Burroughs

B5500

Information
Processing Systems

STREAM PROCEDURE
REFERENCE MANUAL

Burroughs

B 5500

INFORMATION PROCESSING SYSTEM

STREAM PROCEDURE

REFERENCE MANUAL

Equipment and Systems Marketing Division

Sales Technical Services

Systems Documentation

Burroughs Corporation

Detroit, Michigan 48232

In Canada: Burroughs Business Machines Ltd., Toronto, Ontario

PREFACE

One of the programing languages utilized by the Burroughs B 5500 Information Processing System is Extended ALGOL 60. This language has been patterned after familiar programing concepts and fitted into the structure of ALGOL 60. ALGOL 60 was designed to describe computational processes and is an excellent tool for this purpose. However, the formulation of this language was restricted to areas which are machine independent. Implementation of machine-dependent elements was recognized to be the responsibility of the individual computer manufacturer. For example, ALGOL 60 alone is incomplete when a computer is to be used for the execution of computational processes since the means of communicating data to and from a particular computer are not provided.

On the other hand, Extended ALGOL 60 provides the B 5500 programmer with complete input/output facilities, STREAM PROCEDURE statements which allow the use of B 5500 character mode functions, the ability to perform symbolic debugging, plus other useful miscellaneous facilities including the ability to perform partial-word arithmetic and double precision arithmetic. ALGOL 60, together with these Burroughs extensions is referred to in Burroughs literature as Extended ALGOL.

This publication completely describes the STREAM PROCEDURE portion of Extended ALGOL. It is assumed that the reader is familiar with ALGOL 60 and the B 5500 Information Processing System.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

INTRODUCTION

The STREAM PROCEDURE is a programing aid which has been added to ALGOL to facilitate the use of the character mode on the B 5500. The STREAM PROCEDURE can be considered a special-purpose form of a standard ALGOL procedure and enables the programmer to consider operations upon strings of characters as well as word operations.

Some of the problems to which the STREAM PROCEDURE can be applied are those involving complex editing of information on input/output operations, packing and unpacking of data for more effective information storage, and scanning operations for comparison of data. These are but a few of the many applications in which the STREAM PROCEDURE can be of significant value to the programmer.

The method for using a STREAM PROCEDURE is generally the same as the method defined for a standard ALGOL procedure. Each STREAM PROCEDURE must be defined by a declaration and activated by a standard procedure statement. There are, however, several differences between a standard ALGOL procedure and a STREAM PROCEDURE. The major difference occurs in the statement forms that may be used for each type of procedure declaration.

The standard ALGOL statements operate in the word mode, and the STREAM PROCEDURE statements function in the character mode. Since the statements which are defined for each type of procedure operate in two different modes, they cannot be used interchangeably.

In other words, a standard ALGOL statement cannot be used within a STREAM PROCEDURE declaration, and a stream statement may not be used outside of a STREAM PROCEDURE declaration.

The discussion which follows will detail the various aspects of the language constructs which have been defined for the STREAM PROCEDURE language.

It is convenient to approach the description of the programing language by examining four main areas of discussion: general concepts, the stream declaration, the stream statements, and the STREAM PROCEDURE applications. The survey of general concepts will cover the basic ideas concerning streams of information, a brief outline of the STREAM PROCEDURE language, address index control, and a short description of STREAM PROCEDURE operations. The discussion pertaining to the stream declaration describes the detailed syntax for the STREAM PROCEDURE declaration and the corresponding explanations and rules for using the various constructs defined by the syntax. The stream statements discussion will detail the syntax and the characteristics for each statement which has been defined for the stream language. The final area, STREAM PROCEDURE applications, will describe the methods for building a STREAM PROCEDURE, and provide several examples of STREAM PROCEDURES.

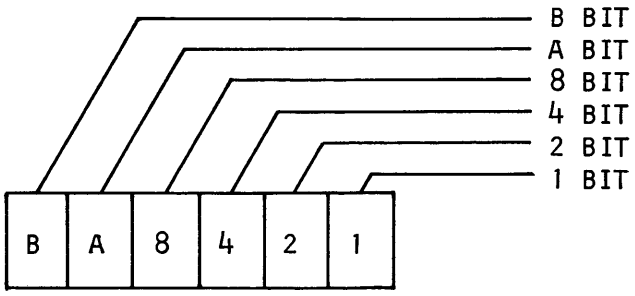
SECTION 1

GENERAL CONCEPTS

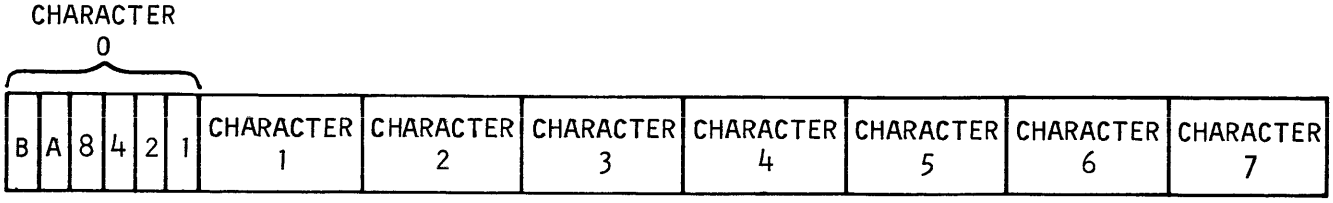
INFORMATION STREAMS.

1-1. Several methods can be used to supply data to a program. The data may be in two forms, edited or non-edited. Edited data is supplied either as the result of a READ statement, or as interim results. Non-edited data may be supplied in the form of a record image in a buffer area. In either case, the data should be viewed as a stream of information.

1-2. A stream of information is, in reality, a string of bits combined in some specified fashion to form meaningful patterns of data. Two higher levels of organization may exist in a stream of information: characters and words. A character is a specific combination of six bits. The bit structure in a character is represented below where each block represents a bit.

