General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Produced by the NASA Center for Aerospace Information (CASI)
AN ASSEMBLER FOR THE MOS TECHNOLOGY 6502 MICROPROCESSOR AS IMPLEMENTED IN JOLT (TM) AND KIM-1 (TM)

The 6502 Assembler implemented at Ohio University for support of microprocessor program development in the Tri-University Program is described.

by

Robert W. Lilley
Avionics Engineering Center
Department of Electrical Engineering
Ohio University
Athens, Ohio 45701

November 1976

Supported by

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia
Grant NGR 33-009-017
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II</td>
<td>MOS TECHNOLOGY 6502 MICROPROCESSOR INSTRUCTIONS</td>
<td>1</td>
</tr>
<tr>
<td>III</td>
<td>6502 ASSEMBLER LANGUAGE</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>A. Assembler Source Statement Format</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>B. Assembler Instructions (Pseudo-Opcodes)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>C. Example Statements</td>
<td>5</td>
</tr>
<tr>
<td>IV</td>
<td>USE OF THE 6502 ASSEMBLER IN THE CMS ENVIRONMENT</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>A. Assembler Operation</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>B. Suggested Usage of the Assembler and Files</td>
<td>7</td>
</tr>
<tr>
<td>V</td>
<td>ASSEMBLER MAINTENANCE SUPPORT UNDER CMS</td>
<td>10</td>
</tr>
<tr>
<td>VI</td>
<td>ACKNOWLEDGMENTS</td>
<td>14</td>
</tr>
<tr>
<td>VII</td>
<td>REFERENCES</td>
<td>14</td>
</tr>
<tr>
<td>VIII</td>
<td>APPENDICES</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>A. Assembler Source Listing</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>B. CMS Exec Procedure Listings</td>
<td>36</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

The development of computer software for microprocessors is materially aided by the assembler program; such programs generally allow the use of mnemonic variable names instead of absolute addressing and they will compute relative addresses for branching instructions and other useful functions. The programmer's task can then emphasize program content rather than the specific forms required by the target microprocessor instruction set.

The NASA Tri-University team at Ohio University is designing low-cost, microcomputer-based navigation receivers, and the assembler described in this paper was implemented for support of this project effort.

Ohio University provides computer services to its departments from a central site utilizing remote communication terminals. The flexibility of the environment provided by IBM's Virtual Machine Facility [1] and the Conversational Monitor System [2,3] make possible the convenient assembler access described herein.

This implementation of the assembler for the MOS Technology 6502 microprocessor chip serves a part of the present need; it forms a model for support of other microprocessors, for which we expect to have applications in the future.

The 6502 Assembler is in current use for development of Omega navigation software for the Ohio University Software-Based Receiver, the developmental models of which use the JOLT (TM) and KIM-1 (TM) microcomputer hardware.

II. MOS TECHNOLOGY 6502 MICROPROCESSOR INSTRUCTIONS

The MOS Technology 6502 Microprocessing Unit (MPU) integrated circuit chip is used in the JOLT and KIM-1 microcomputer units, in combination with appropriate read-only-memory and random-access memory chips. The MPU chip has some 55 unique operations, each of which may be performed upon data in a variety of ways. Some thirteen addressing modes allow flexibility in applying the 6502's basic logical operations to data.

Figure 1 reproduces the JOLT microcomputer reference data for addressing modes and instructions. The KIM-1 data are identical.

The complete descriptions of the 6502 addressing modes and instructions are contained in the JOLT and KIM-1 literature (see references 4, 5, 6); they will not be repeated here. The 6502 Assembler accepts the mnemonic forms of all 6502 instructions as operation codes. Addressing modes are determined by the type of assembler operand entered with the operation code. Assembler statements are described in Section III of this paper.
### Table: JOLT or KIM-1 Operation Codes and Addressing Modes Summary

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
<th>Immediate</th>
<th>Absolute</th>
<th>Register</th>
<th>Index</th>
<th>Address Mode</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>A = B + C</td>
<td>I</td>
<td>A</td>
<td>B, C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>A = B - C</td>
<td>I</td>
<td>A</td>
<td>B, C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>A = B</td>
<td>I</td>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>A = A + 1</td>
<td>I</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>A = A - 1</td>
<td>I</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STR</td>
<td>A = B</td>
<td>I</td>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>A = A</td>
<td>I</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Condition Codes**
- **N**: Negative
- **Z**: Zero
- **C**: Carry
- **V**: Overflow

---

**Figure 1. JOLT or KIM-1 Operation Codes and Addressing Modes Summary**
This description of the 6502 Assembler Language provides a user's guide to writing assembler statements for the MOS Technology 6502 microcomputer chip instruction set. The assembler allows selection of basic 6502 operation codes (opcode) or certain additional operations defined by the assembler which aid the programmer in establishing data areas, constants and program flow control. This description is heavily dependent on Rankin's Cross-Assembler Manual. [7]

Assembler input consists of a group of assembler statements followed by an END assembler instruction. Outputs consist of a "clean" (formatted) version of the input statements, error messages as appropriate, and a hexadecimal output file containing program object code in a format readable by JOLT or KIM-1 hex program loaders.

A. Assembler Source Statement Format. A source line has four parts: a label field, an opcode field, an operand field, and a comment field. The fields are defined by one to any number of spaces separating them. A total line is up to 72 characters long.

Label - A label must start in the first position in the line. If the first position is blank, no label is assumed. A label must start with an alphabetic character and can be 1 to 6 characters in length. All base page labels must appear before their first use as an operand.

"*" in the first position of the line indicates that the line is to be taken as a comment.

Opcode - The opcode is three characters long always proceeded by at least one space. Opcodes are of two forms. The first form causes the creation of a machine instruction (1 to 3 bytes) and is known as a machine operation code (opcode). The second form is either a control statement to the assembler, or it defines constants, addresses, or symbols and is known as pseudo-opcode. Each pseudo-opcode will be explained below.

Operand - Most, although not all, opcodes require an operand for additional information, such as an address. An operand is preceded by, and terminated by, a space. The exact form of the operand determines in part which addressing mode is to be used.

Comments - After the operand (or opcode if no operand required) and a trailing space, the remainder of the line can be devoted to comments and is not processed by the assembler.

1. Addressing Modes. The 6502 has 13 addressing modes. Consult the summary sheet, Figure 1, for which instructions use which modes, and for mnemonics. The following paragraphs give addressing modes and operands required.

- Implied - Any instruction which uses implied addressing does not need an operand.

- Immediate - This mode places the operand value as the second byte of the assembled instruction. The operand has the form =Label[+/- #], = #, or =' &'. Where [...] is optional,
and \( \pm \) means either + or -. \# is a number which has the following prefixes:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Decimal</td>
</tr>
<tr>
<td>&amp;</td>
<td>Octal</td>
</tr>
<tr>
<td>$</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>%</td>
<td>Binary</td>
</tr>
</tbody>
</table>

Relative - Relative addressing is used for branching statements and has two forms. In the first form, the operand is label \([\pm/\#]\) or \#. In this form the operand must be within +127 to -128 of the current program counter (when pointing to the start of the next instruction). In the second form the offset is given by \(\pm/\#\) where \(\pm/\#\) must be in the range +127 to -128 and is the relative jump from the start of the next instruction.

Absolute - Absolute addressing creates 3 bytes of machine code (1 for the instruction and 2 for address). It has the form label \(\pm/\#\), \#, or \(\ast [\pm/\#]\), where \(\ast\) is the value of the program counter.

Zero Page - Zero-page addressing is same as absolute except that the address has a value of less than 256. Note that labels appearing in the operand must be previously defined or improper assembly will result.

Accumulator - Operand is A. A is a reserved symbol and cannot be used otherwise as a label.

Indexed - Indexed addressing can either be zero-page or absolute and use either index register (if allowed by the instruction). The form is Add, X or Add, Y where Add is an address formed as described for addresses in absolute and zero page addressing.

Indexed Indirect - Indexed indirect addressing uses only the X register. The operand is added to X which then points to a full address stored in the base page. Form is (Add, X) where Add is an address as described in zero page addressing.

Indirect Indexed - Y is added to the address in Add with carry added to address in Add+1. The fetch is obtained from the resulting memory location. Form is (Add), Y where Add has same form as above.

Indirect - Use only in the JMP instruction and has form (Add).

B. Assembler Instructions (Pseudo-Opcodes).

- **ORG \#** - Sets program counter to \#
- **END** - Ends Assembly

**Label EQU M** - Assigns the value of M to label rather than the value of the program counter. M may contain a simple expression consisting of Label \([\pm/\#]\), or \#. M must be positive and label previously defined. No code is generated.
[Label] ADR Lab [+/−] - Places value of operand into memory. If both operand and PC are less than 256, 1 byte of code is created, otherwise 2 bytes. Normally the operand is an address.

