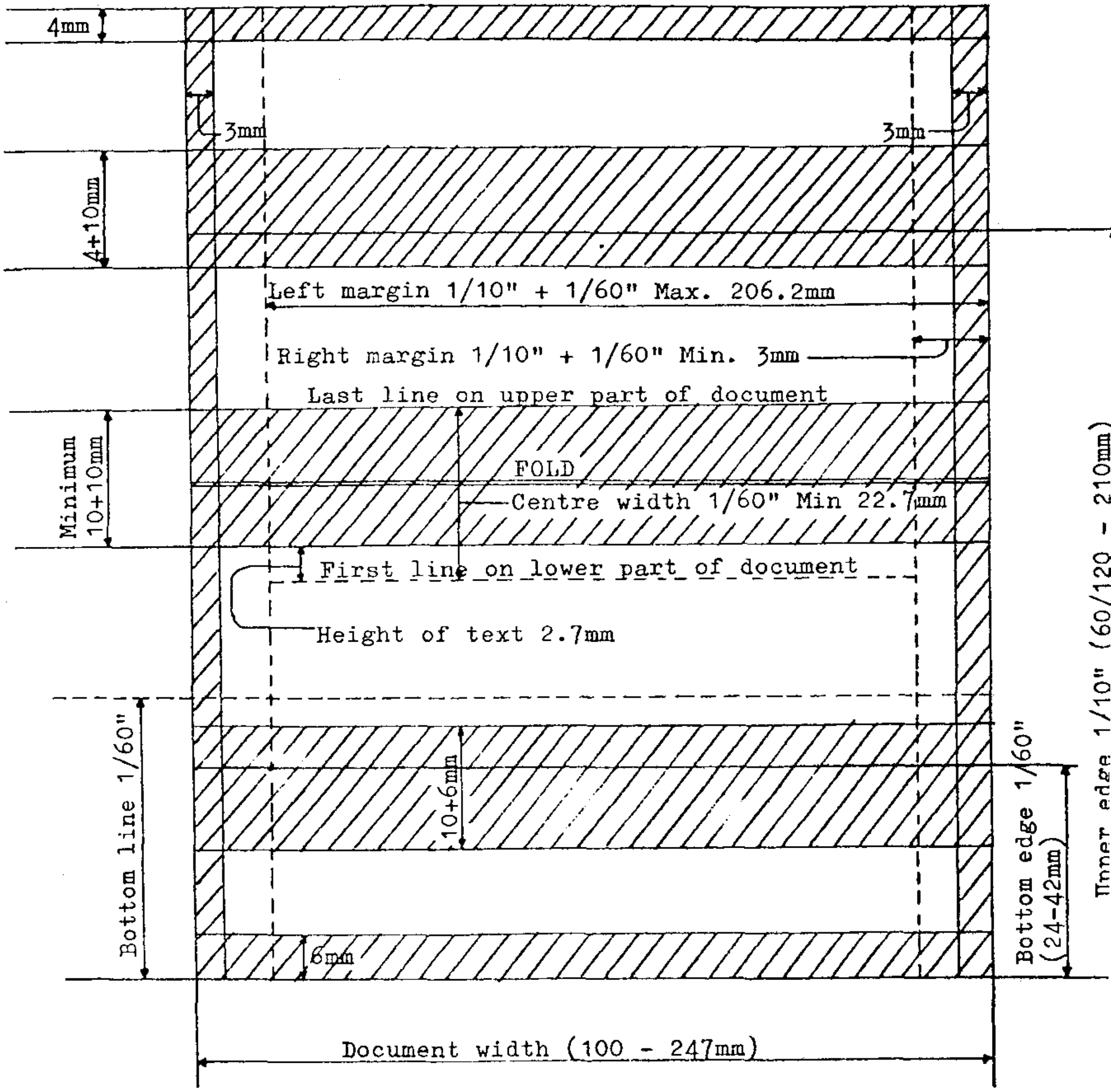
Diagram of parameters for document type 3 (Horizontally folded)


4mm
3mm
3mm
4+10mm
Left margin 1/10" + 1/60" Max. 206.2mm
Right margin 1/10" + 1/60" Min. 3mm
Last line on upper part of document
Minimum 10+10mm
FOLD
Centre width 1/60" Min 22.7mm
First line on lower part of document
Height of text 2.7mm
Bottom line 1/60"
10+6mm
6mm
Bottom edge 1/60" (24-42mm)
Inner edge 1/10" (60/120 - 210mm)
Document width (100 - 247mm)

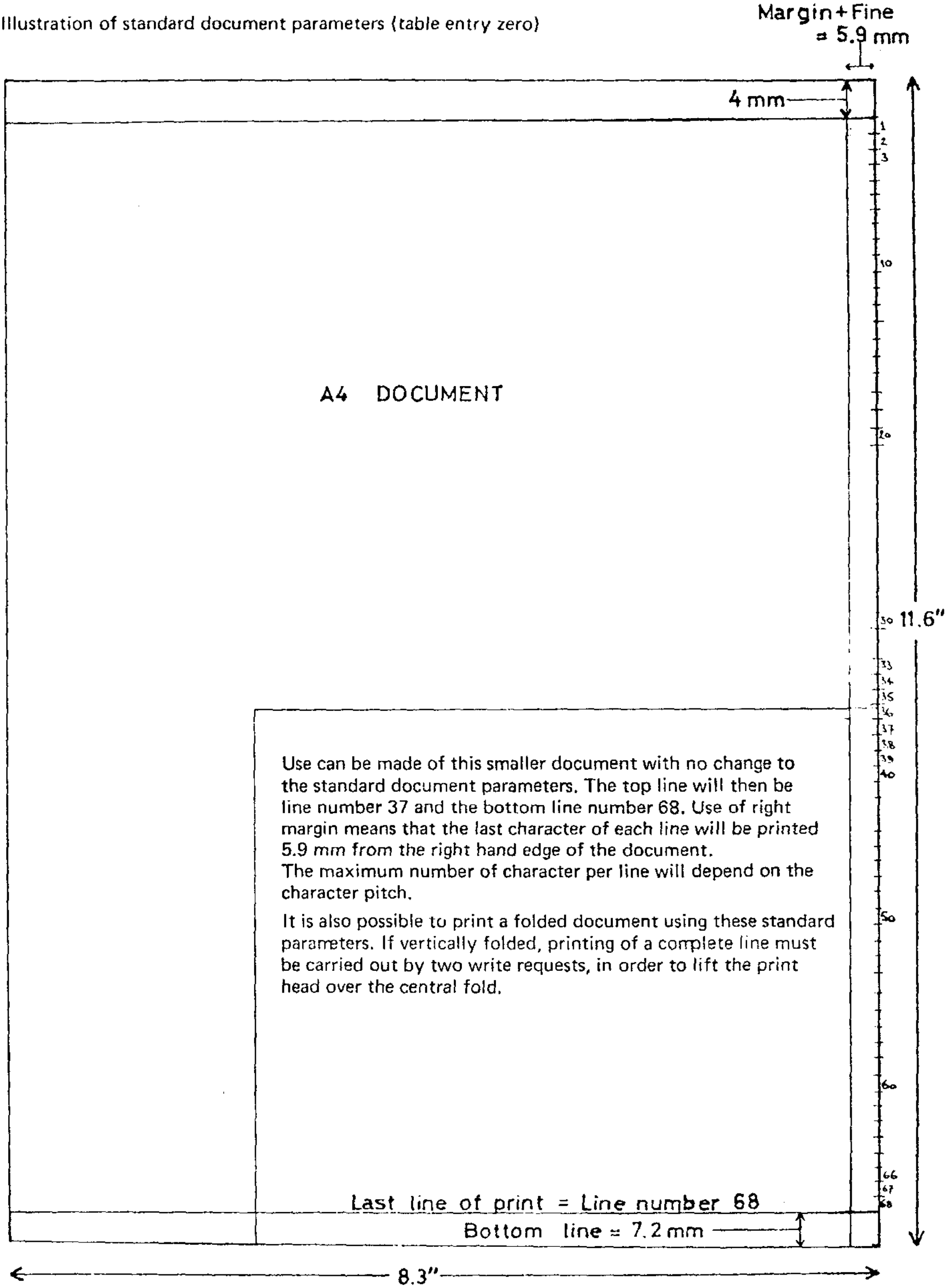
▨ = Areas in which printing is not possible

Table of Standard Document Parameters (entry zero in Parameter Table)

Parameter Type	Value	Description
DT	1	Unfolded document
TO	0	No timeout; printer will wait until document is inserted
LS	10	10/60" between each line on the document
NL	68	Number of lines is 68
BL	17	The distance from the bottom of the document to the bottom of the characters on the last line (number 68) is 17/60" = 7.2 mm.
MA MF	2 2	} The margin is set 2/102 + 2/60" = 14/60" = 5.9 mm from the rightmost edge of the document.
LM	0	Print with right margin. The last character on each line is placed 5.9 mm from the rightmost edge of the document
CM	0	No critical margin; gives faster positioning
HP	0	Normal print pressure; this assumes multipart sets are not being used.
UE	58	58/5" = 11.6", the height of an A4 document
BE	0	No inner pages on the document (like passbooks)
DW CW		}Not required

Note : this document uses right margin. This means that if a smaller document is used, printing may still take place, starting at a higher line number than 1 (i.e. lower on the page), and without using some of the lefthand print positions.
Thus a different document may be handled without the necessity for the user to send any document parameters; see illustration on next page.

Illustration of standard document parameters (table entry zero)



Use can be made of this smaller document with no change to the standard document parameters. The top line will then be line number 37 and the bottom line number 68. Use of right margin means that the last character of each line will be printed 5.9 mm from the right hand edge of the document.
The maximum number of character per line will depend on the character pitch.

It is also possible to print a folded document using these standard parameters. If vertically folded, printing of a complete line must be carried out by two write requests, in order to lift the print head over the central fold.

DUPL Duplicate DUPL

Syntax: [statement-identifier] ⊔ DUPL ⊔ data-item-identifier.

Type: Format control I/O

Description: The contents of the duplication data-item, as defined by the FKI-format list declaration, of the current input field is moved to the string data-item referenced by data-item-identifier.
A duplication data-item in a FKI-input field, may be of the type decimal or string.
The DUPL instruction uses the same conversion rules as the MOVE instruction for conversions from:

string ⟶ string
decimal ⟶ string.

Exception: When moving from string to string type of data item and the size of the receiving data-item is greater than the size of the sending data item, the remaining characters in the receiving data item will be X'00', instead of repeating the last character.

Condition register: = 0 Operation successfully performed
= 2 No duplication data-item associated with the current input field. (See FKI).

Condition mask:

0	1	2	3	4	5	6	7
SUCCESS	–	NO DUPL ITEM	–	$\overline{\text{SUCCESS}}$	DUPL ITEM	–	UNCONDITIONAL

Example: DUPL ⊔ DUPITEM

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	data-item-identifier							

Bytes 1 and 2 are filled by the system.
Operand-1 is a reference to a string data item

DVR *Divide rounded* DVR

Syntax: [statement-identifier]␣DVR␣ data-item-identifier-1, $\left\{\begin{array}{l}\text{data-item-identifier-2}\\\text{literal constant}\end{array}\right\}$

Type: Arithmetic instruction

Function: (Operand-1) ÷ (Operand-2) → Operand-1

Description: Operand-1 is divided by operand-2. The result is augmented by 0.5 and then rounded down. It is stored in operand-1. Operand-2 is unchanged. Both operands must be decimal or binary. Division by zero results in overflow and operand-1 is set to zero.

Condition register :
= 0 if (operand−1) = 0
= 1 if (operand−1) > 0
= 2 if (operand−1) < 0
= 3 if Overflow

Intermediate code format:

Byte 1	0	0	0	0	1	1	0	L
operand-1	data-item-identifier-1							
operand-2	data-item-identifier-2							

Byte 1 is the operation code (X'0C' or X'0D').
L=1 operand-2 is a reference to a literal constant.
L=0 operand-2 is a reference to data-item-identifier-2, array-identifier-2
or a formal parameter.

DYKI *Display Keyboard input* DYKI

Syntax: [statement-identifier] ⊔ DYKI ⊔ data-item-identifier-1, key-table-identifier-1, key-table identifier 2, size-identifier, index-identifier, data-item identifier-2

Type: Format control I/O.

Description: Characters are read from the input data set as mentioned in the FMTCTL declaration in the data division, and stored in a string data item referenced by data-item-identifier 1 (Input buffer). The size of this buffer must be greater than the size as mentioned in the "MAXL" option, of the current input field, to allow the end-of-item key to be entered in the buffer too. Input is performed as a KIA-instruction with the input characters echoed on the output data set as mentioned in the FMTCTL declaration. Only the first input character is checked if it is present in the keytable referenced by key-table-identifier-1. The first four positions in the keytable have a predefined significance. (See below). If this character is not present in key table-1, the current input field on the display is filled with periods.
The second and following input characters are read and checked with key-table-2, from which the first four positions also have a predefined significance (See below). If an input character is present in the key-table, its position number (minus one), as declared in the key table, is returned in a binary data item referenced by index identifier. After completion of the transfer a converted key table index value is returned in this data-item which contents may be zero, a negative value or a positive value with the following meaning.

zero: Power failure has been present.

negative: A key lock switch has been turned.

positive: An index value ranging from 1 to (n-1) corresponding with the position number minus 1 in keytable-1 or keytable-2 is returned; if the transfer was correctly completed.

Note: Index value one corresponds with the second key code in the key table.
'n' is the number of key codes in the key table.
When no-end-of-item key is used to complete the transfer, the index value will be set to an undefined value outside the range -255 to +255.

If an illegal key code is received or the number as specified in MAXL is exceeded, a bell signal is sent to the display and input is restarted. After completion of the transfer a binary data item referenced by size identifier contains the number of characters transferred <u>excluding</u> the end of item key.

When an error occurs before the transfer is completed, an error code is returned in the binary data item referenced by data-item-identifier-2. This error code may be:

0 – no error

1 – number of characters received is less than the number specified in MINL.

2 – not used

3 – I/O error

4 – request aborted.

Expected predefined key table items in keytable-1 and keytable-2:

Position number in keytable. Significance

1 BACKSPACE. When this key code is entered the cursor is moved one position to the left, and a period is displayed in the new cursor position.
If the first character position of the current input field is reached, the same function as CLEAR2 will be executed.

2 CLEAR1. The current input field is erased on the screen and its current input data item is cleared. Cursor is positioned at the first position of the next input field.

3 CLEAR2. The current input field is erased on the screen, and the cursor is positioned at the first position of the next input field.

4 EOI. General end of item key. Checks according to the number as mentioned in MINL is performed

Transfer ended when:

a) Any of the keys listed in keytable-1 (first position in the input field) or keytable-2 (second and following positions) except BACKSPACE, is received.

b) The maximum number of characters as defined by MAXL is reached and the current input field has the "NEOI" – flag set.

c) Power failure occurs

d) Keylock is turned

e) I/O error occurs.

Condition register: = 0 if OK (Error code in data item referenced by data item identifier-2 is zero)

= 2 if Error (Error code in data item referenced by data item identifier-2 is not zero)

Condition mask:

0	1	2	3	4	5	6	7
OK	–	ERROR	–	OK	–	$\overline{\text{ERROR}}$	UNCON-DITIONAL

Example: DYKI␣BUFFER, KTB1, KTB2, SIZE, INDEX, ERRCODE

Intermediate code format:

Byte 1	0 0 1 1	0 0 0 0
Byte 2	external reference	
operand-1	data-item-identifier-1	
operand-2	key table identifier-1	
operand-3	key table identifier-2	
operand-4	size-identifier	
operand-5	index-identifier	
operand-6	data-item-identifier-2	

Bytes 1 and 2 are filled by the system.
Operand-1 is a reference to a string data item.
Operands-2, 3 are references to key tables.
Operands-4, 5, 6 are references to binary data items.

