

## OPEN INSTRUCTIONS

There are two forms of the OPEN instruction, OPEN and OPENS. Each is discussed separately.

### ★ ★ OPEN

#### Function

This instruction, which may be used with any type file, gives the programmer control over the time when a file is opened. Any file having an OPEN instruction associated with it will not be opened when the program is loaded into memory.

Using the OPEN instruction also provides the programmer with the option of transferring control to a user routine after the first section of a magnetic media file or punched tape file is opened. If this option is desired, the name of the routine must be entered on the file specification sheets for this file. The routine could be used to output or input labels.

#### Example

*	REFERENCE	*	OPERATION	*	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24	25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C			O P E N		E X C E P T F I L E

In the example above, the instruction causes the software to perform the necessary steps involved in opening a file called EXCEPTFILE. Since there is an OPEN instruction associated with this file, it is not opened when the program is first loaded into memory.

## Conventions

- General

OPEN instructions should be used with caution. If the source file does not exist, or if there is no room for the destination file, the run may have to be aborted at the time the desired file is to be opened.

Entering a user-routine name on the file specification sheet causes a branch to be taken after each section of the file, except the first, is opened. Using an OPEN instruction causes the branch to be taken to the named routine after the first section is opened.

\* \* \* \*

OPENS

Function

The OPENS instruction allows the programmer to open a specific section of a magnetic media file. The file must be a source, destination, or a source-destination. A file containing a section to be opened with the OPENS instruction is not opened when the program is loaded into memory. The user must open the file initially with the OPEN or OPENS instruction at the first section of the file, or with the OPENS instruction at any subsequent section of the file.

When processing a file sequentially, the programmer may skip the remainder of a section or entire section by using the OPENS instruction in conjunction with the CLOSES instruction. After closing a section, the programmer uses the OPENS instruction to open the next section desired following the skipped section(s). He moves the relative address at which the file is to be opened (section number and relative sector number in the case of disc files, or section number and relative card and track numbers in the case of CRAM files) to a user-defined area prior to executing the OPENS. This area, referenced by an operand of the OPENS instruction, identifies the section to be opened.

Using the OPENS instruction also provides the programmer with the option of having control transferred to a user routine after the specified section is opened. If this option is desired, the name of the routine must be entered on the file specification sheets for this file.

Example

- OPENS for Disc Files

REFERENCE	OPERATION	OPERANDS																																										
7		8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
C	OPENS	MASTERFILE, ADRESAREA1																																										

REFERENCE	MODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE																																						
7		8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
D	A	ADRESAREA1	A	4	B																																							
D	F	SECTION	F	0	1	B																																						
D	F	ZERO	F	1	1	B0																																						
D	F	RELSECT	F	2	2	B																																						

SECTION NUMBER	MUST BE ZERO	RELATIVE SECTOR NUMBER	

In the preceding example, the OPENS instruction causes the software to perform the necessary steps involved in opening a file called MASTERFILE at the section and relative sector specified in an area called ADRESAREA1.

The programmer defines ADRESAREA1 as a 4-character binary area. The second character position must always be zero.

- OPENS for Magnetic Tape Files

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
C											O P E N S		M A S T E R F I L E , A D R E S A R E A 2																														

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
D											A	D R E S A R E A 2	A	1	B																												

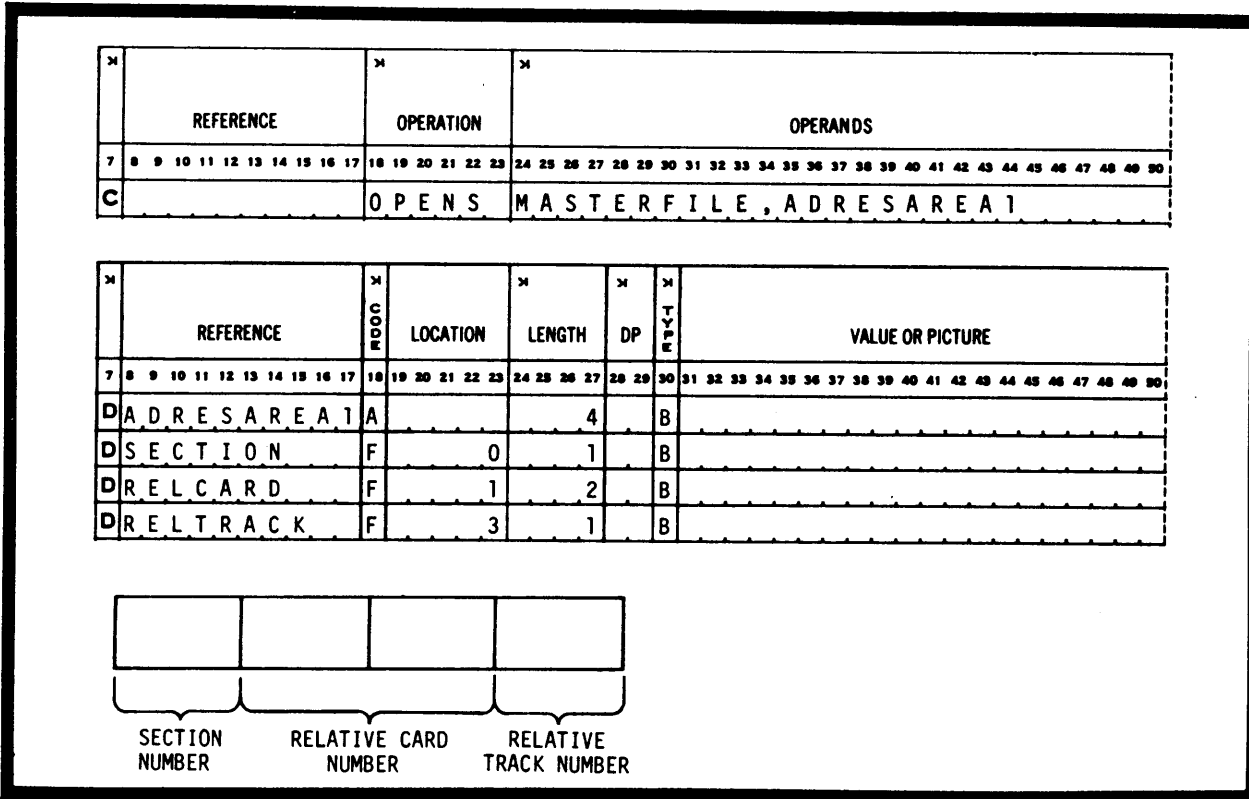
  

SECTION  
NUMBER

In the above example, the OPENS instruction causes the software to perform the necessary steps involved in opening a file called MASTERFILE at the section specified in ADRESAREA2.

The programmer defines ADRESAREA2 as a 1-character binary area.

• OPENS for CRAM Files



In the example above, the OPENS instruction causes the software to perform the necessary steps involved in opening a file called MASTERFILE at the section, relative card, and relative track specified in an area called ADRESAREAL.

The programmer defines ADRESAREAL as a 4-character binary area.

Conventions

• Disc

The first sector in each section of a file is relative sector zero.

• CRAM

The first card in each section of the file is relative card zero.

The first track on each card is relative track zero.

## OPENT

### Function

The OPENT instruction specifies that certain sections of a multi-section random file on disc or CRAM will not be opened. By opening a random file with an OPENT instruction, the programmer may skip selected sections of the file and open all other sections.

This instruction is used when the programmer knows before opening a random file that he cannot access all sections of the file. Use of the OPENT instruction, in conjunction with the CLOSET instruction, to process only specified sections of a random file is particularly applicable when processing online with peripherals out of service.

To identify the file sections to be skipped, the programmer reserves a work area for a variable-length list (TABLE in the example below) of these section numbers. Any section of the file not included in the list will be opened.

All entries in the list must be 1-character binary numbers and must be listed in ascending order. A minimum of one entry is required. The last entry in the list must always be a 1-character binary zero. If the only entry is a 1-character binary zero end sentinel, all sections of the file will be opened.

### Example

- OPENT for Disc Files

×	REFERENCE	OPERATION	OPERANDS																											
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30	31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50																									
C		OPENT	TRANSFILE, TABLE																											

×	REFERENCE	CODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE																							
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24 25 26 27	28 29	30	31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50																						
D	T A B L E	A		4																										
D		F	0	1		B	1																							
D		F	1	1		B	2																							
D		F	2	1		B	7																							
D	S E N T I N E L	F	3	1		B	0																							

In the above example, the OPENT instruction causes the software to skip (not to open) sections 1, 2, and 7 of a random disc file called TRANSFILE. The software will open the sections between the skipped sections (3, 4, 5,

and 6 in this example). When the binary zero end sentinel is reached (after section 7 in this example), all remaining sections of the file will be opened.

- OPENT for CRAM Files

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50					
C											O	P	E	N	T		D	A	T	A	F	I	L	E	,	T	A	B	L	E																		

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50							
D	T	A	B	L	E						A							4																																
D											F		0					1						B		3																								
D											F		1					1						B		4																								
D											F		2					1						B		5																								
D	S	E	N	T	I	N	E	L			F		3					1						B		0																								

In the above example, the OPENT instruction causes the software to skip (not to open) sections 3, 4, and 5 of a CRAM file called DATAFILE. The software will open the sections (1 and 2 in this example) preceding the skipped sections. When the binary zero end sentinel is reached (after section 5 in this example), all remaining sections of the file will be opened.

Conventions

Before processing the file, the programmer selects the sections to be skipped. If he subsequently wishes to open these skipped sections, the programmer must close (with a CLOSET instruction) all currently opened sections of the file. The programmer may then remove the desired section number entries from the list, thus permitting the software to open the specified sections when OPENT is used.

Entries in the work area (the section numbers of the sections to be skipped) must be 1-character binary numbers and must be listed in ascending order. The list must always contain at least one entry, and a 1-character binary zero end sentinel must follow the last entry. If the end sentinel is the only entry in the list, all sections of the file will be opened.

## ROPEN INSTRUCTIONS

### INTRODUCTION

During a program run, a file may be used as a source file, destination file, source-destination file, or piggyback file, depending upon the usage specified on the file specification worksheets.

After the file is opened and processed, but before it is closed, one of the variations of the ROPEN instruction may be used to close the file and reopen it as another type (source, destination, etc.). When a file is to be reopened this way, its secondary use must also be specified on the file specification worksheets.

### NOTE

Although a file may be reopened by using both a CLOSE and OPEN instruction, using an ROPEN instruction is a faster method.

There are four ROPEN instructions which correspond to the four types of file usage:

ROPENS reopens a file as a source file  
ROPEND reopens a file as a destination file  
ROPENR reopens a file as a source-destination file  
ROPENP reopens a file as a piggyback file.

These four instructions are used only with disc and magnetic tape files.



## ROPENS

### Function

The ROPENS instruction closes a file and reopens it as a source file. If this instruction is used, an S (for source) must be specified on the file specification worksheets for the secondary file usage.

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50				
C											R	O	P	E	N	S	E	X	C	E	P	T	F	I	L	E																					

In the example above, the ROPENS instruction causes the software to close the file (EXCEPTFILE) and reopen it as a source file. Any partially filled buffer is output before the file is closed.

### Conventions

- An ROPENS instruction must reference a file that is currently open.
- Another variation of the ROPEN instruction may be used after the ROPENS instruction; however, the instruction must reopen the file as its original file type (subsequent ROPEN instructions must alternately reopen the file first as a source file, and then as the original file type).

ROPEND

Function

The ROPEND instruction closes a file and reopens it as a destination file. When this instruction is executed, the old file is destroyed, allowing a new file to be written in the same area. If this instruction is used, a D (for destination) must be specified on the file specification worksheets as the secondary file usage.

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50								
REFERENCE											OPERATION						OPERANDS																																		
C											R O P E N D						E X C E P T F I L E																																		

In the example above, the ROPEND instruction causes the software to close the file (EXCEPTFILE) and reopen it as a destination file. Any partially filled buffer is output before the file is closed.

Conventions

- An ROPEND instruction must reference a file that is currently open.
- Another variation of the ROPEN instruction may be used after the ROPEND instruction; however, the instruction must reopen the file as its original file type (subsequent ROPEN instructions must alternately reopen the file first as a destination file, and then as the original file type).

## ROPENR

### Function

The ROPENR instruction closes a file and reopens it as a source-destination file. If this instruction is used, an R (for source-destination) must be specified on the file specification worksheets as the secondary file usage.

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		R O P E N R	E X C E P T F I L E

In the example above, the ROPENR instruction causes the software to close the file (EXCEPTFILE) and reopen it as a source-destination file. Any partially filled buffer is output before the file is closed.

### Conventions

- An ROPENR instruction must reference a file that is currently open.
- Another variation of the ROPEN instruction may be used after the ROPENR instruction; however, the instruction must reopen the file as its original file type (subsequent ROPEN instructions must alternately reopen the file first as a source-destination file, and then as the original file type).

ROPENP

Function

The ROPENP instruction closes a file and reopens it as a piggyback file. When this instruction is executed, the file is reopened at the end of the existing file. If this instruction is used, a P (for piggyback) must be specified on the file specification worksheets for the secondary file usage.

X	REFERENCE	X	OPERATION	X	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24	25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C			R O P E N P		E X C E P T F I L E

In the example above, the ROPENP instruction causes the software to close the file (EXCEPTFILE) and reopen it as a piggyback file. Any partially filled buffer is output before the file is closed.

Conventions

- An ROPENP instruction must reference a file that is currently open.
- Another variation of the ROPEN instruction may be used after the ROPENP instruction; however, the instruction must reopen the file as its original file type (subsequent ROPEN instructions must alternately reopen the file first as a piggyback file, and then as the original file type).

## CLOSE INSTRUCTIONS

There are three forms of the CLOSE instruction, CLOSE, CLOSEO, and CLOSES. Each is discussed separately.

### \* CLOSE

#### Function

This instruction may be used with any type file. The CLOSE instruction provides the programmer with control over the time when a file is closed and with the option of transferring control to a user routine before the last section of a magnetic media file or punched paper tape file is closed. If this option is desired, the name of the routine must be entered on the file specification sheets for this file. The routine could be used to output or input labels.

#### Example

X	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		C L O S E	E X C E P T F I L E

In the above example, the instruction causes the software to perform the necessary steps involved in closing a file called EXCEPTFILE. If the CLOSE instruction associated with this file is not executed before the FINISH instruction is encountered, the file is closed automatically.

#### Conventions

##### • General

Entering a user-routine name on the file specification sheets causes a branch to be taken before each section, except the last, of the file is closed. Using a CLOSE instruction causes the branch to be taken to the named routine before the last section is closed.

\* \* \* \*

## CLOSEO

### Function

This instruction is used with magnetic media scratch files; therefore, the file specifications worksheet must specify SCR in the type of dating period. The CLOSEO instruction is used to obsolete scratch files during or at the end of a run to allow immediate reuse of the space occupied by the scratch file.

Using the CLOSEO instruction also provides the programmer with control over the time the file is closed and with the option of transferring control to a user routine before each section of the file is closed. If this option is desired, the name of the routine must be entered on the file specification sheets for this file.

### Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		C L O S E O	S C R T C H F I L E

In the above example, the instruction causes the software to perform the necessary steps involved in closing and obsoleting a file called SCRTCHFILE. If the CLOSEO instruction associated with this file is not executed before the FINISH instruction is encountered, the file is closed automatically but is not obsoleted.

### Conventions

- General

Entering a user-routine name on the file specification sheets causes a branch to be taken before each section, except the last, if the file is closed. Using a CLOSE instruction causes the branch to be taken to the named routine before the last section is closed.

- Disc and CRAM

When CLOSEO is used to obsolete a file, the space occupied by the disc or CRAM file becomes free for use by any other file. Only the current section of the file is obsoleted if the file contained more than one section.

- Magnetic Tape

CLOSEO may be used with magnetic tape for compatibility reasons. However, scratch files on magnetic tape are already obsolete at the time they are created.

CLOSES

Function

This instruction is used to close the current section of a magnetic media file. Since the CLOSES instruction also sets a flag indicating to the software that further sections exist, this instruction must not be used to close the last section of a file.

The use of the CLOSES instruction provides the programmer with the option of transferring control to a user routine before the current section is closed. If this option is desired, the name of the routine must be entered on the file specification sheets for this file.

Example

7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		C L O S E S	E X C E P T F I L E

The instruction coded above causes the software to perform the necessary steps involved in closing the current section of the file called EXCEPTFILE.

Conventions

• General

The CLOSES instruction must not be used to close the last section of a file.

Entering a user-routine name on the file specification sheets causes a branch to be taken before the current section of the file is closed.

CLOSET

Function

The CLOSET instruction specifies that certain sections of a multi-section random file on disc or CRAM need not, or cannot, be closed. This instruction references a list (TABLE in the example below) of the section numbers which are not to be closed during the current processing run. The programmer uses the CLOSET instruction to skip the listed sections and close all other sections.

Sections included in the list may have been selected before processing the file and consequently skipped by using the OPENT instruction. In this case, CLOSET references the same list referenced by OPENT and causes the software to close all currently open files.

If peripheral units become inoperable during a processing run, sections of a file that are mounted on those units cannot be closed. When this situation occurs, the programmer may use CLOSET to reference a list of the sections which cannot be closed. The software will skip the listed sections when closing the other sections of the file.

Example

REFERENCE	OPERATION	OPERANDS																											
C	CLOSET	DATAFILE, TABLE																											

REFERENCE	BOOK	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
TABLE	A		4			
	F	0	1		B	3
	F	1	1		B	4
	F	2	1		B	5
SENTINEL	F	3	1		B	0

In the above example, the CLOSET instruction causes the software to skip (not to close) sections 3, 4, and 5 of a random file called DATAFILE. The software will close the sections (1 and 2 in this example) preceding the skipped sections. When the binary zero end sentinel is reached (after section 5 in this example), all remaining sections of the file will be closed.



Conventions

Entries in the work area (the section numbers of the sections to be skipped) must be 1-character binary numbers and must be listed in ascending order. The list must always contain at least one entry, and a 1-character binary zero end sentinel must follow the last entry. If the end sentinel is the only entry in the list, all sections of the file will be closed.

## GET INSTRUCTIONS

There are many variations of the GET instruction: GET, RGET, SGET, SGETL, SGETC, and LGET. Each is discussed separately.

### GET

This instruction consecutively accesses blocks of records from source or source-destination files and places these blocks in a buffer area. Only one record within that particular block is available to the program at a time. Fixed- or variable-length records are handled automatically.

The first execution of the GET instruction accesses the first block of records in a named file and makes the first record in that block available to the program. When a block is composed of more than one record, the next execution of the GET instruction makes the second record in the first block available to the program. This pattern continues until all the records in the block are depleted; then, the next time the GET instruction is executed the next block in sequence is accessed.

After all records have been presented to the program and the GET instruction is again executed, a link is automatically taken to the end-of-file exit routine named on the file specification sheets for this file.

If the programmer requests translation on the file specification sheets, the GET instruction translates input data into processor internal code. If an illegal code is encountered during translation, a branch occurs to the user routine specified on the file specification sheets.

Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		GET	TRANSFILE
C		GET	TRANSFILE, WORKAREA

In the first example, the instruction accesses the first block of information from a file called TRANSFILE and reads it into a buffer area. The first record in this buffer area is available to the program.

In the second example, the instruction functions the same as in the first except that the record is also moved into the specified work area. In essence, this eliminates the need for coding a MOVE instruction. Unlike the MOVE instruction, however, this technique does not adjust (truncate or expand) the record being moved into the specified work area to fit the designated size of the work area. Therefore, the programmer must take care that the record being moved does not exceed the size of the work area and destroy data in adjacent memory locations.

\* \* Conventions

• General

Variable-length indicators (VLI) are not translated when translation of records is specified. They are converted from decimal to binary on input if the presence of decimal indicators was specified on the file specification sheets.

• Disc and CRAM

The work area option is not permitted if random accesses are made or the WRITSP instruction is used with this file.

The GET instruction follows the chaining in a chained file.

The GET instruction does not stop at the end of a section, at a null block, or at the end of a bucket. Processing continues with the next bucket, block, or section. If the programmer requested a branch to a user routine after each section is opened, that routine is completed before any data from the new section is processed.

\* \* \* \*

LGET

Function

This instruction is used to sequentially input 80-character, user-label blocks from a magnetic tape file. The LGET instruction may only be used in a user routine after OPEN or in a user routine before CLOSE. These user routines are entered only if an OPEN or CLOSE instruction is used with the file, and the routine name is specified on the file specification sheets.

After the file is opened or just before it is closed, the software transfers control to the user routine if desired. The LGET instruction is used in this routine to read the label blocks into the input buffer area. After the program reads the labels, a RELINK instruction must be executed to return control to the software. If a tape mark is read, control is automatically returned to the software. (Tape marks always signal the end of a group of labels.) For complete details concerning label formats, see FILES, tab 2, "Magnetic Tape Files."