[Label] ASC@@@ - Stores up to 40 ASCII characters in memory. 1 byte of code is generated for each character.

[Label] OCT Num [, Num...] - Stores in memory up to 40 octal numbers where Num is 1 to 40 Octal numbers separated by commas. Each number results in 2 bytes of code. If number has prefix of - , the 2 S complement of the number is stored. Base prefixes must not be used in Num.

[Label] HEX Num [, Num...] - Same as OCT except that Num is in HEX and 1 byte is generated for each number.

[Label] DCM Num [, Num...] - Same as OCT except Num is in decimal.

[Label] INT Num [, Num...] - Same as DCM except 1 byte is generated for each number.

[Label] BCD Num [, Num...] - Same as INT except numbers are stored in BCD and negative numbers stored as 10's complement.

C. Example Statements.

1 * This program is to demonstrate addressing modes and does not represent an actual program
2 *
3 0010 ORG $10
4 0010 Label1 BSS 10
5 001A 10 ADR Label1
6 0100 ORG 256
7 0100 J EQU 10
8 0100 31 Label2 ASC '123'
8 0101 32
8 0102 33
9 0103 00 OCT 177400, -1
9 FF
9 0105 FF
9 FF
10 0107 00 HEX 00, -00, AA
10 0108 00
10 0109 AA
11 010A 28 DCM 9000
11 23
12 010C 10 INT 16
13 010D 99 BCD 99, 00, 01
13 010E 00
13 010F 01
IV. USE OF THE 6502 ASSEMBLER IN THE CMS ENVIRONMENT

At Ohio University, central computer resources are available through use of remote terminals connected to an IBM System/370 Model 158 computer system running the Virtual Machine Facility (VM/370). The 6502 Assembler is stored on the Conversational Monitor System (CMS) virtual disk assigned to virtual machine AVENCTR. This disk is also available to virtual machine AVIONICS in read-only mode. In practice, AVENCTR access is used for assembler maintenance and disk storage. AVIONICS machine access is generally used for operation of the assembler to produce hex input tapes for the JOLT or KIM-1 microcomputers used in Omega navigation work.

The remainder of this section is devoted to describing the CMS environment in which the user accesses the 6502 assembler, and it presumes a working knowledge of CMS on the part of the reader. Assembler maintenance is discussed in Section V.

A. Assembler Operation. First, the user must establish communication with the VM/370 system. Either by dialing the telephone number for 110-baud or 300-baud operation, as appropriate for the terminal device, or by powering-up the terminal for direct-connected devices, the user will receive the "VM/370 ONLINE" message. Type a carriage return (CR) in reply. After the prompting dot, type LOGON AVIONICS (CR). The system will reply ENTER PASSWORD and type a mask to avoid unauthorized observation of the password. After the mask is typed, enter the password in current use for machine AVIONICS. The
system replies with the logon message, and ends with a prompting dot. The user is now ready to begin CMS programming.

The 6502 Assembler is capable of operation in various modes, with various requirements from the user. In all cases, the assembler requires source program input either from the terminal or from a CMS disk file. Terminal input is entered as user responses, line-by-line, to prompting dots produced by the assembler. CMS disk files are produced by use of the CMS Editor. The EDIT command begins the file-building process.

CMS files have compound names consisting of three parts. The user must enter all three parts to describe completely the file. For 6502 Assembler operation, the first two parts of the file description are arbitrary. The user may name his file anything he wishes as long as the name does not conflict with an existing file name in his CMS library. (The CMS "LISTFILE" command may be used to print library file names.) The third part of the file description is, for our purposes, either "A" or "C". This character tells which CMS "disk" will be used for file storage.

As the AVIONICS and AVENCTR machines are currently configured, the A-disk is permanent storage for CMS files. The disk will remain, even though the user logs off the machine, to return later. Upon LOGON, a C-disk is formed, as temporary storage for CMS file use in the current terminal session. This implies that additional storage space is available to the user for temporary files of his own, or for work files produced by the 6502 Assembler. The user can put files on the C-disk by using the character "C" as the third part of his file description, or by copying files from the A-disk. The 6502 Assembler will use the C-disk for all its work files to avoid crowding the A-disk (permanent file space) with temporary files which will no longer be needed after assembly is complete. For reasons which will become clear later, it is suggested that the user begin file construction on the C-disk, copying correct files to the A-disk when desired for permanent storage.

B. Suggested Usage of the Assembler and Files. After LOGON, the user has a choice of using the Assembler to assemble input files directly from the terminal, or from CMS input files. To assemble from the terminal, the user begins the process by entering

\texttt{JASM TERMINAL} \hspace{1cm} \text{(See Footnote 1)}

in response to a prompting dot from CMS. This command invokes the 6502 Assembler and replies "EXECUTION BEGINS..." and then issues a prompting dot. At this point, the user may type 6502 Assembler statements directly to Pass 1 of the Assembler. When the Assembler "END" statement is entered, the Assembler enters Pass 2 and produces an output listing on the terminal including any error messages necessary. When Pass 2 is completed, the following files will be found on the C-disk:

1 The \texttt{JASM} command, and others to be described later, is a CMS EXEC procedure, written for this application. CMS allows these pseudo-commands to be built and executed by the user. See Appendix B for EXEC listings.
JOLT CLEAN C - A listing of the user's input code, arranged in columnar form for ease in reading. This file could be copied to the A-disk (or to another C-disk file name) for retention and subsequent updating, if desired. In fact, it must be copied or renamed before another JASM command is issued or it will be replaced by another JOLT CLEAN C file from the latest assembly. If the file is renamed, even though it is left on C-disk, the new JOLT CLEAN C file from the next assembly will not replace it.

JOLT HEXCODE C - A file containing the hex "object code" in hextape format for reading by JOLT of KIM-1. To obtain this file on paper tape for input to the microcomputer, the user must output the file to the Model 33 Teletype (TM) unit using the CMS "TYPE" command. If the user is operating on another terminal type which has no paper tape capability, he must save the JOLT HEXCODE C file by copying it to the A-disk, logging off, and logging on later using the Model 33 terminal and issuing the TYPE command for the file name he assigned the file when he copied it.

For example:

```
LOGON AVIONICS
JASM TERMINAL
(entry of assembler statements)
COPY JOLT CLEAN C TEST PROG A
COPY JOLT HEXCODE C TEST HEX A
LOGOFF
```

Later, on the Model 33:

```
LOGON AVIONICS
TYPE TEST HEX A (with tape punch on before CR).
```

These commands produce the assembled hexcode file, which the user saved on the A-disk as TEST HEX A, and the cleaned-up input file, which the user saved as TEST PROG A before logoff. Later, when the Model 33 terminal with tape punch became available, the user logged on and typed the hex file with the punch turned on to produce a tape for entry to the JOLT of KIM-1.

In many cases, this JASM TERMINAL mode may serve the user's need; one disadvantage to this method exists, however. Note that input from the terminal goes directly to the Assembler. Therefore, input errors may not be corrected by the user before the assembly proceeds. The CMS system allows another operational mode which circumvents this disadvantage. By building a CMS file of Assembler input statements using the CMS Editor, the user can take advantage of this very powerful edit capability to correct known errors in his input prior to assembly. After LOGON, the sequence is as follows:

-8-
EDIT MYFILE TEST A

or  EDIT MYFILE TEST C

Where MYFILE and TEST are arbitrary names assigned to the file by the user, and the A or C denotes on which CMS disk the file will reside. It matters little whether the user specifies A or C at this point; remember, however, that any files to be retained after LOGOFF must be on the A-disk. The user then enters CMS input mode by issuing the INPUT command and begins typing Assmebler input statements according to the formats given earlier. In INPUT model, all typed input is stored, line-by-line, in the file MYFILE TEST. If an error is detected, the user may elect to finish typing input and then go back to correct it. If he desired to correct is when it occurred, he could enter EDIT mode with a null CR and use CMS edit commands to change the erroneous line, then re-enter INPUT mode to continue building the file.

When the file is complete, the user must enter EDIT mode with the null CR and store the file on disk using the CMS FILE command.

(Note: Use of the CMS AUTOSAVE command can aid in minimizing loss of data in case of a machine malfunction. See the CMS Command Guide.)

