This technical report has been made electronically available on the World Wide Web through a contribution from

Douglas G. LaVerne

In honor of

Melvin E. LaVerne

Scientist and Author at
Oak Ridge National Laboratory

Office of Scientific and Technical Information
Office of Science
U.S. Department of Energy
August 2010
PROGRAMMERS MANUAL
FOR THE
ORACLE COMPILER

August 23, 1957
revised
June 15, 1959
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
Unclassified

PROGRAMMERS MANUAL

FOR THE ORACLE COMPILER

M. E. LaVerne          B. J. Osborne
R. R. Bate             A. C. Downing
R. R. Coveyou          J. Harrison
J. G. Sullivan         E. C. Long
A. A. Grau

June 15, 1959

Unclassified
The first version of the ORACLE Compiler was planned and written by Capt. R. R. Bate, U. S. Army, then associated with Oak Ridge National Laboratory. Later versions, including the present one, were planned and written by M. E. LaVerne, of the Applied Nuclear Physics Division, with the cooperation of members of A Nu P and the Mathematics Panel. This manual has been prepared by members of the Mathematics Panel from working papers written by members of both divisions.

# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. DESCRIPTION</td>
<td>1</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Limitations</td>
<td>1</td>
</tr>
<tr>
<td>Item Format</td>
<td>2</td>
</tr>
<tr>
<td>Key Words</td>
<td>3</td>
</tr>
<tr>
<td>Cross-Item References</td>
<td>4</td>
</tr>
<tr>
<td>Definitions</td>
<td>4</td>
</tr>
<tr>
<td>Full cross-item reference</td>
<td>4</td>
</tr>
<tr>
<td>Shorthand cross-item references</td>
<td>6</td>
</tr>
<tr>
<td>Subroutine Entries</td>
<td>6</td>
</tr>
<tr>
<td>Classification of Subroutines</td>
<td>8</td>
</tr>
<tr>
<td>Permanent Constants, Temporary Storage, and Inter-Segment Storage</td>
<td>9</td>
</tr>
<tr>
<td>Special Rules</td>
<td>11</td>
</tr>
<tr>
<td>Protected addresses</td>
<td>11</td>
</tr>
<tr>
<td>References to absolute addresses</td>
<td>11</td>
</tr>
<tr>
<td>Forbidden item numbers</td>
<td>11</td>
</tr>
<tr>
<td>Conflicting item numbers</td>
<td>11</td>
</tr>
<tr>
<td>Subroutine sentinals</td>
<td>12</td>
</tr>
<tr>
<td>Corrector routine</td>
<td>12</td>
</tr>
<tr>
<td>Basic Item</td>
<td>12</td>
</tr>
<tr>
<td>Zero Items</td>
<td>13</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Alphameric Items</td>
<td>14</td>
</tr>
<tr>
<td>Private Library</td>
<td>14</td>
</tr>
<tr>
<td>Arrangement of the Final Routine</td>
<td>15</td>
</tr>
<tr>
<td>Directory</td>
<td>17</td>
</tr>
<tr>
<td>Breakpoints</td>
<td>19</td>
</tr>
<tr>
<td>Edit of Compiled Program</td>
<td>19</td>
</tr>
<tr>
<td>Correction of Paper Tape Items</td>
<td>20</td>
</tr>
<tr>
<td>PROGRAMING</td>
<td>22</td>
</tr>
<tr>
<td>Programing in General</td>
<td>22</td>
</tr>
<tr>
<td>Programing with the Compiler</td>
<td>23</td>
</tr>
<tr>
<td>Use of the Temporary Storage</td>
<td>24</td>
</tr>
<tr>
<td>Use of the Intersegment Storage</td>
<td>25</td>
</tr>
<tr>
<td>Computations Involving Large Quantities of Data</td>
<td>25</td>
</tr>
<tr>
<td>OPERATING PROCEDURE</td>
<td>27</td>
</tr>
<tr>
<td>Structure of the ORACLE Compiler</td>
<td>27</td>
</tr>
<tr>
<td>Operating Instructions for Compiler</td>
<td>28</td>
</tr>
<tr>
<td>GLOSSARY</td>
<td>29</td>
</tr>
</tbody>
</table>
Introduction

Compiler routines have many forms and differ widely in their characteristics because of the machines and programmers for which they are prepared. This compiler is based upon the premise that the following characteristics are desirable for an ORACLE compiler:

1. The coding system should facilitate ORACLE coding.
2. The compiler should be able to select required sub-routines from a large library.
3. Selection of commonly used constants from a list should be possible.
4. Allocation of storage should be automatic.
5. Cross-references between subroutines should be possible without annoying hand computation of addresses.
6. Memory space should be used economically.
7. A directory of the location of subroutines, permanent constants, and temporary storage should be produced.
8. The record of the final routine should be complete.

In order to use this compiler to assemble a routine, relatively short pieces of code, called items, are prepared. These are coded as if each had its first word in location 000.

Any item may refer to standard subroutines or items in the compiler library. The compiler will assemble all of these and translate them into an ORACLE program.

Limitations

A routine that is so large that it must be broken into segments which are brought into high speed storage as they are needed is not
automatically handled by this compiler. Automatic checking for coding errors is still very rudimentary.

- Maximum length of any paper tape item (including extra words) = \(2^{10}_H\).
- Maximum number of items = \(0A0_H\) (160\(D\)).
- Maximum number of words in a alphameric item = \(078_H\).
- Maximum number of E -- references = \(160_H\).
- Maximum number of words in breakpoint list = \(31_H\).

