PHILCO

COMPUTER DIVISION
Willow Grove, Pa.

ELECTRONIC DATA PROCESSING SYSTEMS

PHILCO 2000

Input-Output Programming System (IOPS)

TM-18

TM-18

PHILCO 2000

INPUT-OUTPUT PROGRAMMING SYSTEM

(IOPS)

April 1962

PHILCO CORPORATION
A Subsidiary of Ford Motor Company
Computer Division - 3900 Welsh Road
Willow Grove, Pa.

PHILCO 2000

INPUT-OUTPUT PROGRAMMING SYSTEM

(IOPS)

This manual replaces Programming Research and Development Note No. 18 (RD-18) dated September 1, 1961.

TM-18, April 1962

PREFACE

An input-output programming system enables a programmer to transfer information from input-output devices to memory, and from memory to input-output devices. Depending upon the sophistication of the system, he may also be permitted to specify the form in which he desires his data.

This manual discusses IOPS, a comprehensive input-output programming system. It includes information concerning, (1) the media used for the input or output of data, (2) the orders necessary for the transfer of data, and (3) the many forms to which the programmer can have his data converted.

A detailed knowledge of the Philco 2000 input-output operations, although helpful, is not necessary for an understanding of IOPS. However, a knowledge of TAC is assumed.

CONTENTS Page

INTRODUCTION

IOPS is a series of TAC library generators for the Philco 2000 computer system. It provides programmers with a flexible, convenient tool for performing a variety of input-output transfer, editing, and conversion operations. Through the use of easily written statements, the routines inherent in IOPS automatically generate all the coding necessary to perform the specified input-output operation.

When using IOPS for an input-output operation, therefore, the programmer need not be concerned with many of the instructions that would normally have to be included in the program when writing an input-output routine. IOPS also does the "housekeeping", such as keeping track of the words in a block, inserting the proper number of filler characters, and inserting end-of-line and/or end-of-block characters.

IOPS statements are TAC Language statements. As such, symbolic addresses must be written in acceptable TAC form (1-7 alphanumeric characters long).

IOPS reads and produces data tapes and binary tapes. On data tapes a record of information is considered to be (1) a card of input or output, or (2) a printed line of output. On binary tapes a record of information may consist of any number of words or blocks.

IOPS generates magnetic tape macro-instructions and uses the subroutine PROC to process the input-output orders (see R & D Note 25). To call in and initialize PROC, the programmer must include an INIT statement in his program. The INIT must appear before the first logical executable IOPS order. A comprehensive discussion of the INIT statement is presented as Appendix B. IOPS makes no data storage assignments. The programmer must include ASTOR statements for the areas which will buffer the input or output information.

CHAPTER I

ENVIRONMENTAL STATEMENTS

There are three kinds of statements in IOPS -- ENVIRONMENTAL, ORDER, and FORMAT statements. Used together, these three types of statements provide a means of transmitting information, in prescribed forms, between core and I/O (input-output) units.

ENVIRONMENTAL statements supply information about the medium used for input or output, and may be of two types -- IOUNITS or IOUNITSF.

IOUNITS

An IOUNITS (or IOUNITSF) Statement includes parameters which specify the <u>type</u> of I/O unit, the unit used (in the case of magnetic tape), the external form of the data, the address to which the IOPS program will exit if the particular unit requested is inoperable, and the starting addresses of memory areas to or from which information is buffered.

IOUNITS is written in the following general form when indicating an input and an output unit:

LOCATION	COMMAND	ADDRESS
	.	. . .
	IØUNITS	TYPE, UNIT, GROUP SIZE, RECORD SIZE, ERROR ADDRESS, CSA1, CSA2;
		TYPE, UNIT, GROUP SIZE, RECORD SIZE, ERROR ADDRESS, CSA1, CSA2$
	.	. . .

(Note the use of the semicolon to separate complete unit descriptions.)

- The parameter, TYPE, specifies the type of input or output medium used, and may be one of the following forms:

TYPE	INTERPRETATION
DTI	DATA TAPE INPUT Input data is on magnetic tape in card format
DTØ	DATE TAPE OUTPUT Output data is to be recorded on magnetic tape, edited for the printer.
BTI	BINARY TAPE INPUT Binary input is on magnetic tape in IOPS format (see Appendix C)
BTØ	BINARY TAPE OUTPUT Binary output is to be recorded on magnetic tape in IOPS format
BTIØ (or BTØI)	BINARY TAPE INPUT OR OUTPUT Binary input and output is to be read from and recorded on magnetic tape in IOPS format
PTI	PAPER TAPE INPUT Input data is on paper tape in card format
PCØ	PUNCHED-CARD OUTPUT Output data is to be recorded on magnetic tape edited for the Card Punch

(Note that only binary tapes are used for both input and output.)

- UNIT identifies the particular input-output unit used, and may be any of the following:

UNIT	INTERPRETATION
SYMBØL	The symbolic address of a word whose contents serve to identify the magnetic tape unit (sum of the 4 bits at T15 and T23 is the unit number).
nT	The magnetic tape connected to channel n of the Input-Output Processor. (n is an integer 0-16.) If $n > 16$, nT will be treated as a symbol.
PT	The paper tape system connected to the paper tape channel of the central computer.

- GROUP SIZE is the number of records contained in a block of information on magnetic tape. The most frequently used Group Size is 12 cards per block.

- RECORD SIZE is the number of computer words per record (usually 10 words per card).

  Group and Record Size Parameters are used only with input data tapes (DTI), punched-card output tapes (PCØ), or input paper tape (PTI). These two parameters need not be included in an IOUNITS statement which describes either an output data tape to be printed (DTØ) or binary tapes, but the commas which would have separated them must be retained.

- ERROR ADDRESS is an absolute or symbolic address to which a jump is made when a non-recoverable tape error exists. (Refer to page 50.)

  The Error Address Parameter may be omitted from an IOUNITS statement when used with any Type Parameter other than PTI. When thus omitted, the error address is that specified in the PROC INIT statement. (See Appendix B.)

- CSA1 denotes the core starting address of a block (128 words) that is to be used by IOPS as an input or output buffer area.

- CSA2 denotes the core starting address of a second block to be reserved for input or output. This parameter is optional, and need not be included in an IOUNITS statement. If CSA2 is included and is not equal to CSA1, then double buffering will occur. This option of double buffering is not available with binary tapes or with paper tape input.

  (Note: ASTOR statements must be included in the program to reserve storage for these buffer areas.)

Example:

LOCATION	COMMAND	ADDRESS
	. IØUNITS .	. . . DTI, 11T, 12, 10, ERRIN, BUFF1, BUFF2; DTØ, 12T , , , ERRØUT, BUFF3, BUFF4$ . . .

(Spaces preceding a parameter are ignored.)

Only one IOUNITS statement may be used in a program or a compilation error will occur. The IOUNITS statement may be placed anywhere in a program.

The following example further illustrates the use of the IOUNITS statement:

```
(1)  IØUNITS  DTI, 9T, 12, 10 ERRIN, BUF1, BUF2;
(2)           DTØ, 10T, , , ERRØUT, BUF3, BUF4;
(3)           PCØ, 11T, 16, 8, ERRØUT, BUF5, BUF6;
(4)           BTØ, 12T, , , ERRØUT, BUF7;
(5)           BTI, 13T, , , ERRIN, BUF8;
(6)           BTIØ, 14T, , , , BUF9;
(7)           PTI, PT, 12, 10, ERRIN, BUF10 $
```

- Line (1) describes tape unit 9 as an input data tape. There are 12 records per block and 10 words to a record; the system error address is ERRIN, and the input data is to be double buffered.

