PREFACE

This release serves as a preliminary user document and supplement to the forthcoming FORTRAN II reference manual for UNIVAC Solid-State Systems (UP 3843). It contains a brief description of the FORTRAN II compiler, and a machine-generated annotated process chart of the compiler.

The chart, beginning on page 6, was produced as a by-product of a special-purpose compiler used in developing the FORTRAN II compiler, and is reproduced directly from a copy printed by the USS Printer. Standard charting techniques are generally followed, with the following alterations in symbology to accommodate these techniques to the characters available on the Printer:

The Operation Box (rectangle) is formed by lines of hyphens above and below, colons at left and right, and periods at corners.

The Decision Box (oblong) is formed by lines of hyphens above and below, and sets of parentheses arranged as (at left and) at right.

Connecting lines are indicated by rows of periods (horizontal), colons (vertical), and O's (at corners and as connectors).

Direction of flow is indicated by parentheses representing arrows. An arrow pointing to the right is indicated by ), and arrow pointing left is indicated by (.

Entrances are indicated by (---IN---); exits, by EXIT; and remote connectors, by symbolic entries referring to subheadings in the accompanying annotation.

The reader should note that "missing" page numbers have been omitted in order to keep double pages facing each other.

Blank pages have been inserted where necessary to keep the first and second pages of double-page routines facing each other.
1. FORTRAN II Compiler Pass 1

The translator is divided into two major co-routines, 'SCAN' and 'GEN'.

SCAN has the duty of reading cards, condensing identifiers and constants into single entities and to feed items, in a convenient internal code, one at a time to GEN.

GEN has the duty of producing object code from these items. Control is passed between GEN and SCAN in a fashion such that each routine looks like a subroutine of the other.

The program begins by printing the title line, feeding a card, and going to the initialization routine, STEP Z1.

TABLE OF CONTENTS

A. Array Subscripting ........................................... 34
B. Binary and Arithmetic Operators .............................. 32
C. Constant Scanner .............................................. 14
D. Do Loop Control .............................................. 40
E. Equivalence Processing ....................................... 50
F. Function Calls ................................................ 42
G. Generator Control ............................................ 6
H. Assembler Structure .......................................... 17
L. Linked Memory Subroutines .................................. 12
N. 'Get Next Character' Routine ............................... 10
P. Function and Subroutine Declarations ..................... 48
Q. Special Scanning Routines .................................. 16
S. Scanner Control ............................................... 8
T. Symbol Table Search ......................................... 11
U. Unary Operators and Special Generators ..................... 38
W. Input/Output (Read Punch Print) .............................. 46
X. Processing Format String ..................................... 44
Z. Initialization and Termination ............................... 49

TABLE OF FORMATS

Information inside the compiler is treated in two principal formats, one for the symbol table entries in the scanner, and another for generator co-routine.

Symbol table equivalents are in the format

KM AAAA LLLL

where LLLL is a link to the next symbol, for searching

K equals 0: Simple Variable
M is 0: No memory assignment as yet AAAAA is 0000
M is 1: Assigned AAAAA in unique storage
M is 2: Equivalence, not yet assigned. AAAA is link to other members of the equivalence class.

M is 3: Assigned AAAA in common.

M is 4: A formal parameter, whose subroutines are assigned AAAA, AAAA+1, and AAAA+2 in unique.

M is 5 The symbol is a 10 digit constant. If AAAA is 0, this constant has not been needed in object program yet, else it is assigned to location AAAA in unique.

K equals 3: Array

AAAA links to the dimension table entry, M is ignored. The dimension table has N+1 entries if there are N subscripts to this array.

AAAA+0: 3 M BBBB RRRR
AAAA+1: 0 0 TTTT SSSS
AAAA+2: 0 0 CCCC OOOO
AAAA+3: 0 0 CCCC OOOO ETC

SSSS is link back to symbol table entry.

CCCC words, if present, are links to symbol table entries for constants (except for the last dimension).

TTTT is the total length of the array

M is 0: No memory assignment has been made as yet, BBBB is 0.

M is 1: The address BBBB is for A(1), i.e. the first cell of the array, in unique storage.

M is 2: Equivalence array A(RRRR), BBBB is link to other elements in equivalence class.

M is 3: Same as M equal to 1 except common storage.

M is 4: Formal parameter, base address is stored in BBBB of unique storage.

K equals 5: Label

AAAA is the assignment in program storage.

M is 0: Unassigned as yet.

M is 1: Temporary assignment for Do Loops. AAAA links to an item in llist,

AAAA+0: 02 TTTT XXXX
AAAA+1: SS SSSS LLLL

where XXXX is llist link,
TTTT is temporary assignment of the label,
SS SSSS is like a permanent symbol table entry for labels, and LLLL is a link back to the symbol table entry.

M is 2: AAAA is the assignment for the label.

K equals 6: Function

M is 2: Assigned AAAA in program storage.
M is 5: Assigned AAAA, external reference.
M is 9: Special operator for scanner only.

K equals 7, 8, or 9 Operator, reserved word.

KM AAAA is code for operators.

In equivalence loops, a special meaning is given for K equal to
9, when M AAAA is a change in reference point of the equivalence
loop, plus 500000.

Generator Code Formats

K T SSSS COOP

For operands, P is the sign, 0 plus, 5 minus

T is the type: 0 floating, 1 integer, 2 unspecified.
K equals 0: Simple variable, or a constant (if C is 5).
SSSS is a link to the corresponding symbol

table entry.

K equals 1: Computed result in RA.
K equals 2: Index Register 1 (do variable).
K equals 3: Array
SSSS links to dimension table entry when this
array is sent from scan, and then after the

subscript for the array is processed, SSSS
links to an entry on the ARAS list. See rou-
tine A for the formats in ARAS.

K equals 4: Temp Storage
SSSS is the assignment in unique.

K equals 5: Label
Here SSSS is a link to the corresponding
symbol table entry.

K equals 6: Function
SSSS is link to symbol table

K equals 7: Special
In the operand stack this is sometimes used
for an array without a subscript.

K equals 7, 8, or 9: Operator
KT SSSS is the same as the symbol table entry
KM AAAA. KT is the priority of the operator.
99 means action for the operator immediately
upon entry to GEN. 98 means the operator is
a UNARY operator. Else T equal to 1, 3, 6,
or 8 means immediate action before entering
on the operator stack (see GEN control)

Reserved word codes which follow give the symbol table entries
for all reserved identifiers and special characters, together
with a symbolic reference corresponding to the assembly listing of

)))FORTRAN(((
Reserved word codes

<table>
<thead>
<tr>
<th>ITEM:</th>
<th>CODE:</th>
<th>SYMBOLIC:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>9941050000</td>
<td>99 SIGN&amp;</td>
</tr>
<tr>
<td>-</td>
<td>9941040000</td>
<td>99 SIGN-</td>
</tr>
<tr>
<td>/</td>
<td>8941150000</td>
<td>84 SIGN/</td>
</tr>
<tr>
<td>%</td>
<td>9941010000</td>
<td>99 SIGN%</td>
</tr>
<tr>
<td>*</td>
<td>9941110000</td>
<td>99 SIGN*</td>
</tr>
<tr>
<td>$</td>
<td>7341140000</td>
<td>73 SIGN$</td>
</tr>
<tr>
<td>:</td>
<td>7000000000</td>
<td>70 0000</td>
</tr>
<tr>
<td>;</td>
<td>7841130000</td>
<td>78 SIGN;</td>
</tr>
<tr>
<td>+</td>
<td>9941055000</td>
<td>99 SIGN&amp;</td>
</tr>
<tr>
<td>(</td>
<td>9941170000</td>
<td>99 SIGN#</td>
</tr>
<tr>
<td>)</td>
<td>9941010000</td>
<td>99 SIGN%</td>
</tr>
<tr>
<td>NO</td>
<td>9941320000</td>
<td>70 0000</td>
</tr>
<tr>
<td>LIST</td>
<td>6941330000</td>
<td>69 WDNOD</td>
</tr>
<tr>
<td>CORE</td>
<td>6941370000</td>
<td>69 WDLIS</td>
</tr>
<tr>
<td>TRACE</td>
<td>6941360000</td>
<td>69 WDCOR</td>
</tr>
<tr>
<td>TO</td>
<td>6940500000</td>
<td>69 WDTRC</td>
</tr>
<tr>
<td>THROUGH</td>
<td>9941380000</td>
<td>69 SCAN1</td>
</tr>
<tr>
<td>GO</td>
<td>9941310000</td>
<td>99 WDTRU</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>9941300000</td>
<td>99 WDGO</td>
</tr>
<tr>
<td>IF</td>
<td>9941070000</td>
<td>99 ASS1</td>
</tr>
<tr>
<td>DO</td>
<td>9941000000</td>
<td>99 WDIF</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>6940500000</td>
<td>99 WDDO</td>
</tr>
<tr>
<td>PAUSE</td>
<td>9841410000</td>
<td>69 SCAN1</td>
</tr>
<tr>
<td>STOP</td>
<td>9841420000</td>
<td>93 WDPOZ</td>
</tr>
<tr>
<td>END</td>
<td>9941340000</td>
<td>98 WDSUB</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>9941440000</td>
<td>99 WDPUN</td>
</tr>
<tr>
<td>SUBROUTINE</td>
<td>9941450000</td>
<td>99 WDSUB</td>
</tr>
<tr>
<td>READ</td>
<td>9941460000</td>
<td>99 WRRED</td>
</tr>
<tr>
<td>PRINT</td>
<td>9941470000</td>
<td>99 WDPRT</td>
</tr>
<tr>
<td>FORMAT</td>
<td>9941490000</td>
<td>99 WDFMT</td>
</tr>
<tr>
<td>RETURN</td>
<td>9941500000</td>
<td>99 WDRTN</td>
</tr>
<tr>
<td>DIMENSION</td>
<td>9941510000</td>
<td>99 WDDIM</td>
</tr>
<tr>
<td>COMMON</td>
<td>9941520000</td>
<td>99 WDCOM</td>
</tr>
<tr>
<td>EQUIVLENCE</td>
<td>9941530000</td>
<td>99 WDEQU</td>
</tr>
<tr>
<td>SIN</td>
<td>9841200000</td>
<td>98 SINF</td>
</tr>
<tr>
<td>COS</td>
<td>9841210000</td>
<td>98 COSF</td>
</tr>
<tr>
<td>SQRT</td>
<td>9841190000</td>
<td>98 SQRTF</td>
</tr>
<tr>
<td>TAN</td>
<td>9841220000</td>
<td>98 TANF</td>
</tr>
<tr>
<td>ARCTAN</td>
<td>9841230000</td>
<td>98 ATANF</td>
</tr>
<tr>
<td>LN</td>
<td>9841240000</td>
<td>98 LNF</td>
</tr>
<tr>
<td>EXP</td>
<td>9841250000</td>
<td>98 EXPF</td>
</tr>
<tr>
<td>ABS</td>
<td>9841260000</td>
<td>98 ABSF</td>
</tr>
<tr>
<td>FLOAT</td>
<td>9841540000</td>
<td>98 FLOTF</td>
</tr>
<tr>
<td>FIX</td>
<td>9841550000</td>
<td>98 FIXF</td>
</tr>
<tr>
<td>PUNCH</td>
<td>9941590000</td>
<td>99 WDPCCH</td>
</tr>
<tr>
<td>CALL</td>
<td>9941600000</td>
<td>99 WDCAL</td>
</tr>
<tr>
<td>NOT</td>
<td>9941610000</td>
<td>98 BCOMP</td>
</tr>
<tr>
<td>OR</td>
<td>7941630000</td>
<td>79 BOR</td>
</tr>
<tr>
<td>AND</td>
<td>8041620000</td>
<td>80 BLAND</td>
</tr>
<tr>
<td>CARDS</td>
<td>6941660000</td>
<td>69 WDFRG</td>
</tr>
</tbody>
</table>
---IN---

G1. SCAN NEXT ITEM.

