UNIDEC ASSEMBLER

C. Stephen Carr

Utah University

Prepared for:
Advanced Research Projects Agency

6 June 1968
UNIDEC ASSEMBLER

June 6, 1968

Computer Science
Information Processing Systems
University of Utah
Salt Lake City, Utah
The difficulties in using the Project's PDP-8 computer are primarily due to its inadequate input/output facilities. Assemblies with paper tape and Macro-8 (manufacturer supplied) nominally require an hour to perform. As an alternative, the Unidec assembler runs on the Univac 1108 and passes assembled PDP-8 code over the electronic link between the two machines. The source statements are punched on cards for input into the 1108 in a format nearly identical to that of Macro-8. A printed listing and the object code are produced as fast as the cards can be read.

Of course, most users will prefer to write all programs for the 1108 and access the display facilities through the Graphics Monitor*. Indeed, this is why additional input/output facilities have not been purchased for the small PDP machine. However, Unidec meets the needs of those who require special PDP-8 programs.

The syntax and operation of Unidec differ sufficiently from that of Macro-8 to warrant a separate user manual. However, the systems are sufficiently similar that excerpts from the DEC manual (Macro-8 Programming Manual-8-8-S, Digital Equipment Corp., Maynard, Mass.) have been copied unaltered. The implementation on the 1108 is entirely new.

**FUNDAMENTALS**

**Characters**

Programs in the Unidec language use the following characters:

**Letters:** A B C D . . . X Y Z

**Digits:** 1 2 3 4 5 6 7 8 9 0

**Punctuation Characters:**

<table>
<thead>
<tr>
<th>Character</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>[space]</td>
<td>Combine symbols or numbers</td>
</tr>
<tr>
<td>+</td>
<td>Combine symbols or numbers (add)</td>
</tr>
<tr>
<td>-</td>
<td>Combine symbols or numbers (subtract)</td>
</tr>
<tr>
<td>!</td>
<td>Combine symbols or numbers (OR)</td>
</tr>
<tr>
<td>[space]</td>
<td>Terminate line</td>
</tr>
<tr>
<td>,</td>
<td>Assign symbolic address</td>
</tr>
<tr>
<td>=</td>
<td>Define parameters</td>
</tr>
<tr>
<td>;</td>
<td>Terminate coding line</td>
</tr>
<tr>
<td>$</td>
<td>End of pass</td>
</tr>
<tr>
<td>*</td>
<td>Set location counter</td>
</tr>
<tr>
<td>.</td>
<td>Has value equal to current location counter</td>
</tr>
</tbody>
</table>
Indicates start of a comment

Combine symbols or numbers (and)

Generate ASCII constant

Define literal on current page

Define page 0 literal

Define a macro

All other characters are illegal when not in a comment or a TEXT field, and cause the error message E to be printed.

Source statements are entirely field free:

GO, TAD TOTAL/MAIN LOOP/

but it is easier to read if spaces are inserted:

GO, TAD TOTAL /MAIN LOOP/

Either ; (semicolon) or the end of the card (column 72) terminates the line. The semicolon is considered identical to the end of a card except that it will not terminate a comment. Example:

TAD A /THIS IS A COMMENT ; TAD

The entire expression beyond the slash is considered a comment.

Use of the semicolon allows several lines of coding to be on a single line: For example, a sequence of instructions to rotate the C(AC) and C(L) six places to the right might look like:

... 
RTR 
RTR 
RTR 
...
The above sequence may be rewritten as:

RTR; RTR; RTR

This type of format is particularly useful when the instructions work together conceptually.

A neat printout (or program listing, as it is usually called) makes subsequent editing, debugging, and interpretation much easier than if the coding were laid out in haphazard fashion.

**Elements**

Any group of letters, digits, and punctuation characters which represents binary values less than $2^{12}$ is an element.

**Integers**

Any sequence of numbers delimited by punctuation characters forms a number. Example:

```
1
12
4372
```

The radix control pseudo-instructions set the radix to be used in number interpretation. The pseudo-instruction DECIMAL indicates that all numbers are to be interpreted as decimal until the next occurrence of the pseudo-instruction OCTAL and conversely.

**Symbols**

A symbol is a string of one or more alphanumeric characters delimited by a punctuation character. Symbols are composed according to the following rules:
1. The characters must be either alphabetic (A-Z) or numeric (0-9).

