A USERS MANUAL FOR A PDP-11 CROSS ASSEMBLER (CRASS) FOR THE TI-9900 MICROPROCESSOR

Irving S. Zaritsky

David W. Taylor Naval Ship Research and Development Center
Bethesda, Maryland 20884

Naval Supply Systems Command
Research and Technology Division
Washington, D.C. 20376

March 1981

90

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

Microcomputer Assembly Language
Microprocessor Assembler
Cross Assembler

This report describes CRASS, a two-pass cross-assembler for the TI-9900 microprocessor. The assembler was written in GIRL-FORTRAN to be used on a PDP-11 computer with an RT-11 operating system. It uses the standard TI-9900 mnemonics and allows for most of the same assembly time directives.
CRASS has the following features:

- Symbolic memory addressing
- Portability - Since CRASS is written mostly in FORTRAN, it can be easily adapted to any machine having a FORTRAN IV compiler.
- Left to right arithmetic expression handling in binary, decimal, or hexadecimal
- Logical expression handling
- Specified program starting address; however, there is a default value.
- Less program assembly time than with the TI assembler since the code which is created is non-relocatable and therefore does not require a linkage step.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>1</td>
</tr>
<tr>
<td>ADMINISTRATIVE INFORMATION</td>
<td>1</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>USING CRASS</td>
<td>2</td>
</tr>
<tr>
<td>GENERAL DISCUSSION</td>
<td>2</td>
</tr>
<tr>
<td>CONSTANTS AND EXPRESSIONS</td>
<td>3</td>
</tr>
<tr>
<td>Hollerith Constants</td>
<td>3</td>
</tr>
<tr>
<td>Numerical Constants</td>
<td>3</td>
</tr>
<tr>
<td>Identifiers</td>
<td>3</td>
</tr>
<tr>
<td>Logical and Numerical Operators</td>
<td>4</td>
</tr>
<tr>
<td>Logical and Numerical Expressions</td>
<td>4</td>
</tr>
<tr>
<td>INPUT FORMAT</td>
<td>4</td>
</tr>
<tr>
<td>OPERAND FORMATS</td>
<td>5</td>
</tr>
<tr>
<td>LIMITATIONS AND DIFFERENCES WITH THE TI ASSEMBLER</td>
<td>7</td>
</tr>
<tr>
<td>ERROR CODES</td>
<td>9</td>
</tr>
<tr>
<td>PROGRAM DESCRIPTION</td>
<td>10</td>
</tr>
<tr>
<td>DATA STRUCTURES USED BY CRASS</td>
<td>10</td>
</tr>
<tr>
<td>Instruction-Directive Tree</td>
<td>10</td>
</tr>
<tr>
<td>Identifier Tree</td>
<td>11</td>
</tr>
<tr>
<td>Label String</td>
<td>12</td>
</tr>
<tr>
<td>Token String</td>
<td>12</td>
</tr>
<tr>
<td>Item String</td>
<td>13</td>
</tr>
<tr>
<td>FORMING THE OBJECT CODE WORD</td>
<td>14</td>
</tr>
<tr>
<td>Instruction Code Skeletons</td>
<td>14</td>
</tr>
<tr>
<td>Object Code Formats</td>
<td>15</td>
</tr>
<tr>
<td>PROGRAM FLOW</td>
<td>16</td>
</tr>
<tr>
<td>Overview</td>
<td>16</td>
</tr>
<tr>
<td>Description and Example</td>
<td>16</td>
</tr>
<tr>
<td>BRIEF SUBROUTINE DESCRIPTIONS</td>
<td>19</td>
</tr>
<tr>
<td>Subroutine ADDREG</td>
<td>19</td>
</tr>
<tr>
<td>Subroutine LEXSCN</td>
<td>19</td>
</tr>
<tr>
<td>Subroutine ADDNAM (NODE, IDENT)</td>
<td>20</td>
</tr>
<tr>
<td>Subroutine INSTRU</td>
<td>20</td>
</tr>
<tr>
<td>Subroutine TOKSCN</td>
<td>20</td>
</tr>
<tr>
<td>Subroutine ITMSCN</td>
<td>21</td>
</tr>
<tr>
<td>Subroutine LINOUT</td>
<td>21</td>
</tr>
</tbody>
</table>

iii
ABSTRACT

This report describes CRASS, a two-pass cross-assembler for the TI-9900 microprocessor. The assembler was written in GIRL-FORTRAN to be used on a PDP-11 computer with an RT-11 operating system. It uses the standard TI9900 mnemonics and allows for most of the same assembly time directives.

CRASS has the following features:

- Symbolic memory addressing.
- Portability - Since CRASS is written mostly in FORTRAN, it can be easily adapted to any machine having a FORTRAN IV compiler.
- Left to right arithmetic expression handling in binary, decimal, or hexadecimal.
- Logical expression handling.
- Specified program starting address; however, there is a default value.
- Less program assembly time than with the TI assembler since the code which is created is non-relocatable and therefore does not require a linkage step.

ADMINISTRATIVE INFORMATION

This work was performed in the Computer Science Division of the Computations, Mathematics, and Logistics Department under the sponsorship of NAVSUP 043C, Task Area TF60531091, Work Unit 1800-008.

INTRODUCTION

This report describes CRASS, a two-pass cross-assembler for the TI9900 microprocessor. The assembler was written in GIRL-FORTRAN to be used on a PDP-11 computer with an RT-11 operating system. It uses the standard TI9900 mnemonics and allows for most of the same assembly-time directives.

CRASS has the following features:

- Symbolic memory addressing.
- Portability - Since CRASS is written mostly in FORTRAN, it can be easily adapted to any machine having a FORTRAN IV compiler.
- Left to right arithmetic expression handling in binary, decimal, or hexadecimal.

*A complete listing of references is given on page 83.
Logical expression handling

Less program assembly time than with the TI assembler since the code which is created is non-relocatable and therefore does not require a linkage step.

Although program debugging would take less assembly time with relocatable code, the time difference is not considered significant for our purposes.

Several uses for the TI9900 microprocessor are envisioned. All these projects will require rapid software development. These projects include use of the TI microprocessor to:

- Control an "intelligent" Logistics Communication Terminal between the SNAP II* and NAVMACS** systems.
- Control the interface between a hardware associative memory and the PDP-11 computer.
- Provide the computing power for an experimental distributed microprocessor.
- Provide the interface for other similar man-machine and machine-machine projects (such as signature verification).

USING CRASS

GENERAL DISCUSSION

An assembly language program may be assembled on the PDP-11 by executing the CRASS.SAV file. If this file is on the system disk, the form is:

```
R CRASS
```

The program will respond by asking for the input and output file names:

```
"PLEASE ENTER FILE NAMES IN COMMAND STRING FORM"
```

The command string format for CRASS is:

```
"output file 1 [, output file 2] = input file"
```

The first output file contains the absolute object code and has a default extension name "ABS." The second output file is optional. It contains the source and identification listing if the "LIST" directive is included in the assembly program. This file has a default extension name "LST." Note that the RKI disk unit must be turned on, since CRASS places one of its temporary scratch files there.

---

*Shipboard Nontactical ADP Program.

**Naval Modular Automated Communications System.
CONSTANTS AND EXPRESSIONS

Hollerith Constants

Hollerith constants are delineated by either a pair of quotes (') or a pair of dollar signs ($), for example:

```
LABL1 BYTE 'A', 'X', $Y$;
LABL2 DATA $AB$, 'XY';
LABL3 TEXT 'ABCDE';
LABL4 TEXT $TODAY'S$ $DATE$ IS$;
LABL5 TEXT 'COST IS $5.00';
```

The following statements produce identical code:

```
BYTE 'A', 'B', 'C', 'D', 'E', 'F';
DATA 'AB', 'CD', 'EF';
TEST 'ABCDEF';
```

A discussion of the limitations on these directives may be found in the section on limitations and differences with the TI assembler and also in the TI9900 Manual, Section 7, page 10.

Numerical Constants

Numerical constants may be expressed in decimal, binary, or hexadecimal form. Default is decimal; binary numbers are preceded by a percent sign (%) and hexadecimal numbers are preceded by a "greater than" operator (>). The minus sign (-) precedes either "%" or ">" for negative numbers. The following limitations are in effect:

```
Integer +32767
Binary  +7 F F F F F F F F F
Hexadecimal -7FFF, -7F11
```

Identifiers

All identifiers must begin with an alphabetic character. The length of identifiers is limited (solely by the fixed-input format for labels) to six characters.

Logical and Numerical Operators

CRASS allows for the following logical operations:
### Operator Function

- **+**: Logical OR
- **-**: Logical AND
- **. -**: Logical NOT (one's complement)
- **/**: Modulo

Arithmetic functions are performed with the four standard operators:

+ , *, -, and /.

### Logical and Numerical Expressions

All expressions are evaluated strictly on a left to right basis with no parentheses allowed.

For example: `X EQU 3+4*5;`

\n\nX is set to 35, not 23.

Also, expressions must be resolved within two passes.

For example,

```
DATA X+3;
X MOV R1,R2;
```

will be correctly evaluated. However,

1) `A EQU B+C`
2) `B EQU C+D`
3) `C EQU -10;`
4) `D EQU -5;`

will not be correctly evaluated since "B" will not be resolved until the second pass examines statement number two.

Logical expressions follow the same structure and are evaluated in the same manner as numerical expressions.

For example:

```
X BYTE F.+1001.*10;
```

places "0A" into the appropriate byte and

```
Y BYTE F.*1001.+10;
```

places "0B" into the appropriate byte
INPUT FORMAT

Assembly statements consist of the following four fields:

```
1 8 13
LABEL INSTRUCTION OR DIRECTIVE OPERAND FIELD COMMENTS
```

The label may be placed anywhere in positions 1-6. Up to six alphanumeric characters are allowed, the first of which must be alphabetic. An asterisk (*) is placed anywhere in the label field, the entire statement is treated as a comment.

Instructions and assembler directives must begin in position 8. Since the collection of instructions and directives available with CRASS is smaller than that offered by the TI assembler, the reader is referred to Table 2 on page 6 and Table 3 on page 7 for instructions and directives available with CRASS. The reader is also referred to pages 6-18 through 6-60 and pages 7-8 through 7-14 from the TI Manual.

Operands may be placed anywhere from position 13 to position 72. Operand fields are separated by commas and the last field must be terminated by a semicolon (;). Comments may be placed after the semicolon. Instructions and directives which do not use the operand field do not require a semicolon. Arithmetic expressions must not contain any embedded blanks. Operand formats are discussed in the next section.

OPERAND FORMATS

The entire T19900 instruction set requires twelve formats for the operand field. The format descriptions are listed in Table 1. The formats for the instruction set are summarized in Table 2. The formats for the directives are summarized in Table 3.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Modifiable source address</td>
</tr>
<tr>
<td>D</td>
<td>Modifiable destination address</td>
</tr>
<tr>
<td>W</td>
<td>Unmodified workspace register</td>
</tr>
<tr>
<td>DIS</td>
<td>Displacement in bytes (displacement is in words for the TI assembler)</td>
</tr>
<tr>
<td>C</td>
<td>Count: integers 0-15</td>
</tr>
<tr>
<td>IOP</td>
<td>Immediate operand value</td>
</tr>
<tr>
<td>BIT</td>
<td>Integer 0-255</td>
</tr>
</tbody>
</table>
TABLE 1 (Continued)

<table>
<thead>
<tr>
<th>Operand Format No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( y, d )</td>
</tr>
<tr>
<td>2</td>
<td>DIS</td>
</tr>
<tr>
<td>3</td>
<td>S, W</td>
</tr>
<tr>
<td>4</td>
<td>S, C</td>
</tr>
<tr>
<td>5</td>
<td>W, C</td>
</tr>
<tr>
<td>6</td>
<td>S</td>
</tr>
<tr>
<td>7</td>
<td>NONE</td>
</tr>
<tr>
<td>8</td>
<td>W, IOP</td>
</tr>
<tr>
<td>9</td>
<td>S, W (uses ( W_i ) and ( W_{i+1} ))</td>
</tr>
<tr>
<td>A</td>
<td>BIT</td>
</tr>
<tr>
<td>B</td>
<td>IOP</td>
</tr>
<tr>
<td>C</td>
<td>W</td>
</tr>
</tbody>
</table>

TABLE 2 - INSTRUCTION OPERAND FORMAT NUMBERS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>JCT</td>
<td>2</td>
<td>RTWP</td>
<td>7</td>
</tr>
<tr>
<td>AB</td>
<td>1</td>
<td>JH</td>
<td>2</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>ABS</td>
<td>6</td>
<td>JHE</td>
<td>2</td>
<td>SB</td>
<td>1</td>
</tr>
<tr>
<td>AI</td>
<td>8</td>
<td>JL</td>
<td>2</td>
<td>SBO</td>
<td>A</td>
</tr>
<tr>
<td>ANDI</td>
<td>8</td>
<td>JLE</td>
<td>2</td>
<td>SBZ</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>JLT</td>
<td>2</td>
<td>SETO</td>
<td>6</td>
</tr>
<tr>
<td>BL</td>
<td>6</td>
<td>JMP</td>
<td>2</td>
<td>SLA</td>
<td>5</td>
</tr>
<tr>
<td>BLWP</td>
<td>6</td>
<td>JNC</td>
<td>2</td>
<td>SOC</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>JNE</td>
<td>2</td>
<td>SOCB</td>
<td>1</td>
</tr>
<tr>
<td>CB</td>
<td>1</td>
<td>JNO</td>
<td>2</td>
<td>SRA</td>
<td>5</td>
</tr>
<tr>
<td>CI</td>
<td>8</td>
<td>JOC</td>
<td>2</td>
<td>SRC</td>
<td>5</td>
</tr>
<tr>
<td>CLR</td>
<td>6</td>
<td>JOP</td>
<td>2</td>
<td>SRC</td>
<td>5</td>
</tr>
<tr>
<td>COC</td>
<td>3</td>
<td>LDCR</td>
<td>4</td>
<td>STCR</td>
<td>4</td>
</tr>
<tr>
<td>CZC</td>
<td>3</td>
<td>LI</td>
<td>8</td>
<td>STST</td>
<td>C</td>
</tr>
<tr>
<td>DEC</td>
<td>6</td>
<td>LIWI</td>
<td>1</td>
<td>SWPB</td>
<td>6</td>
</tr>
<tr>
<td>DECT</td>
<td>6</td>
<td>LWPI</td>
<td>1</td>
<td>S2C</td>
<td>1</td>
</tr>
<tr>
<td>DIV</td>
<td>9</td>
<td>MOV</td>
<td>1</td>
<td>TB</td>
<td>A</td>
</tr>
<tr>
<td>IDLE</td>
<td>7</td>
<td>MOVB</td>
<td>1</td>
<td>TB</td>
<td>A</td>
</tr>
<tr>
<td>IN</td>
<td>6</td>
<td>MYP</td>
<td>9</td>
<td>X</td>
<td>6</td>
</tr>
<tr>
<td>INC</td>
<td>6</td>
<td>NEG</td>
<td>6</td>
<td>XOP</td>
<td>9</td>
</tr>
<tr>
<td>INCT</td>
<td>6</td>
<td>NOP</td>
<td>7</td>
<td>XOR</td>
<td>3</td>
</tr>
<tr>
<td>INV</td>
<td>6</td>
<td>ORI</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JEQ</td>
<td>2</td>
<td>OUT</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TABLE 3 - DIRECTIVE OPERAND FORMATS

<table>
<thead>
<tr>
<th>Directive</th>
<th>Directive Number</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>AORG</td>
<td>1</td>
<td>expression; (address)</td>
</tr>
<tr>
<td>BSS</td>
<td>3</td>
<td>expression;</td>
</tr>
<tr>
<td>BYTE</td>
<td>4</td>
<td>expression₁, . . . , expressionₙ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N&lt;19, if PC is odd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N&lt;20, if PC is even</td>
</tr>
<tr>
<td>DATA</td>
<td>5</td>
<td>expression₁, . . . , expressionₙ; N&lt;10</td>
</tr>
<tr>
<td>DXOP</td>
<td>8</td>
<td>format 6</td>
</tr>
<tr>
<td>END</td>
<td>9</td>
<td>none</td>
</tr>
<tr>
<td>EQU</td>
<td>10</td>
<td>expression;</td>
</tr>
<tr>
<td>EVEN</td>
<td>11</td>
<td>none</td>
</tr>
<tr>
<td>LIST</td>
<td>13</td>
<td>none</td>
</tr>
<tr>
<td>TEXT</td>
<td>17</td>
<td>Literal-list of N characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N&lt;19, if PC is odd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N&lt;20, if PC is even</td>
</tr>
<tr>
<td>/</td>
<td>20</td>
<td>expression;</td>
</tr>
</tbody>
</table>

### NOTES:
1. Slash is equivalent to the AORG directive.
2. Slash must be in column 8.

### LIMITATIONS AND DIFFERENCES WITH THE TI ASSEMBLER

The following (known) differences are listed in no particular order.

1) The input format for CRASS is quite rigid
   - Label - anywhere in columns 1-6
   - Instructions and Directives - must begin in column 8
   - Operand Field - anywhere in columns 13-72.

2) The jump instructions accept the value in the operand field as a quantity of bytes, whereas the TI assembler uses the operand value as a word quantity. The operand value must be in the range -256 to +254 bytes.

3) Registers 0 through 15 must be referred to as "R0," "R1," . . . , "R15"

4) Hollerith (literal) data must be bounded by either a pair of dollar signs ($) or a pair of apostrophies (').

5) The current location may be referred to, in CRASS, with an exclamation point (!). The TI assembler uses a dollar sign ($).
6) DXOP. The format for this directive in CRASS is:

   name DXOP extended operations number;

   The format used by the TI assembler is:

   DXOP name, extended operations number;

   Note that the extended operation name should not be used as an ordinary label.

7) Assembly time constants and other operands must be resolved within two passes of the source program. For example, the following statements will be correctly resolved:

   A EQU B;
   B EQU 2;

   For the following statements, "A" would not be properly resolved:

   A EQU B;
   B EQU C;
   C EQU 3;

8) To redefine the program counter (PC), both CRASS and the TI assembler will accept:

   AORG absolute value;

   CRASS will also accept a slash (in column 8):

   / absolute value;

9) Expression evaluation. Both assemblers evaluate expressions from left to right. However, on the TI assembler, a unary minus is performed first. For example,

   Label1+Value1+(-Value2)

   is legal with the TI assembler but not with CRASS.

10) Logical operators are legal in CRASS but do not appear to be in the TI assembler.

    The operators are:

    .+ Logical OR
    .* Logical AND
    .- Logical NOT (1's complement)
    ./ Modulo

11) Individual BYTE, DATA, and TEXT statements are limited to a total of 20 characters in the operand field, 19 if the current value of the program counter (PC) is odd. The following statements are examples of the BYTE, DATA, and TEXT directives in which the operands contain the maximum number of allowable characters.
BYTE 1, 2, 3, 4, 5, 6, 7, 8, 9,
0, >A, >B, >C, >D, >E, >F,
'W', 'X', 'Y', 'Z';

DATA 'AB', 'CD', 'EF', 'GH', 'IJ', 'KL', 'MN', 'OP', 'QR', 'ST';

TEXT $ABCDEFGHIJKLMNOPQRST$;

12) CRASS requires \(133,200_8 (46,720)\) bytes, or

\(55,500_8 (23,360_{10})\) words plus space for the I/O buffers in order to execute on a PDP-11.

This memory requirement can be reduced should space become a consideration.

ERROR CODES

CRASS separates error conditions into three categories:

1) Multiple label definition
2) Nonexistent instruction name or assembler directive
3) All errors which originate in the operand field, such as:
   a) Undefined memory references
   b) Semantic errors, such as not using a workspace register as the source field operand for input format 5
   c) Exceeding jump displacement limitations
   d) Exceeding 20 characters for the DATA, BYTE, or TEXT directives.