G2. IS IT AN OPERATOR?

G3. OPERAND STACKED?

G4. IS IT AN ARRAY?

G5. SCAN NEXT ITEM.

G6. WHAT KIND OPERATOR?

G7. PUT OP IN OHOLD.

G8. GENERATOR CONTROL

THIS ROUTINE CONTROLS THE
GENERATOR CO-ROUTINE.

THE NORMAL EXIT AT THE COMPLETION OF A GENER-
ATED ITEM IS TO G1, WHICH STARTS THE
PROCESSING OF THE NEXT ITEM, AT THE END OF
GENERATING CODE FOR CERTAIN OPERATORS, EXIT
OCCURS TO G10 RATHER THAN G1, SINCE WE MAY
WISH TO PERFORM SEVERAL OPERATIONS BEFORE
SCANNING ANOTHER ITEM.

G1. SCAN NEXT ITEM.

ACTIVATE THE SCANNER CO-ROUTINE.

NORMALLY THIS MEANS WE ENTER STEP S1.

G2. IS IT AN OPERATOR?

IF THE ITEM SCANNED IS AN OPERATOR, GO TO G6.

G3. OPERAND STACKED?

PUT THE ITEM AT THE TOP OF THE OPERAND STACK.

G4. IS IT AN ARRAY?

IF THE OPERAND IS A DIMENSIONED VARIABLE,
GO TO A1.

G5. SCAN NEXT ITEM.

IF THE NEXT ITEM IS A LEFT PARENTHESIS, WE
TENTATIVELY HAVE A FUNCTION CALL SO WE GO TO
STEP F1.

OTHERWISE WE GO BACK TO STEP G2.

G6. WHAT KIND OPERATOR?

IF THE OPERATOR JUST SCANNED IS ONE THAT
REQUIRES IMMEDIATE ACTION (CODE 99), BRANCH
TO THE ROUTINE FOR THIS OP.

IF WE HAVE A UNARY OPERATOR (CODE 9b) SUCH
AS LN OR ABS, GO TO G20.

OTHERWISE WE HAVE A BINARY OPERATOR
OR A DELIMITER WHOSE PRECEDENCE IS TO BE
TESTED.

G7. PUT OP IN OHOLD.

PUT THE OPERATOR JUST SCANNED INTO LOCATION
'OHOLD' BEFORE DECIDING WHAT TO DO WITH IT.

G10. P(RATOR)IP(OHOLD)

CHECK THE PRECEDENCE OF THE TOP OPERATOR ON
THE OPERATOR STACK AGAINST THE PRECEDENCE OF
THE OPERATOR IN 'OHOLD'.

IF IT IS LESS (E.G., IN A+B*C; + IS LESS
THAN *), WE MUST WAIT BEFORE OPERATING
FURTHER SO WE GO TO G19.

IF IT HAS GREATER PRECEDENCE OR
EQUAL PRECEDENCE HOWEVER, THE OPERATOR ON TOP
OF THE STACK IS REMOVED AND WE BRANCH TO THE
APPROPRIATE ROUTINE FOR THIS OP.

PRECEDENCE IS 70 FOR VARIOUS KINDS OF LEFT
PARENTHESSES; 73 FOR ! ; 75 FOR EQUALS
78 FOR COMMA; 79 FOR OR; 80 FOR AND;
82 FOR PLUS AND MINUS; 84 FOR UNARY MINUS,
FOR MULTIPLY; AND FOR DIVIDE; 87 FOR POWER,
AND 93 FOR UNARY OPERATORS

G19. COMMA OR SEMICOLON

IF OHOLD HAS A PRECEDENCE WHOSE UNITS DIGIT
IS1,3,6; OR 8 IT MEANS WE ARE TO BRANCH TO
THIS OP NOW THAT THE PRECEDENCE HAS BEEN
CHECkED. AT PRESENT THIS IS USED ONLY FOR
SEMICOLON (EDN OF STATEMENT) OR COMMA AND THE
MEANS BRANCH TO THE ROUTINE SPECIFIED BY THE
CURRENT MODE.
OTHERWISE WE GO TO G20 TO PUT OHOHLD
ON THE OPERATOR STACK
G20+OPERATOR STACKED
THE OPERATOR IS PUT ON TOP OF THE OPERATOR
STACK AND WE RETURN TO G1.
CODING DETAILS:
UPON ENTRY TO GET, REGISTER A CONTAINS THE
CURRENT ITEM AND REGISTER X CONTAINS THE
PREVIOUS ITEM. THESE ARE IN GENERATOR CODE.
WHICH IS EXPLAINED IN THE TABLE OF FORMATS
IN THE BEGINNING OF THE FLOWCHARTS.
S1. NEXT CHARACTER

S1. NEXT CHARACTER

S2. WHAT KIND

S3. LOOK FOR IJLMN

S4. NEXT CHARACTERS

S5. SEARCH SYMBOL TABLE

S6. TRANSLATE TO GEN CODE

S7. SCAN CURR

S8. SCANNER CONTROL

S9. This routine controls the Scanner co-routine.

S10. Normal entry to the scanner is to step S11

S11. Next character

S12. What kind

S13. If the character is numeric, it is the

S14. Beginning of a constant so we go to C1

S15. A decimal point also means a constant; go to

S16. Step C2

S17. If the character is alphabetic it means the

S18. First letter of an identifier, so we go to

S19. S3

S20. If the character is blank, return to S1;

S21. Otherwise we have a special character. Each

S22. Special character is treated exactly as an

S23. Identifier to length 1 and we go to step S5.

S24. S3

S25. Look for IJLMN

S26. If this character is the letter I through N;

S27. Record for future reference that this

S28. Identifier is integer type. Also prepare to

S29. Build up to five characters of every iden-

S30. Tifier in a computer word, in the form

S31. ZZZZIAXNNN with leading blanks.

S32. Next characters

S33. Successively get characters from the card

S34. (routine N) until the first non-alphanumeric

S35. Character appears, if the terminal character

S36. Is not blank, put it back on the card so it

S37. Will come through again next time.

S38. Search symbol table

S39. Activate routine T to search for this ident-

S40. If not found, it is entered in the

S41. Table as a simple variable. If found, the

S42. Code found is used in step S10.

S43. Translate to gen code

S44. We have an item which we want to send

S45. To the generator, but it is in symbol table

S46. Format rather than generator format.

S47. Specifications of these formats are given at

S48. The beginning of the flowchart listings,

S49. The conversion is made at this point. If the

S50. Special code 69 occurs here a branch is made

S51. To the special scanner operator which never

S52. Gets to the generator co-routine, such as

S53. TRACE, LIST, CARDS, ETC., THE APOSTROPHE OPERAT

S54. (meaning end of card); routine G, is one of

S55. These special scanner operators; the

S56. Others are mentioned in step U29.

S57. Send to GEN

S58. The coded item is sent to GEN, usually

S59. This is to step G1, upon reentry, scan will
**N0. GET NEXT CHARACTER ROUTINE**

* N1. WAS CHAR PUT BACK
  * IF A CHARACTER HAS BEEN 'PUT BACK' ON THE CARD
  * RE-EMIT THIS CHARACTER AND EXIT.

* N2. END OF WORD
  * IF WE ARE NOT AT THE END OF THE CURRENT
  * TEN-COLUMN PART OF THE CARD, GO TO STEP N10;
  * ELSE WE MUST BRING UP ANOTHER SECTION OF THE
  * CARD.

* N3. END OF CARD
  * IF WE ARE AT THE END OF THIS CARD, GO TO
  * STEP N20.

* N4. GET NEW WORD
  * BRING UP THE NEW WORD. THIS MEANS USUALLY
  * THAT THE NEXT TEN ZONES AND NEXT TEN
  * NUMERICS ARE Brought UP. SPECIAL ACTION IS
  * TAKEN ON THE 8TH WORD OF 60-COLUMN CARDS
  * TO STOP AFTER COLUMN 72 AND ON THE FIRST
  * WORD TO START EITHER AT COLUMN 7 OR AT
  * COLUMN 1 IF THERE IS A LABEL.

* N10. EXTRACT NEXT CHAR
  * REMOVE THE NEXT CHARACTER FROM THE CARD AND
  * EXIT.

* N20. GET NEW CARD
  * UNLOAD HSR BUFFER IF IT HAS NOT ALREADY BEEN
  * UNLOADED. IF NO CARD IS CURRENTLY IN PROCESS,
  * GIVE 2223 ERROR HALT.

* N21. MOVE BUFFERS
  * INITIATE READING NEXT CARD, AND TRANSFER
  * HSR INTERFACE TO PRINTER INTERFACE.
  * PRINT OUT THE CARD IMAGE, TOGETHER WITH
  * LEVEL + BAND.
  * RESET EMITTER AND GO TO N4.
(---IN---)

1134

---SCRAMBLE---

1144

O

(---YES---)

T2. IS STACK EXHAUSTED

YES! EXIT

---NO---

1149

(---NO---)

T3. DOES SYMBOL MATCH

NO! "-

YES! "-

1158

---EXIT---

T4. GET TABLE ENTRY

EXIT

1

T. SYMBOL TABLE SEARCH

THIS SUBROUTINE IS USED TO LOOK UP IDENTIFIER
SPECIAL CHARACTERS, CONSTANTS, AND STATEMENT
NUMBERS (LABELS) IN THE BIG TABLE. IF NOT
IN THE TABLE, THE ITEM IS ENTERED IN.

T1. SCRAMBLE

MULTIPLY ITEM BY 10101010 AND THEN ADD
(0 FOR CONSTANTS, LENGTH FOR IDENTIFIERS,
OR 99 FOR STATEMENT NUMBERS). TAKE THE
RESULT MOD 100; GIVING THE STACK HEAD NUMBER
FOR THIS SYMBOL.

