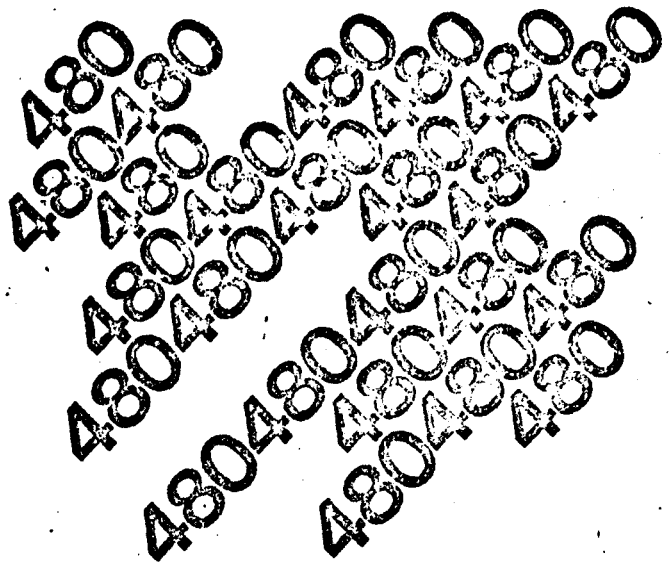

System 480

PROGRAMMERS' REFERENCE MANUAL

(DRAFT)
6-15-73

ENTREX

ENTREX, INC.
168 Middlesex Turnpike
Burlington, Mass. 01830
617-273-0480

This Programmers' Reference Manual constitutes a draft or preliminary version. A final publication is scheduled for September 1, 1973. It is requested that if the reader has any comments or corrections that they be sent to:

Dale Lydigsen
Entrex, Inc.
168 Middlesex Turnpike
Burlington, Massachusetts 01803
(617)-273-0480

TABLE of CONTENTS

SECTION 1
INTRODUCTION

SECTION 2

ARGUMENTS

SECTION 3
STATEMENTS

SECTION 4

SENTENCES

SECTION 1
INTRODUCTION

The Entrex S480 presents extended editing and validating features in the form of a high level Cobol-like language. This language allows a user to perform his own input validation checks and Output Reformatting with Arithmetic, Logical, Output and Program Control Statements. The following features are in addition to the character and field error detection described in "Formatting Techniques" (S-8):

Character Error Detection
- Field Type (Alpha, Numeric, etc.)
- Field Boundary Check

Field Error Detection
- Range Checks
- Check Digit Verification
- Value Table Lookup
- Mandatory Entry/Complete
- Batch Balancing
- Ascendency Check

The extended editing features provide record and batch error detection. The following features allow for a much greater degree of flexibility and may be performed depending on specified conditions:

Record Error Detection
- Complex Range Checks
- Contents Checks
- Arithmetic Crossfoot Checks

Batch Error Detection
- Complex Contents Checks
- Complex Arithmetic Checks
- Batch Totals and Subtotals

Output Formatting is accomplished by means of Editor Formats Programs using powerful output statements. This manual describes record edit, batch edit, and Output Formats and gives examples of each.

This programmers' reference manual is concerned only with the syntactic and operational description of these basic program building blocks. It, therefore, is geared to that individual responsible for the technical

design of the keypunch/unit-record shop, and assumes at least a minimal or cursory familiarity with programming terminology and functions.

DEFINITIONS

A program is made up of one or more English language sentences. Each sentence is made up of a function word or a function word with one or more arguments. Two types of programs are used:

- Error Detection programs
- Output Reformat programs

The major difference between the two is that Error Detection Programs check or validate data, whereas an Output Reformat Program may change and generate data. For instance, an Error Detection Program may compute A x B and compare the results with C, whereas an Output Reformat Program may compute A x B and store the results in C.

RECORD EDIT

A record edit is performed at the end of each record in ENTRY mode. In VERIFY mode it is performed only if the record is changed (if CORRECT key is depressed).

Immediately after an operator releases (manually or automatically) out of a record, the information just keyed is passed through the record edit format. All operations pertain only to the immediate record just released. Data cannot be carried over from record to record.

A record edit is used to edit or check data that can be corrected by the operator at the time it is initially keyed. If the operator cannot correct the data the edit should be performed either at batch edit time or through a separate edit run. This procedure prevents interrupting the operator's keying cadence.

The program will continue either until a RELEASE or STOP statement or until the final program statement is executed. A RELEASE statement returns the system to the ENTRY mode until the end of the next record when it will again retain control. A STOP statement also returns the system to the ENTRY mode however, no further execution of the program will occur.