**Item Format**

An item contains ORACLE instructions, constants, and additional words which provide information to the compiler. Each item consists of the following parts in order:

1. Nine key words.
2. Sequence of instruction words and extra words.
3. Constants other than those on the compiler list.
   - (i) Binary constants.
   - (ii) Decimal fractions to be converted by compiler.
   - (iii) Decimal integers to be converted.
   - (iv) Packed decimal numbers to be converted to (8,32) floating point.
   - (v) Two-word floating point decimal numbers.*

The compiler decodes and orients the instruction part, leaves the binary constants unchanged, converts the decimal constants, and inserts a specified number of zero words after the constants.

*The present compiler does not handle these.
Any number of items may be placed on a paper tape but each item must end with a carriage return and each tape must have a zero sentinel.*

Key Words

The nine key words, all hexadecimal integers, at the beginning of each item on paper tape are:

- Item number \( h \)
- Number of instruction words \( v \)
- Number of extra words \( x \)
- Number of binary constants \( c \)
- Number of decimal fractions \( d \)
- Number of decimal integers \( e \)
- Number of packed decimal floating point numbers to be converted to \((8,32)\) \( f \)
- Number of \((40,40)\) floating point words \( g \) (\(=0\))
- Number of zero words \( z \)

Carriage Return**

The total number of all words in an item, including extra words, is restricted to \(2^{40}_H\). That is

\[ 5 + v + x + c + d + e + f + 2g = 2^{40}_H. \]

If it is necessary to consult the magnetic tape version of an item, it must be noted that the latter uses five key words because the decimal-to-binary conversions have been made. The key words are:

\( h, v, v+x, c+d+e+f+2g, \) and \( z \).

*A zero sentinel may be a carriage return following the carriage return which terminates the last item; however, a zero word or an additional space will have the same effect.

**A carriage return must always follow the last key word.
Cross-Item References

1. Definitions

Some complication in specifying the location of a word is introduced by the extra words which, although they appear in the variable part of an item before compilation, are eliminated when the instruction part is assembled. We must carefully distinguish the following four methods of specifying location.

a. Absolute address
   The fast-storage location of a word in the assembled routine.

b. Item relative address
   The location of a word in the assembled routine relative to the zeroeth word of its item. Extra words are not counted.

c. Item position
   The position of a word counting both ordinary and extra words from the zeroeth word of an item. Key words are not counted.

d. Item tape position
   The position of a word on a paper item-tape counting both ordinary and extra words from the zeroeth word of an item. This may differ from the item position because of typing errors.

The last two methods of specifying the location are used in making corrections to paper tape items.

2. Full cross-item reference

A reference to an address within the same item is made in exactly the same manner as in normal ORACLE coding. A reference to an
address in another item is made as follows:

a. The address part of the instruction is replaced by FFF.

b. Immediately following, an extra word with no item relative address is inserted which specifies the number of the other item and the relative address in the other item.

The item relative address is placed in the left half and the item number in the right half of the extra word. In case two or three cross-item references are made in the same word, two or three extra words in order are used.

**Example:** The pseudo code in 03F of a given item consists of instructions to place the contents of the word in item relative address 007 of item 00 200 into relative address 020 of the given item. This will read:

```
03F: 24 FFF 5F 020
x: 00 007 00 200
```

Then to put the word in relative address 009 of item 00 201 into relative address 013 of item 00 204, the following words should be:

```
040: 24 FFF 5F FFF
x: 00 009 00 201
x: 00 013 00 204
```

Note that the extra words are not considered to occupy any memory cell because they are eliminated by the compiler. If the items in the immediately preceding example begin at the following addresses:

<table>
<thead>
<tr>
<th>ITEM</th>
<th>CELL</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 200</td>
<td>300</td>
</tr>
<tr>
<td>00 201</td>
<td>310</td>
</tr>
<tr>
<td>00 204</td>
<td>320</td>
</tr>
</tbody>
</table>
then the final coding will read:

```
03F:  24 307  5F 020
040:  24 319  5F 333.
```

3. **Shorthand cross-item references**

In order to shorten the cross-item references to frequently used items the following shorthand reference to the five special items numbered 00 800, 00 900, 00 A00, 00 B00, 00 C00 is used: the address is replaced by the relative address plus the item number, provided the relative address is less than 100.

**Example:** To bring the 010 word of item 00 900 into the accumulator write

```
24 910 -- -- --
```

Full cross-item references to these special items are still possible and must be used if the reference is to a relative address greater than OFF.

**Subroutine Entries**

Using full cross-item references, the standard entry into a closed subroutine may be written as follows: let the 021 word of some item be an entry into the 000 word of a subroutine whose item number is 00 423. Then this word would be

```
021:  24 021  43 FFF
   x:  00 000  00 423.
```

If the subroutine is to be entered at the right of word 005, then the code would be

```
021:  24 021  41 FFF
   x:  00 005  00 423.
```
In general, then, a subroutine entry may have the form

\[
\omega-1: \quad 24 \quad 24 \quad 24 \quad FFF \\
x: \quad 00 \quad \omega \omega \omega \quad hh \quad hhh
\]

where \(\omega \omega \omega\) is the relative address of the word in item hh hhh to which control is to be transferred.

A subroutine entry is normally indicated, however, by the abbreviated form

\[
FF \quad \omega \omega \omega \quad hh \quad hhh,
\]

where FF serves as an indicator to the compiler of a subroutine entry. If \(\omega \omega \omega < 800\) the entry order set up by the compiler will be a transfer to the left of \(\omega \omega \omega\), while if \(\omega \omega \omega \geq 800\), the transfer will be to the right of the \(\omega \omega \omega\)-800 word in item hh hhh.