If the programmer requests translation on the file specification sheets, the LGET instruction translates the entire 80 characters being input.

Example

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50						
REFERENCE											OPERATION						OPERANDS																																
C											L	G	E	T	O M A S T R F I L E																																		

In the above example, the instruction inputs an 80-character label block from a file called OMASTRFILE. The label block is placed in the input buffer area normally associated with this file. To access the fields in the label, redefine the record on the data layout sheets for the file by specifying SAME in the location positions. For a complete explanation of redefining a record see the discussion of location in the NEAT/3 REFERENCE MANUAL, INTRODUCTION AND DATA, tab 3, "Data Layout Sheets."

Conventions

● General

The work area option is not available with this instruction.

The input buffer area for the file must be at least 80 characters long.

## RGET

### Function

This instruction accesses a block of records from a specific disc sector, or a specific CRAM card and track, of a randomly-processed file and places the block in the input buffer. The first record in the block is available to the program. Fixed- or variable-length records are handled automatically.

Before executing the RGET instruction, the relative address of the desired block must be moved into an area previously defined by the programmer. This area contains the section number and the relative sector number of the desired block in a disc file; or the section number, the relative card number, and the relative track number of the desired block in a CRAM file. The name of this area appears as the second operand in the RGET instruction.

The first (or only) section of a file is considered to be section one. The relative number of the first disc sector or the first CRAM card of any section is zero. The relative number of the first CRAM track of any CRAM card is zero. In the case of a disc file, the software adds the relative sector number to the actual section starting address to obtain the actual sector address of the desired block. In the case of a CRAM file, the software adds the relative card number to the starting address of the section to access the desired card, and adds the starting track address of this card to the relative track number to access the desired block. The programmer, however, need not concern himself with actual addresses.

If a null block is read into the buffer area, and if the optional third operand of RGET is used, a branch is automatically taken to the routine referenced in the third operand. While the null block is in memory, a record may be inserted into it if desired. This procedure is useful when randomly building a chained file with buckets.

After using the RGET instruction to access a block and to present the first record to the program, the programmer generally uses a variation of the SGET instruction to present the remaining records in the block. The action taken when the end-of-block is reached depends upon the instruction used. (See SGET, SGETL, and SGETC.)

The RGET instruction does not translate the information being input.

### Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
C		RGET	MASTERFILE, ADDRESS AREA
C		RGET	MASTERFILE, ADDRESS AREA, NULL IN

The 2-operand RGET instruction accesses from the file called MASTERFILE the block located at the relative address which appears in the area called ADDRESAREA. The block is placed in the buffer and the first record in the block is presented to the program. Whether or not the block accessed is a null block, control is passed to the next instruction in sequence.

The 3-operand RGET instruction functions just as the 2-operand instruction except when a null block is read. If the block accessed is a null block, a branch is taken to NULLIN, the routine named by the third operand. If the block is not a null block, control is passed to the next instruction in sequence.

The programmer defines ADDRESAREA on a data layout sheet as a 4-character binary area or an 8-character decimal area. The name used here could also be that of an item in a table, in which case the current item in the table is used.

Area definitions are shown below for CRAM files and disc files. Note that the second field of the ADDRESAREA for a disc file must be zero.

● 4-Character Binary Area

For a Disc File																																For a CRAM File																															
REFERENCE							REC CODE	LOCATION							LENGTH							DP		TYPE																																							
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32												
D	A	D	D	R	E	S	A	R	E	A	A									4			B																																								
D	S	E	C	T	I	O	N	N	O		F									1			B																																								
D	Z	E	R	O	F	I	E	L	D		F									1			B		0																																						
D	S	E	C	T	O	R	N	O			F									2			B																																								
D	A	D	D	R	E	S	A	R	E	A	A									4			B																																								
D	S	E	C	T	I	O	N	N	O		F									1			B																																								
D	C	A	R	D	N	O					F									2			B																																								
D	T	R	A	C	K	N	O				F									1			B																																								

● 8-Character Decimal Area

For a Disc File																																For a CRAM File																															
REFERENCE							REC CODE	LOCATION							LENGTH							DP		TYPE																																							
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32												
D	A	D	D	R	E	S	A	R	E	A	A									8			U																																								
D	S	E	C	T	I	O	N	N	O		F									2			U																																								
D	Z	E	R	O	F	I	E	L	D		F									2			U		0																																						
D	S	E	C	T	O	R	N	O			F									4			U																																								
D	A	D	D	R	E	S	A	R	E	A	A									8			U																																								
D	S	E	C	T	I	O	N	N	O		F									2			U																																								
D	C	A	R	D	N	O					F									3			U																																								
D	T	R	A	C	K	N	O				F									3			U																																								

## Conventions

- Disc or CRAM

The question on file specification sheet 2, "Are any random accesses made to this file during this program?" must be answered YES when RGET is used with a file.

If the relative address specified does not fall within the limits of the file, a branch is taken to the error exit for random processing instructions (file specification sheet 1).

- CRAM

If a CRAM card is not to be updated after it is read, the card should be released from the capstan immediately following the execution of the RGET instruction. The user indicates that the current CRAM card is to be released by setting the FileReference.\$CARDFLAG in the file table to binary 2. If all CRAM cards are to be released after they are read, the user must set the flag to binary 4. If no cards are to be released after reading (the cards are to be updated), the user need not set the FileReference.\$CARDFLAG.

SGETFunction

This instruction consecutively accesses blocks of records from any randomly-processed file on disc or CRAM and places these blocks in an input buffer area. Only one record in the block is available to the program at one time. Fixed- or variable-length records are handled automatically.

Although the SGET instruction functions with any randomly-processed file used as input from disc or CRAM, it is generally used with chained source-destination files. The SGET instruction is specifically designed to work with bucket logic. Through the use of a branch operand and the Null flag (NULLFLAG), this instruction informs the program when a null block, end-of-bucket, or end-of-section occurs. SGET gives the programmer the ability to conduct a limited search for a record and to branch if that record is not found in the expected bucket or section. The entire file need not be examined.

After a branch is taken, the program must examine the Null flag to determine the cause of the branch. This is done by a COMPARE instruction which uses the file reference name as a qualifier to \$NULLFLAG for one of the operands (COMP MASTERFILE.\$NULLFLAG, 2ND OPERAND). If desired, the second operand may be a literal number indicating one of the three possible binary values of the Null flag (COMP MASTERFILE.\$NULLFLAG, '1').

VALUE	NULLFLAG MEANING
Binary 1	A Null block was read into the input buffer area.
Binary 2	Nothing was read. The end-of-section (E-O-S) was encountered.
Binary 4	Nothing was read. The end-of-bucket (E-O-B) was encountered.

If more than one of these conditions exists, the Null flag is set according to the following priorities:

- The null block condition, indicated by binary 1, takes precedence over any other condition.
- The end-of-section condition, indicated by binary 2, takes precedence over the end-of-bucket condition, indicated by binary 4.

If a branch is taken due to a null block, another SGET presents the next block to the program. If a branch is taken due to E-O-B, another SGET presents the first block from the next bucket. If a branch is taken due to E-O-S, another SGET execution is not permitted and an error results if the instruction is used. The fact that the record was not found indicates that an exception of some type exists. After noting this condition (perhaps in an exception file), RGET is used to find the next desired record.

The following illustrations include a list of functions performed by successive executions of SGET when a null block, E-O-B, and E-O-S are encountered.



RECORD BLOCK	CONDITION	PROG	FUNCTION
73 74 75	E-O-B A block containing data is the last block in the bucket	RGET SGET SGET SGET SGET	Reads block and presents record 73. Presents record 74. Presents record 75. Detects E-O-B, sets NULLFLAG to 4 and branches. Reads next block and presents first record.
75 NULL BLOCK	E-O-B A null block is the last block in the bucket.	SGET SGET SGET SGET	Presents record 75. Accesses next block, detects null block, sets NULLFLAG to 1 and branches. Detects E-O-B, sets NULLFLAG to 4 and branches. Reads first block from next bucket, presents first record in block.
75 NULL BLOCK	E-O-B & E-O-S A null block is the last block in the bucket and section.	SGET SGET SGET SGET	Presents record 75. Detects null block, sets NULLFLAG to 1 and branches. Detects E-O-S, sets NULLFLAG to 2 and branches. Error - Do not use SGET after E-O-S.

The SGET instruction does not translate the input information.

Example

REFERENCE	OPERATION	OPERANDS
7 8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C	SGET	MASTERFILE, BRROUTNAME

In the above example, the instruction accesses the next record in a file called MASTERFILE. If no more records exist in the current block and this is the last block in a bucket (end-of-bucket) or the last block in a section (end-of-section), a branch is made to the routine called BRROUTNAME. If no more records exist in the current block and neither E-O-B nor E-O-S occurs, the next block in the file is read. If the new block read is a null block, a branch is made to BRROUTNAME. If the new block is not a null block, the first record in the new block is presented to the program.

Conventions

- Disc and CRAM

SGET cannot be used after an end-of-section branch.

SGET is generally used after RGET to step through a block. The SGET instruction is used only when processing randomly; the GET instruction is used for sequential processing.

The VLI of a variable-length record accessed by the RGET instruction must be in binary format.

## SGETC

The following discussion assumes the reader is familiar with the RFILE instruction, which is also discussed under this tab.

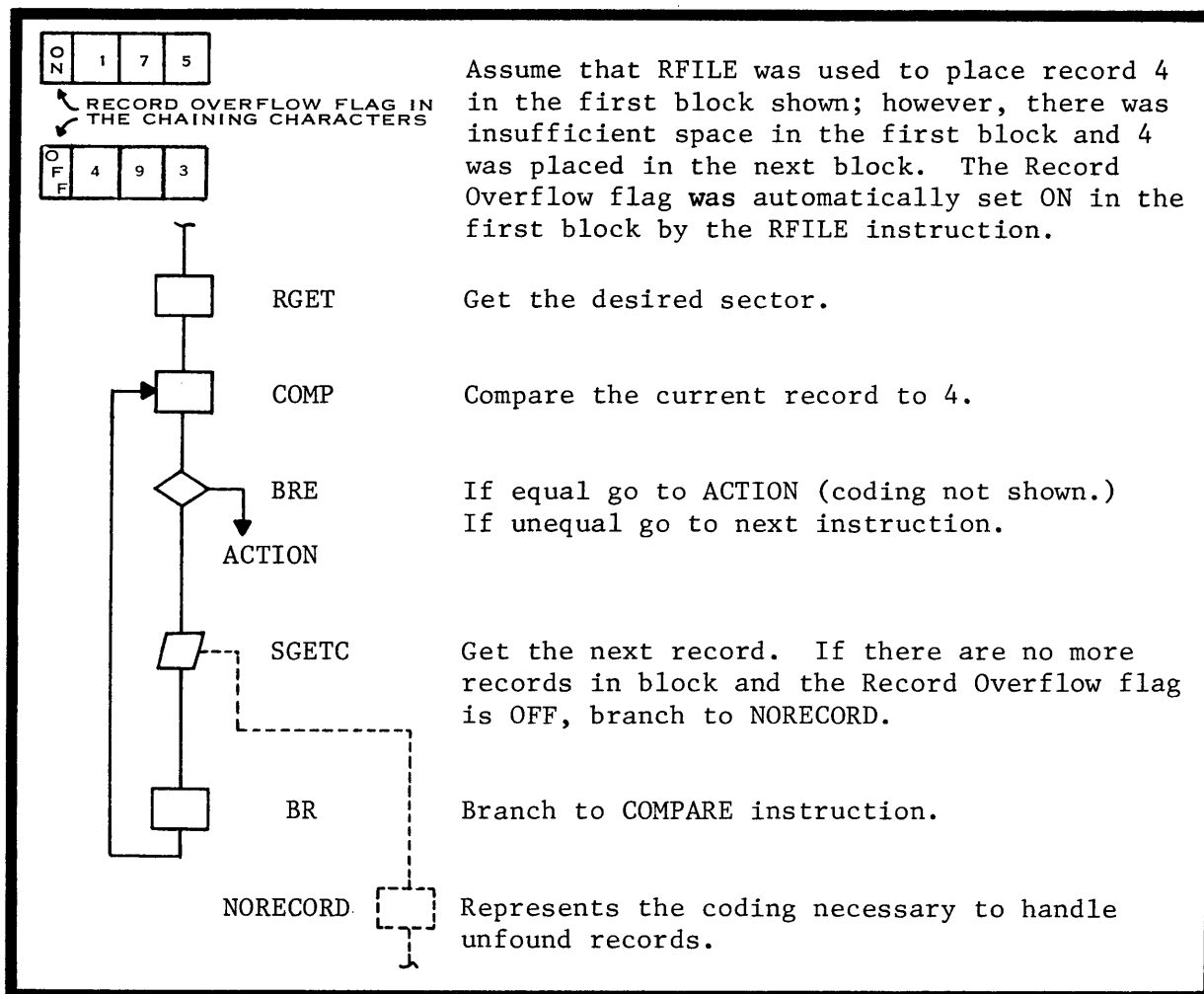
### Function

The SGETC instruction is used only with randomly-processed chained files to access the next record in sequence within a bucket. The SGETC instruction, which differs slightly from the SGET instruction, contains an additional branch operand.

The SGETC instruction is normally used to search for a record placed in the file by the RFILE instruction. In each block accessed by the RFILE instruction which contains insufficient space for the record to be added, RFILE sets the Record Overflow flag ON to indicate that it has attempted to place the new record in a subsequent block. The SGETC instruction is used to access records until end-of-block, when SGETC checks the Record Overflow flag to determine whether or not RFILE attempted to place the new record in a subsequent block. Searching for the desired record with successive executions of SGETC continues until the record is found, or until, at end-of-block, the Record Overflow flag is found OFF in the current block. If the latter condition occurs, a branch is taken to the third operand of the SGETC instruction.

A branch is taken to the routine referenced in the second operand when the SGETC instruction encounters a null block, end-of-bucket, or end-of-section. The Null flag (NULLFLAG) settings are identical to the Null flag settings for the SGET instruction.

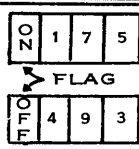
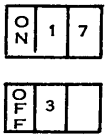

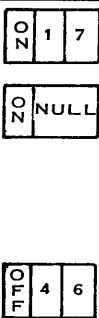
The illustrations below show how the SGETC instruction is used to locate a specific record added to the file by the RFILE instruction.



In the flowchart, the RGET instruction accesses the first block and presents record 1 to the program. Since record 1 does not equal record 4, the SGETC instruction presents the next record (record 7). SGETC then presents record 5. The next time SGETC is executed, the Record Overflow flag in the current block is checked. Since the flag is ON, SGETC does not branch to the routine named in the third operand, but instead presents the first record in the next block. Record 4, the desired record, has been located.

If, after presenting records 1, 7, and 5 to the program, the Record Overflow flag is found OFF, the search for the desired record is terminated by branching to the routine named by the third operand. The OFF setting of the Record Overflow flag indicates that RFILE did not add the desired record to any subsequent block.

The branch taken due to the OFF setting of the Record Overflow flag (when all records in the current block have been accessed) takes precedence over the branch taken when end-of-section or end-of-bucket occurs. The following illustrations show the result of successive executions of SGETC under various conditions.

RECORD BLOCK	CONDITION	PROG	RESULT
	The program is searching for record 4.	RGET SGETC SGETC SGETC	Reads block and presents record 1. Presents record 7. Presents record 5. Since flag is ON in this block, reads the next block, presents record 4.
	E-O-B & E-O-S The program is searching for record 4 which does not exist in the file. Record 3, which caused the flag to be ON when it was added by RFILE, is the last in the bucket and section.	RGET SGETC SGETC SGETC SGETC SGETC	Reads block and presents record 1. Presents record 7. Since the flag is ON in this block, reads next block and presents record 3. Since the flag is OFF in this block, branches to routine named in third operand. Detects E-O-S, sets NULLFLAG to 2, branches to the routine named in second operand. Error - Do not use SGETC after end-of-section.
	The program is searching for record 4 which was not placed in the file.	RGET SGETC SGETC SGETC	Reads block and presents record 1. Presents record 7. Since flag is OFF in this block, branches to routine named in third operand. (RFILE did not place record 4 in any subsequent block.) Reads next block and presents first record.
	The program is searching for record 4. SGETC inputs a null block since records were deleted.	RGET SGETC SGETC SGETC	Reads block and presents record 1. Presents record 7. Since flag is ON in this block, reads next block; because this is a null block, a branch is taken to the routine named in the second operand (regardless of the flag setting in this block). Since the last block was a null block, next block is read (regardless of flag setting in null block) and presents record 4.

The SGETC instruction does not translate the input information.

Example

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
C											S	G	E	T	C		M	A	S	T	E	R	F	I	L	E	,	B	R	R	O	U	T	N	A	M	1	,	B	R	R	O	U	T	N	A	M	2

The above instruction accesses the next record in the current block of records from MASTERFILE. If no more records exist in the current block and the Record Overflow flag is ON, a new block is input. If the Record Overflow flag is OFF, a branch is made to BRROUTNAM2. This branch signals the end of the search since RFILE did not place the desired record in any subsequent block.

If the SGETC instruction inputs a null block, the branch to BRROUTNAM1 is taken. If the SGETC instruction is executed again, a new block is input. A branch is also made to BRROUTNAM1 if end-of-bucket or end-of-section is encountered as a result of this instruction.

Conventions

- Disc and CRAM

SGETC may not be executed after a branch that is caused by end-of-section. An error will occur.

SGETC, used only when processing randomly, is designed to work in conjunction with the RFILE instruction.

## SGETL

### Function

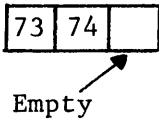
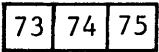
The SGETL instruction, used when processing randomly, accesses the next record in sequence. This instruction, which is coded with two branch operands, also provides sensitivity to end-of-block.

Since the SGETL instruction is sensitive to end-of-block, this instruction allows the user to add records to the end of a block with the INSERT instruction. The SGETL instruction may also be used in creating random file directory entries of the key for the last record in each block.

A branch is taken to the routine referenced in the third operand when the end-of-block is reached. That is, if an SGETL instruction is executed and all the records in the current block have already been presented to the program, the branch is taken. The location immediately following the last record is available for the insertion of a new record. If the INSERT instruction is used and there is sufficient space in the block for the record, the normal rules of INSERT apply. (See "INSERT Instruction" discussed later under this tab.)

A branch is taken to the routine referenced in the second operand for the same reasons as outlined for the SGET instruction (null block, end-of-section, or end-of-bucket), and the Null flag is set in the same manner.

The illustration below shows the result of successive executions of the SGETL instruction when the routine referenced by the third operand provides for the insertion of a record and when the current block is the last in the bucket.

RECORD BLOCK	CONDITION	PROG.	RESULT
	E-O-B The program is seeking to place record 75 into its proper location in the file. SGETL is used instead of SGET to make the program sensitive to end-of-block.	RGET SGETL SGETL	Reads block and presents record 73. Presents record 74. Makes the blank location available to the program and branches to the routine referenced in the third operand. This user routine inserts record 75 into the block and returns control to an SGETL instruction.
		SGETL SGETL	Detects end-of-bucket, sets NULLFLAG to 4, and branches to the routine named in the second operand. Reads next block and presents first record.

The end-of-block branch takes precedence over the end-of-bucket and end-of-section branches. The null block branch is taken instead of the end-of-block branch when a null block is read. After reading a null block, if the SGETL instruction is executed again, a new block is input. The precedence of branches assigned to the second operand is the same as for SGET.

The SGETL instruction does not translate the information being input.

Example

X	REFERENCE																X	OPERATION	X	OPERANDS																																X
	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55			
C												S	G	E	T	L	M	A	S	T	E	,	B	R	R	O	U	T	N	A	M	1	,	B	R	R	O	U	T	N	A	M	2	*								

In the above example, the instruction accesses the next record in the current block of records from MASTERFILE. If no more records exist in the current block, a branch is made to BRROUTNAM2. If a new block is read as a result of this instruction and it is a null block, a branch is taken to BRROUTNAM1. If end-of-bucket or end-of-section is encountered as a result of this instruction, a branch is also made to BRROUTNAM1.

Convention

- Disc and CRAM

SGETL may not be executed after a branch that is caused by end-of-section. An error will occur.

SGETL is used only when processing randomly.



## PUT INSTRUCTIONS

There are two forms of the PUT instruction, PUT and LPUT. Each is discussed separately.

### PUT

#### Function

The PUT instruction is used after a record has been constructed in or moved to the output buffer area. After the instruction is executed, the record previously constructed in the output buffer is no longer available to the program; the next record location is available. The PUT instruction that fills the buffer causes the software to automatically output the block to the named file. Both fixed- and variable-length records are handled automatically; however, it is the programmer's responsibility to ensure that the variable-length indicator (VLI) reflects the true record length. (The programmer always manipulates the VLI in binary when the record is in memory).

