1979

A software development system for the Motorola M6800

John Michael Moore

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation


This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
A SOFTWARE DEVELOPMENT SYSTEM FOR THE MOTOROLA M6800

by

JOHN MICHAEL MOORE

A Thesis Submitted

in

Partial Fulfillment

of the

Requirements for the Degree of

MASTER OF SCIENCE

in

Electrical Engineering

Approved by:

Prof. E. R. Salem
( Thesis Advisor )

Prof. Roy Czernikowski

Prof. George A. Brown

Prof.
( Department Head )

DEPARTMENT OF ELECTRICAL ENGINEERING

COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

JUNE, 1979
TABLE OF CONTENTS

INTRODUCTION 1

CHAPTER 1 DESIGN PROCEDURE 2

CHAPTER 2 INITIALIZING THE OPERATING SYSTEM

2.1 LOADING THE OPERATING SYSTEM 4
2.2 ENTERING THE OPERATING SYSTEM 4
2.3 CONSOLE INPUT/OUTPUT 5
2.4 PRINTER OUTPUT 5

CHAPTER 3 OPERATING SYSTEM COMMANDS

3.1 COMMAND SUMMARY 6
3.2 DETAILED COMMAND DESCRIPTION 7
3.3 USER DEFINED COMMANDS 15

CHAPTER 4 SOURCE PROGRAM DEVELOPMENT

4.1 ENTERING A SOURCE PROGRAM 16
4.2 USING THE TAPE SYSTEM FOR SAVING SOURCE TAPES 17

CHAPTER 5 PROGRAM ASSEMBLY

5.1 USING THE ASSEMBLER 19
5.2 OUTPUT FROM THE ASSEMBLER 20

CHAPTER 6 USING THE DEBUGGER PACKAGE 22

CHAPTER 7 DIFFERENCES BETWEEN THIS ASSEMBLER
AND THE MOTOROLA CROSS ASSEMBLER 24

CHAPTER 8 OPERATING SYSTEM MESSAGES

8.1 ASSEMBLER ERRORS 26
8.2 EXPLANATION OF OPERATING SYSTEM MESSAGES 26

DESIGN DOCUMENT APPENDIX
ABSTRACT

The goal of this project was to produce a useable software development system for the Digital Group M6800 microcomputer. The design was to have implemented the command language of the Digital Group Z-80 operating system and have the capability of assembling source programs written using the Motorola M6800 assembly language mnemonics. The program developed to fulfill these requirements was assembled using the Motorola M6800 cross assembler and occupies approximately 7K bytes of memory. The functions provided include an editor package for the entry and modification of source programs, an assembler which meets the design goal and a debugger package to facilitate the checkout and execution of the programs which will be developed using this system.
INTRODUCTION

This document describes the use of a software development system for the Motorola M6800 microprocessor. A Digital Group microcomputer which houses the processor provides 26K bytes of programmable memory, a video display/keyboard console, printer, and cassette tape storage system. Facilities are provided in memory resident software for entry and storage of assembly language programs as well as an assembler and debug package for conversion to and test of object code. The assembler supports the standard Motorola assembly language and a subset of the Motorola M6800 Cross Assembler directives. The command language is designed to be similar in use to an operating system provided for the Digital Group Z-80 processor which is also supplied with the computer.

Two levels of operation exist in the operating system. Level 1 of the system is entered upon completion of the ROM based bootstrap loader which loads the system into memory. This is the basic Digital Group operating system which was built upon and expanded to the total software development system. In the level 1 mode of operation, a menu is displayed showing the available system commands. Functions are initiated by entering the number which is associated with the position of the command in the menu. The operation of this portion of the system is described in the Digital Group system documentation. Program 7 on the menu is a loader for cassette tapes written in the Motorola software standard. This loader reads object tape files into the system at 300 bits/sec from an external tape device connected to input port 2. Program 8 initiates the Editor/Assembler which is level 2 of the operating system. In level 2 of the system commands are initiated by entering a four character command name followed by a list of parameters. The purpose of this report is to provide a reference to the effective use of the level 2 command set.
CHAPTER 1 DESIGN PROCEDURE

The software development system for the Digital Group M6800 microcomputer was designed in a modular fashion. This was accomplished by defining the system using a structured English description of each function and then iteratively refining that description into final form. The structured English description has been formatted using a text processor known as PDL. PDL stands for program design language and is marketed by Caine Farber and Gordon of California. The PDL listing is contained in Appendix A. This processor accepts plain text as input and produces a formatted listing. The PDL description is equivalent to a flow chart with the exception that it cannot conveniently represent constructs which violate the rules of structured programming. It has the additional advantage that it is machine processable and unlike a conventional flow-chart may conveniently be updated to reflect the current software approach as a program is developed. For this project the PDL description was used not just for algorithm documentation but as a vehicle to describe the system during the program design phase.

Each program module has one entry point and one exit point. This module design goal was carried to sections of inline code as far as possible. Implementing the design in PDL prior to coding facilitated using only the structures of simple alternation and iteration as well as sections of sequential operations. The operations of alternation and iteration are shown in the PDL listing by various levels of indentation. The indentation of statements is controlled by the use of keywords to identify alternation and iteration constructs. The alternation construct is implemented using the keywords IF - ELSE - ENDF. The else section is optional. The iteration construct is implemented using the keywords DO - ENDO. The remaining text in the listing is intended to provide a high level understanding of the program module operation.
Control statements are used to identify modules within the system. These control statements are not printed on the listing but serve to identify the module name which is shown at the top of each page. The module names are identified in the table of contents for the design document. In addition to listing the module in the table of contents the processor identifies each use of the module name within another module. This is shown by an entry in the REFERENCE PAGE field of the listing line where the reference was made. This facility was used during the design to develop the approach in a top-down manner. Each module function was identified in a general fashion. Then, where similar functions were identified or segmentation of the module was necessary due to its size, the statements of a module were further explained as a sub-module.

In order to follow the design using the PDL, listing the block entitled "PROCESS THE NEXT CONSOLE COMMAND" should be used as the starting point. Each entry in the REFERENCE PAGE field provides the page number where the function shown on that line is explained further. By successively following the reference page trail, the design can be understood from the top level down to the level from which the code was generated for each module.
CHAPTER 2 INITIALIZING THE OPERATING SYSTEM

2.1 LOADING THE OPERATING SYSTEM

The operating system is loaded from the tape cassette interface by starting the tape recorder at the load point of the tape and depressing the front panel reset button at the start of the low frequency tone. The level 1 system menu will be displayed on the video display when the tape load is complete.

2.2 ENTERING THE OPERATING SYSTEM

Level 2 of the operating system is entered by selecting program 8 from the level 1 menu. A sign-on message "M6800 Assembler / Editor" is provided to verify entry into level 2. When level two is entered for the first time during a programming session the NEWF command must be used to initialize the system memory to a known state. After this initial entry, the NEWF command must be used only when necessary to completely restart the system since it erases the entire system memory. The commands available in the operating system are summarized in the command summary and are further described in the detailed command description section. Each command consists of four letters followed by a list of parameters such as start address, line number etc. Entries in the parameter list are separated from the command and each other by one or more spaces. The tab key may also be used for this purpose because it is interpreted by the system as a series of spaces. Limited checking is performed by the system of the validity of each entry. If a command is entered which is unknown to the system, the error message WHAT? will be returned on the console. Entry of a command which is to modify the current text file will return the error message BAD FILE if the system does not have an active file. When a valid command is recognized by the system it is executed and the system returns for another command.
2.3 CONSOLE INPUT/OUTPUT

Commands and programs are entered at the console. This device consists of the keyboard and video display. The console driver software is written to emulate a conventional CRT terminal. As lines are entered into the system, the video output device scrolls upward and the new line is displayed on the bottom of the screen. The keyboard is the input device and must be operated in uppercase mode. Input from the keyboard is buffered prior to being decoded by the system. This allows correction of errors in the command line prior to termination of the command by a carriage return. Striking the DEL key removes characters from the command buffer. As the characters are removed they are echoed back to the console display.

Additional special controls have been implemented in the input system. Typing a CTRL - L clears the video screen, CTRL - X cancels the current input line, CTRL - I tabs the display, and CTRL - @ cancels the auto line numbering mode. The tab stops are fixed at intervals of 8 spaces.

2.4 PRINTER OUTPUT

The PRON and PROF commands enable and disable the system printer. When the printer is enabled, output to the console is copied to the printer. With the printer enabled, the system runs slower due to software delays incorporated into the printer driver. This delay operates whether or not the printer is turned on. It can be used in this way solely to slow down the scrolling rate of the console.
CHAPTER 3 OPERATING SYSTEM COMMANDS

3.1 COMMAND SUMMARY

ALTR  displays and modifies memory

ASSM  initiates assembly of current file.

AUTO  initiates the auto line number mode

BREK  inserts a breakpoint in memory

NEWF  initiates a system restart and clears non-system memory

DELT  deletes a range of lines in the current file

DUMP  displays a block of memory

EXEC  starts execution of target binary program

EXIT  transfers control to the level 1 operating system

FILE  initiates file manager operation

LIST  lists the current file

LOAD  loads a file into memory from cassette tape

MOVE  moves a block of memory

PROF  terminates printer operation
PRON causes a copy of the console output to be listed on the printer

RDTP reads a Digital Group standard binary tape into memory

REST restores memory from breakpoint

RSEQ renumbers the current file

SAVE stores a file onto the tape device

SYMB lists the symbol table on the console device

WRTP writes a digital group standard binary tape from memory

3.2 DETAILED COMMAND DESCRIPTION

This section gives a detailed description of the command mode functions. For each entry the command name is shown with the required parameters. For commands where a sequence is required for use, the steps are shown listed as 1,2,3... Commands with similar sequence numbers are alternatives at the same step. This is shown as in the first entry for the ALTR command. The function is initiated by typing the command name ALTR, a four character address followed by a carriage return. The current contents of the address are displayed on the following line. Entry of a two character Hex number loads that value into memory. Entry of a space followed by a carriage return does not modify the displayed memory but displays the contents of the next memory location. These level 2 steps may be repeated as often as desired but entering a carriage return with a null line terminates the command and returns to the command mode. Many of the commands require that parameters be appended to the command. Where a command has optional parameters, a space
or carriage return is shown to emphasize that option (sp. = space bar, ret. = carriage return).

ALTR COMMAND

The alter command is used at the command level to examine and modify memory locations.

1. ALTR HHHH lists the current contents of memory location HHHH

2. VV ret. loads memory location HHHH with the hex value VV

2. sp. ret. space causes the contents of HHHH+1 to be displayed leaving the contents of HHHH unchanged.

3. ret. terminates the alter function.

AUTO COMMAND

The auto line number command prompts the programmer with line numbers for insertion into the current file. Line numbers are generated, starting with the number specified, at an increment of 0001 until the command is canceled or when the number generated is found to be in the file. Entering a carriage return after typing the four character command enters the auto mode starting with line number 0001.

1. AUTO LLLL generates line numbers starting with LLLL

1. AUTO ret. generates line numbers starting with 0001
ASSM COMMAND

The assemble command initiates a symbolic assembly of the current source file. There are two options which may be specified with the ASSM command. Requesting the E option suppresses the generation of the full listing and instead, lists on the console only those lines where errors are found. The L option preserves the contents of the symbol table prior to assembly and allows two assemblies to share a common symbol table.

1. ASSM DDDD OOOO performs a symbolic assembly of the current file and loads the object program at the address specified by DDDD with an origin address at OOOO. The origin specification is optional and is overridden by an ORG statement in the source program.