When an error is caught on pass two, a message which describes the line status is sent to the terminal. The line status codes are given in Table 4.

<table>
<thead>
<tr>
<th>Error</th>
<th>Line Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>2</td>
</tr>
<tr>
<td>Instruction</td>
<td>4</td>
</tr>
<tr>
<td>Operand Field</td>
<td>8</td>
</tr>
</tbody>
</table>

TABLE 4 - LINE STATUS ERROR CODES
The line status is cumulative for each category of error found in a statement. Thus, a statement with a multiply-defined label and an incorrect instruction name will have a line status of $2 + 4 = 6$.

PROGRAM DESCRIPTION

DATA STRUCTURES USED BY CRASS

CRASS employs the Graph Information Retrieval System to create, store, and retrieve its key data structures. Both the Identifier and Op Code-Directives are stored in EPAM tree data structures.

These graphs are composed of source node - link - sink node triples as translated by GIRL. Another way of referring to these triples is to say that the source node is related to the sink node by the link. For example, if the following relationships were to be inserted into a graph, using GIRL, they would appear as follows:

1) A is related to C by B
   \[ G \text{ A B C} \]

2) A is related to C and D and E, by B
   \[ G \text{ A B (C, D, E)} \]

3) A is related to C by B and C is related to E by D
   \[ G \text{ A B C D E} \]

4) A is related to C, D, and E, by B and also D is related to A by F
   \[ G \text{ A B (C, D F A, E)} \]

Also discussed in this section are the "label," "token," and "item" (intermediate code) strings which are created in the first pass.

Examples:

1) Instruction-Directive Tree

This structure is created by a GIRL program called CRSGEN. It is executed before CRASS is run and need be re-executed only if either the instruction set or assembler directive set is changed. The entire program is listed in Appendix A.

As an example, the GIRL statement representing the entire subgraph for directives and op codes beginning with the letter "D" is shown:
DATA DIV
DEC DORG
DECT DXOP
DEF

(Note that the dollar signs ($) indicate that, although the particular node represents a unique state within the graph, it does not require a unique name.)

C
G START D $ (A $ T $ A $ DIRECT "5,"
G 1 E $ (C $ (OPFIN("6,"'//06'),
G 2 T $ OPFIN("6,"'//06','//40')),
G 3 F $ DIRECT "6"))
G 4 (I $ V $ OPFIN("9','//3C'),
G 5 O $ R $ G DIRECT "7",
G 6 X $ 0 $ P $ DIRECT "8")

The root node for this graph is called "START." There are two types of terminal nodes. The terminal node for each directive (final link = "DIRECT") contains a unique identifying integer from 1 to 20. For example:

- DATA - 5
- DEF - 6
- DORG - 7
- DXOP - 8

The terminal node for each op code (final link = "OPFIN") describes:

a) which of the twelve formats the operand field and object code may take, and

b) a skeleton code which forms the basis for the object code. For example, the DIV instruction uses format nine and has a skeleton code of "3C." The statement:

DIV R2, R1;

produces "3C42" as the object code.

The nodes which are represented by dollar signs represent uniquely defined random nodes.

2) Identifier Tree

This graph is created largely during the assembler's first pass. When an identifier is first encountered, it is added to the graph, letter by letter. That is, each letter is a potentially new link in the graph.

If the identifier is first encountered in the label field, it is given a value equal to the current location of the program counter (PC) and its status is set to "defined-fullword." If the first encounter is in the operand field, the
identifier is given a value of zero and its status is set to "undefined." The root node for this graph is called "SYMBOL." The following GIRL statements represent a sample subgraph containing a highly correlated set of identifiers:

```
BLACK  BLOCK4
BLOCK  BLOCK5
BLOCK1

G  SYMBOL  B  $  L  $  (A  S  C  $  K  $  STOP  ("address,"  "status")),
G  1       0  $  C  $  K  $  (STOP  ("address,"  "status")),
G  2       ONE  STOP  ("address,"  "status"),
G  3       FOUR  STOP  ("address,"  "status"),
G  4       FIVE  STOP  ("address,"  "status")))
```

New "label," "token," and "item" strings are created for each statement during the assembler's first pass. After each statement is examined during the first pass, the label and token strings are destroyed, but the item string is saved on a file as the intermediate code for that statement.

3) Label string

The label string is composed of a "multivalued list" (MVL) as described by Zaritsky, 3 pages 2-3. The source (root) node is called "STRING" and the link "LABELS." For example, the label "BLOCK1" has the following label string:

```
G  STRING  LABELS  (B,L,O,C,K,1)
```

4) Token string

The token string is the assembler's first attempt at codifying the operand field. The operand field is broken up into "tokens:" arithmetic operators [+ - * /], numbers, alphanumerics, literals, separators [, ; (] and special characters [! @ % > .]. The token string consists of three separate MVL's:

a) The first MVL consists of a source node/link pair, "TOKEN, STRING," and a set of sink nodes consisting of a sequence of token code numbers representing the operand field. The token codes are listed in Table 5.

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>+</td>
<td>16</td>
<td>NUMBER</td>
</tr>
<tr>
<td>9</td>
<td>*</td>
<td>17</td>
<td>ALPHANUMERIC</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>18</td>
<td>@</td>
</tr>
<tr>
<td>11</td>
<td>/</td>
<td>19</td>
<td>%</td>
</tr>
<tr>
<td>12</td>
<td>.</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>!</td>
<td>21</td>
<td>)</td>
</tr>
<tr>
<td>14</td>
<td>;</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>LITERAL.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

12
For example, the operand field:

@TABL->lA(Rl),*R2+;

will result in the following token code string:

G TOKEN STRING (18, 17, 10, 20, 16, 21, 17, 12, 9, 17, 8, 14)

[@, TABL, -, >, lA, (, Rl, , *, R2, +, ;]

b) The second MVL describes each token's position in the input buffer. It consists of a source node/link pair, "TOKEN, BUFPOS." For the preceding example, if the at-sign were placed in column 1, the MVL would appear as follows:

G TOKEN BUFPOS ("1","2","6","7","8","10","11","14","15","16","18","19")

c) The third MVL describes the length of each token. Although this string could have been eliminated and computed from the second MVL, the time savings warranted its inclusion. For the preceding example, the MVL would appear as follows:

G TOKEN LENID ("1","4","1","1","2","1","1","2","1","1","2","1","1")

5) Item String

The token string is reduced to a string of "items" which include:

- arithmetic operators [+ - * /]
- logical operators [.-.*./]
- literals
- separators [, ;]
- numbers
- identifiers
- register types [Ri *Ri @ Ri *Ri+]
- "current PC" operator [!] 
- at-sign [@]

The item codes are listed in Table 6.

<table>
<thead>
<tr>
<th></th>
<th>ITEM TYPE CODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reg</td>
</tr>
<tr>
<td>1</td>
<td>*Reg</td>
</tr>
<tr>
<td>2</td>
<td>@Reg</td>
</tr>
<tr>
<td>3</td>
<td>*Reg+</td>
</tr>
<tr>
<td>4</td>
<td>.+ logical OR</td>
</tr>
<tr>
<td>5</td>
<td>.* logical AND</td>
</tr>
<tr>
<td>6</td>
<td>.- logical NOT</td>
</tr>
<tr>
<td>7</td>
<td>./ modulus</td>
</tr>
<tr>
<td>8</td>
<td>+</td>
</tr>
<tr>
<td>9</td>
<td>*</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>/</td>
</tr>
<tr>
<td>12</td>
<td>,</td>
</tr>
<tr>
<td>13</td>
<td>!</td>
</tr>
<tr>
<td>14</td>
<td>;</td>
</tr>
<tr>
<td>15</td>
<td>literal</td>
</tr>
<tr>
<td>16</td>
<td>number</td>
</tr>
<tr>
<td>17</td>
<td>identifier</td>
</tr>
<tr>
<td>18</td>
<td>@</td>
</tr>
</tbody>
</table>

The item string consists of four separate MVL's:
a) The first MVL consists of a source node, link pair "ITEM, ITMTYP" and the sink nodes consist of a sequence of item codes derived from the token string. The item codes are taken from Table 6.

The token string from the preceding example would result in the following MVL:
```
C ITEM ITMTYP ("18", "17", "10", "16", "0", "12", "3", "14")
[@ TABL - >TA R1 , *R2+ ;]
```

b) The second MVL is the "ITEM,VALUE" pair.

The sink nodes contain the following types of information:

- **registers** - register number
- **identifiers** - assigned address or 0 if undefined
- **numbers** - decimal value
- **literals** - first two characters in ASCII format
- **operators** - 0
- **separators** - 0
- **exclamation point** - 0

In the example being used, the MVL (during pass two) would appear as follows:
```
G ITEM VALUE ("0", "address of TABL", "0", "26", "1", "0", "2", "0")
```

c) The third MVL, "ITEM,BUFPOS", is identical to the "TOKEN,BUFPOS" string.

d) The fourth MVL, "ITEM,LENID", is identical to the "TOKEN,LENID" string.

**FORMING THE OBJECT CODE WORD**

**Instruction Code Skeletons**

The TI9900 microprocessor instructions take up one 16-bit word (four contiguous hexadecimal integers). Associated with each instruction is an instruction code skeleton, described in the 9900 Family Systems Design Book, pages 6-18 through 6-60. This skeleton may consist of one to four hexadecimal integers which form the heart of each resultant object code word. CRASS takes the skeleton and modifies it as per the operand field to create the object code word. For example: The instruction **DIV** has a skeleton instruction code "3C₁₆" (0011 1100₂) and an operand format of 9. It will be shown shortly that the operand field "R2,R1;" translates to "04₂¹₆." Therefore, the statement: **DIV R2,R1;** translates to "3C42" for the final object code.

Note that an object (data) word is created for immediate operands and also that each memory reference requires an object (data) word.
Object Code Formats

Although there are twelve operand field formats, for the purposes of this assembler there are only eight different object code formats, which are listed in Table 7.

<table>
<thead>
<tr>
<th>Operand Format</th>
<th>Operand Format Number</th>
<th>Machine Code Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 a) S,D</td>
<td>1</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC R C D R d R C s R s</td>
</tr>
<tr>
<td>b) S,W</td>
<td>3,9</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC R d R C s R s</td>
</tr>
<tr>
<td>c) S,C</td>
<td>4</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC C R C s R s</td>
</tr>
<tr>
<td>2 a) DIS</td>
<td>2</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC DIS</td>
</tr>
<tr>
<td>b) BIT</td>
<td>A</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC BIT</td>
</tr>
<tr>
<td>3 W,C</td>
<td>5</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC W</td>
</tr>
<tr>
<td>4 S</td>
<td>6</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC R C s R s</td>
</tr>
<tr>
<td>5 none</td>
<td>7</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC</td>
</tr>
<tr>
<td>6 W,IOP</td>
<td>8</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC W</td>
</tr>
<tr>
<td>7 IOP</td>
<td>B</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC</td>
</tr>
<tr>
<td>8 W</td>
<td>C</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IC W</td>
</tr>
</tbody>
</table>

Explanation of abbreviations:
- IC = Instruction Code
- JC = Jump Code
- W = Workspace register
- C = Count
- S = Source
- D = Destination
- R_n, R_d = Register reference, if set to zero, the next word contains a memory reference.
- DIS = Displacement
- RC = Register Code
- \*R1 = 00
- \*R1 = 11
- \*R1 = 10
- \*R1 = 11

15
PROGRAM FLOW

Overview

CRASS requires two passes to create non-relocatable, absolute code. A program listing is given in Appendix B and the variables used are described in Appendix C. The major functions of each pass are:

First pass - Assign address values to labels
- Reduce operand field to an item string
  (intermediate code)
- Obtain instruction format and skeleton
- Attempt to resolve EQU statements

Second pass - Second attempt to resolve EQU statements
- Create and output absolute code
- Provide source and identifier listing if desired

Description and Example

To facilitate the program description which follows, the statement:

```
XYZ MOV @ABC->23C(R2),*R13+; (comments)
```

will be converted to object code as an example.

Before CRASS examines any assembly code:

1) User and scratch files are defined
2) The data graph which describes the instructions and directives is read in.
3) The identifiers R0, R1, ..., R15 are defined as registers.

Begin pass one:

1) Data strings and line statement error flags are re-initialized.
2) The ASCII input line is lexically scanned by subroutine LEXSCN.
   a) The label is converted to a string.
      C STRING LABELS (X,Y,Z)
   b) The instruction is converted to a string.
      C STRING OPCOD (M,O,V)
   c) The operand field is converted to a string
      (Note that the capitalized names are from labeled common /GIRL,C1/).
3) If the statement contains a label, Subroutine ADDNAM is called to test for multiple definition and to add the label to the identifier tree if this was the first occurrence of the label. The label is then placed on a special scratch file.

G SYMBOL X $ Y $ Z $ STOP ("address," "status")

where address = the current value of the PC and status = "defined"

4) Subroutine INSTRU then tries to match the "STRING, OP_COD" list with either a valid instruction or an assembler directive from the Op code - Directive graph. If it is a directive, a unique identifying integer (from 1 to 20) is returned. If it is an instruction, an operand field format and a skeleton code are returned. For the instruction "MOV" these are

format #1
skeleton "CO16"

5) Subroutine TOKSCN then converts the token string to an item string. The item string is the intermediate code for transition from pass one to pass two. The item string for this example is:

@, variable, minus, number, register[type 0], comma, register[type 3], semicolon

Note that the values for the ITEM,STRING come from Table 6.

G ITEM (STRING ("18", "17", "10", "16", "0", "12", "13", "14"),
G VALUE ( 0, address, 0, decimal value, reg. no., 0, reg. no., 0),
G BUFPOS ( . . ),
G LENID ( . . ))

TOKSCN also assigns source and destination register codes. For this example, TOKSCN "sees"

@R2,*R13+
resulting in 
\[ \begin{align*}
R_s &= 2 = 0010_2 \\
R_d &= 13 = 1101_2 \\
RC_s &= 2 = 10_2 \\
RC_d &= 3 = 11_2 
\end{align*} \]

6) ITMSCN is then called to create the machine code words for this statement. From Tables 2, 1, and 7, and page 6-19 of the TI 9900 Family Systems, the first machine code word would be:

<table>
<thead>
<tr>
<th>IC</th>
<th>RC_d</th>
<th>R_d</th>
<th>RC_s</th>
<th>R_s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0123</td>
<td>45</td>
<td>6789</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>11</td>
<td>1101</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

which equals: "CF62_16." If the identifier "ABC" has been defined (that is, it occurred in the label field of a statement prior to this one) as, for example, "23F_16," then the memory reference is computed as \(23F_16 - 23C_16 = 3\). Therefore, the object code created is:

\[ \text{CF62 0003} \]

If the statement containing "ABC" as a label did not occur until after the present one, the memory reference would have to be resolved in the second pass.

7) The intermediate code is then placed on a scratch file by Subroutine LNOUT.

Before pass two is begun, the read/write pointers of the two scratch files containing the intermediate code and the label names are reset to the beginning.

Begin pass two:

1) Data Strings and line statement error flags are re-initialized.

2) Subroutine LINEIN is called to read in the intermediate code for the next statement.

3) Subroutine ITMSCN is called for a second attempt to complete the machine code. If there are no errors, all memory references and "EQ", "DATA," and "BYTE" directives will be resolved at this time.
4) An error checking routine, ERROUT, is then called to determine whether any line statement error flags have been turned on and, if so, to report any errors.

5) If requested by the user (assembler directive "LIST"), Subroutine SRCLST will output a source code listing of the current statement.

6) The object code is then placed on an output file by Subroutine ABSOUT.

After pass two is completed, if "LIST" is requested, the scratch file containing the label names is examined and Subroutine VARLST creates an identifier listing. Otherwise, the program is finished.

BRIEF SUBROUTINE DESCRIPTIONS

The main (driving) program was described in the previous section on program flow. It calls the following subroutines:

- ITMSCN
- ADDRREG
- LEXSCN
- ADDNAM
- INSTRU
- TOKSCN
- ASSIGN[RT-11 system library routine]
- LINEIN
- ITMSCN
- ERROUT
- SCRLST
- ABSOUT
- VARLST
- LINOUT

With a minimum of detail, this section describes the eighteen subroutines and two functions in CRASS.

Subroutine ADDRREG

Function:

To add the identifier names $R_0$, $R_1$, $R_7$, $R_{15}$ to the identifier graph and declare them to be workspace registers.

Called By: Main Routine

Subroutine Called: ADDNAM

Subroutine LEXSCN

Function:

To perform a lexical scan of each input statement to create:

1) Label string
2) Instruction string
3) Token string to represent the operand field.
The tokens would consist of arithmetic operators, numbers, alphanumerics, literals, separators, and special characters.

Called By: Main Routine

Subroutine ADDNAM (NODE, IDENT)

Function:
To search the identifier graph for the requested variable from the NODE, IDENT string and add it to the graph if not found. ADDNAM checks for multiple definition of a label and, if the label is new, saves it on a scratch file so that an identifier listing can be created at the end of the program.

Called By: Main Routine OPFLD ADDREG TOKSCN

Subroutine INSTRU

Function:
To search the Opcode - Directive graph for the requested instruction. If it is not found, an error flag is set and a return is made. If an assembler directive is matched, a unique integer identifying that directive is returned. This integer is used later by Subroutine ITMSCN. If an instruction is matched, its machine code skeleton and its operand field format number are returned.

Called By: Main Routine

Subroutine TOKSCN

Function:
To reduce the token string to an item string as described in the sections on data structures and program flow and also to assign values to these items. The items consist of:

<table>
<thead>
<tr>
<th>Item</th>
<th>Item Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifiers</td>
<td>- Address value or 0 if undefined</td>
</tr>
<tr>
<td>Registers</td>
<td>- Register number and code</td>
</tr>
<tr>
<td>Numbers</td>
<td>- Decimal value</td>
</tr>
<tr>
<td>Literals</td>
<td>- First two ASCII characters</td>
</tr>
<tr>
<td>Arithmetic operators</td>
<td>0</td>
</tr>
<tr>
<td>Logical operators</td>
<td>0</td>
</tr>
<tr>
<td>Separators</td>
<td>0</td>
</tr>
<tr>
<td>Exclamation mark</td>
<td>0</td>
</tr>
<tr>
<td>At-sign</td>
<td>0</td>
</tr>
</tbody>
</table>
Subroutine ITMSCN

Function:
To scan the item string, left to right, to create the final machine code.

a) First Pass - The number of bytes (halfwords) needed for each instruction is determined and the PC is updated by that amount to assign address values to labels. To update the PC, the following assembler directives are examined:

- AORG EVEN
- BSS TEXT
- BYTE "/"
- DATA

The following directives are also examined on the first pass:

- DXOP
- END
- EQU

b) Second Pass - A second attempt is made to resolve EQU directives. Also, the machine code is constructed and placed on an output file.
Subroutine OPFLD (Il)

Function:
To examine a single operand field as delineated by:

beginning to comma
beginning to semicolon
comma to comma
comma to semicolon

where Il is the location in the operand buffer of the lefthand delineator.

For example:

1 1
1 5 0 5
@TABLI(R3),*R2+;

To examine the source (left) operand, Il is set to 0. To examine the destination
(right) operand, Il is set to 11. OPFLD attempts either to extract a register
number or to compute a decimal value for the field. The decimal value is converted
to hexadecimal and placed into a special array.

OPFLD also determines whether an extra object code word must be allotted for a
memory reference.

Called By: ITMSCN

Subroutines Called: DECHEX TMPSTR
COMPUT ADDNAM

Subroutine TMPSTR (ISTLOC, LENGTH)

Function:
To take a sequence of "LENGTH" ASCII characters from the operand field
beginning at location ISTLOC, and convert it to a GIRL multivalued list (MVL). For
example,

Operand Field:

1 1 2 2
1 5 0 5 0 5
@ABC->123(R1),@TABLI(R2);

If ISTLOC = 16,
LENGTH = 5,
the resultant MVL will be:
G STRING STRING (T,A,B,L,ONE)

Called By: TOKSCN
OPFLD
Subroutine VARLST

Function:

If the "LIST" option is requested, to place the status for each identifier and register on the source listing output file.