LIMITATIONS (To prevent system degradation)

1. Data cannot be carried forward from one record to another record. (This can be done at batch edit time.)

2. Totalling of any kind, except within a record, is not possible. A record edit is performed within one record. The system will initialize all variables within the program for each new record.

3. Three variables are allowed per terminal (one variable may be used over and over to perform many operations within a record) since variables may not be used for cumalative operations between records only a few variables are needed.

4. Data cannot be changed. A record edit is used to check data within a record only. Changing data can be done via an output format at output time.

5. The output statement is not valid.

Error Handling

A program may handle data input errors in one or two ways:

1. The program may specify the insertion of an error character into a specified field using the FLAG statement.

2. The program may display an error message using the PAUSE statement.

Program Example

The following example performs content checking, range checking, cross-footing and extension checking.

ENTRE YSTEM 480
EDITOR CODING FORM

Program Name Program Example
Application "Record End" Edit

Originator
Date

PAGE

LINE #

```
1  DECLARE TOTAL, PRODUCT.
2  IF (2) = 33 OR = 35 OR = 42, FLAG (2).
3  IF (3) = Ø. IF (4) > 1ØØ OR < 42, PAUSE
4   'RANGE CHECK ERROR IN FIELD 3'.
5  ADD (1)+(2)+(3) TO TOTAL. IF TOTAL ≠ 6,
6   FLAG (6).
7  IF (1Ø) = Ø. GOTO !NEXT.
8  MOVE (8)x(9) TO PRODUCT. IF PRODUCT ≠ 1Ø
9   PAUSE 'REKEY FIELDS 8, 9 AND 1Ø'.
10 !NEXT RELEASE, AT END STOP.
```

PAGE

LINE #

```
1
2
3
4
5
6
7
8
9
10
```

BATCH EDIT

A Batch Edit is an Error Detection Program executed to check or validate data, and includes:

- .Complex Range Checks
- .Complex Content Checks
- .Complex Arithmetic Crossfooting
- .Complex Batch Totalling/Subtotalling

It is performed upon batch termination in either ENTRY or VERIFY mode or both. Immediately after an operator terminates a batch(not interrupts), the program will perform all specified operations on that batch. Batch End variables are cumalative and therefore batch totalling, subtotalling and record to record checks may be performed. The variables are initialized just once for the entire batch as opposed to once for each record. Initializing a variable means assigning it a value of zero so that the programmer need not initialize the accumulators. The number of variables allowed is 100.

Execution of the program will continue until a STOP statement is encountered It is also possible to perform an edit on a terminated batch using an edit batch function.

Limitations

1. Batch edits are error detection formats therefore data cannot be changed. Changing data only be accomplished by an output formatting program.

Error Handling

Both of the methods described in Record Edit error handling above apply in Batch Edit.

Another error handling method available with batch edit formats is the creation of an error log. The error log associated with the batch being processed is created with the OUTPUT statement. The error log is capable of accepting any information, formatted in any way. During edit program execution OUTPUT statements generate a disk file which is associated with the particular batch in process.

Batch Edit Program Example

The following example performs content checks, range checks, arithmetic crossfoot checks, batch totalling and subtotalling. It creates an error log of those records with range check errors, subtotal errors and lists entered and accumulated batch totals if they do not match.

OUTPUT REFORMATTING

The output reformats reformats a batch and outputs it to any existing output device. All statements are valid for output reformatting. This allows for conditional output of records and data changes based on some predetermined factors.

An Output Reformat Program is executed by a Supervisor request, and follows the same rules that govern Batch Edit Programs.

Limitations

The OUTPUT statement initiates output to a specified output device rather than to a disk resident error log. It will not operate an error log. An error log is generated by a Batch Edit Program only.

Output Reformat Edit Program example

The following example outputs one record for each input record. It also outputs a trailer record including the system maintained date and block count. A record count is kept to serialize output records. An extension is calculated and then punctuated.

ENTREX S. STEM 480
EDITOR CODING FORM

Program Name: Program Example
Application: "Batch End" Edit

Originator: _____
Date: _____

PAGE 1

LINE #	
1	DECLARE TOTAL1, TOTAL2, RECNO.
2	ADD 1 TO RECNO.
3	WHEN PGM 2, GOTO !STOTAL.
4	WHEN PGM 3, GOTO !TOTAL.
5	IF (2) = 8, IF (3) ≠ 1 FLAG (3).
6	IF (4) < 29 OR > 56, PERFORM !RANGE.
7	IF (5) > 600 OR < 250, PERFORM !RANGE.
8	IF (8) / (9) ≠ 10, FLAG (10).
9	ADD (12) TO TOTAL1.
10	ADD (13) TO TOTAL2.