1. ASSME DDDD OOOO performs the same function as above but only errors are listed

1. ASSML DDDD OOOO assembles using the symbol table from the last assembly

1. ASSMLE DDDD OOOO combination of above

BREK COMMAND

The breakpoint command causes the contents of the break address to be loaded with an SWI instruction. When this instruction is executed, program control is passed to the operating system and the program status is displayed on the console. The program status display has the following format.

CC AA BB XXXX SSSS PPPP
(CC = condition codes, AA = Accumulator A, BB = Accumulator B, XXXX = Index Register, SSSS = Stack Pointer, PPPP = program counter)

After a breakpoint is executed any of the valid system commands may be initiated. In addition to the status display, the program status is saved in memory locations CCRSAV – SPTSAV.

1. BREK HHHH

inserts a breakpoint at memory location HHHH

DELT COMMAND

The delete command deletes lines from the current file. If the two line numbers specified as the deletion range are not in the file, the deletion range defaults to a range which is bounded by the specified range.

1. DELT SSSS EEEE

deletes a range of lines starting at line number SSSS and ending with EEEE

DUMP COMMAND

The dump command displays the contents of memory. The display is formatted in blocks of 8 bytes and is displayed with the start address of the block.

1. DUMP SSSS EEEE

displays a block of memory starting at address SSSS and ending with the address at the end of the next 8 byte boundary above EEEE

ENTER A LINE INTO THE CURRENT FILE

Lines are added to the current file by entering the 4 place decimal line number followed by the text of the line.
1. DDDD text. enters a line with the number DDDD into the current file.

1. DDDD ret. removes line number DDDD from the current file.

EXEC COMMAND

The execute command passes program control to a binary program. This command may be used to execute a program produced by the assembler or loaded from tape.

1. EXEC HHHH passes control to the object program at address HHHH.

EXIT COMMAND

The exit command passes program control to the level 1 operating system and displays its menu.

1. EXIT passes control to the level 1 operating system.

FILE COMMAND

The file manager directs the use of the active source files and in particular, the current source file. It is used to create or delete RAM files and identify the current file.

1. FILE FLNAME AAAAA Adds a file with the one to six character name FLNAME to the list of active files with the start address AAAAA.
1. FILE FLNAME 0000 deletes the file named FLNAME from the list of active files.

1. FILE FLNAMEret makes file FLNAME the current file.

1. FILESret. lists names, start addresses, and end addresses of the active files starting with the current file.

LIST COMMAND

The list command lists the contents of the current source file. The file is listed line by line on the console and a space is inserted between the line number and the start of the line text.

1. LISTret. lists the entire file.

1. LIST SSSS EEEE lists a range of lines starting with line number SSSS and ending with line number EEEE.

LOAD COMMAND

The load command appends a file from the tape device to the end of the current file. The current file can be a null file when the load is done for the initial load or one with a source program already loaded for program merging.

1. LOAD appends a file from the tape device to the current file.
MOVE COMMAND

The move command copies a block of memory from one memory range to another. The ranges specified may overlap because the copy is done from the top up or bottom down depending on the direction in the address space selected.

1. MOVE SSSS EEEE DDDD moves the block of memory starting at address SSSS and ending with EEEE to a block which starts at address DDDD

NEWF COMMAND

The new files command is used for system initialization. This command clears all memory available to the programmer and should be performed after power-up to reset all memory to a known state. Executing the NEWF command destroys all files and should be used with extreme caution after system initialization.

PROF COMMAND

The printer off command terminates the operation of the print device.

1. PROF suppresses the simultaneous listing of the console display

PRON COMMAND

The printer on command turns on the printer driver. The printer driver copies all output directed to the console to the printer and contains a software delay to provide the proper timing for the printer.

1. PRON initiates simultaneous listing of the console display
RDTP COMMAND

The read tape command reads a binary tape into the system. This command may be used for binary tapes only, such as those produced by the WRTP command or the level 1 tape writer.

1. RDTP SSSS EEEE loads a binary tape into the system starting at address SSSS and ending at address EEEE

REST COMMAND

The restore command restores the data displaced by the SWI instruction inserted by the breakpoint command.

1. REST AAAAA restores the data displaced by the breakpoint at AAAAA

RSEQ COMMAND

The resequence command renumbers the current file. Line numbers are inserted starting with 0010 and continuing at intervals of 0010

1. RSEQ renumbers the current file

SAVE COMMAND

The save command stores the contents of the current file on the tape device. Prompts are provided to start and stop the tape drive.

1. SAVE saves the current file on tape
SYMB COMMAND

The symbols command lists the contents of the assembler symbol table on the console.

1. SYMB lists the symbol table on the console

WRTP COMMAND

The write binary tape command writes a binary tape in the Digital Group standard format. The tape produced may be loaded using the RDTP command or with the system bootstrap loader.

WRTP SSSS EEEE writes a tape of the contents of memory from address SSSS to address EEEE.

3.3 USER DEFINED COMMANDS

Provisions have been made in the operating system for two user-defined commands. These commands may be implemented by entering the four character command name and program start address in the command table of the operating system. When the command name is entered at the console the command will be called by the operating system and is passed the input buffer which can be used to receive a command parameter string. Executing a return instruction reenters the operating system.
4.1 ENTERING A SOURCE PROGRAM

Entering an M6800 source program into the system requires that a file be created using the file manager. This must be done prior to entering statements into the file using the editor package. A file is created by entering the command FILE then the file name followed by its start address. When a file is created it becomes the current file and is the target file for the other commands. The editor package consists of the set of programs invoked using the DELT, AUTO, RSEQ, and LIST commands as well as entry of a line number from the console. These operations always operate on the current file so that once the file of interest is made current using the file manager, source statements may be entered.

Statements are entered into the current file by typing the 4 place decimal line number of the line desired followed by the text of the new line. This works whether the line number was typed directly or generated by the auto line number program. One difference exists between entry of a line using the AUTO mode and typing the line number manually. The AUTO mode prompts the programmer with a space following the line number. This space is not entered into the input buffer. Spaces following line numbers entered manually are entered into the file and will indicate to the assembler that there is no label on the line. If the line number of the statement is presently in the file, then the text for that line is replaced by the new statement. If the line is not presently in the file the new line is placed in the file between the line with the next lower line number and the line with the next greater line number. If a line number is entered with no statement following it then that line is not entered into the file. Lines are removed from the file by typing the number of the line followed by a carriage return. The editor may be invoked at any time in the command mode, but it always operates on the current file.
The contents of the current file may be viewed by invoking the LIST command. The LIST program operates independently from the other commands on the current file and may be used to list sections of the file while editing. The DELT program deletes a range of lines from the current file. This program operates through the editor by generating dummy null lines over the specified range. Errors in using the DELT program are in general, not recoverable. Therefore, it should be used with caution on files which are not backed up on tape. The RSEQ program is used to renumber the file in increments of 10 and will provide space to insert lines in the file. During program entry the AUTO command generates line numbers only for lines not presently in the file. Alternate use of the RSEQ and AUTO command allows any number of statements to be inserted in an existing file.

4.2 USING THE TAPE SYSTEM FOR SAVING SOURCE TAPES

Source files may be saved on and loaded from cassette tapes. The SAVE command writes the contents of the current file to the cassette interface. Because the tape operation is independent of the address range of the current file, tapes may be saved from a file which is located at one address and reloaded to another address. Also, the SAVE is independent of the length of the file. The LOAD operation appends the data from the tape device to the end of the current file. If the file is empty prior to the LOAD then the contents of the tape become the file. If the file has some statements in it prior to the LOAD then the contents of the tape are appended to the end of the file. After the tape has been loaded, the RSEQ command may be used to assign unique line numbers to the statements in the file. Using the commands of the editor system in conjunction with the SAVE and LOAD commands it is possible to merge small files into larger files and to relocate whole blocks of statements within a large file. To do this, the blocks of statements to be merged are written into separate tape files by deleting the lines which surround them in their present files. The files are merged by reading the first part of the new file into memory, appending the second part of the file from tape, and so on. When this is complete, the RSEQ command will supply
the proper line numbers. This procedure may also be used to build libraries of commonly used subroutines. These subroutines could be stored separately on tape and merged as appropriate into a new program.
CHAPTER 5  PROGRAM ASSEMBLY

5.1 USING THE ASSEMBLER

The assembler accepts source input produced by the Editor package. The input format is as described in the Motorola M6800 programming manuals; exceptions are noted in the differences section of this manual. The assembler is invoked by typing ASSM followed by a parameter list. The first parameter is the 'errors only' mode. In this mode only errors encountered in the assembly are listed. This option is selected by entering ASSME followed by the rest of the parameters. The link option is invoked by entering ASSML followed by the parameter list. When the link option is selected the symbol table is not cleared prior to assembly. This feature is used to link two separate assemblies with a common symbol table. To do this, both source programs are assembled using the L option. Cross-program references will be flagged as undefined during the first assembly. Assembling both files a second time will resolve the common symbolic references which were undefined during the first pass. The L and E options may be used together and in any order.

The second parameter of the ASSM command is the binary load address. This is a four digit hexadecimal number which specifies where in memory the binary output of the file is to be located. If this address is not the origin address range used in the assembly, the object file can be moved later using the MOVE command. The load address is not an optional parameter. If it is not specified the BAD PARAMETERS error message will be returned.

The third parameter of the ASSM command is the binary file origin address. The origin address is a four character hex value and is optional. If it is not specified the assembly will default to a start address of 0000. In addition, an ORG directive placed in the source file will always over-ride the start address specification.
5.2 OUTPUT FROM THE ASSEMBLER

The assembler produces two output files. The binary output file is the memory image produced by the assembly and consists of instruction opcodes and data. This file is loaded into memory at the address specified in the ASSM command string. The listing output is displayed on the console and optionally on the printer. The printer is required to properly display the listing. Each listing line consists of the following information: line number, error code (if any), instruction address, opcode, operand data, statement label, instruction mnemonic, operands, and comment. At the end of the assembly listing a count of the number of errors found in the program is shown. It is possible that the number of errors shown on the listing in the error field does not equal the error count. This is due to the fact that only one error is listed for each line. Multiple errors on the same line would be added to the error count but not shown completely in the error code field for the line.

During assembly, a symbol table is produced which shows symbolic references and their values. Although not normally output by the assembler, the contents of this table may be displayed using the SYMB command. The output from this command gives each symbol, the symbol status, and value. The symbol status is represented by a code indicating how it was used by the assembler. Symbols with a status of U through D at the end of the assembly are regarded as errors.

SYMBOL STATUS CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>Undefined but not used</td>
</tr>
<tr>
<td>R</td>
<td>Undefined but used as an operand reference</td>
</tr>
<tr>
<td>D</td>
<td>Double defined</td>
</tr>
<tr>
<td>E</td>
<td>Defined - extended address</td>
</tr>
<tr>
<td>O</td>
<td>Defined - direct address</td>
</tr>
</tbody>
</table>
The R symbol status designates a forward referenced label. A symbol used in this way is never assigned a page zero address by the assembler. The reason for this is to eliminate the "Phasing Error" produced by the Motorola M6800 cross assembler. At the end of the assembly the symbol table dump should be consulted to identify an opportunity to use the direct addressing mode with a symbol. This would appear as a symbol with a status of R or E and having 00 as the most significant byte of its value.
CHAPTER 6 USING THE DEBUGGER PACKAGE

The debugger package is a set of programs used to execute and test a program produced by the assembler. This package consists of the EXEC, BREK, REST, DUMP, ALTR, RDTP, and WRTP commands. A binary program may be entered into memory manually from the console, from tape using the RDTP command, or as output from the assembler. The debugger programs may be used regardless of the source of the binary file.