EDFLD *Edit Input Field* EDFLD

Syntax: [statement-identifier] ␣ EDFLD ␣ data-item-identifier-1, key table. identifier, size-identifier, index-identifier, data-item-identifier-2.

Type: Format control I/O

Description: Editing is performed in the string data item referenced by data-item-identifier-1 (buffer).
The size of this data item must be greater than the number mentioned in "MAXL" of the current input field. Depending on the contents of the binary data item referenced by size-identifier, the following operations are performed.

(size-identifier) = 0 The contents of the data item of the current input-field is moved to the data item referenced by data-item identifier-1. If the "REWRT" flag is set for the current input field, the contents of data-item-identifier-1 is displayed on the screen in its proper position without editing according to a picture definition.
The cursor is placed at the first character position of the field.

(size-identifier) ≠ 0 The cursor is placed at a character position, the number of which is contained in the binary data item referenced by size-identifier. "1" corresponds with the first character position.

Then characters are read from the input device, declared in the FMTCTL declaration, into the character positions in the buffer (data-item-identifier-1). The character positions correspond with the cursor position within the current input field. If a keycode is received which is present in the first <u>four</u> positions of key table, referenced by key-table-identifier, the corresponding function is executed and reading is resumed.

On a illegal keycode, a bell signal is sent to the output device (FMTCTL and reading is resumed.

Expected predefined keytable items in keytable :

Position number in keytable	Significance
1	→ Non destructive space. Cursor is moved one position to the right. No action if cursor is at the right-most position of the current input field, or beyond the last significant character (i.e. at X'00' in the buffer).
2.	← Non destructive backspace. Cursor is moved one position to the left. No action if cursor is at the left-most position of the current input field.
3.	INS. Insert character. The characters from the current cursor position up to the last position in the field are shifted one step to the right. Any character shifted beyond the end of the line is dropped.

EDFLD *Continued* **EDFLD**

Position number in keytable	Significance
4	DEL. Delete character. The character at the current cursor position is deleted. Characters to the right of the current cursor position are shifted one step to the left. Cursor is not moved.
5	CLEAR1. Clear input field and input data item. Cursor is moved to the first position of the current input field. Terminate EDFLD. The contents of data item referenced by size identifier, is set to zero.
6	CLEAR2. The current input field is erased on the screen and the cursor is positioned, at the first position of the input field. Terminate EDFLD.
7	CLEAR3. Clear remaining positions in the field. The characters from the current cursor position, up to the last character in the field (inclusive) are cleared. Terminate EDFLD.
8	EOI. Common end of item. Contents of the binary data item referenced by index identifier, is set to three.
9 or higher	Terminate EDFLD.

Editing is terminated when:

a) A keycode is received, which is present in position 5 or higher, in the keytable.
b) Power failure has occurred.
c) A keylock switch is turned.
d) I/O error occurs.

After completion of this instruction, a value with following significance is returned in the binary data item referenced by index identifier:

zero:	power failure has occurred.
negative:	a keylock switch has been turned.
positive:	an index in the range from

1 to (n-4) is returned, corresponding to positions 5 to N in the keytable (n is the keycode position in the keytable). Index value 3 is returned when the common EOI code, from position 8 in the keytable, is received.
In the binary data item referenced by size identifier is returned the effective length of the operation. The effective length is the number of resulting non-null characters in the buffer (data-item-identifier-1).
A null character has code X'00'.

In the binary data-item referenced by data-item identifier-2, is returned a code with following significance :

Contents	Significance
0	OK
1	The effective length is less than "MINL" (not set when CLEAR1)
2	not used.
3	I/O error
4	request aborted.

Condition register: = 0 if OK
= 2 if ERROR (The data item referenced by data-item-identifier-2, contains the error code).

Condition mask:

0	1	2	3	4	5	6	7
OK	–	ERROR	–	$\overline{\text{OK}}$	–	$\overline{\text{ERROR}}$	UNCON-DITIONAL

Example: EDFLD ⊔ SPINPUT, SPKTAB3, SPBINW1, SPBINW2, SPBINW4.

Intermediate object code:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	data-item-identifier-1							
operand-2	key table identifier							
operand-3	size-identifier							
operand-4	index-identifier							
operand-5	data-item-identifier-2							

Bytes 1 and 2 are filled by the system
Operand-1 is a reference to string data item.
Operand-2 is a reference to a key table.
Operands-3, 4, 5 are references to binary data items.

EDIT Edit EDIT

Syntax: [statement-identifier]␣EDIT␣ data-item-identifier-1, $\left\{\begin{array}{l}\text{data-item-identifier-2}\\ \text{format-list-identifier}\end{array}\right\}$

Type: String instruction

Description: This instruction uses the format list to convert decimal and string data items into an edited string. The data items specified in the format list are edited according to the specified format and stored in a string data item indicated by operand-1.

Format-list-identifier is a reference to an edit format list which is composed of format declarations (FRMT, FCOPY, FMEL etc.) Instead of a format-list-identifier, operand-2 may be a reference to a string data-item. This data item must contain format-list characters as present in the format-literalpool. (output CREDIT linker). Item size must be great enough to contain these characters. The CALL FMOVE instruction may be used to fill the data-item.

Condition register: Not significant.

Example: EDIT FIELD, FORM1

Intermediate code format:

Byte 1	0	1	1	0	0	0	0	L
operand-1	data-item-identifier-1							
operand-2	format-list-identifier							

Byte 1 is the operation code (X'60' or X'61')
Operand-1 is a reference to a string data item.
L=1 operand-2 is a reference to a format list.
L=0 operand-2 is a reference to a string-data-item.

EDSUB *Edit Substring* EDSUB

Syntax: [tatement-identifier] ⊔EDSUB⊔ data-item-identifier-1, pointer-identifier,
{data-item-identifier-2}
{format-list-identifier }

Type: String instruction.

Description: Editing as specified in the formatlist is performed into a subfield of the string-data-item indicated by operand-1, beginning at pointer-identifier.
Upon completion, the binary-data-item indicated by pointer identifier is updated and points to a position immediately after the last position affected by the editing.
The first character in the string data item is counted as zero when setting the pointer.
Instead of a format-list-identifier, operand-3 may be a reference to a string data-item. This data-item must contain format-list characters as present in the format-literalpool. (output CREDIT linker). Item size must be great enough to contain these characters. The CALL FMOVE instruction may be used to fill the data-item.

Condition register: Unchanged.

Example : EDSUB ⊔ BUF, P1, FRM001

Intermediate code format:

Byte 1	0	1	1	0	1	1	0	L
operand-1	data-item-identifier-1							
operand-2	pointer-identifier							
operand-3	format-list-identifier							

Byte 1 is the operation code (X'6C', X'6D')
operand-1 is a reference to a string-data-item.
operand-2 is a reference to a binary-data-item.
L=1 operand-3 is a reference to a format list.
L=0 operand-3 is a reference to a string data item.

EDWRT *Edit and Write* **EDWRT**

Syntax: [statement-identifier] EDWRT [,NW] data-set-identifier,
{format-list-identifier }
{data-item-identifier }

Type: I/O instruction.

Description: The data items specified in the format list are edited into a buffer with a format as indicated by the format list. After editing, the buffer is output to the device indicated by the data set.
Format-list-identifier is a reference to an edit format list which is composed of format declarations (FRMT, FCOPY, FMEL, etc.).
Instead of a format-list-identifier, operand 2 may be a reference to a string data-item. This data item must contain format-list characters as present in the format list (translated by CREDIT linker). Item size must be great enough to contain these characters. The CALL FMOVE instruction may be used to fill the data-item.
Data-set-identifier is a reference to the relevant data set. ,NW, indicates that the no wait option is required.
One buffer per output data set is allocated to the program. The size of the buffer, is defined in the data set declaration. An EDWRT operation on a data set should be completed before another EDWRT is executed on that data set.
The first two bytes in the buffer can contain a control character. The first byte must always be unequal to zero and the second byte contains the control character. The function of the control character is device dependent. The control character may have the following values:

● General Terminal Printer or line printer:
X'2B' Print the line without advancing the paper.
X'30' Advance two lines before printing.
X'31' Skip to top of form before printing. (only for line printer)
Other codes: One line feed and carriage return is executed before printing.

Special characters allowed in the user buffer and not restricted to the first word in the buffer:
X'11' Tabulation character. This character should be followed by two ISO-7 digit characters giving the tabulation position. (only for GTP).

● Teller terminal printer (PTS6222, PTS6223):
Voucher/passbook printing.
X'2B' Print the line without advancing the paper.
X'30' Advance two line steps before printing
X'31'–X'39' Advance paper 1–9 line steps before printing.
Other codes: One line step and carriage return is executed before printing.

Journal/tally roll printing.
X'30' Advance two line steps before printing. (two steps = one line feed)
Other codes: One line step is executed before printing.

Special characters allowed in the user buffer:
X'09' The printhead is moved to the rightmost print position of the voucher. This character should be present in the last buffer position.
X'0D' The printhead is moved to the rightmost position of the journal station. This character should be present in the last buffer position.

Video display or plasma display
X'2B' The text is displayed from current cursor position.
X'30' Cursor is advanced two lines and positioned at the beginning of the line, before the text is displayed.
X'31' Erase display and position cursor on home position before the text is displayed.
Other codes: Advance cursor one line before the text is displayed.

- Teller terminal printer PTS6371
  The control character present in the second character of the buffer, as follows:
  /2B – printing is carried out from the last position of the previously printed line on this device. However, if the character pitch has been set, or if positioning has been carried out to the same line, since the previous line was printed, the printing will be from the tabulation position on the present line.
  /30 – the paper is advanced two lines, and the printing carried out from the tabulation position.
  /31 – journal: the paper is advanced three lines and the printing carried out from the tabulation position. This will make the previously written data readable through the window on the journal station.
  – document: printing is started from the tabulation position on line 1.
  Any other value in the control code will cause one line feed before printing from the tabulation position.
  The requested length must include the two bytes used for the control code, but if it is two, only the action specified by the control code is carried out.
  The maximum line length on the two print stations is limited to the following, based on normal character width.

	Journal	Document
10 characters/inch	33	80
12 characters/inch	40	
15 characters/inch	50	

One expanded character equals two normal characters.

Special characters allowed in the user buffer and not restricted to the first word in the buffer :

Characters Valid for All Displays

/AE : Displayed as point (/E2)

/11 : Tabulation character. This character should be followed by two ISO-7 digits giving the tabulation position.

/07 : Bell is sent to the display.

Characters Valid for PTS 6344 only

/12 : Underline start. Output of characters which follow this character are provided with underline.

/13 : Underline stop. Output of characters which follow after this character are not provided with underline. Underline stop mode will also appear at request end.

/14 : Fast output. First character following /14 will be transmitted in fast output mode up to requested length.
Note that cursor will remain unchanged.

/1C : Data to keyboard.

/1D : Master clear to keyboard

/1E : Low intensity start. Output of characters which follow after this character, are displayed at low intensity.

/1F : Low intensity stop. Output of characters which follow after this character are displayed at normal intensity.
Normal intensity mode will also appear at request end.

Characters Valid for PTS 6371 printer only

/12 : Underline start. Output of characters which follow this character are provided with underline.

/13 : Underline stop, Output of underlined characters stops. Underlining also stops at request end .

/19 : Start / Stop expanded character mode. Characters following the first occurrence of this character in the buffer are printed as double width characters, until the next occurrence of this code in the buffer.

/1A : Each character in the range /30 – /3C which is preceded by this code is printed as an OCR–A character. Any other legal character preceded by this code is printed as a space.

/1B : Each of the characters described below, which is preceded by this code, is printer as a special character. Any other legal code that is preceded by this code is printed as a space.
Codes /20–/29 are for use when the National Character Variation currently in use (see DSC1) does not contain the character required. They are printed as Space, § , @, #, ◇ , £, Space, ¤ , → and ↓ respectively.
Codes /30–/39 are printed as numerics with a greater width than normal, and are more the size of alphabetic characters.
Codes /3A–3F are logotypes, defined by the user. The character generator for these codes is a separate unit which must be in the printer. If it is not, these characters are printed as spaces.

/AE : Each character in the range /30–/39 which is preceded by this code is printed as a roomless point numeric.
Any other legal character preceded by this code is printed as a space.

Condition register: = 0 if I/O successful (OK)
= 1 if End of file (EOF)
= 2 if Error (ERR)
= 3 if Begin or End of Device (BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	OK	EOF	ERR	Unconditional

Example: EDWRT DSTPTR, FRM001

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	L
Byte 2	external reference							
operand-1	W	data-set-identifier						
operand-2	format-list-identifier							

Bytes 1 and 2 are filled by the system. Byte 2 contains a reference to an external system routine.
W is the wait bit.
W = 0 no wait
W = 1 wait
Operand-1 is a reference to the relevant data set.
10/100 refers to the first data set.
L = 0 operand-2 is a reference to a string-data-item.
L = 1 operand-2 is a reference to a format list.

ERASE *Erase* ERASE

Syntax: [statement-identifier] ⊔ ERASE ⊔ control value,

data-item-identifier-1, $\left\{\begin{array}{l}\text{data-item-identifier-2}\\\text{literal constant}\end{array}\right\}$

Type: Format control I/O

Description: Depending on control value, one of the following operations, on the current format list, is performed.

Control Value	Significance	
0	The lines, ranging from the line number contained in the binary data item, referenced by data-item-identifier-1, to the line number contained in the binary data item referenced by data-item-identifier-2 are erased on the screen. When the second line number (referenced by data-item-identifier-2) is zero, then all lines of the current format list are erased starting at the line number contained in the data-item referenced by data-item-identifier-1. Both data items may contain the same line number.	
1	All input fields (FKI+FINP) of the current format list with an input field number ranging from the number contained in the binary data item referenced by data-item-identifier-1 up to the number contained in the binary data-item referenced by data-item-identifier-2 are erased on the screen.	
2	As control value 1, but also data-items belonging to the input field are cleared.	
3	As control value 1, but only data-items belonging to the input field are cleared.	
4	As control value 1, but erasing is not performed on input fields with the "NCLR" flag set.	
5	As control value 2, but erasing is not performed on input fields (and the corresponding data items) with the "NCLR" flag set.	
6	As control value 3, but no resetting on to the input fields belonging data items is performed, which have the "NCLR" flag set.	
9	As control value 1	Note:
10	As control value 2	These control values are similar to the control values 1 up to including 6, but only FKI-type input fields are taken into account.
11	As control value 3	
12	As control value 4	
13	As control value 5	
14	As control value 6	

The last field number to be erased may also be indicated by a literal constant of the type binary.

ERASE *Continued* **ERASE**

Condition register: = 0 if OK
= 2 if ERROR

Condition mask:

0	1	2	3	4	5	6	7
OK	–	ERROR	–	$\overline{\text{OK}}$	–	$\overline{\text{ERROR}}$	UNCONDITIONAL

Example: ERASE, SPBINW1, = '0'

Intermediate object code:

Byte 1	0	0	1	1	0	0	0	L
Byte 2	external reference							
operand-1	control value							
operand-2	data-item-identifier-1							
operand-3	data-item-identifier-2							

Bytes 1 and 2 are filled by the system.
operand-1 is a control value.
operands-2,3 are references to binary data items.
L=1 operand-3 is a reference to a literal constant.
L=0 operand-3 is a reference to a data item.

EXIT *Exit* EXIT

Syntax: [statement-identifier] ⊔EXIT

Type: Scheduling instruction.