PAGE _____

LINE #	
1	RELEASE.
2	!TOTAL IF TOTAL2 ≠ (2), OUTPUT 'ACCUMULA
3	TED TOTAL =', TOTAL2, 'ENTERED TOTAL',
4	(2).
5	RELEASE.
6	!RAINGE ENTER, OUTPUT 'RANGE ERROR IN REC
7	ORD #', RECNO. EXIT.
8	!STOTAL IF (1) ≠ TOTAL1, OUTPUT 'RECORD#
9	' RECNO 'INCORRECT SUBTOTAL=' TOTAL1.
10	RELEASE.

CODING

The coding sheet used to code Edit, Sort, and Output formats is Entrex Order No. M-10.

The form outlines pages consisting of ten lines of forty characters each as they appear in their respective libraries.

As many pages as needed to accomodate the coding can be used and will represent one page (or screen) in the format libraries.

The statements are coded in a "free-form" style. The language consists of English sentences made up of statements.

Statements are separated by spaces (or a comma and a space). Sentences are ended with a period.

ENTRE: YSTEM 480
EDITOR CODING FORM

Program Name PROGRAM EXAMPLE
Application "OUTPUT REFORMAT" EDIT

Originator
Date

PAGE 1

LINE #

```
1   DECLARE RECNO, PRODUCT.
2   ADD 1 TO RECNO.
3   MOVE (5) * (6) TO PRODUCT.
4   OUTPUT RECNO, (1), (3), (2), (4), PRODUC
5   T, 0000, (10), 'XV2', (9), (8).
6   RELEASE, AT END GOTO !FINLSH.
7   !FINISH, OUTPUT 'TLR1', 0000, <DATE>,
8   'A', <BLK 6>, 0000.
9   OUTPUT <EOF>.
10  STOP.
```

PAGE

LINE #

1
2
3
4
5
6
7
8
9
10

SECTION 2

ARGUMENTS

Within the EDITOR language, there are five legal argument types: field, numeric literal, alphameric literal, variable and arithmetic expression. Within the context of a particular statement, some argument types may be illegal.

FIELD NUMBER

A field number is defined as a number from 1 to 2Ø47 and must be enclosed in parenthesis.

Examples: (5)
(22)
(2Ø13)

Additionally, System/480 provides the facility to handle sub-field or sub-string specifications. The specification is in the form of (n:p-q) where the 'n' represents the field number, and 'p' and 'q' represent the relative positions within a field of the first and last character of the sub-string. When only one character of the field is desired, the form (n:p) would suffice.

This sub-string feature allows the system to now manipulate data down to the character instead of field level. Throughout this document, 'field #' should be taken to mean any sub-field or sub-string within the field number.

Examples: (5:2-4)
(22:3)
(2Ø13:4-12)

NUMERIC LITERAL

A numeric literal is defined as an unsigned or overpunched string of digits comprising an integer value. The maximum legal size of a numeric literal is 14 characters.

The overpunch may appear in any position in the string except the first (due to the possibility of confusing the string with a variable name).

Examples: 9824

9824
+

9824

ALPHAMERIC LITERAL

An alphameric literal is defined as a string of up to 132 characters enclosed in single or double quotation marks. Single quotation marks may appear within a literal enclosed within double quotation marks and vice versa. Any keyboard character is legal within an alphameric literal.

Examples: 'MESSAGE'
"'HELP!'"

Another form of an alphameric literal would be nnn'X' where nnn is the number of times (1-132) the single character in quotes would be repeated. In other words 'ØØØØ' may also be written 4'Ø'. This form of an alphameric literal requires that only one character be specified within quotes. This feature facilitates the filling of blanks or zeroes in an output record for both the user and the system. It also make more explicit the definition of the size of a variable when used in a MOVE statement.

Examples: 5'Z'
120'6'

VARIABLE

A variable is defined as a name associated with a value. The name may be 1-8 characters in length, the first character being A-Z and all following characters being A-Z or Ø-9. The value may be either alphameric or numeric. A numeric variable may contain up to 14 digits including a sign. An Alphameric variable may contain up to 2Ø characters. The type (alpha or numeric) and logical size of a variable is generally defined with a MOVE statement as follows:

.MOVE field/sub-field to variable -
size = size of field/subfield
type = type of field/subfield

.MOVE numeric literal to variable
size = 14
type = numeric

.MOVE alpha literal to variable
size = size of alpha literal
type = alphameric

.MOVE variable to variable
size = size of first variable
type = type of first variable

.MOVE arithmetic expression to variable
size = 14
type = numeric

Examples: X
P131
TOTAL
SUBTOTAL
D1A43BC

ARITHMETIC EXPRESSION

An arithmetic expression is defined as two or more of any of the previously defined argument types connected by any of the following arithmetic operators: + (plus), - (minus), * (times), / (divided by). An arithmetic expression is interpreted and performed simply from left to right (e.g., there is no hierarchy of operators), and it's result is always considered to be 14 numeric positions and right-justified.

