henrik A. Langebaek & Kier S.A.
Carrera 7 No. 48-75
Apartado Adreo 6287
**Bogotá**, I O.E.
Tel: 269-8877
Telex: 44400
Cable: AARIS Bogotd

Instrumentacion
H.A. Langebaek & Kier S.A.
Carrera 63 No. 49-A-31
Apartado 54098
**Medellin**
Tel: 304475

**COSTA RICA**
Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
**San Jose**
Tel: 24-38-20, 24-08-19
Telex: 2367 GALGUR CR
Cable: GALGUR

**ECUADOR**
Computadoras y Equipos
Electrónicos del Ecuador
P.O. Box 6423 CCI
Eloy Alfaro No. 1824,3°Piso
**Quito**
Tel: 453 482
Telex: 2548 CYEDE ED
Cable: CYEDE-Ouito

Medical Only
Hospitalar S.A.
Casilla 3590
Robles 625
**Quito**
Tel: 545-250
Cable: HOSPITALAR-Quito

**EL SALVADOR**
IPESA
Bulevar de los Heroes 11-48
**San Salvador**
Tel: 252787

**GUATEMALA**
IPESA
Avenida Reforma 3-48,
Zona 9
**Guatemala City**
Tel: 316627,314786,66471-5,ext.9
Telex: 4192 Teletro Gu

**MEXICO**
Hewlett-Packard Mexicana,
S.A. de C.V.
Av. Periférico Sur No. 6501
Tepepan, Xochimilco
**Mexico** 23, O.F.
Tel: 905-676-4600
Telex: 017-74-507

Hewlett-Packard Mexicana,
S.A. de C.V.
Ave. Constitución No. 2184
**Monterrey**, N.L.
Tel: 48-71-32, 48-71-84
Telex: 038-410

**NICARAGUA**
Roberto Terán G.
Apartado Postal 689
Edificio Terán
**Managua**
Tel: 25114, 23412,23454,22400
Cable: ROTERAN Managua

**PANAMA**
Electrónico Balboa, S.A.
Aparatado 4929
Panama 5
Calle Samuel Lewis
Edificio "Alfa", No.2
**Cuidad de Panama**
Tel: 64-2700
Telex: 3483103 Curundu,
Canal Zone
Cable: ELECTRON Panama

**PERU**
Compañía Electro Médica S.A.
Los Flamencos 145
San Isidro Casilla 1030
**Lima 1**
Tel: 41-4325
Telex: Pub. Booth 25424 SISIORO
Cable: ELMED Lima

**SURINAME**
Surtel Radio Holland N.V.
Grote Hofstr. 3-5
P.O. Box 155
**Paramaribo**
Tel: 72118, 77880
Cable: Surtel

**TRINIDAD & TOBAGO**
CARTEL
Caribbean Telecoms Ltd.
P.O. Box 732
69 Frederick Street
**Port-of-Spain**
Tel: 62-53068

**URUGUAY**
Pablo Ferrando S.A.C.el.
Avenida Italia 2877
Casilla de Correo 370
**Montevideo**
Tel: 40-3102
Telex: 702 Public Booth Para
Pablo Ferrando
Cable: RADIUM Montevideo

**VENEZUELA**
Hewlett-Packard de Venezuele
C.A.
P.O. Box 50933
Caracas 105
Los Ruices Norte
3a Transversal
Edificio Sagre
**Caracas** 107
Tel: 239-4133 (20 lines)
Telex: 25146 HEWPACK
Cable: HEWPACK Caracas

**FOR AREAS NOT LISTED, CONTACT:**
Hewlett-Packard
Inter-Americas
3200 Hillview Ave.
**Palo Alto**, California 94304
Tel: (415) 856-1501
TWX: 910-373-1260
Cable: HEWPACK Palo Alto
Telex: 034-8300, 034-8493