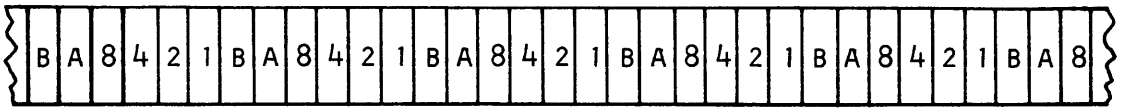


1-3. The next level of organization possible is the word. Each word is composed of eight characters. The structure of a word is illustrated below.

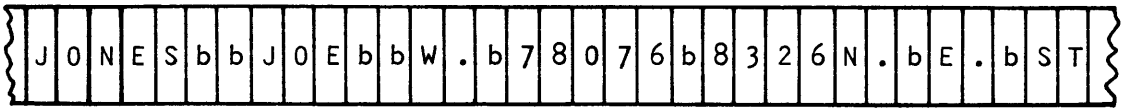


1-4. A stream of information might be considered as a string of bits which have been combined to form characters, and the characters further combined to produce words or other meaningful structures. A stream of information is often referred

to as a string. The input is called the source string and the output is called the destination string. A stream of information is illustrated below.



This stream of information could represent, for instance, the following data.



The STREAM PROCEDURE operates upon a stream of information by passing it through a register, modifying the information in a specific manner, and returning the information to a predetermined storage area. This process is illustrated below.

ADDRESSING CONTROL.

1-5. Once a STREAM PROCEDURE has been given a stream of information, the procedure will control the flow of this information. To accomplish this task, three addressing indexes have been specified for the stream language. These are:

a. The source index.

b. The destination index.

c. The control index.

The first two indexes, the source and destination indexes, might be referred to as the information addressing indexes. The control index might be referred to as the program addressing index.

1-6. The source index contains the address of the information to be processed by the STREAM PROCEDURE. This index is denoted in the STREAM PROCEDURE language by

the symbol, SI. The source index is composed of three registers: the M, G and H registers. The M register is 15 bits in size and contains the address of the current word in the source string. The G register is three bits in size and contains the address of the character currently being referenced in the source word. The H register is also three bits in size, and indicates the bit position within the character currently being referenced.

1-7. The destination index contains the address of the location in memory where the processed information will be stored. This index is represented in the language by the symbol, DI. The destination index is also comprised of three registers: the S, K and V registers. The organization of the destination index is analogous to that of the source index, where the S register contains the address of the word currently being referenced in the destination string, the K register indicates the character position in the word, the V register reflects the location of a specific bit position in the character.

1-8. To provide a better understanding of the stream information address indexes, the various portions of the indexes are labeled by the following subscripts: w for word, c for character, and b for bit. The following summarizes the details of the various portions of the information addressing indexes, SI and DI.

SI_w or DI_w - source or destination area word address.

SI_c or DI_c - source or destination character position in a word.

SI_c - 0 for the leftmost character in the word.

SI_c - 7 for the rightmost character in the word.

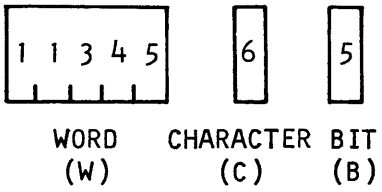
SI_b or DI_b - source or destination bit position in the character.

SI_b - 0 for the leftmost bit in the character (B bit in the zone portion of a character).

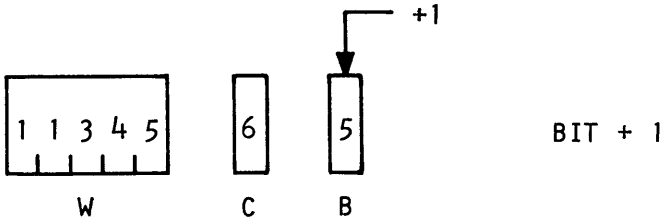
SI_b - 5 for the rightmost bit in the character (1 bit in the numeric portion of a character).

1-9. The operation of the information addressing indexes, SI and DI, proceeds in the following manner. In the bit-addressing operation, the bit portion of the address index will overflow when the number 5 is exceeded. The bit portion of the address index overflows into the character portion of the index, thereby increasing the character count by one. Upon exceeding the number 7, the character portion of the address index overflows into the word portion of the address index. This will cause the word portion of the address index to be increased by one. This process is illustrated below.

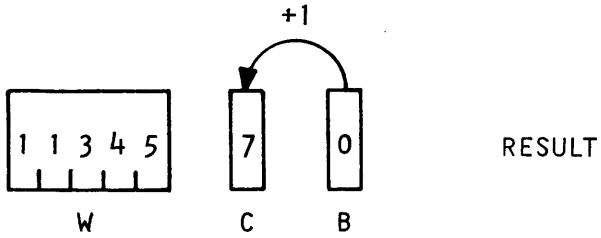
1-10. Assume that an information address index is set to the following arbitrary address.



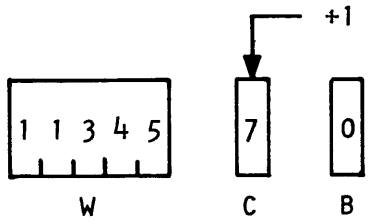
If the B portion of the index is increased by one, the B portion of the index is thereby exceeded.



The B portion automatically overflows into the C portion and the B portion is set to zero.

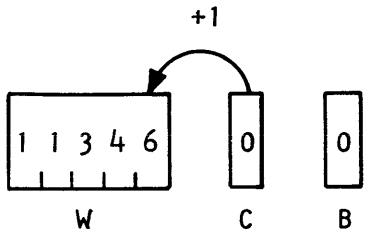


If the C portion of the index is then increased by one, the limit of the C portion is thereby exceeded.



CHARACTER + 1

The C portion automatically overflows into the W portion and the C portion is set to zero.



RESULT

Thus the information address indexes can be counted up as illustrated above.

1-11. The control index contains the address of the program word and the syllable within that program word. The symbol defined in the language for this index is CI. There are two parts of the control index and these are identified by subscripts in the following manner:

CI_W - address of the program control word.

CI_S - program syllable in the program word being referenced.

CI_S - 0 for the leftmost syllable.

CI_S - 3 for the rightmost syllable.