Description: Execution of the task is terminated, but may be restarted by the activate instruction.

Condition register: Not significant.

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.

GETABX *Get Current input field number* GETABX

Syntax: [statement-identifier] ⊔ GETABX ⊔ data-item-identifier

Type: Format control I/O

Description: The number of the current input field is returned in a binary data item referenced by data-item-identifier. When no input field is current, the content of the binary data item will be zero. The field type is indicated in the condition register.

Condition register:
= 0 The current input field is an FKI-type field
= 1 The current input field is an FINP-type field
= 2 No input field is current.

Condition mask:

- 0	1	2	3	4	5	6	7
FKI-type	FINP-type	NO CURRENT INP FLD	–	no FKI type	no FINP type	current inp fld.	UNCON-DITIONAL

Example: GETABX FLDNUM.

Intermediate object code:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	data-item-identifier							

Bytes 1 and 2 are filled by the system.
operand-1 is a reference to a binary item.

GETCTL *Get control value* GETCTL

Syntax: [statement-identifier] ⊔ GETCTL ⊔ control value,

data-item-identifier

Type: Format control I/O.

Description: One of the values, following the options APPL, MAXL, MINL or SCHK, from the current input field is transferred to a binary data item, referenced by data-item-identifier. Options are specified in the FKI– or FINP– format list declarations. Value zero returned if the requested option is not defined in the FKI or FINP description.

control value	significance
0	The "APPL" value is transferred.
1	The "MAXL" value is transferred.
2	The "MINL" value is transferred.
3	The "SCHK" value is transferred.

Condition register: Unchanged.

Example: GETCTL ⊔ 3, CHECK

Intermediate object code:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	control value							
operand-2	data-item-identifier							

Bytes 1 and 2 are filled by the system
Operand-1 is the control value
Operand-2 is a reference to a binary data item.

GETFLD *Get input field* GETFLD

Syntax: [statement-identifier] ␣ GETFLD ␣ control value,

data-item-identifier-1, data-item-identifier-2

Type: Format control I/O

Description: The input field, of the current format list, which input field sequence number is contained in the binary data-item referenced by data-item-identifier-1, becomes current. The field sequence numbering to be used is specified in control value, which must be a decimal value
0 for FKI-input field sequence numbering.
1 for FINP-input field sequence numbering,
2 for all input field sequence numbering. When the data-item-identifier-1 refers to a binary data-item with contents zero, the last input field of the specified type (in control value) will become current.
If, before the execution of this instruction any empty compulsory field was found (its corresponding data-item is empty and in the FKI-field the muster enter flag ME, was set), then after execution of this instruction the number of this compulsory field will be returned in a binary data-item referenced by data-item-identifier-2 and the condition register is set.
On a successful operation this data item contains zero.

Condition register: = 0 Operation successfully performed, no empty, compulsory field was found. (Compulsory field is defined in the FKI input field declaration).
= 2 The addressed input field sequence number was not found within the current format list.
= 3 Operation successfully performed but an empty compulsory field was found.

Condition mask:

0	1	2	3	4	5	6	7
OK	–	ERR	EMPTY	$\overline{\text{OK}}$	–	$\overline{\text{ERR}}$	Unconditional

Example: GETFLD 0, INPF1, ERFLD

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	control value							
operand-2	data-item-identifier-1							
operand-3	data-item-identifier-2							

Bytes 1 and 2 are filled by the system.
Operand-1 is the control value 0, 1 or 2
Operands-2, 3 are references to binary data items.

GETID *Get task identifier* GETID

Syntax: [statement-identifier] ␣ GETID ␣ data-item-identifier

Type: Scheduling instruction.

Description: The current task identity is transferred to a data-item indicated by data-item-identifier.
The data-item may be of the type binary or string. In case of a string data-item only the first two character positions are affected.

Condition register: Unchanged.

Example: GETID, TASKID

Intermediate code format:

<table>
<tr><td>Byte 1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>Byte 2</td><td colspan="8">external reference</td></tr>
<tr><td>operand-1</td><td colspan="8">data-item-identifier</td></tr>
</table>

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.
Operand-1 is a reference to a binary or string data item

GETTIME *Get clock* GETTIME

Syntax: [statement-identifier] ␣ GETTIME ␣ data-item-identifier

Type: clock control.

Description: The current time of the system clock is returned in a string data item indicated by data-item-identifier.
The string data item must have a length of at least six characters.
The time is returned as |H|H|M|M|S|S|

H = hour
M = minute
S = second

Condition register: Unchanged.

Example: GETTIME TIME

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	data-item-identifier							

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.
Operand-1 is a reference to a string data item.

IASSIGN *Assign index file* IASSIGN

Syntax: [statement-identifier] ␣ IASSIGN ␣ data-set-identifier, data-item-identifier, file-name-identifier-1, file-name-identifier-2, volume-name-identifier

Type: I/O instruction

Description : The index file name (8 bytes including trailing blanks), present in the string-data-item referenced by file-identifier-1, is assigned to the data file referenced by data-set-identifier. The file code in the data set, which is already used for the data file assignment, now determines to which data file this index file will be assigned. The master index file, which name (8 bytes) is contained in the string-data-item referenced by file-name-identifier-2, is read into memory. Volume name identifier refers to a string-data-item (6 characters inclusive trailing blanks) in which is a reference to the volume on which master index file and index file are present. Maximum four index files may be assigned to one data file using different file codes. Index files must be assigned as common files.

Before an index file is assigned, the data file must be assigned. If an assignment is unsuccessful an error code is returned in the binary data item referenced by data-item-identifier.
The contents of this data item may be:

0	assignment successful performed
-1	request error
1	Disk I/O error
2	No free entry in common device table.
3	Not sufficient memory space available for master index or file descriptor blocks
4	Volume name unknown
5	File already assigned from this task
6	File name unknown
7	File section missing or found twice
8	Faulty disk format
9	More than 4 extents exist
10	No data file assigned.
11	4 index files already assigned
12	Size of disk buffers not sufficient
13	Request busy. Reissue request.

Condition register: = 0 if assignment successful
= 2 if assignment unsuccessful

Condition mask :

0	1	2	3	4	5	6	7
SUCC	–	UNSUCC	–	$\overline{\text{SUCC}}$	–	$\overline{\text{UNSUCC}}$	Unconditional

Example: IASSIGN ␣ DFILE, ERRCODE, INDXFIL, MIXFIL, VOLNAM

IASSIGN *Continued* IASSIGN

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	0	0	data-set-identifier					
operand-2	data-item-identifier							
operand-3	file-name-identifier-1							
operand-4	file-name-identifier-2							
operand-5	volume name-identifier							

Bytes 1 and 2 are filled by the system.
Byte 2 contains a reference to an external system routine.
Operand-1 is a reference to a data set.
10/100 refers to the first data set.
Operand-2 is a reference to a binary data item
Operands-3, 4, 5 are references to string data items.

IB — Indexed branch — IB

Syntax: [statement-identifier] ⊔ IB ⊔ index-identifier $\left\{\begin{array}{l}\text{, statement-identifier}\\ \text{, external-identifier}\end{array}\right\}\ldots$

Type: Branch instruction.

Description: A branch is made to one of the identifiers in the identifier list according to the contents of the data item specified by index-identifier. The first identifier in the list corresponds with the index value one. If the index is zero, or greater than the number of identifiers in the list, the instruction following the indexed branch is executed.

Condition register: Not significant.

Example: IB INDEX, SYS20, SYS40

Intermediate code format: (long branch)

Byte 1	0	0	1	1	0	0	1	0
operand-1	index-identifier							
Byte n	list length							
operand-2	statement-identifier-1							
operand-3	statement-identifier-n							

Byte 1 is the operation code (X'32').
Operand-1 refers to a binary data item.
Byte n is filled by the CREDIT translator and contains the number of identifiers present in the address list.
Operands-2,3etc. contain an index to a branch address table (T:BAT).

Intermediate code format: (short branch)

Byte 1	0	0	1	1	0	1	1	B
operand-1	index-identifier							
Byte n	list length							
Byte n+1	displacement-1							
Byte n+2	displacement-n							

Byte 1 is the operation code (X'36', X'37').
B = 0 forward branching
B = 1 backward branching
Operand-1 refers to a binary data item.
Byte n is filled by the CREDIT translator and contains the number of identifiers present in the address lits.
Bytes n+1, n+2 contain a displacement.

IINS Indexed Insert IINS

Syntax: [statement-identifier] ⊔ IINS ⊔ [.NW,]
data-set-identifier, data-item-identifier-1,
data-item-identifier-2

Type: I/O - instruction.

Description: The data record, present in the string data-item referenced by data-item-identifier-1, is written as a new record to the data file referenced by data-set-identifier and all associated index files are updated. All index files must be assigned (as common files) when this instruction is executed. The new data record will be written after the last record, pointed by last record number pointer (LRN), in the file. The last record number pointer is updated. If an index record with the same key already exists, the new record is placed before the old one.
In the binary-data-item referenced by data-item-identifier-2 is returned the number of remaining records in the data file. If this number is greater than 32.767, 32.767 is returned.
When in the status code bit 10, "End of medium" is obtained, one index record is lost, and the index files must be rebuilt
Bit 3 "End of File" can be used as a warning for this situation.

Condition register:

= 0 if I/O successful	(OK)
= 1 if End of file	(EOF)
= 2 if error	(ERR)
= 3 if Begin or End of device	(BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	Unconditional

Example: IINS DSDF1, BUF1, FRNUM

Intermediate object code:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	0	data-set-identifier					
operand-2	data-item-identifier-1							
operand-3	data-item-identifier-2							

Bytes 1 and 2 are filled by the system
Byte 2 contains a reference to an external system routine
W is the wait bit
W=0 no wait
W=1 wait
operand 1 is a reference to a data set.
10/100 refers to the first data set.
Operand 2 is a reference to a string data item.
Operand 3 is a reference to a binary data item.

INSRT *Insert* INSRT

Syntax: [statement-identifier] INSRT data-item-identifier-1, pointer-identifier-1, size-identifier, data-item-identifier-2, pointer-identifier-2

Type: String instruction.

Function: (Operand-4) $\xrightarrow{\text{inserted}}$ Operand 1
(pointer-identifier-2) (pointer-identifier-1)

Description: Starting at pointer-identifier-2, the contents of operand-4 are inserted into operand-1 beginning at pointer-identifier-1.
The number of characters to be inserted is given in the data item specified by size-identifier. This insertion is accomplished by shifting the original contents of operand-1 to the right starting at pointer-identifier-1.

If a non-space or non-zero character is shifted out of operand-1, the condition register is set to overflow. Each character shifted out is lost.

The first characters of operand-1 and operand-4 are counted as zero when setting the pointers.

Operand-1 and operand-4 must be string data items.

Condition register: = 3 if Overflow

Condition mask:

0	1	2	3	4	5	6	7
–	–	–	over-flow	–	–	–	–

Example: INSRT TEXT1,P1,LNGTH,TEXT2,P2

Intermediate code format:

Byte 1	0	1	1	0	0	1	0	0
operand-1	data-item-identifier-1							
operand-2	pointer-identifier-1							
operand-3	size-identifier							
operand-4	data-item-identifier-2							
operand-5	pointer-identifier-2							

Byte 1 is the operation code (X'64').
Operands-1,4 are references to string data items.
Operands-2,3,5 are references to binary data items.

INV *Invert* INV

Syntax: [statement-identifier] ⊔ INV ⊔ data-item-identifier

Type: Logical instruction

Function: $\overline{\text{(data-item-identifier)}}$→ data-item-identifier

Description: The content of data-item-identifier is inverted (replaced by complement).

Date-item-identifier must refer to a boolean data item (length 1 bit). The condition register is set according to the *previous* value of data-item-identifier.

Condition register: = 0 if (data-item-identifier) = 0

Condition mask:

0	1	2	3	4	5	6	7
DI=0	–	–	–	DI≠0	–	–	–

Intermediate code format:

Byte 1	0	1	0	0	0	0	1	0
operand-1	data-item-identifier							

Byte 1 is the operation code (X'42').

Operand-1 is a reference to a boolean data item.

IREAD *Indexed Random Read* IREAD

Syntax: [statement-identifier] ␣ IREAD ␣ [.NW,] [.NEA,]
data-set-identifier, data-item-identifier-1,
size-identifier, data-item-identifier-2

Type: I/O instruction

Description: A data record is read from the data file indicated by data-set-identifier and stored in a string data-item indicated by data-item-identifier-1. The string-data-item indicated by data-item-identifier-2 must contain the symbolic key of the desired record.
The number of requested bytes is put in the binary data-item-referenced by size-identifier, which on completion of this instruction will contain the number of bytes transferred.
.NW indicates that no wait option is required.
.NEA indicates that exclusive access should not be set for this record.
On a successful read the accessed record is available for this task under exclusive access.
Exclusive access is automatically released after:

- a write or rewrite of the record.
- a delete function.

Exclusive access may be released explicity by the "Release exclusive access" function.
The current record number (CRN) will point to the current data record and a CRN will point to the current index record.

Condition register:

= 0 if I/O successful	(OK)
= 1 if End of file	(EOF)
= 2 if Error	(ERR)
= 3 if Begin or End of device	(BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	Unconditional

Example: IREAD .NEA, DSDK1, BUF1, SIZE, KEY

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	EA	data-set-identifier					
operand-2	data-item-identifier-1							
operand-3	size-identifier							
operand-4	data-item-identifier-2							

Bytes 1 and 2 are filled by the system
Byte 2 contains a reference to an external system routine.
W is the wait bit.
EA is the exclusive access bit.

W=0 no wait	EA=1 no exclusive access
W=1 wait	EA-0 exclusive access

Operand-1 is the reference to the data set.
10/100 refers to the first data set.
Operands-2,4 are references to string data items.
Operand-3 is a reference to a binary data item.

IRNEXT *Indexed Read Next* IRNEXT

Syntax: [statement-identifier] ␣ IRNEXT ␣ [.NW,] [.NEA,]
data-set-identifier, data-item-identifier-1,
size-identifier

Type: I/O instruction

Description: The data-record, with the symbolic key following the previous symbolic key in the index file, is read when the instruction executed before was an indexed random read, indexed insert or indexed read next.
The contents of the data-record will be stored in the string data-item referenced by data-item-identifier-1. Data-set-identifier refers to a data-file. The number of requested bytes is put in the binary data-item referenced by size-identifier, which on completion will contain the number of bytes transferred.
.NW indicates that no wait option is required
.NEA indicates that exclusive access should not be set for this record.
Exclusive access is automatically released after:

- a write or rewrite of the record
- a delete function.

Exclusive access may be released explicity by the "Release exclusive access" function. The current record number (CRN) will point to the current data record and a CRN will point to the current index record.

Condition register:

= 0 if I/O successful	(OK)
= 1 if End of file	(EOF)
= 2 if Error	(ERR)
= 3 if Begin or End of device	(BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	Unconditional

Example: IRNEXT DSDK1, BUF1, SIZE

IRNEXT *Continued* **IRNEXT**

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	EA	data-set-identifier					
operand-2	data-item-identifier-1							
operand-3	size-identifier							

Bytes 1 and 2 are filled by the system
Byte 2 contains a reference to an external system routine
W is the wait bit.
EA is the exclusive access bit.
W=0 no wait EA=1 no exclusive access.
W=1 wait EA=0 exclusive access
Operand-1 is a reference to a data set
10/100 refers to the first data set.
Operand-2 is a reference to a string data item.
Operand-3 is a reference to a binary data item.

IRWRITE *Indexed Rewrite* IRWRITE

Syntax: [statement-identifier] ⊔ IRWRITE ⊔ [.NW,]
data-set-identifier, data-identifier-1,
data-item-identifier-2

Type: I/O instruction.