**Example:** If the 021 word of item 00 010 is to be a subroutine entry into item 00 425, then this word should read

\[
FF \quad 000 \quad 00 \quad 423.
\]

When the routine is compiled, if item 00 010 is placed by the compiler starting in cell 080 and item 00 423 starts in cell 100, then the compiler will change this word to read

\[
0A1: \quad 24 \quad 0A1 \quad 43 \quad 100
\]
in the final code.

If the entry is desired to be in the 005 word on the right of item 00 423, then the word should read

\[
FF \quad 805 \quad 00 \quad 423.
\]

Provided the two items are placed in the same position in the memory by the compiler, as in the last example, the word will be changed
Classification of Subroutines

The last two bits of the second hex digit of the item number are used to indicate whether the item is an interpretive routine and whether it is three-address or single-address.

<table>
<thead>
<tr>
<th>Last two bits of the second hex digit of item number</th>
<th>Type of routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal</td>
</tr>
<tr>
<td>1</td>
<td>Three-address, single-word to be interpreted, no sentinel.</td>
</tr>
<tr>
<td>2</td>
<td>Single-address, many words to be interpreted followed by a sentinel.</td>
</tr>
<tr>
<td>3</td>
<td>Three-address, many-words to be interpreted followed by a sentinel.</td>
</tr>
</tbody>
</table>

Following an entry to a subroutine of type 2 or 3 will be a list of words to be interpreted. This list must be ended with a sentinel, which is a word of the form FA---, but only one sentinel can follow any one interpretive subroutine entry. It is sometimes required in the final routine that 800 be added to one of the addresses in the last interpreted word to signal the end of the list. The sentinel indicates which address of the preceding word is to be increased by 800 as follows:
FA 800 00 000 Left address of three-address word.
FA 008 00 000 Left address of single-address word.
FA 000 00 008 Right address of single-address word.
FA 000 00 000 No address increased.

Permanent Constants, Temporary Storage, and Inter-Segment Storage

References to permanent constants, temporary storage, and inter-segment storage are made just as if they occupied special shorthand items numbered as follows:

00 DOO Inter-segment storage.
00 EOO Permanent constants.
00 FOO Temporary storage.

The item 00 EOO is stored with the compiler on magnetic tape. It is reserved for numbers or words frequently used. A list of words assigned to item 00 EOO is available as ORACLE Memo Number 5.

The items 00 FOO and 00 DOO do not actually exist since all that is required for them is space in the final routine. References to item 00 EOO, 00 FOO, or 00 DOO are made in the same manner as references to a shorthand item.

The inter-segment storage item 00 DOO, is designed for use by programs made up of more than one segment or memory load. The space left in the final routine for temporary storage is just enough for all the storage positions between FOO and the greatest address of a temporary referred to in the routine, inclusive. Similarly, the space for inter-segment storage is just enough for all the storage positions between DOO
and the greatest address of inter-segment storage referred to, inclusive. Permanent constants to which reference is made are inserted into the routine.

**Example:** If the routine refers to the following constants and storage:

$$\begin{align*}
\text{P05, F09, 080 of 00 F00, 102 of 00 F00} \\
\text{D04, D07} \\
\text{E80, E84, E85}
\end{align*}$$

then the space left for temporary storage will be 103 words, for inter-segment storage 008 words, the three permanent constants corresponding to E80, E84, E85 will be inserted in consecutive cells.

For obtaining more than one consecutive permanent constant, one may call for them in this manner.

$$\begin{align*}
\text{24 FFF} \\
\text{xx yyy 00 E00}
\end{align*}$$

xx = number of constants desired + 1

yyy = number of first constant

$$\begin{align*}
\text{24 FFF} \\
\text{00 EA0 00 E00 = 1 constant beginning at E00.} \\
\text{04 EA0 00 E00 = 5 constants beginning at E00.}
\end{align*}$$

Caution must be observed in the use of temporary storage. Most of the subroutines use temporaries in item 00 F00, so it is necessary to determine which ones are used by any subroutine incorporated in a code to avoid storage conflicts.

It is usually safe to assume that basic subroutines in the compiler library will not use temporaries higher than F0F.
Special Rules

1. Protected addresses

The address portion of a stop, shift, hunt, or curve plotter instruction is not changed by the compiler unless that portion is FFF, in which case the usual cross-item reference specified by the extra word is made.

Special note should be made of linkage words of some subroutines which require

\[ M(A) \sim M(B) \]

in which case, an appropriate order should be used in the order position: otherwise M(A) and M(B) will not be changed to absolute addresses by the compiler. Full cross-item references may also be used.

2. References to absolute addresses

Reference to an absolute address may be made by a full cross-item reference to the fictitious item OC FFF: the relative address part of the extra word is inserted unchanged by the compiler in place of FFF.

3. Forbidden item numbers

Paper tape items bearing the following numbers may not be given to the compiler: 00 000, 00 DOO, 00 E00, 00 F00, 00 801, 00 901, 00 A01, 00 B01, 00 C01, OC FFF, FF FFF.

4. Conflicting item numbers

If an item on the private library has the same number as an item in the compiler library, the compiler chooses the former if it is referred to by another item selected from the private library. If several item numbers are the same in the private library the first item located by the compiler is used.
5. **Subroutine sentinels**

The left address of a three address word cannot be FA- as the compiler will treat this as the sentinel signifying the end of the list of three address words.

A word contained in the variable part of an item cannot begin with FF unless it is a subroutine entry, when using single address coding.