- Line (2) describes tape unit 10 as an output data tape to be edited for an off-line High Speed Printer. The system error address is ERROUT, and the output data is to be double buffered.

- Line (3) describes tape unit 11 as a punched-card output tape to be edited for off-line punching (16 cards per block, 8 words per card). ERROUT is the system error address, and the output data is to be double buffered.

- Line (4) describes tape unit 12 as a binary output tape. The system error address is ERROUT, and the output data is to be single buffered.

- Line (5) describes tape unit 13 as a binary input tape. The information which was recorded on this tape as a result of a WRITEBT order is single buffered. ERRIN is the system error address.

- Line (6) describes tape unit 14 as a binary input or output tape. Information written on or read from this tape is single buffered. The system error address is that appearing in the INIT statement.

- Line (7) describes the Paper Tape Channel as an input channel. The paper tape format is made to resemble card to tape format, 12 records per block, 10 words per record. ERRIN is the system error address, and the information is to be single buffered.

IØUNITSF

An IOUNITSF statement is normally used in FORTRAN programs that are being prepared for ALTAC compilation. IOUNITSF is written in the same form and with the same parameters as the IOUNITS statement.

An IOUNITS statement produces coding which assumes that the programmer is supplying both v (vertical format) and s (data select) control characters for every line that is to be printed; IOUNITSF automatically produces an s character of 0 and makes the following changes in the v character:

PRINTER PROGRAM-CONTROL CHARACTER		V	S	MEANING
0	becomes	1 1	0 0	Double space
△ (a space symbol)	becomes	1	0	Single space
1	becomes	7	0	Skip to top of page
+	becomes	0	0	No space
A filler character	becomes	1	0	Single space
Anything else	becomes	7	0	Skip to top of page

There is no other difference between IOUNITS and IOUNITSF.

CHAPTER 2

ORDER STATEMENTS

ORDER statements specify the transmission of information and usually contain a list of the quantities to be transmitted. The list also indicates the sequence in which the quantities are to be transmitted.

ORDER STATEMENTS PARAMETERS

The following shows a general form of an ORDER statement:

LOCATION	COMMAND	ADDRESS
	.	. . .
	ORDER	UNIT;SYMBØL;LIST$
	.	. . .

Note that the statement parts (parameters) are separated by semicolons in the address field.

PARAMETER	ACTION	EXAMPLES
ORDER	This parameter specifies the operation to be performed with a particular I/O (input-output) unit.	READ WRITE READBT WRITEBT REWIND
UNIT	For an explanation of the UNIT parameter, see IOUNITS, page 4.	K 7T LAMBDA

PARAMETER	ACTION	EXAMPLES
SYMBØL	This is an address identifying the FØRMIN or FØRMØUT statement associated with the I/Ø order. (For an explanation of FØRMIN and FORMOUT, see Chapter 3.)	BETA NAME.TAG
LIST	LIST refers to a sequence of locations which contain or will contain the information that will be transmitted.	A;DELTA;GAMMA

LIST

A parameter or element of a LIST is any legitimate decimal or symbolic TAC address composed of up to 40 characters. A LIST can be comprised of any number of these elements, each separated by semicolons.

An element of a LIST may be written in any one of the following six basic forms, indicating Index Register Modification (IRM) or Non-Index Register Modification (NIRM).

BASIC FORMS	EXAMPLES	EXPLANATION
NIRM	ALPHA or 1000	This element refers to the contents of symbolic memory location ALPHA, or to decimal location 1000.

BASIC FORMS	EXAMPLES	EXPLANATION
NIRM (Decimal Integer)	ALPHA+7(5)	This element refers to the contents of 5 consecutive locations starting at symbolic memory location ALPHA+7.
NIRM (Symbol)	1000(ALPHA)	Symbolic memory location ALPHA must contain in bits 1-15, the number of consecutive locations to be processed, starting at decimal location 1000.
IRM	0,4	This element refers to the contents of the memory location whose address is contained in Index Register 4.
IRM (Decimal Integer)	3,4(2)	The effective memory address is the contents of Index Register 4 incremented by 3. This element therefore refers to the contents of two consecutive memory locations starting at the effective memory address.
IRM (Symbol)	RHO,3X(Q.DEL)	The effective address RHO,3X is the starting address of the number of locations (defined by bits 1-15 in location Q.DEL) to be processed.

CLASSES OF ORDER STATEMENTS

ORDER statements may be divided into three classes:

- DATA TAPE ORDERS: Order Statements specifying transmission and conversion of information.
- BINARY TAPE ORDERS: Order Statements specifying transmission only.
- Orders dealing essentially with the manipulation of tapes.

The general form of these orders is as follows:

	GENERAL FORM	
	COMMAND	ADDRESS
CLASS I	ORDER	UNIT;SYMBØL;LIST $
CLASS II	ORDER	UNIT;LIST $
CLASS III	ORDER	UNIT $

Note that the SYMBØL parameter is omitted from CLASS II orders because no conversion of binary information is required. Also note that both the SYMBØL and LIST parameters are omitted from CLASS III orders because neither transmission (in most cases) nor conversion of data is required.

The ORDERS available with IOPS are:

CLASS I or DATA TAPE ORDERS	CLASS II or BINARY TAPE ORDERS	CLASS III or TAPE MANIPULATION ORDERS
READ WRITE	READBT WRITEBT	BACKUP ENDFILE RUNØUT REWIND REWINDLØ

CLASS I ORDERS

Class I Orders concern data tapes and provide for the transfer of coded information. All three parameters (Unit, Symbol and List) are usually included in these order statements.

READ

The READ order is used only with data tapes; information on these tapes is in card format, x words per card, y cards per block. Each card is considered as one record, and each READ order starts a new record.

An example of a READ statement follows:

LOCATION	COMMAND	ADDRESS
	.	. . .
	READ	10T;SYMBØL;ALPHA;BETA(3);GAMMA(2)$
	.	. . .

According to the above statement, information is to be read from magnetic tape 10, converted according to the FORMIN statement identified as SYMBØL, and stored in six memory locations (in location ALPHA, in <u>three</u> consecutive locations starting at location BETA, and in <u>two</u> consecutive locations starting at location GAMMA).

WRITE

WRITE statements follow the same general form as READ statements. For example, if information in symbolic memory locations ALPHA, BETA, BETA+1, BETA+2, GAMMA and GAMMA+1 is to be written (according to FORMOUT statement K) on the data tape connected to channel 5 of the Input-Output Processor, the ORDER statement could be written as follows:

LOCATION	COMMAND	ADDRESS
	.	. . .
	WRITE	5T;K;ALPHA;BETA(3),GAMMA(2)$
	.	. . .

This WRITE order could also have been written as:

```
WRITE   5T;K;ALPHA;BETA(XRAY);GAMMA(YØKE)$
```

where XRAY and YOKE are the symbolic addresses of two words containing the quantities 3 and 2, respectively, in bits 1-15.

The elements of a LIST can also be index register modified. For example, if the absolute address assigned to the symbol ZEBRA, is in index register 7 and the contents of three sequential locations starting with location ZEBRA+9 are to be converted and transferred to magnetic tape 5, the ORDER statement should be of the form:

LOCATION	COMMAND	ADDRESS
	.	. . .
	WRITE	5T;K;9,7(3) $
	.	. . .

If the index register counter has been set to one, the contents of the index register would be incremented by one. (Index Registers 1 and 2 should not be used with LIST entries.)