T2. IS STACK EXHAUSTED

IF THIS STACK HAS BEEN ENTIRELY PROCESSED,
INSERT THIS ITEM INTO THE TABLE ON THIS
STACK. EXIT.

T3. DOES SYMBOL MATCH

COMPARE THE CURRENT ITEM IN THE STACK
AGAINST THE DESIRED SYMBOL. IF THERE IS NO
MATCH, GO BACK TO T2.

T4. GET TABLE ENTRY

GET THE CORRESPONDING ENTRY FOR THE SYMBOL WE
HAVE JUST FOUND. EXIT.

CODING DETAILS

AT INPUT, REGISTER A CONTAINS THE LENGTH OF
SYMBOL; REGISTER X CONTAINS THE CODE TO USE
IF NOT FOUND IN THE TABLE, RL CONTAINS THE
EXIT INSTRUCTION, TEMP2 CONTAINS THE SYMBOL.
AT EXIT, RL CONTAINS THE LOCATIONS OF THE
TABLE ENTRY IN ITS M ADDRESS, AND TEMP4 IS
THE EQUIVALENT OF THE SYMBOL.
linked memory subroutines.
these subroutines are used implicitly in many
places of the program, to store and retrieve
information from a pooled memory area.
the format for pooled memory is
stack head: go link good
available stack: go link good
other items are in two word format:
link info1 link 111112222
link+1 info2 11111111
zero link indicates the end, the pool is
kept between locations mem1 and memu.
the symbol table and stacks work down from
mem1, dimensions and equivalence entries
are inserted up from mem1.
in this section, entrance l1 is called 'ins'
and it is for inserting items, while entrance
l10 is for deleting items from stacks and it
is called 'rem'.
l1 is available empty
if the available stack is not empty: remove an
item and go to l4.
l2: mem1 memu
if there is no room for another item, give
the item full error alarm.
l3: reserve two
decrease memu by 2, we will use these two
locations for the new item.
l4: insert item
put the new item into the memory, fix up
links properly, exit.
coding details for ins:
rb1 contains stack head location
rl contains exit instruction
ra contains info2, rx contains info1
at exit, rl is new contents of stack head;
rx is info2.
l10: is stack empty
if stack has no items, go to exit2.
l11: remove item
remove top item of stack
l12: make location available
pull the location just freed onto the available
stack, exit1.
coding details for rem:
rb1 is the stack head location;
rx is the empty exit (exit2);
rl is the ordinary exit.
output: rb1 is the location; rl is info1.
info2 is still in memory.
**C1.** SET TYPE INTEGER
**C1.** INITIALIZE N TO THE NUMBER JUST SCANNED;
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT

---

**C1.** SET TYPE INTEGER
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT

---

**C1.** SET TYPE INTEGER
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT

---

**C1.** SET TYPE INTEGER
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT

---

**C1.** SET TYPE INTEGER
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT

---

**C1.** SET TYPE INTEGER
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT

---

**C1.** SET TYPE INTEGER
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT

---

**C1.** SET TYPE INTEGER
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT

---

**C1.** SET TYPE INTEGER
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT

---

**C1.** SET TYPE INTEGER
**C2.** SET FLOATING TYPE;
**C3.** NEXT CHARACTER
**C4.** WHAT KIND
**C5.** E H OR M
**C6.** ADJUST FOR TYPE
**C7.** IS IT A LABEL
**C8.** LOOK UP IN TABLE
**C10.** NORMALIZE
**C11.** NEXT CHARACTER
**C12.** WHAT KIND
**C13.** NEXT NUMBERS
**C14.** ADJUST EXPONENT
C20: GET N CHARACTERS
GET NEXT N CHARACTERS FROM CARD INCLUDING BLANKS AND BUILD MACHINE CODE CONSTANT. GO TO C32.

C30: GET N CHARACTERS
SET HOLLERITH SWITCH IN ROUTINE N1. THIS SWITCH SIGNALS THAT ROUTINE TO TRANSMIT CHARACTERS IN CARD CODE ON 90-COLUMN SYSTEMS AND ALSO TO SUPPRESS A SPECIAL HIGH-SPEED SKIP OVER BLANK COLUMNS WHICH IT USUALLY HAS. GET THE NEXT N CHARACTERS FROM THE CARD, AND BUILD AN ALPHA CODE CONSTANT IN CARD CODE.

C31: ZERO FILL
IF N IS LESS THAN 5, ADD ZEROES TO FILL THE CONSTANT, RESTORE THE HOLLERITH SWITCH TO NORMAL. IF N IS MORE THAN 5, CRAZY CONSTANTS ARE GENERATED.

C32: TYPE UNSPECIFIED
SET THE TYPE OF THIS CONSTANT TO UNSPECIFIED. GO TO C7.
* Q. SPECIAL SCANNING ROUTINES
*    ENTRANCE TO Q1 OCCURS WHEN THE END OF CARD
*    WHICH IS DETECTED BY AN APOSTROPE INSERTED
*    BY ROUTINE N1 IS SENSED. ENTRANCE Q10 IS
*    USED TO DIVERT NORMAL CONTROL OF SCAN IN
*    ORDER TO EMT A STRING OF INSERTED ITEMS
*    BEFORE RESUMING ORDINARY SCANNING.
* Q1. SEND SEMICOLON
*    SEND SEMICOLON TO GEN ROUTINE, INDICATING
*    END OF THE STATEMENT.
* Q2. END OF DO RANGE
*    IF THE CARD JUST COMPLETED IS THE END OF A
*    DO RANGE, GO TO Q40.
* Q3. ANY LABEL
*    IF COLS 1-5 OF THE NEXT CARD ARE BLANK, TO S1.
*    IF COL 1 IS NUMB SIGN GO TO SPECIAL RESERVED
*    WORD ENTERING PROCEDURE.
* Q4. SCAN FROM COL 1
*    SET TO SCAN THIS CARD AT COLUMN 1 RATHER THAN
*    COLUMN 7, AND SET THE LABEL SWITCH (C7) TO
*    JUMP TO THE CHECKING ROUTINE MENTIONED IN THE
*    COMMENT JUST BEFORE STEP Q40. THEN
*    RETURN TO S1.
* Q10. ADJUST CO-Routine LIKES
*    STORE CURRENT STARTING PLACE FOR SCAN CO-RUTNE
*    IN EXIT OF THIS DIVERT SUBROUTINE. WE WILL
*    COME BACK TO THIS AFTER ALL SPECIAL ITEMS
*    HAVE BEEN INSERT IN THE PSEUDOCODE.
* Q11. NEXT ITEM
*    LOOK AT THE NEXT ITEM TAKEN FROM THE INSERTION TABLE. IF IT IS ZERO, WE ARE DONE
*    INSERTING AND SO WE EXIT TO RE-START THE SCAN
*    CO-Routine.
* Q12. SEND TO GEN
*    SEND ITEM TO GEN. THEN RETURN TO Q11.
* CODING DETAILS: DIVT2 IS USED TO RE-INSERT THE
* PREVIOUSLY SCANNED ITEM AT THE END OF THE
* OTHER INSERTS. DIVT1 IS USED TO RESET GEN
* TO ENTER AT Q1. DIVT IS THE NORMAL ENTRY.
* REGISTER A CONTAINS THE STARTING T-TABLE
* ENTRY. THIS ROUTINE IS ENTERED FROM GEN.
I. ASSEMBLER STRUCTURE

TABLE OF CONTENTS

THIS SECTION IS A COMPLEX OF SUBROUTINES FOR ASSEMBLING THE MACHINE
LANGUAGE INSTRUCTIONS. THE NAMES OF THESE VARIOUS LEVELS AND THEIR
FUNCTIONS ARE

II. ASM1 MACRO ASSEMBLER ... ASSEMBLES 1 TO 5 INSTRUCTIONS AND/OR
PSEUDO-INSTRUCTIONS.

I25. ASM2 ASSEMBLES ENCODED INSTRUCTIONS; FIXING UP THE ADDRESSES OF OPERAND

I30. ASM25 HALF ASSEMBLER LIKE ASM2 EXCEPT IT DEALS WITH ONE ADDRESS M,C ONLY.

I35. ASM28 SPECIAL ASSEMBLER FOR ADDRESSES OF SIMPLE VARIABLES AND CONSTANTS.

I50. ASGN FINDS ADDRESSES OF OPERANDS

I60. LS5 FINDS ADDRESSES OF LABELS

I70. CASEN FINDS ADDRESSES OF CONSTANTS.

I80. ASM3 ASSEMBLES INSTRUCTIONS AND FIXES UP REFERENCES TO NEXT INST.

I90. ASM4 PROCESSES ASSEMBLED INSTRUCTIONS AND LOCATIONS, IN OR OUT OF SEQUENCE, AND PERHAPS LISTS THEM.

I95. ASM5 PUT ONE ITEM ON OUTPUT CARD.
---IN---

1518

I80, IS NXLOC SET

1525

I81, FILL PLEV INST

1529

I82, ASSEMBLER 4

EXIT

* 1. I80, ASSEMBLER 3
   * THIS SUBROUTINE ASSEMBLES ABSOLUTE INSTRUCTIONS AND FIXES UP REFERENCES TO NEXT.
   * A ONE-CYCLE DELAY IS KEPT, AN INSTRUCTION IS NO PUT OUT UNTIL THE NEXT INSTRUCTION COMES ALONG.
   * I80, IS NXLOC SET
   * IF NO PARTICULAR LOCATION FOR THE CURRENT INSTRUCTION HAS BEEN CHOSEN, CHOOSE THE NEXT LOCATION IN THE INTERLACE SEQUENCE.
   * I81, FILL PLEV INST
   * FILL BLANK ADDRESSES IN PREVIOUS INSTRUCTION, IF ANY, WITH THE LOCATION OF THIS ONE.
   * I82, ASSEMBLER 4
   * ACTIVATE ROUTINE I91 TO OUTPUT THE PRECEDING INSTRUCTION, EXIT.
   * CODING DETAILS: RX IS ORROSWOFF WHERE RM ARE RELOCATION DIGITS FOR M AND C, S IS SIGN, AND FF ARE 0 OR 1 FOR NON-BLANK OR BLANK ADDRESS, RESPECTIVELY. RA IS THE INSTRUCTION; RL IS THE EXIT, ASM31-ASM37 ARE SPECIAL ENTRANCES FOR THE MOST COMMON CASES IN SETTING RX.
I. I90. ASSEMBLER 4,

THIS SUBROUTINE PROCESSES ASSEMBLED INSTRUCTIONS AND LOCATIONS. ENTRY I90 IS USED FOR OUT-OF-SEQUENCE LINES; I91 FOR THE PROGRAM SEQUENCE.