Description: The data-record indicated by its logical record number, which is present in a binary or decimal data-item referenced by data-item-identifier-2, will be overwritten with the contents of the buffer referenced by data-item-identifier-1, except for the key field.
Data-set-identifier refers to the data file to be processed.
The record must be under exclusive access, which is released after a successful rewriting of the record. Also all index files must be assigned (as common files) when this instruction is executed.
When the key areas in the new data record and the old one, are not the same, bit 1 (key not found) will be set in the status code.
.NW indicates that no wait option is required.

Condition register:

= 0 if I/O successful (OK)
= 1 if End of File (EOF)
= 2 if Error (ERR)
= 3 if Begin or End of device (BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	Unconditional

Example: IRWRITE ⊔ DSDK1, BUF1, RECNR

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	0	data-set-identifier					
operand-2	data-item-identifier-1							
operand-3	data-item-identifier-2							

Bytes 1 and 2 are filled by the system
Byte 2 contains a reference to an external system routine
W is the wait bit.
W=0 no wait
W=1 wait
Operand-1 is a reference to a data set.
10/100 refers to the first data set.
Operand-2 is a reference to a string data item.
Operand-3 is a reference to a binary or decimal data item.

KI Keyboard input KI

Syntax: [statement-identifier]⊔KI ⊔[.NW,][.NE,] data-set-identifier, data-item-identifier, key-table-identifier, size-dientifier, index-identifier

Type: I/O instruction.

Description: Alphanumeric characters are read from the keyboard indicated by data-set-identifier and stored in the string data item indicated by operand-4. The task waits, until transfer is completed.

The number of requested characters is given in the data item specified by size-identifier, which on completion of input will contain the number of characters transferred. The data item specified by index identifier is filled with the position number of the terminating character in the key table. The first character of the key table is counted as one. Key-table-identifier refers to the relevant key-table. .NW and .NE indicate that the no wait and no echo options are required.

KI can also be used to read the SOP switches. In this case the pointer contains the position number of the pressed switch. The rightmost switch on the SOP panel is counted as one.

Transfer of alphanumeric characters is ended if:

1) One of the terminating characters listed in the key table is input.
2) A character neither alphanumeric nor listed in the key table is input.
3) The size of the string data item is reached.
4) Power failure occurs.
5) Requested number of characters is reached.
6) A keylock switch is turned.

In case 2), 3) and 5) above the pointer will contain an undefined value and the condition register will be set to ERROR.
In case of a power failure the pointer is set to zero and no indication is given in the condition register.
All character positions not affected by the input are set to X'00'.
In case 6) the index value will be negative, thus indicating that a key-lock switch is turned.
The possible negative values in the index for keyboards PTS6236, 6271 and 6272 are:
−1: key-lock no.4 turned OFF
−2: key-lock no.3 turned OFF
−3: key-lock no.2 turned OFF
−4: key-lock no.1 turned OFF
−5: key-lock no.4 turned ON
−6: key-lock no.3 turned ON
−7: key-lock no.2 turned ON
−8: key-lock no.1 turned ON
If all keys are OFF, the keyboard is considered to be inactive.

KI Continued KI

Condition register:

= 0 if I/O successful (OK)
= 1 if End of file (EOF)
= 2 if Error (ERR)
= 3 if Begin or end of device (BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{OK}$	$\overline{EOF}$	$\overline{ERR}$	unconditional

Example: KI DSKB,INBUF,KTAB1,INLEN,INDEX

Intermediate code format:

Byte 1	0 0 1 1	0 0 0 0
Byte 2	external reference	
operand-1	W E	data-set-identifier
operand-2	operand-4	
operand-3	key-table-identifier	
operand-4	size-identifier	
operand-5	index-identifier	

Bytes 1 and 2 are filled by the system. Byte 2 contains a reference to an external system routine.
Byte 3 Bit 0 is the wait bit.
W is the wait bit
W=0 no wait
W=1 wait
E is the echo bit
E=0 no echo
E=1 echo
Operand-1 is the reference to the relevant data set.
10/100 refers to the first data set.
Operand-2 is a reference to a string data item.
Operand-3 is a reference to a key table which is assumed to be literal.
Operands-4,5 are references to binary data items.

LB *Long branch* LB

Syntax: [statement-identifier]⊔LB⊔ $\left[\begin{Bmatrix}\text{equate-identifier,}\\ \text{condition-mask,}\end{Bmatrix}\right]\begin{Bmatrix}\text{statement-identifier}\\ \text{external-identifier}\end{Bmatrix}$

Type: Branch instruction.

Description: The next instruction to be executed is indicated by operand-2, when operand-1 matches the contents of the condition register. Otherwise the instruction following the long branch instruction will be executed.

If operand-1 is omitted, an unconditional branch (value 7) is generated.

Condition register: Not changed.

Example:

```
LB    SYSOPN
LB    3,SYSCLOS
```

Intermediate code format:

Byte 1	0 0 1 1	1	CND
Byte 2	Index to T:BAT		

Byte 1 is the operation code (X'38' up to X'3F')
CND is the condition mask field.
Byte 2 contains an index to a branch address table (T:BAT).

MATCH *Match* MATCH

Syntax: [statement-identifier] ␣ MATCH ␣ data-item-identifier-1, pointer-identifier-1, size-identifier-1, data-item-identifier-2, pointer-identifier-2, size-identifier-2

Type: String instruction.

Function:
(Operand-4) → (Operand-1)
(pointer-identifier-2) (pointer-identifier-1)

Description: This instruction searches the specified part of operand-1 in an attempt to find a match with the specified part of operand-4. The parts of operand-1 and operand-4 involved are defined by their respective pointer-identifier and size-identifier. The search commences at the character in operand-1 indicated by pointer-identifier-1, and continues for the number of characters specified in size-identifier-1. The characters in operand-4 to be searched for begin at the position specified by pointer-identifier-2. The number of characters to be searched for is specified by size-identifier-2.

If a match is found, pointer-identifier-1 will contain the address within operand-1 at which the match occurs and the condition register is set to *zero.* (Equal).

If no match occurs, pointer-identifier-1 will have an undefined value.

Operand-1 and operand-4 must be string data items.

The first characters in operand-1 and operand-4 are counted as zero.

Condition register: = 0 if match.

Condition mask:

0	1	2	3	4	5	6	7
EQUAL	–	–	–	UNEQUAL	–	–	–

Example: MATCH TEXT1,P1,L1,TEXT2,P2,L2

MATCH *Continued* **MATCH**

Intermediate code format:

Byte 1	0 1 1 0 | 1 0 0 0
operand-1	data-item-identifier-1
operand-2	pointer-identifier-1
operand-3	size-identifier-1
operand-4	data-item-identifier-2
operand-5	pointer-identifier-2
operand-6	size-identifier-2

Byte 1 is the operation code (X'68').
Operands-1,4 refer to string data items.
Operands-2,3,5,6 refer to binary data items.

MOVE *Move* MOVE

Syntax: [statement-identifier] ␣MOVE␣ data-item-identifier-1, $\left\{\begin{array}{l}\text{data-item-identifier-2}\\ \text{literal constant}\end{array}\right\}$

Type: Arithmetic instruction.

Function: (Operand-2) → operand-1

Description: Operand-2 is moved to operand-1. The contents of operand-2 remain unchanged.
The following mixed transfers are allowed:
Operand-1 is binary and operand-2 is decimal;
Operand-1 is decimal and operand-2 is string;
or
Operand-1 is decimal and operand-2 is binary.

These conversions are done according to the type of the receiving data item.

Condition register: Unchanged.

Rules:

Operand-2 / Operand-1	BIN	BCD	STRG
BIN	1	3	X
BCD	4	2	5
STRG	X	6	1

BIN→BIN
STRG→STRG

1. The content of operand-2 is moved to operand-1 from *left to right.* If the content of operand-2 is shorter than operand-1, the last character of operand-2 is repeated until operand-1 is filled. If operand-2 is longer than operand-1, the move ends when operand-1 is filled.

BCD→BCD

2. The content of operand-2 is moved to operand-1 from *right to left.* If operand-2 is shorter than operand-1, the remaining positions of operand-1 are filled by the character X'F'. If operand-2 is longer than operand-1, the move ends when operand-1 is filled. The sign is always moved to the leftmost (i.e. *most significant*) position.

BCD→BIN 3. The content of operand-2 is converted from decimal to binary and moved to operand-1. If the value of operand-2 is outside the range – 32768 to 32767, the result is unpredictable and overflow is indicated.

BIN→BCD 4. The content of operand-2 is converted from binary to decimal and moved to operand-1. If operand-1 is shorter than is required by the value of operand-2, the least significant digits only are moved. The sign is moved to the leftmost (i.e. most significant) position.

STRG→BCD 5. The content of operand-2 is converted from string to decimal and moved to operand-1 from *right* to *left.* Non-numeric ISO-7 characters in operand-2 are ignored (i.e. skipped). The sign of operand-1 is set negative if operand-2 contains a leading "–" sign. If operand-2 is shorter than operand-1, the remaining positions of operand-1 are filled by the character X'F'.

BCD→STRG 6. The contents of operand-2 is converted from decimal to string and moved to operand-1. from *left* to *right.* The BCD space characters, X'F' will be supressed. The sign in the decimal data item will be converted and stored in the first character position of the string data item referenced by operand-1.

X'B', plus sign is converted to '+' character
X'D', minus sign is converted to '–' character
X'O', zero is converted to ' ' character

If operand-1 is longer than operand-2, the result in the string data item is padded to the right with null characters (X'OO'). If operand-1 is shorter than operand-2 only the right most digits are moved.

Examples:

```
MOVE    FIELD1,=W'825'    FIELD1 is declared as BIN
MOVE    WORK1,INPBUF      WORK1 and INPBUF are declared as
                            BCD
MOVE    FIELD1,WORK1
MOVE    WORK1,FIELD1
MOVE    WORK1,=C'ABCDEF'
```

Intermediate code format:

Byte 1	0	0	0	0	0	0	0	L
operand-1	data-item-identifier-1							
operand-2	data-item-identifier-2							

Byte 1 is the operation code (X'00' or X'01').
Operand-1 is a reference to a data item.
L=1 operand-2 is a reference to a literal constant.
L=0 operand-2 is a reference to data-item-identifier-2, array-identifier-2 or a formal parameter.

MUL *Multiply* MUL

Syntax: [statement-identifier] ␣ MUL ␣data-item-identifier-1, $\left\{\begin{array}{l}\text{data-item-identifier-2}\\\text{literal constant}\end{array}\right\}$

Type: Arithmetic instruction.

Function: (Operand-1) x (Operand-2) → operand-1

Description: Operand-1 is multiplied by operand-2 and the result is placed in operand-1. The contents of operand-2 remain unchanged. A single data item may be used for both operand-1 and operand-2. In this case the data item is merely multiplied by itself. Both operands must be decimal or binary.

Condition register:

= 0 if (operand--1) = 0
= 1 if (operand—1) > 0
= 2 if (operand—1) < 0
= 3 if overflow

Example: MUL WORK1,INPBUF Both identifiers are declared as BCD or BIN.

Intermediate code format:

Byte 1	0	0	0	0	1	0	0	L
operand-1	data-item-identifier-1							
operand-2	data-item-identifier-2							

Byte 1 is the operation code (X'08' or X'09').
Operand-1 is a reference to a data item.
L=1 operand-2 is a reference to a literal constant.
L=0 operand-2 is a reference to data item or a formal parameter.

MWAIT *Multiple Wait* MWAIT

Syntax: [statement-identifier] ␣ MWAIT ␣ index-identifier, data-set-identifier-1 [,data-set-identifier-2] . . .

Type: I/O instruction.

Description: This instruction is used to wait for the completion of the first of a series of events initialized with the nowait option.
After completion of one of the events, execution is continued. The binary-data-item, referenced by index-identifier, will receive the index value of the data set in the list, which has just completed its operation.
First data set in the list will be referenced with index value 1.
The contidion register is set according to the status of the last operation of the relevant data set.

Condition register:

= 0 if I/O successful	(OK)
= 1 if End of file	(EOF)
= 2 if Error	(Err)
= 3 if Begin or End of device	(BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	uncond

Example: MWAIT INDX, DSKB1, DSGTP

Intermediate Code Format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	index-identifier							
byte n	list-length							
operand-2	0	0	data-set-identifier-1					
operand-3	0	0	data-set-identifier-2					

Byte 1 and 2 are filled by the system.
Byte 2 contains a reference to an external system routine.
Operand-1 is a reference to a binary data item.
byte n is filled by the CREDIT translator and contains the number of data sets present in the list.
Operands-2,3 etc. are the references to the relevant data sets.

NKI Numeric keyboard input NKI

Syntax: [statement-identifier]␣NKI␣ [.NW,] [.NE,] data-set-identifier, data-item-identifier, key-table-identifier, size-identifier index-identifier

Type: I/O instruction.

Description: Numeric characters are read from a keyboard indicated by data-set-identifier and stored in a string data item indicated by operand-4. The number of requested characters is given in the data item specified by size-identifier, which on completion of input will contain the number of characters transferred. The data item specified by index-identifier, is filled with the position number of the terminating character in the key table. The first character of the key-table is counted as one. Key-table-identifier refers to the relevant key table. .NW and.NE indicate that the no wait and no echo options are required.

Transfer of numeric characters is ended if:
1) One of the terminating characters listed in the key table is input.
2) A character neither numeric nor listed in the key table is input.
3) The size of the string data item is reached.
4) Power failure occurs.
5) Requested number of characters is reached.
6) A key-lock switch is turned

In case 2), 3) and 5) above, the pointer will contain an undefined value and the condition register is set to ERROR.

In case of power failure the pointer is set to zero and no indication is given in the condition register.

All character positions not affected by the input are set to X'00'.

In case 6) the index value will be negative, thus indicating that a key-lock switch is turned.

The possible negative values in the index for keyboards PTS6236, 6271 and 6272 are:
−1: key-lock no.4 turned OFF
−2: key-lock no.3 turned OFF
−3: key-lock no.2 turned OFF
−4: key-lock no.1 turned OFF
−5: key-lock no.4 turned ON
−6: key-lock no.3 turned ON
−7: key-lock no.2 turned ON
−8: key-lock no.1 turned ON
If all keys are OFF, the keyboard is considered to be inactive.

NKI *Continued* **NKI**

Condition register:	= 0 if I/O successful	(OK)
	= 1 if End of file	(EOF)
	= 2 if Error	(ERR)
	= 3 if Begin or End of device	BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	unconditional

Example: NKI DSKBN,INBUF,KTAB2,INLEN,INDEX

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	E	data-set-identifier					
operand-2	data-item-identifier							
operand-3	key-table-identifier							
operand-4	size-identifier							
operand-5	pointer-identifier							

Bytes 1 and 2 are filled by the system.
Byte 2 contains a reference to an external system routine.
W is the wait bit.
W=0 no wait
W=1 wait
E is the echo bit
E=0 no echo
E=1 echo
Operand-1 is a reference to the relevant data set.
10/100 refers to the first data set.
Operand-2 is a reference to a string data item.
Operand-3 is a reference to a key table which is assumed to be literal.
Operand-4 is a reference to a binary data item.
Operand-5 is a reference to a binary data item.

PAUSE *Pause* PAUSE

Syntax: [statement-identifier]␣PAUSE

Type: Scheduling instruction.

Description: The execution of the task is inhibited until a restart instruction is issued by another task.

Condition register: Not significant.

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are filled by the system. Byte 2 is a reference to an external system routine.