<u>CLASS II ORDERS</u> Orders in this class provide an intermediate storage facility involving only binary tapes. Since no conversion is required with these orders, only two parameters (Unit and List) are specified in the address field.

READBT Because a READBT order only accepts information in IOPS format (see page 45), this order should only be given for a tape written with the WRITEBT order, which insures that information written on that tape is in IOPS format.

Information on a binary tape is transmitted a record at a time and each record starts a new block. The size of the binary record is of variable length because it is determined by the number of words specified in the LIST of a WRITEBT statement. After all or a part of a record is read, the tape is positioned at the beginning of the next record.

The following is an example of a READBT order:

LOCATION	COMMAND	ADDRESS
	.	. . .
	READBT	11T;ALPHA(3);BETA(DELTA);5,6(YØKE)$
	.	. . .

Assume that the quantities 2 and 4 are in bits 1-15 of symbolic memory locations DELTA and YØKE, respectively, and that index register 6 contains the absolute address assigned to the symbol TAG. Then, according to the above READBT statement, the binary information that is read from magnetic tape 11 would be stored in 3 consecutive locations starting with location ALPHA, in 2 consecutive locations starting with location BETA, and in 4 consecutive locations starting with location TAG+5.

After a READBT operation is completed, the tape is positioned at the beginning of the next record (which, in IOPS, starts a new block). Attempting to read more words than exist in the record will result in an IOPS error.

WRITEBT

A WRITEBT order causes binary information to be recorded on magnetic tape in IOPS format.

The following is an example of a WRITEBT statement:

LOCATION	COMMAND	ADDRESS
	.	. . .
	WRITEBT	9T;ALPHA(1000);BETA(500);GAMMA(985)$
	.	. . .

According to the above statement, one record of 2485 words (1000 words starting from location ALPHA, followed by 500 words starting from location BETA, followed by 985 words starting from location GAMMA) would be written on tape 9.

The larger the LIST of binary information to be transferred the more efficient will be the execution of the pertinent section of the program.

CLASS III ORDERS

Except for the ENDFILE and RUNOUT statements, statements or orders in Class III deal only with the manipulation of tapes. They specify neither transmission nor conversion of information.

BACKUP

The BACKUP order refers to binary records only. BACKUP orders are used to space backwards over a binary record and are written as follows:

LOCATION	COMMAND	ADDRESS
	.	. . .
	BACKUP	13T $
	.	. . .

The above order causes the tape specified by the Unit Parameter (13) to be backspaced one record, so that the next block to be read forward will be the beginning of the record just spaced over. If the tape is not in binary format, the tape will be moved backwards until the beginning of tape is reached or until a block appears to be in the proper binary format.

ENDFILE

The ENDFILE statement is used with both output data and binary tapes. When an ENDFILE statement is used with an output data tape, it causes information that remains to be written on the tape indicated by the Unit Parameter, to be edited in the form specified by the Type Parameter of the IOUNITS (or IOUNITSF) statement. ENDFILE also causes an additional block containing an absolute data-select stop character to be written directly after the last data block on that tape.

If printer output is specified by the Type parameter in the IOUNITS (or IOUNITSF) statement, the absolute stop character is determined by the data select character of the previous block. For example, if the data select character for the previous block was $06_{(8)}$, the absolute stop would be $66_{(8)}$ and the printer would halt only on data select 6.

If punched-card output is specified by the Type parameter in the IOUNITS (or IOUNITSF) statement, the additional record will contain the absolute stop character (Octal 72) and a series of blank cards, since the data select for PCØ is 10. (See page 19.)

When ENDFILE is used with a binary tape, a special record with an end-of-file control word (see Appendix C) is written after the last binary record. This special record can be passed over in a forward direction by a READBT order with no LIST, and in the reverse direction by a BACKUP order.

The following is a representative example of an ENDFILE statement:

LOCATION	COMMAND	ADDRESS
	.	. . .
	IØUNITS	DTØ,12T,,,,BLKA$
	.	. . .
	ENDFILE	12T $
	.	. . .

According to the example above, ENDFILE will edit (for the Printer) the data remaining in the buffer area, and will write the block on magnetic tape 12. Immediately after this last data block, an additional block containing an absolute stop character (which is based on the data select character of the previous block) is written.

RUNOUT

The RUNOUT order refers to data tapes only and has no effect on binary tapes. Although RUNOUT is mainly used with output data tapes to ensure the writing of partly filled output buffer blocks, it can also be used with input data tapes.

When a RUNOUT order is given to an input data tape (DTI), it so positions the tape that a READ order following this RUNOUT will force a new block of information to be brought into memory; any unprocessed record in the previous block will be lost. If the input tape is double-buffered, RUNOUT causes the tape to be backspaced one block.

When a RUNOUT order is given to an output data tape (DTØ or PCØ), the information which partly (or completely) fills the buffer block is edited and written on tape. Any subsequent WRITE order will automatically start a new block.

RUNOUT performs the same output operation as ENDFILE, except that with RUNOUT no terminal block (with a stop character) is added.

RUNOUT will complete the editing of the buffer block for the card punch and will write the block on tape 9, when written as follows:

LOCATION	COMMAND	ADDRESS
	.	. . .
	IØUNITS	PCØ,9T,12,10,BLK2 $
	.	. . .
	RUNØUT	9T $
	.	. . .

If the Unit parameter is omitted in a RUNOUT statement, all output blocks will be transmitted; input tapes will be positioned or backspaced as previously indicated. The following is an example of such an order:

LOCATION	COMMAND	ADDRESS
	.	. . .
	RUNØUT	$
	.	. . .

REWIND or REWINDLO

A REWIND or REWINDLO order causes the magnetic tape specified by the Unit Parameter in its address field to be rewound (with or without lockout).

Written as follows,

LOCATION	COMMAND	ADDRESS
	.	. . .
	REWIND	12T $
	.	. . .

REWIND causes magnetic tape 12 to be rewound to the beginning-of-tape position, regardless of its previous position.

With output tapes, it is mandatory that the programmer issue an ENDFILE or RUNOUT order prior to the REWIND or REWINDLO order. If this is not done, data will be lost, or subsequent use of the tape will cause an IOPS error condition.

CHAPTER 3

FORMAT STATEMENTS

FORMAT statements specify the form in which data is desired and the type of conversion to be performed. There are two types of FORMAT statements -- FORMIN and FORMOUT statements.

FORMIN statements are used with READ orders; FORMOUT statements with WRITE orders.

FORMAT statements are not executed, and may be placed anywhere in the program. They specify the arrangement of data by providing field specification information (such as field widths, the number of decimal digits after the decimal point, etc.), and data conversion information (such as integer, fixed-, or floating-point conversions).

FORMIN and FORMOUT statements are of the following general form:

LOCATION	COMMAND	ADDRESS
	°	° ° °
SYMBØL	FØRMIN	field descriptors and modifiers $
	°	° ° °
SYMBØL	FØRMØUT	field descriptors and modifiers $
	°	° ° °

Each FORMIN or FORMOUT statement defines a record or a sequence of records to be transmitted.

In a FORMOUT statement, Data Select(s) and Vertical Format(v) control characters must be specified for a data tape described as DTØ in an IOUNITS statement. However, only the v character need be specified with DTØ in an IOUNITSF statement (see page 7). With PCØ, neither v nor s characters should be specified in the FORMOUT statement, since the s is automatically set to 10 by IOPS, and the v does not apply.

FIELD DESCRIPTORS

Field Descriptors and Modifiers are presented as elements of a FORMIN or FORMOUT statement and are separated by commas, semicolons, or slashes.