Called By: Main Routine
Subroutine Called: DECHEX

Subroutine COMPUT (OPRAND, OPRATR)

Function:

To take operands from the two-word array OPRAND () and perform a computation on:

OPRAND(1) OPRATR OPRAND(2)

as determined by the arithmetic or logical operator in OPRATR. The result is placed in OPRAND(1). If this routine is called more than once for a single operand field, the effect is one of left-to-right (no precedence) expression handling. For example, if Subroutine OPFLD is called to handle

3+5*6

COMPUT will be called twice, first to compute 3+5 and then to compute 8*6.

Called By: OPFLD

Subroutine DECHEX (DECNUM, HEX(1))

Function:

To convert the decimal value in DECNUM to a four-digit hexadecimal number. The four digits are converted to ASCII characters and placed in the four-byte array, HEX().

Called By: ITMSCN ABSOUT OPFLD SCRLST VARLST ERRROUTE

Subroutine called: TWOCMP
Subroutine FMTI (ADRCOD(1))

Function:
To take the source and destination register values and the source and destination register codes as input and (except for the instruction code) create the machine code for output format number one. This code is placed in the three-byte array, ADRCOD().

Called By: ITMSCN

Subroutine SRCLST (THISPC)

Function:
If the "LIST" directive is requested, to place the following information on an output file:
- Input Line number
- Address in both decimal and hexadecimal (THISPC)
- Line status (non-zero indicates error)
- Object Code
- Input statement
- Comments (if any)

A sample set of instructions and directives is given in Appendix D. The source and identifier listing created by this subroutine from the instructions and directives of Appendix D are given in Appendix E.

Called By: Main Routine
Subroutine Called: DECHEX

Subroutine ABSOUT

Function:
To place the object code on an output file. This routine is called during pass two for each statement which is not a comment. See Appendix F for the format used for generated object code.

Called By: Main Routine
Subroutine Called: DECHEX

Subroutine ERROUT (THISPC)

Function:
To report to the terminal during the second pass any line statement with an error. A single variable, LNSTAT, is used to describe any errors.
If error free, \( \text{LNSTAT} = 0 \)
If label error, \( \text{LNSTAT} = 2 \)
If instruction error, \( \text{LNSTAT} = 4 \)
If operand error, \( \text{LNSTAT} = 8 \)

For any combination of errors, the value of LNSTAT is cumulative. For example, if a single line has both an instruction and an operand field error, the line status would be reported as "12."

Called By: Main Routine
Subroutine called: DECHEX

Subroutine TWOCPM (HEX(1))
Function:
To convert a hexadecimal number (from HEX()) of four ASCII digits to its two's complement form and place it, again in ASCII form, back into the four-byte array HEX().
Called By: TOKSCN
           DECHEX

Function LVRTSH (WORD, BITS)
Function:
To perform a right logical shift. The content of WORD is moved by the number of bits in BITS.
Called By: ITMSCN

Function LVLFSH (WORD, BITS)
Function:
To perform a left logical shift. The content of WORD is moved by the number of bits in BITS.
Called By: TOKSCN
           ITMSCN

25
UPDATING CRASS

An understanding of GIRL$^2$ is required to update (or correct any errors in) CRASS.

CRASS consists of two source files:

1) CRASS1.GRL - all routines contain a mixture of GIRL and FORTRAN statements. The first line of code defines the GIRS buffer size.

2) CRASS2.FOR - all routines consist entirely of FORTRAN.

If CRASS1.GRL is modified, it must be converted to all-FORTRAN by the GIRL preprocessor.$^2$ It must then be compiled with the following switch:

```
.R FORTRAN
*CRASS1 = CRASS1/N:7
```

No special switches are needed when CRASS2.FOR is compiled:

```
*CRASS2 = CRASS2
```

A copy of the GIRS$^3$ object file GIRS.OBJ is required to link the files into a "SAV" file:

```
.R LINK
*CRASS=CRASS1,CRASS2,GIRS,SYSLIB/F
```

Note that CRASS1.OBJ must be the first object file in the linking sequence.

If the set of instructions or assembler directives is to be modified, CRSGEN.GRL must be accessed. This program generates the graph which describes the entire legal set of instructions and assembler directives. It is written in GIRL and is listed in Appendix A. Of course, any change in the generator must be matched in the CRASS1 semantics routine ITMSCN. Also, since the GIRS buffer size is defined here, it must be identical to the first line of code from CRASS1.GRL.

PROPOSED ADDITIONS OR IMPROVEMENTS

1) More precise diagnostics
2) More flexible input format
3) Removal of the limitation for resolving assembly time constants (EQU directives) in two passes.
4) Allow parenthesis precedence in expression evaluation
5) Removal of, or at least improvement on, the 20 character limitation for the BYTE, DATA, and TEXT statements.

ACKNOWLEDGMENTS

The contributions and helpfulness of J. Carlberg, of DTNSRDC Code 1824, are gratefully acknowledged.
NOSAVE, PRINT, COMMENTS

DEFINE EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST.

1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH.
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT.
3 NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUESTION.
5 V, W, X, Y, Z, START, OPFIN, DIRECT, TOKODE, VARABL, NUMBER, LITERAL

EXECUTE CALL ASSIGN(99, 'SY.CRA.S.QRF', 12)

SET TOKEN CODES FOR ALL TOKENS WHICH MAY BE THE FIRST CHARACTER OF AN "ITEM"

TOKODE PLUS "8"
TOKODE STAR "9"
TOKODE MINUS "10"
TOKODE SLASH "11"
TOKODE COMMA "12"
TOKODE EXCLAM "13"
TOKODE SCOLON "14"
TOKODE LITERAL "15"
TOKODE NUMBER "16"
TOKODE VARABL "17"
TOKODE ATSIGN "18"
TOKODE PERCNT "19"
TOKODE GTTHAN "20"
TOKODE LPAR "21"
TOKODE PERIOD "22"

STORE DIRECTIVE VALUES AND INSTRUCTION FORMATS AND SKELETON CODES

START A *'ATEMP (OPFIN("1", '/10'),
1 B * (OPFIN("1", '/10'),
2 S * OPFIN("6", '/07', '/40'))
AITEMP (I * OPFIN("8", '/02', '/20'),
1 N * D * I * OPFIN("B", '/02', '/40'),
2 O * R * G * DIRECT "1")

START B *'BTEMP (OPFIN("6", '/04', '/40'),
1 E * S * DIRECT "2",
2 L * (OPFIN("6", '/06', '/80'),
3 W * P * OPFIN("6", '/04'))
BTEMP (S * S * DIRECT "3",
1 Y * T * E * DIRECT "4")

START C *'CTEMP (OPFIN("1", '/80'),
1 B * OPFIN("1", '/90'),
2 I * OPFIN("8", '/02', '/80'))
CTEMP (L * R * OPFIN("6", '/04', '/20'),
1 O * C * OPFIN("3", '/20'),
2 Z * C * OPFIN("3", '/24'))

START D *'DTEMP (A * T * A * DIRECT "5",
1 E * (C * (OPFIN("6", '/06'),
2 T * (OPFIN("6", '/06', '/30'),
3 F * DIRECT "6"))
DTEMP (I * V * OPFIN("9", '/30'),
1 O * R * Q * DIRECT "7",
2 X * D * P * DIRECT "B")
G START E$ (N $ D $ DIRECT "9",  
1 G $ U $ DIRECT "10",  
2 V $ E $ N $ DIRECT "11")

G START I$ 'ITEMP (D $ (L $ E $ OPFIN("7", '/03', '/04'),  
1 T $ DIRECT "12"))

G START J$ 'ITEMP (D $ (L $ E $ OPFIN("6", '/05', '/08'),  
1 T $ OPFIN("6", '/05', '/03'),  
2 V $ OPFIN("6", '/05', '/04'))

G START J$ 'ITEMP  
JTEMP (E $ Q $ OPFIN("2", '/13'),  
1 G $ T $ OPFIN("2", '/15'),  
2 H $ (OPFIN("2", '/18'),  
3 E $ OPFIN("2", '/14'))

G START L$ (D $ C $ R $ OPFIN("4", '/30'),  
1 I $ (OPFIN("8", '/02'),  
2 M $ I $ OPFIN("11", '/03'),  
3 S $ T $ DIRECT "13"),  
4 W $ P $ I $ OPFIN("11", '/02', '/03'))

G START M$ (D $ V $ (OPFIN("1", '/00'),  
1 B $ OPFIN("1", '/00'),  
2 P $ Y $ OPFIN("9", '/38'))

G START N$ (E $ Q $ OPFIN("6", '/05'),  
1 D $ P $ OPFIN("7", '/10'))

G START O$ (R $ I $ OPFIN("8", '/02', '/00'),  
1 U $ T $ OPFIN("6", '/02', '/30'),  
2 W $ P $ OPFIN("7", '/03', '/05'))

G START P$ (A $ Q $ E $ DIRECT "14")

G START R$ (E $ F $ DIRECT "15",  
1 O $ R $ Q $ DIRECT "16",  
2 T $ (OPFIN("7", '/04', '/58'),  
3 W $ P $ OPFIN("7", '/03', '/05'))

G START S$ (E $ F $ DIRECT "16",  
1 2 3)
START S *STEMP (OPFIN("1", '/01'),
1 0 *OPFIN("10", '/01'),
2 7 *OPFIN("10", '/1E'))
STEMP E *T D *OPFIN("6", '/07')
STEMP L *A *OPFIN("5", '/0A')
STEMP D *C *(OPFIN("1", '/EO'),
1 B *OPFIN("1", '/F0'))
STEMP R *(A *OPFIN("5", '/0B'),
1 C *OPFIN("5", '/0B'),
2 L *OPFIN("5", '/0F'))
STEMP T *(C R *OPFIN("4", '/34'),
1 S T *OPFIN("12", '/02', '/C0'),
2 W P *OPFIN("12", '/02', '/A0'))
STEMP W *P B *OPFIN ("6", '/06', '/C0')
STEMP Z *C *(OPFIN("1", '/40'),
1 B *OPFIN("1", '/50'))
START T *(B *OPFIN("10", '/1F'),
1 E X T *DIRECT "17",
2 I T L *DIRECT "18")
START U N L *DIRECT "19"
START X *(OPFIN("6", '/04', '/80'),
1 D *(P OPFIN("4", '/2C'),
2 R *OPFIN("3", '/28'))
START SLASH *DIRECT "20"
CALL LV DUMP(0, 0, 99)

WRITE(99) EXCLAM, QUOTE, POUND, DOLLAR, PERCENT, AMPERS, APOST,
1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3 NINE, COLON, SCOLON, LTHAN, EQUALS, GTTHAN, QUEST,
5 V, W, X, Y, Z, START, OPFIN, DIRECT, TOKODE, VARABL, NUMBER, LITERAL

COMPLETE
APPENDIX B

GIRL-FORTRAN PROGRAM LISTING OF CRASS
REAL*4 DEFEXT(2), SCRACH(3)

COMMENTS

LOGICAL*1 LABEL, OPCODE, OPBUF, HEX, HEXCOD, BUF(40), HEXTBL, LINE
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCIIA, ASCIIIF, ASCIIIZ, ASCCHAR
LOGICAL*1 LABERR, DIRERR, OPPER, TPERR, INSTER, DUMMY, DUMB
LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG, 1
   DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG

COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4, 40)
COMMON /ERRFLG/ LABERR, DIRERR, OPPER, TPERR, INSTER, DUMMY
COMMON /NAMFLG/ DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
COMMON /INFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA, 1
   LSTFLG
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES, 1
   MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM, 2
   OPER1, OPER2, ER, ERLABL, WRDNUM, BYNUM, ERROR
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST, 1
   LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH, 2
   ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, 3
   NINE, COLON, SCOLON, LESS THAN, EQUALS, GREATER THAN, 4
   V, W, X, Y, Z
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP, 1
   ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS, 2
   OPFJN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCIIA, ASCIIIF, ASCIIIZ

DIMENSION LABEL(6), OPCODE(4), OPBUF(40), FILL(100)
DIMENSION ASCORL(58), FILLSPC(39)

EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPBUF(1))
EQUIVALENCE (ASCGRL(1), EXCLAM), (HEXCOD(1, 1), BUF(1))

DEFINE STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
   1 ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
   2 LOGOR, LOGAND, LOGNOT, MODULO

DATA LINENO, PC, ERROR, FIRST, SECOND /0, 256, 0, 1, 2/
DATA BLANK, ASCII0, ASCII9, ASCIIA, ASCIIIF, ASCIIIZ
   1 /1H, 1H0, 1H9, 1HA, 1HF, 1H2/
DATA HEXTBL /1H0, 1H1, 1H2, 1H3, 1H4, 1H5, 1H6, 1H7, 1HB, 1H9,
   1 1HA, 1HB, 1HC, 1HD, 1HE, 1HF/
DATA DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG, DIRFLG
   1 /1, 2, 4, B, 16, 32, 64, 128, 256/
DATA DEFEXT/6RASSABS, 3RLST/
DATA SCRACH/3RRK1, 6RUSER, .3RTMP/

EXECUTE
CALL ASSIGN(99, 'SY CRASS QRF', 12)
CALL ASSIGN(14, 'SY USER VAR', 11, 'SCR')

CALL ASSIGN(10, 'SY USER ASS', 11)
CALL ASSIGN(12, 'SY USER ABS', 11)
CALL ASSIGN(13, 'SY USER LST', 11)
C
TYPE 2
2 FORMAT( 'PLEASE ENTER FILE NAMES IN COMMAND STRING FORM'/)
  FILSPC(7)=0
  IF(ICSII(FILSPC,DEFEKT...O) NE 0) STOP 'INVALID COMMAND STRING'
  IF(IASIGN(12,FILSPC(1),FILSPC(2),FILSPC(5),0) NE 0) STOP 'INVALID IASIGN 1'
  IF(FILSPC(7).EQ 0) GO TO 5
  IF(IASIGN(13,FILSPC(6),FILSPC(7),FILSPC(10),0) NE 0) STOP 'INVALID IASIGN 2'
  5 IF(IASIGN(10,FILSPC(16),FILSPC(17),FILSPC(19),32) NE 0) STOP 'INVALID IASIGN 3'
  IF(IASIGN(1,I,SCRACH(1),SCRACH(2),-1,2) .NE. 0) STOP 'BAD SCRATCH FILE REQUEST'
C
READ(99) EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST.
1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT.
3 NINE, COLON, SCOLON, LSTTH, EQUALS, GTTHAN, QUEST,
5 V.W.X.Y.Z, START, OPFIN, DIRECT, TOKODE, VARABL, NUMBER, LITRAL
C
PASS = FIRST
LABERR = FALSE.
ENDFLG = FALSE.
LSTFLG = FALSE.
C
BEGIN NAME TABLE BY CREATING REGISTER NAMES: RO, ..., R15
CALL ADDRREG
C
READ IN NEXT LINE OF CODE
10 LINENO = LINENO + 1
C
READ(10,1)LINE
1 FORMAT(72A1)
C
SET LINE STATUS FLAGS
TYPERR = FALSE.
LABERR = FALSE.
DIRERR = FALSE.
OPERR = FALSE.
INSTER = FALSE.
COMMTS = FALSE.
RESOLV = TRUE.
WRDNUM = 1
BYTNUM = 1
LNSTAT = 0
DIRNUM = 0
C
CLEAR ASCII BUFFERS AND GIRL STRINGS
DO 20 II=-1,40
20 BUF(II) = BLANK
STRING-(LABELS,OPCOD)
TOKEN-(STRING,BUFPOS,LENID)
ITEM-(ITMTYP,VALUE,BUFPOS,LENID)

BREAK LINE STATEMENT UP INTO TOKEN STRINGS
CALL LEIXSCN

C IS THIS ENTIRE LINE A COMMENT?
22 IF(COMNTS) GO TO 40

C DOES THIS STATEMENT CONTAIN A LABEL?
Q STRING+LABELS/30
C
C ADD LABEL TO NAME TREE AND ASSIGN CURRENT PROGRAM COUNTER (PC)
C AS ITS VALUE
    NODE = STRING
    IDENT = LABELS
    CALL ADDNAM(NODE,IDENT)
C
C EXAMINE INSTRUCTION
30 CALL INSTRU
C
C CONVERT OPERAND FIELD FROM A TOKEN STRING TO AN ITEM STRING
C (INTERMEDIATE CODE)
CALL TOKSCN
C
C DETERMINE CORRECT NUMBER OF TARGET WORDS OF ABSOLUTE CODE NEEDED FOR
C THIS INSTRUCTION AND UPDATE PC BY THAT AMOUNT. THIS IS NEEDED IN THE
C FIRST PASS IN ORDER TO ASSIGN CORRECT ADDRESSES TO LABELS.
C DO NOT SCAN ITEM STRING IF INSTRUCTION OR DIRECTIVE WAS IN ERROR.
    IF(INSTER.EQ..TRUE.) GO TO 40
    CALL ITMSCN
C
C OUTPUT INTERMEDIATE CODE TO LUN 11
40 CALL LINOUT
C
C END OF INPUT?
    IF(ENDFLG) GO TO 50
    GO TO 10
C
C BEGIN SECOND PASS
50 PASS = SECOND
    PC = 256
    ENDFLG = .FALSE.
    REWIND 11
    IF(LSTFLG) WRITE(13,100)
100 FORMAT('LINE NO. ADDRESS LINE STATUS OBJ CODE LABEL
      INSTR OP FIELD')
C
C REINITIALIZE LINE FLAGS
60 TYPERR = .FALSE.
    OPERR = .FALSE.
    LABERR = .FALSE.
    DIRERR = .FALSE.
    INSTER = .FALSE.
    RESOLV = .TRUE.
subroutine addreg

common /asemb/ pass, regsrc, regdes, regcod(2), pc, linend, adress, 1   majval, coment, idstat, lnstat, errnum, fmt, dirnum, 2   oper1, oper2, er, erlabl, wrdnum, bytnum, error,  
common /girlrch/ exclam, quote, pound, dollar, percnt, ampers, apost, 1   lpar, rpar, star, plus, comma, minus, period, slash, 2   zero, one, two, three, four, five, six, seven, eight, 3   nine, colon, scolon, lsthn, equals, otthan, quest, 4   atsign, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, 5   u, v, w, x, y, z,  
common /girlr/ string, token, item, bufpos, lenid, labels, opcod, stop, 1   itmtyp, value, symbol, reg, regstr, regats, regpls, 2   varabl, number, literal, logor, logand, lognot, modulo, 3   opfin, direct, start, tokode

dimension numbuff(10)

equivalence(zero, numbuff(1))

c create register name strings ro.... , r15 and place into name table
c tagged as registers and given addresses equal to the register number

c ro - r9
NODE = STRING
IDENT = REG
ADDRESS = -1

STRING REG R
DO 10 11 = 1,10
ADDRESS = ADDRESS + 1
NUM = NUMBUF(II)
STRING REG = 2 NUM
CALL ADDNAM(NODE, IDENT)
10 CONTINUE

C RIO - R15
STRING REG = 2 ONE
DO 20 11 = 1,6
NUM = NUMBUF(II)
ADDRESS = ADDRESS + 1
STRING REG = 3 NUM
CALL ADDNAM(NODE, IDENT)
20 CONTINUE
C COMPLETE

SUBROUTINE LEXSCN

LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, BUF(40), HEXTBL, LINE
LOGICAL*1 BLANK, ASCII10, ASCII19, ASCII2, ASCII3, ASCII7, ASCII8
LOGICAL*1 LABERR, DIRERR, OPER, TYPERR, INST, DUMMY, DUMB
LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSBUF(II), DFGLG, RESFLG,
1 DIRFLG, ENDFLH, COMNTS, EXTRA, LSTFLG, H01, H0L2