PERF *Perform* PERF

Syntax: statement-identifier ␣ PERF ␣ subroutine-identifier [,actual-parameter] . . .

Type subroutine control instruction.

Function: PP = subroutine-identifier.

Description: Control is given to a subroutine which is written in the CREDIT language. The subroutine may be in the same module as the perform instruction or in another CREDIT module.

The return address is saved on the stack. In a virtual memory system the current segment number, return address and parameter list are saved on the stack.

A maximum of eight parameters can be passed.

When an array element is passed, then the array name and index are passed each as a parameter. A format-table reference may also be passed, but not a reference to an element in the format-table. Literal parameters of the type "X" are not allowed.

Condition register: Not significant.

Example:

```
PERF␣SUB1,ARRAY,INDEX
PERF␣SUB2,DSVDU
```

Intermediate code format:

Byte 1	1 0 0 0 0 0 0 0
operand-1	subroutine identifier
operand-2	parameter

Byte 1 contains the operation code (X'80')
Operand-1 is a reference to a subroutine.
Operands-2,3 etc. are references to the parameters.

PERFI *Indexed perform* PERFI

Syntax: [statement-identifier] ␣ PERFI ␣ index-identifier, subroutine-identifier . . .

Type: Subroutine control instruction.

Description: Control is passed to the subroutine in the subroutine identifier list according to the contents of the data item specified by index-identifier.

The return address is saved on the stack. In a virtual memory system the current segment number, return address and parameter list are saved on the stack.

The first identifier in the subroutine list has the index value one. If the index is zero or greater than the number of identifiers in the subroutine list, the instruction following the indexed perform is executed.

In a system with MMU, literals, key tables and format lists can only be passed as parameter, using the PLIST directive. The number of parameters must be the same for each subroutine mentioned in this instruction. Literal parameters of the type 'X' are not allowed.

Condition Register: Not significant.

Intermediate code format:

Byte 1	0	0	1	1	0	0	1	1
operand-1	index-identifier							
Byte n	list-length							
operand-2	subroutine-identifier-1							
operand-3	subroutine-identifier-n							

Byte 1 is the operation code (X'33').

Operand-1 refers to a binary data item.

Byte n is filled by the CREDIT translator and contains the number of identifiers present in subroutine identifier list.

Operands-2,3etc. are references to the subroutine.

PRINT *print* **PRINT**

Syntax: [statement-identifier] ⊔ PRINT ⊔ data-set-identifier,
data-item-identifier-1, $\left\{\begin{array}{l}\text{data-item-identifier-2}\\ \text{literal}\end{array}\right\}$

Type: Format control : C

Description: From the current format list the line number contained in the binary data item referenced by data-item-identifier-1, up to the line number contained in the binary data-item referenced by data-item-identifier-2 is output to the data set indicated by data set identifier. The first line number on the output device is always one. The contents of the data-item (second line number) referenced by data-item-identifier-2 may be equal or greater than the first line number contained in the data-item referenced by data-item-identifier-1.
When the second line number is zero all the lines from the current format list are output to the data set from the first line number up to the last one. Two spaces, which serve as control character (i.e. line feed and carriage return), are always output as the last line. The second line number may be indicated by a literal of the type binary.

Condition register:

= 0 if I/O successful	(OK)
= 1 if End of file	(EOF)
= 2 if Error	(ERR)
= 3 if Begin or End of device	(BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	UNCON-DITIONAL

Example:

```
PRINT ⊔ DSGP, LINE 5, LINE 15
PRINT ⊔ DSGP, LINE 5, = '0'
```

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	0	0	data-set-identifier					
operand-2	data-item-identifier-1							
operand-3	data-item-identifier-2							

Bytes 1 and 2 are filled by the system.
operand-1 is a reference to the relevant data set.
operands-2,3 are references to binary data items.
L=1 operand-3 is a reference to a literal constant.
L=0 operand-3 is a reference to a data item.

READ *Read* READ

Syntax: [statement-identifier] READ [.NW,] [.NEA,] data-set-identifier
data-item-identifier, size-identifier

Type: I/O instruction

Description: Characters are read from the device indicated by data-set-identifier into a string data item indicated by operand-2.
.NW indicates that the no wait option is required.
The transfer of characters is ended if:

1) An end-of-record condition is encountered
2) The string size is reached

The number of characters which are transferred, is returned by the system in the data item specified by size-identifier.
If the data item size is exceeded, error is set in the condition register.

The disk sequential access method is as follows.
The record to be read depends on the last data management function called by this task. If no data management function has been called by this task for this file (indicated by data set identifier) after the file was assigned, the current record number (CRN) will point to the first record of the file to be read. The logical record number can be fetched with GET CURRENCY (DSC1). If the reading is successful, the record is set under exclusive access for this task.
.NEA option indicates that exclusive access should not be set for this record.
When the data-set-identifier refers to a data communication data set this instruction will read data from the line. Time out must be set before this instruction is executed, with the DSC1 instruction and control value X'0B'. In a DC task time out must be set to zero.

For intertask communication data-set-identifier refers to a data set in which the input file code is defined. When a READ (unaddressed) is issued by a task, first a check is performed on the queue of the task that issues the READ for RWRITE (addressed). If an addressed write is in the queue for this task, the instruction is completed. When no match occurs the WRITE (unaddressed) queue is checked, if not empty the first one is removed from the queue and the instruction is completed. Else the instruction is put into the queue for unaddressed read.

Condition register:
= 0 if I/O successful (OK)
= 1 if End of file (EOF)
= 2 if Error (ERR)
= 3 if Begin or End of device (BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	Unconditional

Example:
```
READ      DSKBN, BUF1, SIZE
READ      .NEA, DSDK1, BUF1, SIZE
```

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	E	data-set-identifier					
operand-2	data-item-identifier							
operand-3	size-identifier							

Bytes 1 and 2 are filled by the system
Byte 2 contains a reference to an external system routine.
W is the wait bit
W=0 no wait
W=1 wait
E is the echo bit
E=1 echo
E=0 no echo
Operand-1 is the reference to the relevant data set
10/100 refers to the first data set.
Operand-2 is a reference to a string data item.
Operand-3 is a reference to a binary data item.

RET *Return* **RET**

Syntax: [statement-identifier] ⊔RET⊔ [equate-identifier / decimal-integer]

Type: Subroutine control instruction.

Description: Control is passed back to the calling module and execution is continued at the instruction following the original perform or indexed perform instruction.

The return address is found on the stack. In a virtual memory system the proper segment number and return address are found on the stack.

Decimal-integer specifies a displacement (number of bytes) which has to be added to the normal return address. This displacement excludes the length of the parameter list, in bytes, which may have been passed to the subroutine.

Condition register: Not significant.

Example: RET
RET 2

Intermediate code format:

Byte 1	0	0	1	1	0	1	0	0
Byte 2	displacement							

Byte 1 is the operation code (X'34').
Byte 2 contains displacement.

RREAD *Random Read* RREAD

Syntax: [statement-identifier] ␣ RREAD ␣ [.NW,] [.NEA,]
data-set-identifier, data-item-identifier-1,
size-identifier, data-item-identifier-2

Type: I/O instruction

Description: A record is read from the file indicated by data-set-identifier and stored in a string data item indicated by data-item-identifier-1.
The number of requested characters is given in the data-item specified by size-identifier, which on completion of input will contain the number of characters transferred.
The logical record number is given in a binary or decimal data item indicated by data-item-identifier-2.
.NW indicates that no wait option is required.
.NEA option indicates that exclusive access should not be set for this record.
On a successful read, the accessed record is available for this task under exclusive access. (Not accessible by other tasks).
The current record number (CRN) will point to the current data record.
Exclusive access is automatically released after:

- a write of the record
- a delete function.

The exclusive access may be released explicity by the "Release exclusive access" function.
For intertask communication data-set-identifier refers to a data set in which the input file code is defined.
Data-item-identifier-2 refers to a binary data item, which contains the task identifier of the addressed task. If the addressed task has not issued a RWRITE (to this task) or WRITE, then this request will be queued on the addressed task. In the other case the instruction will be completed.

Condition register:

= 0 if I/O successful	(OK)
= 1 if End of File	(EOF)
= 2 if Error	(ERR)
= 3 if Begin or End of device	(BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	Unconditional

Example: RREAD DSDK, BUFRC, LENGTH, RECNR

RREAD *Continued* **RREAD**

Intermediate
code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	EA	data-set-identifier					
operand-2	data-item-identifier-1							
operand-3	size-identifier							
operand-4	data-item-identifier-2							

Bytes 1 and 2 are filled by the system
Byte 2 contains a reference to an external system routine.

W is the wait bit.	EA is the exclusive access bit.
W=0 no wait	EA=0 exclusive access.
W=1 wait	EA=1 no exclusive access.

Operand-1 is the reference to the relevant data set.
10/100 refers to the first data set.
Operand-2 is a reference to a string data item.
Operand-3 is a reference to a binary data item.
Operand-4 is a reference to a binary or decimal data item.

RSTRT *Restart* RSTRT

Syntax: [statement-identifier] ⊔RSTRT⊔ task-identifier

Type: Scheduling instruction.

Description: The task in pause mode indicated by task-identifier is restarted. Task-identifier is a reference to a binary or string data item. In case of a string data item the first two bytes must contain the task identity.

Condition register: Not significant.

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	task-identifier							

Bytes 1 and 2 are filled by the system.
Byte 2 is an external reference to a system routine.
Operand-1 is a reference to a binary or string data item

RWRITE *Random Write* **RWRITE**

Syntax: [statement-identifier] ⊔ RWRITE ⊔ [.NW,]
data-set-identifier, data-item-identifier-1, data-item-identifier-2

Type: I/O instruction

Description: The record present in a string data item, indicated by data-item-identifier-1 is written to the file indicated by data-set-identifier. Before it is written the status of the record is checked whether it is "FREE" or "USED".
When "FREE" the status is changed to "USED" and the record is written.
When the status is "USED" the record will be written only if it is under Exclusive access for this task.
If it is not under exclusive access the error "record protected" will be sent.
The logical record number, for disc file, is in the binary or decimal data item indicated by data-item-identifier-2. After a random write, exclusive access is released. Random Write may be used to write on a display, data-item-identifier-2 refers in this case to a binary-data-item which contains the cursor position, where writing start.
For intertask communication, data-set-identifier refers to a data set in which the output file code is defined.
Data-item-identifier-2 refers to a binary data item, which contains the task identifier of the addressed task. If the addressed task has not issued a RREAD (to this task) or READ then this request will be queued on the addressed task. In the other case the instruction will be completed

Condition register:

= 0 if I/O successful	(OK)
= 1 if End of File	(EOF)
= 2 if Error	(ERR)
= 3 if Begin or End of Device	(BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	Unconditional

Example: RWRITE DSDK, BUFRC, RECNR

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	data-set-identifier						
operand-2	data-item-identifier-1							
operand-3	data-item-identifier-2							

RWRITE *Continued* **RWRITE**

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.
W is the wait bit
W=0 no wait
W=1 wait
Operand-1 is the reference to the relevant data set.
10/100 refers to the first data set.
Operand-2 is a reference to a string data item.
Operand-3 is a reference to a binary or decimal data item.

SB Short branch SB

Syntax: [statement-identifier]␣SB␣ $\left[\left\{\begin{array}{l}\text{equated-identifier,}\\ \text{condition-mask,}\end{array}\right\}\right]$ statement-identifier

Type: Branch instruction.

Description: The instruction to be executed is indicated by statement-identifier, if operand-1 matches the contents of the condition register. Otherwise the instruction following the short branch will be executed. If operand-1 is omitted an unconditional branch (value 7) is generated.

Statement-identifier may only refer to a statement which is within the limit of 255 bytes before the short branch (incl. 2 bytes of the short branch), or 255 bytes after the short branch.

Condition register: Not changed.

Example:

```
SB    INP3
SB    2,INP4
```

Intermediate code format:

Byte 1	0	1	0	1	B	CND
Byte 2	displacement					

Byte 1 is the operation code (X'50' up to X'5F').
B=0 forward branching.
B=1 backward branching.
CND is the condition mask field.
Byte 2 contains the displacement.

SET *Set* SET

Syntax: [statement-identifier]⊔SET⊔ data-item-identifier

Type: Logical instruction.

Function: 1 → data-item-identifier

Description: The content of data-item-identifier is set to one. (TRUE)
Data-item-identifier must refer to a boolean data item. (length 1 bit)
The condition register is set according to the *previous* value of the content of data-item-identifier.

Condition register: = 0 if (data-item-identifier) = 0

Condition mask:

0	1	2	3	4	5	6	7
DI=0	–	–	–	DI≠0	–	–	–

Intermediate code format:

Byte 1	0	1	0	0	0	0	0	1
operand-1	data-item-identifier							

Byte 1 is the operation code (X'41').
Operand-1 is a reference to a boolean data item.

SETCUR *Set Cursor* SETCUR

Syntax: [statement-identifier] ␣ SETCUR

Type: Format control I/O

Description The cursor will be positioned at the first character position of the current input field.

Condition register: = 0 if cursor positioned correctly
= 2 if I/O error

Condition mask:

0	1	2	3	4	5	6	7
OK	–	ERROR	–	$\overline{\text{OK}}$	–	$\overline{\text{ERROR}}$	Unconditional

Example: SETCUR

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are filled by the system
Byte 2 is a reference to an external system routine.

SETTIME *Set Clock* SETTIME

Syntax: [statement-identifier] ␣SETTIME ␣data-item-identifier

Type: Clock control.

Description: The system clock is set to the time specified in a string data item, indicated by data-item-identifier.
The string data item must have a length of six characters, in which is specified H H M M S S

H = hour
M = minute
S = second

The system clock is updated by the real time clock thus giving correct time of day.

Condition register: Unchanged.

Example: SETTIME, TIME

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	data-item-identifier							

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.
Operand-1 is a reference to a string data item.

SUB *Subtract* SUB

Syntax: [statement-identifier] ⊔ SUB ⊔data-item-identifier,$\left\{\begin{array}{l}\text{data-item-identifier-2}\\\text{literal constant}\end{array}\right\}$

Type: Arithmetic instruction.

Function: (Operand-1) – (Operand-2) → Operand-1

Description: Operand-2 is subtracted from operand-1 and the result is placed in operand-1.
Operand-2 is unchanged. Both operands must be binary or both operands must be decimal. The condition register is set according to the content of operand-1.

Condition register:
= 0 if (operand-1) = 0
= 1 if (operand-1) > 0
= 2 if (operand-1) < 0
= 3 if overflow

Condition mask:

0	1	2	3	4	5	6	7
=0	>0	<0	over flow	≠0	⩽0	⩾0	uncondi-tional

Example:

```
SUB    FIELD1,FIELD2      FIELD1 and FIELD2 are declared as BIN.
SUB    WORK1,=D'4317'     WORK1 is declared as BCD.
```

Intermediate code format:

Byte 1	0 0 0 0	0 1 0	L
operand-1	data-item-identifier-1		
operand-2	data-item-identifier-2		

Byte 1 is the operation code (X'04' or X'05').
L=1 operand-2 is a reference to a literal constant.
L=0 operand-2 is a reference to data-item-identifier-2, array-identifier-2 or a formal parameter.
Operand-1 is reference to a binary or decimal data item.

SWITCH *Switch task on same level* SWITCH

Syntax: [statement-identifier] ⊔ SWITCH

Type: Scheduling instruction.

Description: The running task will be interrupted and queued last in the dispatcher queue. Control is given to another task on the same level, which is the first one in the dispatcher queue.

Condition register: Unchanged.

Example: SWITCH

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Byte 1 contains the operation code (X'30')
Byte 2 is a reference to an external system routine.