The user now has a 6502 Assembler input file stored on disk ready for assembly. He should now issue:

JASM MYFILE TEST A or JASM MYFILE TEST C

to retrieve the file and invoke the Assmebler. From this point on, assembly proceeds as above, with JOLT HEXCODE C and JOLT CLEAN C files built. A program listing is typed at the terminal with error messages as appropriate. The file copying considerations are the same as above, except that the user may want to replace his input file with the JOLT CLEAN C file to take advantage of the neat formatting performed by the Assmebler, facilitating later file update or correction. To do this, the user must issue:

COPY JOLT CLEAN C MYFILE TEST A (REPLACE)

This command takes the JOLT CLEAN FILE and replaces the user file (here assumed to be on disk A) with it. The user may also want to copy the JOLT HEXCODE C file for later use, if it is error-free and valuable.

One other option is allowed for assemblies from CMS input files. If the user is working with a large assembly, the time taken to type the listing on the terminal for each assembly may be prohibitive. Instead, the user may wish to keep a "master" listing on paper and update it by hand as program development proceeds. Then he can obtain a program listing only after a series of changes to the input file, or after a major change which renders his paper listing obsolete. To obtain an assembly without terminal listing he should issue:
JASM MYFILE TEST A NOLIST

Where, once again, MYFILE and TEST are arbitrary names assigned by the user, and A denotes on which CMS disk the file is residing. The NOLIST option prevents the printing of any terminal output during assembly. Instead, the listing goes into file JOLT PRINTOUT C for possible later reference.

The user may well want to scan this disk-resident printout file for errors after each NOLIST assembly. To do this, issue:

JERRS

The JERRS command causes a scan of the JOLT PRINTOUT C file and prints any error messages and the 6502 Assembler lines which generated them for the user's review. Note that JOLT PRINTOUT C is only produced when assembly is done using the NOLIST option on the JASM command. If JERRS is issued after a normal assembly, no PRINTOUT file will be found and no output of errors will be given.

The user may find it useful to refer to the flow chart of Figure 2 and to the sample terminal session of Figure 5 for additional information.

V. ASSEMBLER MAINTENANCE SUPPORT UNDER CMS

Maintenance of the 6502 Assembler generally is done using the AVENCTR virtual machine. The master copy of the Assembler is located on the A-disk as file ASM6502 FORTRAN AO, making it private to this machine. The file contains the FORTRAN source statements for the assembler (See Appendix D). A second copy is stored as ASM6502 PFORT R AN AO, in a packed format for backup in case of loss of the primary copy.

Maintenance usually takes the form of some update or alteration of the source code and then a series of tests to verify that the resulting Assembler operates properly. For this purpose, a set of maintenance EXEC procedures are provided in the AVENCTR A-disk library. MBUILD provides for compilation of the ASM6502 FORTRAN file into a MAINT MODULE C file which is then executed for testing. Later use of the MAINT MODULE C file is made by MASM, which works as does JASM, described earlier, except that the MAINT MODULE file is used instead of the operational JOLT MODULE A file.

Correct forms for these maintenance EXECs are:

- MBUILD TERMINAL
- MBUILD ASMBL TEST C
- MBUILD ASMBL TEST C NOLIST
- MASM TERMINAL
- MASM TEST FILE C
- MASM TEST FILE C NOLIST
Figure 2. 6502 Assembler Data Flow.
.logon avionics
ENTER PASSWORD:
..SSSSSSS
LOGON AT 13:47:00 EDT FRIDAY 11/05/76
CMS VERSION 3.0 - 10/13/76 06:36
.edit jolt test c
D (192) P/0
READY: 1-CYL C-DISK ONLINE
NEW FILE:
EDIT:
.input
INPUT:
.org $100
.lda label1
.sta label2
.brk
.label1 bss 1
.label2 bss 1
.end

EDIT:
.file
R;

.jasm jolt test c
EXECUTION BEGINS...

END PASS 1: 0 ERRORS

1 $100 ORG $100
2 $100 AD LDA LABEL1
07
01
3 $103 8D STA LABEL2
08
01
4 $106 20 BRK
5 $107 20 LABEL1 BSS 1
6 $108 20 LABEL2 BSS 1
7 $ END END PASS 2: 0 ERRORS

.F;
.type jolt hexcode c
;09120001070102010000000155
;00
.F;
.type jolt clean c

Figure 3. Sample Terminal Session.
Figure 3. Sample Terminal Session (Cont.).

-13-
Again, file names are arbitrary, and the same files are built as described earlier for hex codes and clean listings. To avoid interference with current operating modules, however, all these maintenance files have MAINT as the first part of the file name.

When maintenance and testing are complete, the operative MAINT MODULE C must be copied to replace the current version of JOLT MODULE C on the AVENCTR A-disk. Then, subsequent use of JASM on either AVIONICS or AVENCTR machines will result in assemblies using the updated Assembler.

VI. ACKNOWLEDGMENTS

The author acknowledges the origin of the 6502 Assembler with Roy R. Rankin at Stanford University, and the assistance of Mr. Lynn Smith at Microcomputer Associates, Inc., who provided a copy of this program, which appears in altered form in this paper.

Richard Salter, of the Ohio University Omega Team, has been the principal Assembler user and offered many suggestions for improvements in operation. Ralph W. Burhans is the Project Engineer for Ohio University's portion of the NASA Joint University Program, and Dr. Richard H. McFarland is Director of the Avionics Engineering Center. Their contributions are appreciated.

VII. REFERENCES


VIII. APPENDICES

A. Assembler Source Listing.

B. CMS Exec Procedure Listings.
   1. PROFILE
   2. JASM
   3. JERRS
   4. MASM
   5. MBUILD
IMPLICIT INTEGER*4 (A-Z)
C CROSS-ASSEMBLER FOR 6502 MICROPROCESSOR (REV. 1.0)
C
C ROY R. FANKIN
C STANFORD UNIVERSITY
C 4/5-497-1822
C DEC. 8, 1975
C
C SYSTEM/370 MODIFICATIONS - F. W. LILLEY, OHI0 UNIV. AVIONICS
C AUGUST, 1976
C
C THIS PROGRAM READS AN INPUT FILE CONTAINING THE ASSEMBLER CODE
C (LOGICAL UNIT 5, LUIN) ONCE FOR EACH OF THE
C PROGRAM'S TWO PASSES, AN ASSEMBLY LISTING IS OUTPUT TO THE LINE
C PRINTER (LOGICAL UNIT 6, LU) ALONG WITH AN ABSOLUTE FILE,
C READABLE BY JOLT'S DEMON OR MOS TECHNOLOGY'S TIM, TO THE PUNCH
C FILE (LOGICAL UNIT 4, LUP).
C
C FILE 11 IS A SEQUENTIAL SCRATCH FILE FOR STORING INPUT TEXT
C FOR PASS 2
C
C THE SYMBOL TABLE IS SET TO ALLOW 300 ENTRIES. TO ALLOW MORE ENTRIES
C SET THE DIMENSION OF ISYM(3,300) AND SYM(300) TO THE SIZE OF THE
C DESIRED SYMBOL TABLE LENGTH AND SET NSYTMC TO THE NUMBER OF
C ENTRIES. THIS NEED BE DONE ONLY IN THE MAIN PROGRAM.
C
C DIMENSION IN(40),LABEL(3),EQU(2),IADD(2),IPC(2),CRUF(40)
C INTEGER LU,LUR,LUIN
C REAL LC,CODE(3)
C DIMENSION POUT(30)
C DIMENSION ISYM(3,300),SYM(300)
C DATA STAR'/',BLANK'/',LU'/',LUIN'/',LUP'/',NSYTMC '/',NSYTMC'/'
C DATA MMAX/26/
C INTEGER GSK
C REAL ZIO
C VOR=0
C ONE=1
C NONE=-1.
C NITY=30
C WLOZ=2
C CSW=0
C GSK=11
C MN=0
C IPASS=1
C PASS=1
C NCP=0
C RC=0
C LINE=0
C
10 IF(LINE.EQ.0)LINE=-LINE
IF(DSW.EQ.1)GO TO 701
READ(LUIN,1)IN
CALL LENM(IN)
WRITE(GSK,1)IN
GO TO 702
701 READ(GSK,1)IN