1-12. The control index is composed of the C and L registers. The C register is 15 bits in length and contains the program word currently being executed. The L register is two bits in length and contains the address of the syllable currently being executed. There are four syllables in each program word. The operation of the control index is directly analogous to program-word addressing in the word mode.

STREAM PROCEDURE LANGUAGE DESCRIPTION.

1-13. The declaration for the STREAM PROCEDURE is composed of declarations and statements. The statements used in a STREAM PROCEDURE are of a special nature, and are covered briefly at this point in order to provide the proper orientation for the material which follows. The stream statements provide methods for addressing various portions of a string of information, transferring characters or portions of characters to specified areas, unconditional transfers within a STREAM PROCEDURE, and for performing various kinds of simple arithmetic.

STREAM PROCEDURE OPERATION.

1-14. The general operation of a STREAM PROCEDURE occurs as follows. The STREAM PROCEDURE declaration is defined, and then each time it is to be used, a ⟨stream call statement⟩ is given. Information is supplied to the procedure in two ways. The procedure may receive either the actual value of some parameter or the address where the value of this parameter is located or to be located. The STREAM PROCEDURE proceeds to operate upon the information, using the statements which have been provided for the STREAM PROCEDURE. Characters are transferred, destination areas are filled with edited information, portions of characters or strings are modified, and so on, until the information supplied to the STREAM PROCEDURE has been exhausted or a programed conditional transfer effected. Exit is then made to the main program for resumption of word mode operations.

SECTION 2

THE STREAM PROCEDURE DECLARATION

SYNTAX.

2-1. The following is the syntax used for the STREAM PROCEDURE declaration.

⟨stream procedure declaration⟩ ::= STREAM PROCEDURE ⟨stream procedure heading⟩
⟨stream block⟩ | ⟨type⟩ STREAM PROCEDURE
⟨stream procedure heading⟩⟨stream block⟩

⟨stream procedure heading⟩ ::= ⟨procedure identifier⟩⟨stream formal parameter
part⟩;⟨value part⟩

⟨stream formal parameter part⟩ ::= (⟨formal parameter list⟩)

⟨stream block⟩ ::= ⟨stream block head⟩;⟨compound stream tail⟩

⟨stream block head⟩ ::= BEGIN ⟨stream declaration⟩ | ⟨stream block head⟩ ;
⟨stream declaration⟩

⟨compound stream tail⟩ ::= ⟨stream statement⟩ END |
⟨stream statement⟩ ; ⟨compound stream tail⟩

⟨stream declaration⟩ ::= ⟨stream variable declaration⟩ | ⟨label declaration⟩

⟨stream variable declaration⟩ ::= LOCAL ⟨stream variable list⟩ | ⟨empty⟩

⟨stream variable list⟩ ::= ⟨stream simple variable⟩ |
⟨stream variable list⟩ , ⟨stream simple variable⟩

⟨label declaration⟩ ::= LABEL ⟨label list⟩

⟨label list⟩ ::= ⟨label⟩ | ⟨label list⟩ , ⟨label⟩

⟨label⟩ ::= ⟨identifier⟩

⟨procedure identifier⟩ ::= ⟨identifier⟩

⟨formal parameter list⟩ ::= ⟨formal parameter⟩ | ⟨formal parameter list⟩
⟨parameter delimiter⟩⟨formal parameter⟩

⟨formal parameter⟩ ::= ⟨identifier⟩

⟨value part⟩ ::= VALUE ⟨identifier⟩;| VALUE ⟨identifier list⟩ , ⟨identifier⟩;

⟨identifier list⟩ ::= ⟨identifier⟩ | ⟨identifier list⟩ , ⟨identifier⟩

⟨parameter delimiter⟩ ::= ,|)"⟨letter string⟩"(

⟨letter string⟩ ::= ⟨letter⟩ | ⟨letter string⟩⟨letter⟩ | ⟨space⟩ |
⟨letter string⟩⟨space⟩

SEMANTICS.

2-2. There are two main constructs to be considered in the STREAM PROCEDURE declaration: the ⟨stream procedure heading⟩ and the ⟨stream block⟩.

STREAM PROCEDURE HEADING.

2-3. The ⟨stream procedure heading⟩ has the following general format:

⟨procedure identifier⟩ (FP1, FP2, FP3, ..., FPn);
VALUE FP1, FP2;

2-4. The reserve words STREAM PROCEDURE specify that this structure is a procedure declaration and, specifically, a STREAM PROCEDURE declaration. The ⟨stream procedure heading⟩ contains an identifier which provides a name that is unique to the block within which the procedure appears.

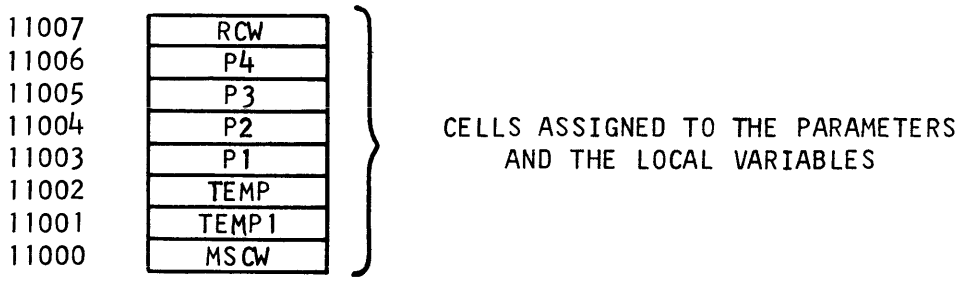
2-5. The ⟨formal parameter part⟩, indicated by (FP1, FP2,...., FPn), should

contain a list of all of the parameters to be used within the STREAM PROCEDURE. The ⟨value part⟩, shown as VALUE FP1, FP2; specifies those formal parameters whose value will be supplied to the STREAM PROCEDURE.