Examples: (1) + (2)
RATE * WEIGHT
(21)/90 + TEMP - (6)

SECTION 3

STATEMENTS

With the EDITOR language there are basically two types of statements: 'Action' statements and 'Conditional' statements. Action statements are used to perform arithmetic, editing, output, error signalling and program control functions. Conditional statements are used to perform logical and special tests. The statements shown below are described on the following pages:

A D D

i. Format:

```
                / FIELD    \
               /  LITERAL   \
ADD           <   VARIABLE   >           TO          <VARIABLE> .
               \  ARITH.    /
                \   EXPR.  /
```

ii. Description:

The data defined by the source argument is added to the current contents of the destination argument and the sum replaces the current contents of the destination argument. The contents of the source argument remain undisturbed.

If the logical size of the destination argument is exceeded during this operation, the overflow indicator is turned on and can be sensed and utilized by the program. However, if the physical size of the system accumulators (14 decimal digits) is exceeded at any time during this operation, results of this and future arithmetic operations are unpredictable.

iii. Examples:

ADD (1) TO TOTAL.

ADD (2) + (3) TO CREDITS.

ADD WEIGHT TO RATE.

ADD 1ØØ TO COUNT.

ADD (13:3-5) TO TEMP.

D E C L A R E

i. Format:

DECLARE <VARIABLE> , <VARIABLE> , <VARIABLE> , ...

ii. Description:

The DECLARE statement is used purely as an EDITOR program debugging tool. All variables must be declared within an EDITOR program prior to being used. This minimizes the possibility of an EDITOR programmer referencing invalid variables within his or her program.

iii. Example:

DECLARE TOTAL1, TOTAL2, COUNT . . .

D I V I D E

i. Format:

DIVIDE ⟨ FIELD / LITERAL / VARIABLE / ARITH. EXPR. ⟩ INTO ⟨VARIABLE⟩ .

ii. Description:

The data defined by the source argument is divided into the current contents of the destination argument and the quotient replaces the current contents of the destination argument. The contents of the source argument remain undisturbed.

If the logical size of the destination argument is exceeded during this operation, the overflow indicator is turned on and can be sensed and utilized by the program. However, the physical size of the system accumulators (14 decimal digits) is exceeded at any time during this operation, results of this and future arithmetic operations are unpredictable.

iii. Examples:

```
DIVIDE (1) INTO TOTAL.

DIVIDE (2) + (3) INTO CREDITS.

DIVIDE WEIGHT INTO RATE.

DIVIDE 1ØØ INTO COUNT.

DIVIDE (13:3-5) INTO TEMP.
```

F L A G

i. Format:

FLAG FIELD.

ii. Description:

The FLAG statement is one of two error signalling statements. It is used to insert an S480 error character into the left most position of a specified field, sub-field, or character. This statement would most probably be used in conjunction with a conditional statement.

iii. Example:

FLAG (2).

FLAG (2:3).

FLAG (2:3-5).

G O T O

i. Format:

GOTO !LABEL

ii. Description:

The GOTO statement is a program control statement which is used to modify the sequential execution of EDITOR sentences by allowing a program branching capability. A branch may be executed to anywhere within an EDITOR program with the exception of into a subroutine or out of a subroutine.

iii. Example:

GOTO !TEST

I F

i. Format:

Simple

IF $\langle A \rangle_1$ R $\langle A \rangle_2$

Compound

IF $\langle A \rangle_1$ R $\langle A \rangle_2$ OR IF $\langle A \rangle_3$ R $\langle A \rangle_4$

Compound, implied first argument.

IF $\langle A \rangle_1$ R $\langle A \rangle_2$ OR R $\langle A \rangle_3$

(i.e.: IF $\langle A \rangle_1$ R $\langle A \rangle_2$ OR IF $\langle A \rangle_1$ R $\langle A \rangle_3$)

WHERE: ARGUMENT = FIELD
LITERAL
VARIABLE
ARITH. EXPR

RELATIONSHIP = '=' or '≠' or '>' or '<'

ii. Description:

A. The IF statement is used for performing simple and compound logical comparisons. Comparisons are considered to be either alphameric or numeric. Alphametic comparisons are performed one character at a time from left to right. Numeric comparisons are performed on numeric values which is to say numeric arguments which look differently but have equal values are considered equal. For instance, '-ØØ21' is equal to '-21' and equal to '2J' (least significant digit oversign). When comparing an alphameric argument, the comparison is a numeric one.