The EXEC command executes the program starting at the address specified in the command. The program under test can return to the operating system by executing an SWI instruction or by jumping to the operating system entry point - address 0500 (hex). Provided that the stack has not been unbalanced by the program under test, the operating system can also be reentered by executing a return instruction. The BREK command saves the contents of memory at the address specified in the command and inserts an SWI (Software Interrupt) opcode at the address. The contents of memory are restored using the REST command or by inserting a different break address. When the SWI is executed by the program under test, program control passes to the a system program which displays the processor status when the SWI was executed. The status display is formatted as follows:

| FLAGS | A REG | B REG | X REG | STACK | PNTR | PROG | CNTR |

After a breakpoint has been executed from a program under test the system is in the command mode and any of the system commands can be used. The DUMP and ALTR commands are used to display and modify the contents of memory. These programs provide the capability of patching a program for testing purposes. The tape programs RDTP and WRTP interface to the Digital Group tape reader and writer and may be used to write and read object tapes. The tape format used in this command is not compatible with the tapes used by the SAVE and LOAD commands. For both the RDTP and WRTP commands the start address and
stop address must be entered as part of the command string. Therefore, when
WRTP command is used the specified start and stop addresses must be
remembered so that the file can be subsequently reloaded.
CHAPTER 7  DIFFERENCES BETWEEN THIS ASSEMBLER AND THE MOTOROLA CROSS ASSEMBLER

There are a few differences between the features supported by this assembler and the Motorola cross assembler. The entire instruction set of the M6800 processor may be assembled with the system, but only a limited number of assembler directives are supported. The assembler directives which are supported are summarized below.

**SUMMARY OF SUPPORTED ASSEMBLER DIRECTIVES**

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>END</td>
<td>Allowed but not used by the system. Assembly is terminated when end of file is reached.</td>
</tr>
<tr>
<td>FCB</td>
<td>Supported to the extent that single byte definitions only are allowed. Declaration of multiple bytes requires multiple FCB directives.</td>
</tr>
<tr>
<td>FDB</td>
<td>Can be used to define one double byte. Definition of multiple double bytes requires multiple FDB directives.</td>
</tr>
<tr>
<td>ORG</td>
<td>Fully supported</td>
</tr>
<tr>
<td>RMB</td>
<td>Fully supported</td>
</tr>
<tr>
<td>EQU</td>
<td>Fully supported</td>
</tr>
<tr>
<td>FCC</td>
<td>Supports single or multiple byte string definition. The string placed in the operand field must be enclosed in slash signs, e.g. /TEST/</td>
</tr>
</tbody>
</table>
Symbols as well as expressions are supported as operands in addition to constants expressed as decimal, hexadecimal, octal, binary and ASCII strings. Labels are limited to 6 characters in length and operand expressions are limited to 40 characters in length. Expressions are stored internally as 16 bit values so all arithmetic operations are evaluated with 16 bit accuracy.
CHAPTER 8 OPERATING SYSTEM MESSAGES

8.1 ASSEMBLER ERRORS

The assembler scans each line for the number and type of operands and for the validity of the instruction mnemonic. Listed below are the possible assembler error messages which are displayed in the listing between the line number and address fields of each line. A blank in this position indicates that the assembler found no errors in that line.

<table>
<thead>
<tr>
<th>ERROR CODE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>DOUBLE DEFINE The value of the label was defined more than once during the assembly.</td>
</tr>
<tr>
<td>U</td>
<td>UNDEFINED The symbolic reference in the operand field was not defined during the assembly.</td>
</tr>
<tr>
<td>I</td>
<td>INVALID MNEMONIC The string in the mnemonic field is not a valid M6800 instruction.</td>
</tr>
<tr>
<td>M</td>
<td>MISSING OPERAND The required number or type of operands for the instruction indicated was not found.</td>
</tr>
<tr>
<td>B</td>
<td>BRANCH ERROR The range of the indicated branch is less than -128 or greater than +127.</td>
</tr>
</tbody>
</table>

8.2 EXPLANATION OF OPERATING SYSTEM MESSAGES

The operating system generates error messages in response to certain
commands which are entered erroneously or when it finds a file which does not conform to the proper data structure. Listed below are the possible system messages and the conditions which can cause their being displayed.

<table>
<thead>
<tr>
<th>MESSAGE</th>
<th>CAUSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHAT?</td>
<td>The command name just entered is not recognized by the system</td>
</tr>
<tr>
<td>N.I.</td>
<td>The command name which was entered is a recognized system command but, is reserved for a function which is NOT IMPLEMENTED. Two commands CST1 and CST2 currently return this code.</td>
</tr>
<tr>
<td>BAD FILE</td>
<td>This command is sent by the system when the programmer attempts to use one of the system programs which is capable of modifying the current file when there is no active file. It is also sent when the system encounters improper data in a file. The best recovery from this condition is to delete the file and reload it from tape. It is also possible to dump the file using the DUMP command and try to reconstruct the file manually using the ALTR command. Details of how to do this may be found in the Editor / Assembler Design Document.</td>
</tr>
<tr>
<td>TOO MANY SYMBOLS</td>
<td>The symbol table has overflowed its bounds during assembly. If the system has been configured to use an abbreviated symbol table</td>
</tr>
</tbody>
</table>
(less than 256 symbols), lengthening the symbol table will correct this problem. Otherwise the program must be broken up into smaller pieces and assembled separately.

**START RECORDER**

During SAVE and LOAD operations this command prompts the programmer to turn on the tape recorder.

**STOP RECORDER**

During SAVE and LOAD operations this message prompts the programmer to turn off the tape recorder and indicates that the system is ready to accept the next command.
APPENDIX
# Table of Contents

## Editor Section
- Process the next console command ........................................... 2
- Dump Memory .............................................................................. 3
- Alter Memory ............................................................................ 4
- Clean System Memory ............................................................... 5
- Activate Auto Line Number Mode ............................................... 6
- Exit to Menu Display ............................................................... 7
- Read Binary Tape ...................................................................... 8
- Execute Binary Program ............................................................ 9
- Insert a Breakpoint ................................................................... 9
- Resume Code from Breakpoint .................................................. 11
- Display Status on Breakpoint .................................................... 12
- Move Binary Program .............................................................. 13
- Print Contents of Symbol Table ................................................ 14
- Delete Lines from Current File .................................................. 15
- Data Organization for File Parameter Storage Buffer .................. 16
- Modify the List of Active Files .................................................. 17
- List the Current File ................................................................ 18
- Re-run the Current File ............................................................ 19
- Data Organization of Current File Ram ...................................... 20
- Edit the Current File ............................................................... 21
- Load Memory from Tape ........................................................... 22
- Store Memory on Tape .............................................................. 23
- Console Error Strategy ............................................................. 24
- Process Console Error ............................................................... 25

## Assembler Section
- Data Organization ................................................................... 26
- Assembly Language Information Table Data Organization .......... 27
- Assemble the Current File ......................................................... 28
- Execute Pass 1 ......................................................................... 29
- Execute Pass 2 ......................................................................... 30
- Parse the Current Source Line and Process It ......................... 31
- Parse the Label Field ............................................................... 32
- Parse the Mnemonic Field ......................................................... 33
- Process Instruction Parameters ............................................... 34
- Parse the Rest of the Line ....................................................... 35
- Parse Register or Operands ....................................................... 36
- Parse Register ......................................................................... 37
- Parse Register and Operands .................................................... 38
- Parse Comment Only ............................................................... 39
- Parse Register Only ............................................................... 40
- Parse Register and Operands Only ........................................... 41
- Process Assembler Directive .................................................... 42
- Process Single WTH Instructions .............................................. 43
- Process Table 'A-I Instruction' ................................................. 44
PROCESS TABLE A-2 INSTRUCTION ................................................. 48
PROCESS TABLE A-3 INSTRUCTION ................................................. 49
PROCESS TABLE A-4 INSTRUCTION ................................................. 50
PROCESS TABLE A-5 INSTRUCTION ................................................. 51
PROCESS TABLE A-6 INSTRUCTION ................................................. 52
PROCESS TABLE A-7 INSTRUCTION ................................................. 53
PROCESS TABLE A-R INSTRUCTION ................................................. 54
CHECK DUAL REGISTER MODE ....................................................... 55
CHECK IMMEDIATE MODE .......................................................... 56
CHECK INDEXED MODE ............................................................. 57
CHECK DIRECT OR EXTENDED ....................................................... 58
PROCESS END DIRECTIVE ......................................................... 59
PROCESS EPV DIRECTIVE .......................................................... 60
PROCESS FC0 DIRECTIVE .......................................................... 61
PROCESS FCH DIRECTIVE .......................................................... 62
PROCESS FCC DIRECTIVE .......................................................... 63
PROCESS ORI DIRECTIVE .......................................................... 64
PROCESS RMB DIRECTIVE .......................................................... 65
PROCESS ASSEMBLER ERROR ....................................................... 66
ENTER THE LABEL IN THE SYMBOL TABLE ...................................... 67
RETRIEVE VALUE FROM SYMBOL TABLE ........................................ 68
EVALUATE OPERAND ............................................................... 69
EVALUATE CHARACTER STRING ..................................................... 70
EVALUATE EXPRESSION .............................................................. 71
EVALUATE CONSTANT ............................................................... 72
OUTPUT LISTING INFO ............................................................... 73
OUTPUT HINARY INFO ............................................................... 74

SYSTEM UTILITIES ........................................................................ 75
CONVERT FROM BINARY TO ASCII ................................................. 76
CONVERT ASCII HINARY TO BINARY ............................................. 77
CONVERT ASCII OCTAL TO BINARY .............................................. 78
CONVERT HINARY TO ASCII DECIMAL .......................................... 79
PROCESS THE NEXT CONSOLE COMMAND