The PUT instruction translates the data being output if translation is requested on the file specification sheets.

#### Example

X	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		PUT	N M A G Z F I L E
C		PUT	N M A G Z F I L E , U P D A T E A R E A

In the first example, the instruction puts a record in the output buffer area for NMAGZFILE and makes the next record location available.

In the second example, the instruction moves a record from the work area in memory called UPDATEAREA to the output buffer area. The record in the work area is still available to the program, but the record in the output buffer is not available to the program; the next record location is available.

#### NOTE

When the length of a variable-length record from a chained source-destination file is altered in a work area, do not use PUT with a work area option (use the DELETE and INSERT instructions).

## Conventions

- General

Variable-length indicators (VLI) are not translated when translation of records is requested. They are converted from binary to decimal on output if decimal indicators were requested on the file specification sheets.

The work-area operand must be used if maximum packing of records in a block is desired. The record must be constructed in the work area.

The PUT instruction is used to create source-destination files; however, when processing a source-destination file, the WRITSP or WRITBI instructions must be used to output updated blocks.

If the buffer area (block) is partially full, the software automatically outputs the partial block before closing the file.

- Printer

The first four characters of the printline must contain the printer control block.

If multiple buffers are used, records should be constructed in a work area and then moved to the buffer.

\* \* \* \*

LPUT

Function

This instruction is used to output 80-character, user-label blocks to a magnetic tape file. The LPUT instruction may only be used in a user routine after OPEN or in a user routine before CLOSE. The routine name is specified on the file specification sheets.

After a file is opened or just before it is closed, the software transfers control to the user routine, if desired. The LPUT instruction is used in the routine to write the label block from the output buffer area into the file. After the program writes the labels, a RELINK instruction must be executed to return control to the software. The software then writes a tape mark. (Tape marks indicate the end of a group of labels.)

The first three characters of the label must contain the proper label identifier, for example, UHL for user header label. For complete details concerning label formats, see FILES, tab 2, "Magnetic Tape Files."

If the programmer requests translation on the file specification sheets, the LPUT instruction translates the entire 80-characters being output.

Example

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50			
C																	L	P	U	T																										

In the above example, the instruction outputs an 80-character label block to a file called NMASTERFILE. To access the proper fields when building a label, redefine the record on the data layout sheets for the file by specifying SAME in the location positions. For a complete explanation of redefining a record see the discussion of location in the NEAT/3 REFERENCE MANUAL, INTRODUCTION AND DATA, tab 3, "Data Layout Sheets."

Conventions

● General

The work-area option is not available with this instruction.

The output buffer area for the file must be at least 80 characters long.

## WRITSP INSTRUCTION

### WRITSP

#### Function

This instruction is only used with source-destination files. The WRITSP instruction sets a flag which informs the software to automatically write an updated block of information back into a source-destination file. When a variation of the GET instruction is encountered, the software looks at the flag; if it is ON, the block is written back before another block is read. If a variation of GET is not encountered, the software writes the block back just before closing the file.

#### Example

X	REFERENCE	X	OPERATION	X	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24	25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C			W R I T S P		D A T A F I L E

In the above example, the WRITSP instruction causes the current updated block of information to be written back into a file called DATAFILE. The actual writing takes place just before the next block is read, or if another GET instruction is not encountered, the block is output just before this file is closed.

#### Conventions

- Disc and CRAM

If the WRITSP instruction is used, the work-area option may not be used with GET instructions that refer to this file.

## WRITBI INSTRUCTION

### WRITBI

#### Function

This instruction is used only with source-destination files. Unlike the WRITSP instruction, which outputs the current block just before another block from that file is read or just before that file is closed, the WRITBI instruction writes the current block back to the file immediately. The WRITBI instruction must be used when the buffer area is used for more than one file at a time, to ensure that all updated blocks are written back to their respective files.

#### Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		W R I T B I	D A T A F I L E

In the above example, the WRITBI instruction causes the current block to be written back immediately to the file called DATAFILE.

#### Conventions

- Disc and CRAM

If the WRITBI instruction is used, the work area option may not be used with GET instructions that refer to these files.

## INSERT INSTRUCTION

### INSERT

#### Function

This instruction is used only with chained source-destination files that have records sorted within the bucket. The INSERT instruction places a record from a named work area into a specific position within a file. Both fixed- and variable-length records are handled automatically; however, it is the programmer's responsibility to ensure that the variable-length indicator (VLI) reflects the true record length in binary format.

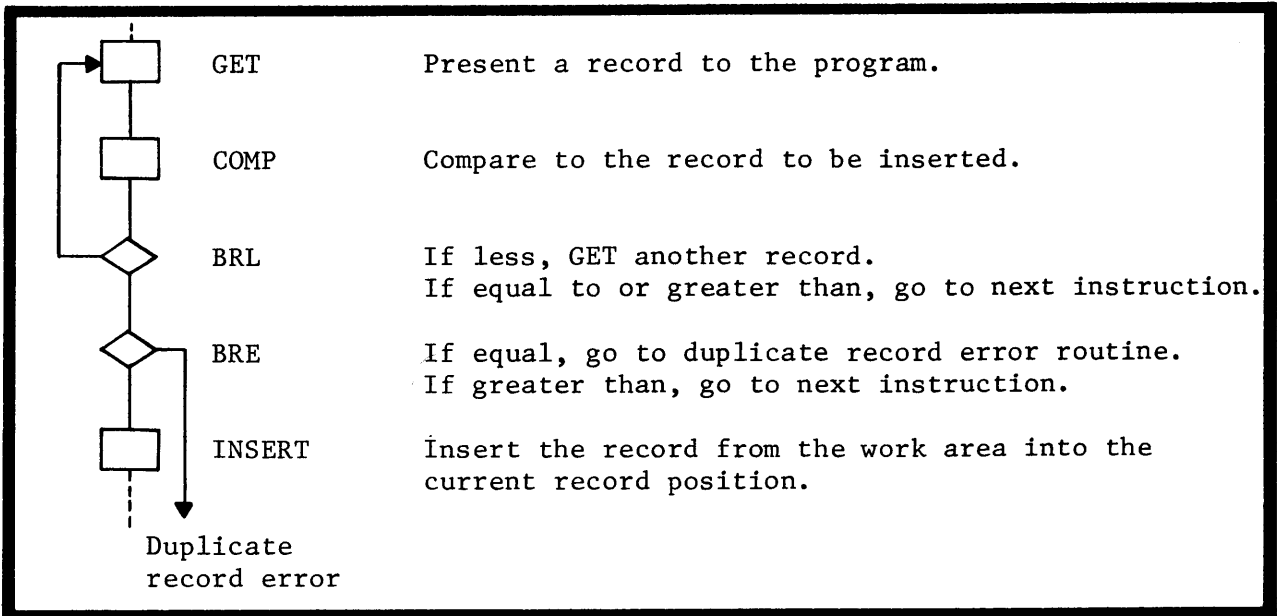
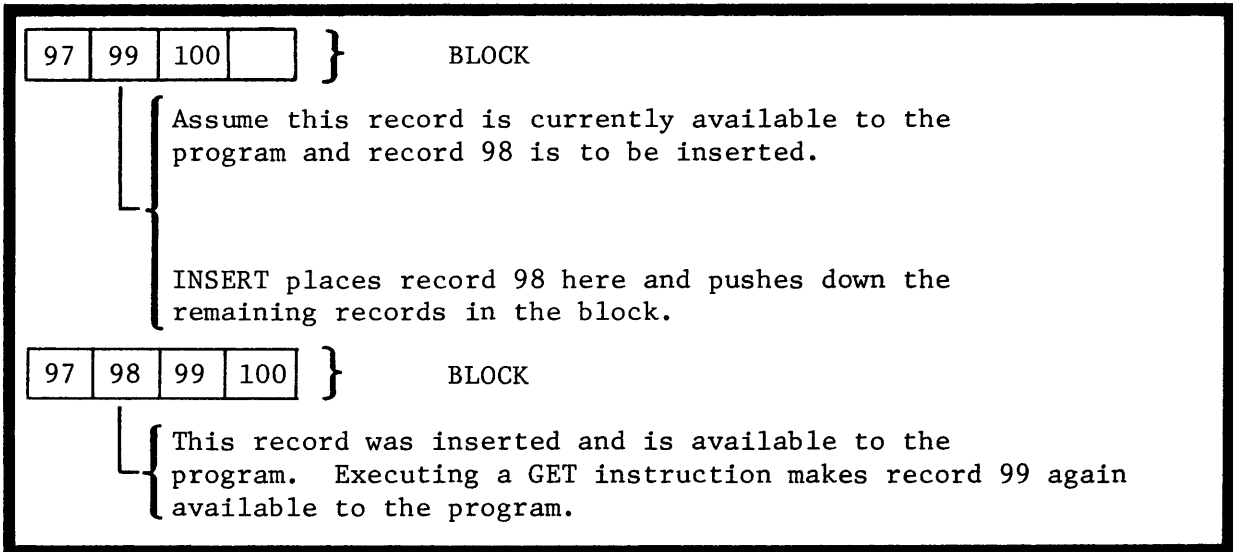
When the record is inserted into the desired location, those records following the inserted record are pushed down in the block and the inserted record is made available to the program. The WRITSP flag is automatically set ON to ensure that the block is written back into the file. (See "WRITSP Instruction" under this tab for further information on the WRITSP flag.)

If insufficient room exists for the record inserted, the last record(s) in the block (and in certain cases the inserted record itself) is pushed down to the beginning of the next block. This pushdown, which includes reading, formatting, and rewriting subsequent blocks, continues automatically until room is finally found in a block. That block may be in the main file area or the overflow area. Once pushdown is completed, the block originally in memory is restored and the inserted record is made available to the program.

If insufficient room exists in a bucket and no overflow area is present, the error exit for random macros (named on the file specification sheets) is taken; for this reason, the use of an overflow area is highly recommended.

#### NOTE

To allow room for push-down, two buffers must be assigned to the file that uses the INSERT instruction.



The proper position for inserting the record may be located by using the GET, RGET, SGET, SGETL, or SGETC instruction and the COMPARE instruction. (Also see "MARK and RESET Instructions" under this tab.)

If the user wishes to make a random access immediately following the execution of the INSERT instruction, he may request that the newly inserted record not be read back into memory by setting the \$NONRESTORE flag ON before coding the INSERT instruction (see the NEAT/3 REFERENCE MANUAL, APPENDIX, tab 1, "File-Oriented System Tags").

Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		INSERT	DATAFILE, WORKAREA

In the above example, the instruction moves the record from the named work area, inserts it into the current record position for a file named DATAFILE, and turns on the WRITSP flag. The work area must be used with this instruction.

Conventions

● Disc and CRAM

Two buffers must be specified for the file that has an INSERT instruction associated with it.

The work-area operand must be used.

The variable-length indicator of a record to be inserted in the file must be in binary format.

The use of an overflow area is highly recommended.

To change the length of a record in a chained, source-destination file, delete the record specifying a work area. After altering the record length, use the INSERT instruction to place the record back into the file. (See "DELETE Instruction" under this tab.)

To add a record to a chained source-destination file that has unsorted records within the buckets, use the RFILE instruction.



## RFILE INSTRUCTION

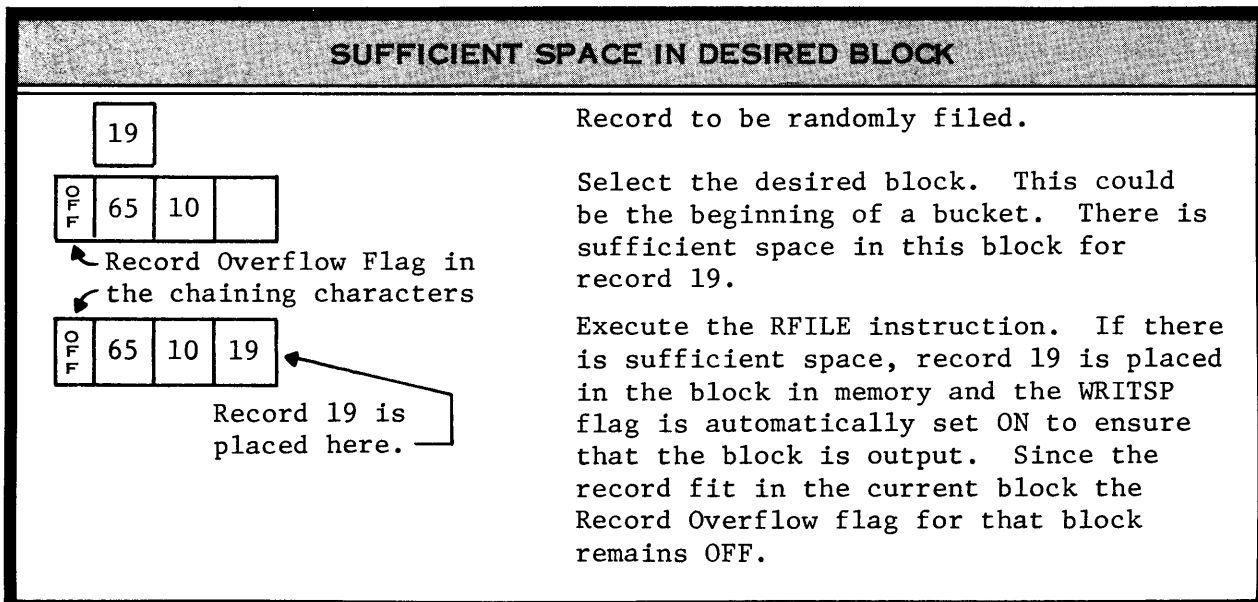
### RFILE

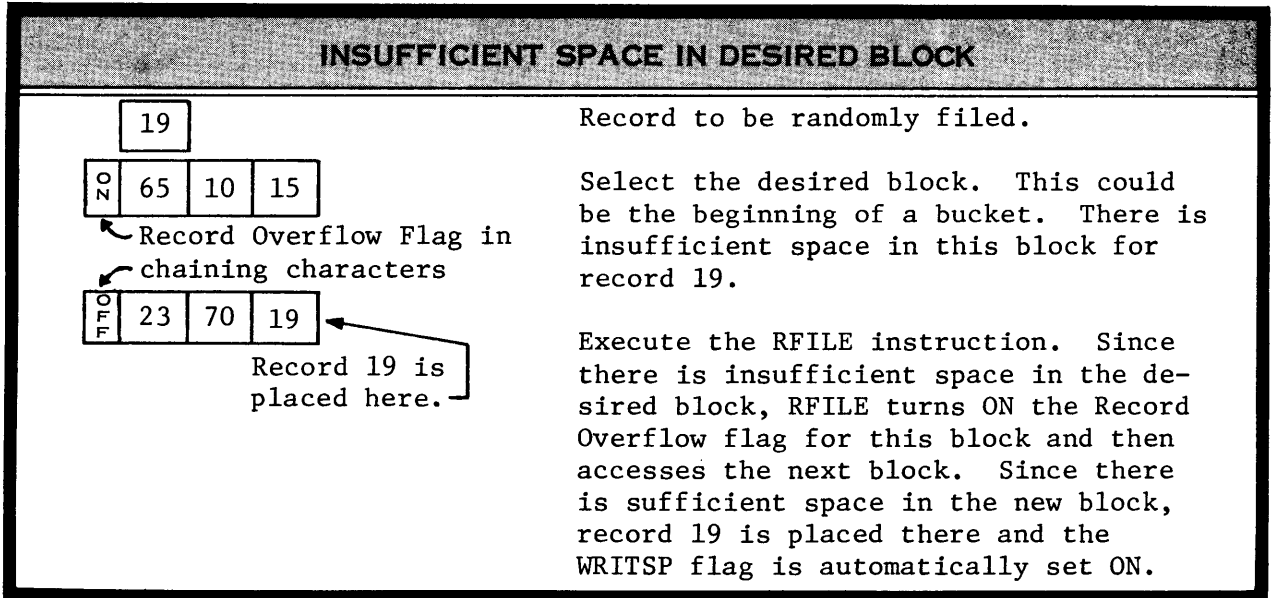
#### Function

This instruction is normally used only with chained source-destination files that have unsorted records within the buckets. The RFILE instruction places a record from a named work area into the block currently in memory. The WRITSP flag is automatically set ON to ensure that the block is written back into the file. (For further information on the WRITSP flag, see "WRITSP Instruction" under this tab.) Both fixed- and variable-length records are handled automatically. The VLI of a variable-length record must be in binary format.

The desired block is usually read into memory using the RGET instruction. If the record does not fit in the desired block, RFILE searches the succeeding blocks until one is found with sufficient room. When a block with sufficient room is found, the record is placed in that block and the WRITSP flag is turned ON to ensure that the block is written back into the file. If room cannot be found in a block in the bucket, the record is placed in a block in the overflow area. If no overflow area exists, the random macro error exit (named on file specification sheets) is taken.

Each time a new block must be accessed in search of sufficient space, the Record Overflow flag in the chaining characters for the block currently in memory is set ON. This flag informs the SGETC instruction that the record being sought may be on one of the following blocks. (See "SGETC Instruction" under this tab.)





Example

X	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		R FILE	DATAFILE, FILEAREA

In the above example, the RFILE instruction moves the record from a work area named FILEAREA and places it in the current block of a file called DATAFILE. If sufficient room does not exist, RFILE turns ON the Record Overflow flag in the chaining characters for the current block, reads in a new block, and attempts to place the record in the new block. The Record Overflow Flag is turned ON so that the SGETC instruction can follow the path taken by RFILE. The WRITSP flag is also turned ON.

Conventions

- Disc and CRAM
  - The work-area operand must be used.
  - Variable-length records to be added to the file must contain binary variable-length indicators.
  - The use of an overflow area is highly recommended.
  - Use RFILE if the records are unsorted within the bucket.
  - Use INSERT if the records are sorted within the bucket.

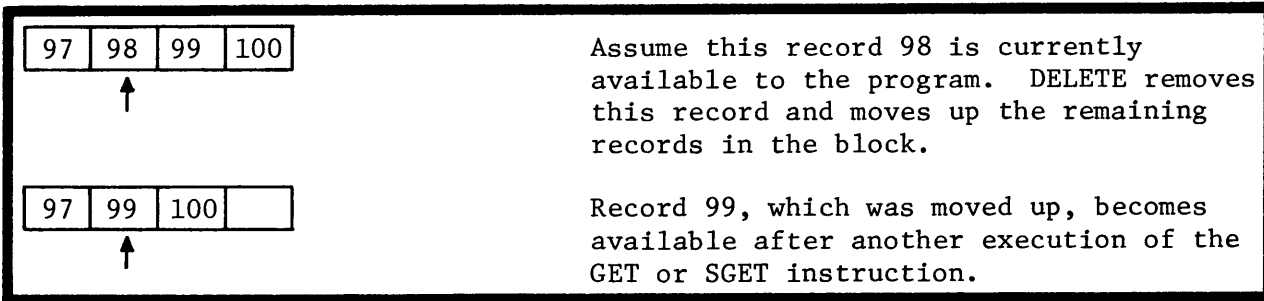
## DELETE INSTRUCTION

### DELETE

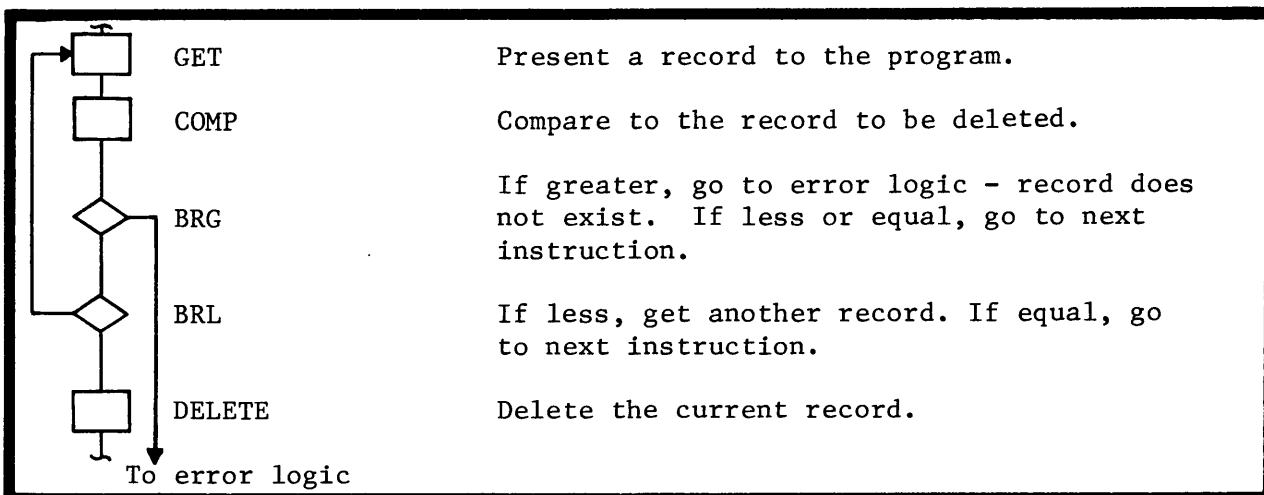
#### Function

This instruction is only used with chained, source-destination files. The instruction removes from the named file the record currently available to the program. Both fixed- and variable-length records are handled automatically.