I90=SET **********

SAVE COMMENT RESERVED FOR NEXT INSTRUCTION IN PROGRAM SEQUENCE; AND INSERT THE COMMENT **********.

I91=PRINT, MAYBE

IF LIST MODE IS ON, PRINT THE ASSEMBLED LINE AND THE COMMENT.

I92=ASSEMBLER 5

PUT THE CONTROL WORD INTO THE OUTPUT (ROUTINE 195) AND ALSO STORE THE COMMENT FOR THE NEXT INSTRUCTION LINE.

I93=ASSEMBLER 5

PUT THE INSTRUCTION WORD INTO THE OUTPUT (ROUTINE 195). EXIT.

CODING DETAILS:

ASM43, ASM44 PUT REGISTER A AS OUT-OF-SEQUENCE LINE INTO NEXT LOCATION OF UNIQUE STORAGE.

ASM42 PUTS TEMP2 AS OUT-OF-SEQUENCE INTO LOC SPECIFIED BY 7 ADDRESS OF RA, RELOCATION DIGIT FOR M BEING SPECIFIED IN REGISTER L.

ASM41: ASM4 HAVE CONTROL WORD IN REGISTER A,

INSTRUCTION WORD IN REGISTER X.
195. ASSEMBLER 5

THIS SUBROUTINE IS THE SOLE COMMUNICATION BETWEEN THE COMPILED AND THE OUTPUT CARDS.

195. STORE WORD

PUT THE OUTPUT WORD IN THE PUNCH INTERFACE.

196. END OF CARD

IF THE CARD IS NOT FULL YET, EXIT.

197. CHECK CARD

UNLESS NO CARDS MODE IS IN EFFECT, UNLOAD THE BUFFER. THEN THE 2ND READ STATION IS NON-
BLANK, SUM CHECK THE IMAGE AVAILABLE THERE.
GIVE 1112 HALT IF THIS FAILS; AND DUMP MSR BUFFER.

198. COMPUTE CHECK SUM.

COMPUTE SUM OF NUMERIC PORTIONS OF FIRST SEVEN WORDS, AND PLACE IN WORD 4 OF CARD.

199. PUNCH

PUNCH CARD; INCREASE SEQUENCE NUMBER; EXIT.
1. 150. ASSIGN SUBROUTINE
   * THIS SUBROUTINE FINDS; OR MAKES; THE MEMORY
   * ASSIGNMENT FOR SIMPLE VARIABLES, ARRAYS, OR
   * TEMP STORAGES. IT IS NOT A TRUE SUBROUTINE;
   * FOR IF THE ITEM TURNS OUT TO BE A CONSTANT
   * OR HAPPY ARRAY; IT JUMPS INTO THE MIDDLE OF
   * ASM28 ROUTINE.
   * 150. IS IT A TEMP
   * IF THE ITEM TO BE ASSIGNED IS A TEMP STORAGE;
   * GO TO 152.
   * 151. WHAT IS TABLE ENTRY
   * IF THE TABLE ENTRY INDICATES THIS ITEM IS
   * DEFINED IN UNIQUE OR COMMON; GOTO DEFX.
   * IF THE ITEM IS A PARAMETER; GOTO TO THE PARAMETER
   * EXIT. IF THE ITEM IS A CONSTANT; GO TO STEP
   * 138 IN ASM28; OR IF DOING A FUNCTION CALL GO
   * TO CASIN; STEP 170.
   * IF THE ITEM IS UNEQUIVED AND EQUIVALENCED TO
   * OTHER ITEMS; GOTO TO E1.
   * IF THE ITEM IS UNEQUIVED; NOT EQUIVALENCED;
   * ASSIGN IT IN UNIQUE STORAGE AND GO TO DEFX.
   * FINALLY IF THE ITEM IS A HAPPY ARRAY; ASSUME
   * WE HAVE BEEN CALLED BY ASM28 ADJUST OP CODE
   * FOR INDEXING IF NECESSARY; THEN CONVERT TO A
   * SIMPLE VARIABLE AND RECYCLE AT 151.
   * 152. REINSTATE TEMP
   * UNLESS PROCESSING A DO STATEMENT; THE TEMP
   * STORAGE LOCATION IS PUT BACK ON THE LIST OF
   * POTENTIAL TEMP STORAGES FOR FURTHER USE.
   * GO TO DEFX.
   * CODING DETAILS: RA IS THE OPERAND STACK ENTRY;
   * RL IS THE EXIT FOR A PARAMETER; RX IS THE
   * EXIT FOR A DEFINED NON-PARAMETER.
(---IN---)

125, ASSEMBLE 2.5 ON M

126, ASSEMBLE 2.5 ON C

127, ASSEMBLE 3

................. EXIT

I. I25, ASSEMBLER 2.
   THIS SUBROUTINE ASSEMBLES MACHINE LANGUAGE
   INSTRUCTIONS OF AN ALMOST SYMBOLIC NATURE.
   THE OP-CODE IS THE TRUE OP BEFORE INDEXING;
   AND THE ADDRESSES ARE EITHER ABSOLUTE, REFER
   TO NEXT INSTRUCTION, OR REFER TO OPERANDs.
   IN PARTICULAR, AN ARRAY OPERAND IS ALLOWED.
   AND THIS MAY CAUSE MANY INSTRUCTIONS TO BE
   GENERATED, IF THE OPERAND IS NOT A LABEL,
   HOWEVER, THE ASSUMPTION IS MADE THAT IT GOES
   IN M ADDRESS AND THAT C ADDRESS REFERS TO NAT
   I25, ASSEMBLE 2.5 ON M
   SEND THE M ADDRESS TO ASM2.5 FOR ASSEMBLY.
   (IF IT IS AN OPERAND, WE WILL NEVER COME BACK
   FROM ASM2.5, SEE THAT ROUTINE.)
   I26, ASSEMBLE 2.5 ON C
   SEND C ADDRESS TO ASM2.5 FOR ASSEMBLY.
   I27, ASSEMBLE 3
   SEND THE COMPILED INSTRUCTION TO ASM3 FOR
   OUTPUT AND FINAL TOUCHeS. EXIT.

CODING DETAILS: ADDRESS 9999 MEANS NEXT, ADDRESS
9911 MEANS OPERAND STACK + 1, FOR EXAMPLE; 9901 IS THE TOP OF THE OPERAND STACK. ADDRESS
LESS THAN 9901 ARE ABSOLUTE.
AT INPUT RA IS A CODED INSTRUCTION; RL IS EXIT
LINE.
130. ASSEMBLERS 2.5 AND 2.8
ASM2.5 DOES HALF THE JOB OF ASM2.  Q.V.
ASM2.8 IS USED FOR SIMPLE VARIABLES, TEMP
STORAGES, AND SUBSCRIPTS.

130. WHAT ADDRESS
IF THE ADDRESS TO BE ASSEMBLED IS ABSOLUTE,
SET CORRESPONDING 8-DIGIT ZERO AND EXIT.
IF THE ADDRESS REFERS TO NEXT INSTRUCTION,
TRANSMIT THIS INFORMATION AND TEMPORARILY SET
THE ADDRESS ZERO, EXIT.
OTHERWISE THE ADDRESS IS AN OPERAND
AND FURTHER TESTS ARE NECESSARY.

131. WHAT KIND OF OPERAND
FETCH THE OPERAND SPECIFIED AND CHECK TO SEE
WHAT KIND IT IS:
FOR A SIMPLE VARIABLE ON TEMP STORAGE, GO TO
ASM2.8 STEP 135; AFTER WHICH WE EXIT FROM ASM2.
FOR AN ACCUMULATOR SYMBOL THIS IS A BAD MESS,
FOR AN INDEX VARIABLE, ASSUME WE WERE CALLED BY
ASSEMBLER 1 FOR A STORE OPERATION, TRANSFERN
BACK TO ASM1 EMITTING THE INSTRUCTIONS
TO LOAD R1:
FOR AN ARRAY VARIABLE GO TO STEP 144.
FOR A LABEL GO TO THE LABEL ASSIGN ROUTINE
(IO) AND THEN EXIT.

135. ASSIGN VARIABLE
GO TO ROUTINE 130 TO GET THE ASSIGNMENT FOR
THIS SIMPLE VARIABLE.
IF IT IS NOT A PARAMETER GO TO STEP 142
IF IT IS A CONSTANT, WE GET TO STEP 136.
OTHERWISE IT IS A PARAMETER.

136. CHECK UP CODE
FOR SIMPLE VARIABLE PARAMETER, WE CHOOSE
ONE OF THREE SUBROUTINES IN THE OBJECT CODE
DEPENDING WHETHER THE OP IS TO BE LLL, LDA,
OR STL. FOR A STA WE OBTAIN A SNAP AND
OPERATIONS: WE GO LLL OR UP RL.

137. CHECK FOR ZERO
IF THE CONSTANT IS ZERO AND IF THIS IS A
ZERO SUBSCRIPT ON A PARAMETRIC ARRAY GO TO
STEP 146; OTHERWISE FOR A ZERO CONSTANT ADU
ONE TO THE OP CODE AND SET 7 AND C TO NAT,
GO TO ASM3 AND THEN EXIT FOR ASM2.

139. CHECK FOR IIR
SEE IF THE OP IS _DA AND IF IT CAN BE CHANGED
INTO IIR! IF SO, DO THIS AND EXIT
FROM ASM2 VIA ASM3.

140. ASSIGN CONSTANT
USE THE C/SIN ROUTINE (170); DISPLAY THE
COMMENT 'CONST'.

142. ASSEMBLE 3
GIVE APPROPRIATE COMMENT, THEN EXIT FROM
ASM2 VIA ASM3.

144. GET SUBSCRIPT
IN ASM2.5 WE HAVE AN ARRAY OPERAND.
IF THE SUBSCRIPT IS NOT ALREADY IN REGISTER
A, COMPILe COJE TO STORE A IN TEMP IF NECESSARY, AND THEN TO LOAD A WITH THE SUBSCRIPT
(USING ASM2.8; STEP 135).

145. WHAT KIND ARRAY

(REFER TO STEP A24 WHERE THE VARIOUS CASES OF
ARRAY WERE DEFINED.) HAPPY ARRAYS DO NOT
COME THROUGH THIS PART BUT WE MUST BRANCH
3 WAYS FOR THE OTHER TYPES OF ARRAYS:
SAD, GO TO 147;
ORDINARY, GO TO 148;
PARAMETRIC, GO TO NEXT STEP.

146. PARAMETER CODE

IF CORE MODE IS ON, COMPILK
ADD IF; LDX RA, STA PAR, ADD RX, RX.
ELSE COMPILK ADD PAR (OR LDA PAR IF SUBSRIPTION
IS ZERO), ADD NXT RA, GO TO 149.