TB *Test and branch* TB

Syntax: [statement-identifier]⊔TB⊔ $\left\{\begin{array}{l}\text{equate-identifier}\\ \text{condition-mask}\end{array}\right\}$, data-item-identifier, statement-identifier

Type: Branch instruction.

Description: The content of data-item-identifier is compared with zero and the condition register is set according to the result of this comparison. If operand-1 matches the condition register, the next instruction to be executed is found at the address specified by statement-identifier.
If operand-1 does not match the condition register, the instruction following the test and branch will be executed.

Statement-identifier may only refer to a statement which is within the limit of 255 bytes before the test and branch (incl. 3 bytes of the test and branch) or 255 bytes after the test and branch.

Data-item-identifier refers to a boolean data item.

Example: TB FALSE,LKMX1,TTGO

Condition register: = 0 if (data-item-identifier) = 0

Condition mask:

0	1	2	3	4	5	6	7
DI=0	–	–	–	DI≠0	–	–	–

Intermediate code format:

Byte 1	0	1	0	0	1	0	B	C
operand-1	data-item-identifier							
Byte n	statement-identifier							

Byte 1 is the operation code (X'48' up to X'4B').
B=0 forward branching.
B=1 backward branching.
C=0 condition mask is zero.
C=1 condition mask is four.
Operand-1 is a reference to a boolean data item.
Byte n contains a displacement.

TBF *Test and branch on false* TBF

Syntax: [statement-identifier] ␣ TBF ␣ data-item-identifier, statement-identifier

Type: Branch instruction.

Description: The content of data-item-identifier is compared with zero and the condition register is set according to the result of this comparision. If the content of operand-1 is zero, the next instruction to be executed is found at the address specified by statement-identifier. If the content of operand-1 is one, the instruction following the test and branch on false will be executed.
Statement-identifier may only refer to a statement which is within the limit of 255 bytes before the test and branch on false (incl. 3 bytes of the test and branch on false) or 255 bytes after the test and branch on false.
Data-item-identifier refers to a boolean data item.

Example: TBF LKMX1, TTGO

Condition register: = 0 if (data-item-identifier) = 0

Intermediate code format:

Byte 1	0	1	0	0	1	0	B	0
Operand-1	data-item-identifier							
Byte n	statement-identifier							

Byte 1 is the operation code (X'48' or X'4A').
B = 0 forward branching
B = 1 backward branching
Operand-1 is a reference to a boolean data item.
Byte n contains a displacement.

TBT   *Test and branch on true*   TBT

Syntax: [statement-identifier] ␣ TBT ␣ data-item-identifier, statement-identifier

Type: Branch instruction.

Description: The contents of data-item-identifier is compared with zero and the condition register is set according to the result of this comparision.
If the contents of operand-1 is one, the next instruction to be executed is found at the address specified by statement-identifier.
If the contents of operand-1 is zero, the instruction following the test and branch on true will be executed.
Statement-identifier may only refer to a statement which is within the limit of 255 bytes before the test and branch on true (incl. 3 bytes of the test and branch on true) or 255 bytes after the test and branch on true.
Data-item-identifier refers to a boolean data item.

Example: TBT LKMX1, TTGO

Condition register: = 0 if (data-item-identifier) = 0

Intermediate code format:

Byte 1	0	1	0	0	1	0	B	1
operand-1	data-item-identifier							
Byte n	statement-identifier							

Byte 1 is the operation code (X'49' or X'4B').
B = 0 forward branching
B = 1 backward branching
Operand-1 is a reference to a boolean data item.
Byte n contains a displacement.

TBWD — *Tabulate backward* — TBWD

Syntax: [statement-identifier] ␣ TBWD

Type: Format control I/O

Description: Tabulation backward from the current input field.
The current input-field number, according to the FKI-input field numbering, is decreased with one and this new input-field (current input field-1), of the current format list is made current, also when CTAB option was specified for this new current input field.

Condition register:
= 0 Operation successful.
Cursor is set to the first position of the new current input field
= 1 operation successful
Cursor remains in its old position, because the CTAB flag is set for this current input field.
= 2 Addressed input field not found in current format.
Cursor remains in its old position
= 3 An empty compulsory field was found before this instruction was executed
The compulsory field stays current and cursor remains in its old position

Condition mask:

0	1	2	3	4	5	6	7
SUCC	SUCC CTAB	NOT FOUND	EMPTY COMP. FIELD	$\overline{\text{SUCC}}$	NO SUCC CTAB	FOUND	UNCON- DITIONAL

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.

TDOWN *Tabulate Down* TDOWN

Syntax: [statement-identifier] ␣ TDOWN

Type: Format control I/O

Description: Tabulation to the nearest FKI-input field on the next line. The FKI-input field on the line immediately following the current line, with a starting column nearest to the starting column of the current input field, becomes current. When the two nearest columns are found on the next line, the left FKI-input field will become current. If no FKI-input field is found on the next line, the following lines are searched in sequence.

Condition register:
= 0 Operation successful.
Cursor is set to the first position of the new current input field
= 1 operation successful
Cursor remains in its old position, because the CTAB flag is set for this current input field.
= 2 Addressed input field not found in current format
Cursor remains in its old position.
= 3 An empty compulsory field was found before this instruction was executed.
The compulsory field stays current and cursor remains in its old position.
(Not relevant for THOME).

Example:

```
LINE 2   12   ①   20   ②   40   ③
LINE 4   12   ④   22   ⑤   40   ⑥
```

FKI-input field number 2, starting in column 20 is current. TDOWN results now in FKI-input field number 5 becoming current.

Condtion mask:

0	1	2	3	4	5	6	7
SUCC	SUCC CTAB	NOT FOUND	EMPTY COMP. FIELD	$\overline{\text{SUCC}}$	NO SUCC CTAB	FOUND	UNCON-DITIONAL

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Byte 1 is interpreted by the system.
Byte 2 reference to an external system routine.

TEST *Test* TEST

Syntax: [statement-identifier]⊔TEST⊔ data-item-identifier

Type: Logical instruction.

Function: (data-item-identifier) ↔ 0

Description: The content of data-item-identifier is compared with zero and the condition register is set according to the result of this comparison.

Data-item-identifier must refer to a boolean data item (length 1 bit).

Condition register: = 0 if (data-item-identifier) = 0

Condition mask:

0	1	2	3	4	5	6	7
DI=0	–	–	–	DI≠0	–	–	–

Intermediate code format:

Byte 1	0	1	0	0	0	0	1	1
operand-1	data-item-identifier							

Byte 1 is the operation code (X'43').
Operand-1 is a reference to a boolean data item.

TESTIO *Test I/O completion* TESTIO

Syntax: [statement-identifier] ␣ TESTIO ␣ data-set-identifier.

Type: I/O instruction.

Description: The data set indicated by data-set-identifier is tested for completion of the I/O. (without wait).

Condition register: = 0 if I/O is completed (OK)
= 1 if I/O is not completed (EOF)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	–	–	$\overline{\text{OK}}$	$\overline{\text{EOF}}$		uncon-ditional

Example: TESTIO DSVO

Intermediate object code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	0	0	data-set-identifier					

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.
Operand-1 is a reference to a data set.
10/100 refers to the first data set.

TFWD *Tabulate forward* TFWD

Syntax: [statement-identifier] ␣ TFWD

Type: Format control I/O

Description: Tabulation forward from the current input field.
The current input field number, according to the FKI-input field numbering, is increased with one and this new input field (current field number + 1) of the current format list is made current.

Condition register:

= 0 Operation successful
Cursor is set to the first position of the new current input field

= 1 operation successful
Cursor remains in its old position, because the CTAB flag is set for this current input field.

= 2 Addressed input field not found in current format.
Cursor remains in its old position

= 3 An empty compulsory field was found before this instruction was executed
The compulsory field stays current and cursor remains in its old position

Condition mask:

0	1	2	3	4	5	6	7
SUCC	SUCC CTAB	NOT FOUND	EMPTY COMP. FIELD	$\overline{\text{SUCC}}$	NO SUCC CTAB	FOUND	UNCON-DITIONAL

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.

THOME *Tabulate Home* THOME

Syntax: [statement-identifier] ⊔ THOME

Type: Format control I/O

Description: The first FKI-input field of the current format list is made current.

Condition register: = 0 Operation successful
Cursor is set of first position of the new current input field.
= 1 Operation successful
Cursor remains in its old position, because the CTAB flag is set for this current input field
= 1 Operation successful.
Cursor remains in its old position, because the CTAB flag is set for this current input field.
= 2 Not relevant
= 3 Not relevant

Condition mask:

0	1	2	3	4	5	6	7
SUCC	SUCC CTAB	NOT FOUND	EMPTY COMP. FIELD	$\overline{\text{SUCC}}$	NO SUCC CTAB	FOUND	UNCONDITIONAL

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.

TLDOWN *Tabulate left down* TLDOWN

Syntax: [statement-identifier] ␣ TLDOWN

Type: Format control I/O

Description: Tabulation to the left-most FKI-input field on the following line.
The first FKI-input field on the line immediately following the current one becomes current.
If no FKI-input field is found on that line, the following lines are searched in sequence.

Condition register:
= 0 Operation successful.
Cursor is set of first position of the new current input field.
= 1 Operation successful.
Cursor remains in its old position, because the CTAB flag is set for this current input field.
= 2 Addressed input field not found in current format.
Cursor remains in its old position
= 3 An empty compulsory field was found before this instruction was executed.
The compulsory field remains current and the cursor remains in its old position.
(Not relevant for THOME)

Condition mask:

0	1	2	3	4	5	6	7
SUCC	SUCC CTAB	NOT FOUND	EMPTY COMP. FIELD	$\overline{\text{SUCC}}$	NO SUCC CTAB	FOUND	UNCON-DITIONAL

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are filled by the system
Byte 2 is a reference to an external system routine.

TLEFT Tabulate left TLEFT

Syntax: [statement-identifier] ⊔ TLEFT

Type: Format control I/O

Description: Tabulation to the left-most input field on the current line.
The left-most FKI-input field on the same line as the current FKI-input field, becomes current.
Note: This input field always exists.

Contion register:
= 0 Operation successful
Cursor is set to the first position of the new current input field
= 1 Operation successful
Cursor remains in its old position, because the CTAB flag is set for this current input field
= 2 Not relevant
= 3 An empty compulsory field was found before this instruction was executed.
The compulsory field stays current and cursor remains in its old position.

Condition mask:

0	1	2	3	4	5	6	7
SUCC	SUCC CTAB	NOT FOUND	EMPTY COMP. FIELD	$\overline{\text{SUCC}}$	NO SUCC CTAB	FOUND	UNCON- DITIONAL

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are filled by the system.
Byte 2 is a reference to an external system routine.

TRIGHT *Tabulate right* TRIGHT

Syntax: [statement-identifier] ␣ TRIGHT

Type: Format control I/O

Description: Tabulation to the right-most input field on the current line.
The right-most FKI-input field on the same line as the current FKI-input field, becomes current.
Note: This input field always exists.

Condition register:

= 0 Operation successful
Cursor is set to the first position of the new current input field

= 1 Operation successful
Cursor remains in its old position, because the CTAB flag is set for this current input field

= 2 Not relevant

= 3 An empty compulsory field was found before this instruction was executed.
The compulsory field stays current and cursor remains in its old position.

Condition mask:

0	1	2	3	4	5	6	7
SUCC	SUCC CTAB	NOT FOUND	EMPTY COMP. FIELD	$\overline{\text{SUCC}}$	NO SUCC CTAB	FOUND	UNCON-DITIONAL

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are fully filled by the system
Byte 2 is a reference to an external system routine.

TSTCTL *Test control flag* TSTCTL

Syntax: [statement-identifier] ⊔ TSTCTL ⊔ control value

Type: Format control I/O

Description: One of the control flags, of the current input field is tested and the condition register is set according to the result.
The control flags are specified in a FKI-format list declaration.
Control value specifies which flag has to be tested.

Control value	Significance
0	Test "ALPHA" flag
1	Test "REWRT" flag
2	Test "ME" flag
3	Test "NEOI" flag
4	Test "NCLR" flag
5	Test "CTAB" flag
6	Test "VERIF" flag.

Condition register: = 0 when a flag is not set.

Condition mask:

0	1	2	3	4	5	6	7
NOT SET	–	–	–	SET	–	–	UNCONDITIONAL

Example: TSTCTL 2

Intermediate object code:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	control value							

Bytes 1 and 2 are filled by the system
Byte 2 is a reference to an external system routine.
Operand-1 is the control value.

TUP *Tabulate Up* TUP

Syntax: [statement-identifier] ␣ TUP

Type: Format control I/O

Description: Tabulation to the nearest FKI-input field on the preceding line. The FKI-input field on the line immediately preceding the current line, with a starting column nearest to the starting column of the current input field, becomes current. When the two nearest columns are found on the preceding line, the left FKI-input field will become current. If no FKI-input field is found on the preceding line, the line preceding that one is searched etc.

Condition register:

= 0 Operation successful
Cursor is set to the first position of the new current input field.

= 1 Operation successful
Cursor remains in its old position, because the CTAB flag is set for this current input field.

= 2 Addressed input field not found in current format.
Cursor remains in its old position.

= 3 An empty compulsory field was found before this instruction was executed. The compulsory field stays current and cursor remains in its old position.
(Not relevant for THOME).

Example:

```
          12   ①    20   ②    30   ③    40   ④
LINE 2   ______________________________________________
          12   ⑤               25      ⑥
LINE 4   _____________       ___________________
```

FKI-input field number 6, starting in column 25 is current.
TUP results now in FKI-input field number 2 becoming current.

Condition mask:

0	1	2	3	4	5	6	7
SUCC	SUCC CTAB	NOT FOUND	EMPTY COMP. FIELD	$\overline{\text{SUCC}}$	NO SUCC CTAB	FOUND	UNCON-DITIONAL

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							

Bytes 1 and 2 are filled by the system
Byte 2 is a reference to an external system routine.

UNUSE *Unuse* UNUSE

Syntax: [statement-identifier] ␣UNUSE␣ block-identifier

Type: Storage control instruction.

Description: The user workblock or swappable workblock, attached to the current task will be detached. A swappable workblock is restored on disk.

The condition register is unequal to zero if the referenced workblock does not exist.

Condition register:
= 0 workblock correctly detached from the task.
= 2 no workblock of this type was attached to the task.

Intermediate object code:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	UWB/SWB type							
Byte n	Block number							

Bytes 1 and 2 are filled by the system.
Operand-1 is type of user or swappable workblock.
Byte n is the index to the user or swappable workblock.

UPDFLD *Update Input Field* UPDFLD

Syntax: [statement-identifier] UPDFLD ⊔ control value, data-item-identifier

Type: Format control I/O

Description: The contents of the string data item referenced by data-item-identifier (buffer), is moved to the data-item of the current input field according to the rules as valid for the MOVE-instruction. Depending on control value the new contents of the data-item is redisplayed on the screen.

Control value	Significance
0	Redisplay only if for the current input field the "REWRT" flag is set.
1	Redisplay always.

Condition register: = 0 if OK
= 2 if I/O error.

Condition mask:

0	1	2	3	4	5	6	7
OK	–	ERROR	–	$\overline{\text{OK}}$	–	$\overline{\text{ERROR}}$	UNCON-DITIONAL

Example: UPDFLD 1, SPINPUT

Intermediate object code:

Byte 1	0 0 1 1	0 0 0 0
Byte 2	external reference	
operand-1	control value	
operand-2	data-item-identifier	

Bytes 1 and 2 are filled by the system
Byte 2 is a reference to an external system routine
Operand-1 is the control value
Operand-2 is a reference to a string data item.