B. The results of the IF comparison are used to determine the logical direction of an EDITOR program. If the comparison is true, the next statement is executed; if it is false the next sentence is executed.
Note: Although there is no explicit 'AND' connector, the AND function can be implied by utilizing a series of conditional IF statements.

iii. Example:

```
IF (1) = (2)
IF (4) ≠ 'XY'
IF TOTAL > 1ØØ
IF (1) * (2)= X/3
IF (1) = 'AB' OR IF (2) = 'CD'
IF TEMP < 99 OR >62
IF DATE = TODAY, IF AMT < '45ØØ'
```

M O V E

i. Format:

```
                  ⟨ FIELD    ⟩
                  | LITERAL  |
MOVE              | VARIABLE |          TO        <VARIABLE> .
                  | ARITH.   |
                  ⟩ EXPR.    ⟨
```

ii. Description:

The data defined by the source argument is duplicated in the destination argument, destroying the current contents of the destination argument. The contents of the source argument remain undisturbed.

One of the properties of a MOVE statement is that it is a way of defining the logical size and type (alpha or numeric) of a variable. This is simply done by allowing the variable to take on the attributes of the data being moved to it.

iii. Examples:

```
MOVE (1) TO TOTAL.

MOVE (2) + (3) TO CREDITS.

MOVE WEIGHT TO RATE.

MOVE 1ØØ TO COUNT.

MOVE (13:3-5) TO TEMP.

MOVE 'DEPT. NO.' TO HEADER.
```

M U L T I P L Y

i. Format:

```
MULTIPLY  / FIELD    \     TIMES     <VARIABLE> .
          | LITERAL  |
          | VARIABLE >
          | ARITH.   |
          \   EXPR.  /
```

ii. Description:

The data defined by the source argument is multiplied by the current contents of the destination argument and the product replaces the current contents of the destination argument. The contents of the source argument remain undisturbed.

If the logical size of the destination argument is exceeded during this operation, the overflow indicator is turned on and can be sensed and utilized by the program. However, the physical size of the system accumulators (14 decimal digits) is exceeded at any time during this operation, results of this and future arithmetic operations are unpredictable.

iii. Examples:

MULTIPLY (1) TIMES TOTAL.

MULTIPLY (2) + (3) TIMES CREDITS.

MULTIPLY WEIGHT TIMES RATE.

MULTIPLY 1ØØ TIMES COUNT.

MULTIPLY (13:3-5) TIMES TEMP.

O U T P U T

i. Format:

OUTPUT ⟨FIELD / LITERAL / VARIABLE / CONTROL⟩ , ⟨FIELD / LITERAL / VARIABLE / CONTROL⟩ .... ⟨FIELD / LITERAL / VARIABLE / CONTROL⟩ .

ii. Description:

The OUTPUT statement in the EDITOR allows the creation of a new record consisting of parts of or the entire current record; alpha or numeric literals; alpha or numeric variables; and control functions.

A. Argument Modifiers - may be used to further define any field, variable, or arithmetic expression to allow for character editing. An argument modifier consists of a vertical Bar (|) followed by an edit specification, and immediately follows the argument which it modifies.

The following are legal edit specifications:

|LS - Truncate all leading spaces

|LZ - Truncate all leading zeroes

|RS - Truncate all trailing spaces

|RZ - Truncate all trailing zeroes

|PK - Packed decimal format

|'MASK' - Where MASK is an alphameric literal whose largest size is 20 characters including all MASK characters. If the argument is longer than the MASK, the argument will be truncated and any floating or fixed dollar sign will be lost.

Legal Characters for 'MASK'

_ An underscore in the edit mask is replaced by the corresponding digit from the specified variable.

ii. Description:

Legal Characters for 'MASK' continued

Ø A zero is used to indicate zero suppression. It is placed in the right most position where zero suppression is to take place. It is replaced with the corresponding character from the variable unless that character is a zero.

* An asterisk is used for asterisk protection and zero suppression. It is put in the right most position where asterisk protection is to take place.

$ A dollar sign entered immediately to the left of the zero suppression code or asterisk protection code causes the insertion of a dollar sign in the position to the left of the first significant digit.