147. SAD CODE

IF CORE MODE IS ON, COMPILK ADD IF;
LDX RA, IIR RELATIVE, ADD RX, RX.
ELSE COMPILK ADD FUDGE, ADD NXT RA,
GO TO 149.

148. ORDINARY CODE

COMPILK ADD NXT RA,

149. COMPILE OP

NOW COMPILK THE ORIGINAL OP-CODE DESIGNED FOR
THIS ARRAY OPERAND, PLUS 4 IF INDEXING IS
SPECIFIED, PUT NAME OF ARRAY AS COMMENT.
CODING DETAILS WILL BE OMITTED SINCE ASM2.5
AND ASM2.8 ARE ONLY FOR INTERNAL USE BY ASM2.
I. I60, LSW FOR ASSIGNING STATEMENT LABELS.
   THIS ROUTINE HANDLES THE LOGIC FOR LABEL
   ADDRESSES, THE PROBLEMS SOLVED ARE THOSE
   OF FORWARD REFERENCES AND OF POTENTIAL GO TO
   OUT OF  DO LOOPS.

   I60, CHECK LABEL
   CHECK THAT THE OPERAND WHICH IS SUPPOSED TO
   BE A LABEL IS ACTUALLY A STATEMENT NUMBER,
   IF NOT, GIVE THE BAD LABEL ALARM.

   I61, IN  DO LOOP
   IF WE ARE IN A DO LOOP GO TO STEP I64 UNLESS
   WE WANT THE ABSOLUTE LOCATION OF THE LABEL

   I62, ASSIGN
   IF THE LABEL IS UNDEFINED, PICK LOCATION,
   DEFINE IT, AND EXIT. IF THE LABEL IS TEMPORARY
   UNDEFINED (SEE BELOW), GO STEP I62 ON
   THE AUXILIARY WORD. IF THE LABEL IS ALREADY
   DEFINED, SIMPLY EXIT.

   I64, TEMP ASSIGN
   IN DO LOOP (AS OPPOSED TO DONT LOOP) WE MAKE
   A TEMPOARY ASSIGNMENT FOR THE LOCATION TO GO
   TO WHICH STORES RB1 BEFORE GOING TO THE
   ACTUAL LOCATION. THE EXTRA INFORMATION IS
   KEPT IN LLIST, IN THE FORM

   SYMBOL TABLE ENTRY LLLL: 51AAAAAAA
   AAAA: 02TTT
   AAAAA+1: XXSSSSSSSS
   WHERE XXSSSS IS THE OLD SYMBOL TABLE ENTRY,
   TTIT IS THE TEMPORARY ASSIGNMENT.

   IN THIS STEP WE CREATE THE LLIST ENTRY IF
   NONE HAS BEEN MADE YET FOR THIS LABEL. OTHER-
   WISE WE USE THE TEMPORARY ADDRESS, ALSO IF
   THE LABEL HAD NO PERMANENT ADDRESS AND THE
   LABEL HAS NOT OCCURRED IN COLS 1-5, WE SET
   THE PERMANENT ADDRESS EQUAL TO THE TEMPORARY
   ADDRESS. EXIT.

   CODING DETAILS:
   ENTRANCE LSW IS USED FOR THE BRANCH ON DO
   LOOP. ENTRANCE LSW OF IS USED FOR GETTING
   ABSOLUTE LOCATIONS AS WITH A FORMAT OR
   ASSIGN STATEMENT, RL IS THE EXIT LINE.

   OUTPUT IS 02AAAA0000 IN REGISTER A.
I. I70. CASING ASSIGNING CONSTANTS.
I70. ALREADY ASSIGNED
IF THE CONSTANT HAS ALREADY BEEN ASSIGNED,
OUTPUT THE ASSIGNMENT, EXIT.
I71. PICK UNIQUE
PICK THE NEXT LOCATION IN UNIQUE STORAGE FOR
THIS CONSTANT
I72. COMPILE CONSTANT
OUTPUT THE CONSTANT OUT-OF-SEQUENCE USING ASSI
EMBLER 4190; EXIT.
CODING DETAILS, IRX IS EXIT LINE, RB1 IS SYMBOL
TABLE REFERENCE, OUTPUT IS 01AAAA0000 IN RL.
1. **ASSEMBLER 1**
   
   **THIS IS A MACRO-ASSEMBLER**
   **WHICH IS GIVEN A LIST OF TWO-DIGIT INSTRU-**
   **CTION NUMBERS. THESE NUMBERS ARE EITHER**
   **REFERENCES TO A LIST OF STANDARD INSTRUCTIONS**
   **WHICH ARE PROCESSED BY ASSEMBLER 2, OR THEY A**
   **RE REFERENCES TO PSEUDO-INSTRUCTIONS NUMBER**
   **12 THRU 123. THE PSEUDO-INSTRUCTIONS ARE**
   **GIVEN HERE IN THIS SECTION. THE PURPOSE OF**
   **ASM1 IS TO STEP THROUGH ALL 2-DIGIT**
   **CODES AS AN INTERPRETIVE ROUTINE.**
   **LOOP** REPRESENTS THE PLACE TO RETURN TO
   **STEP TO THE NEXT 2-DIGIT CODE.**

2. **CHECK SPECIAL CASES**
   
   **I1; I2; AND I3 ARE USED TO PROVIDE SLIGHTLY**
   **BETTER CODE FOR CERTAIN BINARY OPERATORS**
   **OR FOR IF-STATEMENTS WITH LABELS EQUAL; BY**
   **CHANGING THE ORDER OF OPERATION, LOOP.**

3. **CHECK SUBSCRIPT**
   
   **WHEN A BINARY OPERATION BETWEEN TWO ARRAY**
   **VARIABLES IS USED, A TEST IS MADE HERE TO**
   **SEE WHETHER EITHER SUBSCRIPT IS ALREADY IN**
   **THE ACCUMULATOR; FOR EFFICIENCY, LOOP.**

4. **CLEAR ACC**
   
   **IF THE ACCUMULATOR IN THE OBJECT PROGRAM**
   **IS IN USE; COMPILe THE INSTRUCTION**
   **STA TEMP, , LOOP.**

5. **SET ACC AVAL**
   
   **THE ACCUMULATOR IS SET AVAILABLE, SINCE THE**
   **PREVIOUSLY COMPUTED RESULT IS TO BE USED**
   **NEXT; LOOP.**

6. **TRACE**
   
   **IF TRACE MODE IS NOT ON; EXIT FROM ASM1.**
   **OTHERWISE PREPARE THE INSTRUCTION LDX NAME**
   **PREPARATORY TO TRACING, LOOP.**

7. **OP V2**
   
   **EITHER OP V2 NXT OR**
   **LDS V2; OP RL IS COMPILED,**
   **WE ARE WORKING ON THE BINARY OPERATION**
   **V1 OP V2**

8. **OP V1**
   
   **NOTE: V1 OP V2 EQUALS V2 PO V1**
   **EITHER PO V1 NXT OR**
   **LDS V1; PO RL IS COMPILED, LOOP.**

9. **OP RL**
   
   **THIS PSEUDO OP IS USED TO SELECT ONE OF TWO**
   **ALTERNATIVES, WHICH ARE GIVEN AS PARAMETERS**
   **TO ASM1, DEPENDING ON WHICH OPERAND IS IN**
   **RL AND WHICH IS IN RA. LOOP.**

10. **LDS X**
    
    **COMPILe OP RL NXT, LOOP.**

11. **LDR**
    
    **THIS COMPILES THE LINKAGE TO SUBROUTINES,**
    **INCLUDING THE CONTROL INFORMATION TO BRING A**
    **NEW SUBROUTINE IF THE SUBROUTINE HAS NOT**
4291
I12. OP RL NXT

4293
I13. LIR3 NXT SUS

2311
I14. STORE INTO R81

2318
I15. ATL CONDITIONALLY

2323
I16. SHFT

2327
I18. UNARY OPERATOR

4336
I19. GO TO 3F; 21

4344
I20. TGR 9F 3F

2355
I21. NINEF DO
8. **ARITHMETIC OPERATORS**

   * This section contains the generators for
   * arithmetic operators, entered from step 60
   * on from step G10. An odd numbered step here
   * indicates an entry from step 6 (when symbol
   * is first sensed) and an even numbered step
   * in this section indicates an entry from
   * G10 (off the operator stack).

   **8.1.** **MINUS SIGN**
   * Check if the preceding item is an operand or
   * right parenthesis. If so, a binary minus
   * operator is substituted and we go to step G7.
   * If not, a unary minus operator is substituted
   * and we go to step G20.

   **8.2.** **PLUS SIGN**
   * Check as in step 8.1 for unary or binary.
   * On binary plus change to the binary add
   * operator and go to step G7.
   * A unary plus is ignored. Go to G1.

   **8.3.** **SUBTRACTION OP**
   * Check if the operator in analyst is a right
   * parenthesis and if the top of the operator
   * stack is left parenthesis after an IF. In
   * this case and if the signs of the top two
   * operands are equal, subtraction is not
   * carried out. If operator is removed
   * from the stack and we go to step B77.
   * Otherwise negate the top operand and
   * change to binary plus, step B6.

   **8.4.** **NEGATION OP**
   * Change sign of top operand, exit to G10.

   **8.5.** **ADDITION OP**
   * Check types of operands. If they are mixed
   * give an error alarm.
   * If both are floating point, go to 890.
   * If fixed point: check if we are adding a
   * constant in an array subscript. If not,
   * go to B99. However, if we are adding 0 + V
   * the addition is suppressed.
   * In the array subscript case: record if the
   * constant is greater than +1. Multiply the
   * constant by the appropriate dimension
   * and add this to the base. Exit to G10.

   **8.6.** **ASTERISK**
   * Check for second asterisk and change to a
   * multiply or power operator. Go to G7.

   **8.7.** **MULTIPLY**
   * If float-float, go to B90.
   * If mix-type, give error alarm.
   * If fix-fix: set 0 = V equal to 0. In other
   * cases, we may check for possibility of
   * adding 1 and/or shift commands to implement
   * multiplication; else go to B90.

   **8.8.** **LEFT PARENTHESIS**
   * Put a left parenthesis operator on the stack.
   * Step G20. When it comes off the stack it will
   * necessarily be forced off by its matching
RIGHT PARENTHESES AND IN THIS CASE
WE WILL SIMPLY GO TO STEP G1.

916: DIVISION
GIVE ERROR ALARM IF MIXED TYPE, OTHERWISE
GO TO B90.

920: EXponentiation
GIVE ERROR ALARM IF FIX**FLOAT
IF RAISING TO THE SECOND POWER, GO TO THE
UNARY SQUAREING OPERATOR STEP U10.
OTHERWISE GO TO B90.

922: AND OR
FOR BOOLEAN AND OR WE SET THE TYPE OF THE
RESULT TO UNSPECIFIED, THEN GO TO B69.