**FA---** sentinels must follow extra words when used as

\[
\begin{align*}
\alpha: & \quad \text{FFF } 2 \text{ 789 153} \\
x: & \quad 00 \text{ 017 00 B01} \\
x: & \quad \text{FA } 800 \text{ 00 000}.
\end{align*}
\]

6. **Corrector routine**

If a program is compiled to a specific place (as cell 320, etc.) using only items from the public library (drive 3 - compiler library), and calls for the corrector routine with the basic item word, then the compiler puts the corrector routine at that point and the item from the public library follows.

**Example:**

10320 03005 basic item word.

The compiled program is as follows:

FC8 at 320 corrector routine

03005 at 327 item called for.

**Basic Item**

The programmer specifies that a particular item is the basic item of the program. The compiler searches the basic item for references to other items. These are in turn searched for references to additional items. This process continues until no new references are found.
The basic item must therefore refer directly or indirectly to all other items to be included in the final program. The location of the basic item is also specified by the programmer. All other items follow the basic item in the memory.

The basic item and its location are specified in a word on paper tape called the basic item word. Its format is

\[yx \ \alpha\alpha\alpha\ \text{hh hh}\]

where

- \(y = 1\), if corrector routine (item FC8) is desired;
- \(y = 0\), if no corrector is desired;
- \(x = 1\), if a paper tape copy of the compiled program is desired;
- \(x = 0\), if no copy is desired;
- \(x = 5\), if both edit of program and paper tape copy are desired;
- \(x = 2\), if edit of program is desired (see section on edit of program);
- \(\alpha\alpha\alpha\) = location of the basic item. Add 800 if a breakpoint list follows (see section on breakpoints);
- \text{hh hh} = basic item number.

This word is followed by a space and then by the breakpoint list, if any. It is unnecessary that a zero sentinel follow the basic item word.

The breakpoint list is ended by a zero sentinel.

Following this comes the title tape if an edit is desired.

Zero Items

The key words of a zero item are the item number, seven zeros, and a nonzero number \(z\) specifying the number of zero words to be supplied by the compiler in the final program.
Such items are arranged in ascending order.

e.g. 1) 00 086
     2) 00 A00
     3) 00 B00
     4) -- ---
     5) 00 D00 - always proceeds the last zero item.

Should item 00 D00 not be used, storage for the zero item(s) will terminate at and include 7FF.

Actual zeros are not put in the memory locations specified unless the memory is cleared before the code is read in. This involves special instructions to the operator.

To end a zero item, a carriage return must follow the last key word.

**Alphameric Items**

Alphameric items are typed as follows:

1. Item Number (AXXX)
2. Space
3. Alphameric information
4. Ps 0000 Rs

They will be treated as regular items and can be on the same tape with other items with the customary zero sentinel after the last item.

Neither the item number nor the body of the alphameric item can be changed via correction tape.

**Private Library**

The items initially prepared on paper tape and used in compiling are written by Segment I of the Compiler on the magnetic tape present on drive 2. This record of items constitutes the *private library.*
If this record is made on a tape which can be removed and stored, it is possible to retain this library for further use in later compiling. The operating instructions applicable to compiling with a private library should then be used.

At the point where the basic item word is supplied to the machine, the compiler automatically writes a block of zero words on drive 2 which serves as an end-of-library sentinel for a subsequent search. It must therefore be noted that if, following the addition of paper tape items to the private library, a basic item word has not been supplied to the compiler, the block of zeros must be written on the tape using manual instructions before the tape is removed.

If compiling is effected using both a private library and additional items on paper tape, the private library after use will contain the latter as well.

To add an item which the compiler has declared missing on a previous attempt to compile is equivalent to compiling with a private library and the proper operating instructions should therefore be followed.

**Arrangement of the Final Routine**

The compiler starts the basic item in any memory location specified by the basic item word. Other items are inserted by the compiler immediately following the basic item. The assignment of their order and addresses is made automatically by the compiler. The final arrangement in the memory is as follows:

1. Basic item (paper tape or drive 2).
2. Items from private library (drive 2).
3. Items from compiler library (drive 3).
4. Permanent constants from item 00 E00.

5. Temporary storage.
   Space from end of temporary storage to beginning of zero items or inter-segment storage (unassigned).


7. Inter-segment storage.

If a group of items is supplied to the compiler in some desired sequence, then only if basic item is first on tape will the assembled routine contain these items in the same sequence. No transfer order from one item to the next is needed, but care must be exercised to see that each item is referred to directly by the basic item or by one of the items preceding it in the sequence.
**Directory**

The compiler punches out a directory which gives information in sequence as summarized in the following table.

<table>
<thead>
<tr>
<th>Form of word (one or more)</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. List of items as arranged on magnetic tape</td>
<td>00 xxx hh hhh</td>
</tr>
<tr>
<td>2. Sentinel</td>
<td>FF FFF FF FFF</td>
</tr>
<tr>
<td>3. Memory location of program</td>
<td>00 yyy 00 zzz</td>
</tr>
<tr>
<td>4. Free memory space</td>
<td>00 aaa 00 bbb</td>
</tr>
<tr>
<td>5. Sentinel</td>
<td>FF FFF FF FFF</td>
</tr>
<tr>
<td>6. List of permanent constants</td>
<td>00 ccc 00 ddd</td>
</tr>
<tr>
<td></td>
<td>ddd E00 + xxx</td>
</tr>
<tr>
<td></td>
<td>xxx address of constant in item 00 E00</td>
</tr>
<tr>
<td></td>
<td>Form of word (one or more)</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------</td>
</tr>
<tr>
<td>7.</td>
<td>Sentinel</td>
</tr>
<tr>
<td>8.</td>
<td>List of items</td>
</tr>
<tr>
<td></td>
<td>arranged by</td>
</tr>
<tr>
<td></td>
<td>item number</td>
</tr>
<tr>
<td>9.</td>
<td>Sentinel</td>
</tr>
<tr>
<td>10.</td>
<td>List of items</td>
</tr>
<tr>
<td></td>
<td>arranged by absolute</td>
</tr>
<tr>
<td></td>
<td>address</td>
</tr>
<tr>
<td>11.</td>
<td>Sentinel</td>
</tr>
<tr>
<td>12.</td>
<td>Breakpoint list</td>
</tr>
<tr>
<td>13.</td>
<td>Sentinel</td>
</tr>
<tr>
<td>14.</td>
<td>Code length on</td>
</tr>
<tr>
<td></td>
<td>magnetic tape</td>
</tr>
<tr>
<td>15.</td>
<td>Sentinel</td>
</tr>
</tbody>
</table>