Field Descriptors define the size of external fields and the nature of the data within them as well as the types of conversions to be performed. Modifiers, as their name implies, modify descriptors, thus providing additional flexibility in data conversion.

IOPS descriptors include: the Iw, Dw.d, Fw.d, Ew.d, Aw, Ww, Øw, Cw, nH, and Sw descriptors.

DESCRIPTOR	EXTERNAL FORM	INTERNAL FORM
Iw	DECIMAL INTEGER	BINARY INTEGER
Dw or Dw.d	DECIMAL INTEGER or FIXED-POINT DECIMAL	FIXED-POINT BINARY
Fw or Fw.d	DECIMAL INTEGER or FIXED-POINT DECIMAL	FLOATING-POINT BINARY
Ew or Ew.d	FIXED-POINT DECIMAL or FLOATING-POINT DECIMAL	FLOATING-POINT BINARY
Aw	ALPHANUMERIC	ALPHANUMERIC
Ww	ALPHANUMERIC	ALPHANUMERIC
Øw	OCTAL	BINARY-CODED OCTAL
Cw	HEXADECIMAL	BINARY-CODED HEXADECIMAL
nH	ALPHANUMERIC	A FIXED FIELD IN A FORMOUT STATEMENT.
Sw	Number of spaces to be inserted during output, or number of characters to be skipped during input.	

Iw

- When used in a FORMIN statement, the Iw descriptor permits conversion from decimal integer to binary integer form.

Iw causes a field of information w characters wide to be converted to a binary integer. (w must be an unsigned decimal integer.) This converted quantity is then stored in a memory location specified in the LIST of the ORDER statement. A field may be as large as 14 decimal digits. Its binary equivalent occupies a full computer word scaled 47T, unless it is modified by a position factor (see Modifiers).

The w count includes spaces, plus and minus* signs, and any number of decimal digits; the use of any other character will result in an IOPS system error. Leading spaces (those before the first non-space character) are treated as spaces. All other spaces are regarded as zeros. When a sign is punched, it must precede the number, although spaces may separate it from the number.

Examples:

EXTERNAL FORM	Iw	DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER
256	I3	256
-256	I4	-256
+256	I4	+256
-ΔΔΔ256	I7	-256
ΔΔ367	I5	+367
ΔΔ-367	I6	-367
ΔΔ-ΔΔΔ367	I9	-367
ΔΔ36Δ7ΔΔ	I8	+360700
ΔΔ+ΔΔΔ16Δ	I9	+160

(In all examples, spaces are indicated by Δ 's.)

* In an input field the 11 punch and the 8-4 multiple punch are recognized as a minus sign.

- When used in a FORMOUT statement, the Iw descriptor permits conversion from binary integer to decimal integer form.

  Iw causes the contents of a memory location (which is assumed to be scaled 47T unless an nT modifier is used) specified in the LIST of the ORDER statement to be converted to a decimal integer and written in a field _w_ characters wide. _w_ can be as large as 120, and includes spaces, sign, and any number of decimal digits.

  The least significant digit will be positioned at the right end of the field. Leading zeros and plus signs are suppressed (i.e., replaced with spaces) unless a Zero or Sign Modifier (see page 36) is used to force their presence.

  If _w_ is smaller than the number of digits in the quantity to be transmitted, only the _w_ rightmost digits are printed.

Examples:

DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER	Iw	EXTERNAL FORM
+256	I3	256
+256	I5	ΔΔ256
-367	I4	-367
-367	I6	ΔΔ-367
-367	I3	367
+256	I2	56

Dw.d or Dw

- When used in a FORMIN statement, the Dw.d descriptor permits conversion from fixed-point decimal or decimal integer form to fixed-point binary form.

Dw.d causes a decimal field _w_ characters wide (including any fractional part) to be converted to a fixed-point binary number. (Both _w_ and _d_ must be unsigned integers.) The right-most _d_ digits of the field is the fractional part unless a decimal point is punched in the input field, in which case the decimal point takes precedence over the _d_ specification. The number may have as many as 14 decimal digits, with its binary equivalent occupying a full computer word scaled 47T, unless it is modified by a position factor. (See Modifiers.) The descriptor may be written without the _d_ specification (Dw); when this is done, d = 0 is assumed.

The _w_ count includes spaces, plus or minus sign, decimal point, and any number of decimal digits; use of any other character will result in an IOPS system error. Leading spaces are ignored, all other space characters are regarded as zeros. When a sign is punched, it must precede the number, although spaces may separate it from the number.

When a fixed-point decimal field is read into memory according to an unmodified* Dw.d descriptor, the field is scaled 47T in a word and the fractional part of a field, if any, is lost.

The following examples show the loss of the fractional part when Dw.d is unmodified. Other examples (page 34) show the fractional part preserved when nT _is_ used with the Dw.d descriptor.

Examples:

EXTERNAL FORM	Dw.d	DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER (Scaled 47T)
6541	D4.2	65
6541	D4	6541
-65.41	D6.0	-65
-65.41	D6.4	-65
ΔΔ-Δ123	D7.4	0
ΔΔ+Δ12Δ34Δ	D10.2	1203

* A Terminal Position Factor, nT, is normally used with the Dw.d descriptor (see Modifiers, page 34).

- When used in a FORMOUT statement, the Dw.d descriptor permits conversion from fixed-point binary to fixed-point decimal form.

  Dw.d causes the contents of a memory location specified in the LIST of the ORDER statement to be converted to a decimal quantity, <u>w</u> characters wide, with <u>d</u> decimal places to the right of the decimal point. (If <u>d</u> is omitted, d = 0 is assumed.)

  Leading zeros and plus signs are suppressed, unless the Zero or Sign Modifiers are used to force their presence.

  The following examples show the effect of an <u>unmodified</u> Dw.d output descriptor.

  <u>Examples</u>:

DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER (Scaled 45T)	Dw.d	EXTERNAL FORM (Scaled 47T)
+6.25	D5.2	25.00
13.5	D4	Δ54.
-5.25	D6.2	-21.00

  Other examples (page 34) show Dw.d properly modified.

Fw.d
or
Fw

- When used in a FORMIN statement, the Fw.d descriptor permits conversion from fixed-point decimal or decimal integer form to floating-point binary form.

  Fw.d causes a field of data <u>w</u> characters wide (including any fractional part) to be converted to a floating-point binary number and stored in a memory location specified in the LIST of the ORDER statement. The right-most <u>d</u> digits of the field is the fractional part unless a decimal point is punched in the input field, in which case, the decimal point takes precedence over the <u>d</u> specification. The descriptor may be written without the <u>d</u> specification (Fw), in which case, d = 0 is assumed. Both <u>w</u> and <u>d</u> must be unsigned integers.

w may indicate a field of any width, but because only a maximum of 10 places of accuracy can be maintained in a floating-point word, only the first 10 digits in the field will be significant, the remaining digits will be treated as if they were zeros.

The w count includes spaces, plus or minus sign, decimal point, and any number of decimal digits. Use of any other character will normally result in an IOPS system error.* Leading spaces are ignored, all other spaces are interpreted as zeros.

When a sign is punched, it must precede the number, although spaces may separate it from the number.

Examples:

EXTERNAL FORM	Fw.d	DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER
-237	F4	-237.
-237	F4.2	-2.37
53.61	F5.1	53.61
Δ-3Δ480	F7.2	-304.80
-39	F3.3	-.039

- When used in a FORMOUT statement, the Fw.d descriptor permits conversion from floating-point binary form to fixed-point decimal form.