A dollar sign in the left most position of the MASK is considered fixed. A fixed dollar sign is placed in the same location each time.

.,Ø Decimal points, commas and blanks are placed in the output field in the relative positions they were written in the MASK unless they are to the left of significant digits.

CR - The characters CR or a minus sign in the last positions of the edit MASK are undisturbed if the sign of the variable field is negative. If the sign is plus the CR or minus sign is blanked out.

N.B., Zero and asterisk are mutually exclusive. If they should both occur the one in the least significant position will take precedence.

B. Control functions - may be used at any time within an OUTPUT statement. They must be enclosed in angle brackets. The following are legal control functions:

<ALL mm-nn>

The ALL function allows the outputting of multiple fields with one control statement.

- if nn is not specified output is from field # mm to the end of the record.
- if neither mm nor nn is specified, the entire record is output.

1./ Description:

Control Functions Continued

<BATCH>

When the control function is encountered, it causes the current batchname to be inserted into the next ten character positions of the output record.

<BLK n>

The BLK function allows the EDITOR to output the physical tape block count. 'n' specifies the size of the field in characters within which the count is output. If n is smaller than the actual count the count will be truncated. If n is zero there is no output regardless of the count.

Note: $\emptyset < n \leq 5$

<COUNT XXXX>

This control function, when encountered indicates that number of characters contained in the current record are to be inserted into the output record. The count may be from 1-4 characters with leading or trailing spaces but no imbedded spaces. The count may be in binary or decimal digits, specified by placing a B or D in the correct positions. Examples of COUNT :

DDSS	Two digit decimal count two trailing spaces
BBSS	Two digit binary count two trailing spaces
DDD	Three digit decimal count
SSBB	Two digit binary count with two leading spaces.

<DATE X>

The DATE function allows EDITOR access to the system Global date which consists of 6 characters in the format in which it was entered by the user. The optional use of 'X' provides the following facilities:

Ø Six-character formatless

X eight-character of format mmXddXyy where 'X' is any legal character except underscore (_) which is used to signify blank.

ii. Description:

Control Functions Continued

<DEFER>

This control function which may appear anywhere within an OUTPUT statement would be used to specify that the arguments within an OUTPUT statement do not make up a complete output record but only a partial one. This would facilitate building just one output record from many input records.

<EOF>

The EOF function allows for the closing on files. If a pad character has been specified it will pad out the current block, and write an industry compatible tape mark. If no pad character is specified it will write a short block followed by an industry compatible tapemark. If no tape drive is available instruction is ignored.

<HEX XX>

The EBCDIC equivalent of 'XX' is generated.

<JOB>

This causes the name of the standard job used to enter the current batch to be inserted into the next eight character positions of the output record.

<LABEL>

This control function, which may appear anywhere within an OUTPUT statement would be used to specify that the current record is a label and not a data record. The occurrence of such a record would cause the output buffer to be handled as if an EOF were encountered, with the exception of writing a tape mark, after which the label would be output regardless of any specified blocking options.

<LF>

The LF function causes a line feed and carriage return to be executed by the printer.

<PGM>

This causes the number of the input format under which the current record was created to be inserted in the output record.

ii. Description:

Control Functions Continued

/<RWND>

The RWND function causes an unconditional rewind of the tape. If no tape is mounted instruction is bypassed.

This function would when executed insert blanks into the output record starting from the current character position up to and including the character position as specified by nnnn.

<SKIP nnnn>

This function would when executed insert blanks into the output record starting from the current character position up to and including the character position as specified by nnnn.

<TOP>

The TOP function allows for the positioning of forms to the first available print line as determined by the carriage control tape. On devices where this function would not be valid it will be ignored.

iii. Examples:

OUTPUT (1), (2), (3), '123', ABC

OUTPUT <ALL> ,<DATE> .

OUTPUT (1), <LF> , (2), <LF> , (3), <TOP> .

OUTPUT <EOF> ,<RWND> .

OUTPUT FILENAM, <DATE>, < BLK#5).

OUTPUT <SKIP 6Ø>, 'TOTAL', TOT |'_ _ _ $. _ _ ØCR'.

OUTPUT '!', <JOB> ,<BATCH> .

OUTPUT <PGM >,<ALL> .