2-6. The operation of formal parameters specified for a STREAM PROCEDURE is as follows. When the STREAM PROCEDURE is called, it will be called by a ⟨stream call procedure statement⟩ which is defined in the Extended ALGOL Reference Manual. The stream call procedure statement must have a one-for-one correspondence of actual parameters to the formal parameters of the STREAM PROCEDURE declaration. When the ⟨stream call procedure statement⟩ is executed during program operation, the parameters will be placed in the program stack. The following example will serve to illustrate how the stack will appear.

```
STREAM PROCEDURE TEST (P1, P2, P3, P4);
VALUE P1, P2;
BEGIN
    LOCAL TEMP, TEMP1;
    .
    .
    .
END;
```

2-7. The stack that would be developed for the ⟨stream procedure declaration⟩ above would be:



All parameters and ⟨stream simple variable⟩'s are then accessed relative to the RCW. P3 and P4 are call-by-name parameters because they are not listed in the value part. They would therefore be placed into the stack as data descriptors. P1 and P2 are call-by-value parameters and would be placed into the stack as values. TEMP and TEMP1 will be either descriptors or operands according to program use.

STREAM BLOCK.

2-8. The ⟨stream block⟩ has the same format as that of a standard ALGOL block. The only difference is in the types of statements which are provided for forming the respective blocks. The general format for a ⟨stream block⟩ is illustrated below:

```
BEGIN LABEL L1, L2,..., Ln;
      LOCAL V1, V2,..., Vn;
      ⟨stream statement⟩ ; ...
      ⟨stream statement⟩ ; ...
      .
      .
      .
      ⟨stream statement⟩ END
```

2-9. The ⟨label declaration⟩ has the same purpose in a STREAM PROCEDURE as in a standard ALGOL procedure: to declare all labels which are to be used within this procedure.

2-10. The ⟨stream variable declaration⟩ is used to specify the variables which are LOCAL to the ⟨stream block⟩.

2-11. The concept of LOCAL as it applies to the STREAM PROCEDURE should be explained more fully at this point. The variables which are declared LOCAL to the STREAM

PROCEDURE are all of a temporary nature and used only within the ⟨stream block⟩ for housekeeping and similar purposes. Variables which are declared LOCAL may not be passed in or out of the procedure as formal parameters. An example of a complete STREAM PROCEDURE declaration is provided to illustrate this concept.

```
STREAM PROCEDURE PACKIT (A, B, C, SOR, DEST); VALUE A, B, C;

    BEGIN LABEL L1, L2, L3;
          LOCAL TANKA, TANKB, TANKC;
          L1:  ⟨stream statement⟩ ;...
               .
               .
          L2:  ⟨stream statement⟩ ;...
               .
               .
          L3:  ⟨stream statement⟩ END
```

SECTION 3

THE STREAM STATEMENT

GENERAL.

3-1. The ⟨stream statement⟩s provide the language for expressing character mode operations.

SYNTAX.

3-2. The following is the syntax used for the stream statements.

⟨stream statement⟩ ::= ⟨unlabeled stream statement⟩ |
⟨label⟩ : ⟨stream statement⟩

⟨unlabeled stream statement⟩ ::= ⟨unconditional stream statement⟩ | 3.1
⟨conditional stream statement⟩* 3.39

⟨unconditional stream statement⟩ ::= ⟨stream address statement⟩ | 3.2
⟨destination string statement⟩ | 3.13
⟨stream go to statement⟩ | 3.34
⟨stream tally statement⟩ | 3.36
⟨stream nest statement⟩ | 3.37
⟨compound stream statement⟩ |
⟨skip bit statement⟩ | 3.34
⟨stream release statement⟩ 3.38

SEMANTICS.

3-3. The syntax listed above indicates that there are several different types of ⟨stream statement⟩s. Some of these statements have several different forms. Before presenting a detail discussion of the ⟨stream statement⟩s and the various forms of these statements, the operation of the information addressing indexes during transfer operations should be discussed.

---

* The ⟨conditional stream statement⟩ is defined in paragraphs 3-75 through 3-81.

TRANSFER OPERATION CONTROL.

3-4. There are two operations which are common to a large portion of the STREAM PROCEDURE language: transfer of characters and transfer of words. The operation of the information addressing indexes, SI and DI, can be critical during transfer of characters or words.

3-5. Before any transfer of words is made, SI and DI must be set to the first character of a word (SI_c and DI_c must equal zero). This is done automatically as illustrated below.

IF $SI_b \neq 0$ OR $SI_c \neq 0$ THEN $SI_w \leftarrow SI_w + 1$; $SI_b \leftarrow SI_c \leftarrow 0$

IF $DI_b \neq 0$ OR $DI_c \neq 0$ THEN $DI_w \leftarrow DI_w + 1$; $DI_b \leftarrow DI_c \leftarrow 0$

IF $SI_b = 0$ AND $SI_c = 0$ THEN $SI_w \leftarrow SI_w$ (no change)

IF $DI_b = 0$ AND $DI_c = 0$ THEN $DI_w \leftarrow DI_w$ (no change)

3-6. After this adjustment to the information addressing indexes has been made, words may be transferred from the source string or a literal to the destination string.

3-7. Before any transfer of characters is made, SI and DI must be set to the first bit of a character (SI_b and DI_b must be equal to zero). This is accomplished automatically as follows:

IF $SI_b \neq 0$ THEN $SI_c \leftarrow SI_c + 1$; $SI_b \leftarrow 0$ (overflow into SI_w may occur)

IF $SI_b = 0$ THEN $SI_c \leftarrow SI_c$ (no change)

IF $DI_b \neq 0$ THEN $DI_c \leftarrow DI_c + 1$; $DI_b \leftarrow 0$ (overflow into DI_w may occur)

IF $DI_b = 0$ THEN $DI_c \leftarrow DI_c$ (no change)

3-8. After this adjustment in the information addressing indexes has been made, characters may be transferred from the source string or a literal to the destination string.

3-9. To summarize the above concepts, whenever information is being transferred from the source string or from a literal to the destination string, care must be taken to ensure that overflow does not occur since this will cause information to be either misplaced or destroyed. As each of the ⟨stream statement⟩s is discussed in the paragraphs that follow, care will be taken to indicate areas where trouble of this nature may occur.

STREAM ADDRESS STATEMENT.

SYNTAX.