The records following the deleted one move up in the block. The next record is available to the program only after another execution of the GET or SGET instruction. The records in the following blocks do not move up in the bucket. If the deleted record was the only one in the block, the block becomes a null block.



The record to be deleted may be located in the file by using the GET, RGET, or SGET instruction and the COMPARE instruction.



Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		DELETED	DATAFILE
C		DELETED	DATAFILE, WORKAREA

In the first example, the instruction removes the current record from a file called DATAFILE. The remaining records in the block are moved up.

In the second example, the instruction functions the same as in the first except that the record is also moved into the specified work area. This option must be used if a record is to be altered in length and then inserted back into the file.

Conventions

- Disc and CRAM

To change the length of a record in a chained source-destination file, delete the record specifying a work area. After altering the record length, use the INSERT instruction to place the record back into the file.

A GET or SGET instruction must be executed to access the record following the deleted record.

## MARK AND RESET INSTRUCTIONS

The MARK and RESET instructions are interrelated and are therefore explained together.

### MARK and RESET

#### Function

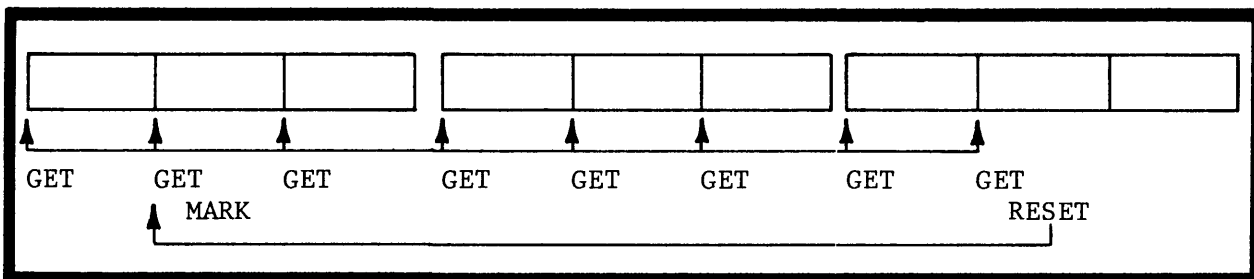
These two instructions are used to save the location of a record from a disc or CRAM file and to place that record back into memory at a later time. MARK and RESET may be used with source and source-destination files, and with destination files if the programmer specifies the secondary usage of the file as source or source-destination (on file specification sheet 2).

The MARK instruction stores the address of the current block, as well as the location of the current record within that block. This information is placed in a 10-character binary area reserved by the programmer.

PAGE	×	LINE	×	REFERENCE	×	LOCATION	×	LENGTH	×	DP	×	TYPE	×	VALUE OR PICTURE																																						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50			
						D	S	A	V	E	A	R	E	A	A	A									1	0			B																							

The programmer must ensure that the record location saved by the MARK instruction is still valid at the execution of the RESET instruction. The insertion of a record in a position preceding the saved record location in this bucket, or the deletion of a record preceding the saved record location in this block, will invalidate the saved record location.

The RESET instruction uses the information previously stored with the MARK instruction to move the block back into memory and to make the same record within the block available to the program. If the programmer wishes to place the record back into a work area that is being used with the file, he must use a MOVE instruction to move it from the buffer to the work area.





Conventions

● Disc and CRAM

If the RESET instruction is used, the programmer must answer yes to the question "Are any random accesses made to this file during this program?" on file specification sheet 2.

MARK and RESET may be used with a destination file only when the programmer specifies source or source-destination as its secondary usage on file specification sheet 2.

## BLKCHK INSTRUCTION

### BLKCHK

#### Function

This instruction is generally used in creating chained destination files that will become chained source-destination files; however, it may be used with any type file.

When the BLKCHK instruction is used, the current size of a block (less any block header) to be output is compared to a binary number that is stored in an area reserved by the programmer. If the current block size is equal to or greater than the stored binary number, a branch is taken.

After the branch is taken, the BLKOUT instruction may be used to write out the incomplete block. (See "BLKOUT Instruction" under this tab.) In this way, the programmer may place incomplete blocks in a file to allow for the future insertion of records.

#### Example

X	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		BLKCHK	NEWFILE, BLKSIZE, BRROUTNAME

The programmer must reserve a 2-character binary area to contain the desired constant.

X	REFERENCE	CODE	LOCATION	LENGTH	DP	T	Y	P	VALUE OR PICTURE
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24 25 26 27	28 29	30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50		
D	BLKSIZE	A		2		B	400		

In the example above, the instruction compares the current length of a block to the number stored in BLKSIZE. If the block length is equal to or greater than 400 characters, a branch is made to the routine with the name BRROUTNAME. The BLKOUT instruction may be used in this routine to write out the block to NEWFILE.



NOTE

The BLKCHK instruction cannot be used to check for the maximum block size as specified on the file specification sheet (the PUT instruction that causes the block to reach maximum size also causes it to be output and the current block size to be reset to zero).

## BLKOUT INSTRUCTION

### BLKOUT

#### Function

This instruction is generally used in creating chained destination files that will become chained source-destination files. The BLKOUT instruction outputs both partially full blocks and null blocks to the named file. This allows room in the file for the future insertion of records.

#### Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		BLKOUT	NEWFILE

In the example above, the BLKOUT instruction outputs the current block in the output buffer area to NEWFILE.

## DEFAULT INSTRUCTION

### DEFAULT

#### Function

The DEFAULT instruction may be used with any sequentially processed disc or CRAM file to control the closing of one section of a file and the opening of the next section.

This instruction is generally used to close the current section of a file before the end of section is reached, and then to open the following section. For CRAM files, however, this instruction may also be used to close sections of associated files and to open their next sections if this process (file alternation) is not performed automatically when one file reaches end of section.

When the DEFAULT instruction is executed for a destination file, all buffers not yet output are written out, including any partially full buffer. The current section of the file is then closed and the next section is opened.

When a section of a source file is closed with the DEFAULT instruction, any data remaining in the section at the time the DEFAULT instruction is executed is not read. (For source files on disc, a message is placed in the log to indicate that some of the data in the file may not have been processed.) When a DEFAULT instruction is executed for the last section of a source file, a branch is taken to the end-of-file routine specified on the file specification sheets for the file. The last section is not closed until a FINISH or a CLOSE instruction is encountered.

The programmer may name from one to six files as operands of a single DEFAULT instruction; however, all the files must be on the same type of peripheral.

#### Closing the Current Section

As an example of the first use of the DEFAULT instruction, assume that a programmer desires to process only a selected range of records from two sections of a disc source file (SOURCEFILE) and then place these processed records into two sections of a disc destination file (DESTFILE). The programmer would code the following DEFAULT instruction and incorporate it within the coding of the main body of the program.

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
REFERENCE											OPERATION						OPERANDS																										
C											DEFALT						SOURCEFILE, DESTFILE																										

In the example above, the DEFALT instruction terminates input from SOURCEFILE. The current section is then closed and the next section is opened. Since the file is a disc file, a message is placed in the log to indicate that some of the data may not have been processed. This instruction also closes the current section of DESTFILE and opens a new section. All buffers not yet output are written to DESTFILE before the current section is closed.

### Alternating Associated Files

For CRAM files, the DEFALT instruction may also be used to process multi-section, associated files when file alternation is not performed automatically (the term "associated files" refers to all files being processed concurrently on the same CRAM deck).

When a sequentially processed CRAM file reaches end of section, the software determines the location of the next section of the file. If the next section is on a different deck, any associated destination file is automatically alternated (its current section is closed and its next section opened) if both (or all) specify the same symbolic unit designator. Since file alternation does not occur automatically for other type of associated files, the DEFALT instruction must be used to alternate:

- Associated source and source-destination files.
- Associated destination files whose next sections are not on the same deck (not the same SUD) as the next section of the file that originally reached end of section.

When the programmer uses the DEFALT instruction to alternate CRAM associated files, he must specify the file that originally reached end of section as one of the operands of the instruction and then incorporate the instruction in the end-of-section routine for that file.

As an example of this use of the DEFALT instruction, assume that a programmer is processing two source files (SOURCEFILE1 and SOURCEFILE2) whose first sections are on the same CRAM deck and whose second sections are on another deck. Assume also that SOURCEFILE1 reaches end of section. Since the CRAM deck must be changed in order to open the files' next sections, and since SOURCEFILE2 is not closed automatically, the programmer must code the following DEFALT instruction to close SOURCEFILE2. This instruction must be incorporated within the end-of-section routine for SOURCEFILE1, the file that originally reached end of section.

M	REFERENCE	M	OPERATION	M	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50		
C		D E F A L T			S O U R C F I L E 1 , S O U R C F I L E 2

In the example above, the DEFALT instruction terminates input from SOURCFIIE2. The current section is closed and, after the CRAM deck is changed, the new sections of both source files are opened. Notice that the file originally reaching end of section (SOURCFIIE1) is specified as one of the operands.

## COMPARE INSTRUCTION

### COMPARE (COMP)

#### Function

Compare is one of the most useful of the general instructions. Its purpose is to compare two operands that have been previously defined on the data layout sheets.

The COMP instruction compares the first operand to the second.

- If the first operand is greater than the second, the G flag is turned ON in the central processor.
- If the first operand is equal to the second, the E flag is turned ON in the central processor.
- If the first operand is less than the second, the L flag is turned ON in the central processor.

Examples:

<u>1st Operand</u>	<u>2nd Operand</u>	<u>Flag</u>
2	1	G
3	3	E
1	2	L

The COMP instruction also compares signs:

+2	-2	G
-2	-2	E
-2	+2	L

(The one exception is that +0 and -0 compare as equal.)

The COMP instruction also compares decimal positions:

2.	.2	G
.2	.2	E
.2	2.	L

Example

x	REFERENCE	x	OPERATION	x	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24	25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		C	O M P	T	O T A L F L D A , T O T A L F L D B

Both operands can be reference tags, or either operand (but not both) can be a literal. Both operands must be in the same form (alphanumeric, decimal, binary, etc.); they may, however, be of different lengths with different decimal point alignments.

Conventions

A branch instruction (or successive branch instructions) should immediately follow the compare instruction. Only the branch instruction preserves the state of the G, E, and L flags; other instructions may change these flags.

The following table lists the data types that can be compared and the maximum lengths of operands for the different data types:

CAN COMPARE TYPE TO TYPE		MAXIMUM LENGTH OF OPERAND	
		SOURCE	DESTINATION
B	B	8	8
D	D	20	20
U&Z	U&Z	19	19
D	U&Z	20	19
U&Z	D	19	20
X&S	X&S	64K	64K
K*	K*	10	10
P#	P#	10	10

B -- Binary  
D -- Signed Decimal  
U&Z -- Unsigned Decimal  
X&S -- Alphanumeric Character Set  
K -- Unsigned Packed Decimal  
P -- Signed Packed Decimal

\* When comparing unsigned packed decimal data using the NCR Century 100, both operands must have the same number of decimal positions because the COMP instruction treats packed operands as binary integers and ignores decimal points.

# Signed packed decimal data cannot be compared on the NCR Century 100.

On the NCR Century 200, both type K data and type P data may be compared separately. The operands of the COMP instruction are aligned on the decimal point before they are compared.

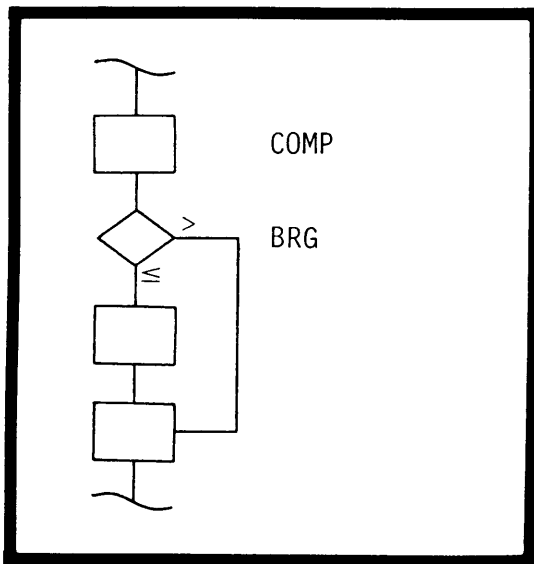
## BRANCH INSTRUCTIONS

Branch or one of its variations is generally used following the COMP instruction to branch to a different portion of the program. If the COMP instruction renders a greater-than decision, it may be necessary to branch to one portion of the program; if equal-to, to another portion; if less-than, to still another.

### BRANCH GREATER (BRG)

#### Function

The BRG (Branch-if-Greater) instruction tests the G flag in the central processor. If the G flag is ON, the BRG instruction transfers control to the instruction named in the operands column. If the G flag is OFF, control continues to the next instruction in sequence.



#### Example

X	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		B R G	A D D T O T A L

ADDTOTAL is the reference tag of the instruction to which control is transferred if the G flag is ON.



## BRANCH EQUAL (BRE)

### Function

The BRE (Branch-if-Equal) instruction tests the E flag. If the E flag is ON, the BRE instruction transfers control to the instruction named in the operands column. If the E flag is OFF, control continues to the next instruction in sequence.

### Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		B R E	E N D O F P A G E

ENDOFFPAGE is the reference tag of the instruction to which control is transferred if the E flag is ON.

---

## BRANCH LESS (BRL)

### Function

The BRL (Branch-if-Less) instruction tests the L flag. If the L flag is ON, the BRL instruction transfers control to the instruction named in the operands column. If the L flag is OFF, control continues to the next instruction in sequence.

### Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		B R L	C L R P R N T L N E

CLRPRNTLNE is the reference tag of the instruction to which control is transferred if the L flag is ON.

BRANCH GREATER OR EQUAL (BRGE)

Function

The BRGE (Branch-if-Greater-or-Equal) instruction tests the L flag. If it is OFF, the BRGE instruction transfers control to the instruction named in the operands column. If the L flag is ON, control continues to the next instruction in sequence.

Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		BRGE	ADDTOTAL

ADDTOTAL is the reference tag of the instruction to which control is transferred if either the G or the E flag is ON.

BRANCH LESS OR EQUAL (BRLE)

Function

The BRLE (Branch-if-Less-or-Equal) instruction tests the G flag. If it is OFF, the BRLE instruction transfers control to the instruction named in the operands column. If the G flag is ON, control continues to the next instruction in sequence.

Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		BRLE	ENDOFFPAGE

ENDOFFPAGE is the reference tag of the instruction to which control is transferred if either the L or the E flag is ON.

BRANCH UNEQUAL (BRU)

Function

The BRU (Branch-if-Unequal) instruction tests the E flag. If the E flag is OFF, the BRU instruction transfers control to the instruction named in the operands column. If the E flag is ON, control continues to the next instruction in sequence.

Example

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50					
C											B	R	U				A	D	D	T	O	T	A	L																								

ADDTOTAL is the reference tag of the instruction to which control is transferred if the E flag is OFF.



UNCONDITIONAL BRANCH (BR)

Function

The BR instruction transfers control unconditionally to the instruction named in the operands column.

Example

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50							
C											B	R					A	D	D	S	U	M																												

ADDSUM is the reference tag of the instruction to which control is transferred.

\*\*\*

## LINK AND RELINK INSTRUCTIONS

### LINK

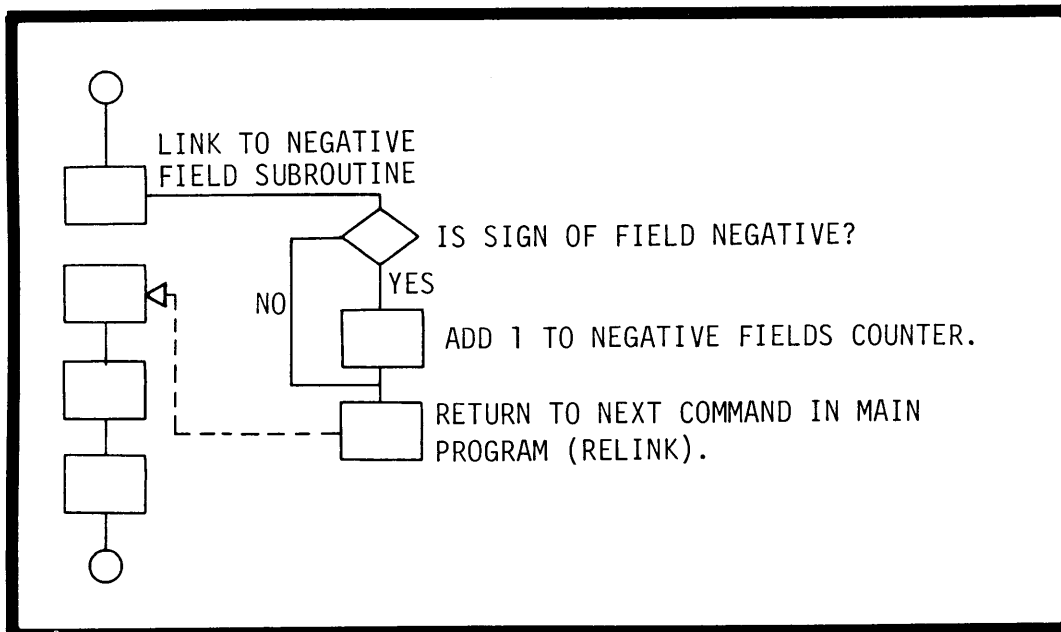
#### Function

The LINK instruction transfers control to the subroutine named in the operands column. A subroutine is a self-contained set of instructions located outside the main program flow. The LINK instruction provides a method for transferring control to the subroutine when it is needed by the main program. (LINK also enables a subroutine to link to a sub-subroutine to perform some portion of its task.)

Before transferring control to the subroutine, the LINK instruction saves the address of the next instruction in the main program so that control can be returned to that address at the end of the subroutine. (See RELINK.)

Do not confuse LINK, which saves the address to which it is to return, with BR, which transfers control unconditionally to the address specified in its operand. If the programmer wishes to return from a subroutine to the point in the program from which he departed, he should use LINK and RELINK. If he does not wish to return to that point, however, he should use BR. For example, a programmer uses a branch instruction with an error-handling subroutine because, when returning from the subroutine, control will always be transferred to the same point in the program (GET another record, for example) instead of to the point of departure.

It may be important for a programmer who is working with many numeric data fields to know if certain fields are negative. He may want to link to the following subroutine many times during his program to determine if certain fields are negative and to act upon those fields that are.



Example

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49			
C											L	I	N	K			N	E	G	F	I	E	L	D																					

NEGFIELD is the reference tag of the first instruction in the subroutine to which control is transferred.

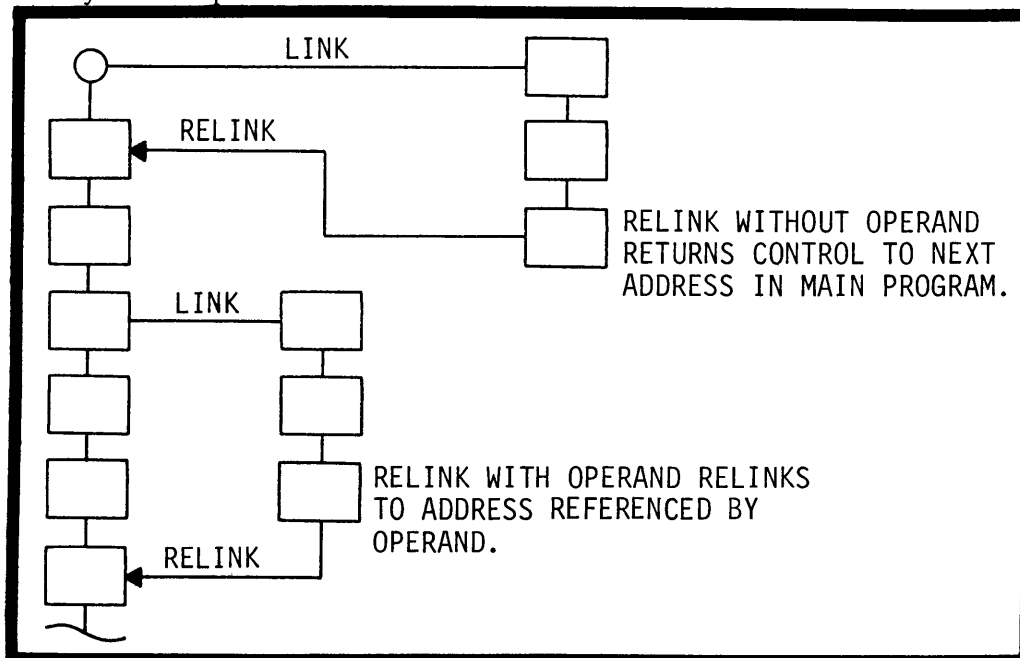
Conventions

There should be a RELINK executed for every LINK executed.