**Interpretation:**
- **sss**: Absolute address of first word of item number `hh hhh`.
- **aaa**: Absolute address in compiled program.
- **x**: Portion of word in which breakpoint is to occur (see section on breakpoints).
- **bbb**: Item relative address.
- **ccc**: Last three digits of item number.
- **xxx**: Number of blocks code takes on drive 0.
Breakpoints

Because the usual breakpoint indication is used for other purposes in the pseudo-code, a separate paper tape is usually used to tell the compiler where breakpoints are to be inserted in the final coding. As an alternative the full cross-item reference can be used for this purpose by adding 800 to the relative address portion of the extra word. For example, a word in item hh hhh will read:

without breakpoint: 24 006 5F F00
with breakpoint: 24 FFF 5F F00
00 806 hh hhh.

The usual breakpoint indication in a protected address is a third alternative; e.g., 10 800.

The breakpoint tape contains in order the basic item word, a list of breakpoint words, a zero word, and a carriage return. For a breakpoint in relative address of item hh hhh the breakpoint word takes the form:

x0 cxxx hh hhh, where
x = 8 80 cxxx hh hhh breakpoint on left
x = 4 40 cxxx hh hhh breakpoint on right
x = C 00 cxxx hh hhh breakpoint on left and right.

Breakpoints cannot be on words in items D00, E00, F00. The breakpoint list cannot specify adding breakpoints on first and second addresses of three address words.

Edit of Compiled Program

If desired, a programmer may receive via narrow magnetic tape an alphanumeric edit of the program.
The format gives the item relative address, the absolute address, four words of code, and the item number. The item number is printed only at the beginning of each item and the items are separated by a carriage return. A title is printed on the first page.

An alphameric identification tape must be furnished consisting of a minimum of two carriage returns and Ps 0000 Rs, or a maximum of 60 printing characters preceding each carriage return. Tabs are available at 22, 33, 44, and 55 spaces from the left margin, with the center of the format between 30 and 31 spaces. This tape must follow the basic item word or the breakpoint list if present. It is necessary to have a reference to a FOO address.

The instructions for setting the typewriter precede the edit of the code.

**Correction of Paper Tape Items**

For typing errors made in the paper tape items, the following format should be used to type correction tapes. These are used as the routine is compiled and must be fed into the machine preceding the item or items in error.

<table>
<thead>
<tr>
<th>FF FFF FF FFF</th>
<th>Starts each correction tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 041 00 021</td>
<td>Replaces the word having item tape position 041 in item 00 020 with 1B 021 5F C01.</td>
</tr>
<tr>
<td>1B 021 5F C01</td>
<td></td>
</tr>
<tr>
<td>0A 062 00 032</td>
<td>Puts 1B 032 5F C03 into item tape position 062 in item 00 032 and moves the word which was there to 063, etc.</td>
</tr>
<tr>
<td>1B 032 5F C03</td>
<td></td>
</tr>
<tr>
<td>0D 073 00 040</td>
<td>Deletes the word in item tape position 073 in item 00 040 and moves the word in 074 up to 073, etc.</td>
</tr>
<tr>
<td>00 000 00 000</td>
<td></td>
</tr>
<tr>
<td>00 000 00 FFF</td>
<td>Sentinel for end of corrections.</td>
</tr>
</tbody>
</table>

**PROGRAMMERS MANUAL FOR THE ORACLE COMPILER**

Revised 9/2/59
Key words are corrected in the same manner, the difference being a 1 in the first hex digit.

10 001 00 046 Replaces the 2nd key word, v,
00 000 00 06A in item 00 045 with 6A*.

Corrections and items may be on the same paper tape but each tape must end with a zero sentinel.

Corrections via correction tape will not correct items on public or private library but only items on paper tape(s).
II. PROGRAMING

Programming in General

Before specific instructions can be written down for any computation with a stored program computer, it is necessary that the programmer have a clear picture of the manner in which the computer will carry out the calculation. This is usually obtained by preparing a flow chart which indicates the various logical components of the calculation. In preparing such a flow chart, it is convenient to start with the inner loops of the calculation. The necessary steps for setting up these loops and for completing the calculation are attached later. After the flow chart is complete, the same process is convenient. The instructions for the inner loops -- the heart of the calculation -- are written first. Then attention is given to the necessary details for reading in the data, converting and arranging them, and for initially setting the variable addresses in the heart of the calculation. Clearly it is difficult to decide exactly how to set the variable addresses within a program before it is known precisely which addresses these will be.