REF

PAGE

* 1 DO FOREVER
* 2   CLEAN THE INPUT BUFFER EXCEPT FOR LINE NUMBER FIELD
* 3       IF AUTO LINE NUMBER MODE IS ACTIVE
* 4           CONVERT ASCII TO BINARY (LINE NUMBER)
* 5               INCREMENT LINE NUMBER
* 6                   IF NEW LINE NUMBER IS PRESENTLY IN THE FILE
* 7                     CANCEL AUTO LINE NUMBER MODE
* 8                       ENDF
76  9             CONVERT FROM BINARY TO ASCII (LINE NUMBER)
* 10              ELSE
* 11                     CLEAR THE LINE NUMBER PORTION OF THE INPUT BUFFER
* 12                    ENDF
* 13                   GET A LINE OF INPUT TEXT
* 14                   IF LINE CANCEL COMMAND IS NOT IN INPUT STRING
* 15                             IF AUTO LINE NUMBER CANCEL COMMAND IS IN INPUT STRING
* 16                         CANCEL AUTO LINE NUMBER MODE
* 17                             ELSE
* 18                             DO CASE OF FIRST FOUR ELEMENTS OF INPUT BUFFER
* 19   AUTO:
* 20                       ACTIVATE AUTO LINE NUMBER MODE
* 21  ASSM:
* 22
* 23 LOAD:
* 24             ASSEMBLE THE CURRENT FILE
* 25 SAVE:
* 26              LOAD MEMORY FROM TAPE
* 27 OELT:
* 28                      STORE MEMORY ON TAPE
* 29 FILE:
* 30                          DELETE LINES FROM CURRENT FILE
* 31 LIST:
* 32                      MODIFY THE LIST OF ACTIVE FILES
* 33
* 34 BRK:
* 35                   LIST THE CURRENT FILE
* 36                   INSERT A BREAKPOINT
* 37 EXIT:
* 38                   EXIT TO MENU DISPLAY
* 39 write:
* 40                    WRITE BINARY TAPE
* 41 RTPI:
* 42                          READ BINARY TAPE
* 43 WRT:
* 44                   CLEAR SYSTEM MEMORY
* 45 REST:
* 46                   RESTORE LINE FROM BREAKPOINT
* 47 DUMP:
* 48                    DUMP MEMORY
* 49 ALTD:
* 50 ALTER MEMORY
* 51 WSEND:
* 52                    REDO/LREPLACE THE CURRENT FILE
* 51 EXEC:
  EXECUTE BINARY PROGRAM
* 52
* 53 MUVE:
  MOVE BINARY PROGRAM
* 54
* 55 PRUN:
  ENABLE PRINTER
* 56
* 57 PRUF:
  VISABLE PRINTER
* 58
* 59 SYMB:
  PRINT CONTENTS OF SYMBOL TABLE
* 60
  ENDO
* 61
  IF THE FIRST FOUR CHARACTERS OF INPUT BUFFER REPRESENT A DECIMAL NUMBER
* 62
  EDIT THE CURRENT FILE
* 63
  ENDF
* 64
  ENDF
* 65
  IF COMMAND IS NOT A VALID CHOICE
* 66
  PROCESS CONSOLE ERROR( INVALID COMMAND )
* 67
  ENDF
* 68
  ENDF
* 69
  ENDO
* 70

**************************************************************************

***** TOO MANY LINES IN SEGMENT
DUMP MEMORY

REF

PAGE

******************************************************************************
* 1 GET START AND END ADDRESSES FROM INPUT BUFFER
* 2 CONVERT TO BINARY (START AND END ADDRESS)
* 3 POINT = START ADDRESS
* 4 DO WHILE POINTER <= END ADDRESS
* 5 CONVERT POINTER TO ASCII AND SAVE IN OUTPUT BUFFER
* 6    DU 8 TIMES
* 7    CONVERT <POINTER> TO ASCII AND STORE IN OUTPUT BUFFER
* 8    INCREMENT POINTER
* 9    PRINT CONTENTS OF OUTPUT BUFFER
* 10   ENDO
* 11   ENDO
* 12   RETURN

******************************************************************************
ALTER MEMORY

* 1 GET ADDRESS OF CHANGE LOCATION FROM INPUT BUFFER
* 2 CONVERT TO BINARY AND STORE IN ADDRESS POINTER
* 3 DO UNTIL AN EMPTY LINE IS INPUT
* 4 LOAD OUTPUT BUFFER WITH ASCII VALUE OF POINTER
* 5 OUTPUT A LINE TO THE CONSOLE
* 6 GET A LINE OF INPUT
* 7 IF THE LINE IS NOT EMPTY
* 8 IF THE FIRST CHARACTER IS NOT A SPACE
* 9 CONVERT VALUE OF FIRST TWO CHARACTERS TO BINARY
* 10 STORE IN <ADDRESS POINTER>
* 11 ENDF
* 12 ENDF
* 13 ENDO
* 14 RETURN

***************************************************************************
CLEAN SYSTEM MEMORY

REF PAGE

**************************************************************************

* 1 DU FOR MEMORY POINTER = END OF ASSEMBLER TO END OF MEMORY
* 2 <MEMORY POINTER> = 0
* 3 ENDO
* 4 RETURN
**************************************************************************
ACTIVATE AUTO LINE NUMBER MODE

1. IF THERE IS AN ACTIVE FILE

2. SIGNAL AUTO LINE NUMBER MODE ACTIVE

3. GET NEXT LINE NUMBER FROM INPUT BUFFER OR 0001

4. CONVERT STRING TO ASCII, LAST LINE NUMBER

5. WRITE INTO FIRST FOUR CHARACTERS OF INBUF

6. ELSE PROCESS CONSOLE ERROR (IAO FILE)

7. RETURN

8. ENDIF

9. RETURN
WRITE BINARY TAPE

* 1 GET START ADDRESS AND STOP ADDRESS FROM INPUT BUFFER
* 2 CONVERT TO BINARY
* 3 STORE IN DIGITAL GROUP TAPE WRITER START AND END ADDRESSES
* 4 PRINT "CASSETTE BEING WRITTEN" MESSAGE
* 5 JUMP TO DIGITAL GROUP TAPE WRITER PROGRAM

*****************************************************************************
READ BINARY TAPE

<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GET START ADDRESS AND STOP ADDRESS FROM INPUT BUFFER</td>
</tr>
<tr>
<td>2</td>
<td>CONVERT TO BINARY</td>
</tr>
<tr>
<td>3</td>
<td>STORE IN DIGITAL GROUP TAPE READER START AND END ADDRESSES</td>
</tr>
<tr>
<td>4</td>
<td>JUMP TO DIGITAL GROUP TAPE READER PROGRAM</td>
</tr>
</tbody>
</table>

* * *
EXECUTE BINARY PROGRAM

REF PAGE

*******************************************************************************************************
* *
* 1 GET JUMP ADDRESS FROM INPUT BUFFER *
* 2 CONVERT ASCII TO BINARY(JUMP ADDRESS) *
* 3 JUMP TO JUMP ADDRESS *
* *
*******************************************************************************************************
**INSERT A BREAKPOINT**

1. Get break address from input buffer
2. Resume code from breakpoint
3. Convert ASCII to binary (break address)
4. Store byte at break address in break code save
5. Insert 'SWI', at break address
6. Signal 'BREAK' point active
7. Return
RESTORE CODE FROM BREAKPOINT

REF PAGE

*******************************************************************************
* *
* 1  IF BREAK POINT ACTIVE
* 2  STORE BREAK CODE SAVE AT BREAK ADDRESS
* 3  SIGNAL BREAK POINT NOT ACTIVE
* 4  ENDIF
* 5  RETURN
* *
*******************************************************************************
DISPLAY STATUS ON BREAKPOINT

REF PAGE

* 1 RETRIEVE PROCESSOR STATUS FROM STACK
* 2 LOAD OUTPUT BUFFER WITH ASCII VALUES OF STATUS
* 3 OUTPUT A LINE TO THE CONSOLE
* 4 PROCESS THE NEXT CONSOLE COMMAND

*******************************************************************************

*******************************************************************************
MOVE BINARY PROGRAM

* REF
* PAGE
* 1 GET SOURCE START ADDRESS FROM INPUT BUFFER
* 2 CONVERT FROM ASCII TO BINARY (SOURCE START ADDRESS)
* 3 GET SOURCE END ADDRESS FROM INPUT BUFFER
* 4 CONVERT FROM ASCII TO BINARY (SOURCE END ADDRESS)
* 5 GET DESTINATION START ADDRESS FROM INPUT BUFFER
* 6 CONVERT FROM ASCII TO BINARY (DESTINATION START ADDRESS)
* 7 MOVE COUNT = SOURCE END ADDRESS - SOURCE START ADDRESS
* 8 IF SOURCE START ADDRESS < DESTINATION START ADDRESS
* 9 DESTINATION START ADDRESS = DESTINATION START ADDRESS + MOVE COUNT
* 10 DU MOVE COUNT TIMES
* 11 <DESTINATION START ADDRESS> = <SOURCE END ADDRESS>
* 12 DECREMENT SOURCE END ADDRESS, DESTINATION START ADDRESS
* 13 ENDU
* 14 ELSE
* 15 DU MOVE COUNT TIMES
* 16 <DESTINATION START ADDRESS> = <SOURCE START ADDRESS>
* 17 INCREMENT SOURCE START ADDRESS, DESTINATION START ADDRESS
* 18 ENDO
* 19 ENDF
* 20 RETURN
*
PRINT CONTENTS OF SYMBOL TABLE

REF PAGE

******************************************************************************
*   * 1 DO FOR THE NUMBER OF SYMBOLS IN THE SYMBOL TABLE
*   * 2 RETRIEVE SYMBOL VALUE
*   * 3 CONVERT BINARY TO ASCII (SYMBOL VALUE)
*   * 4 ENCODE SYMBOL STATUS
*   * 5 OUTPUT A LINE OF TEXT(STATUS,SYMBOL,VALUE)
*   * 6 ENDO
*   * 7 RETURN

******************************************************************************
DELETE LINES FROM CURRENT FILE

**REF**
**PAGE**

**----------------------------------------------------------------------**
* 1  GET THE FIRST LINE NUMBER TO BE REMOVED AND LAST LINE NUMBER TO BE REMOVED FROM THE INPUT STREAM
  2  CLEAN THE INPUT BUFFER
  3  CONVERT ASCII TO BINARY(FIRST LINE NUMBER TO BE REMOVED)
  4  CONVERT ASCII TO BINARY(LAST LINE NUMBER TO BE REMOVED)
  5  LINE NUMBER = FIRST LINE NUMBER TO BE REMOVED
  6  DO UNTIL LINE NUMBER GE LAST LINE NUMBER TO BE REMOVED
  7  CONVERT BINARY TO ASCII(LINE NUMBER)
  8  STORE LINE NUMBER IN INPUT BUFFER
  9  EDIT THE CURRENT FILE
 10  LINE NUMBER = LINE NUMBER + 1
 11  ENDU
 12  RETURN

**----------------------------------------------------------------------**
# DATA ORGANIZATION FOR FILE PARAMETER STORAGE BUFFER

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FILE NAME</td>
<td>START ADDRESS</td>
<td>END ADDRESS</td>
</tr>
<tr>
<td>2</td>
<td>ASCII</td>
<td>BINARY</td>
<td>BINARY</td>
</tr>
<tr>
<td>3</td>
<td>6 BYTES</td>
<td>2 BYTES</td>
<td>2 BYTES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>NAME 1</td>
<td>START ADD 1</td>
<td>END ADD 1</td>
</tr>
<tr>
<td>6</td>
<td>NAME 2</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>7</td>
<td>***</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>***</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>NAME 5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MODIFY THE LIST OF ACTIVE FILES

* 1 IF A BLANK FOLLOWS 'F I L E'
  * 2 SCAN INPUT BUFFER FOR FILE NAME
  * 3 RIGHT JUSTIFY THE CHARACTERS OF THE NAME
  * 4 SCAN INPUT STREAM FOR START ADDRESS
  * 5 IF START ADDRESS IS NOT IN INPUT BUFFER
  * 6 START ADDRESS = DUMMY NUMBER (DEFAULT TO MAKE IT CURRENT)
  * 7 ENDF
  * 8 CONVERT THE START ADDRESS FROM ASCII TO BINARY
  * 9 POSITION THE FILE MANAGER POINTER AT THE START OF FILE PARAMETER STORAGE BUFFER
  * 10 UD UNTIL FILE MANAGER POINTER POINTS TO FILE NAME OR END OF FILE MANAGER STORAGE BUFFER
  * 11 POINT TO THE NEXT LINE OF FILE INFO
  * 12 ENDO
  * 13 IF THE FILE NAME IS PRESENTLY IN THE LIST OF ACTIVE FILES
    * 14 SAVE THE START ADDRESS AND FILE END ADDRESS IN TEMPORARY BUFFER
    * 15 REMOVE THE FILE NAME AND ITS PARAMETERS FROM FILE PARAMETER STORAGE BUFFER
    * 16 HUMP ALL FILE PARAMETERS BELOW THIS FILE UP ONE ROW AND FILL THE LAST PARAMETER ROW WITH /*
       * 17 BLANKS
    * 18 ENDOF
    * 19 IF THE START ADDRESS IS NOT ZERO
    * 20 HUMP ALL PARAMETERS OWN ONE POSITION IN THE FILE PARAMETER STORAGE BUFFER
    * 21 (TEMP BUFFER IS ABOVE FILE AREA AND IS MOVED INTO CURRENT BUFFER)
    * 22 THUS MAKING IT CURRENT
    * 22 ENDOF
  * 23 ENDF
  * 24 UD FOR EACH ACTIVE FILE IN FILE PARAMETER STORAGE BUFFER
  * 76 * 25 CONVERT FROM BINAKY TO ASCII (START ADDRESS)
  * 76 * 26 CONVERT FROM BINAKY TO ASCII (END ADDRESS)
    * 27 FILL UP OUTPUT BUFFER WITH FILE NAME, START ADDRESS, END ADDRESS,CR
    * 28 OUTPUT A LINE OF TEXT BUFFER FROM ABOVE)
  * 29 ENDO
  * 30 RETURN