The floating-point quantity in the memory location specified in the LIST is converted to a fixed-point quantity that is rounded to the d^th digit after the decimal point. It is the w right-most characters of this rounded quantity that are printed or punched. When d is omitted, d = 0 is assumed. The w count includes sign, decimal point, and any number of decimal digits. Leading spaces are also included to make up the w count, where necessary.

* If the input field appears in a form for the E conversion (see page 26), the field will be handled as a floating decimal number.

If the quantity reserved for output is less than 1 and if $w > (d+1)$, a zero will precede the decimal point. Plus signs are suppressed unless a Sign Modifier is used to force their presence.

Examples:

DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER	Fw.d	EXTERNAL FORM
.73	F4.2	0.73
-.329	F4.2	0.33
-.329	F7.4	-0.3290
-.329	F3.4	290
-.329	F8.4	Δ-0.3290

With the Fw.d descriptor there can be as many as 10 significant digits of output. If more than 10 digits are needed to express the number, the additional digits will be zeros.

Ew.d
or
Ew

● When used in a FORMIN statement, the Ew.d descriptor permits conversion from fixed-point decimal or floating-point decimal (mantissa-exponent) form to floating-point binary form.

The acceptable mantissa-exponent forms of floating-point decimal fields are illustrated below:

±mantissa±exponent
±mantissaE±exponent
±mantissaEexponent

Although the mantissa may be of any magnitude, only the first ten decimal digits will be significant. The exponent may be any integer in the range -600 to 600.

Ew.d causes a field of data w characters wide to be converted to a floating-point binary number and stored in a memory location specified in the LIST. The w count includes blanks, E, plus or minus sign, decimal point, and any number of decimal digits; use of any other character will result in an IOPS system error.

The right-most d digits of the mantissa is the fractional part, unless a decimal point is punched in the field, in which case, the decimal point takes precedence over the d specification. The descriptor may be written without the d specification (Ew); when this is done, d = 0 is assumed. Both w and d must be unsigned decimal integers.

Examples:

EXTERNAL FORM	Ew.d	DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER
-1.62E+5	E8	$-.162\times10^{6}$
-162E5	E6.2	$-.162\times10^{6}$
-162+5	E6.2	$-.162\times10^{6}$
1567E-2	E7.1	$.1567\times10$
15E-4	E5	$.15\times10^{-2}$
15E0	E4	$.15\times10^{2}$
Δ1Δ5ΔΔE3	E8	$.105\times10^{8}$

- When used in a FORMOUT statement, the Ew.d descriptor permits conversion from floating-point binary form to floating-point decimal (mantissa-exponent) form.

The general form of the output field if an Exponent Modifier (see page 35) is not used, is as follows:

±0.xxxxxxxxxx±eee

The mantissa will have a zero to the left of the decimal point when $w > (5+d)$. For example, if w is 10 and d is 4, the output field will be of the form 0.xxxx±eee. The exponent part, whose output form is always ±eee, is also included in the w count.

The floating-point quantity in the memory location specified in the LIST is first converted to decimal mantissa-exponent form. The mantissa is then rounded to the dth digit after the decimal point, and the w right-most characters of the rounded mantissa-exponent quantity are printed or punched. (Both w and d are unsigned decimal integers.) When d is omitted, a d = 0 is assumed. The w count includes sign, decimal point, any number of decimal digits, and the signed exponent. Leading spaces are included to make up the w count where necessary. Plus signs are suppressed unless a Sign Modifier is used to force their presence.

With the Ew.d descriptor there can be as many as 10 significant digits of output. If more than 10 digits are needed to express the mantissa, the additional digits will be zeros.

Examples:

DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER	Ew.d	EXTERNAL FORM
$+.538 \times 10^{3}$	E9.3	0.538+003
$+.538 \times 10^{3}$	E9.2	Δ0.54+003
$-.538 \times 10^{3}$	E13.4	ΔΔ-0.5380+003
$+.7031 \times 10^{-4}$	E11.5	0.70310-004
$+.6391 \times 10^{5}$	E14.7	Δ0.6391000+005
$-.6391 \times 10^{5}$	E5.3	9+005
$-.538 \times 10^{3}$	E6.5	00+003

(The last two examples above, although not completely representative, are included to enhance the reader's understanding of the editing process.)

Aw

- When used in a FORMIN statement, Aw describes a field of alphanumeric-coded characters which is to be transferred to a word in memory.

Aw causes a field w characters wide to be transferred and stored left-justified with trailing blanks in a memory location specified in the LIST of an ORDER statement. All characters are acceptable to the Aw descriptor. w should not be greater than 8 because a word in memory can only accomodate a maximum of eight alphanumeric characters.

Examples:

EXTERNAL FORM	Aw	ALPHANUMERIC REPRESENTATION of INTERNAL BINARY QUANTITY
ΔΔ$1.25	A7	ΔΔ$1.25Δ
ΔΔΔVALUE	A8	ΔΔΔVALUE
DATA3	A5	DATA3ΔΔΔ

- When used in a FORMOUT statement, Aw describes the field that is to contain an arrangement of alphanumeric-coded characters.

  According to this descriptor, the left-most w characters of a location specified in the LIST of the ORDER statement will be transferred to the output record.

  Examples:

ALPHANUMERIC REPRESENTATION of INTERNAL BINARY QUANTITY	Aw	EXTERNAL FORM
ΔΔ$1.25Δ	A6	ΔΔ$1.2
ΔΔΔΔABLE	A8	ΔΔΔΔABLE
DATA3ΔΔΔ	A4	DATA

Ww

- When used in a FORMIN statement, Ww describes a field of alphanumeric-coded characters which is to be transferred to a word in memory.

  Ww causes a field w characters wide to be transferred and stored right-justified with leading zeros in a memory location specified in the LIST of an ORDER statement. All characters are acceptable to the Ww descriptor. As is the case with the Aw descriptor, the width of a field should not exceed 8 characters.

  Examples:

EXTERNAL FORM	Ww	ALPHANUMERIC REPRESENTATION of INTERNAL BINARY QUANTITY
Δ$.25Δ	W6	00Δ$.25Δ
ΔΔΔVALUE	W8	ΔΔΔVALUE
DATA3	W5	000DATA3

- When used in a FORMOUT statement, Ww describes the field that is to contain an arrangement of alphanumeric-coded characters.

  According to this descriptor, the right-most w characters of a location specified in the LIST of the ORDER statement will be transferred to the output record.

Examples:

ALPHANUMERIC REPRESENTATION of INTERNAL BINARY QUANTITY	Ww	EXTERNAL FORM
Δ$1.25ΔΔ	W4	25ΔΔ
ΔΔΔΔABLE	W8	ΔΔΔΔABLE
ΔΔΔΔINFO	W4	INFO

Øw

- When used in a FORMIN statement, Øw describes a field of octal characters to be transferred to a word in memory.

Øw causes the right-most three bits of w six-bit characters in the input field to be combined and stored right-justified (terminating at the 47th bit position, or at the n^{th} bit position if an nT modifier is used) in a word specified in the LIST of the ORDER statement. For example, the letter "G" (binary 010111) would be converted to octal 7 (binary 111).

All characters are acceptable to the Øw descriptor; however, as illustrated in the preceding example, only the decimal digits 0-7 can be unmistakeably represented with this descriptor.

w should not be greater than 16, since 16 is the maximum number of octal digits that can be represented in a 48-bit computer word. Blank characters are included in the w count, but are ignored in the conversion process.