ORIGINAL PAGE IS OF POOR QUALITY
702 ERROR=0  
LINE=LINE+1  
1 FORMAT(40A2)  
IBUF=0  
ICHAP=MASK2(SRE(IN(1)))  
C  
C  **COMMENTS**  
C  
IF(ICHAP.NE.18)GO TO 12  
IF(PASS.EQ.2)WRITE(LU,2)LINE,IN  
GO TO 10  
2 FORMAT(1X,14,11X,48A2)  
C  
C  LABEL?  
C  
12 IF(ICHAP.EQ.BLANK)GO TO 30  
C  
**PROCESS LABEL**  
C  
LABEL(1)=BLANK  
LABEL(2)=BLANK  
LABEL(3)=BLANK  
IPCL=6  
CALL GWORD(IN,ISUF4,ITY,LABEL,IPBL,IFLAG,ZRO)  
IF(PASS.EQ.2)GO TO 30  
IF(IFLAG.NE.-1)GO TO 20  
WRITE(LU,8)LINE,LABEL  
NERR=NERR+1  
8 FORMAT(1X,14,'*** LABEL TOO LONG; USE ',3A2,' *** ')  
C  
C  **PUT LABEL INTO SYMBOL TABLE**  
C  
20 CALL SYMT(LABEL,PC,ISYM,NSYT,LU,NSYT,SYMB,LINE,NERR)  
C  
**GET CPCCODE**  
C  
30 CALL CPCCD(IN,ISUF,BYTE,CODE,PASS,NSYT,ISYM,SYM,PC,IERROR,  
   AGBUF,JK)  
C  
**ERROR PRINT**  
C  
IF(IERROR.EQ.0.OR.PASS.EQ.1)GO TO 35  
NERR=NERR+1  
LINE=LINE  
GO TO(201,202,203,204,205,206,207,208,209,210,211),IERROR  
201 WRITE(LU,301)LINE  
301 FORMAT(1X,14,'*** LABEL UNDEFINED ***)  
GO TO 35  
202 WRITE(LU,302)LINE  
302 FORMAT(1X,14,'*** ILLEGAL CPCC CODE ***)  
GO TO 35  
203 WRITE(LU,303)LINE  
303 FORMAT(1X,14,'*** IMMEDIATE VALUE > 255 ***)  
GO TO 35  
204 WRITE(LU,304)LINE  
 
-18-
304 FORMAT(1X,14,1 *** ADDRESS OUTSIDE ADDRESS SPACE ***)
GO TO 36
305 WRITE(LU,305)LINE
306 FORMAT(1X,14,1 *** INVALID INDIRECT ADDRESSING ***)
GO TO 36
307 WRITE(LU,306)LINE
308 FORMAT(1X,14,1 *** INVALID RELATIVE ADDRESS ***)
GO TO 36
309 WRITE(LU,307)LINE
310 FORMAT(1X,14,1 *** ILLEGAL ADDRESSING MODE ***)
GO TO 36
311 WRITE(LU,311)LINE
312 FORMAT(1X,14,1 *** MORE THAN 40 ASCII CHARACTERS ***)
GO TO 36
313 IF(PASS,GT,1)GO TO 40
IF(JK,GT,1)PC=PC+BYTE*JK-BYTE
PC=PC+BYTE
GO TO 10
C
C PUNCH OUTPUT TAPE AND LISTING
C
40 IF(PASS,NE,IPASS)GO TO 100
IF(JK,GT,0)GO TO 60
CALL CONV1(IPC,TWO,IPF,MON,E,PC)
PC=PC+BYTE
IF(BYTE,GE,0,GR,BYTE,GT,3)GO TO 59
CALL CONV1(10PCD,CONE,IPF,MON,E,CODE(1))
WRITE(LU,3)LINE,IPC(2),IPC(1),10PCD,IN
3 FORMAT(1X,14,2X,2A2,1X,A2,2X,4142)
CALL CONV1(IADD,TWO,IPF,MON,E,CODE(2))
IF(KN,NE,0)GO TO 59
POUT(1)=IPC(2)
POUT(2)=IPC(1)
KN=2
59 POUT(VN+1)=10PCD
POUT(VN+2)=IADD(1)
POUT(VN+3)=IADD(2)
VN=VN+4BYTE
IF(VN,GT,VMAX)CALL PUNCH(PCUT,IN,LUP)
IF(BYTE,NE,1)GO TO 100
BYTE=BYTE-1
WRITE(LU,4)(IADD(1),1=1,BYTE)
4 FORMAT(12X,A2)
GO TO 10
95 WRITE(LU,7)LINE,IPC(2),IPC(1),IN
7 FORMAT(1X,14,2X,2A2,5X,4942)
IF(JK,LT,0,GR,BYTE,GT,3)GO TO 59
GO TO 10
56 IF(MN,GT,2)CALL PUNCH(PCUT,MN,LUF)
GO TO 10
60 CALL Conv1(IPC,TW,IPTR,MONE,PC)
IF(MN,GE,0)GO TO 63.
PCUT(1)=IPC(2)
PCUT(2)=IPC(1)
MN=2
23 PC=FC+BYTE
CODE(1)=OPBUF(KK)
CALL Conv1(IADD,BYTE,IPTR,MONE,CODE(1))
PCUT(MN+1)=IADD(1)
PCUT(MN+2)=IADD(2)
MN=MN+BYTE
IF(MN.GE.MNMAX)CALL PUNCH(PCUT,MN,LUF)
TP(KK,EQ,1)GO TO 62
WRITE (LU,9)IPC(2),IPC(1),IADD(1)
9 FORMAT(7X,2A2,1X,A2)
TP(BYTE,EQ,2)WRITE (LU,4)IADD(2)
61 CONTINUE
GO TO 10
52 WRITE (LU,3)LINE,IPC(2),IPC(1),IADD(1),IN
IF(BYTE,EQ,2)WRITE (LU,4)IADD(2)
GO TO 61
100 IF(PASS,EQ,3)WRITE (LU,2)LINE,IN
J=PASS-1
IF(LU,NE,1)WRITE (LU,75)J,NERR
75 FORMAT(/' END PASS ',I11,' ',1X,I13,' ERRORS'/)
IF(PASS,EQ,3)GO TO 76
LINE=C
NERR=0
PASS=2
PC=0
SEND DSK
DST=1
GO TO 10
76 IF(MN,GT,0)CALL PUNCH(PCUT,MN,LUF)
WRITE (LUP,13)
13 FORMAT('';00')
STOP
END

FUNCTION ISOLT(IPTR,IBUF)
IMPLICIT INTEGER*2 (A-Z)
INTEGER IPTR
C THIS FUNCTION ISOLATES THE IPTR'TH CHARACTER IN IBUF so
C THAT ISOLT CONTAINS THE CHARACTER AND A SPACE
DIMENSION IBUF(1)
DATA BLANK/" "/
IK=(IPTR+1)/2
IK=IPTR+1-2*IK
IF(IK+EQ,C)ISOLT=MASK2(SRT(IBUF(IK)))
IF(IK+EQ,C)ISOLT=MASK2(IBUF(IK))
RETURN
END

ORIGINAL PAGE IS OF POOR QUALITY
SUBROUTINE PROC(LABEL, LTH, NSYT, SYMB, ISYM, BYTE, PASS, VAL, 
* ERROR, MODE, PC)
IMPLICIT INTEGER*2 (A-Z)

THIS SUBROUTINE SortS Out THE VARIOUS ADDRESS MODES AND DETERMINES
THE VALUE OF THE OPERAND

LABEL OPERAND FIELD
LTH LENGTH OF OPERAND FIELD
NSYT NUMBER OF ENTRIES IN SYMBOL TABLE
SYMB, ISYM SYMBOL TABLE
BYTE LENGTH OF INSTRUCTION
PASS ASSEMBLER PASS NUMBER
VAL VALUE OF OPERAND
ERROR ERROR FLAG
MODE ADDRESSING MODE
PC PROGRAM COUNTER

DIMENSION LABEL(1), ISYM(3,1), ICF(3)
DATA AT/"/DCLLAF/=",DCENT/"/",PLUS/+",MINUS/-"/
DATA COMMA/"/",PLANK/","EQUA2/"/",PAREN/("/",A/A/
DATA X/X",Y/Y",CPAREN/1")/",QUOTE/1

REAL VAL, PC
DIMENSION SYMB(1)
ICHAR=ISCLT(1,LABEL)
IF(ICHAR.EQ.EQUAL)GO TO 10
IF(ICHAR.EQ.PAREN)GO TO 20
IF(LABEL(1).EQ.A)GO TO 30

ABS OR 2-PAGE ADDRESSING?