******************************************************************************
LIST THE CURRENT FILE

```assembly
* 1 SCAN INPUT BUFFER FOR START LINE NUMBER AND END LINE NUMBER
* 2 IF START LINE NUMBER SPECIFIED
* 3 TEXT POINTER = START ADDRESS OF SPECIFIED LINE
* 4 ELSE
* 5 TEXT POINTER = START ADDRESS OF CURRENT FILE
* 6 ENDF
* 7 IF END ADDRESS SPECIFIED
* 8 TEXT POINTER = END ADDRESS OF LINE NUMBER SPECIFIED
* 9 ELSE
* 10 TEXT POINTER = END ADDRESS OF CURRENT FILE
* 11 ENDF
* 12 DO WHILE TEXT POINTER IS LESS THAN OR EQUAL TO CURRENT FILE END ADDRESS
* 13 LINE LENGTH = <TEXT POINTER>
* 14 INCREMENT TEXT POINTER
* 15 LINE NUMBER = <TEXT POINTER> & <TEXT POINTER+1>
  7b 16 CONVERT FROM BINARY TO ASCII (LINE NUMBER)
* 17 DO UNTIL TEXT POINTER GE TEXT POINTER + LINE LENGTH
* 18 INCREMENT TEXT POINTER
* 19 ENDDO
* 20 OUTPUT A LINE OF TEXT (OUTPUT BUFFER, LENGTH)
* 21 IF <TEXT POINTER> NE LINE MARKER
  27 22 PROCESS CONSOLE ERROR (CORRUPT FILE)
* 23 ENDDO
* 24 ENDF
* 25 RETURN
```
RENUMBER THE CURRENT FILE

REF

PAGE

**************************************************************************

* 1 LINE NUMBER=10
* 2 TEXT POINTER = CURRENT FILE START ADDRESS
* 3 TEXT LINE LENGTH = 0
* 4 DO UNTIL TEXT POINTER GE CURRENT FILE END ADDRESS
* 5 TEXT LINE LENGTH = <TEXT POINTER>
* 6 INCREMENT TEXT POINTER
* 7 <TEXT POINTER> & <TEXT POINTER + 1> = TEXT LINE NUMBER
* 8 INCREMENT TEXT POINTER
* 9 TEXT POINTER = TEXT POINTER + TEXT LINE LENGTH
* 10 TEXT LINE NUMBER = TEXT LINE NUMBER + 10
* 11 IF <TEXT POINTER> NE LINE MARKER
* 12 PROCESS CONSOLE(CORRUPT FILE)
* 13 ENDIF
* 14 ENDDU
* 15 RETURN

**************************************************************************
DATA ORGANIZATION OF CURRENT FILE RAM

# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# # 1  LINE  LINE  TEXT   END
# # 2  LENGTH NUMBER  OF LINE
# # 3  1 BYTE  2 BYTES  NBYTES  marker
# # 4  BINARY  BINARY  ASCII  ASCII
# # 5  %
# # 6  L1  N1  TEXT OF LINE  CR
# # 7  L2  N2  TEXT2  CR
# # 8  %
# # 9  %
# # 10  %
# # 11  %
# # 12  LX  NX  TEXTX  CR
# # 13  %
# # 14  THE FILE START ADDRESS POINTS TO L1 AND THE END
# # 15  OF FILE ADDRESS POINTS TO THE LAST CR
# # 16  %
# # 17  SPACES ARE STORED WITH THE MSB SET AND LS 7 BITS INDICATING
# # 18  THE NUMBER OF CONSECUTIVE SPACES
# # %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
EDIT THE CURRENT FILE

**CONVERT THE LINE NUMBER IN THE INPUT BUFFER TO BINARY AND STORE IN CURRENT LINE NUMBER**

**INITIALIZE TEXT POINTER TO CURRENT FILE START ADDRESS**

**TEXT LINE NUMBER = 0**

**TEXT LINE LENGTH = 0**

**IF THERE IS AN ACTIVE FILE**

**UNTIL TEXT LINE NUMBER GE CURRENT LINE NUMBER OR CURRENT FILE END ADDRESS**

**TEXT POINTER = TEXT POINTER + TEXT LINE LENGTH**

**INCREMENT TEXT POINTER**

**TEXT LINE LENGTH = <TEXT POINTER>**

**INCREMENT TEXT POINTER**

**TEXT LINE NUMBER = <TEXT POINTER> & <TEXT POINTER + 1>**

**IF TEXT POINTER WE LINE MARKER**

**PROCESS CONSOLE ERROR (CORRUPT FILE)**

**TEXT POINTER = TEXT POINTER - 3**

**TEXT POINTER POINTS TO START OF DESIRED LINE**

**IF TEXT LINE NUMBER = CURRENT LINE NUMBER**

**TEXT POINTER = TEXT POINTER + 3 + TEXT LINE LENGTH**

**NUMBER OF CHARACTERS TO REMOVE = 4 + TEXT LINE LENGTH**

**COMPRESS THE FILE (NUMBER OF CHARACTERS TO REMOVE)**

**CURRENT FILE SIZE ADDRESS = CURRENT FILE END ADDRESS - NUMBER OF CHARACTERS TO REMOVE**

**ENDIF**

**IF INPUT BUFFER LENGTH > 4**

**NUMBER OF CHARACTERS TO OFFSET = INPUT BUFFER LENGTH + 4**

**EXPAN! FILE (NUMBER OF CHARACTERS TO OFFSET)**

**TEXT POINTER NOT MODIFIED**

**CURRENT FILE SIZE ADDRESS CURRENT FILE END ADDRESS + NUMBER OF CHARACTERS TO OFFSET**

**INCREMENT TEXT POINTER**

**<TEXT POINTER> = INPUT BUFFER LENGTH - 4**

**INCREMENT TEXT POINTER**

**<TEXT POINTER> & <TEXT POINTER + 1> = LINE NUMBER**

**MOVE CHARACTERS FROM INPUT BUFFER INTO <TEXT POINTER>**

**<TEXT POINTER> = LINE MARKER (CR)**

**ENDIF**

**ELSE**

**PROCESS CONSOLE ERROR (BAD FILE)**

**ENDIF**

**RETURN**
LOAD MEMORY FROM TAPE

**REF**

**PAGE**

*1. IF THERE IS AN ACTIVE FILE*

*2. PRINT TAPE START MESSAGE*

*3. DELAY LEADER*

*4. TAPE LOAD ADDRESS = CURRENT FILE START ADDRESS*

*5. UD UNTIL TAPE CHARACTER = S*

*6. READ TAPE CHARACTER*

*7. ENDO*

*8. READ TAPE CHARACTER AND STORE IN MSB TAPE LENGTH*

*9. READ TAPE CHARACTER AND STORE IN LSB TAPE LENGTH*

*10. CURRENT FILE END ADDRESS = CURRENT FILE END ADDRESS + TAPE LENGTH*

*11. UD UNTIL TAPE LENGTH = 0*

*12. READ TAPE CHARACTER AND STORE INTO <LOAD ADDRESS>*

*13. INCREMENT LOAD ADDRESS*

*14. DECREMENT TAPE LENGTH*

*15. ENDO*

*16. DELAY TRAILER*

*17. PRINT TAPE STOP MESSAGE*

*18. ELSE*

*27.1 PROCESS CONSOLE ERROR(BAD FILE)*

*20. ENDIF*

*21. RETURN*

*
# CONSULE ERROR STRATEGY

# # 1 THE CONSULE ERROR PROCESSOR IS THE ONLY UNSTRUCTURED # #
# 2 PORTION OF THE OPERATING SYSTEM. WHEN AN ERROR IS DISCOVERED #
# 3 AT THE COMMAND LEVEL, PROCESSING OF THE CURRENT COMMAND #
# 4 IS TERMINATED AND CONTROL PASSES THROUGH THE ERROR #
# 5 PROCESSOR BACK TO THE COMMAND PROCESSOR. THIS PROCEDURE #
# 6 ALLOWS MORE THAN ONE EXIT FROM A SECTION OF CODE #
# 7 AND AS SUCH IS A STRUCTURE VIOLATION. THIS HAS BEEN ALLOWED #
# 8 TO LOWER THE POSSIBILITY THAT A MIS SPECIFIED COMMAND #
# 9 WILL BE INTERPRETED AS CORRECT #

***************************************************************
***************
* ASSEMBLER SECTION *
***************
DATA ORGANIZATION

# 1: SOURCE INPUT FILE
# 2
# 3 THE ASSEMBLER SOURCE INPUT FILE IS ORGANIZED AS SHOWN
# 4 IN THE EDITOR SECTION ABOVE. THE TEXT PORTION OF EACH LINE IS
# 5 ORGANIZED AS FOLLOWS:
# 6
# 7 LABEL FIELD, 1-N BLANKS, MNEMONIC FIELD, 1-N BLANKS, OPERANDS, 1-N BLANKS, COMMENTS
# 8
# 9 THE LABEL FIELD MAY BE COMPOSED OF 1 TO 6 CHARACTERS AND MAY OPTIONALLY
# 10 BE A BLANK IF THE LINE HAS NO LABEL OR A COMMENT CHARACTER IF THE
# 11 ENTIRE LINE IS A COMMENT
# 12
# 13 THE OPCODE FIELD MAY CONTAIN ONE OF THE MOTOROLA STANDARD SYMBOLIC
# 14 ASSEMBLY LANGUAGE MNEMONICS OR ASSEMBLER DIRECTIVES
# 15
# 16 THE OPERAND FIELD MAY HAVE THE NUMBER AND TYPES OF OPERANDS
# 17 SUPPORTED BY THE OPCODE IN THAT LINE
# 18
# 19 THE COMMENT FIELD IS COMPOSED OF SUPPORTING COMMENTARY SUBJECT
# 20 TO THE LINE LENGTH RESTRICTIONS OF THE EDITOR
# 21
# 22 2: SYMBOL TABLE
# 23
# 24 START ADDRESS OF SYMBOL TABLE
# 25 1 BYTE SYMBOL STATUS, 6 BYTE ASCII SYMBOL, 2 BYTE BINARY VALUE
# 26 1 BYTE SYMBOL STATUS, 6 BYTE ASCII SYMBOL, 2 BYTE BINARY VALUE
# 27
# 28 *  
# 29
# 30 1 BYTE SYMBOL STATUS, 6 BYTE ASCII SYMBOL, 2 BYTE BINARY VALUE
# 31 END OF ASCII SYMBOL TABLE ADDRESS
# 32
# 
ASSEMBLY LANGUAGE INFORMATION TABLE DATA ORGANIZATION