COMMON /ASC/
LINE(72), HEX(4), HEXCOD(4,10)
COMMON /ERRFLG/
LABERR, DIRE, OPER, TYPERR, INST, DUMMY
COMMON /NAMFLG/
DEFFLG, LABFLG, RESFLG, MLTFLG, BYTFLG, DUMB
COMMON /LINFNL/
INSFLG, OPRBUF, RESFLG, DIRFLG, ENDFLH, COMNTS, EXTRA,
1 LSTFLG
COMMON /ASSEMB/
PASS, REGSRC, RECGES, REGCOD(2), PC, LINENO, ADDRESS,
1 MAJVAL, COMENT, IDSTAT, LNSTAT, ERNUM, FMT, DIRDIM,
2 OPER, OPER, ER, ERAN, WRDNUM, BYTNUM, ERROR
COMMON /DATUM/
FIRST, SECOND, HEXTBL(16)
COMMON /GIRL/
EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3 NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
5 V, W, X, Y, Z
COMMON /GIRL/
STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCODE, STOP,
1 IMTP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGP,
2 VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/
BLANK, ASCII10, ASCII19, ASCII2, ASCII3, ASCII7, ASCII8

DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)
DIMENSION ASCORL(58)

EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(0), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))
EQUIVALENCE (ASCORL(1), EXCLAM), (HEXCOD(1,1), BUF(1))

C THIS ROUTINE IS A LEXICAL SCAN DESIGNED TO CREATE A BASIC TOKEN STRING
C CONSISTING OF OPERATORS, NUMBERS, ALPHANUMERICS, LITERALS, AND SPECIAL
C CHARACTERS. TWO ASSOCIATED STRINGS RELATE THE TOKEN STRING TO THE
C ORIGINAL ASCII STRING BY INDICATING EACH TOKEN'S STARTING POSITION IN
C ARRAY "OPRBUF" AND ALSO THAT TOKEN'S LENGTH
C A LABEL STRING IS CREATED IN ORDER TO PLACE THAT LABEL INTO A NAME
C TABLE CALLED AN "EPAM" TREE.
C AN INSTRUCTION STRING IS CREATED IN ORDER TO SEARCH A PREVIOUSLY
C CREATED EPAM TREE WHICH DESCRIBES THE EXISTING INSTRUCTIONS AND
C DIRECTIVES
C
C *'-LITERAL:  HOL1
   HOL1 = FALSE
C "'-LITERAL:  HOL2
   HOL2 = FALSE
C
C CHECK FOR LABEL AND CREATE LABEL STRING
   DO 10 I1=1,6
      ASCHAR = LABEL(I1)
      IF(ASCHAR .EQ. BLANK) GO TO 20
      GRLCHR = ASCGRL(ASCHAR-BLANK)
   C
C IS THE ENTIRE LINE A COMMENT STATEMENT?
   IF(GRLCHR .EQ. STAR) GO TO 15
   STRING LABELS GRLCHR
10  CONTINUE
   GO TO 20
15  COMNTS = TRUE.
   RETURN
C
C CREATE INSTRUCTION STRING
20  DO 30 I1=1,4
     J1 = I1
     ASCHAR = OPCODE(J1)
     IF(ASCHAR .EQ. BLANK) GO TO 35
     GRLCHR = ASCGRL(ASCHAR-BLANK)
   G STRING OPCOD GRLCHR
30  CONTINUE
C
C CHECK FOR EXTRANEOUS CHARACTERS IN INSTRUCTION FIELD
35  IF(J1 .EQ. 4) GO TO 40
     J1 = J1 + 1
     ASCHAR = OPCODE(J1)
     IF(ASCHAR .NE. BLANK) INSTER = .TRUE.
     GO TO 35
C
C CREATE TOKEN STRING
40  I1 = 0
    GRLCHR = 0
C
C ELIMINATE INITIAL BLANKS
45  I1 = I1 + 1
   IF(I1 .GT. 60) RETURN
   ASCHAR = OPRBUF(I1)
   IF(ASCHAR .EQ. BLANK) GO TO 45
C
C FIRST CHARACTER FOUND
   LENGTH = 0
   LSTSYM = GRLCHR
   GRLCHR = ASCGRL(ASCHAR-BLANK)
C
C IS IT A LITERAL?
   IF(GRLCHR .EQ. DOLLAR) GO TO 60
   IF(GRLCHR .EQ. APOST) GO TO 70
C
C IS IT A NUMBER?
   IF(LSTSYM .EQ. GTTHAN) GO TO 80
   IF((ASCHAR .GE. ASCII9) AND. (ASCHAR .LE. ASCII9)) GO TO 80
C
C IS IT AN ALPHANUMERIC?
   IF((ASCHAR .GE. ASCIIA) .AND. (ASCHAR .LE. ASCIIZ)) GO TO 90
C
C CHARACTER IS EITHER AN OPERATOR OR A SPECIAL SYMBOL
   LENGTH = LENGTH + 1
   ISTLOC = ISTLOC + 1
   G55 TOKEN STRING GRLCHR
   IF(GRLCHR .NE. SCOLON) GO TO 45
   C SEMI-COLON ENDS THE STATEMENT ALL FOLLOWING CHARACTERS ARE COMMENTS
   COMMENT = ISTLOC
   RETURN
   C
   C * LITERAL *
60   BOUND = DOLLAR
   GO TO 71
   C
   C * LITERAL *
70   BOUND = APOST
71   TOKEN STRING LITERAL
   ISTLOC = ISTLOC + 1
   ! = ! + 1
   IF(!! .GT. 60) GO TO 55
   ASCCHAR = OPRBUF(!)
   GRLCHR = ASCGRL(ASCCHAR-DLANK)
   IF(GRLCHR .EQ. BOUND) GO TO 55
   LENGTH = LENGTH + 1
   GO TO 75
   C NUMBER -- (HEX, DECIMAL, OR BINARY)
80   BOUND = ASCIIF
81   TOKEN STRING NUMBER
   GO TO 91
   C
   C ALPHANUMERIC -- (IDENTIFIER OR REGISTER)
90   BOUND = ASCIIZ
91   TOKEN STRING VARIABLE
   ISTLOC = ISTLOC + 1
   ! = ! + 1
   LENGTH = LENGTH + 1
   ASCCHAR = OPRBUF(!)
   IF((ASCCHAR .GE. ASCIIO) .AND. (ASCCHAR .LE. ASCII9)) GO TO 95
   IF((ASCCHAR .GE. ASCIIA) .AND. (ASCCHAR .LE. BOUND)) GO TO 95
   ! = ! - 1
   GO TO 55
   C
   C COMPLETE
   C
   C $ SUBROUTINE ADDNAM(NODE, IDENT)
   LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
   LOGICAL*1 DEFFLG, REQFLG, MLTFLG, BYTFLG, INSFLG, OPFLG, RESFLG,
1       DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
   C
   COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
   COMMON /NAMEFLG/ DEFFLG, LABFLG, REQFLG, MLTFLG, BYTFLG, DUMB
   COMMON /LINFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA.
1       LSTFLG
   COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES.
1       MAJVAL, COMMENT, ISTAT, LNSTAT, ERRNUM, FMT, DIRNUM.
2       OPER1, OPER2, ERLABL, WRDNUM, BYTNUM, ERROR
   COMMON /GRL/ STRING, TOKEN, ITEM, BUFFPOS, LENID, LABELS, OPCOD, STOP.
1       ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS.
2       VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO.
3       OPFIN, DIRECT, START, TOKODE
   COMMON /ASCI/ BLANK, ASCIIO, ASCII9, ASCII1A, ASCII1F, ASCII1Z
   C
DIMENSION NAME(6)

THIS ROUTINE SEARCHES THE NAME TABLE FOR THE REQUESTED IDENTIFIER
AND ADDS IT TO THE TABLE IF NOT FOUND. AN ADDRESS IS RETURNED UNLESS
THE IDENTIFIER IS NOT YET DEFINED (JUMP AHEAD).

DO 10 I1 = 1, 6
10 NAME(I1) = 0
   IF(IDENT NE. REG) ADDRES = 0
   ER = SYMBOL
   IDSTAT = 0
   POS = 0

EXAMINE INPUT STRING
G20 NODE = IDENT "POS=POS+I"/30 'GRLCHR
   NAME(POS) = GRLCHR
   ER = GRLCHR/65 'ER/20
C
IF FAILURE, IDENTIFIER IS A SUBSET OF AN EXISTING NAME
C (EG. "VAR" > "VARI")
G30 ER + STOP(/40 'ADDRES, .2 'IDSTAT)
C
NAME FOUND
IF(IDENT NE. LABELS) RETURN
   ERLABL = ER

TEST FOR MULTIPLE DEFINITIONS
IF((IDSTAT AND. DEFFLG) EQ. 0) GO TO 40
C
USE MOST RECENT DEFINITION
   ERROR = ERROR + 1
   IDSTAT = IDSTAT OR. MLTFLG
   LABERR = .TRUE.
C
UPDATE ADDRESS AND IDENTIFIER STATUS
40 IF(IDENT, EQ. REG) IDSTAT = IDSTAT OR. REGFLG OR. DEFFLG
   IF(IDENT NE. LABELS) GO TO 45
   ADDRESS = PC
   ERLABL = ER
   IDSTAT = IDSTAT OR. DEFFLG
G45 ER STOP(- 1 "ADDRES", -2 "IDSTAT")
C
OUTPUT IDENTIFIER TO NAME FILE
IF( NOT. LABERR) WRITE(14) NAME
RETURN
C
NAME MUST BE ADDED TO THE TREE. ADD NEXT CHARACTER
G60 NODE = IDENT "POS=POS+I"/40 'GRLCHR
   NAME(POS) = GRLCHR
G65 ER GRLCHR *'ER
   GO TO 60
G
COMPLETE
C

* SUBROUTINE INSTRU
L.
LOGICAL*1 LABERR, DIRERR, OPPER, TYPERR, INSTER, DUMMY, DUMB

COMMON /ERRFLG/ LABERR, DIRERR, OPPER, TYPERR, INSTER, DUMMY
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1 MAJVAL, COMENT, IDSTAT, LINSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /GIRL/
STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
1 ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
SUBROUTINE TOKSCN

LOGICAL*1 LABEL, OPDCE, OPBUPF, HEX, HECOD, BUF(40), HECODL, NILE, NEG
LOGICAL*1 BLANK, ASCII, ASCII9, ASCIIA, ASCII12, ASCII12, ASCIIAR
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
LOGICAL*1 DEFFLG, REGFLG, MLTFLO, BYTFLO, LABCLO, INSFLG, OPFLG, RESFLG,
1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG

COMMON /ASC/ LINE(72), HEX(4), HECOD(4, 10)
COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/ DEFFLG, LABCLO, RESFLG, MLTFLO, BYTFLO, DUMB
COMMON /LINFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA
1 LSTFLG
COMMON /ASSEMB/ PASS, REGSRE, REQDES, REQCOD(2), PC, LINOEO, ADDRRES
1 MAJVAR, COMENT, IDSTAT, LNSTAT, ERNUM, FMT, DIRNUM
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /DATUM/ FIRST, SECOND, HEXTMB(16)
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
1 LPAR, RPAMR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3 NINE, COLON, SCOLON, LSTTHAN, EQUAL, GTHAN, QUEST,
5 V, W, X, Y, Z
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPDCE, STOP,
1 ITMTYP, VALUE, SYMBOL, REQ, REGSTR, REGALS, REQFLG,
2 VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCIIA, ASCII12, ASCII12

DIMENSION LABEL(6), OPDCE(4), OPBUPF(60)

EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(4), OPDCE(11))
EQUIVALENCE (LINE(13), OPBUPF(1))

DATA DEC, BIN, HEXA /1, 2, 3/

THIS ROUTINE
C 1) COMBINES (REDUCES) THE TOKENS FROM LEXSCN INTO "ITEMS"
C SUCH AS REGISTER TYPES, LOGICAL AND MATHEMATICAL OPERATORS,
C VARIABLES, NUMBERS, LITERALS, COMMAS, SEMI-COLONS,
C AND EXCLAMATION MARKS
C 2) ASSIGNS: REGISTERS > REGISTER NUMBERS
C OPERATORS > 0
C VARIABLES > ADDRESS VALUE OR 0 IF UNDEFINED
C NUMBERS > DECIMAL VALUE
C LITERALS > FIRST TWO CHARACTERS
C COMMAS > 0
C SEMICOLON > 0
C EXCLAM MK > 0
C
C ITEM TYPE CODES: 0-18
C TOKEN CODES 8-22
C
C 0 REG 12
C 1 *REG 13
C 2 @REG 14
C 3 *REG+ 15 LITERAL
C 4 + LOG OR 16 NUMBER
C 5 * LOG AND 17 IDENTIFIER
C 6 - LOG NOT 18 @
C 7 / MODULUS 19 %
C 8 + 20
C 9 * 21 (
C 10 = 22
C 11 /
C
C INITIALIZE REGISTER CODES AND SOURCE AND DESTINATION REGISTER VALUES
C REGISTER CODES ARE 0 - 3
C
C EXTRACT TOKEN
C 11 = 0
C COMMAS = 0
C
C DEFAULT ASSUME NUMBERS ARE DECIMAL
C NUMTYP = DEC
C
C EXAMINE TOKEN STRING FOR FIRST CHARACTER OF ITEM
g10 token + string "11=111" /return 'token
C token +(bufpos 11 'istloc. lenid 11 'length)
C
C OPERATOR AND DELIMITER VALUES DEFAULT TO 0 IN ORDER TO PAD
C ITEM - VALUE LIST
C VAL = 0
C
C BEGIN ITEM TYPE AND VALUE DEFINITIONS
C tokode + toke 'itmval
C index = itmval - 7
C
C TO 1000, 1000, 2000, 2000, 2000, 2000, 2000, 300, 400, 500,
C LIT NUM VAR
C 600 700 800 900 1000) index
C
C C
**C**** ASTERISK -- (STAR)
C LOOK AHEAD, IF NOT A REGISTER, TREAT AS ARITHMETIC OPERATOR
C
100  TEMPI = I1 + 1
Q  TOKEN+(STRING, TEMPI'NEXT, BUFPOS, TEMPI'NXTLOC, LENID, TEMPI'NXTLEN)
IF(NEXT .NE. VARABL) GO TO 2000
C
C NEXT TOKEN IS EITHER A REGISTER OR IDENTIFIER
C PLACE ON TEMPORARY STRING
CALL TMPSTR(NXTLOC, NXTLEN)
CALL ADDNAM(STRING, STRING)
IF((IDSTAT .AND. REGFLG) .EQ. 0) GO TO 2000
C
C VARIABLE IS A REGISTER
LENGTH = LENGTH + NXTLEN
VAL = ADDR
I1 = I1 + 1
C
C ASSIGN SOURCE OR DESTINATION REGISTER VALUE
IF(COMMAS .GE. 1) GO TO 120
REGSRC = ADDR
GO TO 130
120 REGDES = ADDR
C
C REGISTER MAY HAVE THIS FORM: *REG*
130  TEMP2 = TEMPI + 1
Q  TOKEN + STRING, TEMP2 'LAST
C
C SET REGISTER CODE
ITMVAL = 1
IF(LAST .NE. PLUS') GO TO 140
C *R*
ITMVAL = 3
I1 = I1 + 1
LENGTH = LENGTH + 1
140  REGCOD(COMMAS + 1) = ITMVAL
GO TO 2000
C
C **** COMMA
C
200  COMMAS = COMMAS + 1
GO TO 2000
C
C **** LITERAL -- PLACE UP TO FIRST TWO CHARACTERS INTO "VAL"
C
300  VAL = OPRBUF(ISTLOC)
IF(LENGTH .GT. 1) VAL = LVLFSH(VAL, 8) OR. OPRBUF(ISTLOC + 1)
GO TO 2000
C
C **** NUMBER -- PLACE DECIMAL VALUE INTO VAL
C
400  GO TO (410, 430, 450) NUMTYP
C
C DECIMAL
410  DO 420 KI = 1, LENGTH
MI = KI + ISTLOC - 1
VAL = 10 * VAL + (OPRBUF(MI) - ASC110)
420  CONTINUE
GO TO 2000
C
C BINARY
430  DO 440 KI = 1, LENGTH
MI = KI + ISTLOC - 1
VAL = 2 * VAL + (OPRBUF(MI) - ASC110)
440  CONTINUE

42
GO TO 2000

C C HEXADECIMAL
C FIRST CHECK FOR VALUES GT 32767
450   NEG = .FALSE.
     IF((LENGTH .NE. 4) OR. (DPRBUF(ISTLOC) .LE. ASCII0 + 7))
     I GO TO 453
     NEG = .TRUE.
     CALL TWOCMP(DPRBUF(ISTLOC))
453   DO 460 KI = 1, LENGTH
     HEXCHR = DPRBUF(KI + ISTLOC - 1)
     DO 455 LI = 1, 16
     J1 = LI
     IF(HEXCHR .EQ. HEXTBL(J1)) GO TO 457
455   CONTINUE
457   DECHAR = J1 - 1
     VAL = 16 * VAL + DECHAR
460   CONTINUE
     IF(.NOT. NEG) GO TO 2000
     VAL = -VAL
     CALL TWOCMP(DPRBUF(ISTLOC))
     GO TO 2000

C **** IDENTIFIER -- COULD BE A REGISTER (CODE = 0)
C
C PLACE ON TEMPORARY STRING
500   CONTINUE
     CALL TMPSTR(ISTLOC, LENGTH)
     CALL ADDNAM(STRING, STRING)
     VAL = ADDRES

C REGISTER?
     IF((IDSTAT .AND. REGFLG) .EQ. 0) GO TO 2000
     ITMVAL = 0
510   IF(COMMAS .GE. 1) GO TO 520
C SOURCE
     REGSRC = ADDRES
     GO TO 530
C
C DESTINATION
520   REGDES = ADDRES
C
C CHECK FOR RIGHT PARENSIS
530   TEMP = II + 1
     II = TEMP
     GO TO 2000

C **** AT-SIGN -- 0, NEXT ITEM COULD BE A NUMBER, REGISTER, OR VARIARLE
C
600   TEMP = II + 1
     G TOKEN+(STRING TEMP 'NEXT, BUFPOS TEMP 'NXTLOC, LENID TEMP'NXTLEN)
     REGCOD(COMMAS + 1) = 2
     IF(NEXT .NE. VARABL) GO TO 2000
C
C CHECK FOR REGISTER
     CALL TMPSTR(NXTLOC, NXTLEN)
     CALL ADDNAM(STRING, STRING)
C
C REGISTER?
     IF((IDSTAT AND. REGFLG) .EQ. 0) GO TO 2000
     VAL = ADDRES
     II = TEMP
     LENGTH = LENGTH + NXTLEN
     ITMVAL = 2
GO TO 510