RELINK

Function

This instruction is used at the end of a subroutine or sub-subroutine to remove the address stored in the link list when the LINK instruction was last executed. Control is returned to that address if no operand is specified in the RELINK instruction; if an operand is specified, RELINK transfers control to the routine referenced by that operand.



Example

×	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		RELINK	
C		RELINK	SUMRYPRINT

In the first example, the address saved by the last LINK instruction is removed from the link list and control is transferred to that address.

In the second example, the address saved by the last LINK instruction is removed from the link list and control is transferred to the instruction whose reference tag is SUMRYPRINT.

Conventions

At times the software stores a link in the link list to aid the programmer (e.g. printer end-of-page routine). In this case the programmer needs only to RELINK without a corresponding LINK instruction.

If there is no link in the link list when relinking, the software stores a log message and displays a message to the operator.

## MOVE INSTRUCTION

The MOVE instruction enables the user to move data internally from one area to another (e.g. move a title line from a constant area to the printer buffer).

MOVE has three basic variations: the standard MOVE, the conversion of data type with MOVE, and the editing MOVE.

The basic format for all MOVE instructions follows:

		REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	
C		MOVE	OPERANDA , OPERANDB	

The first operand can be either a literal or the reference tag of the operand to be moved; the second operand must be the reference tag of the destination operand. (If the first operand is a literal, it is converted to the data type of the second operand.)

Since the move is accomplished from right to left, overlapping fields cannot be moved without destroying the overlapping location. To correct this situation, move one character at a time (as illustrated below) when moving overlapping fields.

×	REFERENCE	×	LOC CODE	LOCATION	×	LENGTH	×	DP	×	TYPE	VALUE OR PICTURE
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24 25 26 27	28 29	30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50				
D	N U M B E R S	A				3					
D	O P E R A N D A 1	F		0		1					
D	O P E R A N D A 2	F		1		1					
D	O P E R A N D B 1	F		1		1					
D	O P E R A N D B 2	F		2		1					

×	REFERENCE	×	OPERATION	×	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	
C	1 S T M O V E		M O V E		O P E R A N D B 1 , O P E R A N D A 1
C	2 N D M O V E		M O V E		O P E R A N D B 2 , O P E R A N D A 2





STANDARD MOVE INSTRUCTIONS

Function

The standard MOVE instruction simply transfers the first operand to the second. If the data does not fit the destination operand, it is adjusted (truncated or expanded) to fit.

Examples

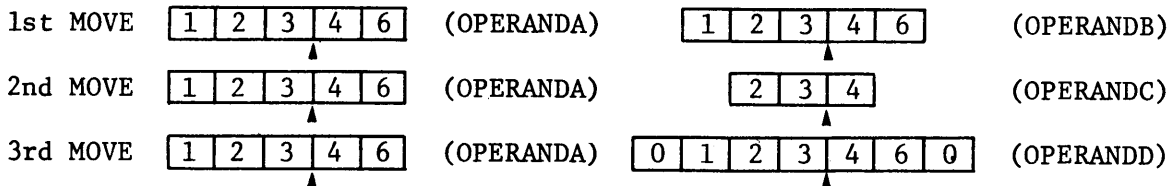
• Unsigned Decimal Numeric Data

In a decimal numeric MOVE, source data is aligned on the decimal point. (A decimal point is always assumed in decimal numeric data.) Consider the following data definitions and the MOVE instruction:

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
REFERENCE											MODE	LOCATION										LENGTH	DP	TYPE	VALUE OR PICTURE																		
DNAME											A											20																					
OPERANDA											F	0										5	2	U																			
OPERANDB											F	5										5	2	U																			
OPERANDC											F	10										3	1	U																			
OPERANDD											F	13										7	3	U																			

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
REFERENCE											OPERATION										OPERANDS																						
C1STMOVE											MOVE										OPERANDA, OPERANDB																						
C2NDMOVE											MOVE										OPERANDA, OPERANDC																						
C3RDMOVE											MOVE										OPERANDA, OPERANDD																						



NOTE: Carets (▲) indicate implied decimal points.

In the first example, the source data aligned on the decimal point fits perfectly into the destination operand.

In the second example, the data is aligned on the decimal point and the first and last characters are truncated because they do not fit into the destination operand.

In the third example, the data is aligned on the decimal point and zeros are filled into the extra character positions.

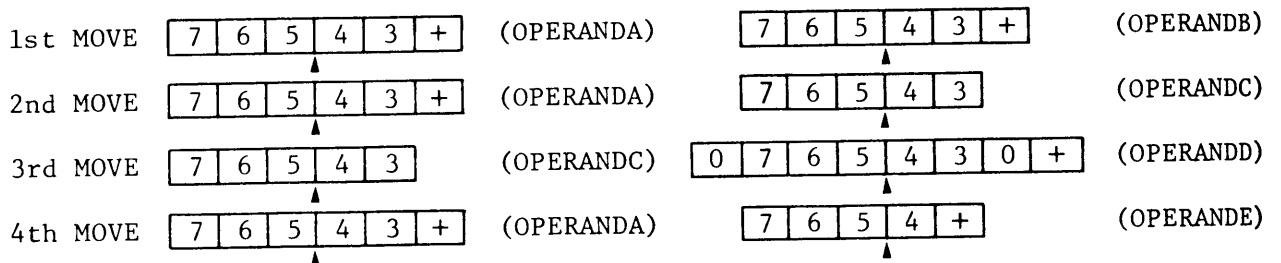
• Signed Decimal Numeric Data

The MOVE instruction will move the sign of the source data. Unsigned decimal operands can also be moved to signed decimal operands, or vice versa. Such a move simply adds or deletes a sign. (If a sign is added, however, it is always positive.) Consider the following data definitions and the MOVE instruction:

REFERENCE	MODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
7 8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27	28 29 30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50		
DNAME	A		3 0			
OPERANDA	F	0	6	2	D	
OPERANDB	F	6	6	2	D	
OPERANDC	F	1 2	5	2	U	
OPERANDD	F	1 7	8	3	D	
OPERANDE	F	2 5	5	1	D	

REFERENCE	OPERATION	OPERANDS
7 8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C1STMOVE	MOVE	OPERANDA, OPERANDB
C2NDMOVE	MOVE	OPERANDA, OPERANDC
C3RDMOVE	MOVE	OPERANDC, OPERANDD
C4THMOVE	MOVE	OPERANDA, OPERANDE



In the first example the source data aligned on the decimal point fits perfectly into the destination operand.

In the second example, the data is aligned on the decimal point and the left character is truncated because there is no place for it. The sign is dropped because OPERANDC is defined as unsigned.

In the third example, an unsigned field is moved to a signed field and the extra character positions are filled with zeros.

In the fourth example, the data is aligned on the decimal point and the right character is truncated.

• Binary Data

The MOVE instruction moves binary data according to the same general rules applied to moving decimal numeric data. Consider the following data definitions and the MOVE instruction:

REFERENCE	MODE	LOCATION	LENGTH	DP	SYSTEM	VALUE OR PICTURE
D NAME	A		8			
D OPERANDA	F	0	2		B	
D OPERANDB	F	2	3		B	
D OPERANDC	F	5	1		B	
D OPERANDD	F	6	2		B	

REFERENCE	OPERATION	OPERANDS
C 1ST MOVE	MOVE	OPERANDA, OPERANDD
C 2ND MOVE	MOVE	OPERANDA, OPERANDC
C 3RD MOVE	MOVE	OPERANDA, OPERANDB

(OPERANDA)

1st MOVE    10101111|10001000                    10101111|10001000                    (OPERANDD)

2nd MOVE    10101111|10001000                    10001000                                    (OPERANDC)

3rd MOVE    10101111|10001000                    00000000|10101111|10001000                    (OPERANDB)

The first example illustrates a binary move to an operand of equal length.

In the second example, two binary characters are moved to an operand containing only one character, and the left character is truncated.

In the third example, two binary characters are moved to an operand containing three binary characters and the extra character is zero-filled.

● Packed Data

Packed data, like unpacked data, aligns on the decimal point. Consider the following data definitions and the MOVE instruction:

REFERENCE	CODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
D N A M E	A		10			
D O P E R A N D A	F	0	2	2	K	
D O P E R A N D B	F	2	3	2	K	
D O P E R A N D C	F	5	2	0	P	
D O P E R A N D D	F	7	3	2	P	

REFERENCE	OPERATION	OPERANDS
C 1 S T M O V E	M O V E	O P E R A N D A , O P E R A N D B
C 2 N D M O V E	M O V E	O P E R A N D C , O P E R A N D D

1st MOVE    34 56    (OPERANDA)    00 34 56    (OPERANDB)

2nd MOVE    27 5+    (OPERANDC)    27 50 0+    (OPERANDD)

In the first example, the packed data is aligned on the decimal point and the extra character positions are zero-filled to the left.

In the second example, the packed data is aligned on the decimal point and the extra character positions are zero-filled to the right.

• Alphanumeric Data

Alphanumeric data is left-justified and space-filled to the right. Consider the following data definitions and the MOVE instruction:

REFERENCE	MODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
D N A M E	A		17			
D O P E R A N D A	F	0	4		X	
D O P E R A N D B	F	4	4		X	
D O P E R A N D C	F	8	3		X	
D O P E R A N D D	F	11	6		X	

REFERENCE	OPERATION	OPERANDS
C 1 S T M O V E	M O V E	O P E R A N D A , O P E R A N D B
C 2 N D M O V E	M O V E	O P E R A N D A , O P E R A N D C
C 3 R D M O V E	M O V E	O P E R A N D A , O P E R A N D D

1st MOVE    D E F G    (OPERANDA)    D E F G    (OPERANDB)

2nd MOVE    D E F G    (OPERANDA)    D E F    (OPERANDC)

3rd MOVE    D E F G    (OPERANDA)    D E F G ☐ ☐    (OPERANDD)

(☐ is the symbol used to indicate a space.)

The first example illustrates a simple move to an operand of equal length.

In the second example, the right character is truncated to make the data fit the destination operand.

In the third example, the extra characters are space-filled.

- MOVE to Zero-Fill a Numeric Field

A numeric field can be zero-filled with the MOVE instruction simply by moving a literal zero to the destination field.

X	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		M.O.V.E.	'0', OPERAND N

The literal character ('0') is right-justified in the numeric destination field; the rest of the field is automatically zero-filled to the left (regardless of decimal positions).

- MOVE to Clear an Alphanumeric Field

An alphanumeric field can be space-filled with the MOVE instruction simply by moving a literal space to the destination field.

X	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		M.O.V.E.	' ' , OPERAND A

In accordance with the rules for moving alphanumeric data, the literal space character (' ') is left-justified and the rest of the field is automatically space-filled to the right.

Conventions

The following table lists the data types that can be moved with the standard MOVE instruction and the permitted lengths of operands for the different data types:

CAN MOVE TYPE TO TYPE		MAXIMUM LENGTH OF OPERAND		
		SOURCE	DESTINATION	
B	B	8	8	B -- Binary D -- Signed Decimal K -- Unsigned Packed Decimal X&S -- Alphanumeric Character Set U&Z -- Unsigned Decimal P -- Signed Packed Decimal
D	D	20	20	
U&Z	U&Z	19	19	
D	U&Z	20	19	
U&Z	D	19	20	
P	P	10	10	
K	K			
K	P			
P	K			
X&S	X&S	64K	64K	

Binary Data (B): right-justified and zero-filled to the left.

Decimal Numeric Data (D, U, P, K): first aligned on the decimal point, if there is a decimal point. After alignment, both the integer and decimal portions are zero-filled or truncated.

Alphanumeric Data (X): left-justified, then space-filled to the right.

The standard MOVE can also accommodate the following data moves:

CAN MOVE TYPE TO TYPE		MAXIMUM LENGTH OF OPERAND		
		SOURCE	DESTINATION	
B	X&S	64K	64K	B -- Binary D -- Signed Decimal U&Z -- Unsigned Decimal P -- Signed Packed Decimal K -- Unsigned Packed Decimal E -- Edited X&S -- Alphanumeric Character Set
D				
U&Z				
P				
K				
E				

All moves to an alphanumeric field (X) are left-justified and space-filled to the right.

CONVERSION MOVE INSTRUCTION

Function

This variation of the MOVE instruction converts the data type of the first operand to the data type of the second as the MOVE is being executed.

Examples

- Binary-to-Decimal and Decimal-to-Binary

Consider the following data definitions and the MOVE instruction:

REFERENCE	MODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
D N A M E	A		8			
D O P E R A N D A	F	0	1		B	
D O P E R A N D B	F	1	1		B	
D O P E R A N D C	F	2	3		U	
D O P E R A N D D	F	5	3		U	

REFERENCE	OPERATION	OPERANDS
C 1 S T M O V E	M O V E	O P E R A N D A , O P E R A N D C
C 2 N D M O V E	M O V E	O P E R A N D D , O P E R A N D B

1st MOVE (Binary-to-Decimal)     00010001     (OPERANDA)     0 1 7     (OPERANDC)  
   |← 1 char →|                                    |← 3 char →|

2nd MOVE (Decimal-to-Binary)     1 3 6     (OPERANDD)     10001000     (OPERANDB)  
   |← 3 char →|                                    |← 1 char →|

In the first example, the binary data character is converted to its decimal numeric equivalent, moved to a 3-character destination operand, right-justified, and zero-filled to the left.

In the second example, the decimal numeric source data is converted to its binary equivalent and moved to a 1-character destination operand.



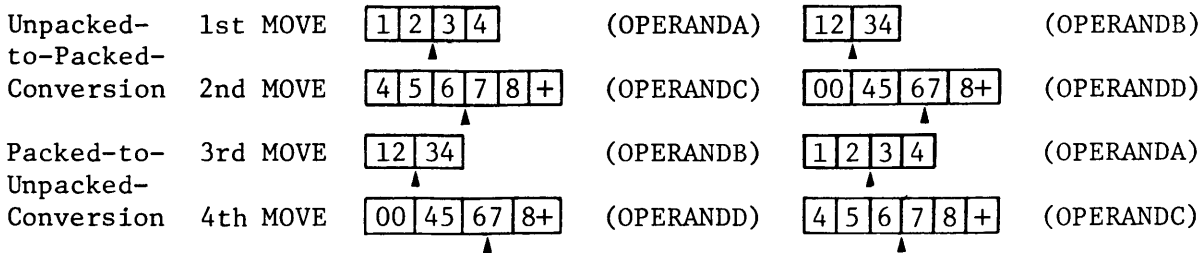
● Unpacked-to-Packed and Packed-to-Unpacked (Fields)

Data can also be packed and unpacked by use of the MOVE instruction. Consider the following data definitions and the MOVE instruction:

REFERENCE	MODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
NAME	A		16			
OPERANDA	F	0	4	2	U	
OPERANDB	F	4	2	2	K	
OPERANDC	F	6	6	2	D	
OPERANDD	F	12	4	2	P	

REFERENCE	OPERATION	OPERANDS
1ST MOVE	MOVE	OPERANDA, OPERANDB
2ND MOVE	MOVE	OPERANDC, OPERANDD
3RD MOVE	MOVE	OPERANDB, OPERANDA
4TH MOVE	MOVE	OPERANDD, OPERANDC



In the first example, unpacked source data consisting of four characters is converted to packed data and moved to a destination operand of two characters.

In the second example, unpacked source data consisting of six characters is converted to packed data, moved to a destination operand consisting of four characters, aligned on the decimal point, and zero-filled.

The third and fourth examples are the reverse of the first two, converting packed data to unpacked data.

Conventions

The following table lists the data types that can be moved with the conversion variation of the MOVE instruction and the permitted lengths of operands for the different data types:

CAN CONVERT TYPE TO TYPE		MAXIMUM LENGTH OF OPERAND	
		SOURCE	DESTINATION
B	D	8	20
B	U&Z	8	19
K	B	20	8
U&Z	B	19	8
D	P	20	10
U&Z	P	19	10
D	K	20	10
U&Z	K	19	10
P	D	10	20
P	U&Z	10	19
K	D	10	20
K	U&Z	10	19

B -- Binary  
 D -- Signed Decimal  
 U&Z -- Unsigned Decimal  
 P -- Signed Packed Decimal  
 K -- Unsigned Packed Decimal

\*\*\*\*

EDITING MOVE INSTRUCTIONFunction

The editing variation of the MOVE instruction permits the user to rearrange the format of data for printing. Editing may involve the deletion of unwanted data; the selection of pertinent data; the application of different formatting techniques; the insertion of symbols (such as commas and decimal points); and the application of such techniques as zero suppression, floating currency symbols, check-protect symbols, and sign control.

The editing MOVE instruction edits the contents of the source operand into the destination operand, using the editing mask defined on the data layout sheet. The mask, which is actually a definition of the destination operand, is composed of a string of characters that describe the format of the data to be output. (The characters that may be used in the editing mask are explained in detail in the section titled INTRODUCTION AND DATA, tab 3, "Data Layout Sheets," Editing Masks.)

Examples

There are two basic variations of the editing MOVE instruction: the decimal-numeric editing MOVE (which, in turn, has five variations) and the alphanumeric editing MOVE.

● Decimal-Numeric Editing MOVE

The source operand in a numeric editing MOVE is aligned on the decimal point of the mask. (Decimal points are assumed for all decimal-numeric source data.) If the data does not fit the destination operand, it is adjusted (truncated or expanded) to fit.

NOTE: The DP column of the data layout sheet must be filled in when using the decimal-numeric editing MOVE, even when the entry is zero.

There are five variations of the decimal-numeric editing MOVE:

- Sign editing
- Insertion editing
- Zero-suppression and replacement editing
- Check protect editing
- Floating currency editing

- Sign Editing

The source data for this MOVE must be defined as either signed or unsigned decimal numeric data. (If unsigned, it is assumed to be positive.) There are four valid sign symbols that may be used in the destination operand: +, -, CR (credit), and DB (debit). The + and - symbols may appear at either end of the mask (but not both). CR and DB may be placed only at the right end of the mask; they are inserted only if the source data is negative.

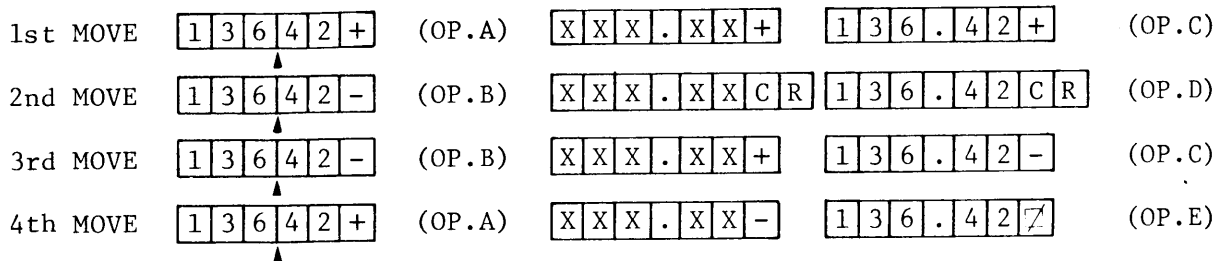
A + in the mask is output as a + if the value of the data being output is positive and as a - if the value of the data is negative. A - in the mask is output as a - if the value of the data being output is negative and replaced with a space if the value of the data is positive.

Consider the following data definitions and the MOVE instruction:

REFERENCE	EDOC	LOCATION	LENGTH	DP	SYT	VALUE OR PICTURE
7 8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27	28 29	30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	
D N A M E	A		3 4			
D O P E R A N D A	F	0	6	2	D	
D O P E R A N D B	F	6	6	2	D	
D O P E R A N D C	F	1 2	7	2	E	X X X . X X +
D O P E R A N D D	F	1 9	8	2	E	X X X . X X C R
D O P E R A N D E	F	2 7	7	2	E	X X X . X X -

REFERENCE	OPERATION	OPERANDS
7 8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C 1 S T M O V E	M O V E	O P E R A N D A , O P E R A N D C
C 2 N D M O V E	M O V E	O P E R A N D B , O P E R A N D D
C 3 R D M O V E	M O V E	O P E R A N D B , O P E R A N D C
C 4 T H M O V E	M O V E	O P E R A N D A , O P E R A N D E



NOTE: Carets indicate implied decimal points. The decimal points are inserted into the destination operand by means of the editing mask.

• Insertion Editing

In this variation of the decimal numeric editing MOVE instruction, an insertion character is moved from the mask to the destination operand despite the fact that the insertion character is not contained in the source data. Insertion characters are suppressed, however, if the previous character in the destination operand is suppressed. (See Zero-Suppression and Replacement Edit.)