Syntax: [statement-identifier] ␣ USE ␣ block-identifier, data-item-identifier

Type: Storage control instruction.

Description: After the USE instruction, the task may access the user or swappable work block specified by block-identifier and data-item-identifier, which must be binary. If the data-item does not specify an existing user or swappable workblock, i.e. it contains a number higher than the highest numbered user or swappable work block, the condition register will be set unequal to zero.
A user or swappable workblock is released from the task as a result of a UNUSE instruction specifying the same user or swappable work block type.

Example: USE UB1, BLOCKNO

Condition register:
=0 if index value within permitted limits.
=1 if swappable workblock under exclusive access.
=2 if index value out of permitted limits.

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	UWB/SWB type							
Byte n	block number							
operand-2	data-item-identifier							

Bytes 1 and 2 are filled by the system.
Operand-1 is the type of user or swappable workblock.
Byte n is the index to the user workblock or swappable workblock.
Operand-2 is a reference to a binary data item.

WAIT *Wait* WAIT

Syntax: [statement-identifier]␣WAIT␣ data-set-identifier

Type: I/O instruction

Description: If the most recently started operation on the data set indicated by data-set-identifier is not yet completed, execution of the next instruction is inhibited. After completion of the operation, execution is continued.

If the operation is already completed before this instruction is executed, the program continues as normal without taking any action on this instruction.

Condition register:

= 0 if I/O successful	(OK)
= 1 if End of file	(EOF)
= 2 if Error	(ERR)
= 3 if Begin or End of device	(BEOD)

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	$\overline{\text{OK}}$	$\overline{\text{EOF}}$	$\overline{\text{ERR}}$	uncon-ditional

Example: WAIT DSKBN

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	data-set-identifier						

Bytes 1 and 2 are filled by the system. Byte 2 contains a reference to an external system routine.
W is the wait bit. This has no significance for WAIT.
Operand-1 is a reference to the relevant data set.
10/100 refers to the first data set.

Syntax: [statement-identifier] ␣ WRITE ␣ [.NW,] data-set-identifier,
data-item-identifier [,size-identifier]

Type: I/O instruction.

Description: The contents of the string data item, indicated by operand-2 is output to the devices specified by data-set-identifier.

The number of characters to be output can be given in the data item specified by size-identifier, which on completion of the output will contain the number of characters transferred.

When the data-set-identifier refers to a data communication data set, this instruction will send data on the line. Time out must be set before this instruction is executed, with the DSC1 instruction and control value X'0B'.

For intertask communication data-set-identifier refers to a data set in which the output file code is defined. When a WRITE (unaddressed) is issued by a task, first a check is performed on the queue for RREAD (addressed). If an addressed read is in the queue for this task, the instruction is completed. When no match occurs the READ queue (addressed) is checked, if not empty the first one is removed from the queue and the instruction is completed. Else the instruction is put into the queue for unaddressed write.

.NW indicates that no wait is desired.

The first two bytes in the string data item must contain a control character. The first byte must always be unequal to zero and the *second byte* must contain the control character (Except for disk). The function of the control character is device dependent. The control character may have the following value:

- General Terminal Printer or Line Printer
  X'2B': print the line without advancing the paper.
  X'30': advance two lines before printing.
  X'31': skip to top of form before printing. (only for line printer)
  Other codes: one line feed is executed before printing.
  Special characters allowed in the user buffer and not restricted to the first word in the buffer:
  X'11' Tabulation character. This character should be followed by two ISO-7 digit characters giving the tabulation position. (Only for GTP).
- Teller Terminal Printer PTS6222, PTS6223
  Voucher/passbook printing
  X'2B': print the line without advancing the paper.
  X'30': advance two line steps before printing.
  X'31'–X'39': advance paper 1–9 line steps before printing.
  Other codes: one line step is executed before printing.

  Journal/tally roll printing
  X'30': advance two step lines before printing. (Two steps = one line feed).

Other codes: one line step is executed before printing.
Special characters allowed in the user buffer:
X'09': The print head is moved to the right most print position of the voucher. This character should be present in the last buffer position.
X'0D': The print head is moved to the right most position of the journal station. This character should be present in the last buffer position.

- Teller Terminal Printer (PTS6371)
  The control character present in the second character of the buffer, as follows:
  /2B – printing is carried out from the last position of the previously printed line on this device. However, if the character pitch has been set, or if positioning has been carried out to the same line, since the previous line was printed, the printing will be from the tabulation position on the present line.
  /30 – the paper is advanced two lines, and the printing carried out from the tabulation position.
  /31 – journal: the paper is advanced three lines and the printing carried out from the tabulation position. This will make the previously written data readable through the window on the journal station.
  – document: printing is started from the tabulation position on line 1.

Any other value in the control code will cause one line feed before printing from the tabulation position.
The requested length must include the two bytes used for the control code, but if it is two, only the action specified by the control code, is carried out.
The maximum line length on the two print stations is limited to the following, based on normal character width:

	Journal	Document
10 characters/inch	33	80
12 characters/inch	40	96
15 characters/inch	50	120

One expanded character equals two normal characters.

- Numeric and Signal Display
  X'30'–X'3F': these codes are sent to the first position (i.e. the left most program display tube on the indicator unit).
  X'40'–X'4F': these codes are sent to the second position.
  X'50'–X'5F': these codes are sent to the third position.
  X'60'–X'6F': these codes are sent to the fourth position.

- Disk, sequential access method
  Data-item-identifier indicates the string to be written to the disk file, indicated by the data-set-identifier. The record will be written, directly after the Last-Record-Number (LRN), which is administrated by Data Management. The record status (FREE or USED) is checked by Data Management before the record is written.

- Video display or plasma display
  X'2B': The text is displayed from current cursor position.
  X'30': Cursor is advanced two lines and positioned at the beginning of the line, before the text is displayed.
  X'31': Erase display and position cursor on home position before the text is displayed.
  Other codes: Advance cursor one line before the text is displayed.
  Special characters allowed in the user buffer and not restricted to the first word in the buffer:

Characters Valid for All Displays
/AE: Displayed as point (/E2)
/11: Tabulation character. This character should be followed by two ISO-7 digits giving the tabulation position.
/07: Bell is sent to the display

Characters Valid for PTS 6344 only
/12: Underline start. Output of characters which follow this character are provided with underline
/13 Underline stop. Output of characters which follow after this character are *not* provided with underline. Underline stop mode will also appear at request end
/14 Fast output. First character following /14 will be transmitted in fast output mode up to requested length. Note that cursor will remain unchanged.
/1C: Data to keyboard.
/1D: Master clear to keyboard.
/1E: Low intensity start. Output of characters which follow after this character, are displayed at low intensity.
/1F: Low intensity stop. Output of characters which follow after this character are displayed at normal intensity. Normal intensity mode will also appear at request end.

See EDWRT for special characters in buffer for PTS6371.

Condition register :
= 0 if I/O successful (OK)
= 1 if End of File (EOF)
= 2 if Error (ERR)
= 3 if Begin or End of device (BEOD)

WRITE *Continued* WRITE

Condition mask:

0	1	2	3	4	5	6	7
OK	EOF	ERR	BEOD	OK	EOF	ERR	Unconditional

Example: WRITE DSTJT, OUTSTR

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	data-set-identifier						
Byte n	list-length							
operand-2	data-item-identifier							
operand-3	size-identifier							

Bytes 1 and 2 are filled by the system.
Byte 2 contains a reference to an external system routine.
W is the wait bit.
W=0 no wait
W=1 wait
Operand-1 is a reference to the relevant data set.
10/100 refers to the first data set.
Byte n is filled by the translator.
Operand-2 contains a reference to a string data item.
Operand-3 contains a reference to a binary data item.

XCOPY *Extended Copy* XCOPY

Syntax : [statement-identifier] ␣ XCOPY ␣ data-item-identifier-1, pointer-identifier-1, size-identifier, data-item-identifier-2, pointer-identifier-2

Type : String instruction

Function : (Operand—4) → Operand—1
(Pointer-identifier-2) (Pointer-identifier-1)

Description: Starting at pointer-identifier-2, the content of operand-4 is copied from left to right to operand-1 beginning at pointer-identifier-1.
The number of bytes to be copied is specified by size-identifier.
This XCOPY is possible between two decimal data items, two binary data items or two string data items. Also copying between two data items of different type is allowed, without conversion.
The first characters of operand-1 and operand-4 are counted as zero when setting the pointer.

Condition register : unchanged

Example : XCOPY FIELD1,P1,LNGTH,FIELD2,P2

Intermediate code format :

Byte 1	0	1	1	0	1	0	1	0
operand-1	data-item-identifier-1							
operand-2	pointer-identifier-1							
operand-3	size-identifier							
operand-4	data-item-identifier-2							
operand-5	pointer-identifier-2							

Byte 1 is the operation code (X'6A')
operands-1,4 are references to string data items
or decimal data items.
operands-2,3 5 are references to binary data items.

XSTAT *Extended status transfer call* XSTAT

Syntax: [statement-itentifier] ␣ XSTAT ␣ data-set-identifier, data-item-identifier

Type: I/O instruction.

Description: A 16 bit device dependent status code from the data set indicated by data-set-identifier, is transferred to the binary data item indicated by operand-2.

Extended status codes are explained in appendix 2.

Condition register: Unchanged

Example: XSTAT DSCASS,STATUS

Intermediate code format:

Byte 1	0	0	1	1	0	0	0	0
Byte 2	external reference							
operand-1	W	data-set-identifier						
operand-2	data-item-identifier							

Bytes 1 and 2 are filled by the system. Byte 2 contains a reference to an external system routine.
W is the wait bit. This has no significance for XSTAT.
Operand-1 is a reference to the relevant data set.
10/100 refers to the first data set.
Operand-2 is a reference to a binary data item.

1.4.9 *Declaration Reference*

This section describes the syntax and use of each declaration. The possible values of the variables in declarations is given in appendix 1. The notation conventions are described in section 1.1.5.

CON *Constant* CON

Syntax:

$$[\text{identifier}] \sqcup \text{CON} \sqcup \left\{\begin{array}{l}\text{actual parameter}\\ \text{value}\\ \text{value expression}\end{array}\right\} \left[\begin{array}{l}\text{,actual-parameter}\\ \text{,value}\\ \text{,value expression}\end{array}\right] \ldots$$

Type: Parameter declaration.

Description: This declaration specifies parameter(s), which have to be passed to a subroutine. The CON declaration follows immediately after a PERF, PERFI or CALL instruction.

Note: It is recommended to use the PLIST declaration instead.

FBN
FBNN
FBNP
FBNZ
FBP
FBZ

Format branch on condition

FBN
FBNN
FBNP
FBNZ
FBP
FBZ

Syntax:

$$[\text{identifier}] \sqcup \left\{\begin{array}{l}\text{FBN}\\ \text{FBNN}\\ \text{FBNP}\\ \text{FBNZ}\\ \text{FBP}\\ \text{FBZ}\end{array}\right\} \sqcup \text{ data-item-identifier, identifier}$$

Type: Format list declaration

Description: These instructions perform forward branching only.
See specific format branch on condition, reference.

FB *Format Branch* FB

Syntax: identifier ␣ FB ␣ identifier

Type: Format-list-declaration.

Description: Editing continues at the format-list-declaration referenced by identifier, further down in the same format list. When the identifier refers to the FMEND declaration, editing is terminated.

FBF
FBT

Format Branch on false/true

FBF
FBT

Syntax: [identifier] ⊔ $\left\{\begin{matrix}\text{FBF}\\\text{FBT}\end{matrix}\right\}$ ⊔ data-item-identifier-1, identifier.

Type: Format-list-declaration

Description: If the value of the boolean data item referred to by data-item-identifier is TRUE, the FBF will result in continuation of the editing at the declaration following the FBF. When the contents of the boolean data-item is FALSE, then editing will be continued at the format list declaration referred by identifier.
If the boolean data item is TRUE, the FBT will result in continuation of editing at the format list declaration referred by identifier, when the boolean data item is FALSE, the next format list declaration will be executed.
Branching to the FMEND results in termination of editing.
The contents of the data item may not be changed while the format list concerned is current.

FBN *Format Branch on Negative* FBN

Syntax: [identifier] ␣ FBN ␣ data-item-identifier, identifier.

Type: Format-list-declaration.

Description: If the contents of the binary or decimal data-item, referred by data-item-identifier, is negative, editing continues at the format-declaration indicated by identifier. (Forward branching only).
If the contents of the binary or decimal data-item is positive or zero editing continues at the next format-list-declaration
Branching to the FMEND results in termination of editing.
The contents of the data item may not be changed while the format list concerned is current.

FBNN	*Format branch on not negative*	FBNN

Syntax:	[identifier]␣FBNN␣data-item identifier, identifier.
Type:	Format-list-declaration.
Description:	If the contents of the binary or decimal data-item, referred by data-item-identifier, is not negative or zero, editing continues at the format-list declaration indicated by identifier. (Forward branching only). If the contents of the binary or decimal data-item is negative editing continues at the next format list declaration. Branching to the FMEND results in terminating of editing. The contents of the data item may not be changed while the format list concerned is current.

FBNP	*Format branch on not positive*	FBNP

Syntax: [identifier] ␣ FBNP ␣ data-item-identifier, identifier

Type: Format-list-declaration.

Description: If the contents of the binary or decimal data-item, referred by data-item-identifier, is not positive or zero, editing continues at the format list declaration indicated by identifier. (Forward branching only).
If the contents of the binary or decimal data-item is positive, editing continues at the next format-list-declaration.
Branching to the FMEND results in terminating of editing.
The contents of the data item may not be changed while the format list concerned is current.

FBNZ *Format branch on not zero* FBNZ

Syntax: [identifier] ⊔ FBNZ ⊔ data-item-identifier, identifier

Type: Format-list-declaration.

Description: If the contents of the binary or decimal data item, referred by data-item-identifier, is not zero editing continues at the format-list-declaration indicated by identifier. (Forward branching only).
If the contents of the binary or decimal data item is zero, editing continues at the next format list declaration. Branching to the FMEND results in terminating of editing.
The contents of the data item may not be changed while the format list concerned is current.

FBP *Format Branch on positive* FBP

Syntax: [identifier] ⊔ FBP ⊔ data-item-identifier, identifier.

Type: Format-list-declaration.

Description: If the contents of the binary or decimal data-item referred by data-item-identifier, is positive editing continues at the format declaration indicated by identifier. (Forward branching only).
If the contents of the binary or decimal data-item is not positive or zero editing continues at the next format-list-declaration.
Branching to the FMEND results in terminating of editing.
The contents of the data item may not be changed while the format list concerned is current.

FBZ *Format branch on zero* FBZ

Syntax: [identifier] ␣ FBZ ␣ data-item-identifier, identifier.

Type: Format-list-declaration.

Description: If the contents of the binary or decimal data-item, referred by data-item-identifier is zero, editing continues at the format-list-declaration indicated by identifier. (Forward branching only). If the contents of the binary or decimal data-item is not zero, editing continues at the next format-list declaration. Branching to the FMEND results in terminating of editing.
The contents of the data itemmay not be changed while the format list concerned is current.

FCOPY *Format copy* FCOPY

Syntax: [identifier] ⊔ FCOPY⊔ $\left\{\begin{array}{l}\text{data-item-identifier}\\ \text{literal}\end{array}\right\}$

Type: Format list declaration.

Description: The location, addressed by operand is copied into the output buffer. The data-item or literal must be of the value type string. Null characters (X'00') are not copied into the output buffer. A literal is not allowed when the FCOPY is used in conjunction with one of the input field declarations FINP or FKI.