C **** PERCENT -- % NEXT ITEM IS A BINARY NUMBER
C 700 NUMTYP = BIN
GO TO 10
C
C **** GTTHAN -- > NEXT ITEM IS A HEXADECIMAL NUMBER
C 800 NUMTYP = HEXA
GO TO 10
C
C **** LEFT PAREN -- ( NEXT ITEM MUST BE A REGISTER, CODE = 2
C 900 LENGTH = 0
GO TO 600
C
C **** PERIOD -- . LOGICAL OPERATOR OR MODULO.
C NEXT TOKEN MUST BE + * - /
1000 11 = 11 + 1
LENGTH = 2
G TOKEN + STRING 'TOKE
G TOKODE + TOKE 'ITMVAL
ITMVAL = ITMVAL - 4
C
C **** PLACE VALUES INTO ITEM LISTS
C 2000 ITEM (ITMYP "ITMVAL", VALUE "VAL")
C ITEM (BUFFPOS "ISTLOC", LENID "LENGTH")
C
C DEFAULT ASSUME NUMBERS ARE DECIMAL
NUMTYP = DEC
C
C RETURN IF SEMI-OLON
IF (TOKE EQ SCOLON) RETURN
GO TO 10
C
C COMPLETE
C
C SUBROUTINE ITMSCN
C
LOGICAL*1 LABEL, OPCODE, OPRDF. HEX, HEXTBL, LINE, MASHA
LOGICAL*1 BLANK, ASC10, ASC19, ASC1A, ASC1F, ASC1Z, ASCHAR, TEMP2
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, DUMB
LOGICAL*1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG, ADRCOD(3)
C
COMMON /ASC/ LINE(72), HEX(4), HEXTBL(4, 10)
COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/ DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
COMMON /LINFLG/ INSLFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA
1 LSTFLG
COMMON /ASPEN/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES
1 MAJVAL, COMENT, IDSTAT, LSTAT, ERRNUM, FMT, DIRNUM
2 OPER1, OPER2, ER. ERLABL, WRDNUM, BYNUM, ERROR
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRL/ EXCLAM, QUOTE, POUND, DOLLAR, PERCENT, AMPERS, APOST.
1 LFAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT
3 NINE, COLON, SCOLON, LTHAN, EQUALS, GTTHAN, QUESTION
5 V, W, X, Y, Z
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFFPOS, LENID, LABELS, OPRDF. (10)
1 ITMYP, VALUE, SYMBOL, REG, ADDR, HEXTBL, REGFLG,
VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII9, ASCIIA, ASCII9, ASCIIA
C
DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)
C
EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))
C
DATA EVEN, ODD, SOURCE, DEST, MASK4, MASK8 /0, 1, 0, 1, "17", "377/
DATA REGISTALL, NUMROMMALITERL /0, -1, 16, 12, 15/
C
THIS ROUTINE SCANS THE ITEM STRING LEFT TO RIGHT.
A) FIRST PASS - THE NUMBER OF TARGET WORDS NEEDED FOR EACH INSTRUCTION IS COUNTED AND THE PC UPDATED BY THAT AMOUNT SO THAT LABELS MAY BE GIVEN CORRECT ADDRESS VALUES PRIOR TO THE SECOND PASS.
B) SECOND PASS - THE TARGET MACHINE CODE IS CREATED.
C
UP TO TEN WORDS MAY BE CREATED BY A DIRECTIVE FOR EXAMPLE "DATA" WITH TEN OPERANDS OR "BYTE" WITH TWENTY OPERANDS UP TO THREE WORDS MAY BE CREATED BY AN INSTRUCTION.
C
LEFT TO RIGHT ARITHMETIC WITH NO PARENTHESES IS ALLOWED
C
BYTNUM = 1
WRDNUM = 1
IF(DIRNUM .NE. 0) GO TO 2000
BYTNUM = 2
PC = PC + 2
GO TO (100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200) FMT
C
**** FORMAT 1 S.D
C
100 DESTIN = ALL
C
F FORM FIRST HEX WORD
110 CALL FM1I(ADRCOD(1))
IF(PASS .EQ. FIRST) GO TO 120
TEMP1 = LVRSH(OPER1, 8)
TEMP2 = TEMP1
HEXCOD(1, 1) = TEMP2
TEMP1 = (OPER1 AND MASK8)
TEMP2 = TEMP1 - ASCII0
IF(TEMP2 .GT. 9) TEMP2 = TEMP1 - ASCIIA + 10
HEXCOD(2, 1) = HEXTBL(ADR COD(1) + TEMP2 + 1)
HEXCOD(3, 1) = HEXTBL(ADR COD(2) + 1)
HEXCOD(4, 1) = HEXTBL(ADR COD(3) + 1)
C
C COMPUTE SOURCE AND DESTINATION FIELDS
120 IF(FMT .EQ. 4) RETURN
11 = 0
FIELD = SOURCE
125 CALL OPFLD(11)
C
C DESTINATION FIELD IS RESTRICTED TO SIMPLE REGISTERS FOR FORMATS 3 AND 9
IF(FIELD .EQ. SOURCE) GO TO 130
IF(DESTIN .EQ. ALL) GO TO 130
IF(DESTIN .NE. MAJVAL) OPERR = TRUE
C
C WILL EXTRA WORDS BE NEEDED BEYOND THE BASIC INSTRUCTION WORD?
130 IF(.NOT. EXTRA) GO TO 150
WRDNUM = WRDNUM + 1
BYTNUM = BYTNUM + 2
PC = PC + 2
IF(PASS EQ. FIRST) GO TO 150
DO 140 J1 = 1, 4
140 HEXCOD(J1, WRDNUM) = HEX(J1)
150 IF(FIELD EQ. DEST) RETURN
FIELD = DEST
GO TO 125

C **** FORMAT 2 DISPLACEMENT -128 >= DISP >= 127 (JUMP STATEMENTS)
C 200 THSLOC = PC - 2
IF(PASS EQ. FIRST) RETURN
C
C COMPUTE ABSOLUTE ADDRESS
II = 0
CALL OPFLD(II)
C
C COMPUTE RELATIVE DISPLACEMENT IN WORDS
DISP = ((ADDRS - THSLOC) - 2) / 2
C
C IF DISPLACEMENT EXCEEDS LIMIT, SET ERROR FLAG
C AND USE DISPLACEMENT MODULO(128)
C
C CONVINSr ANTRACID DISPLACEMENT TO HEX
CALL DECHEX(DISP, HEX(1))
TEMP1 = LVRTSH(OPER1, 8)
TEMP2 = TEMP1
HEXCOD(1, 1) = TEMP2
TEMP1 = (OPER1 AND MASKB)
TEMP2 = TEMP1
HEXCOD(2, 1) = TEMP2
HEXCOD(3, 1) = HEX(3)
HEXCOD(4, 1) = HEX(4)
RETURN
C
C **** FORMAT 3 S.W DESTINATION MUST BE A SIMPLE WORKSPACE REGISTER
C 300 DESTIN = REGIST
GO TO 110
C
C **** FORMAT 4 S.C DESTINATION MUST BE AN INTEGER FROM 0 TO 15
C 400 II = 0
FIELD = SOURCE
425 CALL OPFLD(II)
IF(FIELD EQ. SOURCE) GO TO 430
C
C CHECK DESTINATION AGAINST 0-15 VALUE INTEGER
IF(MAJVAL NE. NUM) OPERR = TRUE
IF((ADDRS LT 0) OR (ADDRS GT 15)) OPERR = TRUE
REGDES = ADDR
GO TO 450
C
C EXTRA TARGET WORD NEEDED-
430 IF( NOT EXTRA) GO TO 450
BYTNUM = BYTNUM + 2
WRDNUM = WRDNUM + 1
PC = PC + 2
IF(PASS EQ. FIRST) GO TO 450
DO 440 J1 = 1, 4
440 HEXCOD(J1, WRDNUM) = HEX(J1)
450 IF(FIELD EQ. DEST) GO TO 110
FIELD = DEST
GO TO 425
C
C **** FORMAT 5 W.N
C SOURCE FIELD MUST CONTAIN A SIMPLE WORKSPACE REGISTER
C DESTINATION FIELD MUST CONTAIN AN INTEGER FROM 0 TO 15
500 IF(PASS EQ FIRST) RETURN
   II = 0
   CALL OPFLD(II)
C
C SOURCE FIELD -- W
IF(MAJVAL NE REGIST) OPERR = TRUE
   TEMP1 = LVRTSH(OPER1, B)
   TEMP2 = TEMP1
   HEXCOD(1, 1) = TEMP2
   TEMP1 = (OPER1 AND MASK8)
   TEMP2 = TEMP1
   HEXCOD(2, 1) = TEMP2
   HEXCOD(4, 1) = HEX(4)
C
C DESTINATION FIELD -- 0 <= N <= 15
   CALL OPFLD(II)
   IF(MAJVAL NE NUM) OPERR = TRUE.
   HEXCOD(3, 1) = HEX(4)
   RETURN
C
C **** FORMAT 6 S
C
600   II = 0
   CALL OPFLD(II)
   IF(EXTRA) PC = PC + 2
   IF(PASS EQ FIRST) RETURN
C
   TEMP1 = LVRTSH(OPER1, B)
   TEMP2 = TEMP1
   HEXCOD(II) = TEMP2
   TEMP1 = (OPER1 AND MASK8)
   TEMP2 = TEMP1
   HEXCOD(2, 1) = TEMP2
   TEMP1 = LVRTSH(OPER2, B)
   IF(TEMP1 EQ 0) TEMP1 = ASCII0
   TEMP2 = TEMP1 - ASCII0
   IF(TEMP2 GT 9) TEMP2 = TEMP1 - ASCIIA + 10
   HEXCOD(3, 1) = HEXTBL(REGSRC + TEMP2 + 1)
C
   IF(EXTRA) GO TO 610
C
C NO EXTRA WORD IS NEEDED FOR A SIMPLE REGISTER
   HEXCOD(4, 1) = HEX(4)
   RETURN
C
C EXTRA WORD FOR LITERAL, NUMBER, OR MEMORY LOCATION
610   HEXCOD(4, 1) = HEXTBL(REGSRC + 1)
   BYTNUM = BYTNUM + 2
   WRDNUM = WRDNUM + 1
   DO 620 J1 = 1, 4
   620   HEXCOD(J1, WRDNUM) = HEX(J1)
   RETURN
C
C **** FORMAT 7 N/A
C
700   IF(PASS EQ FIRST) RETURN
   TEMP1 = LVRTSH(OPER1, B)
   TEMP2 = TEMP1
   HEXCOD(1, 1) = TEMP2
   TEMP1 = (OPER1 AND MASK8)
   TEMP2 = TEMP1
   HEXCOD(2, 1) = TEMP2
47
TEMP1 = LVRTSH(OPER2, B)
IF(TEMP1 .EQ. 0) TEMP1 = ASCIIO
TEMP2 = TEMP1
HEXCOD(1, 1) = TEMP2
TEMP1 = (OPER2 AND MASK8)
IF(TEMP1 .EQ. 0) TEMP1 = ASCIIO
TEMP2 = TEMP1
HEXCOD(4, 1) = TEMP2
RETURN

C
C **** FORMAT 8 W. IOP
C
800 PC = PC + 2
IF(PASS .EQ. FIRST) RETURN
C
C SOURCE MUST BE A SIMPLE REGISTER
C
C CONSTRUCT FIRST WORD
C
I1 = 0
CALL OPFLD(I1)
IF(MAJOR .NE. REGIST) OPERR = TRUE
TEMP1 = LVRTSH(OPER1, B)
TEMP2 = TEMP1
HEXCOD(1, 1) = TEMP2
TEMP1 = (OPER1 AND MASK8)
TEMP2 = TEMP1
HEXCOD(2, 1) = TEMP2
TEMP1 = LVRTSH(OPER2, B)
IF(TEMP1 .EQ. 0) TEMP1 = ASCIIO
TEMP2 = TEMP1
HEXCOD(3, 1) = TEMP2
HEXCOD(4, 1) = HEX(4)

C
C CONSTRUCT IMMEDIATE OPERAND DO NOT RESET ITEM STRING POINTER I1
CALL OPFLD(I1)
BYTNUM = BYTNUM + 2
WRDNUM = WRDNUM + 1
DO 810 J1 = 1, 4
810 HEXCOD(J1, WRDNUM) = HEX(J1)
RETURN
C
C **** FORMAT 9 S.W DESTINATION MUST BE A SIMPLE WORKSPACE REGISTER
C
900 DESTIN = REGIST
GO TO 110
C
C **** FORMAT A BIT # -- MUST BE A POSITIVE INTEGER, VALUE IS
C
1000 IF(PASS .EQ. 1) RETURN
I1 = 0
CALL OPFLD(I1)
IF(MAJOR .NE. NUM) OPERR = TRUE.
IF(ADDRESS .LT. 0) OPERR = TRUE.
TEMP1 = LVRTSH(OPER1, B)
TEMP2 = TEMP1
HEXCOD(1, 1) = TEMP2
TEMP1 = (OPER1 AND MASK8)
TEMP2 = TEMP1
HEXCOD(2, 1) = TEMP2
HEXCOD(3, 1) = HEX(3)
HEXCOD(4, 1) = HEX(4)
RETURN
C
C **** FORMAT B IOP
C
1100 PC = PC + 2
IF(PASS .EQ. FIRST) RETURN
TEMPI = LVRTSH(OPER1,B)
TEMP2 = TEMP1
HEXCOD(1,1) = TEMP2
TEMP1 = (OPER1 AND MASKB)
TEMP2 = TEMP1
HEXCOD(2,1) = TEMP2
TEMP1 = LVRTSH(OPER2,B)
IF(TEMP1 EQ. 0) TEMP1 = ASCII
TEMP2 = TEMP1
HEXCOD(3,1) = TEMP2
TEMP1 = (OPER2 AND MASKB)
IF(TEMP1 EQ. 0) TEMP1 = ASCII
TEMP2 = TEMP1
HEXCOD(4,1) = TEMP2

C
C FORM IOP
II = 0
CALL OPFLD(I1)
BYTNUM = BYTNUM + 2
WRDNUM = WRDNUM + 1
DO 1110 J1 = 1,4
1110 HEXCOD(J1,WRDNUM) = HEX(J1)
RETURN
C
C **** FORMAT C W -- OPERAND FIELD IS RESTRICTED TO SIMPLE REGISTERS
C
1200 IF(PASS EQ. FIRST) RETURN
II = 0
CALL OPFLD(I1)
IF(MAJVAL NE REGIST) OPERR = .TRUE.
TEMP1 = LVRTSH(OPER1,B)
TEMP2 = TEMP1
HEXCOD(1,1) = TEMP2
TEMP1 = (OPER1 AND MASKB)
TEMP2 = TEMP1
HEXCOD(2,1) = TEMP2
TEMP1 = LVRTSH(OPER2,B)
IF(TEMP1 EQ. 0) TEMP1 = ASCII
TEMP2 = TEMP1
HEXCOD(3,1) = TEMP2
HEXCOD(4,1) = HEX(4)
RETURN
C
C **** DIRECTIVES ****
C
D2000 TYPE 2001:DIRNUM
D2001 FORMAT(’DIRNUM = ’,I5)
2000 GO TO(2100,2200,2300,2400,2500,2600,2700,2800,2900,3000,3100,
1 3200,3300,3400,3500,3600,3700,3800,3900,4000) DIRNUM
C
C **** AORQ
C
2100 COMNTS = TRUE.
GO TO 4000
RETURN
C
C **** BES
C
2200 TYPE 2201
2201 FORMAT(’’BES’,.’,0).
C
2205 TYPE 2205
2205 FORMAT(’’*’,’ IS NOT YET IMPLEMENTED’)
COMNTS = TRUE
RETURN

49
C **** BSS -- BLOCK STARTING WITH SYMBOL
C
2300 I1 = 0
CALL OPFLD(I1)
PC = PC + ADDRES
RETURN
C
C **** BYTE -- INITIALIZE BYTE DATA, MUST END WITH SEMI-COLON
C
2400 NUMEXP = 1
WRDPOS = EVEN
LIMIT = 20
C IF((PC AND ODD) EQ. 0) GO TO 2405
C LIMIT = 19
C WRDPOS = ODD
2405 PC = PC + 1
C
C COUNT THE COMMAS
I1 = 0
02410 ITEM + ITMTYP."I1 = I1 + 1"/2430 =KOMMA/2410/2420
2420 PC = PC + 1
NUMEXP = NUMEXP + 1
GO TO 2410
C
C IF THIS DIRECTIVE HAD A LABEL, DECLARE IT AS BYTE
02430 STRING + LABELS/2450
C ER + STOP.2 'IDSTAT
IDSTAT = IDSTAT OR. BYTFLG
C ER STOP - .2 "IDSTAT"
C
C BEGIN EVALUATION OF THE EXPRESSIONS. 20 LIMIT IF EVEN. 19 IF ODD
C2450 LIMIT = 20
C IF(WRDPOS EQ. ODD) LIMIT = 19
2450 BYTNUM = NUMEXP
WRDNUM = 1
I1 = 0
IF(NUMEXP .01 LIMIT) OPERR = TRUE.
GO 2470 J1 = 1,NUMEXP
CALL OPFLD(I1)
IF(WRDPOS EQ. ODD) GO TO 2460
C PC IS EVEN
HEXCOD(1,WRDNUM) = HEX(3)
HEXCOD(2,WRDNUM) = HEX(4)
WRDPOS = ODD
GO TO 2470
C PC IS ODD
2460 HEXCOD(3,WRDNUM) = HEX(3)
HEXCOD(4,WRDNUM) = HEX(4)
WRDPOS = EVEN
WRDNUM = WRDNUM + 1
2470 CONTINUE
IF(WRDPOS EQ. EVEN) WRDNUM = WRDNUM -1
RETURN
C
C **** DATA -- INITIALIZE WORD DATA
C
2500 NUMEXP = 1
PC = PC + 2
C
C COUNT THE COMMAS
I1 = 0
02510 ITEM + ITMTYP."I1 = I1 + 1"/2530 =KOMMA/2510/2520
2520 PC = PC + 2
NUMEXP = NUMEXP + 1
GO TO 2510