There is nothing in the foregoing which requires a system of relative addressing. In most calculations, however, it is necessary to reserve a section of memory for the initial addresses, parameters and constants peculiar to the routine. It is convenient to keep these together and to decide in advance where they will be located so that specific addresses can be used when writing instructions referring to these words. Unfortunately, it is either awkward or wasteful of storage space to assign absolute addresses
to these words before the total number of instructions in the routine has been determined. To avoid this, either symbolic addresses or relative addresses must be used. Without an assembly or compiling routine they must later be translated by the programmer to absolute addresses.

**Programming with the Compiler**

First, choose one of the shorthand items as storage for the parameters, constants, etc., which will be needed in the program. It is not necessary to decide in advance how many will be required. As a need arises, say for an additional initial address word or a test word for a loop with variable addresses, it can be added to this shorthand item. In complex problems, it may be convenient to assign two or more of the shorthand items for such storage. Next, write the instructions for the inner loops of the calculation. The choice of item numbers for the program is rather arbitrary. One convenient way is to use a numbering system which includes the box numbers on the flow chart so that identification between the individual items and the flow chart is made automatically.

It is essential to keep in mind that errors will be made in writing a routine -- they may be logical mistakes; they may be omissions; they may be only transposing digits in addresses. In order to minimize the chain reactions which will be induced in the routine when correcting errors, it is advisable to keep each item short -- try to restrict any one item to a single logical or numerical function. Where possible, enter an item only at 000. If it is later necessary to revise completely such an item, no changes will be necessary in other items which transfer to this one. An
ideal length for an item is the number of instructions that can be written on one piece of paper. This much can be easily kept in mind. If an item is longer, the risk of misremembering an address from the first part of the item is greater, and, of course, there is more to rewrite when correcting an error.

**Use of the Temporary Storage**

Most of the basic routines that have been placed in the public library make use of some of the temporaries from F00 through F0F. These are the conversion routines and the routines for evaluating the elementary transcendental functions. If subroutines are to be used, it is advisable to avoid using these storage locations for words which will be needed at a later point in the calculation.

In connection with temporary storage, attention is called to the two groups of fifteen words in item 00 E00 -- the words, E71 through E7F and EFl through EFF -- which are a hybrid of permanent constants and temporaries. These are useful when unpacking a word containing several pieces of information which would otherwise be partially substituted into a cleared storage location. Note that the last hexadecimal character of the relative address agrees with the second character in the 5x or 7x substitution order to be used with each word. Normally the EFl group is used in preference to the E71 group.
Use of the Intersegment Storage

For a large routine it is convenient to separate the problem into three or more phases, the first and the last phases being the input and the output, respectively. When memory space is at a premium, it is desirable to have the input phase of the program in the high-speed storage by itself and, after the calculation is complete, the output phase in storage by itself. The portion of the routine for each phase is then called a segment of the routine. Certain parameters will be needed by all segments of the problem. These can be stored in shorthand item CO DOO, which the compiler places at the end of the memory. When compiling each segment of the program, be certain that it contains a reference to the largest address in CO DOO referred to in any of the segments.

Computations Involving Large Quantities of Data

If a routine involves handling a large amount of data, greatest efficiency is obtained only when the entire high-speed memory is used. In preparing such a routine, if no zero items have been used, then one can definitely say that the first word at the end of the memory, which is not available for storage, will be DOO, the first word in the inter-segment storage. (When zero items are used, the compiler places them at the end of the memory just preceding the CO DOO item. They are arranged in order of increasing item numbers. Consequently, the first word not available for general storage is the 000 word of the zero item with the lowest item number.) In the compiled program, the first available word in the unused portion of storage will have an address one greater than the largest address used in the CO FOO item, in other words, in temporary storage. Thus routines can be written
in which it is not necessary to decide in advance that 1400 or 1600 words of storage will be available for data processing. One method for accomplishing this is to let two words in the shorthand item for constants, parameters, etc., be

\[ 40 \text{ Fxx} \quad 40 \text{ Fxx} \]

and

\[ 40 \text{ DOO} \quad 40 \text{ DOO}, \]

where Fxx is the last temporary used. In programing, then, these words will provide the relative addresses of the first free cell and the last free cell plus one.

To check out a routine using this system, Fxx can be replaced by an artificially large address. The storage space available is then extremely limited and those portions of the routine normally used only for very large quantities of data may be tested on more convenient sized problems.
### Structure of the ORACLE Compiler Routine

The compiler is stored on a magnetic tape which is placed on drive 3. For identification the reel on which it is wound is red. The location consists of four segments, of which each of the first three contain 5400 words, and the fourth contains 5800 words. All segments begin at register 7E0. The function and tape location of each of these segments is given below.

<table>
<thead>
<tr>
<th>Segment</th>
<th>Tape Location</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Blocks C-B, inclusive</td>
<td>Loading of correction tapes, paper item tapes, conversion of numbers; storage of corrected, converted items on drive 2.</td>
</tr>
<tr>
<td>ii</td>
<td>Blocks C-17, inclusive</td>
<td>Compiling of item and permanent constant list; storage of all required items on drive 0 with memory space allotted.</td>
</tr>
<tr>
<td>iii</td>
<td>Blocks 18-23, inclusive</td>
<td>Changing all required addresses to absolute addresses; assembling and storing of program on drive 0.</td>
</tr>
<tr>
<td>iv</td>
<td>Blocks 24-2F, inclusive</td>
<td>Punching the compiled program; reading program into memory, if desired.</td>
</tr>
</tbody>
</table>
Operating Instructions for Compiler

When the compiler is in use, the 2048 word memory is needed and the no-write switch for drive 3 must be on. The tape is zeroed at 020 blocks.