Consider the following data definitions and the MOVE instruction:

REFERENCE	MODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
DNAME	A		3 2			
OPERANDA	F	0	6	2	U	
OPERANDB	F	6	4	2	U	
OPERANDC	F	1 0	5	2	U	
OPERANDD	F	1 5	8	2	E	X,XXX.XX
OPERANDE	F	2 3	9	2	E	\$X,XXXBXX

REFERENCE	OPERATION	OPERANDS
C1STMOVE	MOVE	OPERANDA, OPERANDD
C2NDMOVE	MOVE	OPERANDB, OPERANDD
C3RDMOVE	MOVE	OPERANDC, OPERANDE

	Source	Mask	Destination
1st MOVE	1 3 6 7 4 2 (OP.A)	X,XXX.XX	1,367.42 (OP.D)
2nd MOVE	2 6 4 2 (OP.B)	X,XXX.XX	0,026.42 (OP.D)
3rd MOVE	2 8 6 2 6 (OP.C)	\$X,XXXBXX	\$0,286 2 6 (OP.E)

In the first example, a comma and a decimal point are inserted into the destination operand.

In the second example, a comma, a decimal point, and zeros are inserted. (The leading zeros are not suppressed because zero-suppression characters are not used in the mask.)

In the third example, a comma, a zero, a currency symbol, and a space are inserted.

• Zero-Suppression and Replacement Editing

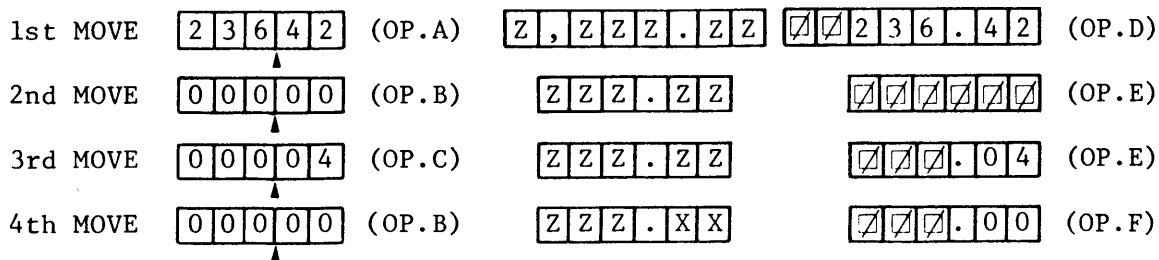
The editing symbol used in this variation of the decimal numeric editing MOVE instruction is Z. The Z may be replaced by either a source data character or a space. Leading zeros are suppressed, as indicated in the examples below. (Inserted characters may be suppressed along with leading zeros -- also illustrated below -- but the decimal point is never suppressed unless the entire operand is suppressed.)

Consider the following data definitions and the MOVE instruction:

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50									
D N A M E											A	3					5																																			
D O P E R A N D A											F	0					5	2	U																																	
D O P E R A N D B											F	5					5	2	U																																	
D O P E R A N D C											F	1					0	5	2	U																																
D O P E R A N D D											F	1					5	8	2	E	Z	,	Z	Z	Z	.	Z	Z																								
D O P E R A N D E											F	2					3	6	2	E	Z	Z	Z	.	Z	Z																										
D O P E R A N D F											F	2					9	6	2	E	Z	Z	Z	.	X	X																										

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50		
C 1 S T M O V E											M O V E					O P E R A N D A , O P E R A N D D																													
C 2 N D M O V E											M O V E					O P E R A N D B , O P E R A N D E																													
C 3 R D M O V E											M O V E					O P E R A N D C , O P E R A N D E																													
C 4 T H M O V E											M O V E					O P E R A N D B , O P E R A N D F																													



In the first example, the leading zero is suppressed. The comma (an insertion character) is also suppressed because the preceding character is suppressed and the comma is no longer necessary.

In the second example, the entire operand (including the inserted decimal point) is suppressed because it consists of nothing but zeros.

In the third example, the leading zeros are suppressed. The decimal point is not suppressed, however, because the entire operand does not consist of zeros. The zero following the decimal point is not considered a leading zero; therefore, it is not suppressed.

In the fourth example, the zero-suppression symbol is not used for the decimal positions; therefore, zeros in the decimal positions are printed.

● Check Protect Editing

The editing symbol used in this editing MOVE is \*. Leading zeros are replaced by asterisks; all other source data characters are preserved. As its name implies, the primary purpose of this variation of the MOVE instruction is to protect the user when printing checks. The check protect symbol \* and the zero suppression symbol Z are not permitted in the same editing mask.

Consider the following data definitions and the MOVE instructions:

REFERENCE	LOC CODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
7 8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27	28 29 30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50		
D N A M E	A		2 9			
D O P E R A N D A	F	0	5	2	U	
D O P E R A N D B	F	5	5	2	U	
D O P E R A N D C	F	1 0	5	2	U	
D O P E R A N D D	F	1 5	8	2	E	*,*,*,*.**,
D O P E R A N D E	F	2 3	6	2	E	***.***

REFERENCE	OPERATION	OPERANDS
7 8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C 1 S T M O V E	M O V E	O P E R A N D A , O P E R A N D D
C 2 N D M O V E	M O V E	O P E R A N D B , O P E R A N D E
C 3 R D M O V E	M O V E	O P E R A N D C , O P E R A N D E

1st MOVE 

2	3	6	4	2
---	---	---	---	---

 (OP.A)    

*	,	*	*	*	.	*	*
---	---	---	---	---	---	---	---

*	*	2	3	6	.	4	2
---	---	---	---	---	---	---	---

 (OP.D)

2nd MOVE 

0	0	0	0	0	0
---	---	---	---	---	---

 (OP.B)    

*	*	*	.	*	*
---	---	---	---	---	---

*	*	*	.	*	*
---	---	---	---	---	---

 (OP.E)

3rd MOVE 

0	0	0	0	4
---	---	---	---	---

 (OP.C)    

*	*	*	.	*	*
---	---	---	---	---	---

*	*	*	.	0	4
---	---	---	---	---	---

 (OP.E)

Note that leading zeros are replaced with asterisks, even when the entire field is composed of zeros.



● Floating Currency Symbol Editing

The editing symbol used in this variation of the decimal numeric editing MOVE instruction is the currency symbol (\$ or £). This variation effectively suppresses zeros, just as the zero suppression and replacement edit does, and inserts the currency symbol as well. If there are leading zeros in the source data, the currency symbol suppresses each in turn (left to right) and replaces the last one. To ensure addition of the currency symbol, the mask should be longer than the source data.

Consider the following data definitions and the MOVE instruction:

REFERENCE	CODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
NAME	A		2 1			
OPERANDA	F	0	4	2	U	
OPERANDB	F	4	3	2	U	
OPERANDC	F	7	3	2	U	
OPERANDD	F	1 0	6	2	E	\$\$\$ . \$\$
OPERANDE	F	1 6	5	2	E	\$\$ . \$\$

REFERENCE	OPERATION	OPERANDS
1ST MOVE	MOVE	OPERANDA , OPERANDE
2ND MOVE	MOVE	OPERANDA , OPERANDD
3RD MOVE	MOVE	OPERANDB , OPERANDD
4TH MOVE	MOVE	OPERANDC , OPERANDE

1st MOVE    0 6 4 2    (OP.A)    \$\$. \$\$    \$ 6 . 4 2    (OP.E)  
                   ↑  
 2nd MOVE    2 6 4 2    (OP.A)    \$\$\$ . \$\$    \$ 2 6 . 4 2    (OP.D)  
                   ↑  
 3rd MOVE    1 4 2    (OP.B)    \$\$\$ . \$\$     \$ 1 . 4 2    (OP.D)  
                   ↑  
 4th MOVE    0 0 0    (OP.C)    \$\$. \$\$        (OP.E)

In the first example, the currency symbol replaces the leading zero.

In the second example, an extra character and currency symbol are added to the mask, so the first currency symbol is inserted and the remaining symbols are replaced by source data on a character-for-character basis.

In the third example, one leading zero is assumed and replaced by a space, and the currency symbol is floated one place as illustrated in the destination operand.

In the fourth example, the entire operand (including the inserted decimal point) is suppressed.

● General Floating Editing Symbol

This variation of decimal numeric editing was designed primarily to be used as an international floating currency symbol technique in which the user specifies the symbol.

The editing symbol in this variation of the decimal numeric editing MOVE instruction is the multiple colon (::). This variation effectively suppresses zeros, just as the zero suppression and floating currency symbols do, and inserts the requested character as well. If there are leading zeros in the source data, the general floating symbol suppresses each in turn (left to right) and replaces the last one with the requested character. To insure the insertion of the requested character, the mask should be one character longer than the source data.

If the colon is expressed singularly (:) it is treated as an insertion character; however, if multiple colons are used consecutively (:::) or are separated by a single insertion character (:,:) they are considered to be general floating symbols. The character which is to be inserted immediately follows the numeric editing mask and is separated from the mask by a comma. The character to be inserted can be any of the NCR-Century character set. If the character is a + or a minus (-), the standard rules of sign editing are applied. If the character is not entered, the general floating symbol is treated as individual insertion characters.

Consider the following data definitions and the MOVE instruction:

REFERENCE	EDC	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
DNAME	A		4	1		
OPERANDA	F	0	4	2	U	
OPERANDB	F	4	7	2	U	
OPERANDC	F	11	6	2	D	
OPERANDD	F	17	9	2	E	::, ::::.XX,+
OPERANDE	F	26	6	2	E	::::.XX,R
OPERANDF	F	32	9	2	E	::, ::::.XX,-

REFERENCE	OPERATION	OPERANDS
C1STMOVE	MOVE	OPERANDA, OPERANDE
C2NDMOVE	MOVE	OPERANDB, OPERANDD
C3RDMOVE	MOVE	OPERANDC, OPERANDF

1st MOVE            4 5 1 2 (A)            : : : . X X            R 4 5 . 1 2 (E)

2nd MOVE            0 5 8 6 9 1 (B)            : : , : : : . X X            [ ] [ ] → 5 8 6 . 9 1 (D)

3rd MOVE            9 7 1 3 6 2 - (C)            : : > : : : . X X            - 9 , 7 1 3 . 6 2 (F)

In the first example, the R is inserted to the left of the significant value in the field.

In the second example, the three leading zeros are suppressed; two by spaces and the third by the right arrow (→). Note also that the comma is overridden.

In the third example, the placement and handling of the minus sign are shown. Had the source field been positive, the sign would have been suppressed.

NOTE

The general floating symbol must not be used in the same editing mask with floating currency symbols, Z's or \*'s. As in other floating symbols, the general floating symbol must never be specified to the right of an X in an editing mask.

• Alphanumeric Editing MOVE

In an alphanumeric editing MOVE, the letter B in the mask is replaced by a space in the destination operand. An alphanumeric MOVE, of course, is left-justified and space-filled to the right. If the data is too long to fit the mask, the software truncates the excessive right-hand characters.

Although the letter B is the most commonly used insertion character in an alphanumeric editing MOVE, other insertion characters (described under INTRODUCTION AND DATA, tab 3, "Data Layout Sheets," Editing Masks) are permitted.

Consider the following data definitions and the MOVE instruction:

REFERENCE	MODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
D N,A,M,E	A	4,0				
D O,P,E,R,A,N,D,A	F	0,9			X	
D O,P,E,R,A,N,D,B	F	9,12	0,0	E	X,B,X,X,X,X,X,X,X,X,X	
D O,P,E,R,A,N,D,C	F	2,1	1,0	0,0	E	X,B,X,X,X,X,X,X,X,X
D O,P,E,R,A,N,D,D	F	3,1	9,0	0,0	E	X,B,X,X,X,X,X,X,X

REFERENCE	OPERATION	OPERANDS
C 1ST MOVE	MOVE	O,P,E,R,A,N,D,A, O,P,E,R,A,N,D,B
C 2ND MOVE	MOVE	O,P,E,R,A,N,D,A, O,P,E,R,A,N,D,C
C 3RD MOVE	MOVE	O,P,E,R,A,N,D,A, O,P,E,R,A,N,D,D

1st MOVE    S P E T E R S O N    X B X X X X X X X X X X    S  P E T E R S O N

2nd MOVE    S P E T E R S O N    X B X X X X X X X X X    S  P E T E R S O N

3rd MOVE    S P E T E R S O N    X B X X X X X X X X    S  P E T E R S O

Conventions

The following table lists the data types that can be moved by the editing variation of the MOVE instruction and the maximum lengths of operands for the different data types:

MOVE INSTRUCTION OPERANDS			
MOVE TYPE TO TYPE		DATA TYPES ARE TREATED AS FOLLOWS IN THE MOVE INSTRUCTION	MAXIMUM LENGTH
B	E	B- Binary Characters	8
P		P- Signed Packed Decimal Characters	10
K		K- Unsigned Packed Decimal Characters	
U		U- Unsigned Decimal Characters	19
Z		Z- Unsigned Decimal Characters	
D		D- Signed Decimal Characters	20
X		X- Alphanumeric Character Set	43
S	S- Alphanumeric Character Set		
	E- Edited		

Refer to INTRODUCTION AND DATA, tab 3, "Data Layout Sheets" for a complete description of all mask characters.

★ ★ ★ ★

## ADD AND SUBTRACT INSTRUCTIONS

### ADD

#### Function

The ADD instruction performs the addition of two numbers that have been previously defined on data layout sheets. These data definitions provide the ADD instruction with the size (number of characters) of each number, the decimal point location, and the type of data (binary, unsigned decimal, signed decimal, etc.).

The ADD instruction has two basic forms: the 2-address form and the 3-address form. This feature offers the programmer a degree of flexibility, since he may elect either to store his result in the same field as the second operand or to store it in a separate field specified by the third operand.

#### Examples

##### • Two-Address ADD Instruction

ADD the contents of the first operand to the contents of the second operand and store the result in the second operand.

X		X		X		X		X																																			
REFERENCE		LOC	LOCATION	LENGTH	DP	TYPE		VALUE OR PICTURE																																			
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
D N A M E		A		12																																							
D O P E R A N D A		F	0	6	2	U																																					
D O P E R A N D B		F	6	6	2	U																																					

X		X		X																																							
REFERENCE		OPERATION		OPERANDS																																							
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
C		A D D		O P E R A N D A , O P E R A N D B																																							

The first operand can be either a literal or a reference tag. The second operand must be a reference tag.

Contents before execution of the ADD instruction:

OPERANDA 111111  
OPERANDB 222222

Contents after execution of the ADD instruction:

OPERANDB 333333  
OPERANDA 111111

(Carets indicate implied decimal points.)



• Three-Address ADD Instruction

Add the contents of the first operand to the contents of the second operand and store the result in a third and separate operand.

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
REFERENCE											MODE	LOCATION						LENGTH			DP		PITY	VALUE OR PICTURE																			
D N A M E											A							1 8																									
D O P E R A N D A											F	0						6			2		U																				
D O P E R A N D B											F	6						6			2		U																				
D R E S U L T											F	1 2						6			2		U																				

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
REFERENCE											OPERATION						OPERANDS																										
C											A D D						O P E R A N D A , O P E R A N D B , R E S U L T																										

Either the first operand or the second operand (but not both) can be a literal; the other must be a reference tag. The third (result) operand must be a reference tag.

Contents before execution of the ADD instruction:

```
OPERANDA 111111
OPERANDB 222222
```

Contents after execution of the ADD instruction:

```
RESULT 333333
OPERANDA 111111
OPERANDB 222222
```

(Caretts indicate implied decimal points.)

## SUBTRACT (SUB)

### Function

The SUB instruction performs the subtraction of two numbers that have been previously defined on data layout sheets. The data definitions provide the SUB instruction with the size (number of characters) of each number, the decimal point location, and the type of data (binary, unsigned decimal, signed decimal, etc.).

The SUB instruction has two basic forms: the 2-address form and the 3-address form. This feature offers the programmer a degree of flexibility since he may elect either to store his result in the same field as the second operand or to store it in a third and separate field.

### Examples

- Two-Address SUB Instruction

Subtract the contents of the first operand from the contents of the second operand and store the result in the second operand.

×	REFERENCE	EDOC	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE																																				
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
	D N A M E											A		1	2																												
	D O P E R A N D A											F		0	6	2	U																										
	D O P E R A N D B											F		6	6	2	U																										

×	REFERENCE	OPERATION	OPERANDS																																								
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
	C											S U B					O P E R A N D A , O P E R A N D B																										

The first operand can be either a literal or a reference tag; the second operand must be a reference tag.

Contents before execution of the SUB instruction:

```
OPERANDB 333333
          ▲
OPERANDA 111111
          ▲
```

Contents after execution of the SUB instruction:

```
OPERANDB 222222
          ▲
OPERANDA 111111
          ▲
```

(Carets indicate implied decimal points.)

• Three-Address SUB Instruction

Subtract the contents of the first operand from the contents of the second operand and store the result in the third operand.

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
REFERENCE											MODE	LOCATION										LENGTH	DP	TYPE	VALUE OR PICTURE																		
DNAME											A											18																					
OPERANDA											F	0										6	2	U																			
OPERANDB											F	6										6	2	U																			
RESULT											F	12										6	2	U																			

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
REFERENCE											OPERATION										OPERANDS																						
C											SUB										OPERANDA, OPERANDB, RESULT																						

Either the first operand or the second operand (but not both) can be a literal; the other must be a reference tag. The third (result) operand must be a reference tag.

Contents before execution of the SUB instruction:

OPERANDB 333333  
 OPERANDA 111111

Contents after execution of the SUB instruction:

RESULT 222222  
 OPERANDA 111111  
 OPERANDB 333333

(Caret indicates implied decimal points.)

CONVENTIONS CONCERNING THE ADD AND SUB INSTRUCTIONS

- The following table lists the data types and the maximum lengths of the operands upon which addition and subtraction operations may be performed. Either type may be taken as the minuend in a subtract operation with the remaining type being taken as the subtrahend.

ADD AND SUBTRACT INSTRUCTION OPERANDS			
ADD OR SUBTRACT TYPE AND TYPE		DATA TYPES ARE TREATED AS FOLLOWS IN THE ADD AND SUB INSTRUCTIONS	MAXIMUM LENGTH
B	B	B- Binary	8
D	D	D- Signed Decimal Characters	20
D	U or Z	U- Unsigned Decimal Characters	19
U or Z	U or Z	Z- Unsigned Decimal Characters	
P*	P*	P- Signed Packed Decimal Characters	10

\* Type P (signed packed decimal data) may be added or subtracted on the NCR Century 200 only. All operands must be of the same type. Both rounding and overflow checking operations (R, C, and L) are available.

- The data being added or subtracted must be numeric (either decimal or binary). All operands must be the same type; i.e., all must be either decimal or binary numbers.
- The composite length of operands cannot be greater than the permitted length of the largest operand. The composite length is determined by aligning the decimal points of the operands and counting the number of character positions used on both sides of the decimal point. For example, consider the following operands:

```

xxxxxxxx.xx
      x.xxx
    
```

Since the first operand is 10 characters long and the second operand has one more character to the right of the decimal point, the composite length of the two operands is 11 characters.

- A result of zero in a signed ADD or SUB operation carries the sign of the second operand (the normal arithmetic result sign of the operation). For example:

	SUB	SUB	ADD	ADD
OPERANDA	-1	+1	-1	+1
OPERANDB	-1	+1	+1	-1
RESULT	-0	+0	+0	-0

5. When subtracting and using all unsigned fields, take care to assure that the result is not negative. If the result is negative and the result field is unsigned, the value in that field is unpredictable. Whenever the result could be negative, the result field should be signed.
6. The execution time of the ADD and SUB instructions depends upon the type of data being used. For example, arithmetic manipulation of signed data requires more processing time than the arithmetic manipulation of unsigned data.

The following list describes all the conditions necessary to execute an ADD or SUB instruction in a minimum amount of time.

- a. Either 2- or 3-address instructions may be used.
- b. All fields must be unsigned decimal.
- c. All fields must contain the same number of decimal positions.
- d. The result field must be at least as long as the longest operand field. The lengths of the operand and the result fields need not be the same.
- e. The difference between the number of integer places in the longest and the shortest fields must be less than 11.
- f. Neither rounding nor overflow checking may be requested. (See "Variations of Arithmetic Instructions" under this tab.)

## MULTIPLY AND DIVIDE INSTRUCTIONS

### MULTIPLY (MULT)

The MULT instruction is used to perform the multiplication of two numbers stored in locations that have been previously defined on data layout sheets. These data definitions provide the MULT instruction with the size (number of characters) of each number, the decimal point location, and the type of data (signed or unsigned decimal).

When either operand of a multiplication operation is zero, the sign of the result field is positive. Otherwise, the normal algebraic laws of signs are observed with the MULT instruction. A signed operand and an unsigned operand are permitted in the same operation; however, if either of the two operands is defined as signed, the result should also be defined as signed.