50
C
C EVALUATE UP TO 10 EXPRESSIONS
2530 LIMIT = 10
   IF(NUMEXP GT LIMIT) OPERR = TRUE
   WRDNUM = 2*NUMEXP
   I1 = 0
   DO 2570 J1 = 1, NUMEXP
      WRDNUM = WRDNUM + 1
      CALL OPFLD(I1)
      DO 2560 L1 = 1, 4
         HEXCOD(L1, WRDNUM) = HEX(I1)
      2560 CONTINUE
   2570 CONTINUE
   RETURN
C
C **** DEF
C
2600 TYPE 2601
2601 FORMAT(’DEF’,$)
   TYPE 2205
   COMNTS = TRUE
   RETURN
C
C **** DORG
C
2700 TYPE 2701
2701 FORMAT(’DORG’,$)
   TYPE 2205
   COMNTS = TRUE
   RETURN
C
C **** DXOP -- DEFINE EXTENDED OPERATION
C
2800 COMNTS = TRUE
   IF(PASS EQ SECOND) RETURN
C
C REMOVE LABEL FROM IDENTIFIER TREE
G
GRLBL = STOP
C
C ADD LABEL TO ‘INSTRUCTION - DIRECTIVE’ TREE
S1 = START
I1 = 0
G20 STRING + LABELS ’I1 = I1 + 1’/30 ’GRLCHR
G S1 + GRLCHR 1/25 ’S1/20
G25 S1 GRLCHR ’$1
   GO TO 20
C
C USE FORMAT ’6’
G30 S1 OPFIN ’6’
C
C DETERMINE EXTENDED OPERATION NUMBER AND INCORPORATE INTO
C XOP INSTRUCTION SKELETON
I1 = 0
   CALL OPFLD(I1)
   IF(MAJVAL NE NUM) OPERR = TRUE
C
C SPLIT XOP NUMBER INTO TWO RIGHT AND TWO LEFT BITS
RHALF = ADDRES AND 3
LFHALF = LVRTSH(ADDRES:2)
C
C CREATE INSTRUCTION SKELETON
ISTBYT = HEXTBL(2 + 1)
SECBYT = HEXTBL(12 + LFHALF + 1)
ISTWAD = LVLFSH(ISTBYT:8) OR SECBYT
BYTONE = LVLFSH(RHALF:2)
BYTONE = HEXTBL(BYTONE + 1)
SECWRD = LVALFSH(BYTEONE,8)
G
SI OPFIN (_ISTWRD, SECWRD)
RETURN
C
**** END -- END OF ASSEMBLY
C
2900 ENDFLG = .TRUE
RETURN
C
**** EQU — DEFINE ASSEMBLY TIME CONSTANT
C
3000 I1 = 0
CALL OPFLD(I1)
G
ERLABL STOP = 1 "ADDRES"
IF(PASS EQ .SECOND) COMNTS = TRUE
RETURN
C
**** EVEN -- IF PROGRAM COUNTER IS AN ODD NUMBER, ADD ONE TO EVEN UP
C
3100 IF((PC AND . ODD) EQ 0) GO TO 3150
PC = PC + 1
HEXCOD(1, WRDNUM) = ASCII0
HEXCOD(2, WRDNUM) = ASCII0
RETURN
3150 COMNTS = .TRUE
RETURN
C
**** IDT
C
3200 TYPE 3201
3201 FORMAT(’ IDT’,$)
TYPE 2205
COMNTS = .TRUE.
RETURN
C
**** LIST -- OUTPUT SOURCE LISTING
C
3300 LSTFLG = .TRUE
COMNTS = .TRUE
RETURN
C
**** PAGE
C
3400 TYPE 3401
3401 FORMAT(’ PAGE’,$)
TYPE 2205
COMNTS = .TRUE.
RETURN
C
**** REF
C
3500 TYPE 3501
3501 FORMAT(’ REF’,$)
TYPE 2205
COMNTS = .TRUE.
RETURN
C
**** RORG
C
3600 TYPE 3601
3601 FORMAT(’ RORG’,$)
TYPE 2205
COMNTS = .TRUE.
RETURN
C
**** TEXT -- INITIALIZE UP TO 20 CHARACTERS IF PC IS EVEN, 19 IF ODD
C 3700 LIMIT = 20
   WRDPOS = EVEN
   IF((PC AND ODD) EQ 0) GO TO 3710
   LIMIT = 19
   WRDPOS = ODD
C
C OBTAIN LOCATION AND LENGTH OF TEXT
G3710 ITEM + (ITMTYP 'ITMVAL, BUFPOS 'ISTLOC, LENID 'LENGTH)
C
C ERROR IF NOT LITERAL OR EXCEEDS 19 OR 20 CHARACTERS
   IF(ITMVAL .NE. LITERAL) OPERR = TRUE
   IF(LENGTH .GT. LIMIT) OPERR = TRUE
   BYTNUM = LENGTH
   PC = PC + LENGTH
   IF(PASS .EQ. FIRST) RETURN
   WRDNUM = 1
C
C PICK UP NEXT CHARACTER
   NXTLOC = ISTLOC
   DO 3770 J1 = 1, LENGTH
   IF(WRDPOS .EQ. ODD) GO TO 3760
   TEMP2 = OPRBUF(NXTLOC)/16
   HEXCOD(1,WRDNUM) = HEXTBL(TEMP2 + 1)
   TEMP2 = OPRBUF(NXTLOC) AND. MASK4
   HEXCOD(2,WRDNUM) = HEXTBL(TEMP2 + 1)
   NXTLOC = NXTLOC + 1
   WRDPOS = ODD
   GO TO 3770
C 3760 TEMP2 = OPRBUF(NXTLOC)/16
   HEXCOD(3,WRDNUM) = HEXTBL(TEMP2 + 1)
   TEMP2 = OPRBUF(NXTLOC) AND. MASK4
   HEXCOD(4,WRDNUM) = HEXTBL(TEMP2 + 1)
   NXTLOC = NXTLOC + 1
   WRDPOS = EVEN
   WRDNUM = WRDNUM + 1
3770 CONTINUE
RETURN
C
C **** TITL
C
C 3800 TYPE 3801
3801 FORMAT(' TITL',*)
   TYPE 2205
   COMNTS = .TRUE.
   RETURN
C
C **** UNL
C
C 3900 LSTFLG = FALSE
   COMNTS = .TRUE.
   RETURN
C
C **** SLASH -- CHANGE PROGRAM COUNTER
C
C 4000 II = 0
   CALL OPFLD(II)
   PC = ADDRES
   RETURN
C
COMPLETE

53
SUBROUTINE LINOUT

LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HECOD, LINE
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
LOGICAL*1 DEFGLG, REGFLG, MLTFLG, BYTFLG, LAFFLG, INSFLG, OPFLG, RESFLG.
1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG

COMMON /ASC/ LINE(72), HEX(4), HECOD(4,10)
COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/ DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
COMMON /LINFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA, 1 LSTFLG
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1 MAJVAL, COMENT, IDSTAT, LINSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTYNUM, ERROR
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, .S, LENID, LABELS, OPCODE, STOP,
1 ITM_TYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGFLPS,
2 VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OFFIN, DIRECT, START, TOWTIME
COMMON /TMPARR/ TEMPI(30), TEMP2(30), TEMP3(30), TEMP4(30)

DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)

EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))

*** OUTPUT INTERMEDIATE CODE (ITEM STRING) FOR A SINGLE LINE
WRITE(11) REGSRC, REGDES, REGCOD, LINENO,
1 COMENT, LINSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTYNUM, ERROR
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, .S, LENID, LABELS, OPCODE, STOP,
1 ITM_TYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGFLPS,
2 VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OFFIN, DIRECT, START, TOWTIME
COMMON /TMPARR/ TEMPI(30), TEMP2(30), TEMP3(30), TEMP4(30)

C *** OUTPUT INTERMEDIATE CODE FOR A SINGLE LINE
WRITE(11) REGSRC, REGDES, REGCOD, LINENO,
1 COMENT, LINSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTYNUM, ERROR
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, .S, LENID, LABELS, OPCODE, STOP,
1 ITM_TYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGFLPS,
2 VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OFFIN, DIRECT, START, TOWTIME
COMMON /TMPARR/ TEMPI(30), TEMP2(30), TEMP3(30), TEMP4(30)

C EXTRACT INTERMEDIATE CODE FROM THE ITEM STRING UNLESS COMMENT
IF (COMNTS .EQ. TRUE) RETURN
LIMIT = 30
II = 0
GO TO 10
G10 ITEM + (ITM_TYP "II=II+1" /20 'ITMVAL', VALUE 'II 'VAL)
G ITEM + (BUFPOS II 'ISTLOC, LENID II 'LENGTH)
TEMP1(II) = ITMVAL
TEMP2(II) = VAL
TEMP3(II) = ISTLOC
TEMP4(II) = LENGTH
GO TO 10
20 II = II - 1
IF (II GT LIMIT) GO TO 99
WRITE(11) TEMP1, TEMP2, TEMP3, TEMP4
RETURN

C C ERROR
99 TYPE 100
100 FORMAT(' *** ERROR - TOO MANY ITEMS IN OPERAND FIELD')
STOP
COMPLETE
SUBROUTINE LINEIN

LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1, ASCII1F, ASCII1Z, ASCIIAR
LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, BUF(40), LINE
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
LOGICAL*1 DEFFLG, REFLG, MTLFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG.
   1 DIRFLG, ENDFLG, COMENTS, EXTRA, LASTFLG, DONE

COMMON /ASC/
   LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ERRFLG/
   LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/
   DEFFLG, LABFLG, REGFLO, MLTFLG, BYTFLG, DUMB
COMMON /LINFLG/
   INSFLG, OPRBUF(1), RESFLG, DIRFLG, ENDFLG, COMENTS, EXTRA.
COMMON /ASSEMB/
   PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
   1 MAJVAL, COMENT, IDSTAT, LINSTAT, ERRNUM, FMT, DIRNUM.
   2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /GIRLCH/
   EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
   1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
   2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT.
   3 NINE, COLON, SCOLON, LSTHAN, EQUALS, GTHAN, QUEST,
   5 VWLXYZ
COMMON /GIRL/
   STRING, TOKEN, ITEM, BUFPOS. LENID. LABELS. OPCOD, STOP,
   1 ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS.
   2 VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO.
   3 OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCII1, ASCII1F, ASCII1Z
COMMON /TMPARR/ TEMP1(30).TEMP2(30), TEMP3(30), TEMP4(30)
DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)
DIMENSION ASCGRL(58)

EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))
EQUIVALENCE (ASCGRL(1), EXCLAM), (HEXCOD(1, 1), BUF(1))

C *** INPUT INTERMEDIATE CODE (ITEM STRING) FOR A SINGLE LINE
READ(11) REGSRC, REGDES, REGCOD, LINENO.
   1 COMENT, LINSTAT, ERRNUM, FMT, DIRNUM.
   2 OPER1, OPER2, ER, ERLABL, LINE.
   3 INSTER, COMENTS, LABERR, OPERR

C IS THE ENTIRE LINE A COMMENT STATEMENT ?
   IF(COMENTS .EQ .TRUE.) RETURN

C RECREATE LABEL AND ITEM STRINGS FROM THE INTERMEDIATE CODE
   DONE = .FALSE
   READ(11) TEMP1, TEMP2, TEMP3, TEMP4
   DO 10 I1 = 1, 30
      IF(DONE) GO TO 20
      ITEM ITMTYP "TEMP(I1)"
      ITEM VALUE "TEMP(I1)"
      ITEM BUFPOS "TEMP(I1)"
      ITEM LENID "TEMP(I1)"
   C LOOK FOR SEMICOLON
      IF(TEMP(I1) EQ 14) DONE = .TRUE
      CONTINUE
   10 CONTINUE
C C CHECK FOR LABEL AND CREATE LABEL STRING
   DO 20 I1 = 1, 6
      ASCHAR = LABEL(I1)
      IF(ASCHAR .EQ BLANK) RETURN
      GRLCHR = ASCGRL(ASCHAR-BLANK)
   G STRING LABELS GRLCHR
SUBROUTINE OPFLD(IU)
LOGICAL*1 LABEL,PCODE, OPBUF, HEX, HEXCOD, HEXTBL, LINE
LOGICAL*1 BLANK, ASC110, ASC119, ASC11A, ASC11F, ASC11Z, ASC1CHAR
LOGICAL*1 Laberr, DIRERR, OPERR, TYPErr, INSTER, DUMMY, DUMB
LOGICAL*1 DEFFLG, REFFLG, MLTFLG, BYTFLG, LABFLOG, INSFLG, OPFLG, RESFLG,
1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG

COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ERRFLG/ LBERR, DIRERR, OPERR, TYPErr, INSTER, DUMMY
COMMON /NAMFLG/ DEFFLG, LABFLOG, REFFLG, MLTFLG, BYTFLG, DUMB
COMMON /LINFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1 LSTFLG
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDR,
  1 MAJVAL, COMMENT, IDSTAT, LNSSTAT, ERRNUM, FMT, DIRNUM,
  2 OPER1, OPER2, ER, ELABD, WRDNUM, BYTNUM, ERROR
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRLC/
  EXCLAM, QUOTE, POUND, DOLLAR, PERCENT, AMPERS, APOST,
  1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
  2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
  3 NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
  5 V, W, X, Y, Z
COMMON /GIRL/
  STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, POCOD, STOP,
  1 ITMYP, VALUE, SYMBOL, REG, REGST, REGATS, REGPOL,
  2 VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO,
  3 OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASC110, ASC119, ASC11A, ASC11F, ASC11Z

DIMENSION LABEL(6), POCOD(4), OPBUF(60)

DIMENSION OPANDL(2)

THE PURPOSE OF THIS ROUTINE IS TO EITHER RETURN THE DECIMAL VALUE OF AN
C OPERAND FIELD (FROM COMMA TO COMMA) OR REGISTER NUMBER IN "ADDR" AND
C THE HEX VALUE IN "HEX(1)" AND TO DETERMINE IF AN EXTRA WORD IS NEEDED
C FOR THE TARGET CODE. IF SO, "EXTRA" = TRUE

MAJVAL = 0
EXTRA = FALSE
OPAND(FIRST) = 0
OPAND(SECOND) = 0
7
OPAND = FIRST

C PICK UP ALL ITEMS UNTIL COMMA OR SEMI-OLON
Q10 ITEM + ITMYP "11 = 11 + 1"/30 'ITMVAL
Q ITEM + VALUE II "ADDRES
C
C IS THIS ITEM A REGISTER?
IF(ITMVAL GT 3) GO TO 15
C
C CHECK FOR &VAR(REG) OR &NUM(REG)
IF(MAJVAL EQ 17) GO TO 10
IF(MAJVAL EQ 16) GO TO 10
MAJVAL = ITMVAL
OPAND(OPRAN) = ADDRES
GO TO 10
\[\text{I} \]
\[\text{C} \]
\[\text{15 ITM = ITMVAL - 3} \]
\[\text{IF (ITM GT 8) GO TO 20} \]
\[\text{C} \]
\[\text{ITEM IS AN OPERATOR} \]
\[\text{OPRATR = ITM} \]
\[\text{C} \]
\[\text{SWITCH TO SECOND OPERAND} \quad \_ \text{OP} \_ \] 
\[\text{OPRAN = SECOND} \]
\[\text{GO TO 10} \]
\[\text{C} \]
\[\text{ITEM IS COMMA, EXCLAM MARK, SEMI-COLON, LITERAL, NUMBER, IDENTIFIER, OR @} \]
\[\text{20 ITM = ITM - 8} \]
\[\text{IF ((ITM LT 1) OR (ITM GT 7)) GO TO 100} \]
\[\text{C} \]
\[\text{*** COMMA OR SEMI-COLON} \]
\[\text{C} \]
\[\text{FIELD HAS BEEN COMPLETELY EXAMINED. NO EXTRA WORDS ARE NEEDED IF FIELD} \]
\[\text{C} \]
\[\text{CONTAINED AN "UNMODIFIED" REGISTER} \]
\[\text{30 ADDRESS = OPRAND(FIRST)} \]
\[\text{C} \]
\[\text{CONVERT OPERAND FROM DECIMAL TO HEXADECIMAL} \]
\[\text{CALL DECHEX(ADDRESS. HEX(1))} \]
\[\text{RETURN} \]
\[\text{C} \]
\[\text{*** EXCLAMATION MARK} \]
\[\text{C} \]
\[\text{40 OPRAND(OPRAN) = PC} \]
\[\text{GO TO 55} \]
\[\text{C} \]
\[\text{*** LITERAL OR NUMBER} \]
\[\text{C} \]
\[\text{50 OPRAND(OPRAN) = ADDRESS} \]
\[\text{55 EXTRA = TRUE} \]
\[\text{MAJVAL = ITMVAL} \]
\[\text{IF (OPRAN EQ. FIRST) GO TO 10} \]
\[\text{C} \]
\[\text{AN OPERAND - OPERATOR - OPERAND TRIPLE HAS BEEN FOUND.} \]
\[\text{C} \]
\[\text{COMPUTE IT AND RETURN RESULT INTO OPERAND NO.} \]
\[\text{1 CALL COMPUT(OPRAND(1), OPRATR)} \]
\[\text{GO TO 7} \]
\[\text{C} \]
\[\text{*** IDENTIFIER (EXCLUDING REGISTER NAMES)} \]
\[\text{C} \]
\[\text{C WAS THE IDENTIFIER DEFINED WHEN THIS STATEMENT OCCURRED?} \]
\[\text{60 IF (ADDRESS GT 0) GO TO 50} \]
\[\text{C} \]
\[\text{OBTAIN ADDRESS} \]
\[\text{G ITEM + BUFPDS II 'ISTRIC} \]
\[\text{G ITEM + LENID II 'LENGTH} \]
\[\text{CALL Tmprstr('ISTRIC. LENGTH)} \]
\[\text{CALL ADDNAM(STRING. STRING)} \]
\[\text{IF (PASS EQ. FIRST) GO TO 50} \]
\[\text{C} \]
\[\text{STILL UNDEFINED?} \]
\[\text{IF (ADDRESS EQ. 0) OPERR = TRUE} \]
\[\text{GO TO 50} \]
\[\text{C} \]
\[\text{*** AT-SIGN @ -- NEXT ITEM IS A NUMBER OR VARIABLE} \]
\[\text{C} \]
\[\text{ANTICIPATE SHORTHAND "@REG" FOR "@0(REG)"} \]
\[\text{70 EXTRA = TRUE} \]
\[\text{ADDRESS = 0} \]
MAJVAL = 16
GO TO 10
100 OPERN = TRUE
COMPLETE

SUBROUTINE TMPSTR(ISTLOC, LENGTH)

LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, BUF(40), LINE
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII, ASCII1, ASCIIIZ, ASCHAR

COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES.
1 MAJVAL, COMENT, IDSTAT, LNSSTAT, ERRNUM, FMT, DIRNUM.
2 JEB1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST.
1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH.
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT.
3 NINE, COLON, SCOLON, LESS, EQUAL, GTHAN, QUEST.

COMMON /GIRL/ STRING, ITEM, BUFSIZE, LABELS, OPNUM, STOP,
1 ITMTYP, VALUE, REG, REGSTR, REGATS, REGPLS.
2 VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO.
3 OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCII, ASCII1, ASCIIIZ

DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)
DIMENSION ASCGRL(58)

EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))
EQUIVALENCE (ASCGRL(1), EXCLAM), (HEXCOD(1, 1), BUF(1))

C STRING - STRING
LOC = ISTLOC
DO 10 K1 = 1, LENGTH
ASCHAR = OPRBUF(LK)
GRLCHR = ASCGRL(ASCHAR - BLANK)
10 STRING STRING GRLCHR
LOC = LOC + 1
CONTINUE
RETURN

C SUBROUTINE VARS

LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, LINE
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII, ASCII1, ASCIIIZ, ASCHAR
LOGICAL*1 DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB, EKS
LOGICAL*1 IDNFR(6), STATUS(6)

COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES.
1 MAJVAL, COMENT, IDSTAT, LNSSTAT, ERRNUM, FMT, DIRNUM.
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST.
1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH.
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT.
3 NINE, COLON, SCOLON, LESS, EQUAL, GTHAN, QUEST.
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP, I MTY I, VALUE, SYMBOL, REQ, REGSTR, REGATS, REGPLS, VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO, OFFIN, DIRECT, START, TOKODE

COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCII1, ASCIIF, ASCIIZ
COMMON /NAMFLG/ DEFFLG, LABFLO, REGFLG, MLTFLG, DYTFLG, DUMB

DIMENSION NAME(6), ALPHA(26), NUMBRS(10)

EQUIVALENCE (ALPHA(1), A), (NUMBRS(1), ZERO)

DATA EKS /1HX/

THIS ROUTINE OUTPUTS THE STATUS FOR EACH IDENTIFIER AND REGISTER

WRITE(13, 1)
1 FORMAT(///, ' **** IDENTIFIER LISTING ****', ///, ' NAME STATUS', ///, 2 12X, 'ADDRESS REG UNDEF MULT DEF BYTE', ///)

READ(14, END = 200) NAME
SY = SYMBOL

DO 10 J1 = 1, 6
IDNTFR(J1) = BLANK
10 STATUS(J1) = BLANK

DO 100 I1 = 1, 6
GRLCHR = NAME(I1)
IF(GRLCHR .EQ. 0) GO TO 120

C OBTAIN ASCII CHAR FROM GIRL VALUE
DO 20 K1 = 1, 26
L1 = K1
20 IF(GRLCHR .EQ. ALPHA(L1)) GO TO 50

DO 30 K1 = 1, 10
L1 = K1
30 IF(GRLCHR .EQ. NUMBRS(L1)) GO TO 40

40 IDNTFR(I1) = ASCII0 + L1 - 1
GO TO 60

50 IDNTFR(I1) = ASCII1 + L1 - 1
60 SY + GRLCHR 'SY'
100 CONTINUE

C G120 SY + STOP(5 'ADDRES. 2 'IDSTAT)
REGTST = IDSTAT .AND. REGFLG
DEFTST = IDSTAT .AND. DEFFLG
MTTST = IDSTAT .AND. MLTFLG
BYTTST = IDSTAT .AND. BYTFLG
IF(REGTST .GT. 0) STATUS(1) = EKS
IF(DEFTST .EQ. 0) STATUS(2) = EKS
IF(MTTST .GT. 0) STATUS(3) = EKS
IF(BYTTST .GT. 0) STATUS(4) = EKS