# 1 THE MNEMONICS ARE ORGANIZED BY COMMON ADDRESSING MODES ACCORDING #
# 2 TO TABLES A-1 THROUGH A-8 OF THE M6800 PROGRAMMING MANUAL #
# 3 THE 3 CHARACTER MNEMONIC IS STORED IN THE TABLE WITH A BASE OPCODE #
# 4 AND ADDRESSING MODE NUMBER. THE ADDRESSING MODE NUMBER TELLS THE #
# 5 OPERAND PARSER WHAT TYPES OF OPERANDS ARE EXPECTED. #
# 6 OFFSETS ARE SET UP DEPENDING ON THE ADDRESSING MODE NUMBER #
# 7 AND ARE ADDED TO THE BASE OPCODE TO DETERMINE THE PROPER #
# 8 OPCODE FOR THE ADDRESSING MODE INDICATED. #

# 10 MNEMONIC   BASE OPCODE   ADDRESSING FORMAT #
# 11 #
# 12 3 BYTES 1 BYTE 1 BYTE #
# 13 #
# 14 ASCII  BINARY  BINARY #
# 15 #
# 16 #
# 17 0 = SINGLE BYTE INSTRUCTION #
# 18 1 = TABLE A-1 M6800 MICROPROCESSOR #
# 19 2 = TABLE A-2 PROGRAMMING MAN. #
# 20 #
# 21 #
# 22 #
# 23 #
# 24 #
# 25 8 = TABLE A-8 #
# 26 9 = ASSEMBLER DIRECTIVE #
# 27 #

#
ASSEMBLE THE CURRENT FILE

* REF
  PAGE

* 1 IF BINARY DESTINATION ADDRESS IS SPECIFIED
  * 2 PROGRAM START ADDRESS=0
  * 3 IF AN 'E' FOLLOWS ASSM
  * 4 SET ERRORS ONLY MODE
  * 5 ENDF
  * 6 IF AN 'L' DOES NOT FOLLOW A S S M
  * 7 CLEAR THE SYMBOL TABLE
  * 8 ENDF
  * 9 GET THE BINARY OUTPUT DESTINATION FROM THE INPUT STREAM
  * 10 IF AN OPTIONAL START ADDRESS IS IN THE INPUT STRING
  * 11 GET PROGRAM START ADDRESS FROM INPUT STREAM
  * 12 ENDF
  32  13 EXECUTE PASS 1
  33  14 EXECUTE PASS 2
  * 15 OUTPUT NUMBER OF ERRORS
  * 16 ELSE
  27  17 PROCESS CONSOLE ERROR(BAD PARAMETERS)
  * 18 ENDF
  * 19 RETURN

*
EXECUTE PASS 1

REF PAGE

**********

*    1  SET THE PROGRAM COUNTER TO PROGRAM START ADDRESS
*    2  DU FOR EACH LINE IN THE CURRENT FILE UNTIL TEXT POINTER = CURRENT FILE END ADDRESS
*    3  PARSE THE CURRENT SOURCE LINE AND PROCESS IT
*    4  INCREASE THE PROGRAM COUNTER BY NUMBER OF BYTES IN THIS INSTRUCTION
*    5  ENOU
*    6  RETURN

**********
EXECUTE PASS 2

REF
PAGE 1

SET THE PROGRAM COUNTER TO PROGRAM START ADDRESS
2
CLEAR ERROR COUNT
3
DO FOR EACH LINE IN THE CURRENT FILE UNTIL TEXT POINTER = CURRENT FILE END ADDRESS
4
PARSE THE CURRENT SOURCE LINE AND PROCESS IT
5
OUTPUT BINARY INFO
6
OUTPUT LISTING INFO
7
ADD NUMBER OF BYTES FOR THIS INSTRUCTION TO PROGRAM COUNTER
8
ENDU
PARSE THE CURRENT SOURCE LINE AND PROCESS IT

* 1 INCREMENT SOURCE POINTER
* 2 SOURCE LINE LENGTH = <SOURCE POINTER>
* 3 INCREMENT SOURCE POINTER
* 4 SOURCE LINE NUMBER = <SOURCE POINTER> & <SOURCE POINTER + 1>
* 5 INCREMENT SOURCE POINTER
* 6 (SOURCE POINTER IS NOW POINTING TO THE START OF THE LABEL FIELD)

35* 7 PARSE THE LABEL FIELD
* 8 IF THE LINE IS NOT A COMMENT
* 9    PARSE THE MNEMONIC FIELD
* 10   PROCESS INSTRUCTION PARAMETERS
* 11   IF THIS LINE HAS A LABEL
* 12   ENTER THE LABEL IN THE SYMBOL TABLE
* 13   ENOIF
* 14   ENOIF
* 15   IF <SOURCE_POINTER> NE LINE MARKER
* 16   PROCESS CUNSOLE ERROR (CORRUPT FILE)
* 17   ENOIF
* 18   RETURN

******************************************************************************
PARSE THE LABEL FIELD

**-----------------------------------------------**
*   * 
* 1  IF THE FIRST CHARACTER IS NOT THE COMMENTS SYMBOL* 
* 2   SIGNAL THIS LINE IS NOT A COMMENT* 
* 3   FIND CHARACTER (THE BLANK FOLLOWING THE LABEL)* 
* 4   IF THE LENGTH OF THE LABEL IS GREATER THAN 6* 
66  5   PROCESS ASSEMBLER ERROR (LINE NUMBER, INVALID LABEL)* 
* 6   ELSE* 
* 7   IF THE LABEL IS OF LENGTH ZERO* 
* 8   FILL THE CURRENT LINE LABEL WITH BLANKS* 
* 9   ELSE* 
*10  WRITE LABEL CHARACTERS INTO CURRENT LINE LABEL* 
*11  ENDIF* 
*12  ENDIF* 
*13  ELSE* 
*14  SIGNAL THIS LINE IS A COMMENT* 
*15  FILL CURRENT LINE LABEL WITH BLANKS* 
*16  ENDIF* 
*17  SAVE POINTER TO START OF MNEMONIC FIELD* 
*18  RETURN* 
**-----------------------------------------------**
PARSE THE MNEMONIC FIELD

*
* 1  PICK UP POINTER TO START OF MNEMONIC FIELD
* 2  SAVE THE 3 CHARACTERS WHICH FOLLOW IN CURRENT LINE OPCODE
* 3  SAVE POINTER TO START OF OPERAND FIELD
* 4  RETURN
*
PROCESS INSTRUCTION PARAMETERS

* reference page 1
* look for the current line mnemonic in assembly language info table
* if mnemonic is found in the table
* current line opcode = base opcode (this table position)
* get the addressing format for this mnemonic from info table
* parse the rest of line
* evaluate operand
* 0: case of addressing format
* 9: process single byte instructions
* 10: process table A-1 instruction
* 12: process table A-2 instruction
* 14: process table A-3 instruction
* 16: process table A-4 instruction
* 18: process table A-5 instruction
* 20: process table A-6 instruction
* 22: process table A-7 instruction
* 24: process table A-8 instruction
* 26: process assembler directive
* 28: end
* 29: false
* process assembler error (line number, invalid statement)
* endif
* return

******************************************************************************
PARSE THE REST OF THE LINE

REF

PAGE

**********************************************************************************************************
* *
* 1   DO CASE OF ADDRESSING FORMAT FROM INFO TABLE
* *
* 2   0:
* *
* 3   PARSE COMMENT ONLY
* *
* 4   1:
* *
* 5   PARSE REGISTER AND OPERAND
* *
* 6   2:
* *
* 7   PARSE REGISTER AND OPERAND
* *
* 8   3:
* *
* 9   PARSE REGISTER OR OPERAND
* *
* 10  4:
* *
* 11  PARSE REGISTER
* *
* 12  5-9:
* *
* 13  PARSE OPERAND ONLY
* *
* 14  ENDO
* *
* 15  RETURN
* *
**********************************************************************************************************
**PARSE REGISTER OR OPERAND**

<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>44</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>66</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

* * *

* * *
<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>66</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

* * *
PARSE REGISTER AND OPERAND

**FILE**

* 43  1  PARSE REGISTER ONLY
*  2  IF REGISTER WAS FOUND
* 44  3  PARSE OPERAND ONLY
*  4  IF (OPERAND) NOT FOUND
* 66  5  PROCESS ASSEMBLER ERROR(MISSING OPERAND)
*  6  ENDIF
*  7  ELSE
* 66  8  PROCESS ASSEMBLER ERROR(MISSING OPERAND)
*  9  ENDIF
* 42 10  PARSE COMMENT ONLY
* 11  RETURN
*
**PARSE COMMENT ONLY**

**REF**

**PAGE**

想法：

* 1  IF <POINTER> NE END OF LINE MARKER

* 2  SIGNAL COMMENT IN LINE

* 3  SAVE POINTER TO START OF COMMENT FIELD

* 4  ENDIF

想法：
PARSE REGISTER ONLY

REF

PAGE

*****************************************************************************
* *
* 1 IF <POINTER> IS A SPACE *
* 2 INCREMENT POINTER *
* 3 ENDIF *
* 4 IF <POINTER> IS AN A OR B *
* 5 IF <POINTER+1> IS A SPACE *
* 6 SIGNAL REGISTER A OR B MODE *
* 7 SAVE POINTER *
* 8 ENDIF *
* 9 ENDIF *
* 10 RETURN *
* *
*****************************************************************************
PARSE OPERAND ONLY

REF PAGE *******************************************************************************************************************
* *
* 1 IF <POINTER>IS #                                      * *
* 2 INCREMENT POINTER                                     * *
* 3 SIGNAL IMMEDIATE OPERAND MODE                         * *
* 4 ENDIF                                                * *
* 5 CLEAR OPERAND LENGTH                                  * *
* 6 START OF OPERAND FIELD = POINTER                      * *
* 7 UNTIL <POINTER> = SPACE,COMMA, OR END OF LINE MARKER  * *
* 8 LOOK AT NEXT CHARACTER                                * *
* 9 INCREMENT OPERAND LENGTH                              * *
* 10 ENDDU                                               * *
* 11 IF A COMMA WAS FOUND FOLLOWED BY X                   * *
* 12 SIGNAL INDEXED MODE                                  * *
* 13 POINTER = POINTER + 2                                * *
* 14 ENDDF                                               * *
* 15 RETURN                                              * *
* *******************************************************************************************************************
**PROCESS ASSEMBLER DIRECTIVE**

<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
<th>***********************</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>* 1 DO CASE OF BASE OPCODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* 2 1:</td>
</tr>
<tr>
<td>59</td>
<td>3</td>
<td>PROCESS END DIRECTIVE</td>
</tr>
<tr>
<td></td>
<td>4 2:</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>5</td>
<td>PROCESS FCB DIRECTIVE</td>
</tr>
<tr>
<td></td>
<td>6 3:</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>7</td>
<td>PROCESS FDB DIRECTIVE</td>
</tr>
<tr>
<td></td>
<td>8 4:</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>9</td>
<td>PROCESS ORG DIRECTIVE</td>
</tr>
<tr>
<td></td>
<td>10 5:</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>11</td>
<td>PROCESS RMB DIRECTIVE</td>
</tr>
<tr>
<td></td>
<td>12 6:</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>13</td>
<td>PROCESS EQU DIRECTIVE</td>
</tr>
<tr>
<td></td>
<td>14 7:</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>15</td>
<td>PROCESS FCC DIRECTIVE</td>
</tr>
<tr>
<td></td>
<td>16 ENDU</td>
<td></td>
</tr>
</tbody>
</table>
PROCESS SINGLE BYTE INSTRUCTIONS