3-10. The following is the syntax used for the stream address statement.

⟨stream address statement⟩ ::= ⟨set address statement⟩ | ⟨store address statement⟩ | ⟨skip address statement⟩ | ⟨recall address statement⟩

⟨set address statement⟩ ::= SI ← ⟨source address part⟩ |
DI ← ⟨destination address part⟩

⟨source address part⟩ ::= LOC ⟨stream simple variable⟩ | SC

⟨destination address part⟩ ::= LOC ⟨stream simple variable⟩ | DC

⟨store address statement⟩ ::= ⟨stream simple variable⟩ ← ⟨stream address index⟩

⟨stream address index⟩ ::= SI | DI | CI

⟨stream simple variable⟩ ::= ⟨variable identifier⟩

⟨skip address statement⟩ ::= DI ← DI ⟨stream arithmetic expression⟩
SI ← SI ⟨stream arithmetic expression⟩

⟨stream arithmetic expression⟩ ::= ⟨adding operator⟩ ⟨stream primary⟩

⟨adding operator⟩ ::= + | -

⟨stream primary⟩ ::= ⟨unsigned integer⟩ | ⟨stream simple variable⟩

⟨recall address statement⟩ ::= ⟨stream address index⟩ ← ⟨stream simple variable⟩

⟨unsigned integer⟩ ::= ⟨digit⟩ | ⟨unsigned integer⟩ ⟨digit⟩

⟨variable identifier⟩ ::= ⟨identifier⟩

SEMANTICS.

3-11. There are four types of the ⟨stream address statement⟩, as indicated by the syntax: ⟨set address statement⟩, ⟨store address statement⟩, ⟨recall address statement⟩, and ⟨skip address statement⟩. Each type of the ⟨stream address statement⟩ has its own particular format and operating conventions; therefore, the various types will be discussed individually.

TYPES OF STREAM ADDRESS STATEMENTS.

3-12. SET ADDRESS STATEMENT. The ⟨set address statement⟩ has four different forms. For convenience of explanation, these four forms may be grouped according to usage. The first two forms cause the source or destination indexes to be set to the address of a variable. This address is the stack address of the variable. The variable then becomes either the source or destination string. The first two forms have the following general format:

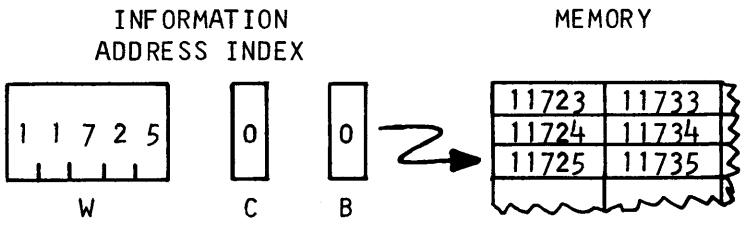
SI ← LOC ⟨Stream Simple Variable⟩ - this form sets the source index to the stack address of the variable indicated by the stream simple variable.

DI ← LOC ⟨Stream Simple Variable⟩ - this form sets the destination index to the stack address of the variable indicated by the stream simple variable.

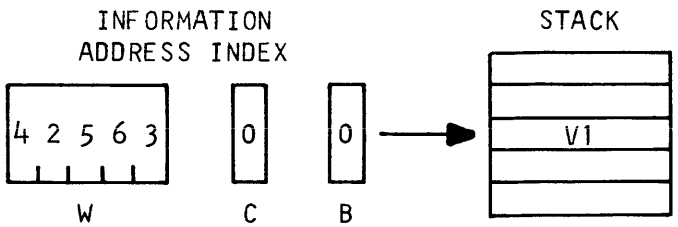
These two forms provide the programmer with the ability to set the source or destination indexes to some particular location in the stack.

3-13. The above two forms provide the programmer with very rapid access to information, particularly for purposes of comparison. However, there are some considerations that should be noted before any use is made of these forms of the ⟨set address statement⟩. If the variable specified by the variable identifier is a call-by-value parameter or a LOCAL variable, the stack location may contain a value. If the variable is a call-by-name parameter, then the stack location will contain an address. In either case, because the source or destination index is set to a stack location, care should be exercised with regard to any subsequent stream transfer operations which reference this location. This is because the stack also contains program control information. If information is transferred into the stack indiscriminately, these control words may be destroyed, thereby causing serious programing problems. The operation of each of these two forms of the ⟨set address statement⟩ is illustrated below.

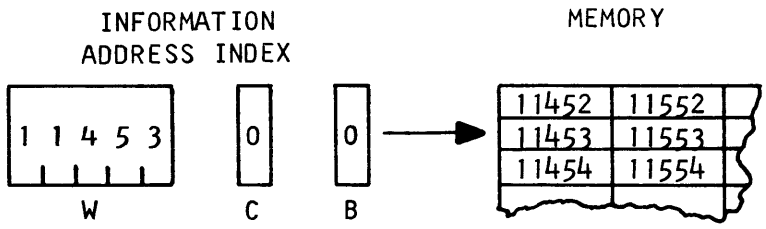
3-14. Assume the information address index is set to some location in memory



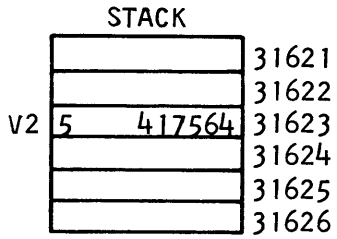
and the statement SI ← LOC V1 is executed, where V1 is an actual parameter or LOCAL variable. The information address index, in this case SI, will now point to the location of V1 in the stack.

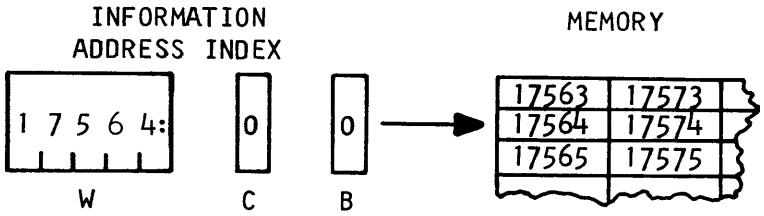


Again assume the information index is set to some location in memory and the



statement SI ← LOC V2 is executed, where V2 is an actual parameter (call-by-name) or a LOCAL variable. The information address index, in this case SI, will now point to a memory location whose address had been contained in a stack location (V2).