925: WORD IF
CHECK THAT A LEFT PARENTHESES FOLLOWS; ELSE
GIVE AN ERROR ALARM, PUT A SPECIAL IF-LEFT-
PARENTHESES ON THE STACK AT STEP G20.

926: IF-LEFT-PARENTHESES
AT THIS POINT WE HAVE PROCESSED THE EXPRESSION
IN AN IF-STATEMENT AND MUST COMPARE IT
AGAINST ZERO. THEREFORE THE CONSTANT ZERO IS
PUT ON TOP OF THE OPERAND STACK.

927: FINISH IF-STATEMENT
SET UP IF MODE, THEN PROCESS THE STATEMENT
NUMBERS, CHECK THAT THERE ARE EXACTLY THREE,
THEN CHOOSE THE BEST CODING SEQUENCE BASED
ON EQUALITIES BETWEEN THESE.

989: GENERATE MACHINE OP
GENERATE CODING FOR THE MACHINE OPS
ADD, SUB, OR BUF, USING ASSEMBLER
1 (ROUTINE I), AND USING ONE OF 16 TABLE
ENTRIES DEPENDING ON WHETHER THE OPERANDS
ARE
0 SIMPLE VARIABLES, ETC.
1 IN THE ACCUMULATOR
2 INDEX REGISTER 1
3 ARRAY VARIABLES

EXIT TO G10.

990: GENERATE LIBRARY REF
GENERATE A REFERENCE TO A BINARY LIBRARY
SUBROUTINE. THERE ARE 6 CASES DEPENDING
ON WHETHER EITHER OPERAND IS NEGATED, AND
DEPENDING WHICH OPERAND WAS MOST CONVENIENT
TO PLACE IN REGISTER L. THESE CASES ARE
SELECTED BY REFERENCING A TABLE ENTRY
AS IN STEP 889 AND GOING TO ASSEMBLER 1
(ROUTINE I). EXIT TO G10.
A. PROCESSING OF ARRAY SUBSCRIPTS

WHEN A DIMENSIONED VARIABLE IS SENT FROM THE
SCANNER, ENTRY IS MADE TO A1. A COMMA
BETWEEN SUBSCRIPTS CAUSES ENTRY TO A10.

A1. IS LEFT PAREN NEXT
SCAN NEXT ITEM, IF IT IS NOT A LEFT
PARENTHESIS, GO TO THE UNDIMENSIONED ARRAY
SWITCH. THIS SWITCH IS NORMALLY SET TO THE
MISSING LEFT PARENTHESIS ALARM WHICH
INSERTS A LEFT PARENTHESIS INTO THE
STATEMENT AND RETURNS HERE.

A2. SET ARRAY MODE
THE MODE STACK RECEIVES FOUR NEW ENTRIES:
2 0000 ARRAY MODE (A10 FOR COMMA);
MISSING RIGHT PAREN FOR $)
2 9999 BASE CALCULATION
2 9998 CURRENT PRODUCT OF DIMENSIONS
2 9997 REFERENCE TO DIMENSION LIST

A3. EMIT X O +
FOR CONVENIENCE, THE CHARACTER ( O + ARE
INSERTED. THIS LEFT PARENTHESIS IS A SPECIAL
ONE WHICH SENDS CONTROL TO STEP A20 WHEN
THE MATCHING RIGHT PARENTHESIS COMES ALONG.
EXIT TO G1.

A10. CHECK INDEX
IF THIS IS THE FIRST SUBSCRIPT AND ITS
CURRENT VALUE IS RB; CODE, INDEXING IS SET
UP AND THE SUBSCRIPT IS REPLACED BY ZER0.
THIS OCCURS ONLY IF THE FIRST SUBSCRIPT IS
DOVAR & CONSTANT, WHERE THE CONSTANT IS
GREATER THAN -30, AND IF WE ARE NOT CALLING
A FUNCTION.

THE IMPORTANT ASSUMPTION MADE HERE THAT
NEITHER UNIQUE NOR COMMON STORE IS TO BE
ASSIGNED TO CORE LOCATIONS 2000 - 2029.
WITH THIS CONVENTION, THE NUMBER OF SAD
ARRAYS (SEE SECTION A24) IS GREATLY REDUCED.

A11. POTENTIAL NEGATIVITY
IF ANY CONSTANTS GREATER THAN 1 OCCURRED
DURING THE LAST SUBSCRIPT ALONG WITH
ANYTHING OTHER THAN DOVAR, THIS ARRAY IS
MARKED AS HAVING A POTENTIALLY NEGATIVE
SUBSCRIPT.

A12. ADJUST MULTIPLIER
IF THERE ARE NO MORE DIMENSIONS, THE EXTRA
SUBSCRIPT ALARM IS GIVEN; ELSE IT IS
MULTIPLIED TO GIVE THE CURRENT PRODUCT OF
DIMENSIONS.

A13. EMIT X O +
FOR CONVENIENCE, THE COMMA IS TRANSFORMED
INTO THE CHARACTERS +0+; THIS LEFT
PARENTHESIS IS LIKE A MULTIPLICATION SYMBOL;
ONLY THE CHECK AT STEP A11 IS MADE FIRST.

A20. INDEXING NEGATIVITY
WE HAVE NOW SCANNED THE ENTIRE SUBSCRIPT
OF THE ARRAY. STEPS A10 AND A11 ARE PERFORMED
THEY ARE REDUNDANT UNLESS THE ARRAY IS
SINGLY SUBSCRIPTED.

A21. CHECK FIXED POINT
IF SUBSCRIPT IS FLOATING, GIVE ERROR ALARM.

A22. COMPUTE SUBSCRIPT
ITSELF; PRODUCE CODE TO LOAD IT WITH TRUE
SIGN INTO THE ACCUMULATOR.

A23. EQUIVALENCE DECL.
IF WE ARE IN AN EQUIVALENCE DECLARATION,
EXIT TO THE EQUIVALENCE ROUTINE E30.

A24. WHAT TYPE ARRAY
THERE ARE FOUR KINDS OF ARRAYS, AND WE DECIDE
NOW WHAT KIND THIS IS.
IF THE ARRAY IS PARAMETRIC, GO TO A27.
CONSTANTS IN THE SUBSCRIPT ARE ADDED TO THE
BASE, IF THE BASE HAS THEREBY BECOME NEGATIVE
OR TOO LARGE OR IF THE SUBSCRIPT IS
POSSIBLY NEGATIVE, THIS IS CALLED A SAD
ARRAY, AND WE GO TO A26.
EXAMPLE1: A(1+2), WHERE J MAY BE NEGATIVE
IF THE SUBSCRIPT IS NOT ZERO. THIS INDICATES
THAT IT WAS ALL CONSTANT EXCEPT PERHAPS FOR
INDEX REGISTER MODIFICATION, SO GO TO A28.
ELSE IT IS AN ORDINARY ARRAY.

A25. CODE 3SLLLL1000
FOR AN ORDINARY ARRAY, THE CODE 3SLLLL1000
IS SET UP WHERE S IS THE STORAGE TYPE;
LLLL IS THE BASE LOCATION; AND I IS 0 OR 4
FOR INDEXING, TO A29.

A26. CODE 2BBBBS000
FOR A SAD ARRAY, THE CODE 2BBBBS000 IS ESET
UP WHERE S IS THE STORAGE TYPE; BBB IS
THE BASE LOCATION PLUS 50000; PLUS 40000 IF
INDEXING, TO A29.

A27. CODE 1BBBBPPP
FOR A PARAMETRIC ARRAY THE CODE 1BBBBPPP
IS SET UP WHERE BBBPPP IS THE BASE LOCATION
PLUS 50000; PLUS 40000 IF INDEXING; AND
PPPP IS THE LOCATION OF THE PARAMETER, TO A29.

A28. CODE AS SIMPLE VAR.
THIS ARRAY IS CHANGED TO LOOK ALMOST LIKE
A SIMPLE VARIABLE.

A29. MOVE SUBSCRIPT
THE STATUS OF THINGS IS CHANGED TO:
SAD: PART: HAP: ORD:
OPERAND STACK ENTRY:
3TAAAA0000 3TAAAA0000 UTAAAA0000 3TAAAA0000
LOCATIONS AAAA AND AAAA+11
00000000000000000000000000000000
00000000000000000000000000000000
2BBBBS000 1BBBBPPP 000000000000000000000000
LOCATIONS CCC and CCC+11
SUBSCRIPT SUBSCRIPT 05LLLLL*** SUBSCRIPT
ZZZZZNNNNN ZZZZNNNNN ZZZZNNNNN ZZZZNNNNN
HERE ZZZZNNNNN IS THE ARRAY NAME, AND
U. UNARY OPERATORS AND SPECIAL GENERATORS

* COMPARE WITH THE INTRODUCTORY REMARKS OF SECTION B; ODD-NUMBERED STEPS INDICATE ENTRY FROM G6; EVEN NUMBERED FROM G10.

U1. EQUALS SIGN

* THIS IS A SWITCH WHICH IS SET IN SEVERAL PLACES, IF THIS EQUALS SIGN OCCURS IN A DO STATEMENT GO TO D3, IF IT IS IN AN INPUT-OUTPUT STATEMENT GO TO D17, OTHERWISE THIS IS A PLAIN OLD EQUALS SIGN AND WE PUT A REPLACEMENT OPERATOR ON THE STACK, G20.

U2. REPLACEMENT SETUP

* IN A MULTIPLE ASSIGNMENT STATEMENT WE ENTER AT STEP U2 THE FIRST REPLACEMENT OPERATOR, STEP U4 SUCCEEDING TIMES, CHECK TYPES, AND IF DIFFERENCE IS PRESENT PUT OUT THE CODE TO FIX OR FLOAT, IF THE TYPES ARE THE SAME, DECIDE WHETHER TO PUT THE RIGHT-HAND SIDE IN REGISTER A OR NOT, REGISTER L IS SELECTED IF THERE IS A MULTIPLE ASSIGNMENT STATEMENT, OR IF THE LEFT-HAND PART IS NOT A SIMPLE VARIABLE OR IF TRACE MODE IS ON.

U4. REPLACEMENT OPERATOR

* THE CODING TO PUT THE RIGHT-HAND SIDE WITH TRUE SIGN INTO THE SELECTED REGISTER IS ACCOMPLISHED BY SELECTING A TABLE ENTRY AND ACTIVATING ASM1 (ROUTINE I).

U10. UNARY OPERATORS

* IN THE CASE OF FIX, EXP, SIN, COS, TAN, ATAN, LN, SORT, CHECK THAT THE ARGUMENT IS FLOATING POINT, SQUAREING THE NOT OPERATOR, AND FLOAT PLUS THE ONES MENTIONED EARLIER ARE CALLED FROM THE LIBRARY SUBROUTINES, USING A TABLE ENTRY AND ACTIVATING ASSEMBLER 1. THERE ARE TWO CASES, DEPENDING WHETHER THE ARGUMENT IS NEGATED OR NOT. IN THE CASE OF ABS, A SPECIAL TABLE ENTRY FOR AN OPEN SUBROUTINE IS USED.