2. The first character must be alphabetic.

3. Only the first six characters of any symbol are meaningful; the remainder, if any, are ignored.

4. A symbol is terminated by any nonalphabetic character.

Unidec's permanent symbol table defines symbols for PDP-8 operation codes, operates commands, and many IOT commands (see Appendix A for a complete list). These may be used without prior definition by the user.

Example:

- JMS is a symbol whose value of 4000 is taken from the operation code definitions.
- A is a user-created symbol. When used as a symbolic address tag, its value is the address of the instruction it tags. This value is assigned by the Assembler.

Note that because of rule 3, a symbol such as INTEGER, for instance, would be interpreted as INTEGE since the seventh letter is ignored. If symbols of more than six characters are used, the programmer should be careful to avoid the error of defining two apparently different symbols whose first six characters are, in fact, identical. For example, the two symbols GEORGE1 and GEORGE2 differ only in the seventh character and would be treated as GEORGE.

It is not necessary to define a symbol before it is used in an expression. They must be defined before the end of PASS1, however. Thus, one may refer to a number of registers by their address tags, and then define the symbols later.
Parameter Assignments

A symbol may be assigned a value by means of a parameter assignment statement which looks like an algebraic statement. The single symbol to the left of the equal sign is assigned the value of the expression on the right. Examples:

A = 6
EXIT = JMP 10
C = A + B

All symbols to the right of an "=" sign must be already defined. The symbol to the left of the "=" sign and its associated value is stored in the Assembler's symbol table.

The use of the "=" does not increment the current location counter. It is, rather, an instruction to the Assembler itself rather than a part of the output binary. The equal sign may be used to redefine a symbol.

Symbol Definition

A symbol may be defined by the user in one of three ways:

1. By use of a parameter assignment. Example:

   DISMIS = JMP RESTOR

2. As a macro name. Example:

   DEFINE LOAD A
   <CLA
   TAD A>

3. By use of the comma. When a symbol is terminated by a comma, it is assigned a value equal to the current location counter.
Example:

```
*100
TAG,    CLA
B,      JMP A
A,      0
       DCA B
       ...

/SET CLC TO 100
```

The symbol "TAG" is assigned a value of 0100, the symbol "B" a value of 0102, and the symbol "A" a value of 0103.

Expressions

All elements, i.e., symbols and numbers (exclusive of pseudo-instruction symbols, and macro names) may be combined with certain operators to form expressions. These operators are:

- **+** plus
  
  This signifies 2's complement addition (modulo 4096_{10}).

- **-** minus
  
  This signifies 2's complement subtraction (modulo 4096_{10}).

- **!** exclamation point
  
  This signifies Boolean inclusive OR (union).

- **&** ampersand
  
  This signifies Boolean AND (intersection).

- **-** space
  
  Space is interpreted in context. It may signify inclusive OR or act as a field delimiter.

Symbols and integers may be combined with any of the above operators. A symbolic expression is evaluated from left to right; no grouping of terms is permitted. Example:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>A+B</th>
<th>A-B</th>
<th>A:B</th>
<th>A&amp;B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0002</td>
<td>0003</td>
<td>0005</td>
<td>7777</td>
<td>0003</td>
<td>0002</td>
</tr>
<tr>
<td>Value</td>
<td>0007</td>
<td>0005</td>
<td>0014</td>
<td>0002</td>
<td>0007</td>
<td>0005</td>
</tr>
<tr>
<td>Value</td>
<td>0700</td>
<td>0007</td>
<td>0707</td>
<td>0671</td>
<td>0707</td>
<td>0000</td>
</tr>
</tbody>
</table>
Unidec makes a distinction between the types of symbols it is processing. These types are 1) permanent symbols, 2) user defined symbols, and 3) macro names. The character "space" is interpreted written in the context of the expression. If a space is used to delimit two or more permanent symbols, space signifies inclusive OR. Example:

CLA is a permanent symbol whose value is 7200.
CMA is a permanent symbol whose value is 7040.

The expression:

CLA , CMA has a value of 7240.

If the symbol following the space is a user defined symbol, space acts as an address field delimiter. Example:

*2117
A, CLA
...  
...  
JMP , A

"A" is a user defined symbol whose value is 2117. The expression JMP , A is evaluated as follows:

The seven address bits of A are taken, i.e.:

A 010 00 1 001 111
   1 001 111

The remaining five bits of A are tested to see if they are 0's (page 0 reference); if they are not, the current page bit is set.