If the word in the stack (V2) is an operand, the word and character portions will be set to the appropriate value and the bit portion will be reset to zero. If the word is a descriptor, only the word portion is set, and the character and bit pointers are reset to zero.

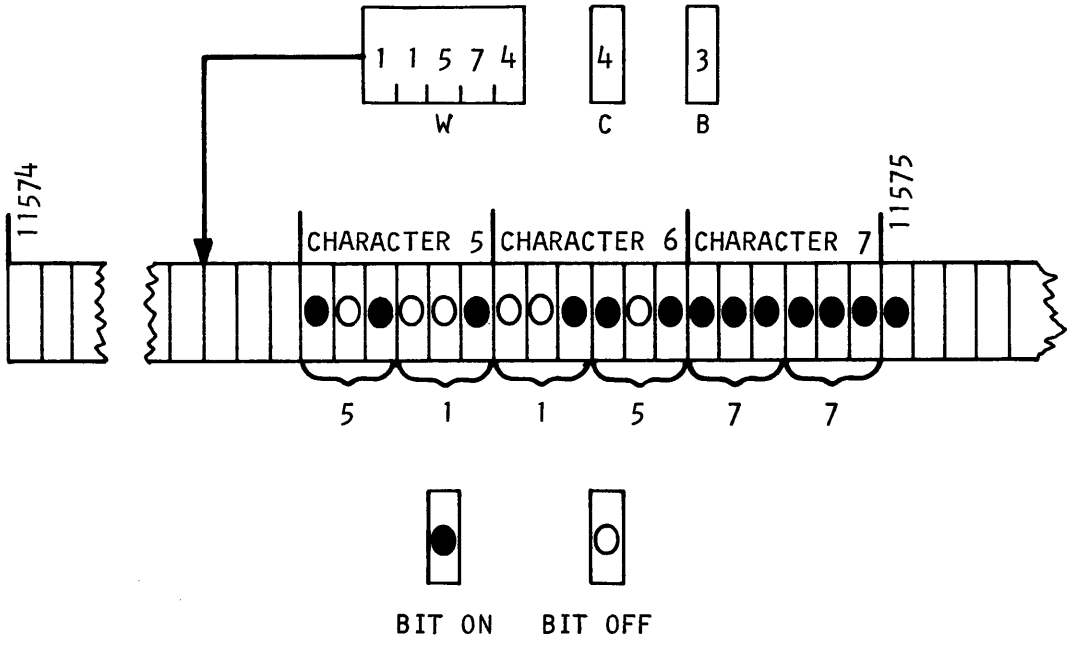
3-15. The last two forms of the ⟨stream address statement⟩ have the following general format:

SI ← SC - this form sets the source index to the value of the next 18 bits in the source string.

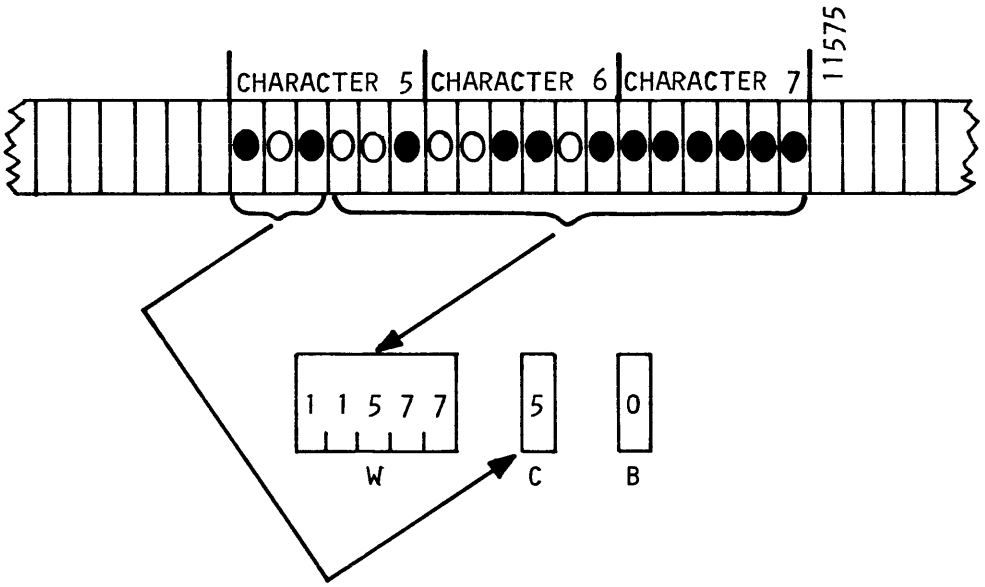
DI ← DC - this form sets the destination index to the value of the next 18 bits in the destination string.

These two forms of the ⟨set address statement⟩ provide the programmer with the ability to set the source or the destination indexes from the source or destination string. The programmer must make certain that the portion of the string which is used to set an index is binary information. If a string of alpha characters is accessed to set the address indexes, a serious error will result in the actual address that is placed in the indexes. The operation of each of the above forms of the ⟨set address statement⟩ is discussed in the paragraphs that follow.

3-16. Assume that an information address index contains the address of a specified information string (see below), and the statement SI←SC is now executed. The bit



pointer is adjusted to zero and the character pointer is counted up one, if necessary ($SI_b \neq 0$). The address index, SI in this case, is set to the value of the next 18 bits of the string. The first three bits set the character portion of the index, and the next 15 bits set the word portion of the index. The bit portion of the index is set to zero.