Example:

```
FCOPY ⊔ = C'BOOK'
FCOPY ⊔ = X'3031'
FCOPY ⊔ = X'41004243'     Result in buffer X'414243'
FCOPY ⊔ = C'AB'

FCOPY   DATAID          } The contents of the data item
FCOPY   DATARR(IND)     } specified is copied into the
                        } output buffer.
```

FCW *Format control word* FCW

Syntax:

```
[identifier] ␣FCW ␣{data-item-identifier}
                   {literal constant    }
```

Type: Format list declaration

Description: The operand must refer to a data-item or literal of the type binary. If the control word for a particular device must be zero, then, in an EDWRT instruction, FCW causes end of record, while in an EDIT instruction the editing is terminated.

Example:

```
FCW   BIN1
FCW⊔= '0'
FCW⊔= X'8000'
```

FEOR   *Format end of record*   FEOR

Syntax: [Identifier] FEOR

Type: Format list declaration.

Description: If the format list is used by an EDWRT or DISPLAY instruction, the edited buffer is output to the data-set specified in the instruction. No output request is performed, when the buffer is empty.

After an EDWRT instruction the condition register will, if there is more than one FEOR in the format list, contain the logical sum of the conditions met at each output.

If used in an EDIT instruction the FEOR statement causes termination of the editing.

Example:

```
FRMT
FILLR   ' ',2
FTEXT   'ALPHANUMERIC TEXT'
FTAB    16
FTEXT   'STRING'
FEOR
FILLR   ' ',2
FTEXT   'REPLACES'
FTAB    18
FTEXT   'TEXT'
FMEND
```

Result on output data set:

```
ALPHANUMERIC        STRING
REPLACES            ↑  TEXT
↑                   |
|                   |
3                   16
```

FEXIT *Format exit* FEXIT

Syntax:	[identifier] ␣ FEXIT.
Type:	Format list declaration.
Description:	Editing is terminated In an Edit and Write (EDWRT) or DISPLAY instruction, the edited buffer will be written to the output device. No output request is performed. When the buffer is empty FEXIT has the same effect as reaching the FMEND.

FHIGH *Format High intensity* FHIGH

Syntax: [identifier] ␣ FHIGH

Type: Format-list-declaration.

Description: The characters following FHIGH will be displayed with normal intensity if it was before low intensity.
This declaration is only valid for the video display PTS 6344 and when the format list, in which the declaration FHIGH occurs, is invoked by the DISPLAY instruction.
FHIGH results in the control character X'1F' being edited into the buffer.

FILLR *Fill repeat* FILLR

Syntax: [identifier] ␣ FILLR ␣ $\left\{\begin{array}{l}\text{'value expression'}\\ \text{'string-character '}\end{array}\right\}\left\{\begin{array}{l}\text{,value expression}\\ \text{,decimal-integer}\end{array}\right\}$

Type: Format list declaration.

Description: String-character is copied an integer number of times (maximum 63) into the edit buffer.

Example: FILLR ' ',5

FINP Format Input FINP

Syntax: [identifier] ␣ FINP ␣ column [APPL=value]

Type: Format list declaration.

Description: This declaration defines a general input field on the screen. The input field starts at the position defined by column, which is a value expression.
The APPL option defines a control value which can be transferred from this field to the program, with the GETCTL instruction.
Value may range from -32,768 to 32 767.
The FINP declaration, should be immediately followed by a FCOPY or FMEL declaration.

FKI — *Format Keyboard input* — FKI

Syntax: [identifier] ␣ FKI␣ column[,APPL=value] [,SCHK=value] [,MINL=value] [,MAXL=value] [.DUPL=data-item. identifier] [,NUM][,ALPHA] [,REWRT] [,ME] [,NEOI] [,NCLR] [,CTAB] [,VERIF]

Type: Format list declaration

Description: This declaration defines an input field on the screen which is to receive input from a keyboard via the DYKI instruction. The start of the field is defined by column, which is a value expression. The declaration, with its options, must be followed by an FCOPY or FMEL declaration, which contains the input field belonging to data-item.

Options	Significance
	Data item must be of the type string or decimal.
MAXL=value	Value can be obtained with the GETCTL instruction and must be in the range 0 to 127 inclusive. Default value for this option is zero.
ME	This option indicates a compulsory input field. It controls the instructions GETFLD, TUP, TDOWN, TLDOWN, TLEFT, TRIGHT, TBWD, TFWD. (This ME option can be tested by the TSTCTL instruction, if requested).
MINL=value	Value can be obtained with the GETCTL instruction and must be in the range 0 to 63 inclusive. Default value for this option is zero.
ALPHA	This option controls the DYKI instruction and allows alpha numeric characters to be entered for this input field. Default is the NUM option. (This ALPHA option can be tested by the TSTCTL instruction, if requested).
APPL=value	Value can be obtained with the GETCTL instruction and must be in the range -32 768 to 32767. Default value for this option is zero. *
CTAB	Cursor setting is prohibited. This option controls the tabulation instruction TUP, TDOWN, TLDOWN, TLEFT, TRIGHT, TBWD, TFWD and THOME. (This CTAB option can be tested by the TSTCTL instruction, if requested).
DUPL=data-item-identifier	The contents of the data-item referrenced by data-item-identifier can be obtained with the DUPL instruction.
NCLR	This option controls the ERASE instruction and prevents erasing of input fields. (This NCLR option can be tested by the TSTCTL instruction, if requested).

Options	Significance
NEOI	The maximum number of input characters (MAXL) is accepted without a termination key. This option controls the DYKI instruction. (This NEOI option can be tested by the TSTCTL instruction, if requested).
NUM	This option controls the DYKI instruction and allows numeric characters to be entered for this input field. Either NUM or ALPHA may be present as option. (This NUM option can be tested by the TSTCTL instruction, if requested).
REWRT	This option controls the UPDFLD instruction and allows redisplaying of the contents of the data item belonging to this input field. (This REWRT option can be tested by the TSTCTL instruction, if requested).
SCHK=value	Value can be obtained with the GETCTL instruction. Value is a value expression and ranges from 1 to 7 inclusive. Default value for this option is zero. *
VERIF	This option indicates that the input field is subject for verification. It does not control any format control instruction but is obtained by the TSTCTL instruction.

* Note that the values used in the APPL and SCHK options are defined by the user for use in the program outside of the format list. Their values and use are therefore completely application-dependent.

FLINK Format link FLINK

Syntax: [identifier] ␣ FLINK ␣ data-item-identifier / format-list-identifier

Type: Format list declaration

Description: A format list as indicated by format-list-identifier is called as a sub-format, and editing continues according to this subformat. Upon the end of the subformat, editing is resumed at the next format-list-declaration. *
Instead of a format-list-identifier, operand-1 may be a reference to a string data-item. This data-item must contain format-list characters as present in the format-literalpool. (output CREDIT linker). Item size must be great enough to contain these characters. The CALL FMOVE instruction may be used to fill the data-item.

Example:

```
FCOPY = C'NEW BALANCE:'
FLINK   FRM012
FTEXT   'TOTAL:'
```

* Note that the new format list will be edited into the output buffer at the current position, the first two characters (the control characters) being omitted. If the new format list is required to start on a new line, the FLINK declaration should be preceded by FEOR and FCOPY (with control characters as relevant). The same is true for the format-list-declaration following the FLINK; it will be edited into the output buffer at the current position.

FLOW — *Format Low Intensity* — FLOW

Syntax: [identifier] ␣FLOW

Type: Format list declaration.

Description: The characters following FLOW will be displayed with low intensity. This declaration is only valid for the video display PTS 6344 and when the format list, in which the declaration FLOW occurs, is invoked by the DISPLAY instruction. FLOW results in the control character X'1E' to be edited into the buffer.

FMEL Format element FMEL

Syntax: [identifier] ␣ FMEL␣ picture-string, data-item-identifier

Type : Format list declaration.

Description : The decimal data item indicated by operand-2 is edited according to picture-string.
Editing is done from right to left.

Picture-string characters:

Character	*Significance*	*Example*
A	Skip if space (left–adjust to leading digit)	Picture 'AAA999' Data item BFF0456 RESULT 0456
B	Insert a blank space	Picture '99B99' Data item B06521 RESULT 65␣21
E	Enter the character following E into the data item. Any ISO-7 character may be entered, except a single quote or backslash	Picture '99E–99' Data item BO1912 RESULT 19–12
F	Insert character following F, before the next printed digit but after suppression of leading zeros and spaces. Any ISO 7 character may be entered, except a single quote or backslash	Picture 'F*ZZZ9V99' Data item B001053 RESULT *10.53
P	Skip this position	Picture 'P99' Data item B543 RESULT 43
T	Skip if space or leading zero (left adjusted to leading non-zero digit)	Picture 'TTT999' Data item BFFF0456 RESULT 456
V	Insert decimal point (not roomless). A decimal point preceding the leftmost digit will be replaced by asterisks in all leading positions.	Picture '99V99' Data item B00123 RESULT 01.23 Picture '**V9' Data item DF01 RESULT ***1
X	Print alphanumeric (space is printed as space and digits as digits)	Picture 'XXX' Data item BF12 RESULT ␣12

Character	*Significance*	*Example*
Y	Enter alphanumeric if data item is non empty, else enter a space.	
Z	Leading zeroes are replaced by spaces	Picture 'ZZZ99' Data item B00123 RESULT ␣␣123
0	Insert zero	Picture '9909' Data item B123 RESULT 1203
9	Print digit (see X)	Picture '999' Data item BF12 RESULT 012
+	Print a + or – sign [2]	Picture '999+' Data item D123 RESULT 123– Picture 'F+ZZZ' Data item DF01 RESULT ␣␣–1 Picture 'F+**V9' Data item DF11 RESULT *–1.1
–	Print a – sign if the[2] data item is negative. Otherwise print space.	Picture '999–' Data item B123 RESULT 123␣
*	Replace leading zero or space by asterisk	Picture '***99' Data item B00123 RESULT **123
.	Insert roomless point[(1)]	
,	Insert comma[(1)]	Picture ''99,99' Data item B01234 RESULT 12,34

(1) If leading zeroes or spaces are being suppressed, any comma or decimal point (roomless or normal) occurring before the first non-suppressed digit will also be suppressed.

(2) + and – may be declared as floating, the function is further the same.

FMELI *Format element immediate* FMELI

Syntax: [identifier] ␣ FMELI ␣ picture-string, data-item-identifier

Type: Format list declaration.

Description: The decimal data item referenced by data-item-identifier is to be edited according to picture string.

The picture-string is included in the format pool and not in the picture pool. This is the only difference with the FMEL declaration.

For picture details, see FMEL.

FMEND Format end FMEND

Syntax: ⊔FMEND⊔

Type: Format list declaration.

Description: This declaration indicates the end of a format list.

FNL — *Format next line* — FNL

Syntax: [identifier] ␣ FNL

Type: Format list declaration.

Description: When this format list declaration occurs in a format list used by the EDIT instruction, editing will be terminated.
EDWRT and DISPLAY instructions using a format list in which a FNL declaration is present, will result in the following actions:

1. an output request is done for the current contents of the buffer, except when the buffer is empty.
2. A space character is inserted in the first position of the buffer and one line spacing control character space, is inserted in the second position. (One line feed). However, the logicical tabulation position is counted as one.
3. The control characters "low intensity" (X'1E'), is edited in the third position only, when the video display PTS 6344 and the instruction DISPLAY are used.

FNUL *Format No underlining* **FNUL**

Syntax: [identifier] ␣ FNUL

Type: Format-list-declaration.

Description: The characters following FNUL will be displayed with no underlining, if it was before underlining. This declaration is only valid for the video display PTS 6344 and when the format list, in which the declaration FNUL occurs, is invoked by the DISPLAY instruction.
FNUL results in the control character X'13' being edited into the buffer.

FRMT *Format* **FRMT**

Syntax: format-list-identifier ⊔FRMT⊔

Type: Format list declaration.

Description: This declaration indicates the beginning of a format list.

FSL *Format start line* **FSL**

Syntax: [identifier] ⊔ FSL

Type: Format list declaration.

Description: When this format list declaration occurs in a format list used by the EDIT instruction, editing will be terminated.
EDWRT and DISPLAY instructions using a format list in which a FSL declaration is present, will result in the following actions:

1. An output request is made for the current contents of the buffer, except when the buffer is empty.
2. A space character is inserted in the first position of the buffer and one line spacing control character '+', is inserted in the second position. However, the logical tabulation position is counted as <u>one.</u>
3. The control character "low intensity" (X'1E) is edited in the third position only, when the video display PTS 6344 and the instruction DISPLAY are used.

FTAB *Format tabulation* FTAB

Syntax: [identifier] ␣ FTAB ␣ value-expression

Type: Format list declaration

Description: The pointer for the buffer is set to the position specified by the value-expression. This column may be to the right or to the left of the current position. The positions in the buffer between the current pointer and the new pointer are filled with space characters. Editing proceeds from the new pointer.
The first position in the buffer is counted as one when setting the pointer.
When calculating the tabulation position, the format-list-declarations FSL, FNL, FHIGH, FLOW, FUL, FNUL, FINP, and FKI each occupy one character in the buffer.

Example:

```
SIXT    FTAB    16
ONE     EQU     1
FIVE    FTAB    ONE+4
```

FTABLE *Format table generation* **FTABLE**

Syntax: [format-table-identifier] ␣FTABLE␣ format-list-identifier, [,format-list-identifier] . . .

Type: Format table declaration

Description: A one dimensional array of format-list-identifiers is declared, which may be referenced in instructions expecting a format-list-reference. Format lists referenced in the format table declaration may not be passed as a parameter in PERF and PERFI instructions only the name of the complete format table can be passed.
In the heading of a subroutine the formal parameter name for the format table must be followed by two brackets.
Following the PROC directive, the formal parameter name must be mentioned in a PFRMT directive, even if no two byte addressing is selected. The rules which apply to elements in a one dimensional array are also valid for format tables

Example:

```
FTB1      FTABLE      FORM1,FORM2,FORM3


EDWRT     DSVDU,FTB1 (INDEX)


PROC      $FFTAB ( )
PFRMT     $FFTAB
```

FTEXT *Format immediate text* FTEXT

Syntex: [identifier] ⌴ FTEXT $\left\{\begin{array}{l}\text{'string-character ... '}\\ \text{X' hexadecimal-digit ...'}\end{array}\right\}$

Type: Formal list declaration

Description: String-character(s) or hexadecimal-digits are copied into the output buffer. Copying is done on character base.
Not allowed characters are the quote or backslash.

Example:

```
FTEXT     'NEW BALANCE'
FTEXT     X'4E554553'
FTEXT     X'30
```

FUL *Format underlining* FUL

Syntax: [identifier] ␣ FUL

Type: Format list declaration.

Description: The characters following FUL will be displayed with underlining, if it was before no underlining. This declaration is only valid for the video display PTS 6344 and when the format-list, in which the declaration FUL occurs, is invoked by the DISPLAY instruction.
FUL results in the control character X'12' being edited into the buffer.

KTAB *Key table* KTAB

Syntax: Key-table-identifier␣KTAB␣ key-value [,key-value] . . . .

Type: Key table declaration.

Description: A key value in the key value list is used as a terminating character (end of record key) in the keyboard input instructions.
If a value expression is used in key value list, only value type X may be used.

Example:

```
KTAB1    KTAB   KCORR,KMUL,KDIV
KTAB2    KTAB   X'0D',X '30' + 2
```

PLIST *Parameter list* PLIST

Syntax: ␣ PLIST ␣ actual-parameter {,actual-parameter} . . .

Type: Parameter declaration.

Description: This declaration specifies parameter(s), which have to be passed to a subroutine. The PLIST declaration follows immediately a PERFI instruction.
A literal constant of the type 'X' is not allowed as parameter.