Examples:

EXTERNAL FORM	Øw	INTERNAL (48 Bits) FORM
2673	Ø4	0......010 110 111 011
7065	Ø4	0......111 000 110 101
6GH	Ø3	0......... 110 111 000
6GΔ	Ø3	0............. 110 111
19	Ø2	0............. 001 001

- When used in a FORMOUT statement, Øw describes the field that is to contain an arrangement of octal characters.

  According to this descriptor, the right-most w 3-bit characters (terminating at the 47th bit position, or at the n^{th} bit position if an nT modifier is used) of a word in the LIST of the ORDER statement will be converted to w 6-bit characters for output.

  Examples:

INTERNAL (48 Bits) FORM	Øw	EXTERNAL FORM
000.....000 111 011	Ø1	3
000.....000 111 011	Ø3	073

Cw

- When used in a FORMIN statement, Cw describes a field of hexadecimal characters which is to be transferred to a word in memory.

  Cw causes w six-bit characters in the input field to be converted to its hexadecimal (4-bit) equivalent and stored right-justified (terminating at the 47th bit position, or at the n^{th} bit position if an nT modifier is used) in a word specified in the LIST of the ORDER statement. Although all characters are acceptable to the Cw descriptor, only the decimal digits 0-9 and the alphabetic characters A-F can be unmistakeably represented with this descriptor.

  A field may be as large as 12 characters. Space characters are included in the w count, but are ignored in the conversion process.

  Examples:

EXTERNAL FORM	Cw	INTERNAL (48 Bits) FORM
38AF	C4	0.....0011 1000 1010 1111
71D	C3	0..........0111 0001 1101
9Δ90	C4	0..........1001 1001 0000

- When used in a FORMOUT statement, Cw describes the field that is to contain an arrangement of hexadecimal characters.

  According to this descriptor, the right-most w 4-bit characters (terminating at the 47th bit position, or at the n^{th} bit position if an nT modifier is used) of a memory location specified in the LIST of the ORDER statement will be converted to w 6-bit characters for output.

  Examples:

INTERNAL (48 Bits) FORM	Cw	EXTERNAL FORM
0......0000 1001 1111	C3	09F
0......0011 1000 1010	C3	38A

nH

- The nH descriptor is only used in FORMOUT statements; it defines a fixed field n characters wide. The characters (alphanumeric or special) are written in the FORMOUT statement immediately to the right of the descriptor.

  Examples:

HOLLERITH FIELD	EXTERNAL FORM
7HΔΔSUM=4	ΔΔSUM=4
6HSAMPLE	SAMPLE

Sw

- When used in a FORMIN statement, the Sw descriptor causes a field of w characters to be spaced over or bypassed in the record. Sw may not be used to bypass a record.

- When used in a FORMOUT statement, this descriptor produces w space characters in the output record.

MODIFIERS

This section deals with the various IOPS modifiers, and the additional flexibility that they provide in the arrangement of data, such as:

- The repetition of similar fields
- The positioning (scaling) of data fields in memory locations
- The controlling of signs and the suppression of leading zeros in output fields.

IOPS modifiers include:

- The Repeat Modifier (nR)
- The Terminal Position Factor (nT)
- The Exponent Modifier (nP)
- The Sign Modifier (Z)
- The Leading Zero Modifier (L)

Modifiers are prefixed to descriptors, each modifier to the descriptor it is to qualify. It is also possible for several modifiers to be used with a single descriptor.

A discussion of IOPS modifiers and their specific functions is presented below.

THE REPEAT MODIFIER nR

When successive fields are to be read or written (printed or punched) in the same form, a repeat modifier may be used to indicate the number of times a particular arrangement is to be repeated.

If nR immediately precedes the descriptor, the R may be omitted, thus, 2RF6.2 and 2F6.2 are equivalent.

<u>Example</u>:

Using the repeat modifier, the statement

LOCATION	COMMAND	ADDRESS
	.	. . .
TAG	FØRMIN	F6.2,F6.2 $
	.	. . .

could be written as

LOCATION	COMMAND	ADDRESS
	.	. . .
TAG	FØRMIN	2F6.2 $ or 2RF6.2 $
	.	. . .

The Repeat Modifier may also be used to repeat a group of descriptors. (See example, page 38.)

THE TERMINAL POSITION FACTOR nT

The Terminal Position Factor, nT, is used to position, and to indicate the position of, the quantity in a computer word. n, which may be any decimal number 0-47, refers to one of the 48 bit positions in a computer word.

When used with an Øw, Ww, or Cw descriptor, nT indicates the right-most position of the least significant character of the input or output field in a word in memory. With the Iw or Dw.d descriptor, nT indicates the position of the binary point. If nT is not specified, 47T is assumed, and the field is positioned at the extreme right end of the word.

The nT modifier is used only with I, D, Ø, W or C descriptors. The following are examples of nT used with a Dw.d descriptor:

Examples:

EXTERNAL FIELD	nTDw.d	DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER (Appropriately Scaled)
6550	46TD4.2	65.5
-65.75	45TD6.0	-65.75
-809375	43TD7.4	-80.9375
Δ+Δ12Δ25	44TD8.3	+12.025
Δ+Δ12Δ25	0TD8.3	.025
6550	47TD5.2	65.

DECIMAL EQUIVALENT of INTERNAL BINARY NUMBER (Scaled 45T)	nTDw.d	EXTERNAL FORM
+6.25	45TD5.2	Δ6.25
+6.25	45TD5.1	ΔΔ6.3
+6.25	45TD5	ΔΔΔ6.
+6.25	46TD5.1	Δ12.5
-13.5	45TD5.1	-13.5
-13.5	43TD6.3	-3.375
-5.25	45TD6.2	Δ-5.25

Examples of nT used with Øw and Cw descriptors are presented below:

EXTERNAL FIELD	DESCRIPTOR WITH MODIFIER	INTERNAL (48 Bits) NUMBER
73	44TØ2	0..............111 011 000
73	41TØ2	0..........111 011 000 000
97	43TC2	0...........1001 0111 0000
97	39TC2	0......1001 0111 0000 0000

INTERNAL (48 Bits) NUMBER	DESCRIPTOR WITH MODIFIER	EXTERNAL FORM
0..............111 011 000	44TØ2	73
0..............111 011 000	41TØ2	07
0..........111 011 000 000	44TØ2	30
0...........1001 0111 0000	39TC2	09
0......1001 0111 0000 0000	43TC2	70

THE EXPONENT MODIFIER nP

The Exponent Modifier, nP, is used as a decimal scale factor with I, D, and F descriptors, or as a position factor with the E output descriptor. (nP has no effect on E input descriptors.)

The quantity described by an nP-modified I, D or F descriptor is multiplied by 10^n, where n may be any negative or unsigned decimal integer. When nP is used with an E output descriptor, the mantissa of the quantity in memory is multiplied by 10^n and the exponent is reduced by n. The modifier can be regarded as acting after the descriptor when describing input, and before the descriptor when describing output.

Examples:

EXTERNAL FIELD	DESCRIPTOR WITH MODIFIER	DECIMAL EQUIVALENT OF INTERNAL BINARY NUMBER
658	1PF3.2	$.658\times10^{2}$
3.987	2PF5	$.3987\times10^{3}$
532+3	-4PE5	$.532\times10^{6}$
-1376E+004	-2PE10.1	$-.1376\times10^{7}$

DECIMAL EQUIVALENT OF INTERNAL BINARY NUMBER	DESCRIPTOR WITH MODIFIER	OUTPUT FORM
$.658\times10^{-1}$	3PF5.2	65.80
$.732\times10^{3}$	-4PF6.4	0.0732
$.486\times10^{-4}$	2PE8.1	48.6-006
$-.3718\times10^{2}$	1PE10.3	-3.718+001

THE SIGN MODIFIER Z

When used in a FORMOUT statement, the Sign Modifier, Z, causes positive numbers to be printed with + signs. (Negative signs are always printed.) When used with an E descriptor, Z controls the sign of the mantissa. The sign of the exponent is always printed.