U12. END OF STATEMENT

* AT THE END OF MOST STATEMENTS WE CHECK THAT THE OPERATOR AND OPERAND STACKS ARE EMPTY, ELSE GIVE THE ERROR *MISSING RIGHT PARENTHESIS OR *MISSING OPERAND OR *EXTRA OPERAND.

U13. WORD 'GO'

* SET LABEL CONTEXT ON AND SCAN THE NEXT ITEM (ROUTINE S), THE WORD TO IS IGNORED BY FORTAN, IF THE NEXT ITEM IS A LABEL, PUT IT IN A BLANK ADDRESS OF THE PRECEDING INSTRUCTION OR ELSE CREATE A JMP INSTRUCTION, THEN GO TO G1.

IF THE NEXT ITEM IS A VARIABLE, COMPIL CODE TO STORE RBI IF WE ARE IN A DO LOOP, THEN
1. U21. "DIMENSION"

2. U27. "COMMON"

3. U29. "CONTROL WORDS"

4. CODE TO JUMP TO THE VARIABLE ITSELF. U21

5. FINALLY IF IT IS A LEFT PARENTHESIS, WE SET UP GO MODE, COMPILE EACH LABEL OUT OF SEQUENCE, THEN WHEN THE RIGHT PARENTHESIS COMES ALONG WE RETURN TO G1 TO PROCESS THE EXPRESSION.

6. U14. END COMPUTED GO.

7. COMPILE CODE TO GET THE EXPRESSION WITH TRUE SIGN IS REGISTER A THEN ADD NEXT RA, JMP TO THE TABLE.

8. U17. "WORD 'ASSIGN'"

9. SET LABEL CONTEXT AND PLACE THE ASSIGN OPERATOR ON THE STACK. THE WORD 'TO' IS IGNORED BY FORTRAN.

10. U18. "ASSIGN OP"

11. CREATE A CONSTANT FOR THE ABSOLUTE LOCATION OF THE LABEL (USE 162); THEN INTERCHANGE OPERANDS AND TREAT ANALOGOUS TO REPLACEMENT AT STEP U2.


13. WHEN A DIMENSION DECLARATION APPEARS THE REST OF THE COMPILER IS RIGGED UP TO HANDLE THIS STATEMENT PROPERLY BY SETTING UP DIMENSION MODE, WHEN A NAME COMES ALONG, A SECOND MODE IS SET UP, AND THIS MODE CREATES THE TABLE ENTRIES FOR AN ARRAY VARIABLE, AT THE END: EXIT TO G1. NO STORAGE ASSIGNMENTS ARE MADE YET; THEY ARE MADE WHEN THE ARRAY IS FIRST REFERENCED.

14. U27. "COMMON"

15. SET UP COMMON MODE, MARK EACH IDENTIFIER THAT COMES ALONG AS COMMON AND ALLOCATE THE STORAGE FOR IT.

16. U29. "CONTROL WORDS"

17. THE WORDS NO TRACE LISTCORE CARDS REALLY NEVER GET PAST THE SCANNER; THEY ARE DETECTED AT STEP 510, THEY MERELY SET INTERNAL SWITCHES INSIDE THE COMPILER, AND RUN OFF TO G1.
0. DO LOOP CONTROL
WHEN THE WORD DO OR THROUGH IS SENSED, ENTRY
IS MADE TO STEP D1.

D1. SET UP FOR LABEL
DO MORE IS SET UP, A SWITCH IS SET SO THAT
WHEN THE NEXT EQUAL SIGN OCCURS, CONTROL GOES
TO STEP D3. SEMI-LABEL CONTEXT IS SET UP
SO THAT THE LABEL FOLLOWING COMES IN AS A
CONSTANT, YET STEP C5 GOES IMMEDIATELY TO C6
IN THE CONSTANT SCANNER. GO TO G1.

D3. ZERO COMMA COUNT
THE FACT THAT A COMMA MAY HAVE OCCURRED
BEFORE THE CONTROLLED VARIABLE IS FORGOTTEN, AT
THE END OF THIS STATEMENT, CONTROL WILL
PASS TO STEP D5. GO TO G1.

D5. CHECK COMMAS
IF LESS THAN TWO COMMAS HAVE OCCURRED,
INSERT "+1" IN THE PSEUDOCODE.

D6. STORE EXP IN TEMP
STORE EXP TO STORE REGISTER A IF THEME IS
A COMPUTED RESULT THERE, SET A SWITCH SO THAT
THE TEMP STORAGE USED TO HOLD COMPUTED
RESULTS ARE MADE PERMANENT STORAGE.
(SEE STEP 19Z).

D7. DO OR DONT
THIS IS A DONT LOOP UNLESS:
1) THE WORD THROUGH WAS NOT USED
2) NO DO IS IN PROGRESS
3) BOTH THE STARTING VALUE AND INCREMENT
ARE CONSTANTS.
IN CASE OF A DONT LOOP, GO TO STEP D10.

D8. BEGIN DOO
SET THINGS UP FOR PUTTING VARIABLE IN AN
INDEX REGISTER, SET SWITCH FOR SPECIAL
HANDLING OF LABELS, COMPIL EXI N 3F;
2 1I116 1, LDL V, TGR 9F. GO TO STEP D40.

D10. LDA INIT 3F
COMPIL LDA WITH INITIAL VALUE.

D11. V + INC
ARTIFICIALLY INSERT ++ INTO THE PROGRAM
THUS RUNNING THROUGH THE ORDINARY ADD
GENERATOR TO CREATE CODE TO PUT THE SUM OF
V + INC IN REGISTER A.

D12. LRL, TGR
COMPLIE 3 LRL FIN, TGR 9F, STA V
D20. LABEL IN TABLE
PUT THE LABEL NUMBER TOGETHER WITH THE PER-
TINENT ADDRESSES FOR LINKING UP CONTROL
(9F129) INTO THE DO STACK; EXIT TO D12.
F. FUNCTION CALLS
   TRANSFER IS MADE TO STEP F1 IF WE HAVE AN
   UNDIMENSIONED IDENTIFIER FOLLOWED BY A LEFT
   PARENTHESIS, NOT OCCURRING IN A DIMENSION DEC.
   F1. ASSIGN F
   IF THIS IS A NEW FUNCTION DEFINE IT; IF IT IS
   A CONSTANT OR SIMPLE VARIABLE, TREAT AS
   IMPLIED MULTIPLICATION.
   F2. SET FUNC MODE
   SET UP FUNCTION MODE AND ALSO PUT A SPECIAL
   LEFT PARENTHESIS OPERATOR ON THE STACK,
   AS WE PASS OVER THE LIST OF PARAMETERS;
   CODE IS COMPILED TO COMPUTE THEM AND STORE
   THEM IN TEMP; IF THE PARAMETER IS A CONSTANT
   OR INDEX REGISTER, AS THE RIGHT PARENTHESIS
   CLOSING THE FUNCTION CALL OCCURS, TRANSFER
   WILL GO TO STEP F4. GO NOW TO STEP G1.
F4. BEGIN REVERSE PASS
   BEGIN NOW A RIGHT-TO-LEFT PASS OVER THE
   PARAMETERS, RESERVE THE UNIQUE STORAGE FOR
   THEM, THEN PROCESS EACH PARAMETER IN TURN.
   THE TYPES OF CODE PRODUCED ARE:
   FOR SIMPLE VARIABLE PARAMETER-PARAMETER
   LIR $5555, URS PARAM; STA LIST
   AND LIST IS MARKED AS TEMP STORAGE.
   FOR A LABEL (I-O SUBROUTINES ONLY); CODE
   00 LLLL 0000 (OUT OF SEQUENCE).
   FOR AN ARRAY; LIR A0; STA LIST,
   FOR A SIMPLE VARIABLE OR TEMP STORAGE;
   00 LLLL 0000 (OUT-OF-SEQUENCE).
F5. LIR3
   AFTER ALL PARAMETERS HAVE BEEN PROCESSED,
   COMPARE THE ADDRESS
   LIR3 $5555 AND THE
   NEXT INSTRUCTION GOES TO LOCATION $5555.
   THE PARAMETERS HAVE BEEN LISTED IN $5555,
   $5557, ETC.
   IF THIS CALL IS NOT IN A CALL STATEMENT, TREAT
   THE RESULT AS A COMPUTED QUANTITY IN
   REGISTER A. GO TO G1.
   NOTE: IF A CALL STATEMENT IS GIVEN WITH
   NO PARAMETERS, NO REFERENCE TO UNIQUE
   STORAGE IS MADE.
* D40. CLOSE OF DO RANGE.
* AS EACH STATEMENT LABEL IS SCANNED IT IS
* CHECKED AGAINST THE TOP OF THE DO STACK
* TO SEE WHETHER THIS STATEMENT IS THE END OF
* THE DO RANGE. IF IT IS, THE NEXT APOSTROPHE
* OPERATOR (END OF STATEMENT) SENDS CONTROL
* TO STEP D40.*
* D40. GO TO 28
* EFFECTIVELY COMPILE GO TO THE INCREMENTATION
* PHASE AT THE BEGINNING OF THE DO LOOP CODING;
* AND SET THE NEXT INSTRUCTION LOCATION TO BE
* 9F; THE ADDRESS FOR EXHAUSTION OF THE DO.
* D41.DO OR DONT
* IF THE LOOP JUST ENDED WAS A DONT LOOP;
* SKIP TO STEP D50.*
* D42.EMPTY LLIST
* TURN OFF THE VARIOUS INDICATORS WHICH ARE
* SET DIFFERENTLY WHILE WE ARE IN A DO LOOP.
* THEN FOR ALL LABELS WHICH WERE GIVEN
* TEMPORARY ASSIGNMENTS; WE HAVE AN LLIST
* ENTRY AND WE NOW OUTPUT THE INSTRUCTIONS
* T I1R10
* WHERE V IS THE DO VARIABLE; T IS THE TEM- 
* PORARY ASSIGNMENT; P IS THE PERMANENT 
* ASSIGNMENT. THE TEMP ASSIGNMENT IS THEN
* FORGOTTEN.
* D50. ANY MORE
* IF ANOTHER DO LOOP ENDS ON THIS 
* STATEMENT, RETURN TO STEP D40. ELSE
* GO TO Q3.
**X1. PROCESSING FORMAT STRING**

* X1a. COMPIL 02
  * THE INSTRUCTION 02 MMMM CCCE IS COMPILED
  * WHERE MMMM IS THE STARTING LOCATION OF THE
  * FORMAT CODE. WITH THIS TRICK, A FORMAT LABEL
  * IS LIKE ANY STATEMENT LABEL.
  * NOW WE TRANSLATE THE FORMAT INTO A SPECIAL
  * PSEUDOCODE. THIS CODE GENERATES INSTRUCTIONS
  * OF THE FORM OP NWW DD, CORRESPONDING
  * TO FORMAT SPECIFICATION 'NNN EWWWDD'.
  * OP-CODES 0-10 CORRESPOND RESPECTIVELY TO
  * (PIEFXAMH/)

X2. RESET OP NWW0
    CLEAR OP, N, W, AND D TO ZERO

X3. NEXT CHARACTER
    GET THE NEXT CHARACTER FROM THE FORMAT LIST.
    IF IT IS BLANK, DO X3 AGAIN.
    IF IT IS A DECIMAL POINT, CYCLE NWW LEFT 1
    AND RETURN TO X3.
    IF IT IS A DECIMAL POINT, CYCLE NWW LEFT 1
    AND RETURN TO X3.
    IF IT IS NUMERIC, SET D TO 100D PLUS CHAR X3
    AND RETURN TO X3.
    IF IT IS ALPHABETIC OR SPECIAL CHARACTER,
    LOOK UP IN A TABLE TO SEE WHAT TO DO.
    AN E/F I A OR M MEANS GO TO X4.
    A PLUS OR MINUS MEANS GO TO X5.
    AN X OR P MEANS GO TO X6.
    A LEFT PARENTHESIS MEANS GO TO X7
    A COMMA SLASH AND RIGHT PARENTHESIS MEAN TO X8.
    THE LETTER M MEANS GO TO X9.
    AN APOSTROPE MEANS WE GO TO XI.