For fastest program execution, multiply the longer operand by the shorter operand. The result is stored in a third operand.

### Example

X	REFERENCE	CODE	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
	7 8 9 10 11 12 13 14 15 16 17		18 19 20 21 22 23	24 25 26 27	28 29	30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
	DNAME	A		10			
	OPERANDA	F	0	3	1	U	
	OPERANDB	F	3	2	1	U	
	RESULT	F	5	5	2	U	

X	REFERENCE	OPERATION	OPERANDS
	7 8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
	C	MULT	OPERANDA, OPERANDB, RESULT

Either the first operand or the second operand (but not both) can be a literal; the other must be a reference tag. The third (result) operand must be a reference tag.

Contents before execution of the MULT instruction:

OPERANDA 627  
 OPERANDB 91

Contents after execution of the MULT instruction:

RESULT 57057  
 OPERANDA 627  
 OPERANDB 91

(Carets indicate implied decimal points.)

Note that the number of decimal places in the result is carried out to the number of places specified on the data layout sheets. For example, if the field length for RESULT were 6 and the DP entry were 1, the result would be 005705; if the field length were 6 and the DP entry were 3, the result would be 570570.

## DIVIDE (DIV)

### Function

The DIV instruction performs the division of one number by another. The data definitions provide the DIV instruction with the size (number of characters) of each number, the decimal point location, and the type of data (signed or unsigned decimal).

When the result (quotient) of a division operation is zero, the sign of the result field is positive. Otherwise, the normal algebraic laws of signs are observed with the DIV instruction.

Divide the contents of the first operand into the contents of the second operand and store the result in the third operand.

### Example

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50								
×	REFERENCE										×	LOC	×	LENGTH	×	DP	×	TYPE	VALUE OR PICTURE																																
	D N A M E										A			1	5																																				
	D O P E R A N D A										F		0		3		1	U																																	
	D O P E R A N D B										F		3		4		1	U																																	
	D R E S U L T										F		7		4		2	U																																	
	D S A V E R E M A I N										F		1	1		4		U																																	

MULTIPLY AND DIVIDE INSTRUCTIONS

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50						
REFERENCE											OPERATION						OPERANDS																																
C											D I V						O P E R A N D A , O P E R A N D B , O P E R A N D C																																

Either the first operand or the second operand (but not both) can be a literal; the other must be a reference tag. The third (result) operand must be a reference tag.

When whole integers are divided and the rounding feature is not used, any remainder can be accessed and moved to a user-defined area; however, this must be done immediately following the DIV instruction. The remainder, which will never be greater than 19 characters, will be treated as positive integers. To access the remainder, the programmer must use a MOVE instruction with >EXEC.REMAINDER as the first operand and the name of an area as the second operand.

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50						
REFERENCE											OPERATION						OPERANDS																																
C											D I V						O P E R A N D A , O P E R A N D B , R E S U L T																																
C											M O V E						> E X E C . R E M A I N D E R , S A V E R E M A I N																																

>EXEC.REMAINDER is a standard software system tag and, as such, does not need to be defined by the programmer. The programmer only needs to define the area into which he wants the remainder placed (SAVEREMAIN in the example).

REMAINDER is defined by the software as 19 unsigned decimal numeric integers. If the area to receive the remainder is less than 19 characters, only the number of characters specified will be made available.

Contents before execution of the DIV and MOVE instructions:

```
OPERANDA 300
OPERANDB 1000
```

Contents after execution of the DIV and MOVE instructions:

```
RESULT      3.33
OPERANDA    300
OPERANDB    1000
SAVEREMAIN  0100
```

Note that the number of decimal places in the result is carried out to the number of places specified on the data layout sheet. For example, if the field length for RESULT were 5 and the DP entry were 1, the result would be 00033; if the field length were 5 and the DP entry were 3, the result would be 03333.



NOTE

Division by zero causes the divide operation to be bypassed. If an overflow check is requested, the overflow branch will be taken. (See "Variations of Arithmetic Instructions" under this tab.)

CONVENTIONS CONCERNING MULT AND DIV INSTRUCTIONS

The following table lists the types and the maximum lengths of data upon which MULT and DIV operations may be performed. In multiplication, either type may be taken as the multiplicand with the remaining type being taken as the multiplier. In division, either type may be taken as the dividend with the remaining type being taken as the divisor.

MULT AND DIV INSTRUCTION OPERANDS			
MULTIPLY OR DIVIDE TYPE AND TYPE		DATA TYPES ARE TREATED AS FOLLOWS IN THE MULT AND DIV INSTRUCTIONS	MAXIMUM LENGTH
D	D	D- Signed Decimal Characters	20
D	U or Z	U- Unsigned Decimal Characters	19
U or Z	U or Z	Z- Unsigned Decimal Characters	19
P*	P*	P- Signed Packed Decimal Characters	10

\* Type P (signed packed data) may be multiplied on the NCR Century 200 only. All operands must contain data of this same type. Both rounding and overflow checking variations (R, C, and L) are available.

To access a remainder, the MOVE >EXEC.REMAINDER instruction must immediately follow the division. The remainder of a DIV instruction operating upon signed packed decimal data will be unsigned decimal.

\*\*\*

## VARIATIONS OF ARITHMETIC INSTRUCTIONS

### ★ ★ PERFORMING ARITHMETIC OPERATIONS WITH OVERFLOW CHECK

#### Function

There are two variations of the arithmetic instructions that will check an arithmetic result to determine if the result overflows the field assigned to store the result. (This check is only to the left of the decimal number.)

The letter C added to an arithmetic instruction directs the instruction to check for overflow and, if overflow occurs, to branch to the user's routine that handles overflows.

The letter L added to an arithmetic instruction also directs the instruction to check for overflow but, if overflow occurs, to link to the routine that handles overflows.

Examples of these two overflow variations are shown below.

*	REFERENCE	*	OPERATION	*	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24	25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C			A D D C		A , B , Z
C			A D D C		A , B , C , Z
C			A D D L		A , B , Z
C			A D D L		A , B , C , Z
C			S U B C		A , B , Z
C			S U B C		A , B , C , Z
C			S U B L		A , B , Z
C			S U B L		A , B , C , Z
C			M U L T C		A , B , C , Z
C			M U L T L		A , B , C , Z
C			D I V C		A , B , C , Z
C			D I V L		A , B , C , Z

Operands A, B, and C are the reference tags of data fields; operand Z is the reference tag of the user's routine that handles overflows.

Example

The ADDC (Add and Check) instruction follows:

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
REFERENCE											ADDC	LOCATION						LENGTH			DP	TYPE	VALUE OR PICTURE																				
DNAME											A							12																									
DOPERANDA											F	0						6			2	U																					
DOPERANDB											F	6						6			2	U																					

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
REFERENCE											OPERATION						OPERANDS																										
C											ADDC						OPERANDA, OPERANDB, ROUTINEZ																										

The last operand is the reference tag of the routine for handling an overflow situation.

Contents before execution of the ADDC instruction:

```

OPERANDA 511122
OPERANDB 522211
          103333
    
```

Overflow occurs. Control is transferred to the routine represented by ROUTINEZ.

Contents after overflow occurs:

```

OPERANDB 522211
    
```

(Carets indicate implied decimal points.)

NOTE

When an overflow check is not requested but overflow occurs, the result operand will contain the arithmetic result minus the overflow.

PERFORMING ARITHMETIC OPERATIONS AND ROUNDING THE RESULT

Function

This 2-address or 3-address variation of the arithmetic instructions rounds off the result of an arithmetic instruction as specified in the data definition. For example, if 1.55 is to be added to 2.5 and stored in an operand that has only one decimal place, the round variation of the arithmetic instruction rounds the result to the nearest tenth before storing the result in the result operand. So,  $1.55 + 2.50 = 4.05$ , which is then rounded to 4.1.

To indicate the rounding variation, the letter R is added to the arithmetic instruction. Examples follow:

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50									
C											A	D	D	R			A	,	B																																	
C											A	D	D	R			A	,	B	,	C																															
C											S	U	B	R			A	,	B																																	
C											S	U	B	R			A	,	B	,	C																															
C											M	U	L	T	R		A	,	B	,	C																															
C											D	I	V	R		A	,	B	,	C																																

Example

The ADDR (Add and Round) instruction follows:

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50												
D	N	A	M	E							A						1	7																																					
D	O	P	E	R	A	N	D	A			F						0						6		2		U																												
D	O	P	E	R	A	N	D	B			F						6						6		2		U																												
D	R	E	S	U	L	T					F						1	2					5		1		U																												

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50															
C											A	D	D	R			O	P	E	R	A	N	D	A	,	O	P	E	R	A	N	D	B	,	R	E	S	U	L	T																		

Contents after execution of the ADDR instruction:

OPERANDA 111345  
 OPERANDB 222411  
 RESULT 33376

BOTH ROUNDING AND OVERFLOW CHECK

Function

The arithmetic instructions have two variations that will round off an arithmetic result and then check for overflow.

When the letters RC are added to an instruction, the instruction rounds off the result and determines if it overflows the assigned field. If an overflow occurs, a branch is taken to the user's routine that handles overflows.

The letters RL added to an arithmetic instruction will also round off the arithmetic result and check for overflow; however, if overflow occurs, a link is made to the user's routine that handles overflows.

Examples of these two variations are shown below.

×	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		A D D R C	A , B , Z
C		A D D R C	A , B , C , Z
C		A D D R L	A , B , Z
C		A D D R L	A , B , C , Z
C		S U B R C	A , B , Z
C		S U B R C	A , B , C , Z
C		S U B R L	A , B , Z
C		S U B R L	A , B , C , Z
C		M U L T R C	A , B , C , Z
C		M U L T R L	A , B , C , Z
C		D I V R C	A , B , C , Z
C		D I V R L	A , B , C , Z

\*\*\*

## LOG INSTRUCTION

### LOG

#### Function

The LOG instruction allows the programmer to store a limited amount of information in the system log. During one run, the programmer may log up to 20 messages, each no longer than 110 characters in addition to the prefix LOG. The message, stored in an area defined by the programmer, must be in the exact format desired when the log is printed; however, a printer control block should not be included. The log print routine automatically attaches the necessary control block to the message.

Whenever the program attempts to log more than 20 messages, the system log is closed to the program and a message noting this condition is placed in the log. All additional log messages generated by the program in that run are disregarded; all additional messages generated by the software are logged as normal.

The programmer must define an area to contain a log message. The first three characters in this area must always be LOG. The programmer may define the message as constants on the data layout sheet, or he may define part of the message and move additional data generated elsewhere by the program into the reserved log message area. This technique is illustrated below.

X		X		X		X		X																																			
REFERENCE							CODE	LOCATION			LENGTH	DP	TYPE	VALUE OR PICTURE																													
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
DLOG MESSAGE							A				1	9																															
DPREFIX							F	0			3		X	LOG																													
DMESSAGE							F	3			1	1	X	TOTAL ITEMS																													
DTOTALS							F	1			4	5	U																														

X		X		X																																							
REFERENCE							OPERATION	OPERANDS																																			
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
C							L	O			G	L, O, G, M, E, S, S, A, G, E																															

The 3-character prefix, LOG, must appear at the beginning of every user log message. In this example, the partial message TOTAL ITEMS is defined as constants by the programmer; the remaining portion of the message, the actual total, is to be moved into the reserved log message area by the programmer.

### Conventions

No more than 20 messages may be logged by the program.

Messages must be 110 characters or less (in addition to the prefix, LOG); the messages should be in the exact format desired when the log is printed.

No printer control block should appear in the message.

## CNSOUT, CNSIN, AND CNSINA INSTRUCTIONS

The CNSOUT, CNSIN, and CNSINA instructions provide the programmer with the means of communicating with the operator during program execution. At any desired points in his program, the programmer may address an informative message (the operand of a CNSOUT instruction) or a request (the operand of a CNSIN or CNSINA instruction) to the operator. CNSOUT does not require a reply from the operator. CNSIN and CNSINA instructions do require replies and both instructions stop the program until the operator's response is completed.

CNSIN requires input in hexadecimal; CNSINA accepts input in either alpha or hexadecimal. CNSIN is intended for use with systems that do not have an I/O Writer, CNSINA for systems equipped with an I/O Writer.

### CNSOUT

#### Function

The CNSOUT instruction is the means by which the programmer relays specific information or directions to the operator during program processing. Since CNSOUT messages are advisory or explanatory in content, no operator reply is required. CNSOUT messages may assume either of two formats: (1) a hard copy printout on the I/O Writer or printer, or (2) a console message in the INFORMATION DISPLAY lights. Output to these three peripherals is based on preferential availability as listed below:

1. the I/O Writer
2. the printer
3. the INFORMATION DISPLAY lights on the console.

If the system includes an I/O Writer, the I/O Writer is always used to print the CNSOUT message. No console display occurs and processing is not interrupted.

If the I/O Writer is not present, the CNSOUT message is output to the printer if it is available (no files open). No console display occurs and processing is not interrupted.

If the printer is busy, the processor enters a wait state and the first two characters of the programmer's message (wait code) are displayed in the INFORMATION DISPLAY lights on the console. The operator must then access the remainder of the CNSOUT message in the console INFORMATION DISPLAY lights.



- Hard Copy Message

The hard copy message is composed of alphanumeric characters. It must be brief enough to be output on one line of the I/O Writer or printer. The maximum number of characters output in one line by the I/O Writer is 60 characters. Since the limitations of the I/O Writer are more strict than the printer, the programmer should adhere to the I/O Writer limitations. If an I/O Writer message is longer than the maximum 60 characters, the extra characters are dropped. The two characters following the hard copy message must be the hexadecimal characters FF, signifying end-of-message; however, these two characters are not printed, and are not included as a part of the 60-character maximum. The end-of-message characters (FF) are specified separately in a hexadecimal field. Format of a typical hard copy CNSOUT message is shown below:

REMOVE TAPE FROM HANDLER 01

- Console Messages

If the system has no I/O Writer and the printer is not available, the CNSOUT message is output through the console (in hexadecimal form, two hexadecimal characters at a time), on the eight INFORMATION DISPLAY lights. The bit configuration for hexadecimal characters is listed below:

<u>Decimal</u>	<u>Hexadecimal</u>	<u>Bit Configuration</u>	<u>Decimal</u>	<u>Hexadecimal</u>	<u>Bit Configuration</u>
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

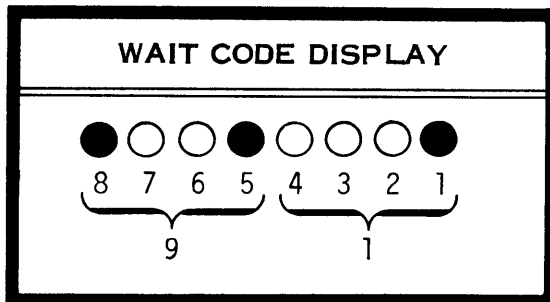
For a detailed explanation of hexadecimal data, see NEAT/3 REFERENCE MANUAL, INTRODUCTION AND DATA, tab 2, "Data Concepts."

The first two characters displayed in the INFORMATION DISPLAY lights are the wait code. The operator then accesses any additional console output, two characters at a time. The final two characters displayed must be the hexadecimal characters FF, signifying end-of-message. To enable the operator to correctly interpret the display, the programmer should explain the display in the run book. For example, assume that the light display for the message REMOVE TAPE FROM HANDLER 01 is 9101FF.

- Wait Code

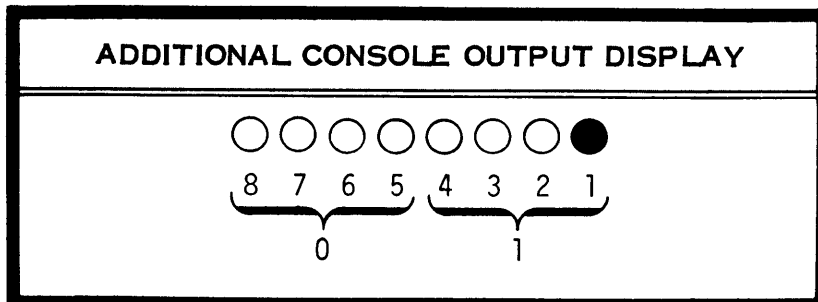
The first two characters of this message (91) tell the operator that this is a programmer message rather than a system message. The 8-light (high-order bit) of the wait code must be on for all user messages. This restriction limits the first character to the hexadecimal range 8 to F (any smaller hexadecimal character would turn the 8-light off). If the first character is a hexadecimal F, the second character is restricted to the hexadecimal range 0 to D (the hexadecimal combination FE is reserved, and FF indicates end-of-message). The wait code followed by FF could be the complete message, since the wait code may simply be a key to the programmer's message in the run book.

The first two characters of the light display 9101FF are shown below:



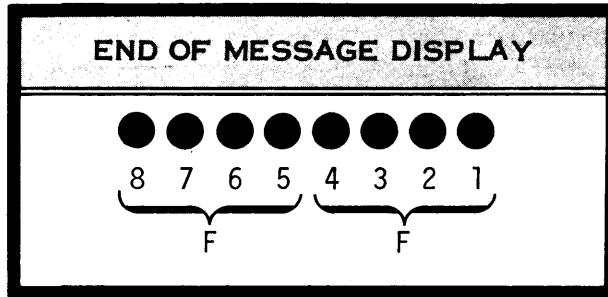
- Additional Console Output

Although the programmer usually limits his message to two hexadecimal characters (wait code) for display, he may use as many 2-character hexadecimal combinations as necessary to output his information. These additional characters are accessed and displayed two at a time in the INFORMATION DISPLAY lights when the ACT switch is pressed. The wait code restrictions on the high-order bit (8-light) do not apply to the remainder of the console output. Therefore the hexadecimal range 0 to F may be used for all the remaining characters of the displayed message, with the exception of the hexadecimal combinations FE and FF. For example, the second two characters of the light display 9101FF is shown below:



• FF

The last two hexadecimal characters of the console display must be FF (all lights on) to signify end-of-message to the operator. This example is shown below:



EXAMPLE

The programmer must define his message to include the format for both the console (wait code display) and hard copy messages. When defining the message area, the entire message (including both message formats) cannot be longer than 77 characters. Both the console and hard copy messages must end with a hexadecimal FF, signifying end-of-message. The message area definition for the previously illustrated hard copy and console messages is shown below:

REFERENCE	POC	LOCATION	LENGTH	DP	TYPE	VALUE OR PICTURE
D M E S S A G E 1	A	3 1				
D	F	0 3			H	9 1 0 1 F F
D P R I N T	F	3 2 7			X	R E M O V E T A P E F R O M H A N D L E R 0 1
D	F	3 0 1			H	F F

NOTE

A hexadecimal field cannot be accessed by the program; therefore, reference entries are not made for hexadecimal fields. If it is necessary for the programmer to access a hexadecimal field, he must also define it as an accessible data type (such as X or B type) and assign a reference entry. See NEAT/3 REFERENCE MANUAL, INTRODUCTION AND DATA, tab 2, "Data Concepts."

The CNSOUT instruction to access this message is shown in the following coding:

*	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		C, N, S, O, U, T	M, E, S, S, A, G, E, 1

Conventions

Every message referenced by a CNSOUT instruction must be defined in two formats: the console message in hexadecimal characters to be displayed in the INFORMATION DISPLAY lights, and the hard copy message to be output to the I/O writer or printer.

The last two characters of both the console and hard copy messages must be the hexadecimal characters FF, signifying end-of-message.

A hexadecimal field cannot be accessed by the program and must therefore be included as a field of a previous definition which is accessible to the program.

The maximum number of characters output in one line by the I/O Writer is 60 characters. Since the limitations of the I/O Writer are more strict than the printer, the programmer should adhere to the I/O Writer's limitations to avoid losing the final part of the message when output is to the I/O Writer.

CNSIN

Function

The CNSIN instruction is another means by which the programmer relays information to the operator during program processing. Since CNSIN requires an operator response, these messages are often in the form of a question. These messages may assume either of two formats: (1) a hard copy printout which is output on the I/O Writer or printer, or (2) a console message in the INFORMATION DISPLAY lights. Output to these three peripherals is based on preferential availability as listed below:

1. the I/O Writer
2. the printer
3. the INFORMATION DISPLAY lights on the console.

If the system includes an I/O Writer, the I/O Writer is always used to print the CNSIN message. At the end-of-message (FF), the I/O Writer is selected for input and the operator's reply is entered through the I/O Writer. No console display is made, and processing is not interrupted.

If the I/O Writer is not present, the CNSIN message is output to the printer if it is available (no files open). The processor enters a wait state and

hexadecimal FF is displayed in the console INFORMATION DISPLAY lights. The operator's response is entered through the console switches before processing is resumed.