IPTR=0
CALL LABPR(LABEL, LTH, IPTR, VAL, NSYT, SYMB, ISYM, ERROR, PC)
IF(VA L.LT.65535.) ERROR=4
47 IF(VA L.GE.256)BYTE=3
47 IF(VA L.LT.256)BYTE=2
47 IF(ERROR.NE.0)BYTE=3
47 IF(PASS.EQ.1)RETURN
47 IF(IPTR.EQ.LTH)GO TO 50
47 IF((IPTR+2).*NE.LTH) ERROR=2
47 IF(ISCLT(IPTR+2,LABEL).EQ.X)GO TO 51
47 IF(ISCLT(IPTR+2,LABEL).EQ.Y)GO TO 52
ERROR=2

INDEX ADDRESS MODES

51 MODE=8
51 IF(VA L.LT.256.)MODE=7
51 RETURN
52 MODE=9
52 IF(VA L.LT.256.)MODE=11
52 RETURN

IMMEDIATE ADDRESSING
10 BYTE=2
10 IF(PASS.EQ.1)RETURN

ORIGINAL PAGE IS OF POOR QUALITY

-21-
IPTR = 1
ICHAR = ISLT(IPTR+1, LABEL)
IF (ICHAR = EQ, PC = CT) GO TO 12
CALL LABPR(LABEL, LNTF, IPTR, VAL, NSYT, SYMB, ISYM, IERROR, PC)
IF (VAL .GT. 255.) IERRCP = 3
MODE = 1
RETURN
12 VAL = SLE(LABEL(2))
IF (LNTF .EQ. 2) VAL = 22
IF (LNTF .GT. 4) IERROR = 2
MODE = 1
RETURN
C ABSOLUTE AND ZERO PAGE ADDRESSING
C 50 MODE = 2
IF (VAL .LT. 256) MODE = 3
RETURN
C INDIRECT ADDRESSING
C 70 BYTE = 2
IF (PASS .EQ. 1) RETURN
IPTR = 1
CALL LABPR(LABEL, LNTF, IPTR, VAL, NSYT, SYMB, ISYM, IERROR, PC)
IF (VAL .GT. 255) IERROR = 4
IF (IPTR .EQ. LNTF) GO TO 21
ICHAR = ISLT(IPTR+1, LABEL)
IF (ICHAR = EQ, PC = CT) GO TO 22
IF (ISLT(IPTR+2, LABEL)) .NE. X .OR. (IPTR+3).NE.LNTF) IERROR = 2
C Indexed Indirect
C IF (VAL .LT. 0 .OR. VAL .GT. 255.) IERRCP = 5
MODE = 6
RETURN
C INDIRECT (JUMP ONLY)
C 21 MODE = 10
RETURN
22 IF (ISLT((IPTR+3), LABEL)) .NE. Y .OR. (IPTR+3).NE.LNTF) IERROR = 2
C INDIRECT INDEXED
C IF (VAL .LT. 0 .OR. VAL .GT. 255.) IERRCP = 5
MODE = 6
RETURN
C ACCUMULATOR ADDRESSING
C 30 BYTE = 1
MODE = 4
RETURN
END
SUBROUTINE LABELP(LABEL, LNGTH, IPTR, VAL, N, SYM, SYM, IERROR, PC)

IMPLICIT INTEGER*2 (A-Z)

C
C THIS ROUTINE DETERMINES THE VALUE OF A LABEL IN THE OPERAND ALONG
C WITH AN ADDED OR SUBTRACTED CONSTANT
C
C LABEL
C LNGTH
C IPTR
C VAL
C NSYM
C SYMB, ISYM
C IERROR
C PC

REAL ANSER, VAL, PC
DATA COMMA, '/', CPAKEN, '/', ZEGO, 'O', 'NINE', 'O', 'BLANK', ' ',
* 'A', 'A', 'Z', 'Z', 'AT', 'O', 'COTLAS', 'O', 'PLUS', 'O', 'MINUS', 'O', 'STAF', 'O'/
DATA FCENT, 'O'/
DIMENSION LABEL(1), SYMB(1), ISYM(3, 1), ICP(3)
REAL EASE
INTEGER KPTR
NONE = 1

C LOOK FOR SYMBOL IN TABLE

VAL = 0
IADD = 1
ICHAP = ISQLT(IPTR + 1, LABEL)
IF (ICHAP .EQ. .STAB) GO TO 44
GO TO 10 I = 1, 26
IF (ICHAP .EQ. ABET(1)) Go TO 14
10 CONTINUE
GO TO 13
14 ICP(1) = BLANK
ICP(2) = BLANK
ICP(3) = BLANK
LTH = 6
CALL CWORD(LABEL, IPTR, LNGTH, ICP, LTH, IFLAG, ONE)
IF (IFLAG .EQ. -1) GO TO 40
IF (NSYM .EQ. 0) GO TO 145
GO TO 48 I = 1, NSYM
IERR = IAB (ICP(1) - ISYM(1, 1)) + IAB (ICP(2) - ISYM(2, 1)) +
* IAB (ICP(3) - ISYM(3, 1))
IF (IERR .EQ. 0) GO TO 46
46 CONTINUE

C LABEL NOT FOUND - ASSUME NOT BASE PAGE

48 IERR = 1
RETURN
44 VAL=FC
   IPTR=IPTR+1
   GO TO 43
45 VAL=SYMB(I)
46 IF(IPTR.EQ.,LNTH)GO TO 47
47 IF(CHAR.EQ.,PLUS)IADD=1
48 IF(CHAR.EQ.,MINUS)IADD=-1
49 KPTR=IPTR
   ICHAR=ISOLT(KPTR, LABEL)
   IF(CHAR.EQ.,COMMA.OR. CHAR.EQ.,CPAREN)GO TO 49
   IF(CHAR.EQ.,PLUS)IADD=1
   IF(CHAR.EQ.,MINUS)IADD=-1
13 IPTR=IPTR+1
   KPTR=IPTR
   ICHAR=ISOLT(KPTR, LABEL)
   BASE=C
   IF(CHAR.EQ.,AT)BASE=8.
   IF(CHAR.EQ.,DOLLAR)BASE=16.
   IF(CHAR.EQ.,CENT)BASE=2.
   IF(CHAR.GE.,ZERO.AND. CHAR.LE.,NINE)BASE=10.
   IF(BASE.EQ.,0)GO TO 40
   IF(BASE.EQ.,10.)IPTR=IPTR-1
   CALL CONV1(LABEL, LNH, IPTR, BASE, ANSER)
   IF(IADD.EQ.,1)VAL=VAL+ANSER
   IF(IADD.EQ.,-1)VAL=VAL-ANSER
47 RETURN
45 IPTR=IPTR-1
   RETURN
40 VAL=0
   IERROR=2
   RETURN
END
SUBROUTINE GWORD(IPUF,IBUF,P,IBUFL,IPBF,IPBL,IFLAG,IFST)
  IMPLICIT INTEGER(A-Z)
C THIS SUBROUTINE TRANSFERS WORDS FROM ONE BUFFER TO ANOTHER. A
C WORD IS DEFINED BY A TRAILING SPACE. INITIAL BLANKS ARE THROWN
C OUT. IF AN CCC NUMBER OF BYTES ARE ENCOUNTERED, THE FINAL BYTE OF
C THE OUTPUT BUFFER IS FILLED WITH A BLANK.
C
C IPUF  INPUT BUFFER IBUF BYTES LONG
C IBUFPP POINTS TO LAST BYTE PROCESSED IN IPUF
C IBUF OUTPUT BUFFER
C IPBF INPUT AS LENGTH (BYTES) OF IPBF, ON RETURN # BYTES MOVED
C IFLAG ON RETURN: 0 NORMAL RETURN
C                -1 END OF IPBF ENCOUNTERED
C                1 END OF IBUF ENCOUNTERED
C IFST 1 - RECORD MAY END WITH +,-, OR
C        0 RECORD ENDS WITH BLANK
C
DIMENSION IBUF(1),IPBF(1)
DATA SPACE'/','CPAREN'/')',PLUS'/','MINUS'/','COMMA'/','/
IFLAG=0
IPBL=0
20 IPUF=IPUF+1
  IF(IPUF.GT.IPBUF)GO TO 100
C
  IK=(IPUF+1)/2
  I=IPUF+1-2*IK
  IF(I.EQ.0)CHAR=CHR(UBUF(IK))
  IF(I.EQ.1)CHAR=CHR(UBUF(IK))
C TEST FOR LEADING SPACES
  IF(CHAR.EQ.SPACES/256.AND.IPBL.EQ.0)GO TO 20
C TEST FOR TRAILING SPACE
  IF(CHAR.EQ.SPACES/256)GO TO 200
  IF(IFST.EQ.0)GO TO 21
  IF(CHAR.EQ.PLUS/256.OR.CHAR.EQ.MINUS/256.OR.CHAR.EQ.COMMA/256)
  *GO TO 200
  IF(CHAR.EQ.CPAREN/256)GO TO 200
C PACK NON-SPACE BYTES
21 IPBL=IPBL+1
  IF(IPBL.GT.IPBL)GO TO 300
  I=IPBL+1/2
  I=IPBL+1-2*I
  IF(I.EQ.0)IPBF(IJ)=CHR(UBUF(IJ))
  IF(I.EQ.1)IPBF(IJ)=CHR(UBUF(IJ))
  IF(IFST.EQ.0).AND.IFST.EQ.1)
  *GO TO 20
C END OF INPUT BUFFER RETURN
100 IFLAG=1
  IPUF=IPUF-1
  200 IPBL=IPBL
     RETURN
C OUTPUT BUFFER OVERFLOW
300 IPBL=IPBL-1
  IFLAG=-1
     RETURN
END
SUBROUTINE OPCD(IN, IUPF, BYTE, CODE, PASS, NSYT, ISYM, SYMB, PC,
IJERROR, OBUF, JK)
IMPLICIT INTEGER *2 (A-Z)