MASK	VARIABLE	RESULT +DATA	RESULT -DATA
'_ _ _0._ _ '	000005	.05	.0N
'_ _$0._ _'	000005	$.05	$.0N
'$_ _0._ _'	000005	$ØØØ.05	$ØØØ.0N
'$_ _*._ _'	000005	$***.05	$***.0N
'_ _ _._ _-'	13560	135.60	135.60-
'_ _ _._ _ CR'	13560	135.60	135.60CR
'_ _ _._ _ØCR'	13560	135.60	135.60ØCR
'$_ _ _0*._ _-'	149363	$*1493.63	$*1493.63-
'$_ _ _*0._ _-'	149363	$Ø1493.63	$Ø1493.63-
'_ _ _,_$0._ _ _-'	1763421	$1,763.421	$1,763.421-
'_ _ _$Ø0._ _CR'	17631	$17Ø6.31	$17Ø6.31 CR
'_ _0_ _ _'	000005	005	00N

Examples of the 'MASK' Edit Specification

P A U S E

i. Format:

PAUSE ALPHAMERIC LITERAL.

ii. Description:

The PAUSE statement is the second of two error signalling statements. It is used to display a specified error message of up to forty characters on the error line of the data/scope terminal. Execution of this statement also causes the error beeper to sound off which remains on until the 'reset' key is struck. This statement would most probably be used in conjunction with a conditional statement. This statement may also be used as a debugging tool when testing new EDITOR programs. This statement may be used without specifying an alphameric literal for display. In this case, the system message 'PAUSE' is displayed.

iii. Example:

```
PAUSE.
PAUSE 'TOTAL IN FIELD 3 INCORRECT'
PAUSE 'ARITHMETIC OVERFLOW, ABORT!'
```

P E R F O R M

i. Format:

```
PERFORM   !LABEL
```

ii. Description:

The PERFORM statement is a program control statement which is used to execute a specified group of sentences from different points within an EDITOR program by allowing a single-level subroutine call capability.

An EDITOR subroutine is a closed subroutine with only one way in and one way out. The entrance and exit (beginning and end) are defined by the special words ENTER and EXIT. A program branch (GOTO) to the entrance of or anywhere within a subroutine is illegal. Also illegal is a program branch out of a subroutine. However, a program branch within a subroutine is legal. In fact, it may be necessary to branch to the subroutine EXIT.

iii. Example:

```
PERFORM  !TEST
:
:
!TEST, ENTER ...EXIT
```

R E L E A S E

i. Format:

RELEASE (AT END, STATEMENT).

ii. Description:

The RELEASE, AT END statement is a program control statement which is used to perform several functions. When executed, this statement will cause program control to release the current record, get the next record and branch to the very beginning of the EDITOR program for further execution. If at the end of a batch (no next record to get), program control will execute the sentence immediately following. If no RELEASE statement is encountered during program execution, the RELEASE function will be performed immediately following execution of the last statement in the progrm. If the AT END option is not employed after the RELEASE and END OF FILE is reached, a STOP statement is implied.

The first time a RELEASE STATEMENT is executed within an EDITOR program, the second record of the batch will be fetched. This implies that the first record of a batch is automatically fetched by the system and ready to be processed at the very start of EDITOR program execution. It is, important to remember this so that the first record is not inadvertantly ignored.

The RELEASE statement is an exception in regard to grammatical rules pertaining to statements and sentences.

iii. Example:

```
RELEASE, AT END STOP.

RELEASE, AT END GOTO !FINISH.


RELEASE.
```

S O R T

i. Format:

SORT ⟨FIELD / LITERAL / VARIABLE / CONTROL⟩ , ⟨FIELD / LITERAL / VARIABLE / CONTROL⟩ ... ⟨FIELD / LITERAL / VARIABLE / CONTROL⟩ .

ii. Description:

The SORT verb is used to generate a sort key to be used during any SORT/MERGE operations. It is identical in format to the OUTPUT statement with the addition of the following two address modifiers.

|AN - Ascending key

|DN - Decending key.

In the absence of either of the above modifiers, ascending is assumed. Note that ascending and descending may be intermixed within any SORT statement.

iii. Examples:

SORT (1:2-4) |DN (5), '1'.

SORT <PGM> , (1) |AN (6:3).

SORT <DATE> ,<ALL 2-4> .

S T O P

i. Format:

STOP.

ii. Description:

The STOP statement is a program control statement which is used to halt execution of an EDITOR program. Should the EDITOR program not contain at least one STOP statement, execution will be halted upon encountering the end of file.

iii. Example:

STOP.

S U B T R A C T

i. Format:

SUBTRACT ⟨ FIELD / LITERAL / VARIABLE / ARITH. EXPR. ⟩ FROM < VARIABLE >.

ii. Description:

The data defined by the source argument is subtracted from the current contents of the destination argument and the difference replaces the current contents of the destination argument. The contents of the source argument remain undisturbed.