THE LEADING ZERO MODIFIER L

Normally, when a quantity is printed or punched in an output field, leading zeros are suppressed (i.e., replaced with spaces). If the Leading Zero Modifier, L, is used in a FORMOUT statement however, leading zeros will not be suppressed and will appear in the output field.

The following chart indicates which MODIFIERS may be used with each FIELD DESCRIPTOR:

		MODIFIERS				
		FORMIN or FORMOUT			FORMOUT only	
		R	T	P	Z	L
DESCRIPTORS	I	X	X	X	X	X
	E	X		Output only	X	X
	F	X		X	X	X
	D	X	X	X	X	X
	W	X	X			
	Ø	X	X			X
	C	X	X			X
	S					
	A	X				

MULTI-RECORD FORMATS

Records with different formats can be described in a single FORMIN or FORMOUT statement by using a slash (/), end-of-record indicator, to separate the descriptors of different records. For example, if the statement

LOCATION	COMMAND	ADDRESS
TAG	FØRMØUT	2H10,I2, F8.2/2H10,I5, 3F12.6 $

were used with an ORDER statement specifying a print operation, all odd numbered lines would be printed according to the descriptors I2 and F8.2, and all even numbered lines according to descriptors I5 and 3F12.6.

Repetition of similar formats can also be accomplished by using parentheses and/or repeat modifiers. For example, 2(I2, Ø5, S2) is equivalent to I2, Ø5, S2, I2, Ø5, S2. <u>Parentheses within parentheses are not permitted in IOPS FORMAT statements</u>. Thus, if the statement

LOCATION	COMMAND	ADDRESS
	•	• • •
TAG	FØRMØUT	2H70,I5, 3F8.2/2H10,2I10, F9.2/ 2H10,12(I6, F4.1)$
	•	• • •

were used with an ORDER statement specifying a print operation, three lines would be printed per page; the first line according to the descriptors I5 and 3F8.2, the second line according to descriptors 2I10 and F9.2, and the third line according to the descriptors represented as 12(I6, F4.1). If the list comprised 32 elements, the contents of the 32nd element would be printed on the first line of a new page, according to descriptor I5, because when the end of a FORMAT statement is reached and the LIST is not satisfied (i.e., not all elements processed) the next element will begin a new record starting with the first descriptor of the FORMAT statement.

Multi-slashes are used to by-pass records, i.e., to skip a card on input, or to produce a record of blanks when printing or punching. For example,

// would cause one record to be skipped.
/// would cause two records to be skipped.

MISCELLANEOUS EXAMPLES

- Only the first 4 columns of each of 100 cards will be read, according to the following statements.

LOCATION	COMMAND	ADDRESS
	.	. . .
	IØUNITS	DTI,8T,12,10,ERR,BUF $
	READ	8T;ABC;INFØ(100) $
ABC	FØRMIN	A4 $
	.	. . .

- The statements below cause 20 four-column fields to be punched on each card, until five cards are punched.

LOCATION	COMMAND	ADDRESS
	.	. . .
	IØUNITS	PCØ,12T,12,10,ERR, BUF $
	WRITE	12T;DEF;INFØ(100) $
DEF	FØRMØUT	20A4 $
	.	. . .

- According to the following statements, 8 alphanumeric characters will be printed per line, until 100 lines have been printed.

LOCATION	COMMAND	ADDRESS
	.	. . .
	IØUNITS	DTØ, 13T,,, ERR, BUFF$
	WRITE	13T; MAT; ØUTPUT (100) $
MAT	FØRMØUT	2H10, A8 $
	.	. . .

- The following statements cause 20 lines of data to be printed, each line having 5 eight-character fields (each field being separated by 4 space characters).

LOCATION	COMMAND	ADDRESS
	.	. . .
	IØUNITS	DTØ, 13T,,,ERR, CSA $
	WRITE	13T; FØRM; DATA (100) $
FØRM	FØRMØUT	2H10, 5(S4, A8) $
	.	. . .

- According to the following, an 8-column field will be punched on each card, until 100 cards have been punched. All odd numbered cards will be punched according to the E descriptor, while all even numbered cards will be punched according to the F descriptor.

LOCATION	COMMAND	ADDRESS
	.	. . .
	IØUNITS	PCØ, 12T, 12, 10, ERR, BUF $
	WRITE	12T; FØRM3; ØUTPUT (100) $
FØRM3	FØRMØUT	E8.3/F8.3 $
	.	. . .

- The statements shown below cause the word "VALUE" to be printed at the top of a new sheet of paper, after which two lines are skipped, and 3 alphanumeric characters printed per line, until 50 lines have been printed.

LOCATION	COMMAND	ADDRESS
	.	. . .
	IØUNITS	DTØ, 13T,,,ERR, CSA $
	WRITE	13T; EDIT; DATA (50) $
EDIT	FØRMØUT	7H70VALUE///50(2H10, A3/) $
	.	. . .

APPENDIX A

ILLUSTRATIVE EXERCISES

SAMPLE PROBLEM 1

Read data from tape 9 having the following card format. (Assume that the tape was prepared by reading 10 words per card, 12 cards per block, code-mode.)

Columns	Instruction
1-10	Sequence information for sorting, do not read.
11-18	Alphanumeric data to be stored in location IDEN.
19-30	Data to be converted to floating point and stored in location VALUE1. If the field contains no decimal point, the last three digits are to be considered the fractional part.
31-40	Data to be converted to fixed point scaled B20 and stored in location VALUE2. If the field contains no decimal point, the last three digits are to be considered the fractional part.
41-45	Scaled data which is to be multiplied by 100, converted to floating point and stored in location SPECIAL. If the field contains no decimal point, the last four digits are to be considered the fractional part.
46-80	Other information to be ignored.

The necessary coding could be of the form:

L	LOCATION	COMMAND	ADDRESS
		.	. . .
		READ	9T;XYZ;IDEN;VALUE1;VALUE2;SPECIAL$
	XYZ	FØRMIN	S10,A8,F12.3,20TD10.3,2PF5.4$
		IØUNITS	DTI,9T,12,10,ERR,BUFF1$
	ERR	HLT	M/77777$
	BUFF1	ASTØR	128$
		.	. . .

SAMPLE PROBLEM 2

Assume that three sequential memory locations contain data to be transferred to tape 12 for off-line punching of an 80 column card. Location DATA contains alphanumeric data which is to be punched in columns 1-8, location DATA+1 contains a fixed-point binary integer to be punched in columns 11-22, and location DATA+2, a fixed-point binary integer to be punched in columns 23-80.

The necessary coding could be as follows:

L	LOCATION	COMMAND	ADDRESS
		.	. . .
		WRITE	12T;UVW;DATA(3)$
	UVW	FØRMØUT	A8;S2,I12;I58$
		IØUNITS	PCØ,12T,12,10,ERR,BUF1,BUF2$
	BUF1	ASTØR	128$
	BUF2	ASTØR	128$
	ERR	HLT	M/22222$
		.	. . .

SAMPLE PROBLEM 3

The main purpose of the following example is to show IOPS coding in a TAC language program. There are 5 data columns per field, 16 fields per record. Each field is to be converted from fixed-point decimal to floating-point binary, maintaining at least two decimal places in each field. Input is from tape 10, and output, edited for the printer, is to be on tape 11.