C THIS SUBROUTINE DECODES THE OPCODES AND CALLS PROC TO DECODE
C LABELS. ON PASS 1 IT RETURNS WITH THE NUMBER OF BYTES REQUIRED
C FOR THE INSTRUCTION. ON PASS 2 IT ALSO RETURNS THE CODE IN
C ARRAY CODE.
C
IN INPUT BUFFER
IUPF INPUT BUFFER POINTER
BYTE NUMBER OF BYTES FOR INSTRUCTION
CODE REAL BUFFER CONTAINING THE CODE
PASS ASME PASS
NSYT NUMBER OF ENTERIES IN SYMBOL TABLE
ISYM SYMBOL TABLE
SYMB SYMBOL TABLE ADDRESSES
PC PROGRAM COUNTER
IERROR ERROR FLAG
OBUF CONSTANT DEFINITION BUFFER
JK NUMBER OF CONSTANTS IN OBUF OR -1 FOR ORG OR BSS

DIMENSION NGPC(270), IN(1), ISYM(31), IUPF(1)
DIMENSION OPCF(3), LABEL(40), MIST(33), MIST2(11, 22)
REAL PC, PCODE

DIMENSION CODE(11), SYMB(1)
DATA PLUS/IH+, MINUS/IH-
DATA NGPC/2HB, 2H, 2HC, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H
& 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H, 2HC, 2H

C FIND OPCODE
Z=O
ALT=Y=60
JK=0

ORIGINAL PAGE IS
OF POOR QUALITY
-26-
LDPDC=4
CALL GWORD(IN,IBUF,AITY,OPCF,LOPC,IFLAG,ZRO)
IF(IFLAG.NE.0)ERROR=8
GO TO 170
ERROR=1AB(OPCF(1)-NOCPC(1,1))+IAB(OPCF(2)-NOCPC(2,1))
IF(ERROR.EQ.0)GO TO 10
CONTINUE
ERROR=8
CODE(1)=-1
BYTE=C
RETURN
IF(I.GT.33)GO TO 50
C
C OPCODES WITH IMPLIED OR RELATIVE ADDRESSING
C
IF(I.LT.25)BYTE=1
IF(I.GT.25)BYTE=2
IF(PASS.EQ.1)RETURN
CODE(1)=MIST(I)
IF(I.LT.26)RETURN
LABEL=80
CALL GWORD(IN,IBUF,AITY,LABEL,LABL,IFLAG,ZRO)
IF(IFLAG.NE.0)GO TO 110
ICHAR=ISCLT(1,LABEL)
IPTR=1
IADD=C
IF(ICHAR.EQ.PLUS)IADD=1
IF(ICHAR.EQ.MINUS)IADD=-1
IF(IADD.EQ.0)IPTR=0
CALL LABPR(LABEL,LABL,IPTR,VAL,NSYT,SYM,ISYM,IEPROR,PC)
IF(IADD.EQ.1)CODE(2)=VAL
IF(IADD.EQ.-1)CODE(2)=-VAL
IF(IADD.EQ.0)CODE(2)=VAL-(PC+2)
IF(CODE(2).LT.-128.OR.CODE(2).GT.128)ERROR=6
RETURN
C
LABEL=82
IF(I.EQ.61)GO TO 60
IF(I.GT.64.AND.I.LT.71)GO TO 70
C
C OPCODES WITH MULTIPLE ADDRESSING MODES
C
CALL GWORD(IN,IBUF,AITY,LABEL,LABL,IFLAG,ZRO)
IF(IFLAG.NE.0)GO TO 110
CALL FPROC(LABEL,LABL,NSYT,SYM,ISYM,BYTE,PASS,VAL,
& IERROR,MODE,PC)
IF(I.LT.59)GO TO 52
C
C ERC, EQU, BSS, AND ADDR PSUEDO INSTRUCTIONS
C
BYTE=C
IF(I.EQ.60)PC=VAL
IF(I.EQ.60)JK=-1
IF(I.EQ.62.AND.PASS.EQ.1)SYM(NSYT)=VAL
CODE(1)=0
CODE(2)=C
IF(I.EQ.53)BYTE=VAL
IF(I.EQ.54)RETURN
BYTE=2
CODE(1)=VAL
CODE(2)=VAL/256
IF(VAL.LT.256.AND.PC.LT.256)BYTE=1
RETURN

ORIGINAL PAGE IS OF POOR QUALITY
MULTIPLE MODE INSTRUCTION STORE

52  IF (I.EQ.44) BYTE=3
    IF (I.EQ.45) BYTE=3
    IF (PASS.EQ.1) RETURN
    CODE(1)=NIST2(MODE,1-33)
    IF (CODE(1).EQ.-1) GO TO 208
    IF (CODE(1).GT.0) GO TO 207
    CODE(1)=-CODE(1)
    BYTE=3
    GO TO 207
208  ERROR=7
207  CODE(2)=VAL
    RETURN
60  PASS=PASS+1
    RETURN