REF PAGE

* NUMBERS OF BYTES IN THIS INSTRUCTION = 1
* CURRENT LINE OPCODE = BASE OPCODE
* RETURN

*
<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NUMER OF BYTES IN CURRENT INSTRUCTION = 2</td>
</tr>
<tr>
<td>2</td>
<td>( BASE OPCODE IS IMMEDIATE, A REG)</td>
</tr>
<tr>
<td>3</td>
<td>DIRECT OFFSET = 10H</td>
</tr>
<tr>
<td>4</td>
<td>EXTENDED OFFSET = 30H</td>
</tr>
<tr>
<td>5</td>
<td>INDEXED OFFSET = 20H</td>
</tr>
<tr>
<td>6</td>
<td>0 REGISTER OFFSET = 40H</td>
</tr>
<tr>
<td>7</td>
<td>A REGISTER OFFSET = 00H</td>
</tr>
<tr>
<td>8</td>
<td>CHECK OUAL REGISTER MODE</td>
</tr>
<tr>
<td>9</td>
<td>CHECK IMMEDIATE MODE</td>
</tr>
<tr>
<td>10</td>
<td>IF CURRENT LINE ADDRESSING MODE NE IMMEDIATE</td>
</tr>
<tr>
<td>11</td>
<td>CHECK INDEXED MODE</td>
</tr>
<tr>
<td>12</td>
<td>IF CURRENT LINE ADDRESSING MODE NE INDEXED</td>
</tr>
<tr>
<td>13</td>
<td>CHECK DIRECT OR EXTENDED</td>
</tr>
<tr>
<td>14</td>
<td>ENDF</td>
</tr>
<tr>
<td>15</td>
<td>ENDF</td>
</tr>
<tr>
<td>16</td>
<td>RETURN</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PROCESS TABLE A-2 INSTRUCTION

<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>3</td>
</tr>
<tr>
<td>*</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>5</td>
</tr>
<tr>
<td>55</td>
<td>7</td>
</tr>
<tr>
<td>57</td>
<td>8</td>
</tr>
<tr>
<td>58</td>
<td>10</td>
</tr>
<tr>
<td>*</td>
<td>11</td>
</tr>
<tr>
<td>*</td>
<td>12</td>
</tr>
</tbody>
</table>

* NUMBER OF BYTES IN CURRENT INSTRUCTION = 2
* (BASE OPCODE IS DIRECT MODE A REG.)
* EXTENDED OFFSET = 20H
* INDEXED OFFSET = 10H
* B REGISTER OFFSET = 40H
* A REGISTER OFFSET = 00H
* CHECK DUAL REGISTER MODE
* CHECK INDEXED MODE
* IF CURRENT LINE ADDRESSING MODE NE INDEXED
* CHECK DIRECT OR EXTENDED
* ENDIF
* RETURN
**PROCESS TABLE A-3 INSTRUCTION**

<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NUMBER OF BYTES IN CURRENT INSTRUCTION = 1</td>
</tr>
<tr>
<td>2</td>
<td>( BASE OPCODE IS REGISTER A )</td>
</tr>
<tr>
<td>3</td>
<td>A REGISTER OFFSET = 00H</td>
</tr>
<tr>
<td>4</td>
<td>B REGISTER OFFSET = 10H</td>
</tr>
<tr>
<td>5</td>
<td>EXTENDED OFFSET = 30H</td>
</tr>
<tr>
<td>6</td>
<td>INDEXED OFFSET = 20H</td>
</tr>
<tr>
<td>7</td>
<td>CHECK DUAL REGISTER MODE</td>
</tr>
<tr>
<td>8</td>
<td>IF CURRENT LINE ADDRESSING MODE NE REGISTER</td>
</tr>
<tr>
<td>9</td>
<td>INCREMENT NUMBER OF BYTES IN CURRENT INSTRUCTION</td>
</tr>
<tr>
<td>10</td>
<td>CHECK INDEXED MODE</td>
</tr>
<tr>
<td>11</td>
<td>IF CURRENT LINE ADDRESSING MODE NE INDEXED</td>
</tr>
<tr>
<td>12</td>
<td>INCREMENT NUMBER OF BYTES IN CURRENT INSTRUCTION</td>
</tr>
<tr>
<td>13</td>
<td>CURRENT OPCODE=CURRENT OPCODE + EXTENDED OFFSET</td>
</tr>
<tr>
<td>14</td>
<td>ENDP</td>
</tr>
<tr>
<td>15</td>
<td>ENDP</td>
</tr>
<tr>
<td>16</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

******************************************************************************
PROCESS TABLE A-5 INSTRUCTION

* REF PAGE

* 1 NUMBER OF BYTES IN CURRENT INSTRUCTION = 2
* 2 ( BASE DPCODE IS IMMEDIATE )
* 3 IMMEDIATE OFFSET = 00H
* 4 DIRECT OFFSET = 10H
* 5 EXTENDED OFFSET = 30H
* 6 INDEXED OFFSET = 20H
* 7 CHECK IMMEDIATE MODE
* 8 IF CURRENT LINE ADDRESSING MODE NE IMMEDIATE
* 9 CHECK INDEXED MODE
* 10 IF CURRENT LINE ADDRESSING MODE NE INDEXED
* 11 CHECK DIRECT OR EXTENDED
* 12 ENDIF
* 13 ENDIF
* 14 RETURN

******************************************************************************
<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>***</td>
</tr>
<tr>
<td>*</td>
<td>( BASE UPCODE IS DIRECT )</td>
</tr>
<tr>
<td>*</td>
<td>2 NUMEROFBYTESINCURRENT INSTRUCTION = 2</td>
</tr>
<tr>
<td>*</td>
<td>3 DIRECT OFFSET = 00H</td>
</tr>
<tr>
<td>*</td>
<td>4 EXTENDED OFFSET = 20H</td>
</tr>
<tr>
<td>*</td>
<td>5 INDEXED OFFSET = 10H</td>
</tr>
<tr>
<td>57</td>
<td>6 CHECK INDEXED MODE</td>
</tr>
<tr>
<td>*</td>
<td>7 IF CURRENT LINE ADDRESSING MODE NE INDEXED</td>
</tr>
<tr>
<td>58</td>
<td>8 CHECK DIRECT OR EXTENDED</td>
</tr>
<tr>
<td>*</td>
<td>9 ENDF</td>
</tr>
<tr>
<td>*</td>
<td>10 RETURN</td>
</tr>
<tr>
<td>*</td>
<td>***</td>
</tr>
</tbody>
</table>
PROCESS TABLE A-7 INSTRUCTION

**Ref**

**Page**

```
* 1 (ONLY RELATIVE MODE APPLIES )
* 2 NUMBER OF BYTES IN CURRENT INSTRUCTION = 2
* 3 EXTENDED OFFSET = 10H
* 4 INDEXED OFFSET = 00H
* 5 ( INDEXED MODE IS BASE OPERAND )
57* 6 CHECK INDEXED MODE
* 7 IF CURRENT LINE ADDRESSING MODE NE INDEXED
* 8 INCREMENT NUMBER OF BYTES IN CURRENT INSTRUCTION
* 9 CURRENT LINE OPCODE = CURRENT LINE OPCODE + EXTENDED OFFSET
*10 ENDIF
*11 RETURN
*```

*----------------------------------------------------------------------------------------------------------------------------------*
**PROCESS TABLE A-8 INSTRUCTION**

<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

* * *
CHECK DUAL REGISTER MODE

**
* 1 IF B REGISTER MODE
* 2 CURRENT_OPCODE = CURRENT_OPCODE + B REGISTER OFFSET
* 3 SET CURRENT LINE REGISTER MODE FLAG
* 4 ELSE
* 5 IF A REGISTER MODE
* 6 SET CURRENT LINE REGISTER MODE FLAG
* 7 CURRENT_OPCODE=CURRENT_OPCODE + A REGISTER OFFSET
* 8 ENDIF
* 9 ENDF
* 10 RETURN
**

**************************************************************************************************
CHECK IMMEDIATE MODE

REF PAGE

***********************************************
* 1 IF IMMEDIATE MODE
* 2 SET CURRENT LINE IMMEDIATE MODE FLAG
* 3 CURRENT OPCODE=CURRENT OPCODE + IMMEDIATE OFFSET
* 4 ENDF
* .5 RETURN
*
***********************************************
CHECK INDEXED MODE

* 1 IF INDEXED MODE
* 2 CURRENT_OPCODE = CURRENT_OPCODE + INDEXED_OFFSET
* 3 SET CURRENT LINE INDEXED MODE FLAG
* 4 ENDIF
* 5 RETURN
*
CHECK DIRECT OR EXTENDED

REF PAGE

* 1 IF OPERAND IS IN BOTTOM PAGE OF MEMORY AND IS NOT DEFINED NOT PAGE ZERO
* 2 CURRENT OPCODE = CURRENT OPCODE + DIRECT OFFSET
* 3 ELSE
* 4 CURRENT OPCODE = CURRENT OPCODE + EXTENDED OFFSET
* 5 INCREMENT NUMBER OF BYTES IN CURRENT INSTRUCTION
* 6 ENOIF
* 7 RETURN

******************************************************************************
PROCESS END DIRECTIVE

<table>
<thead>
<tr>
<th>REF</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 1</td>
</tr>
</tbody>
</table>

CLEAN ERROR CAUSED BY NULL OPERAND FIELD
PROCESS EWU DIRECTIVE

REF
PAGE

**************************************************************************************************
* *
* 1 NUMBER OF BYTES IN THIS INSTRUCTION=0 *
* 2 LABEL VALUE = OPERAND VALUE *
* 3 LABEL STATUS = OPERAND STATUS *
* 4 RETURN *
* *
**************************************************************************************************
PROCESS FOB DIRECTIVE

REF PAGE

****************************

* 1 NUMBER OF BYTES IN THIS INSTRUCTION = 2
* 2 BASE UP CODE = MSB CURRENT LINE OPERAND
* 3 RETURN

****************************
**PROCESS FCC DIRECTIVE**

**REF**

**PAGE**

`***************************************************************`

`*`  
`*  1  SIGNAL FCC DIRECTIVE`  
`*  2  OPERAND VALUE = STRING START ADDRESS + 1`  
`*  3  NUMBER OF BYTES IN THIS INSTRUCTION = OPERAND LENGTH-2`  
`*  4  RETURN`  
`*`  

`***************************************************************`
PROCESS ORG DIRECTIVE

Page
1  PROGRAM COUNTER = OPERAND VALUE
2  RETURN

**************************************************************************
PROCESS KMA Directive

REF
PAGE

1  NUMBER OF BYTES IN THIS INSTRUCTION = OPERAND VALUE
2  RETURN

**************

-----------------
PROCESS ASSEMBLER ERROR

RLF
PAGE *****************************************************

* *
* 1 ENTER ERROR CODE PASSED TO CURRENT LINE ERROR *
* 2 INCREMENT ERROR COUNT *
* 3 RETURN *
* *
 *****************************************************
**ENTER THE LABEL IN THE SYMBOL TABLE**