CALL DECEHX(ADDRES, HEX)
WRITE(13, 150) IDNTFR, ADDRES, HEX, STATUS
150 FORMAT(1X, 6A1, 1X, 16, 1X, 4A1, 4X, A1, 5X, A1, 8X, A1, 6X, 3A1)
GO TO 5
200 RETURN
Q COMPLETE
/ COMPLETE
SUBROUTINE COMPUT(OPRAND, OPRATR)
    IMPLICIT INTEGER (A-Z)
    DIMENSION OPRAND(2)
    C
    THIS ROUTINE PERFORMS A COMPUTATION ON OPRAND(1) OPRATR OPRAND(2)
    AS DETERMINED BY THE ARITHMETIC OR LOGICAL OPERATOR IN "OPRATR" AND
    PLACES THE RESULT INTO OPRAND(1).
    C
    .+ .* .- / + * - /
    GO TO ( 10, 20, 30, 40, 50, 60, 70, 80) OPRATR
    C
    C LOGICAL OR
    C
    10 OPRAND(1) = OPRAND(1) .OR. OPRAND(2)
    RETURN
    C
    C LOGICAL AND
    C
    20 OPRAND(1) = OPRAND(1) .AND. OPRAND(2)
    RETURN
    C
    C LOGICAL NOT -- ONE'S COMPLEMENT
    C
    30 OPRAND(1) = .NOT OPRAND(2)
    RETURN
    C
    C MODULO
    C
    40 OPRAND(1) = MOD(OPRAND(1), OPRAND(2))
    RETURN
    C
    C ADD
    C
    50 OPRAND(1) = OPRAND(1) + OPRAND(2)
    RETURN
    C
    C MULTIPLY
    C
    60 OPRAND(1) = OPRAND(1) * OPRAND(2)
    RETURN
    C
    C SUBTRACT
    C
    70 OPRAND(1) = OPRAND(1) - OPRAND(2)
    RETURN
    C
    C DIVIDE
    C
    80 OPRAND(1) = OPRAND(1) / OPRAND(2)
    RETURN
    END
SUBROUTINE DECHEX(DECNUM,HEX)
IMPLICIT INTEGER (A-Z)
C
LOGICAL*1 HEX(4),HEXBL,NEO
LOGICAL*1 BLANK,ASCII10,ASCII19,ASCII2A,ASCII2F,ASCII30,ASCII31
C
COMMON /DATUM/ FIRST,SECOND,HEX(16)
COMMON /ASCII/ BLANK,ASCII10,ASCII19,ASCII2A,ASCII2F,ASCII30,ASCII31
C
DIMENSION TEMP(4)
C
THIS ROUTINE CONVERTS THE DECIMAL VALUE IN "DECNUM" TO HEXADECIMAL
C AND PLACES IT INTO HEX() C
C
NEG = FALSE
IF(DECNUM EQ 0) NEG = TRUE
NUMBER = IABS(DECNUM)
      DO 5 L1 = 1,4
          TEMP(L1) = 0
      5       HEX(L1) = ASCII10
          HEXPOS = 5
      10  HEXPOS = HEXPOS - 1
C
COMPUTE REMAINDER FROM MODULUS 16
REM = MOD(NUMBER,16)
C
TEMP(HEXPOS) = REM
HEX(HEXPOS) = HEXTBL(REM + 1)
IF(NUMBER LT 16) GO TO 20
NUMBER = NUMBER / 16
GO TO 10
C
IF NEGATIVE, CONVERT TO TWO'S COMPLEMENT FORM
20  IF( NOT NEG) RETURN
   CALL TWOCMP(HEX(1))
   RETURN
END
C
C
SUBROUTINE FMT1(ADRCOD)
IMPLICIT INTEGER (A-Z)
C
LOGICAL*1 HEXTBL, ADRCOD(3)
C
COMMON /DATUM/ FIRST,SECOND,HEX(16)
COMMON /ASCEN/ PASS,REGSRC,REGDES,RECCOD(2),PC,LINENO,ADDRES,
                   MAJVAL,COMENT,IDSTAT,LNSTAT,ERRNUM,FMT,DIRNUM,
                   OPER1,OPER2,ER,ERLABL,WRSNUM,BYTNUM,ERROR
C
ADRCOD(1) = (RECCOD(2) * 4) + (REGDES/4)
C
ADRCOD(2) = RECCOD(1) + (4 * MOD(REGDES,4))
C
ADRCOD(3) = REGSRC
   RETURN
END
C
C
FUNCTION LVLFSH(WORD,BITS)
IMPLICIT INTEGER(A-Z)
C
THIS FUNCTION PERFORMS A LEFT LOGICAL SHIFT
C
IF(BITS EQ 0) GO TO 10
   IF(BITS GE 16) GO TO 20
LVLFSH = WORD * 2 ** (BITS)
RETURN
10 LVLFSH = WORD
RETURN
20 LVLFSH = 0
RETURN
END

FUNCTION LVRTSH(WORD, BITS)
IMPLICIT INTEGER(A-Z)
C THIS FUNCTION PERFORMS A RIGHT LOGICAL SHIFT
C IF(BITS EQ 0) GO TO 10
IF(BITS GE 16) GO TO 20
LVRTSH = WORD / 2 ** (BITS)
RETURN
10 LVRTSH = WORD
RETURN
20 LVRTSH = 0
RETURN
END

SUBROUTINE ABSOUT
IMPLICIT INTEGER(A-Z)
LOGICAL*1 LABEL, OPCODE, OPRIUFL, HEX, HEXCOD, HEXTBL, LINE
LOGICAL*1 BLANK, ASCII0, ASCII1, ASCII2, ASCII3, ASCII4, ASCII5, ASCII6, ASCII7, ASCII8, ASCII9, ASCII10, ASCII11, ASCII12, ASCII13, ASCII14, ASCII15
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
LOGICAL*1 DEFILG, REGILG, BYFILG, LABIFG, INGLG, OPFLG, REGFLG,
1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
LOGICAL*1 HEXBYT(2, 20), PGMCTR, DUMMY2, LIN

COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMEFLG/ DEFILG, LABILG, REGILG, BYFILG
COMMON /LINFLG/ INGLG, OPFLG, REGFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1 LSTFLG
COMMON /ASSEMB/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APPOST,
1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3 NINE, COLON, SCOLON, LTHAN, EQUALS, GTTHAN, QUEST,
5 V, W, X, Y, Z
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
1 IFTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2 VOPF, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII0, ASCII1, ASCII2, ASCII3, ASCII4, ASCII5, ASCII6, ASCII7, ASCII8, ASCII9, ASCII10, ASCII11, ASCII12, ASCII13
COMMON /LOCAL/ NUM1, NUM2, NUM3, LINCT1, LINCT2, LINCT3, PGMCTR(5),
1 DUMMY2, OLDP, OLDBVT, LIN(48)

DIMENSION LABEL(6), OPCOD(4), OPRBUF(60)

EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(2), OPCOD(1))
EQUIVALENCE (LINE(3), OPRBUF(1))
EQUIVALENCE (HEXCOD(1, 1), HEXBYT(1, 1))
C
DATA PGMCTR /IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO, IHO/  
DATA NUMI, NUM2, NUM3, OLDPC, OLDBYT /0, 1, 0, 256, 0/  
DATA LINCT1, LINCT2, LINCT3 /1, 2, 3/  
C TEST FOR MODIFICATION OF PC  
IF((DIRNUM EQ 3) OR (DIRNUM EQ 20)) GO TO 500  
C  
DO 180 II = 1, BYTNUM  
LIN(LINCT1) = BLANK  
LIN(LINCT2) = HEXBYT(1, II)  
LIN(LINCT3) = HEXBYT(2, II)  
C HAS A LINE BEEN FILLED?  
IF(LINCT3 GE (16*3)) GO TO 120  
C IF NOT FILLED, IS THIS THE LAST LINE OF THE PROGRAM?  
IF(ENDFLG) GO TO 200  
LINCT1 = LINCT1 + 3  
LINCT2 = LINCT2 + 3  
LINCT3 = LINCT3 + 3  
GO TO 180  
C OUTPUT THIS LINE  
120 WRITE(12, 101) PGMCTR, LIN  
101 FORMAT(53A1)  
C UPDATE COUNTERS AND THE PROGRAM COUNTER ARRAY  
LINCT1 = 1  
LINCT2 = 2  
LINCT3 = 3  
C NUM3 = NUM3 + 1  
C CARRY?  
IF(NUM3 LE 15) GO TO 170  
NUM3 = 0  
NUM2 = NUM2 + 1  
C CARRY?  
IF(NUM2 LE 15) GO TO 160  
NUM2 = 0  
NUM1 = NUM1 + 1  
IF(NUM1 GE 16) STOP '**** ERROR, PROGRAM EXCEEDS FFFF'  
IF((NUM1 GE 8) AND (NUM2 GE 8))  
PAUSE '**** WARNING, PROGRAM EXCEEDS 07FF, (CR) TO CONTINUE'  
PGMCTR(1) = HEXTBL(NUM1 + 1)  
PGMCTR(2) = HEXTBL(NUM2 + 1)  
160 PGMCTR(3) = HEXTBL(NUM3 + 1)  
170 CONTINUE  
180 CONTINUE  
GO TO 600  
C BLANK OUT END OF LAST LINE  
200 NOMORE = LINCT3 + 1  
DO 250 JJ = NOMORE, 48  
250 LIN(JJ) = BLANK  
WRITE(12, 101) PGMCTR, LIN  
GO TO 600  
C PC MODIFICATION, ZERO FILL TO PC (FROM "BSS" AND "/")  
C COMPUTE NEW LINE NUMBER (HEX(1)-HEX(3)) AND COUNTERS  
500 PCTEMP = PC  
CALL DECHEX(PCTEMP, HEX)  
NUM1 = HEX(1) - ASCII10  
NUM2 = HEX(2) - ASCII10  
NUM3 = HEX(3) - ASCII10  
63
IF(NUM1 GT 9) NUM1 = HEX(1) - ASCII + 10
IF(NUM2 GT 9) NUM2 = HEX(2) - ASCII + 10
IF(NUM3 GT 9) NUM3 = HEX(3) - ASCII + 10

C CALCULATE DIFFERENCE TO NEW LOCATION
DIFRNC = PC - OLDPC

C SIX POSSIBILITIES
A) NO MODIFICATION
B) READY TO START NEW LINE
C) ZERO FILL TO THE MIDDLE OF THE OUTPUTTED LINE
D) ZERO FILL TO THE END OF THE OUTPUTTED LINE
E) ZERO FILL TO THE END OF THE OUTPUTTED LINE AND PART OF ANOTHER LINE
F) ERROR, PC REQUEST IS LESS THAN CURRENT PC

IF(DIFRNC LT 0) GO TO 700
IF(DIFRNC EQ 0) GO TO 600
IF(LINCTI EQ 1) GO TO 570

C ZERO FILL OLD LINE
530 DO 550 II=1,DIFRNC
LIN(LINCT1) = BLANK
LIN(LINCT2) = ASCII0
LIN(LINCT3) = ASCII0
IF(LINCT3 GE 48) GO TO 560
LINCT1 = LINCT1 + 3
LINCT2 = LINCT2 + 3
LINCT3 = LINCT3 + 3
550 CONTINUE
GO TO 600

C D TYPE 999, NUM1, NUM2, NUM3, OLDPC, OLDBYT, ENDLIN, DIFRNC, PC
D999 FORMAT(IX,8(16,2X))

C OUTPUT OLD LINE AND UPDATE COUNTER
560 WRITE(12,101) PGMCTRLIN
570 DO 580 I = 1,3
580 PGMCTR(I) = HEX(I)

C ZERO FILL THE BEGINNING OF THE NEW LINE
LINCT1 = 1
LINCT2 = 2
LINCT3 = 3
DIFRNC = HEX(4) - ASCII0
IF(DIFRNC GT 9) DIFRNC = HEX(4) - ASCII + 10
IF(DIFRNC EQ 0) GO TO 600
DO 590 II=1,DIFRNC
IF(LINCT3 GT 48) GO TO 600
LIN(LINCT1) = BLANK
LIN(LINCT2) = ASCII0
LIN(LINCT3) = ASCII0
LINCT1 = LINCT1 + 3
LINCT2 = LINCT2 + 3
LINCT3 = LINCT3 + 3
590 CONTINUE
600 OLDPC = PC
OLDBYT = BYTNUM
RETURN

700 TYPE 701, PC, OLDPCC
701 FORMAT(IX, '*** ERROR ***, PC REQUEST OF', 15, ' IS LESS THAN THE CURRENT PROGRAM COUNTER', 15)
STOP
END
SUBROUTINE SRCLST(THISPC)

IMPLICIT INTEGER (A-Z)

LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, HEXTBL, LINE
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCIIA, ASCIIF, ASCIIZ, ASCHAR
LOGICAL*1 DEFFLG, REGFLG, OPERR, TYPERR, INSTER, DUMMY
LOGICAL*1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
LOGICAL*1 PGMCTR, DUMMY2, HXCODE(4), LINE

COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ERRFLG/ DEFFLG, REGFLG, OPERR, TYPERR, INSTER, DUMMY
COMMON /LINFLG/ DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
COMMON /ASSEMB/ REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCIIA, ASCIIF, ASCIIZ
COMMON /LOCAL/ NUM1, NUM2, NUM3, LINCT1, LINCT2, LINCT3, PGMCTR(5),
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCIIA, ASCIIF, ASCIIZ
COMMON /LOCAL/ NUM1, NUM2, NUM3, LINCT1, LINCT2, LINCT3, PGMCTR(5),

C IF THE "LIST" DIRECTIVE IS SET, THIS ROUTINE WILL OUTPUT THE INPUT LINE.
C LINE STATUS AND TARGET CODE
C
C IF THIS LINE A COMMENT?
C CALL DECHEX(THISPC, HEX)
C
C IS THIS LINE A COMMENT?
IF( NOT. COMNTS) GO TO 5
WRITE(13,200) LINENO,LINENumber
RETURN

5 DO 10 NI = 1, 4
10 HXCODE(NI) = HEXCOD(NI, 1)
15 NI = NI - 1
IF(NI .LE. 1) GO TO 20
IF(OPRBUF(NI) .EQ. BLANK) GO TO 15
10 WRITE(13,100) LINENO, THISPC, HEX, LINSTAR, HXCODE, LABEL, OPCODE,
1 (OPRBUF(KI), KI = 1, NI)
100 FORMAT(1X,15,5X,4A1,6X,13,5X,4A1,5X,6A1,1X,4A1,4X,60A1)
IF(WRDNUM .EQ. 1) RETURN
DO 30 IJ = 2, WRDNUM
WRITE(13,125) (HXCODE(J1, J1), J1 = 1, 4)
30 CONTINUE
25 FORMAT(37X, 4A1)
RETURN
END
SUBROUTINE ERROUT(THISPC)
IMPLICIT INTEGER (A-Z)
LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, HEXTBL, LINE
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1, ASCII2, ASCII3, ASCII4
LOGICAL*1 DEFFLG, REGFLG, BYTFLG, LABFLG, INSDLG, OPFLG, RESFLG,
1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
LOGICAL*1 PGMCTR, DUMMY2, HXCODE(4), LIN
COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4,10)
COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /SAME/ DEFFLO, LABFLG, REGFLC, BYTFLG
COMMON /LINFLG/ INSDLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA
1 LSTFLG
COMMON /ASSEMB/ PAss, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRESS,
1 MAJVAL, COMENT, ISTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ERRLDL, WRDNUM, BYNUM, ERROR
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRLCH/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
1 IMTPV, VALUE, SYMBOL, REG, REGSTA, REGATS, REGPLS,
2 VARABL, NUMBER, LITERAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OFPIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII9, ASCII1, ASCII2, ASCII3, ASCII4
COMMON /LOCAL/ NUM1, NUM2, NUM3, LINCT1, LINCT2, LINCT3, PGMCTR(5),
1 DUMMY2, OLDPC, OLDY, OLDY, OLDY, LIN(48)

DIMENSION LABEL(6), OPCOD(4), OPRBUF(60)

EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCOD(1))
EQUIVALENCE (LINE(13), OPRBUF(1))

DATA TEST /0/

LNSTAT = 0
IF (LABERR) LNSTAT = 2
IF (INSTER) LNSTAT = LNSTAT + 4
IF (OPERR) LNSTAT = LNSTAT + 8
IF (LNSTAT EQ 0) RETURN

C OBTAIN HEX VALUE OF PC
CALL DECHEX(THISPC, HEX)

DO 10 11 = 1, 4
10 HXCODE(11) = HEXCOD(11:11)
IF (TEST EQ 1) 10, 70, 20

TYPE 50
50 FORMAT(1X, '*** ERROR ***')

TYPE 60
60 FORMAT(' LINE NO ADDRESS     LINE STATUS OBJ CODE LABEL
1 INSTR OP FIELD')

TEST = 1

TYPE 100: LINENO, THISPC, HEX, LNSTAT, HXCODE, LABEL, OPCOD, OPRBUF

RETURN
END
SUBROUTINE THOCMP(HEX)
IMPLICIT INTEGER (A-Z)
LOGICAL*1 HEX(4), HEXTBL, NEG
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCIIA, ASCIIF, ASCIIZ, ASCHAR
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCIIA, ASCIIF, ASCIIZ
DIMENSION TEMP(4)

THIS ROUTINE CONVERTS A HEX NUMBER OF FOUR ASCII DIGITS TO ITS 2'S COMPLEMENT FORM IN ASCII CODE
PLACE INTEGER EQUIVALENT INTO "TEMP"
DO 30 N1 = 1, 4
   VAL = HEX(N1) - ASCII0
   IF(VAL .GT. 9) VAL = HEX(N1) - ASCIIA + 10
   TEMP(N1) = 15 - VAL
30 CONTINUE
TEMP(4) = TEMP(4) + 1

CHECK FOR OVERFLOW
N1 = 5
40 N1 = N1 - 1
   IF(TEMP(N1) .LE. 15) GO TO 50
   TEMP(N1) = 0
   IF(N1 .LE. 1) GO TO 50
   TEMP(N1-1) = TEMP(N1-1) + 1
   GO TO 40
50 DO 60 N1 = 1, 4
   HEX(N1) = HEXTBL(TEMP(N1) + 1)
60 CONTINUE
RETURN
END
APPENDIX C

VARIABLES IN LABELED COMMON

/ASSEMB/

PASS - Assembler pass number. It is set to either 1 or 2.

REGSRC - Source field workspace register number. Value range 0-15.

REGDES - Destination field workspace register number. Value range 0-15.

REGCOD() - Source (1) and destination (2) register codes. Allowable values:

   Ri = 0
   *Ri = 1
   @Ri = 2
   *Ri+ = 3

PC - Current value of Program Counter. It is reset at the beginning of pass two and has a default value of 10016 (25610). It is user modifiable (see directives AORG, /, BSS).

LINENO - Input line number.

ADDRESS - Used by Subroutine ADDNAM to return an address value for a requested identifier. Value is zero if identifier is not yet defined. It is used by Subroutine OPFLD to return the value of a complete operand field.

MAJVAL - Output from Subroutine OPFLD. It is used by Subroutine ITMSCN to check that an operand field is semantically correct. Refer to Tables 1 and 2.

COMENT - Location of the semicolon in the input line. It indicates where comments begin for that line. Value range is 1 to 60.

IDSTAT - Output from Subroutine ADDNAM. It holds the status of the retrieved identifier. Allowable values include various combinations of the following:

1 - Identifier has been defined
2 - Identifier is a register
4 - Identifier is multipli-defined
8 - Identifier is a halfword (byte).
LNSTAT - Status of the input statement. Allowable values include any cumulative combination of the following:
   2 - Label error
   4 - Instruction or directive error
   8 - Operand error

ERRNUM - Reserved for future program enhancement.