PROCESS REMAINING PSUEDO INSTRUCTIONS

70  CALL NUMBR(IN, IBUF, OBUF, JK, I, BYTE, ERROR)
    RETURN
110  ERROR=2
    RETURN
    END
SUBROUTINE SYMB(LABEL, PC, ISYMB, NSYM, N, NSYM, SYMB, LINE, NERR)  
IMPLICIT INTEGER*2 (A-Z)  
C  
C THIS PROGRAM CHECKS LABELS AND ENTERS VALID LABELS INTO SYMBOL  
C TABLE IN PASS 1.  
C  
C LABEL  
LABEL TO BE ENTERED (LABEL(3))  
C PC  
PROGRAM COUNTER  
C ISYMB  
SYMBOL TABLE (ISYMB(3, NSYM))  
C NSYM  
MAX LENGTH OF SYMBOL TABLE  
C N  
L. U. FOR ERROR OUTPUT  
C NSYM  
LOC OF LAST TABLE ENTRY  
C SYMB  
SYMBOL TABLE ADDRESSES  
C LINE  
SOURCE LINE COUNTER  
C NERR  
# OF ERRORS IN PASS  
C  
DATA SPACE/*'*/  
INTEGER*2 A(25) /A! , B! , C! , D! , E! , F! , G! , H! , I! , J! , K! ,  
REAL FC  
INTEGER N  
DATA A! , A! , X! , Y! , Y/ , S! / , R/ , P! /  
DIMENSION LABEL(3), ISYMB(3, 1), SYMB(1)  
LL=LABEL(I)  
IF(LL.EQ.A.OR.LL.EQ.B.OR.LL.EQ.C.OR.LL.EQ.D.OR.LL.EQ.E.OR.LL.EQ.F.OR.LL.EQ.G.OR.LL.EQ.H.OR.LL.EQ.I.OR.LL.EQ.J.OR.LL.EQ.K.OR.LL.EQ.L.OR.LL.EQ.M.OR.LL.EQ.N.OR.LL.EQ.O.OR.LL.EQ.P.OR.LL.EQ.Q.OR.LL.EQ.R.OR.LL.EQ.S.OR.LL.EQ.T.OR.LL.EQ.U.OR.LL.EQ.V.OR.LL.EQ.W.OR.LL.EQ.X.OR.LL.EQ.Y.OR.LL.EQ.Z) GO TO 40  
ICHAR=VASK2(SR(3)(LABEL(I))))  
15 CONTINUE  
WRITE(N, 1) LINE, LABEL  
1 FORMAT(IS, 4H *** 3A2, ' INVALID, MUST START WITH LETTER****)  
NERR=NERR+1  
RETURN  
C  
SEARCH SYMBOL TABLE FOR DUPLICATE ENTRIES  
C  
100 IF(NSYM.EQ.0) GO TO 200  
10 CONTINUE  
DO 10 I=1, NSYM  
* IAB = IA2 (LABEL(I) = ISYMB(1, I)) + IAB (LABEL(2) = ISYMB(2, I))  
* IA2 (LABEL(3) = ISYMB(3, I))  
IF(IAB.EQ.0) GO TO 20  
10 CONTINUE  
200 NSYM=NSYM+1  
IF(NSYM.EQ.0) GO TO 30  
C ENTER LABEL  
ISYMB(1, NSYM) = LABEL(I)  
ISYMB(2, NSYM) = LABEL(2)  
ISYMB(3, NSYM) = LABEL(3)  
SYMB(NSYM) = PC  
RETURN  
20 WRITE(N, 2) LINE, LABEL  
2 FORMAT(IS, 4H *** 3A2, ' IS DUPLICATE LABEL ****)  
NERR=NERR+1  
RETURN  
30 WRITE(N, 3)  
3 FORMAT(IS, 4H *** SYMBOL TABLE OVERFLOW ****)  
STOP  
40 WRITE(N, 5) LINE, LABEL(I)  
5 FORMAT(IS, 4H *** 1A2, ' IS A RESERVED SYMBOL ****)  
NERR=NERR+1  
RETURN
SUBROUTINE NUMBER(IN,IPTR,OBUF,JK,1,BYTE,ERROR) 
IMPLICIT INTEGER*2 (A-Z) 
C 
C THIS SUBROUTINE DECODES OPERAND OF PSEUDO DFS ASC, OCT, HEX, OCM. 
C INT, AND BCD. 
C IN INPUT BUFFER 
C IPTR POINTER LOCATION FOR IN 
C OBUF OUTPUT BUFFER FOR CONSTANTS 
C JK NUMBER OF CONSTANTS IN OBUF 
C I CODE NUMBER 
C BYTE NUMBER OF BYTES IN CONSTANT 
C ERROR ERROR FLAG 
REAL ANSER 
DATA PLUS/**/**,MINUS/**/**,QUOTE/**/**,COMMA/**/**, 
DIMENSION OBUF(1),IN(1),NUM(3) 
INTEGER ASC(34),/32,65,66,67,68,69,70,71, 
# 72,73,75,76,77,78,79,80, 
# 81,82,83,94,42,43,44,45,46,47,48, 
# 49,50,51,52,53,54,55, 
# 56,57,58,59,60,61,62,63, 
REAL BASE 
INTEGER KPTR, 
ITY=20 
ZRC=0 
ONE=1 
BYTE=1 
IF (1.EQ.66.OR.I.EQ.69)BYTE=2 
JK=0 
JPTR=IPTR 
LUTH=80 
CALL CWORD(IN,IPTR,ITY,OBUF,LNTH,IFLAG,ZRC) 
JPTR=JPTR-LNTH-1 
III=1-64 
GO TO (65,20,30,40,40,40),III 
C 
C SET BASE 
C 
20 BASE=6 
GO TO 10 
30 BASE=16 
GO TO 10 
40 BASE=10 
GO TO 10 
10 JK=0 
C LOOP TO PROCESS OPERAND 
11 IBBL=6 
CALL CWORD(IN,IPTR,ITY,NUM,IBFL,IFLAG,CME) 
KPTR=IPTR 
CME=ISCLT(KPTR,IN) 
IF (CME.EQ.PLUS.OR.CME.EQ.MINUS)GO TO 100 
50 IUBFR=0 
2 FORM(1,64) 
CALL CONV1(NUM,IPFL,IUBFR,BASE,ANSER) 
JK=JK+1 
CME=ANSER/BYTE**8 
IF(CME.GT.256).IFERROR=6 
IF(CME.GT.70).GO TO 110 
IF(I.EQ.0).ANSER=-ANSER 
IF(ANSER.GT.32767.).ANSER=ANSER-65536. 
-30-
DBUF(JK) = ANSER
1 FORMAT(2I3)
   IF(IPT. NE. JPTR) GO TO 10
   BYTE=1
   IF(I. EQ. 66. OR. I. EQ. 68) BYTE=2
   RETURN
100 IF(ICHAR. EQ. MINUS) IM=1
   CC TO 11
C ECO PROCESSOR
110 IF(IM. EQ. 1) ANSER=100.-ANSER
   JNO=ANSER/10.
   DBUF(JK)=JND*16+ANSER-JND*1C
   IF(IPTR. NE. JPTR) GO TO 10
   BYTE=1
   RETURN
C ASCII PROCESSOR
65 IPT=IPT+1
   BYTE=1
66 IPT=IPT+1
   KPTR=IPTR
   ICHAR=ISOLT(KPTR,1N)
   IF(ICHAR. EQ. 040) RETURN
   JK=JK+1
   IF(JK. GT. 40) GO TO 67
   JCHAR=SUB(ICHAR)
   KCHAR=JCHAR/64*64
   JCHAR=JCHAR-KCHAR
   DBUF(JK)=ASCII(JCHAR+1)
   GO TO 66
67 JK=JK-1
   IFERROR=10
   END
SUBROUTINE CONV1(IBUF,IBUFL,IPTR,BASE,ANSER)
IMPLICIT INTEGER*2 (A-Z)
C
C THIS SUBROUTINE CONVERTS EITHER AN ASCII BUFFER IBUFL BYTES
C LONG TO A FLOATING POINT NUMBER, OR CONVERTS A FLOATING POINT
C NUMBER TO HEX.
C
C IBUF  EASE>0 INPUT ASCII BUFFER (ANY LENGTH)
C IBUFL EASE<0 OUTPUT ASCII BUFFER HEX (1 OR 2 WORDS LONG)
C IPTR EASE>0 FOR BASE >0 NUMBER OF BYTES TO BE PROCESSED
C BASE EASE<0 FOR BASE <0 NUMBER OF WORDS TO BE OUTPUT (1 OR 2 ONLY)
C EASE>0 POINTS TO LAST BYTE PROCESSED
C EASE<0 NO FUNCTION
C
C DIMENSION NUM(16),IWB(16),IBUF(1)
REAL BASE
REAL ANSER
INTEGER KPTR
INTEGER IJ
REAL RJ
DATA NUM/'0','1','2','3','4','5','6','7','8','9','A','B','C',
    'D','E','F'/

     IF(BASE.LT.0)GO TO 50
     IPTR=IPTR+1
     JK=0
10  KPTR=IPTR
     ICHAR=ISCLT(KPTR,IBUF)
     IF(CHAR.EQ.'I')GO TO 20
     GO TO 10
20  CONTINUE
C NON-RECOGNIZED SYM ECL
     IPTR=IPTR-1
     GO TO 30
21  JK=JK+1
     IWB(JK)=I-1
     IF(IPTR.GT.0)GO TO 30
     IPTR=IPTR+1
     GO TO 10
C CONVERT RESULT
30  ANSER=0.
     IF(JK.EQ.0)RETURN
     CC 31 L=1
30  ANSER=ANSER+BASE**((JK-L)*IWB(L))
     RETURN
50  IF(ANSER.LT.0)ANSER=65535.*ANSER
     I=1
51  I=AVCD(I,J,256.)
     J=MOD(I,J,16)
     IL=IJ/16
     IBUF(1)=SR(NU(M(IK+1)))*SFR(NU(M(IK+1)))*256
     IF(IBUF.EQ.1)RETURN
     I=I+1
     IJ=ANSER/256
     GO TO 51
END
SUBROUTINE PUNCH(POUT,MN,LU)
IMPLICIT INTEGER*2 (A-Z)

THIS SUBROUTINE CALCULATES THE CHECKSUM AND PUNCHES THE RECORD

POUT BUFFER CONTAINING INFORMATION TO BE PUNCHED
MN NUMBER OF ENTRIES IN POUT (INCLUDING P, C)
LU LOGICAL UNIT OF PUNCH

REAL AMN
REAL CHKSM,ANS
INTEGER LUP
DIMENSION POUT(1),ICHK(2)
REAL MCNE
MCNE=-1
TWC=2
ONE=1
CHKSM=MN-2
DO 10 I=1,MN
  IPTR=0
  IP=POLT(I)
  CALL CONVI(IP,TWC,IPTR,16.,ANS)
  CHKSM=AMCD(CHKSM+ANS,65536.)
10 CONTINUE

AMN=MH=2

CALL CONVI(MHV,ONE,IPTR,MCNE,AMN)
CALL CONVI(ICHK,TWO,IPTR,MCNE,CHKSM)
WRITE(LUP,1)NUM,POUT(I),I=1,MN),ICHK(2),ICHK(1)
1 FORMAT(1H1,40A2)
MN=0
RETURN
END

INTEGER FUNCTION IAB*2(IN)

HALFWORD ABSOLUTE VALUE ROUTINE

INTEGER*2 IN
IF(IN.GT.0)IAB=IN
IF(IN.LE.0)IAB=-IN
RETURN
END

INTEGER FUNCTION MASK2*2(IN)

2-BYTE INPUT NUMBER IS SHIFTED LEFT 6 BITS AND A BLANK
IS ADDED IN THE SECOND BYTE.