If the printer is busy, the processor enters a wait state and the first two characters of the programmer's message (wait code) are displayed in the INFORMATION DISPLAY lights on the console. The operator must then access the remainder of the CNSIN message in the console INFORMATION DISPLAY lights. The operator's response must be entered through the console switches before processing is resumed.

- Hard Copy Messages

The hard copy message is composed of alphanumeric characters. It must be brief enough, including the operator's response, to be output on one line of the I/O Writer or printer. The maximum number of characters output in one line by the I/O Writer is 60 characters. Since the limitations of the I/O Writer are more strict than the printer, the programmer should adhere to the I/O Writer limitations. If an I/O Writer message is longer than the maximum 60 characters, the extra characters are dropped. The two characters following the hard copy message must be the hexadecimal characters FF, signifying end-of-message; however, these two characters are not printed, and are not included as a part of the 60-character maximum. The end-of-message characters (FF) are specified separately in a hexadecimal field. Format of a typical hard copy CNSIN message is shown below:

ENTER 01 IF PUNCHED CARD FILE; 02 IF NOT ↑

NOTE

The arrow in the above illustration is not printed; it indicates the location of the I/O Writer after printing the CNSIN messages. In the PRINT field of the message area definition, a space was provided following the word NOT to separate the hard copy message from the operator's response. See the next illustration and the sample data definition on the next page.

- Console Messages

If neither the I/O Writer nor the printer is available, the CNSIN message is output in the INFORMATION DISPLAY lights in hexadecimal form, two characters at a time, as described in the CNSOUT instruction. The operator enters his response through the console DATA ENTER switches.

Operator's Response

If the CNSIN message is output on the I/O Writer, the operator uses this peripheral for his response. For example, if a card file is being used, the correct response to the message in the previous illustration would be 01. The

operator enters the characters 0 and 1, followed by the nonprinting BELL character which signals the end of message (line) input. Consider the following illustration:

ENTER 01 IF PUNCHED CARD FILE; 02 IF NOT 01β

NOTE

For purposes of illustration, the nonprinting BELL character is represented by the symbol β in the above illustration.

CNSIN assumes all input through the I/O Writer to be hexadecimal. Therefore, the characters 0 and 1 entered by the operator are interpreted as the hexadecimal characters 01.

Since a hexadecimal field cannot be accessed by the program, the input area must be defined as an accessible field type, such as binary. Binary provides a particularly convenient conversion from hexadecimal, since hexadecimal input 00 through 09 carries the same numeric value and bit configuration as binary 0 through 9. Beyond the digit nine, however, binary and hexadecimal characters assume different values.

If the CNSIN message is output to the printer, the processor enters a wait state and the hexadecimal characters FF (all lights on) are displayed in the INFORMATION DISPLAY console lights. The operator enters his response through the console DATA ENTER switches, two characters at a time. See OPERATORS INFORMATION MANUAL, HARDWARE, Consoles tab.

EXAMPLE

The programmer must define his message to include the format for both the console (wait code display) and hard copy messages. When defining the message area, the entire message (including both message formats) cannot be longer than 77 characters. Both the console and hard copy messages must end with a hexadecimal FF, signifying end-of-message. In addition, an input area for the operator's response must be defined. A typical message area definition for the previously illustrated CNSIN message is shown below:

M	REFERENCE	M MOD	LOCATION	LENGTH	M DP	M P P T Y P E	VALUE OR PICTURE	M *	COMMENTS
7	M E S S A G E 2	A		4 4					
8		F	0	2		H B 4 F F			
9	P R I N T	F	2	4 1		X E N T E R , 0 1 , I F , P U N C H E D , C A R D , F I L E , , 0 2 , I F , N O T ,			
10		F	4 3	1		H F F			
11									
12	I N A R E A 1	A		2		B			



The CNSIN instruction permits the programmer to include options in his program. The operator's response can be compared to a predetermined value and the program can branch to special routines or continue through the general program.

## CNSINA

### Function

The CNSINA instruction communicates a request to the operator and accepts his reply during program processing. Operator input is entered in alpha-numeric characters either directly through the I/O Writer or as hexadecimal representation of alphanumeric characters through the console. The CNSINA message may be output in either of two forms: (1) a hard copy that is output on the I/O Writer or printer, or (2) a console message in the INFORMATION DISPLAY lights. The selection of output medium is determined by preferential availability as listed below:

1. the I/O Writer
2. the printer
3. the INFORMATION DISPLAY lights on the console.

If the system includes an I/O Writer, the I/O Writer is always used to print the CNSINA message. At the end of the message, the I/O Writer is selected for input and the operator's reply should be entered through the I/O Writer. No console display is made, and processing is not interrupted.

If the I/O Writer is not present in the system, the CNSINA message is output to the printer if it is available (no files open). The processor enters a wait state and hexadecimal FF is displayed in the console INFORMATION DISPLAY lights. The operator's response must be entered through the console switches before processing is resumed.

If the printer is busy, the processor enters a wait state and the first two characters of the programmer's message (wait code) are displayed in the INFORMATION DISPLAY lights on the console. The operator must then access the remainder of the CNSINA message in the console INFORMATION DISPLAY lights. The operator's response must be entered through the console switches before processing is resumed.

### ● Hard Copy Messages

The hard copy message is composed of alphanumeric characters. It must be brief enough, including the operator's response, to be output on one line of the I/O Writer or printer. The maximum number of characters output in one line by the I/O Writer is 60 characters. Since this limitation is more strict for the I/O Writer than for the printer, the programmer should adhere to the I/O Writer limitation. If an I/O Writer message is longer than the maximum 60 characters, the extra characters are dropped. The two characters following the hard copy message must be the hexadecimal characters FF, signifying end-of-message; however, these two characters are neither printed nor included as a part of the 60-character maximum. The end-of-message characters (FF) are specified separately in a hexadecimal field. Format of a typical hard copy CNSINA message is shown below:



ENTER YES IF APPROPRIATE ↑

NOTE

The arrow in this illustration is not printed; it indicates the location of the I/O Writer after printing the CNSINA message. In the PRINT field of the message area definition, a space is provided following the word APPROPRIATE to separate the hard copy message from the operator's response. See the next illustration and sample data definition.

● Console Messages

If neither the I/O Writer nor the printer is available, the CNSINA message is output in the INFORMATION DISPLAY lights in hexadecimal, two characters at a time, as described in the CNSOUT instruction.

The first two characters displayed in the INFORMATION DISPLAY lights are the wait code. The operator then accesses any additional console output, two characters at a time. The final two characters displayed must be the hexadecimal characters FF, signifying end-of-message. To enable the operator to correctly interpret the display, the programmer should explain the display in the run book. For example assume the light display for the message ENTER YES IF APPROPRIATE is 84 FF.

● Wait Code

The first two characters of this message (84) tell the operator that this is a programmer message rather than a system message. The 8-light (high order bit) of the wait code must be on for all user messages.

The wait code followed by FF represents the complete message since it is actually the key to the programmer's message in the run book.

Operator's Response

If the CNSINA message is output on the I/O Writer, the operator uses this peripheral for his response. For example, if the indicated response to the above request is YES, the operator enters the characters Y, E, and S, followed by the nonprinting BELL character which signals the end of message input. Consider the following illustration:

ENTER YES IF APPROPRIATE YES β

NOTE

For purposes of illustration, the nonprinting BELL character is represented by the symbol β in the above illustration.

Since CNSINA accepts input in alphanumeric, the operator's reply is interpreted exactly as entered on the I/O Writer.

If the CNSINA message is output to the printer, the processor enters a wait state and the hexadecimal characters FF (all lights on) are displayed in the INFORMATION DISPLAY console lights. The operator enters his response through the console DATA ENTER switches, two characters at a time. See OPERATOR'S INFORMATION MANUAL, HARDWARE, consoles tab.

EXAMPLE

The programmer must define his message to include the format for both the console (wait code display) and hard copy messages. The area for alphanumeric input should not exceed 60 characters. Both the console and hard copy messages must end with a hexadecimal FF, signifying end-of-message. In addition, an input area for the operator's response must be defined. A typical message area definition for the previously illustrated CNSINA message is shown below:

X	REFERENCE	X	CODE	LOCATION	X	LENGTH	X	DP	X	T	X	Y	X	P	X	E	X	*																																							
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55									
D	M E S S A G E 4		A			2 8																																																			
D			F		0	2					H	8	4	F	F																																										
D	P R I N T		F		2	2 5					X	E	N	T	E	R																																									
D					2 7	1					H	F	F																																												
D																																																									
D	I N A R E A 3		A			5					X																																														
D	I N P U T		F		0	4					X																																														
D			F		4	1					H	F	F																																												

Following the text of the hard copy message, the programmer should include a space (Ø) to separate his message from the operator's response.

The programmer should allow space on the printline following his message for the operator's response.

The length of the input area (INAREA 3) must be defined to accept the operator's response, plus one additional character to accommodate a hexadecimal FF which is moved into the input area by the software.

NOTE

A hexadecimal field cannot be accessed by the program; therefore, reference entries are not made for hexadecimal fields. If it is necessary for the programmer to access a hexadecimal field, he must also define it as an accessible data type (such as X or B type) and assign a reference entry. See NEAT/3 REFERENCE MANUAL, INTRODUCTION AND DATA, tab 2, "Data Concepts."

The CNSINA instruction to access this message is shown in the following coding:

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		C N S I N A	M E S S A G E 4 , I N A R E A 3

MESSAGE 4 is the reference name of the area in which both console (wait code display) and hard copy message are stored.

INAREA 3 is the reference name of the input area allocated for the operator's reply.

Conventions

Every message referenced by a CNSINA instruction must be defined in two formats: the console message in hexadecimal characters to be displayed in the INFORMATION DISPLAY lights, and the hard copy message to be output to the I/O Writer or printer.

The last two characters of both the console and hard copy messages must be the hexadecimal characters FF, signifying the end of the message.

A hexadecimal field cannot be accessed by the program and must therefore be included as a field of a previous definition which is accessible to the program.

The maximum number of characters output in one line by the I/O Writer is 60 characters. Since the limitation of the I/O Writer is more strict than the printer, the programmer should adhere to the I/O Writer's limitation to avoid losing the final part of the message when output is to the I/O Writer.

The CNSINA instruction requires the operator to respond to the message by inputting information through the console or on the I/O Writer.

## COND AND XPAND INSTRUCTIONS

### COND AND XPAND

#### Function

The COND (condense) and XPAND (expand) instructions, which pack and unpack decimal data in areas and large fields or groups of fields, are used to conserve space on an external storage device.

Any even number of characters between 2 and 1024 may be condensed into an area that is between 1 and 512 characters in length. The field receiving the condensed data must be exactly one-half as long as the source field.

Any number of condensed characters between 1 and 512 may be expanded into an area that is between 2 and 1024 characters in length. The field receiving the expanded data must be exactly twice the length of the source field.

Individual fields consisting of 20 characters or less may be packed by the MOVE instruction. However, using COND and XPAND (for a series of fields treated as a large area) instead of using several MOVE instructions saves coding in memory. For further information concerning the conversion of data with the MOVE instruction see INSTRUCTIONS, tab 2, "MOVE Instructions."

The only types of data that should be condensed and expanded with the COND and XPAND instructions are decimal data (U and D type) and certain symbols shown in the following chart. However, since the COND instruction will condense any area without regard to data definitions, the programmer is responsible for assuring the data type of the fields condensed or expanded.

ACCEPTABLE CHARACTERS			
0 - 00110000	4 - 00110100	8 - 00111000	, - 00101100
1 - 00110001	5 - 00110101	9 - 00111001	- - 00101101
2 - 00110010	6 - 00110110	* - 00101010	. - 00101110
3 - 00110011	7 - 00110111	+ - 00101011	/ - 00101111

If an area containing a data type other than U or D is condensed and then expanded, the result will not be the same as the original area. The result is, however, predictable. Any condensed character whose least significant four bits equal binary 0 through 9 is expanded as the 8-bit characters 0 through 9; any condensed character whose least significant four bits equal binary 10 through 15 is expanded to the 8-bit characters \* through /. For example, the character A (01000001), condensed as binary 1 (0001), is expanded to the 8-bit character 1 (00110001). (Consult the NCR Century Code Chart under INTRODUCTION AND DATA, tab 2, "Data Concepts.")

COND and XPAND help conserve space on external storage media when working with records that contain large amounts of fixed-length decimal data. In defining the record, the programmer should group in one portion of the record as much of the decimal data as is practical. All alpha data and ordinary binary data should be placed in another portion of the record.

The decimal portion of the record may be condensed as it is placed from a work-area into the output buffer by the COND instruction; it may be expanded again on input as it is placed into a workarea from the input buffer by the XPAND instruction.

### CONDENSING THE RECORD

X	REFERENCE	X	CODE	LOCATION	X	LENGTH	X	DP	X	TYPE	VALUE OR PICTURE
	7 8 9 10 11 12 13 14 15 16 17		18	19 20 21 22 23		24 25 26 27		28 29		30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
	D BUFFERDEF		R			1 5 0				X	
	D ALPHABUFER		F		0	5 0				X	
	D NUMRICBUFR		F		5 0	1 0 0				X	

X	REFERENCE	X	CODE	LOCATION	X	LENGTH	X	DP	X	TYPE	VALUE OR PICTURE
	7 8 9 10 11 12 13 14 15 16 17		18	19 20 21 22 23		24 25 26 27		28 29		30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
	D RECORDAREA		A			2 5 0				X	
	D ALPHADATA		F		0	5 0				X	
	D ALPHAFLD1		F		0	1 0				X	
	D NUMRICAREA		F		5 0	2 0 0				X	
	D NUMRICFLD1		F		5 0	1 0				U	
	D NUMRICFLD2		F		6 0	0 5 0 2				D	

CONSTRUCT THE RECORD IN AN AREA,

MOVE ALPHADATA, ALPHABUFER      PLACE THE ALPHA PORTION IN THE BUFFER,

COND NUMRICAREA, NUMRICBUFR      COND THE NUMERIC PORTION IN THE BUFFER,

PUT FILENAME      OUTPUT THE RECORD

COND and XPAND may not be used with variable-length data such as variable-length tables; however, they may be used with the fixed-length decimal portion of a variable-length record. Under these conditions, the programmer must be certain not to condense such binary fields as the VLI or TLI.

RECORD					
V L I	Fixed-Length Non-Decimal Data	Fixed-Length Decimal Table	Fixed-Length Decimal Data	T L I	Variable-Length Table

The fixed-length table and fixed-length decimal data portions of the record may be condensed and expanded.

The binary VLI and TLI, the fixed-length non-decimal data, and the variable-length table portions of the record may not be condensed and expanded.

Example

	REFERENCE	OPERATION	OPERANDS	
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	51 52 53 54 55
C		COND	NUMRICAREA, NUMRICBUFR	
C		XPAND	NUMRICBUFR, NUMRICAREA	

In the first example, the COND instruction condenses the data in NUMRICAREA into an area half the size called NUMRICBUFR.

In the second example, the XPAND instruction expands the data in NUMRICBUFR into an area twice the size called NUMRICAREA.

Conventions

Only U and D type data should be condensed; the destination buffer or work area should be defined as X type, to accommodate both U and D type fields. Only previously condensed X type data should be expanded; the destination buffer or work area should be defined as X type, to accommodate both U and D type fields.

The destination field for COND must be half the size of the source field.

The destination field for XPAND must be twice the size of the source field.

Any even number of characters between 2 and 1024 may be condensed into an area that is between 1 and 512 characters.

Any number of condensed characters that is between 1 and 512 may be expanded into an area that is between 2 and 1024 characters in length.

## FINISH INSTRUCTION

### FINISH

#### Function

The programmer places the FINISH instruction at the logical end of the program. FINISH closes the files that have not been closed and calls Monitor back into memory.

#### Example

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		F I N I S H	

#### Conventions

Every program must have a FINISH instruction.

The FINISH instruction must be placed at the logical end of the program.



## SPREAD INSTRUCTION

### FUNCTION

The SPREAD instruction enables the programmer to spread a constant throughout an area or a field.

There are two formats for this instruction: the first is used when the constant is represented as a literal; the second is used when the constant is assigned a reference. Using a reference to spread a constant is more flexible than using a literal because it allows the programmer to specify new constants during the program.

### FORMAT FOR LITERAL

When a constant is represented as a literal, it can be spread by using the format shown below.

X	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		S P R E A D	' L ' , O P E R A N D B

The first operand is the literal, which must be a single alphanumeric (X-type) character. The second operand, the destination operand, is the reference of the area or field to be filled. This area or field must always be defined as X-type; it can have a maximum length of 65,535.

Example

In the example below, the SPREAD instruction directs the software to fill WORKAREAL with zeroes. Notice that the data definition for WORKAREAL defines the area as X-type data, as required.

X	REFERENCE	X C O D E	LOCATION	X L E N G T H	X D P	X T Y P E	VALUE OR PICTURE
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24 25 26 27	28 29	30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
D	W, O, R, K, A, R, E, A, L	A		3, 0		X	

X	REFERENCE	X O P E R A T I O N	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		S P R E A D	' 0 ' , W O R K A R E A L

NOTE

Although the MOVE instruction can be used to zero-fill an area, it can only move a maximum of 20 characters.

Conventions

The literal must be a single alphanumeric (X-type) character.

The destination operand is the reference of the area or field to be filled. This area or field must be defined as alphanumeric with a length no greater than 65,535.

FORMAT FOR REFERENCE

To use a reference to spread an alphanumeric constant, the constant must be properly defined and assigned a reference to be used as the first operand in the instruction.

In the data definitions, this reference must define a 4-character, X-type area or field that contains a constant. The constant must always be written as four identical characters. Consider the coding and data definitions below.

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
D O P E R A N D A											A												4		X	0,0,0,0																	
D																																											
D W O R K A R E A 1											A												3,0		X																		

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
C											S P R E A D											O P E R A N D A , W O R K A R E A 1																					

In the example, the instruction directs the software to spread the constant (0000) referenced by OPERANDA throughout WORKAREAL. WORKAREAL, the destination operand, must always be defined as alphanumeric with a length no greater than 65,535.

A reference can also be used to spread constants that are not alphanumeric. When this function is desired, the first operand in the instruction must be the reference of a 4-character, alphanumeric (X-type) area. In the data definitions, this data must be defined twice: first as a 4-character, alphanumeric area and second as a 4-character field of the desired data type. The field definition contains the constant, which must be written as four identical characters (or eight identical characters, if the data type is hexadecimal or packed).

Consider the following data definitions and coding.

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
D O P E R A N D A											A											4	X																				
D											F	0										4	H	F F F F F F F F																			
D O P E R A N D B											A											5,0	X																				

7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
C											S P R E A D						O P E R A N D A , O P E R A N D B																										

In the example, the SPREAD instruction directs the software to spread hexadecimal FF's throughout OPERANDB. OPERANDB, the destination operand, must always be defined as alphanumeric with a length no greater than 65,535.

Conventions

When a reference is used to spread a constant, the first operand must be the reference associated with the constant; the second operand must be the reference of the area or field to be filled.

If an alphanumeric constant is spread, the reference must be associated with a 4-character, X-type constant. The constant must be written as four identical characters.

If a constant other than X-type is spread, the reference in the instruction must be associated with a 4-character, alphanumeric (X-type) area. This data must then be redefined as a 4-character field of the desired data type, and the constant must be written as four identical characters (or eight identical characters, if the data type is hexadecimal or packed).

The destination operand must always be defined as an alphanumeric (X-type) area or field. It can have a maximum length of 65,535.

## DISC OFF INSTRUCTION

### DISC OFF (DSCOFF)

#### Function

The disc off instruction is used to inform the operator to remove a disc pack. This instruction causes a message to be displayed on the I/O writer or the printer, depending on which is available. The displayed message informs the operator to remove the disc pack named in the operand of the instruction.

The disc off instruction can be coded in either of two ways: (1) by using a literal operand to specify the symbolic unit designator (SUD) of the disc pack to be removed, or (2) by associating a reference with the SUD and using the reference as the operand. The second method of coding is more flexible because it allows the programmer to change the SUD during the program.

#### Example of a Literal

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		D.S.C.Ø.F.F.	'D.O.I.'

#### Example of a Reference

When a reference is used as the operand of a disc off instruction, it must first be defined as a 3-character, alphanumeric (X-type) area. The symbolic unit designator must be specified in columns 31-33 of the data layout sheet.

	REFERENCE	E D O C	LOCATION	L E N G T H	D P	T Y P E	VALUE OR PICTURE
7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 23	24 25 26 27	28 29	30	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
	D,U,N,I,T,I	A		3		X	D,O,I

	REFERENCE	OPERATION	OPERANDS
7	8 9 10 11 12 13 14 15 16 17	18 19 20 21 22 23	24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
C		D.S.C.Ø.F.F.	U.N.I.T.I

### Conventions

The programmer must ensure that all opened files are closed before he executes the disc off instruction.