3-17. STORE ADDRESS STATEMENT. There are three forms of the ⟨store address statement⟩. The first two forms enable the programmer to store the settings of the source or destination indexes. The third form is used to store the setting of the control index. The first two forms have the following format:

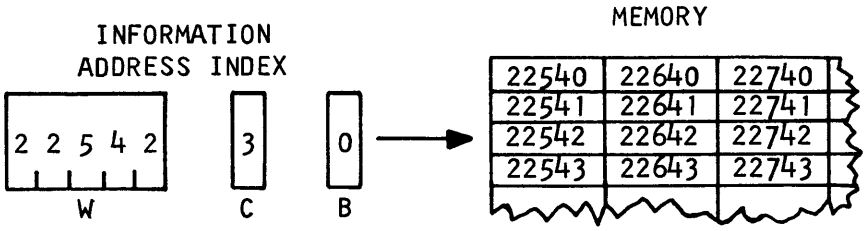
Stream Simple Variable ← SI - this form of the ⟨store address statement⟩ places the address currently in the SI into the location indicated by the variable identifier, in the 18 least significant bits.

Stream Simple Variable ← DI - this form of the ⟨store address statement⟩ places the address presently in the DI into the location indicated by the variable identifier, in the 18 least significant bits.

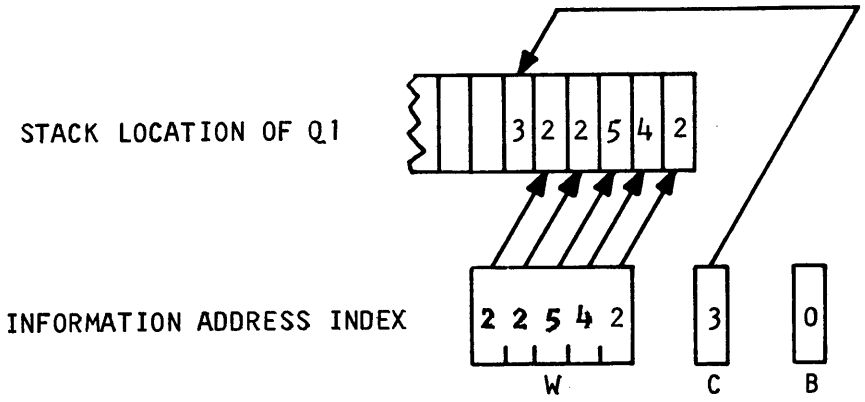
These two forms provide the means for temporarily storing the addresses of the source or destination indexes. Before execution of these forms of the ⟨store address statement⟩, an adjustment may occur in the information address index in order to ensure that it is pointing to the first bit of a character. This adjust-

ment is discussed in detail in paragraphs 3-4 through 3-9. The operation of these two forms of the ⟨store address statement⟩ is illustrated below.

3-18. Assume that an information address index is set to some location in memory



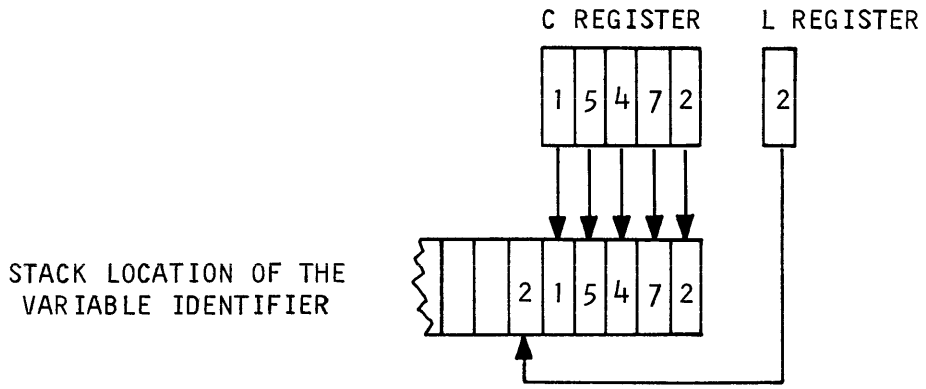
and the statement Q1←DI is executed, where Q1 is a LOCAL variable. The address presently contained in the index, DI in this case, is stored in the stack location which has been assigned to Q1. The word portion is stored in the 15 least significant bit positions. The character portion is stored in the next three high-bit positions.



3-19. The last form of the ⟨store address statement⟩ provides the programmer with the ability to store the setting of the control index. This form of the ⟨store address statement⟩ has the following format:

Variable Identifier ← CI - this form of the ⟨store address statement⟩ places the address currently in the CI into the location indicated by the variable identifier. CI is the C register and the L register.

3-20. The operation of the above form of the ⟨store address statement⟩ proceeds in exactly the same manner as the first two forms of the ⟨store address statement⟩. The word-address is stored into the 15 least significant positions of the word, and the syllable position into the next two high-order positions of the word.



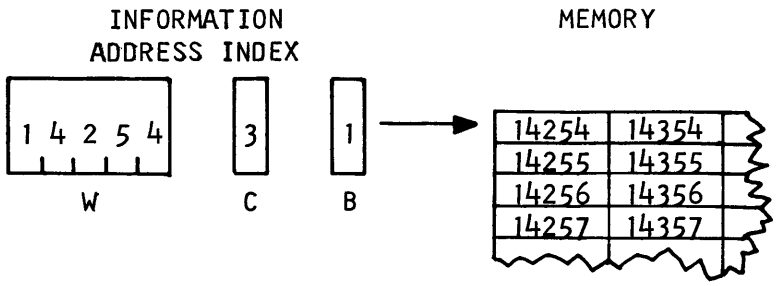
3-21. RECALL ADDRESS STATEMENT. The ⟨recall address statement⟩ has three forms which are the counterparts of the ⟨store address statement⟩. The first two forms allow the programmer to set the source or destination indexes to the location of some variable. The third form is used to recall a previous setting of the control index. The first two forms have the following formats:

SI ← Variable Identifier - this form of the ⟨recall address statement⟩ sets the source index to the value of the variable specified by the variable identifier.