000 011 001 111

The operation code is ORed into the expression:

JMP 101 000 000 000
Address A 000 011 001 111
JMP    A 101 011 001 111

or, written more concisely:

5317.
In addition to the above outlined tests, the page bits of the address field are compared with the page bits of the current location counter. If the page bits of the address field are nonzero and do not equal the page bits of the current location counter, an out-of-page reference is being attempted. If the reference is to an address not on the page where the instruction will be located, the Assembler will set the indirect bit (bit 3) and an indirect address linkage will be generated on the current memory page. If the out-of-page reference is already an indirect one, the error diagnostic I (Illegal Indirect) will be printed. In the case of several out-of-page references to the same address, the link will be generated only once.

Example:

*2117
A, CLA
.
.
*2600
JMP  A

The space preceding the user defined symbol "A" acts as an address field delimiter. The Assembler will recognize that the register tagged "A" is not on the current page (in this case 2600-2777) and will generate a link to it as follows:

in location 2600 the Assembler will place the word

5777 which is JMP 1 2777

in address 2777 (the last location on the current page), the word

2117 (the actual address of "A" will be placed.

The address field of a storage reference instruction may be any valid expression. Example:

A=270
*200
TAD A-20
would produce, in location 200, the word

001 010 101 000 or 1250 (TAD 250)

Although the Assembler will recognize and generate an indirect address linkage when necessary, the programmer may indicate an explicit indirect address by using the special symbol "1". This must be between the operation code and the address field. The Assembler cannot generate a link for an instruction that is already specified as being an indirect reference. In this case, the Assembler will print I (Illegal Indirect).

Current Address Indicator

The single character period (.) has, at all times, a value equal to the value of the current location counter. It may be used as any integer or symbol (except to the left of an equal sign). Example:

*200
JMP .+2

Is equivalent to JMP 202.

*300
.+2400

would produce, in register 0300, the quantity 2700.

*2200
CALL=JMP 1.
0027

Since the second line, CALL=JMP 1., does not increment the current location counter, 0027 would be placed in register 2200 and CALL would be placed in the symbol table with an associated value of

100 110 000 000 or 4600.
Origin Setting

The origin (current location counter) is reset by use of the special character asterisk (*). The current location counter is set to the value of the expression following the "*". The origin is initially set to 0200. All symbols to the right of "*" must already have been defined. Example:

If D has the value 250

then

*D+10 will set the location counter to 0260.

To simplify page handling, the pseudo-instruction PAGE may be used. When "PAGE" is encountered, the origin is reset to the first location of the next page. A page number may be specified by a legal expression following the page pseudo-instruction. Example:

*270
...
...

at this point, either

*400
PAGE

or

PAGE 2

will reset the origin to 0400.

Literals

Since the symbolic expressions which appear in the address part of an instruction usually refer to the locations of registers containing the quantities being operated upon, the programmer must explicitly reserve the registers holding his constants or use a literal. Suppose, for example, that the programmer has an index which is incremented by two. One way of coding this operation would be as follows:
Using a literal, this would become:

```
... 
CLA 
TAD INDEX 
TAD C2 
DCA INDEX 
...
...

C2,2
```

The left parenthesis is a signal to the Assembler that the expression following is to be evaluated and assigned a register in the constants table of the current page. This is the same table in which the indirect address linkages are stored. In the above example, the quantity 2 is stored in a register in a list beginning at the top of the storage page (page address 177), and the instruction in which it appears is encoded with an address referring to that address. A literal is assigned to storage the first time it is encountered; subsequent references will be to the same register.

If the programmer wishes to assign literals to page 0 rather than the current page, he may use square brackets, "[" and "]" in place of the parentheses. However, in both cases, the right of closing member may be omitted. The following examples are acceptable:

```
TAD (777 
AND [JMP
```

Note that in the second example, the instruction AND [JMP has the same effect as AND [5000.

Literals may be nested. For example:

```
*200 
TAD (TAD (30
```
will generate:

```
0200   1376
...  ...
0376   1377
0377   0030
```

This type of nesting may be carried to as many levels as desired.

Literals are stored on each page starting at page address 177 and extending toward page address 0. (Only \(127_{10}\) or \(177_8\) literals may be placed on page 0). If a literal has been generated for a nonzero page and then the origin is set to another page, the current page literal buffer is emptied onto the old page where they are referenced. If the origin is then reset to the previously used page, the same literal will be generated if used again.