```assembly
* 1 SEARCH THE SYMBOL TABLE FOR NEW SYMBOL
* 2 IF NEW SYMBOL IS ALREADY IN TABLE
* 3 IF PRESENT STATUS IS DEFINED
* 4 IF NEW VALUE NE TO OLD VALUE
* 5 CHANGE SYMBOL STATUS TO DOUBLE DEFINED
* 6 ENDIF
* 7 ELSE
* 8 IF THE SYMBOL HAS NOT BEEN USED
* 9 IF THE SYMBOL IS DEFINED NOW
* 10 IF THE VALUE IS ON PAGE ZERO
* 11 CHANGE STATUS TO DEFINED PAGE ZERO
* 12 ELSE
* 13 CHANGE STATUS TO DEFINED NOT PAGE ZERO
* 14 ENDIF
* 15 ELSE
* 16 CHANGE STATUS TO UNDEFINED BUT NOT USED FIRST
* 17 ENDIF
* 18 ENDIF
* 19 ENDIF
* 20 ELSE
* 21 PUT THE NEW SYMBOL INTO THE TABLE
* 22 IF SYMBOL IS DEFINED NOW
* 23 IF THE VALUE IS ON PAGE ZERO
* 24 CHANGE SYMBOL STATUS TO DEFINED PAGE ZERO
* 25 ELSE
* 26 CHANGE SYMBOL STATUS TO DEFINED NOT PAGE ZERO
* 27 ENDIF
* 28 ELSE
* 29 CHANGE SYMBOL STATUS TO UNDEFINED NOT USED FIRST
* 30 ENDIF
* 31 ENDIF
* 32 RET
```

* ************************************************************************************
RETRIEVE VALUE FROM SYMBOL TABLE

* 1 SEARCH SYMBOL TABLE FOR NEW SYMBOL
* 2 IF THE SYMBOL IS IN THE TABLE
* 3 RETURNED SYMBOL STATUS IS STATUS FROM TABLE
* 4 IF RETURNED SYMBOL STATUS IS UNDEFINED NOT USED
* 5 CHANGE TO UNDEFINED BUT USED
* 6 ENDF
* 7 ELSE
* 8 ENTER THE LABEL IN THE SYMBOL TABLE
* 9 SYMBOL STATUS IS UNDEFINED BUT USED
* 10 ENDF
* 11 RETURN

********************************************************************************************************************
EVALUATE OPERAND

1. IF FIRST CHARACTER IS / (FCC OR)
2. OPERAND VALUE = OPERAND START ADDRESS
3. ELSE
4. IF FIRST CHARACTER IS #
5. SET IMMEDIATE MODE FLAG
6. ADVANCE CHARACTER POINTER
7. ENDIF
8. IF FIRST CHARACTER IS * 
9. SIGNAL PROGRAM COUNTER MODE
10. OPERAND VALUE = PROGRAM COUNTER
11. ADVANCE OPERAND POINTER
12. ENDIF
13. SCAN CURRENT LINE OPERAND FOR MATH OPERATOR (+ - * /)
14. IF ANY MATH OPS ARE FOUND
15. EVALUATE EXPRESSION
16. ELSE
17. EVALUATE CHARACTER STRING
18. ENDIF
19. ENDIF
20. RETURN

*******************************************************************************
EVALUATE CHARACTER STRING

REF

PAGE

*****************************************************************************

72 * 1 EVALUATE CONSTANT
* 2 IF UNSUCCESSFUL
68 * 3 IMITATE VALUE FROM SYMBOL TABLE
* 4 ENDIF
* 5 RETURN
*
*****************************************************************************
EVALUATE EXPRESSION

* 1 EXPRESSION RESULT = 0
* 2 EXPRESSION POINTER = START OF EXPRESSION FIELD POINTER
* 3 ARITHMETIC OPERAND = +
* 4 UNTIL EXPRESSION POINTER >= END OF EXPRESSION POINTER
* 5 IF <EXPRESSION POINTER> = - (UNARY MINUS OPERATOR)
* 6 SIGNAL THIS ELEMENT MINUS
* 7 INCREMENT EXPRESSION POINTER
* 8 ELSE
* 9 SIGNAL THIS ELEMENT PLUS
* 10 ENDF
* 11 DO A SCAN FROM EXPRESSION POINTER UNTIL (+ - / *) IS FOUND
* 12 SAVE CHARACTERS AS STRING TO EVALUATE
* 13 ENDO
* 70 EVALUATE CHARACTER STRING
* 14 IF THIS ELEMENT MINUS
* 15 TAKE 2'S COMPLEMENT
* 16 ENDF
* 17 DE CASE OF ARITHMETIC OPERAND
* 18 +:
* 19 ADD ELEMENT VALUE TO EXPRESSION RESULT
* 20 -:
* 21 SUBTRACT ELEMENT VALUE FROM EXPRESSION RESULT
* 22 *:
* 23 MULTIPLY EXPRESSION RESULT BY ELEMENT VALUE
* 24 /:
* 25 DIVIDE EXPRESSION RESULT BY ELEMENT VALUE
* 26 ENDO
* 27 ARITHMETIC OPERAND = <EXPRESSION POINTER>
* 28 INCREMENT EXPRESSION POINTER
* 29 ENDO
* 30 RETURN
*
EVALUATE CONSTANT

* 1  0) CASE OF FIRST CHARACTER
* 2  $:
* 3      EVALUATE HEX CONSTANT
* 4  q):
* 5      EVALUATE OCTAL CONSTANT
* 6 PERCENT SIGN:
* 7      EVALUATE BINARY CONSTANT
* 8  ':
* 9      EVALUATE ASCII CONSTANT
*10  0123456789:
*11  0) CASE OF LAST CHARACTER
*12  H:
*13      EVALUATE HEX CONSTANT
*14  0:
*15      EVALUATE OCTAL CONSTANT
*16  U:
*17      EVALUATE OCTAL CONSTANT
*18  B:
*19      EVALUATE BINARY CONSTANT
*20  DONT CARE:
*21      EVALUATE DECIMAL CONSTANT
*22  ENDO:
*23  IF NONE OF ABOVE ARE SATISFIED
*24  SIGNAL CONSTANT EVALUATION NOT SUCCESSFUL
*25  ENDF:
*26  ENDU
*27  RETURN

**************************************************************************
* 1  IF "ERRORS ONLY" NOT SELECTED OR IF ERROR THIS LINE
* 2  PUT LINE NUMBER IN OUTPUT BUFFER
* 3  PUT COMMENT LINE ERROR OR SPACE IN OUTPUT BUFFER
* 4  IF THIS LINE IS NOT A COMMENT
* 5  PUT VALUE OF PROGRAM COUNTER IN OUTPUT BUFFER
* 6  IF CURRENT LINE INSTRUCTION IS NOT FCC
* 7  IF NUMBER OF BYTES IN INSTRUCTION > 0
* 8  STORE OPCODE AND OPERAND DATA IN OUTPUT BUFFER
* 9  ENOIF
* 10  STORE CURRENT LINE LABEL IN OUTPUT BUFFER
* 11  ENOIF
* 12  STORE CURRENT LINE MNEMONIC IN OUTPUT BUFFER
* 13  IF REGISTER MUXE OPERAND
* 14  LOAD OUTPUT BUFFER WITH "A" OR "B"
* 15  ENOIF
* 16  IF IMMEDIATE OPERAND MODE
* 17  LOAD OUTPUT BUFFER WITH "A"
* 18  ENOIF
* 19  IF THERE IS AN OPERAND
* 20  LOAD OUTPUT BUFFER WITH OPERAND CHARACTERS
* 21  ENIF
* 22  IF INDEXED ADDRESSING MODE
* 23  LOAD ",X" IN OUTPUT BUFFER
* 24  ENOIF
* 25  ENOIF
* 26  IF THERE IS A COMMENT OR IF THE WHOLE LINE IS A COMMENT
* 27  LOAD COMMENT CHARACTERS INTO OUTPUT BUFFER
* 28  ENIF
* 29  OUTPUT A LINE TO THE CONSOLE
* 30  ENOIF
* 31  RETURN
*
**OUTPUT BINARY INFO**

```plaintext
1  IF NUMBER OF BYTES IN THIS INSTRUCTION > 0
2      IF THIS INSTRUCTION IS NOT AN FCC DIRECTIVE
3          <BINARY DESTINATION> = CURRENT OPCODE
4          INCREMENT BINARY DESTINATION
5          IF NUMBER OF BYTES IN THIS INSTRUCTION > 2
6                <BINARY DESTINATION> = MS BYTE OF OPERAND VALUE
7                INCREMENT BINARY DESTINATION
8          ENDF
9          IF NUMBER OF BYTES IN THIS INSTRUCTION > 1
10                 <BINARY DESTINATION> = LS BYTE OF OPERAND VALUE
11                 INCREMENT BINARY DESTINATION
12          ENDF
13 ELSE
14       DO FOR THE NUMBER OF BYTES THIS INSTRUCTION
15                 <BINARY DESTINATION> = <OPERAND POINTER>
16                 INCREMENT BINARY DESTINATION
17       OUTPUT <OPERAND POINTER> TO THE CONSOLE
18                 INCREMENT OPERAND POINTER
19       ENDDO
20          ENDF
21      ENDF
22      RETURN
```

*******************************************************************************
CONVERT FROM BINARY TO ASCII(BINARY ADDRESS, ASCII ADDRESS)

REF

PAGE

******************************************************************************

* 1  DO I = 1 TO 4
* 2  DO UNTIL BINARY ADDRESS GOES NEGATIVE
* 3  BINARY ADDRESS = BINARY ADDRESS - 16**(I-1)
* 4  ASCII ADDRESS(I) = (I-1)TH HEX DIGIT
* 5  ENDO
* 6  BINARY ADDRESS = BINARY ADDRESS + 16**(I-1)
* 7  ENDD
* 8  RETURN

******************************************************************************
CONVERT ASCII TO BINARY

REF PAGE
**********
* 1 CLEAN PARTIAL RESULT
* 2 DO NUMBER OF ASCII DIGITS
* 3 LOOK AT NEXT DIGIT
* 4 IF IT IS ASCII 1
* 5 CARRY = 1
* 6 ELSE
* 7 CARRY = 0
* 8 ENUIF
* 9 SHIFT CARRY INTO PARTIAL RESULT
* 10 ENDO
* 11 RETURN

* *******************************
CONVERT ASCII DIGIT TO BINARY (HEX, DECIMAL, OCTAL)

```
* 1 LOAD DIGIT WEIGHTS WITH VALUES FOR SELECTED BASE
* 2 CLEAR PARTIAL RESULT
* 3 DD FOR EACH DIGIT IN THE NUMBER
* 4       IF THE DIGIT > 9
* 5       DIGIT = DIGIT - 7
* 6       ENDIF
* 7       DD UNTIL ASCII ZERO IS REACHED
* 8       ADD DIGIT WEIGHT TO PARTIAL RESULT
* 9       DECREMENT ASCII DIGIT
* 10      ENDU
* 11     ENDD
* 12    RETURN
```

*******************************************************************************
CONVERT BINARY TO ASCII DECIMAL

***
* 1  DO FOR THE NUMBER OF DIGITS DESIRED
* 2  LOAD ASCII DIGIT WITH ASCII ZERO
* 3  DO UNTIL BINARY VALUE UNDERFLOWS
* 4   BINARY VALUE = BINARY VALUE - DIGIT WEIGHT
* 5   INCREMENT ASCII DIGIT
* 6   ENDDO
* 7   BINARY VALUE = BINARY VALUE + DIGIT WEIGHT
* 8   ENDDO RETURN

***