A. 5A0* in Q
B. B3 7E0
C. 43 000, breakpoint switch off
D. At stop L 1B3
   1. If there is a private library on drive 2 go back on operate.
   2. If there is no private library on drive 2 depress special order**, operate.
E. At stop L 1B8
   1. If there are corrections or paper tape items, put them under reader and go back on operate.
   2. Will keep stopping at L 1B8 until all items are loaded. After last item is loaded, put basic item word under reader, depress special order**, then operate.
F. Will stop at L 059 after paper tape copy has been punched.
   1. Go back on operate and program will be read into cell 000 in fast memory from drive 0.
   2. If program is to start anywhere but 000, the programmer will have to give the instructions for the program to be read into the memory (console instructions).

**The special order register must read 43 000 when you depress special order key, and then operate. If there has been a tape error and the register reads something else, give a 43 000, special order and then operate.
This is a glossary of the terms used in describing the ORACLE Compiler. Most of the definitions have been taken from a glossary published by the Association for Computing Machinery.* Those terms peculiar to this compiler and not in general use in the computing field are indicated by an asterisk.

Address - a label such as an integer or other set of characters which identifies a register, location, or device in which information is stored.

**Absolute Address** - the label assigned by the machine designer to a particular storage location. Specifically, the internal storage location of a word in the assembled routine.

**Relative Address** - a label used to identify a word in a routine or subroutine with respect to its position in that routine or subroutine. Relative addresses are translated into absolute addresses by the addition of some specific "reference" address, usually that at which the first word of the routine is stored.

*Item Relative Address* - the location of a word in the assembled routine relative to the zeroeth word of its item. Extra words are not counted.

*Item Position* - the position of a word counting both ordinary and extra words from the zeroeth word of an item. Key words are not counted.

*Item Tape Position* - the position of a word on a paper item-tape counting both ordinary and extra words from the zeroeth word of an item. This may differ from the item position because of typing errors.

*Report to the Association for Computing Machinery: First glossary of Programming Terminology, by C. W. Adams et al., June 1954.*
Assemble; Assembler, Assembly Routine; Assembly - see Routine.

*Basic Item - the first item of a compiled routine. It is chosen by the programmer who also specifies its location in storage. (See basic item word). The composition of the final routine is determined by the basic item. The compiler selects from the private and public libraries every item referred to by the basic item and in addition every item referred to by the items already selected for inclusion in the final routine.

*Basic Item Word - after all the paper tape items have been recorded on the private library tape, the basic item, bb bbb, and its location, oocx, in the assembled routine are specified by the paper tape word Ox oocx bb bbb, where x = 1 or 0 depending upon whether or not a paper tape copy of the final routine is desired.

Code (noun) - a system of symbols and of the rules for their use in representing information.

Computer Code (Machine Code) - the code representing the operations built into the hardware of the computer.

Instruction Code - the symbols, names, and definitions of all the instructions which are directly intelligible to a given computer or a given executive routine.

Pseudo-code - an arbitrary code, independent of the hardware of a computer, which must be translated into computer code if it is to direct the computer.

Code (verb) - to prepare problems in computer code or in pseudo-code for a specific computer.

Coding - the list, in computer code or in pseudo-code, of the successive computer operations required to solve a given problem.

Absolute, Relative, or Symbolic Coding - coding in which one uses absolute, relative, or symbolic addresses, respectively.

Compile; Compiler, Compiling Routine; Compilation - see Routine.

*Cross-Item Reference - a reference in one compiler item to a relative address in another item.

*Full Cross-Item Reference - a reference making use of an extra word. The address portion of the pseudo-instruction preceding the extra word is FFF.
Shorthand Cross-Item Reference - a reference to a word in one of the shorthand items which is made without the use of an extra-word. The relative address of the desired word in the shorthand item is added to the shorthand item number and this is treated as a pseudo-address of that word.

*Extra-Word* - a word inserted in the instruction field of a compiler item following a pseudo-instruction which refers to a location in another item. The extra-word specifying relative address $\text{XXX}$ in item $\text{hh hhh}$, has the form $00 \text{XXX} \text{hh hhh}$. See pseudo-instruction.

Field - a set of one or more characters (not necessarily all lying in the same word) which is treated as a whole; a unit of information. See also Item, Key.

Instruction - a set of characters which defines an operation together with one or more addresses (or no address) and which, as a unit, causes the computer to operate accordingly on the indicated quantities.

*Pseudo-Instruction* - specifically, a word in the instruction field of a compiler item having the form of a machine instruction but whose addresses either (1) are relative addresses of other words in the item, (2) are shorthand cross-item references, or (3) are replaced by the tag FFF. In the latter case, an extra-word must be provided for each FFF tag in the pseudo-instruction.

Item - a set of one or more fields containing related information; a unit of correlated information relating to a single person or object. Also used as an abbreviation for compiler item.

*Compiler Item* - a set of words treated by the ORACLE Compiler as a unit of a routine. It may contain either instructions, or constants, or both. Additional information in the form of key-words and extra-words is included in an item for use by the compiler in determining the form of the item in the assembled routine. The key-words and extra-words do not appear in the version of the item incorporated in the assembled routine.

A compiler item consists of seven fields, namely,

1. nine key-words;
2. an instruction field, consisting of $y$ pseudo-instructions and $x$ extra words;
(3) \( c \) binary constants;

(4) \( d \) decimal fractions to be converted to binary;

(5) \( e \) decimal integers to be converted to binary;

(6) \( f \) packed floating decimal numbers to be converted to \((8,32)\);

(7) \( g \) unpacked floating decimal numbers to be converted to \((40,40)\).