**Single Character Text Facility**

If a single character is preceded by a double quote, the 8-bit value of the ASCII code for the character is inserted instead of taking the letter as a symbol. Example:

```
CLA
TAD ("A
...
```

will place the constant 0301 in the accumulator.
PSEUDO-INSTRUCTIONS

The pseudo-instructions are directions to the Assembler to perform certain tasks or to interpret subsequent coding in a certain way. By themselves, pseudo-instructions do not generate coding or (in general) effect the current location counter.

**Current Location Counter**

**PAGE**
This pseudo-instruction is used to set the current location counter.

**PAGE n**
This will reset the current location counter (CLC) to the first address of page n, where n is an integer, a previously defined symbol, or a symbolic expression. Examples:

- `PAGE 2` will set the CLC to 0400
- `PAGE 6` will set the CLC to 1400

**PAGE**
When used without an argument, `PAGE` will reset the CLC to the first location on the next succeeding page. Thus, if a program is being assembled into page 1 and the programmer wishes to begin the next segment on page 2, he need only insert the pseudo-instruction `PAGE`, as follows:

```
JMP .-7
PAGE
CLA
```

The current location counter may be explicitly set by use of the asterisk.
Radix Control

Normally, all integers used in a program are taken as octal numbers. If, however, the programmer wishes to have certain numbers treated as decimal, he may use the pseudo-instructions:

DECIMAL  When this pseudo-instruction occurs, all integers encountered in subsequent coding will be taken as decimal until the occurrence of the pseudo-instruction OCTAL which will reset the radix to its original (octal) base.

Text Facility

There is a text facility for single characters and text strings. The single character mode (double quote) has been described previously.

A string of text may be entered by giving the pseudo-instruction TEXT followed by a space, a delimiting character, a string of text, and repeating the same delimiting character. Example:

TEXT    ATEXTA

The character codes are stored two to a register in ASCII code that has been trimmed to six bits. Following the last character, a 6-bit zero is inserted as a stop code. The above statement would produce

2405
3024
0000

TEXT    /BOB/

would produce

0217
0200

Note that while the TEXT pseudo-instruction causes characters to be stored in a trimmed code, the use of the single-character control code (") causes characters to be stored as a full 8-bit ASCII code.
End of Program

The special symbol "$" indicates the end of a program. When the Assembler encounters the "$", it terminates the PASS. Several programs may be assembled at one time. A dollar sign terminates each program.
MACROS

When writing a program, it often happens that certain coding sequences are used several times with just the arguments changed. If so, it is convenient if the entire sequence can be generated by a single statement. To do this, the coding sequence is defined with dummy arguments as a macro. A single statement referring to the macro by name, along with a list of real arguments, will generate the correct sequence in line with the rest of the coding. DEC's Macro-8 provides a weak macro facility which Unidec provides for compatibility.

The macro name must be defined before it is used. The macro is defined by means of the pseudo-instruction DEFINE followed by the macro's name and list of dummy arguments. For example:

A macro to move the contents of register A to register B and also leave the result in the accumulator, would be coded as follows:

DEFINE MOVE DUMMY1 DUMMY2
<CLA TAD DUMMY1 DCA DUMMY2 TAD DUMMY2>

The actual choice of symbols used as dummy arguments is arbitrary; however, they may not be defined or referenced prior to the macro definition.

The above definition of the macro MOVE is identical to the following:

DEFINE MOVE ARG1 ARG2
<CLA:TAD ARG1; DCA ARG2; TAD ARG2>

The actual definition of the macro is enclosed in angle brackets.

When a macro name is processed by the assembler, the real arguments will replace the dummy arguments. For example:
Assuming that the macro MOVE has been defined as above,

*400
A,0 0400 0000
B,-6 0401 7772
MOVE A,B 0402 7200
\$
0403 1200
0404 3201
0405 1201

NOTE: A macro need not have any arguments: For example, a sequence
coding to rotate the C(AC) and C(L) six places to the left
might be encoded as a macro by means of

DEFINE ROTL 6
<RTL;RTL;RTL>

The macro definition can consist of any valid coding except for TEXT or
" type statements.