FMT - Output from Subroutine INSTRU. At the successful completion of a search of the instruction-directive tree, FMT contains the operand format number for the requested instruction. Refer to Tables 1 and 2.

DIRNUM - Output from Subroutine INSTRU. At the successful completion of a search of the instruction-directive tree, DIRNUM contains the directive number which uniquely defines the directive. Refer to Table 3.

OPER1 and OPER2 - Output from Subroutine INSTRU. At the successful completion of a search of the instruction-directive tree, OPER1 and OPER2 contain the ASCII equivalent of the instruction skeleton.

ER - State pointer for the identifier tree.

ERLABL - State pointer for the identifier tree. It points to the last source node in the string describing the current label. Its link is always "STOP." Refer to discussion on the identifier tree.

WRDNUM - Number of words required by either an instruction or a directive. It is computed during the first (pass) call to Subroutine ITMSCN. Value range is:
   Instructions  1-3
   Directives  0-10

BYTNUM - Number of bytes required by either an instruction or a directive. It is computed during the first (pass) call to Subroutine ITMSCN and is an input to Subroutine ABSOUT. Value range is:
   Instructions  2-6
   Directives  0-20

ERROR - Reserved for future program enhancement.

These variables are defined within GIRL² to represent the particular character as named.

/ERRFLG/ - The following logical*1 variables have the indicated meanings when set to .TRUE.: 

LABERR  - Label error such as multiple definition.
DIRERR  - Not used in present version.
OPERR   - Error in operand field.
TYPERR  - Reserved for future program enhancement.
INSTER  - Instruction name or directive name error.
DUMMY   - Filler byte to make this common block contain an even number of bytes.

/NAMFLG/ - The following flags indicate label status. If the status of an identifier includes any of the flags, then that identifier has the following properties:

DEFFLG  - Identifier has been defined and given an address by virtue of placement in the label field.
LABFLG  - Not used in present version.
REGFLG  - Identifier is a register.
MLTFLG  - Identifier appears in the label field more than once.
BYTFLG  - Identifier is halfword size.
DUMB    - Filler byte to make this common block contain an even number of blocks.

/LINFLG/ 

INSFLG  - Not used in present version.
OPFLG   - Not used in present version.
RESFLG  - Reserved for future program enhancement.

/DIRFLG/ 

ENDFLG  - Set to .TRUE. when END directive is encountered.
COMNTS  - Set to .TRUE. if entire input line is a comment statement.
EXTRA   - Input to Subroutine ITMSCN from Subroutine OPFLD. It is set to .TRUE. If there is a memory reference in an operand field, thus requiring an extra machine code word on output.

LSTFLG   - Set to .TRUE. if LIST directive is encountered.
APPENDIX D

SAMPLE SET OF INSTRUCTIONS AND DIRECTIVES
LIST
A R1,R2;
JIM AB R2,R1;
ABS R1;
AI R1,>ABC;
ANDI R2,>ABC;
UNL ,
ABS R1;
AI R1,>ABC;
ANDI R2,>ABC;
B R1;
BILL BL R2;
BLMP R2;
JOE BSS >A;
SID BYTE >10,12B;
C R2,R1;
BYTE 'A';
BYTE 'B';
BYTE 'C';
BYTE 'D';
CB R1,R2;
CI R1,>7FF;
CLR R2;
COC R2,R1;
CIC R2,R1;
WALTER DATA 'AB'; >23;
DEC R1;
DECT R2;
DIV R2,R1;
DXOP ;
JOAN EQU 3+5*6;
CNTHIA EQU JOAN+10+6;
AI R1,JOAN;
AJ R1,RAE;
SHIRA EVEN ;
CAROL BYTE 'A',@B,>EF;
RAE EVEN ;
IDLE ;
IN R1;
INC R2;
INCT R2;
INV R1;
JEQ 300;
JGT #300;
JH 300;
JNE 300;
JL 300;
JLE 300;
JLT 300;
JMP 300;
JNC 300;
JNE 300;
JNO 300;
JOC 300;
JOP 300;
QRTF LDCR R1,B;
PLKJ LI R1,>ABC;
FMODJ LIMI >ABC;
APPENDIX E

SOURCE AND IDENTIFIER LISTING GENERATED FROM APPENDIX A AS INPUT
<table>
<thead>
<tr>
<th>LINE NO.</th>
<th>ADDRESS</th>
<th>LINE STATUS</th>
<th>OBJ CODE</th>
<th>LABEL</th>
<th>INSTR.</th>
<th>OP FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>256</td>
<td>0100</td>
<td>0</td>
<td>AOB1</td>
<td>LIST</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>258</td>
<td>0102</td>
<td>0</td>
<td>B042</td>
<td>JIM</td>
<td>R1,R2;</td>
</tr>
<tr>
<td>3</td>
<td>260</td>
<td>0104</td>
<td>0</td>
<td>0741</td>
<td>ABS</td>
<td>R2,R1;</td>
</tr>
<tr>
<td>4</td>
<td>262</td>
<td>0106</td>
<td>0</td>
<td>0221</td>
<td>AI</td>
<td>R1,&gt;ABC;</td>
</tr>
<tr>
<td>5</td>
<td>266</td>
<td>010A</td>
<td>0</td>
<td>0242</td>
<td>ANDI</td>
<td>R2,&gt;ABC;</td>
</tr>
<tr>
<td>6</td>
<td>270</td>
<td>010E</td>
<td>0</td>
<td>0741</td>
<td>ABS</td>
<td>R1;</td>
</tr>
<tr>
<td>7</td>
<td>272</td>
<td>0110</td>
<td>0</td>
<td>0221</td>
<td>AI</td>
<td>R1,&gt;ABC;</td>
</tr>
<tr>
<td>8</td>
<td>276</td>
<td>0114</td>
<td>0</td>
<td>0242</td>
<td>ANDI</td>
<td>R2,&gt;ABC;</td>
</tr>
<tr>
<td>9</td>
<td>280</td>
<td>0118</td>
<td>0</td>
<td>0441</td>
<td>B</td>
<td>R1;</td>
</tr>
<tr>
<td>10</td>
<td>282</td>
<td>011A</td>
<td>0</td>
<td>0682</td>
<td>BILL</td>
<td>R2;</td>
</tr>
<tr>
<td>11</td>
<td>284</td>
<td>011C</td>
<td>0</td>
<td>0402</td>
<td>BLWP</td>
<td>R2;</td>
</tr>
<tr>
<td>12</td>
<td>286</td>
<td>011E</td>
<td>0</td>
<td>0741</td>
<td>JDE</td>
<td>&gt;A;</td>
</tr>
<tr>
<td>13</td>
<td>296</td>
<td>012B</td>
<td>0</td>
<td>1080</td>
<td>SID</td>
<td>BYTE &gt;10,128;</td>
</tr>
<tr>
<td>14</td>
<td>298</td>
<td>012A</td>
<td>0</td>
<td>0441</td>
<td>CLR</td>
<td>R2;</td>
</tr>
<tr>
<td>15</td>
<td>300</td>
<td>012C</td>
<td>0</td>
<td>04C2</td>
<td>CLR</td>
<td>R2;</td>
</tr>
<tr>
<td>16</td>
<td>301</td>
<td>012D</td>
<td>0</td>
<td>0221</td>
<td>CI</td>
<td>R1,&gt;7FF;</td>
</tr>
<tr>
<td>17</td>
<td>302</td>
<td>012E</td>
<td>0</td>
<td>0221</td>
<td>CI</td>
<td>R1,&gt;7FF;</td>
</tr>
<tr>
<td>18</td>
<td>303</td>
<td>012F</td>
<td>0</td>
<td>1081</td>
<td>CB</td>
<td>R1,R2;</td>
</tr>
<tr>
<td>19</td>
<td>304</td>
<td>0130</td>
<td>0</td>
<td>0281</td>
<td>CI</td>
<td>R1,&gt;7FF;</td>
</tr>
<tr>
<td>20</td>
<td>306</td>
<td>0132</td>
<td>0</td>
<td>0281</td>
<td>CI</td>
<td>R1,&gt;7FF;</td>
</tr>
<tr>
<td>21</td>
<td>310</td>
<td>0136</td>
<td>0</td>
<td>04C2</td>
<td>CLR</td>
<td>R2;</td>
</tr>
<tr>
<td>22</td>
<td>312</td>
<td>0138</td>
<td>0</td>
<td>2042</td>
<td>JOAN</td>
<td>R2,R1;</td>
</tr>
<tr>
<td>23</td>
<td>314</td>
<td>013A</td>
<td>0</td>
<td>2442</td>
<td>JOAN</td>
<td>R2,R1;</td>
</tr>
<tr>
<td>24</td>
<td>316</td>
<td>013C</td>
<td>0</td>
<td>4142</td>
<td>CAROL</td>
<td>DATA &gt;AB,&gt;23;</td>
</tr>
<tr>
<td>25</td>
<td>320</td>
<td>0140</td>
<td>0</td>
<td>0601</td>
<td>DEC</td>
<td>R1;</td>
</tr>
<tr>
<td>26</td>
<td>322</td>
<td>0142</td>
<td>0</td>
<td>0642</td>
<td>DEC</td>
<td>R2;</td>
</tr>
<tr>
<td>27</td>
<td>324</td>
<td>0144</td>
<td>0</td>
<td>3C42</td>
<td>DIV</td>
<td>R2,R1;</td>
</tr>
<tr>
<td>28</td>
<td>326</td>
<td>0146</td>
<td>0</td>
<td>0221</td>
<td>CI</td>
<td>R1,&gt;7FF;</td>
</tr>
<tr>
<td>29</td>
<td>330</td>
<td>014A</td>
<td>0</td>
<td>0221</td>
<td>CI</td>
<td>R1,&gt;7FF;</td>
</tr>
<tr>
<td>30</td>
<td>334</td>
<td>014E</td>
<td>0</td>
<td>4142</td>
<td>CAROL</td>
<td>BYTE &gt;A*,&gt;BS,&gt;EF;</td>
</tr>
<tr>
<td>31</td>
<td>336</td>
<td>0151</td>
<td>0</td>
<td>00</td>
<td>RAE</td>
<td>E,EN;</td>
</tr>
<tr>
<td>32</td>
<td>338</td>
<td>0152</td>
<td>0</td>
<td>0340</td>
<td>IDLE</td>
<td>;</td>
</tr>
<tr>
<td>33</td>
<td>340</td>
<td>0154</td>
<td>0</td>
<td>0582</td>
<td>INC</td>
<td>R2;</td>
</tr>
<tr>
<td>34</td>
<td>342</td>
<td>0156</td>
<td>0</td>
<td>05C2</td>
<td>INCT</td>
<td>R2;</td>
</tr>
<tr>
<td>35</td>
<td>344</td>
<td>0158</td>
<td>0</td>
<td>0541</td>
<td>INV</td>
<td>R1;</td>
</tr>
<tr>
<td>36</td>
<td>346</td>
<td>015A</td>
<td>0</td>
<td>13E7</td>
<td>JEQ</td>
<td>300;</td>
</tr>
<tr>
<td>37</td>
<td>348</td>
<td>015C</td>
<td>0</td>
<td>15E6</td>
<td>JGT</td>
<td>300;</td>
</tr>
<tr>
<td>38</td>
<td>350</td>
<td>015E</td>
<td>0</td>
<td>1BE5</td>
<td>JH</td>
<td>300;</td>
</tr>
<tr>
<td>39</td>
<td>352</td>
<td>0160</td>
<td>0</td>
<td>14E4</td>
<td>JHE</td>
<td>300;</td>
</tr>
<tr>
<td>40</td>
<td>354</td>
<td>0162</td>
<td>0</td>
<td>1AE3</td>
<td>JL</td>
<td>300;</td>
</tr>
<tr>
<td>41</td>
<td>355</td>
<td>0164</td>
<td>0</td>
<td>12E2</td>
<td>JLE</td>
<td>300;</td>
</tr>
<tr>
<td>NAME</td>
<td>STATUS</td>
<td>ADDRESS</td>
<td>REG</td>
<td>UNDEF</td>
<td>MULT</td>
<td>DEF</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>---------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>-----</td>
</tr>
<tr>
<td>RO</td>
<td></td>
<td>0 0000</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RI</td>
<td></td>
<td>1 0001</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td>2 0002</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td>3 0003</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td>4 0004</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
<td>5 0005</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td></td>
<td>6 0006</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td></td>
<td>7 0007</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td></td>
<td>8 0008</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td></td>
<td>9 0009</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td></td>
<td>10 000A</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td></td>
<td>11 000B</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td></td>
<td>12 000C</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R13</td>
<td></td>
<td>13 000D</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R14</td>
<td></td>
<td>14 000E</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R15</td>
<td></td>
<td>15 000F</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JIM</td>
<td></td>
<td>258 0102</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BILL</td>
<td></td>
<td>282 011A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOE</td>
<td></td>
<td>286 011E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SID</td>
<td></td>
<td>296 0120</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOAN</td>
<td></td>
<td>48 0030</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNTHIA</td>
<td></td>
<td>64 0040</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAEE</td>
<td></td>
<td>337 0151</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHIRA</td>
<td></td>
<td>334 014E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAROL</td>
<td></td>
<td>334 014E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAE</td>
<td></td>
<td>337 0151</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRTF</td>
<td></td>
<td>374 0176</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLKJ</td>
<td></td>
<td>376 0178</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FHMGDJ</td>
<td></td>
<td>380 017C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PYTRV</td>
<td></td>
<td>384 0180</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HJFG</td>
<td></td>
<td>388 0184</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFD</td>
<td></td>
<td>390 0186</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WR</td>
<td></td>
<td>392 0188</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X2C</td>
<td></td>
<td>394 018A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRQY</td>
<td></td>
<td>396 018C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LKUY</td>
<td></td>
<td>398 018E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PQYR</td>
<td></td>
<td>402 0192</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDFWG</td>
<td></td>
<td>406 0196</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WTUEY</td>
<td></td>
<td>408 0198</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAZX</td>
<td></td>
<td>410 019A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLMN</td>
<td></td>
<td>412 019C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EFVBG</td>
<td></td>
<td>414 019E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCFE</td>
<td></td>
<td>416 01A0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POI J</td>
<td></td>
<td>418 01A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GWER</td>
<td></td>
<td>420 01A4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WALT</td>
<td></td>
<td>422 01A6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KLEE</td>
<td></td>
<td>424 01A8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TJUF</td>
<td></td>
<td>426 01AA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PETF W</td>
<td></td>
<td>428 01AC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDWER</td>
<td></td>
<td>430 01AE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WDFVE</td>
<td></td>
<td>432 01B0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMEFR</td>
<td></td>
<td>434 01B2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLKJH</td>
<td></td>
<td>438 01B6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SKJHY</td>
<td></td>
<td>440 01B8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LADMT</td>
<td></td>
<td>442 01BA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX F

MACHINE CODE GENERATED FROM APPENDIX A AS INPUT
0100: A0 B1 B0 42 07 41 02 21 0A BC 02 42 0A BC 07 41
0110: 02 21 0A BC 02 42 0A BC 04 41 06 82 04 02 00 41
0120: 00 00 00 00 00 00 00 10 80 80 42 41 42 43 44
0130: 90 81 02 42 OA BC 04 41 00 FF 20 42 24 42 41 42 00 23
0140: 06 01 06 42 3C 42 02 21 00 30 02 21 01 51 41 42
0150: EF 00 03 40 2C 41 05 82 05 C2 05 41 13 E7 15 E6
0160: 1B E5 14 E4 1A E3 12 E2 11 E1 10 E0 17 DF 16 DE
0170: 19 DD 1B DC 1C DB 32 01 02 01 0A BC 03 00 0A BC
0180: 02 E0 0A BC 00 81 D0 81 38 42 05 01 10 00 02 61
0190: 0A BC 2C B1 04 5B 03 80 60 B1 70 42 1D 0E 1E 14
01A0: 07 01 0A C1 E0 81 F0 42 08 A2 0B C1 09 82 37 C2
01B0: 02 C1 02 A1 06 C1 40 42 50 42 1F 12 45 5B 41 4D
01C0: 50 4C 45 00
REFERENCES


## INITIAL DISTRIBUTION

<table>
<thead>
<tr>
<th>Copies</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CHONR/430D</td>
<td>L. Culpepper</td>
</tr>
<tr>
<td>1</td>
<td>NRL</td>
<td>C. Godfrey</td>
</tr>
<tr>
<td>1</td>
<td>NSWC</td>
<td>W. Gorham, Jr.</td>
</tr>
<tr>
<td>1</td>
<td>NUSC</td>
<td>J. Schot</td>
</tr>
<tr>
<td>1</td>
<td>NOSC</td>
<td>H. Feingold</td>
</tr>
<tr>
<td>1</td>
<td>NAVSUP/0431C, G. Bernstein</td>
<td>H. Haussling</td>
</tr>
<tr>
<td>2</td>
<td>NAVSEA</td>
<td>S. Dhir</td>
</tr>
<tr>
<td>1</td>
<td>SEA 312</td>
<td>J. McKee</td>
</tr>
<tr>
<td>1</td>
<td>SEA 612</td>
<td>T. Corin</td>
</tr>
<tr>
<td>1</td>
<td>Rome Air Development Center</td>
<td>A. Cinque</td>
</tr>
<tr>
<td>1</td>
<td>DTIC</td>
<td>J. Brainin</td>
</tr>
<tr>
<td>1</td>
<td>185</td>
<td>H. Sheridan</td>
</tr>
<tr>
<td>1</td>
<td>1855</td>
<td>R. Brengs</td>
</tr>
<tr>
<td>2</td>
<td>Texas Instruments, Inc.</td>
<td>R. Ploe</td>
</tr>
<tr>
<td>1</td>
<td>1802.2</td>
<td>M. Zubkoff</td>
</tr>
<tr>
<td>1</td>
<td>SEA 612</td>
<td>G. Gray</td>
</tr>
</tbody>
</table>

## CENTER DISTRIBUTION

<table>
<thead>
<tr>
<th>Copies</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5211.1</td>
<td>Reports Distribution</td>
</tr>
<tr>
<td>1</td>
<td>522.1</td>
<td>Unclassified Lib (C)</td>
</tr>
<tr>
<td>1</td>
<td>522.2</td>
<td>Unclassified Lib (A)</td>
</tr>
<tr>
<td>1</td>
<td>1576</td>
<td>C. Bell</td>
</tr>
<tr>
<td>1</td>
<td>1576</td>
<td>G. Gleissner</td>
</tr>
<tr>
<td>1</td>
<td>1802.2</td>
<td>F. Frenkiel</td>
</tr>
<tr>
<td>1</td>
<td>1803</td>
<td>S. Rainey</td>
</tr>
<tr>
<td>1</td>
<td>1804</td>
<td>L. Avrunin</td>
</tr>
<tr>
<td>1</td>
<td>1805</td>
<td>E. Cuthill</td>
</tr>
<tr>
<td>1</td>
<td>1806</td>
<td>R. Santamaria</td>
</tr>
<tr>
<td>2</td>
<td>1809.3</td>
<td>D. Harris</td>
</tr>
<tr>
<td>1</td>
<td>182</td>
<td>A. Camara</td>
</tr>
<tr>
<td>1</td>
<td>1821</td>
<td>D. Jefferson</td>
</tr>
<tr>
<td>1</td>
<td>1822</td>
<td>T. Rhodes</td>
</tr>
<tr>
<td>1</td>
<td>1822</td>
<td>S. Berkowitz</td>
</tr>
<tr>
<td>1</td>
<td>1824</td>
<td>J. Carlberg</td>
</tr>
<tr>
<td>1</td>
<td>1824</td>
<td>J. Garner</td>
</tr>
<tr>
<td>1</td>
<td>1824</td>
<td>P. Marques</td>
</tr>
<tr>
<td>20</td>
<td>1824</td>
<td>I. Zaritsky</td>
</tr>
</tbody>
</table>