INTEGER*2 J(2),IN
EQUIVALENCE(J(1),K)
K=0
J(2)=IN
K=K*256+64
MASK2=J(2)
RETURN
END
INTEGER FUNCTION SFR*2(IN)
IMPLICIT INTEGER*2(A-Z)
C           HALFWORD INPUT IS SHIFTED RIGHT 8 BITS, ZEROS APPEAR IN
C           LEFT 8 BITS.
INTERGER K
INTERGER*2 J(2)
EQUIVALENCE(J(1),K)
K=0
J(2)=IN
K=K/256
SRB=J(2)
RETURN
END

INTEGER FUNCTION MASK*2(IN)
C           HALFWORD INPUT: ROUTINE ZEROS LEFT 8 BITS
INTERGER*2 J(2),IN
EQUIVALENCE(J(1),K)
K=0
J(2)=IN
K=K/256
J(1)=0
K=K/256
MASK=J(2)
RETURN
END

SUBROUTINE LPMT(IN)
C           FORMATS INPUT LINE WITH LABEL IN COL. 1-6, OP CODE IN 8-10,
C           AND OPERAND AND COMMENT SEPARATED BY 2 SPACES. HELPS USER
C           BY ALLOWING FREE-FORM INPUT.
LOGICAL*1 IN(80),K(2),RES(80)
INTERGER*2 BLK,'/'
INTERGER*2 L
INTERGER BUF(20),FK,'/'
EQUIVALENCE (BUF(1),RES(1)),(K(1),L)
DO 10 I=1,20
10 BUF(1)=2K
L=BLK
K=1
IF(L.EQ.BLK) GO TO 12
IF(RES(1).NE.IN(1)) GO TO 12
DO 11 I=2,72

ORIGINAL PAGE IS
OF POOR QUALITY
-34-
N=I
K(2)=IN(I)
IF(L.EQ.BLK)GO TO 12
11 RES(I)=IN(I)
GO TO 99
12 J=N
N=M+1
TO 13 I=N,72
N=I
K(2)=IN(I)
IF(L.NE.BLK)GO TO 14
13 CONTINUE
GO TO 99
14 M=N
DO 15 I=N,72
NN=I
K(2)=IN(I)
IF(L.EQ.BLK)GO TO 16
RES(M)=IN(I)
N=M+1
IF(M.GT.72)GO TO 99
15 CONTINUE
GO TO 99
16 NN=NN+1
DO 17 I=NN,72
N=I
K(2)=IN(I)
IF(L.NE.BLK)GO TO 18
17 CONTINUE
GO TO 99
18 M=N
DO 19 I=N,72
NN=I
K(2)=IN(I)
IF(L.EQ.BLK)GO TO 20
RES(M)=IN(I)
N=M+1
IF(M.GT.72)GO TO 99
19 CONTINUE
GO TO 99
20 N=M+2
DO 21 I=NN,72
N=I
K(2)=IN(I)
IF(L.NE.BLK)GO TO 22
21 CONTINUE
GO TO 99
22 TO 23 I=N,72
IF(M.GT.72)GO TO 99
RES(M)=IN(I)
23 N=M+1
199 CC 100 I=1,72
100 IN(I)=RES(I)
RETURN
END

ORIGINAL PAGE IS OF POOR QUALITY
APPENDIX B
FILE: PROFILE EXEC A

&CCNTFCL CFF
CP SPCCCL RDR CLASS *
&IF &READFLAG EG 'CONSOLE &GCTC -NCSTK
&READ ARGS
-NCSTK CP SPCCCL CONS TO * NCTERM START
SET RDMSG SMSG
CP TERMINAL LINES 132
CP DEFINE T3350 AS 193 CYL 1
&STACK YES
&STACK RNL
FORMAT 193 C
ACCESS 193 C
CP PURGE RDR CL T
CP SPCCCL CONSOLE TERM STCF FURGE
&ETYPE READY: 1-CYL C-DISK CKLINE
C1 C2 C3 C4 C5 C6 C7 C8
&EXIT

FILE: JASM EXEC A

&CCNTFCL CFF NCMSG
GLOBAL TXTLIB FKRTLIB
ERASE JOLT PRINTOUT C
FILEDEF 6 TERMINAL
FILEDEF 4 DISK JOLT HEXCCDE C
FILEDEF 11 DISK JOLT CLEAN C1 (RECFM F BLKSIZE 80 LRECL 80)
&IF &INDEX LT 3 &GCTC -NOARC
START 61 62 63
&IF &FETCCDE NE 0 &GCTC -FILE
FILEDEF 5 DISK 61 62 63 (RECFM F LRECL 80)
&IF &INDEX EG 3 &GOTO -LOAD
&IF &A EG NCLIST &GCTC -NCLIST
-LOAD LOADVCD JOLT
START
&EXIT
-NOARC &IF &INDEX EG 0 &EXIT
&IF 61 NE TERMINAL &GCTC -MSG
FILEDEF 5 TERMINAL
&GOTO -LOAD
-MSG &ETYPE COMPLETE FILE NAME NOT SUPPLIED
&EXIT
-FILE &ETYPE FILE 61 62 63 NOT FOUND
&EXIT
-NCLIST FILEDEF 6 DISK JOLT PRINTOUT C
&GOTO -LOAD
FILE: JEPPE EXEC A

&CONTROL OFF NOMSG
STATE JCLT PRINTOUT C
&IF CRETCODE NE 0 &GOTO -FILE
&EGSTACK
VERIFY ON & 25
LOCATE / END PASS/
LOCATE / END PASS/
VERIFY ON 1 96
ZONE 1 5
TOP
CHANGE /-/-/ **
QUIT
END
E JCLT PRINTOUT C
&EXIT
-FILE &TYPE JCLT PRINTOUT C NCT FOUND
&EXIT

FILE: MASM EXEC A

&CONTROL OFF NOMSG
GLOBAL TATLIE FORTLIE
ERASE MAINT PRINTOUT C
FILEDEF 6 TERMINAL
FILEDEF 4 DISK MAINT HEXCODE C
FILEDEF 11 DISK MAINT CLEAN C
&IF &INDEX LE 3 &GOTO -NCARG
STATE &1 &2 &3
&IF CRETCODE NE 0 &GOTO -FILE
FILEDEF 5 DISK &1 &2 &3
&IF &INDEX EQ 0 &GOTO -LOAD
&IF &4 EG NOLIST &GOTO -NOLIST
-LOAD LOADED MAINT
START
&EXIT
-NCARG &IF &INDEX EQ 0 &EXIT
&IF &1 NE TERMINAL &GOTO -MSG
FILEDEF 5 TERMINAL
&GOTO -LOAD
-MSG &TYPE COMPLETE FILE NAME NOT SUPPLIED
&EXIT
-FILE &TYPE FILE &1 &2 &3 NCT FOUND
&EXIT
-NOLIST FILEDEF 6 DISK MAINT PRINTOUT C
&GOTO -LOAD

-38-
FILE: MBUILD EXEC

`CONTCL OFF NOMSG
GLOBAL TXTLIB FORTLIB
STACK LIFE NOSOURCE NOMAP
EXEC FWLFORT ASM6502
LOAD ASM6502 (NOMAP)
CENMOD MAINT MODULE C
ERASE MAINT PRINTOUT C
FILEDEF 6 TERMINAL
FILEDEF 4 DISK MAINT HEXCODE C
FILEDEF 11 DISK MAINT CLEAN C
IF INDEX LT 3 &GOTO -NOARG
STATE &1 &2 &3
IF &FETCODE NE 0 &GTC -FILE
FILEDEF 5 DISK &1 &2 &3
IF INDEX EQ 3 &GOTO -LOAD
IF &4 EQ NCINDEX &GTC -NCINDEX
-LOAD LOADMCC MAINT
START
EXIT
-NCINDEX &1 IF INDEX EQ 0 &EXIT
IF &1 NE TERMINAL &GTC -MSG
FILEDEF 5 TERMINAL
&GTC -LOAD
-MSG &TYPE COMPLETE FILE NAME NOT SUPPLIED
EXIT
-FILE &TYPE FILE &1 &2 &3 NOT FOUND
EXIT
-NCINDEX FILEDEF 6 DISK MAINT PRINTOUT C
&GTC -LOAD

-39-