If the logical size of the destination argument is exceeded during this operation, the overflow indicator is turned on and can be sensed and utilized by the program. However, the physical size of the system accumulators (14 decimal digits) is exceeded at any time during this operation, results of this and future arithmetic operations are unpredictable.

iii. Examples:

```
SUBTRACT (1) FROM TOTAL.

SUBTRACT (2) +(3) FROM CREDITS.

SUBTRACT WEIGHT FROM RATE.

SUBTRACT 1ØØ FROM COUNT.

SUBTRACT (13:3-5) FROM TEMP.
```

W H E N

i. Format:

WHEN CONDITION

ii. Description:

The WHEN statement functions exactly as does the IF statement except that it tests certain conditions or states within the system as opposed to logical relationships. The conditions tested are:

a. WHEN FLAG

The WHEN FLAG statement is used to test for the presence of the ERROR character anywhere in the current record. This should be used sparingly as it greatly decreases system efficiency due to the necessity of completely scanning the record upon encountering this statement.

b. WHEN OVERFLOW

The WHEN OVERFLOW statement is used for checking for logical arithmetic overflow. It refers to the last arithmetic operation that took place, and applies to arguments which are arithmetic expressions as well as the ADD, SUBTRACT, MULTIPLY AND DIVIDE statements. It is important to note that arithmetic overflow occurs in two different forms:

- Logical Overflow - is when a number within a variable exceeds the number of decimal positions specified by the user in a MOVE statement. That is, of course, when the logical size is less than the physical size. In this case a truncation is performed thereby retaining only the specified amount of decimal digits. The overflow switch is turned on and it is up to the user to test this switch with a 'WHEN OVERFLOW' statement.

- Physical Overflow - is when the system encounters a number it cannot handle (greater than 14 digits). In this case a warning message is displayed on the error line and the user would have the option of aborting or proceeding. Subsequent arithmetic operations are not predictable. This overflow type cannot be tested by 'WHEN OVERFLOW'.

ii. Description:

c. WHEN NOT PGM n (where n = Ø-9)

The WHEN PGM statement is used for testing which input format the current record was or was not entered under. The NOT is optional.

d. WHEN RECORD nn (where Ø >nn ≤ 65ØØØ)

The WHEN RECORD statement is used for testing for the relative number within the batch of the current record. It also may be used to check for the beginning of a batch by specifying record number one.

iii. Examples:

```
WHEN FLAG, GOTO ERROR.

WHEN OVERFLOW, PAUSE 'EXCEEDED 999'.

WHEN PGM 4, GOTO !D04

WHEN NOT PGM 4, ADD 1 TO COUNT.

WHEN RECORD 1, PERFORM !HEADER.
```

SECTION 4

SENTENCES

GRAMMAR

A program sentence may comprise one and only one action statement or one and only one 'action' statement preceded by any number of conditional statements. An action statement may be considered an independent clause, therefore, one or more conditional statements alone do not constitute a valid sentence.

```
Examples:  IF (1) = (2).  (illegal - no action statement)
           IF (1) = (2), ADD (5) TO TOTAL1.
           WHEN PGM2, IF (1) = (2), ADD (5) TO TOTAL1.
```

Note: An exception to the above rules of grammar occurs in the RELEASE statement.

CONDITIONAL EXECUTION

When executing a sentence with conditional statements the following rule applies: When a conditional test proves to be false, program control will branch to the next sentence by-passing all statements up to that sentence, otherwise the very next statement will be executed.

SENTENCE LABELS

Sentences may be preceded with a label so that they may be branched to with a GOTO statement or called with a PERFORM statement. A label must be immediately preceded by exclamation point and may be up to 8 characters in length with the first character being A-Z and all other characters A-Z or Ø-9.

Example: !START, ADD 1 TO COUNT.
!FINISH, STOP.

PUNCTUATION

- Period is used as a sentence delimiter just as in the English language. It is critical that the period be used correctly so that sentences with conditional statements will be executed properly.

- Commas are most commonly used to separate statements. When separating two conditional statements, the comma implicitly defines a logical 'AND'. Commas may also be used to separate sentence labels from sentences, RELEASE from AT END and just about anywhere they make sense. Commas are primarily for program legibility and are not really necessary:

  Example: !TEST, WHEN PGM2, IF (1) = X, ADD (2) TO TOTAL.

  equates to:

  !TEST WHEN PGM2 IF (1) = X ADD (2) TO TOTAL.

- Spaces are used to separate all verbs arguments, connectors and statements that are not otherwise separated by period, comma, or arithmetic operator.