The number of words in an item is \(9+v+x+c+d+e+f+2g\). See key-words.

*Item Position, Item Tape Position* - see Address.

**Key** - a set of characters, forming a field, used to identify an item.

*Key-Words* - the first field of a compiler item consists of nine words containing, in order, the hexadecimal integers \( h, v, x, c, d, e, f, g, z \), where \( h \) is the item number and \( z \) specifies the number of zero words to follow the compiled form of the item in the assembled routine. The remaining seven key-words are defined under compiler item.

**Library** - an ordered set or collection of standard and proven routines and subroutines by which problems and parts of problems may be solved, usually stored in relative or symbolic coding. (A library may be subdivided into various volumes, such as floating decimal, double-precision, or complex, according to the type of arithmetic employed by the subroutines.)

**Location** - a unit storage position in the main internal storage, storing one computer word; a storage register.

**Pseudo-code** - see Code (noun).

*Reference* - see cross-item reference.

**Routine** - a set of coded instructions arranged in proper sequence to direct the computer to perform a desired operation or series of operations. See also Subroutine.

**Executive Routine** (Master Routine) - a routine designed to process and control other routines. A routine used in realizing "automatic coding".
Compiler (Compiling Routine) - an executive routine which, before the desired computation is started, translates a program expressed in pseudo-code into machine code (or into another pseudo-code for further translation by an interpreter). In accomplishing the translation, the compiler may be required to:

**Decode** - to ascertain the intended meaning of the individual characters or groups of characters in the pseudo-coded program.

**Convert** - to change numerical information from one number base to another (e.g., decimal to binary) and/or from some form of fixed point to some form of floating-point representation, or vice versa.

**Select** - to choose a needed subroutine from a file of subroutines.

**Generate** - to produce a needed subroutine from parameters and skeletal coding.

**Allocate** - to assign storage locations to the main routines and subroutines, thereby fixing the absolute values of any symbolic addresses. In some cases allocation may require segmentation. See Segment.

**Assemble** - to integrate the subroutines (supplied, selected, or generated) into the main routine, i.e., to:

**Adapt** - to specialize to the task at hand by means of preset parameters.

**Orient** - to change relative and symbolic addresses to absolute form.

**Incorporate** - place in storage.

Interpretive Routine - an executive routine which, as the computation progresses, translates a stored program expressed in some machine-like pseudo-code into machine code and performs the indicated operations, by means of subroutines, as they are translated. An interpretive routine is essentially a closed subroutine which operates successively on an indefinitely-long sequence of program parameters (the pseudo-instructions and operands). It may usually be entered as a closed subroutine and exit may be made by a pseudo-code exit instruction.
Service Routine – a routine designed to assist in the actual operation of the computer. Tape comparison, block location, certain post mortems, and correction routines fall in this class.

Specific Routine – a routine expressed in specific computer coding designed to solve a particular mathematical, logical, or data-handling problem.

Segment (noun) – in a routine too long to fit into internal storage, a part short enough to be stored entirely in the internal storage and containing the coding necessary to call in and jump automatically to other segments. (Routines which exceed internal storage capacity are not automatically divided into segments by this compiler.)

Segment (verb) – to divide a routine in parts each consisting of an integral number of subroutines, each part capable of being completely stored in the internal storage and containing the necessary instructions to jump to other segments.

Sentinel – a symbol marking the beginning or the end of some element of information such as a field, item, block, tape, etc; a tag.

*Zero-Sentinel – a zero word used as a sentinel.

*Shorthand Item – a compiler item numbered 00 800, 00 900, 00 A00, 00 B00, or 00 C00. Cross-item references to words in these items may be made without the use of extra-words by adding the relative address to the shorthand item number. See cross-item reference.

*Shorthand Reference – see cross-item reference.

Storage – any device into which units of information can be copied, which will hold this information, and from which this information can be obtained at a later time.

Internal Storage – storage facilities forming an integral physical part of the computer and directly controlled by the computer. See also Location and Register.

Secondary Storage – storage facilities forming not an integral part of the computer but directly linked to and controlled by the computer; e.g., magnetic drum, magnetic tapes, etc.

Temporary Storage – internal storage locations reserved for intermediate and partial results.
**Working Storage** - a portion of the internal storage reserved for the data upon which operations are being performed.

**Subroutine** - a set of instructions sufficient to direct the computer to carry out a well defined mathematical or logical operation; a subunit of a routine. A subroutine is often written in relative or symbolic coding even when the routine to which it belongs is not.

**Closed Subroutine** - a subroutine not stored in its proper place in the linear operational sequence, but stored away from the routine which refers to it. Such a subroutine is entered by a jump, and provision is made to return, i.e., jump back to the proper point in the main routine after the instructions in the subroutine have been executed.

**Open Subroutine** - a subroutine inserted directly into the linear operational sequence, not entered by a jump. Such a subroutine must be recopied at each point that it is needed in a routine.

**Library Subroutine (Standard Subroutine)** - a member of a subroutine library.

**Tag** - a unit of information, whose composition differs from that of other members of the set so that it can be used as a marker or label; a sentinel.

**Translate** - to change information (e.g., problem statements in pseudo-code, data, or coding) from one language to another without significantly affecting the meaning.

**Word** - a set of characters which occupies one storage location and is treated by the computer circuits as a unit and transported as such. Ordinarily a word is treated by the control unit as an instruction, and by the arithmetic unit as a quantity. Word lengths are fixed or variable depending on the particular computer.

**Zone** - a portion of internal storage allocated for a particular function or purpose.