Restrictions
1. Macros cannot be nested; i.e., another macro name or definition
cannot appear in a macro definition and cannot be brought in as
an argument at reference time.
2. TEXT or " type statements cannot appear in a macro definition.
3. Arguments cannot be:
   a. Macro name
   b. TEXT pseudo-instruction or " special character
4. The symbols used as dummy arguments must not have been previously
   defined or referenced.
5. A macro may not be redefined. Example:

DEFINE__LOOP__A__B

<TAD A
DCA B
TAD COUNT
ISZ B
JMP -.2>

The symbol "COUNT" is not a dummy argument but an actual symbol.
A macro is referenced by giving the macro name, a space, and then the list of real arguments, separated by commas. There must be at least as many arguments in the macro reference as in the corresponding macro definition. When a macro is referenced, its definition is found, expanded, and the real arguments replace the dummy arguments. The expanded macro is then processed in the normal fashion.

```
.....
LOOP...X,Y2
.....
```

is equivalent to:

```
.....
TAD X
DCA Y2
TAD COUNT
ISZ Y2
JMP .-2
.....
```
OPERATING INSTRUCTIONS

1. Set up an 1108 deck.

\[ \text{RUN } \_, \_, \_, \_, \_ \]
\[ \text{G} = \text{PDP}\$ \]
\[ \text{XQT CUR} \]
\[ \text{IN } \text{G} \]
\[ \text{XQT UNIDEC} \]

\[ \text{PDP-8 program} \]

\[ $ \leftrightarrow \text{(end of assembly indicator)} \]

\[ \text{next PDP-8 program} \]

\[ $ \]

2. The assembler uses the ARPAIO program. Therefore,
ready the PDP as explained elsewhere.

3. Run the 1108 deck into the 1004.

4. The assembled program goes directly into the PDP core storage.
A listing is produced on the 1004.

NOTE: The following PDP locations are used by the link program:

1, 3,
10, 11
20, 21
7000 - 7777

All other locations are available. Unidec assumes a starting location of
200g if no other origin information is given.
## APPENDIX A

Unidec Permanent Symbol Table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>0000</td>
<td>SZA</td>
<td>7440</td>
<td>TSF</td>
<td>6041</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAD</td>
<td>1000</td>
<td>SNA</td>
<td>7450</td>
<td>TCF</td>
<td>6042</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISZ</td>
<td>2000</td>
<td>SNL</td>
<td>7420</td>
<td>TPC</td>
<td>6044</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCA</td>
<td>3000</td>
<td>SZL</td>
<td>7430</td>
<td>TLS</td>
<td>6046</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMS</td>
<td>4000</td>
<td>SKF</td>
<td>7410</td>
<td>NOP</td>
<td>7000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>5000</td>
<td>OSR</td>
<td>7404</td>
<td>CLA</td>
<td>7200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOT</td>
<td>6000</td>
<td>HLT</td>
<td>7402</td>
<td>STA</td>
<td>7240</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPR</td>
<td>7000</td>
<td>CIA</td>
<td>7041</td>
<td>SPA</td>
<td>7510</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLA</td>
<td>7200</td>
<td>LAS</td>
<td>7604</td>
<td>I</td>
<td>0400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLL</td>
<td>7100</td>
<td>STO</td>
<td>7240</td>
<td>VEC</td>
<td>7000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMA</td>
<td>7040</td>
<td>STL</td>
<td>7120</td>
<td>JUMP</td>
<td>7400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CML</td>
<td>7020</td>
<td>GLK</td>
<td>7204</td>
<td>DOT</td>
<td>7410</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAR</td>
<td>7010</td>
<td>ION</td>
<td>6001</td>
<td>DASH</td>
<td>7420</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTR</td>
<td>7012</td>
<td>IOF</td>
<td>6002</td>
<td>LINE</td>
<td>7430</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAL</td>
<td>7004</td>
<td>KSF</td>
<td>6031</td>
<td>FRAM</td>
<td>7440</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTL</td>
<td>7006</td>
<td>KCC</td>
<td>6032</td>
<td>SYMB</td>
<td>7200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IAC</td>
<td>7001</td>
<td>KRS</td>
<td>6034</td>
<td>FIN</td>
<td>5407</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMA</td>
<td>7500</td>
<td>KRB</td>
<td>6036</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## APPENDIX B