X4. SET OP CYCLE
    SET OP TO THE APPROPRIATE NUMBER, AND CYCLE
    NWW AND D LEFT 1. RETURN TO X3.

X5. SET SIGN INTO W
    SET W TO 0 OR 1 (PLUS OR MINUS), RETURN TO X3.

X6. ASSEMBLE THIS OP
    MOVE D TO N: THEN ASSEMBLE
    OP-NWWDD INTO THE FORMAT CODE. RETURN TO X2.

X7. ASSEMBLE 2 LINES
    "MOVE D TO N AND ASSEMBLE," THEN INSERT A WORD
    OF ZEROS INTO THE FORMAT CODE. THIS WORD
    IS USED AS A SCRATCH PAD BY THE FORMAT
    PROCESSING PACKAGE. RETURN TO X2.

X8. ASSEMBLE TWO OPS
    IF DECIMAL POINT HAS NOT APPEARED; CYCLE
    NWW LEFT 1. IF PREVIOUS OP IS WAITING
    ASSEMBLE IT, AND CLEAR W. IF CURRENT IS NOT
    A COMMA, ASSEMBLE IT TOO.
    NOTE THAT ON I/ THE COUNT N COMES OUT IN W.

X9. ASSEMBLE M OP
    MOVE D TO N AND ASSEMBLE.

X10. INSERT LITERAL
    OUTPUT 5 CHARACTERS OF THE LITERAL AT A TIME
    UNTIL THE M LITERAL IS COMPLETED.
    THE ROUTINE FOR M LITERALS IN THE CONSTANT
    ENCODER IS USED, WITH ZERO FILL AT THE
    RIGHT. RETURN TO X2.

X11. ASSEMBLE 99 OP
THE Apostrophe signals the end of the statement. Assemble a termination line and go out.
**INPUT-OUTPUT (READ, PUNCH, PRINT)**

This section is without doubt the climax of the compiler. At least 95% of the coding of this compiler program can be active while processing a single I/O list.

**1. SET TWO OPERANDS**

Set up two operands, one for the editing subroutine and one for the driver subroutine (I/O device). Set up to expect a label: go to step 3. We will return to step 2 when the comma is sensed.

**2. CALL FUNCTION**

Use the function call routine (ROUTINE F) to create initial entry to the I/O subroutine. Then set up I/O mode. If an unidimentioned array variable occurs we will go to step 3. On a comma we go to step 8.

At the end of the statement, we go to step 7. Now we go to the comma routine: step 10.

**3. UNIDIM ARRAY**

An unidimentioned array A is converted into A[1:N], where *N* is a dummy variable and *N* is the product of the array dimensions.

**4. SCAN FOR**

Scan next item (CO-routine I), if it is the end of the statement, go to step 5. If it is a left parenthesis: go to step 12. Otherwise insert an IN or OUT operator on the stack then go to 2. In is step 14; out is step 15.

**5. INTERRUPT SEQUENCE**

Create a break in the instruction sequence, for which code will be inserted later. Put a special left parenthesis on the stack. This special left parenthesis is step 20. Go to step 10 again.

**6. IN**

Compile LIR3 SUB, STL V; go to step 10.

**7. OUT**

Compile LOA V; LIR3 SUB; go to step 10. SUB is one of three entries; depending on the type (FLOAT, FIX, UNSPECIFIED) of V.

**8. EQUALS SIGN**

An equals sign has appeared; so we pull the spurious IN or OUT operator off the stack. We now courageously jump into the middle through routine: step 11.

**9. (LIST)**

The right parenthesis matching a left brace has been encountered. If an implied DO loop occurred inside, we use parts of routine D to create coding for the count loop control. Finally the interruptions from step 12 are all linked together properly. Go to step 1.
3669

#50.END

COMPILE LIR3
AND THEN EXIT.

SUB, THE ENDING SUBROUTINE,
(---IN---)

3680

--- P1, COMPILe PREAMBLE ---

3716

--- P2, SET UP CARD ---

3734

--- P3, SCAN PARAMETERS ---

3739

(--- SCAN AHEAD ) DIM1

3743

--- P5, GENERATE THUNKS ---

G2

--- P6, FUNCTION AND SUBROUTINE DECLARATIONS ---

P1. COMPILe PREAMBLE

COMPILe IIR1 0000; BUF 1F LIR1 0000,
(AND LDA 0001; ATL 1F FUNCTION)

P2. SET UP CARD

SET UP THE NAME OF THE FUNCTION INTO THE
OUTPUT CARDS; INITIAlIZE OTHER THINGS LIKE
THE MEANING OF RETURN; A MAIN PROGRAM
IS DISTINGUISHED FROM A SUBPROGRAM
ONLY BY DEFAULT.

P3. SCAN PARAMETERS

SCAN UNTIL THE END OF THE STATEMENT,
COLLECTING ALL PARAMETERS ON THE OPERAND
STACK, WE GET TO STEP P4 AT THE END
OF THE STATEMENT.

P4. SCAN AHEAD

IF THE NEXT ITEM SCANNED IS ANOTHER END OF
STATEMENT OR DIMENSION, GO TO G2.

P5. GENERATE THUNKS

ELSE WE ASSUME ALL DIMENSIONED PARAMETERS
HAVE BEEN NAMED, AND WE COMPILE CODE TO TRANS-
FER FROM THE PARAMETER LIST TO UNIQUE STORAGE
TEN INSTRUCTIONS FOR NON-DIMENSIONED
PARAMETERS AND TWO FOR DIMENSIONED ONES,
THEN OFF TO G2.
(*IN*)

Z1. SET UP HEADER TABLE

Z2. CLEAR SYMBOL TABLE

Z3. INITIALIZE COUNTERS

Z50. END IS SENSED

Z51. PUNCH HEADERS

Z52. READ NEXT CARD

Z71. INITIALIZE AND TERMINATION
Z1 IS ENTERED AT THE BEGINNING OF EACH
PROGRAM AND SUBPROGRAM.
Z1. SET UP HEADER TABLE
THE HEADER CARD INFORMATION IS KEPT IN A 50-
POSITION CIRCULAR TABLE. IF MORE THAN 50
TOTAL ITEMS ARE PUT IN, A FLAG IS SET
SO THAT LOAD-AND-GO OPERATION IS DISALLOWED.
Z2. CLEAR SYMBOL TABLE
ALL SYMBOLS EXCEPT RESERVED WORDS ARE
REMOVED FROM THE SYMBOL TABLE.
Z3. INITIALIZE COUNTERS
VARIOUS THINGS ARE RESET E.G. SUBROUTINE
PACKAGE REQUESTS, STORAGE ALLOCATION REQUESTS
COUNTERS ARE SET UP TO INDICATE A MAIN
PROGRAM THESE WILL BE EFFECTIVE UNLESS
A FUNCTION OR SUBROUTINE DECLARATION FOLLOWS.
START COMPILING BY TROTTHING FORTH TO G1.
Z50. END IS SENSED
AN END CARD MEANS WE SIMULATE A RETURN
STATEMENT (I.E., GO TO EXIT).
Z51. PUNCH HEADERS
PUNCH AND PRINT HEADER INFORMATION.
Z52. READ NEXT CARD
IF NO MORE INPUT CARDS ARE IN THE BUFFER,
PUNCH OUT SEVERAL BLANK CARDS AND STOP.
IF THE NEXT CARD IS THE BEGINNING OF PASS2,
TRANSFER TO THE SECOND PASS UNLESS AN ERROR
OCCURRED IN THE PRECEDING PROGRAMS.
OTHERWISE WE GO TO Z1 TO PROCESS ANOTHER
* E1. EQUIVALENCE DECLARATIONS.
* IT IS ALMOST IMPOSSIBLE TO EXPLAIN HOW THE
* PROCESSING OF EQUIVALENCE DECLARATIONS
* WORKS IN THIS COMPILER.
* EQUIVALENCE CLASSES ARE KEPT
* IN CIRCULARLY-LINKED CHAINS. IT IS EASY TO
* MERGE TWO CHAINS INTO ONE, WHEN AN ITEM
* OF A CHAIN IS FIRST REFERENCED AFTER AN
* EQUIVALENCE DECLARATION, WE GO TO E1. FORMATS
* OF THE CHAIN ENTRIES APPEAR IN THE TABLE
* OF FORMATS.
* E1. SEARCH THROUGH CHAIN
* TRAVERSE THE CHAIN ONCE TO SEE HOW MUCH
* UNIQUE STORAGE IS TO BE RESERVED.
* E2. ASSIGN CHAIN
* TRAVERSE THE CHAIN AGAIN, ASSIGNING EVERY
* VARIABLE IN THE CHAIN RELATIVE TO THE OTHERS.
* GO TO DEFX.
* E3. 'EQUIVALENCE'
* ON THE EQUIVALENCE DECLARATION, VARIOUS
* MODES ARE SET UP, AT THE END OF EACH
* EQUIVALENCE, A CHECK IS MADE TO SEE IF
* ANY OF THE ITEMS WAS PREVIOUSLY DEFINED.
* IF SO, THE ENTIRE CHAIN IS THEN DEFINED,
* AS IN STEP E2.
FORTAN II

UNIVAC
DIVISION OF SPERRY RAND CORPORATION