Read 160 items into the array starting at symbolic location ABLE; reverse the order of information in array BAKER, and print out the contents of array BAKER, columnwise, in fixed-point decimal form, accurate to two decimal places.

L	LOCATION	COMMAND	ADDRESS
		.	. . .
I		PRØGRAM	
		NAME	JØHN $
		SET	M/1000 $
		IØUNITS	DTI, 10T, 12, 10, , CSA1;
			DTØ, 11T, , , , CSA2 $
	START	INIT	16; END $
		READ	10T; INPUT; ABLE (160) $
	INPUT	FØRMIN	16F5.2 $
		TMD	C/HLT, ABLE+159; C/HLT, BAKER $
		TDXLC	0,4 $
		TDXRC	0,5 $
R		RPTSA	160 $
		TMD	1,4 $
		TDM	1,5 $
		WRITE	11T; ØUTPUT; BAKER (160) $
	ØUTPUT	FØRMØUT	2H70, S5, 12HTHEΔARRAYΔIS/
			160(2H10, S10, F5.2/) $
		ENDFILE	11T $
		REWIND	10T $
		REWINDLØ	11T $
	END	HLT	M/3333 $
	ABLE	ASTØR	160 $
	BAKER	ASTØR	160 $
	CSA1	ASTØR	128 $
	CSA2	ASTØR	128 $
		END	START $
		.	. . .

APPENDIX B

PROC INIT STATEMENT

The INIT statement is written in the following format:

LOCATION	COMMAND	ADDRESS
	•	• • •
	INIT	n;ERRØR$
	•	• • •

The parameter, n, is written as an unsigned decimal integer, representing the number (not exceeding 16) of input-output orders which PROC is expected to process at any given time.

The parameter, ERRØR, is written as the symbolic or absolute address of a location to which a return will be made if a non-recoverable error occurs within PROC or IOPS. Appendices E and F indicate the status of the A and Q registers when such an error occurs.

APPENDIX C

BINARY RECORD FORMAT

Binary records are written on tape (by a WRITEBT statement) in a special format, namely IOPS format, and may consist of any number of blocks. At present, each block can contain as many as 126 data words; the 127th word is a control word, and the 128th word is a checksum of the preceding 127 words.

The format of a control word is as follows:

Bit 0 = 1, if this is the first block of a record

Bit 47 = 1, if this is the last block of a record

Left Address - indicates the number of data words in a block

Right Address - indicates the order or place of this block within the record.

In agreement with the above format, the end-of-file control word in the special record that is written as a result of an ENDFILE statement is 1/1T0;1/1T39;1/1T47.

APPENDIX D

HEXADECIMAL CHARACTERS

HEXADECIMAL CHARACTER	DECIMAL EQUIVALENT	BINARY EQUIVALENT
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

APPENDIX E

IOPS SYSTEM ERROR INDICATIONS

	ERROR	REGISTERS	
		A	Q
1.	Attempt to use a tape which has not been defined in the IOUNITS statement	1/1	Unit T15 or T23
2.	Attempt to read a tape described in the IOUNITS statement as "write only".	2/1	Unit T15 or T23
3.	Attempt to write on a tape described in the IOUNITS statement as "read only".	3/1	Unit T15 or T23
4.	Attempt to read more words from a binary tape than there are in the record.	4/1	---
5.	The checksum of the block after reading does not equal the checksum written with the block. This is applicable to binary tape only.	5/1	---
6.	Attempt to store, for printing, more than 120 printable characters for a given line.	6/1	---
7.	Attempt to read from a binary tape which is not in the correct format.	7/1	---
8.	The program was run even though an error occurred at compilation time.	8/1	---
9.	Either a machine error, or the program has destroyed some IOPS coding (2GNC).	9/1	---
10.	More characters are requested from a card than were described in the record size parameter in the IOUNITS statement	10/1	---
11.	Same as 9. (BIWTR)	11/1	---
12.	Illegal character or illegal exponent field in floating to floating (E) input conversion.	12/1	last character processed, T5

IOPS SYSTEM ERROR INDICATIONS (Cont.)

	ERROR	REGISTERS	
		A	Q
13.	Illegal character or illegal exponent field in fixed decimal to floating (F) input conversion.	13/1	last character processed, T5
14.	Illegal character in fixed-point decimal to fixed-point binary (D) input conversion.	14/1	last character processed, T5
15.	Illegal character or exponent field in integer to integer (I) input conversion.	15/1	last character processed, T5
16.	Rewind order to a unit defined as paper tape.	16/1	

APPENDIX F

PROC SYSTEM ERROR INDICATIONS

A Register	Q Register	Meaning
Zeros	Location of macro-instruction in left address	List full
Location of macro-instruction being checked in left address; 1/1T47 (D/1)	Normal exit of check macro-instruction in left address	Not in list
Location of macro-instruction being checked in left address; 1/1T46 (D/2)	Normal exit of check macro-instruction in left address	Checking non-magnetic tape order with magnetic tape check macro-instruction
Location of macro-instruction being checked in left address; 1/1T45 (D/4)	Normal exit of check macro-instruction in left address	Checking for completion of more blocks than were to be processed (B = NBP of original order.)
Unit #T23; 1/1T44 (D/8)	Location of macro-instruction being checked in left address	Rewind not accepted after TIØ
Location of macro-instruction being checked in left address; 1/1T43 (D/16)	Normal exit of check macro-instruction in left address	Checking out of sequence
Unit #T23; 1/1T42 (D/32) Location of macro-instruction being checked in left address	Location of macro-instruction being checked in left address, and location of check macro at T40	Tape in local command
Unit #T23; 1/1T41 (D/64) Location of macro-instruction being checked in left address	Location of macro-instruction being checked in left address, and location of check macro at T40	Write or read only

If a tape error is encountered while executing an input-output order, and if attempts by PROC to correct this error fail, PROC will return to the error address specified for a particular tape unit in the IOUNITS (or IOUNITSF) statement. When this happens, the A and Q registers will contain:

A Register	Q Register
1. The fault register configuration at T15	1. The address of an ERRORS macro used by IOPS at T16.
2. The tape unit at T23	2. The original CSA at T39
3. The assembler counter register configuration at T39	3. The original NBP at T47.
4. The original NBS at T47.	

APPENDIX G

POSSIBLE GROUPING OF CHARACTERS $\alpha\beta$ IN IOPS FORMAT STATEMENTS

α = preceding character Y = permitted

β = succeeding character N = not permitted

α \ β	n	; ,	$	/	.	(	)	I	F	E	D	Ø	A	W	C	S	H	R	T	P	Z	L	+	-
n	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N
; and ,	Y	N	N	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y
$	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
/	Y	N	Y	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y
.	Y	Y	Y	Y	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
(	Y	N	N	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y
)	N	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
I	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
F	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
E	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
D	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
Ø	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
A	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
W	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
C	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
S	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
H(+n)	N	Y	Y	Y	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
R	Y	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y
T	Y	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	Y	Y	Y	Y
P	Y	N	N	N	N	N	N	Y	Y	Y	Y	Y	N	N	Y	N	N	N	N	N	Y	Y	N	N
Z	Y	N	N	N	N	N	N	Y	Y	Y	Y	Y	N	N	Y	N	N	N	N	N	N	Y	Y	Y
L	Y	N	N	N	N	N	N	Y	Y	Y	Y	Y	N	N	Y	N	N	N	N	N	Y	N	Y	Y
+	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
-	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

INDEX

PHILCO®

Famous for Quality the World Over

A SUBSIDIARY OF *Ford Motor Company*

Printed in U.S.A.