### Alphanumeric Character Codes

<table>
<thead>
<tr>
<th>Symbol</th>
<th>ASCII</th>
<th>Fielddata</th>
<th>Trimmed-ASCII</th>
<th>Card</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø (zero)</td>
<td>60</td>
<td>60</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>61</td>
<td>61</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>62</td>
<td>62</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>63</td>
<td>63</td>
<td>23</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>64</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>65</td>
<td>65</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>66</td>
<td>66</td>
<td>26</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>67</td>
<td>67</td>
<td>27</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>70</td>
<td>70</td>
<td>30</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>71</td>
<td>71</td>
<td>31</td>
<td>9</td>
</tr>
<tr>
<td>A</td>
<td>101</td>
<td>06</td>
<td>41</td>
<td>12-1</td>
</tr>
<tr>
<td>B</td>
<td>102</td>
<td>07</td>
<td>42</td>
<td>12-2</td>
</tr>
<tr>
<td>C</td>
<td>103</td>
<td>10</td>
<td>43</td>
<td>12-3</td>
</tr>
<tr>
<td>D</td>
<td>104</td>
<td>11</td>
<td>44</td>
<td>12-4</td>
</tr>
<tr>
<td>E</td>
<td>105</td>
<td>12</td>
<td>45</td>
<td>12-5</td>
</tr>
<tr>
<td>F</td>
<td>106</td>
<td>13</td>
<td>46</td>
<td>12-6</td>
</tr>
<tr>
<td>G</td>
<td>107</td>
<td>14</td>
<td>47</td>
<td>12-7</td>
</tr>
<tr>
<td>H</td>
<td>110</td>
<td>15</td>
<td>50</td>
<td>12-8</td>
</tr>
<tr>
<td>I</td>
<td>111</td>
<td>16</td>
<td>51</td>
<td>12-9</td>
</tr>
<tr>
<td>J</td>
<td>112</td>
<td>17</td>
<td>52</td>
<td>11-1</td>
</tr>
<tr>
<td>K</td>
<td>113</td>
<td>20</td>
<td>53</td>
<td>11-2</td>
</tr>
<tr>
<td>L</td>
<td>114</td>
<td>21</td>
<td>54</td>
<td>11-3</td>
</tr>
<tr>
<td>M</td>
<td>115</td>
<td>22</td>
<td>55</td>
<td>11-4</td>
</tr>
<tr>
<td>N</td>
<td>116</td>
<td>23</td>
<td>56</td>
<td>11-5</td>
</tr>
<tr>
<td>O</td>
<td>117</td>
<td>24</td>
<td>57</td>
<td>11-6</td>
</tr>
<tr>
<td>P</td>
<td>120</td>
<td>25</td>
<td>60</td>
<td>11-7</td>
</tr>
<tr>
<td>Q</td>
<td>121</td>
<td>26</td>
<td>61</td>
<td>11-8</td>
</tr>
<tr>
<td>R</td>
<td>122</td>
<td>26</td>
<td>62</td>
<td>11-9</td>
</tr>
<tr>
<td>S</td>
<td>123</td>
<td>30</td>
<td>63</td>
<td>0-2</td>
</tr>
<tr>
<td>T</td>
<td>124</td>
<td>31</td>
<td>64</td>
<td>0-3</td>
</tr>
<tr>
<td>U</td>
<td>125</td>
<td>32</td>
<td>65</td>
<td>0-4</td>
</tr>
<tr>
<td>V</td>
<td>126</td>
<td>33</td>
<td>66</td>
<td>0-5</td>
</tr>
<tr>
<td>W</td>
<td>127</td>
<td>34</td>
<td>67</td>
<td>0-6</td>
</tr>
<tr>
<td>X</td>
<td>130</td>
<td>35</td>
<td>70</td>
<td>0-7</td>
</tr>
<tr>
<td>Y</td>
<td>131</td>
<td>36</td>
<td>71</td>
<td>0-8</td>
</tr>
<tr>
<td>Z</td>
<td>132</td>
<td>36</td>
<td>72</td>
<td>0-9</td>
</tr>
<tr>
<td>Symbol</td>
<td>ASCII</td>
<td>Fielddata</td>
<td>Trimmed-ASCII</td>
<td>Card</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-----------</td>
<td>---------------</td>
<td>------</td>
</tr>
<tr>
<td>a</td>
<td>141</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>142</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>c</td>
<td>143</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>d</td>
<td>144</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>e</td>
<td>145</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>f</td>
<td>146</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>g</td>
<td>147</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>h</td>
<td>150</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>i</td>
<td>151</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>j</td>
<td>152</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>k</td>
<td>153</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>l</td>
<td>154</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>m</td>
<td>155</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>n</td>
<td>156</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>o</td>
<td>157</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>p</td>
<td>160</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>q</td>
<td>161</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>r</td>
<td>162</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>s</td>
<td>163</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>t</td>
<td>164</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>u</td>
<td>165</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>v</td>
<td>166</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>w</td>
<td>167</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>x</td>
<td>170</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>y</td>
<td>171</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>z</td>
<td>172</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>