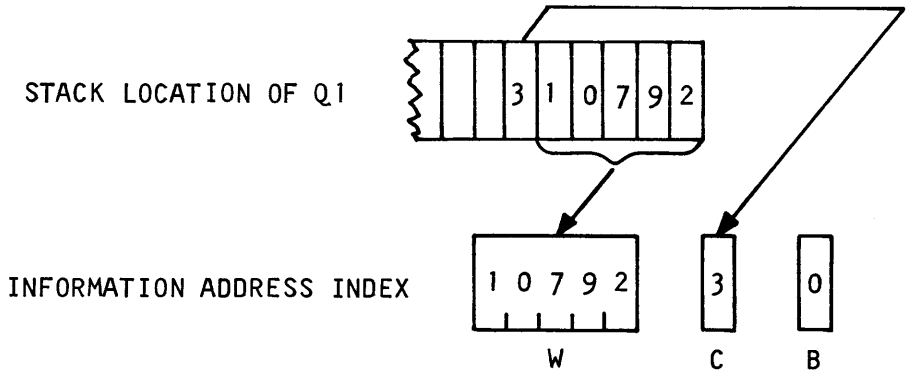
DI ← Variable Identifier - this form of the ⟨recall address statement⟩ sets the destination index to the value of the variable specified by the variable identifier.

The variable identifier, indicated in the above two forms, may be of two types; either a formal parameter, or a quantity which has been declared as LOCAL. The operation of these two forms of the ⟨recall address statement⟩ is discussed below.

3-22. Assume that an information address index is set to some arbitrary address



and the statement DI ← Q1 is executed. The address index, DI in this case, is set to the value of Q1. The word portion of the index is set to the value of the 15 low-order bits. The character portion of the index is set with the next three high-order bits.



3-23. The third form of the ⟨recall address statement⟩ provides the programmer with the ability to recall the address of some previously stored control index setting. The format of this form is as follows:

CI ← Variable Identifier - this form of the ⟨recall address statement⟩ sets the control index to the value of the variable specified by the variable identifier.

This form of the ⟨recall address statement⟩ is the counterpart of the third ⟨store

address statement⟩ discussed earlier. The programmer must use this form to set the address of the control index with variables that had been stored previously by the ⟨store address statement⟩. If other variables are used, they might be outside the scope of the STREAM PROCEDURE in question, thereby causing erroneous results. The operation of this form is directly analogous to the operation of the first two forms of the ⟨recall address statement⟩.

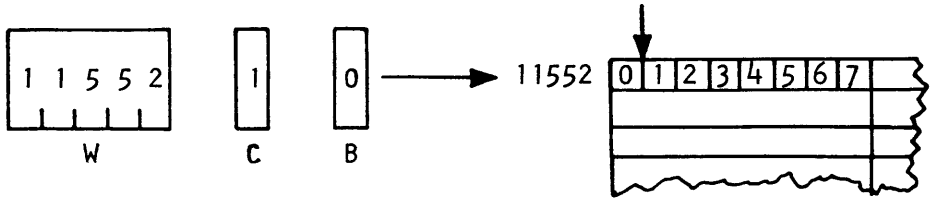
3-24. SKIP ADDRESS STATEMENT. There are two forms of the ⟨skip address statement⟩, one each for controlling the source and destination indexes. The general format for each is as follows:

DI ← DI ± (simple stream variable or integer)

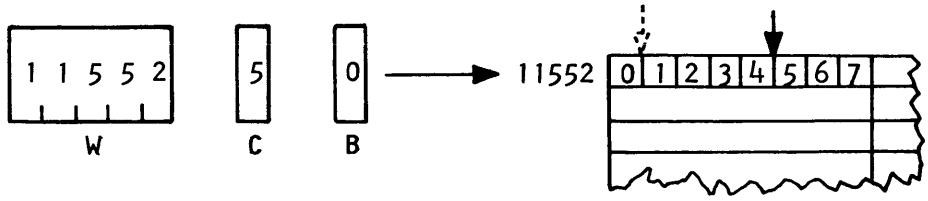
SI ← SI ± (simple stream variable or integer)

These two forms of the ⟨skip address statement⟩ change the source or destination indexes according to the value of the simple variable or integer. The statement gives the programmer the ability to skip over portions of the source or destination string at will. This procedure provides a very flexible means of editing varied forms of data. In one case, an entire string could be edited; in another, selected portions of the string can be selected for editing. The operation of this statement is illustrated below.

3-25. Assume the information address index is set to some specified location



and the statement SI ← SI + X is executed, where X has a value of 4.



DESTINATION STRING STATEMENT.

3-26. The ⟨destination string statement⟩ provides the programmer with the ability to transfer various types of information to the destination. Information may be transferred to the destination string in one of two ways. Information may be transferred from the source string or from a literal representation given in the program.

SYNTAX.

3-27. The following is the syntax used for the destination string statement.

⟨destination string statement⟩ ::= DS ← ⟨transfer part⟩

⟨transfer part⟩ ::= ⟨source string transfer⟩ | ⟨literal transfer⟩

⟨source string transfer⟩ ::= ⟨repetitive indicator⟩ ⟨transfer type⟩

⟨repetitive indicator⟩ ::= ⟨repeat part⟩ | ⟨stream simple variable⟩

⟨repeat part⟩ ::= ⟨empty⟩ | ⟨unsigned integer⟩

⟨transfer type⟩ ::= ⟨transfer word⟩ | ⟨transfer characters⟩ | ⟨transfer and convert⟩ | ⟨transfer and add⟩ | ⟨transfer character portions⟩

⟨transfer words⟩ ::= WDS

⟨transfer characters⟩ ::= CHR

⟨transfer and convert⟩ ::= ⟨input convert⟩ | ⟨output convert⟩

⟨input convert⟩ ::= OCT

⟨output convert⟩ ::= DEC

⟨transfer and add⟩ ::= ADD | SUB

⟨transfer character portions⟩ ::= ZON | NUM

⟨literal transfer⟩ ::= ⟨literal character⟩ | ⟨literal bits⟩

⟨literal characters⟩ ::= ⟨unsigned integer⟩ LIT ⟨string⟩

⟨literal bits⟩ ::= ⟨ri⟩ SET | ⟨ri⟩ RESET

⟨string⟩ ::= "⟨proper string⟩" | "⟨string bracket character⟩"

⟨proper string⟩ ::= ⟨single character⟩ | ⟨proper string⟩ ⟨single character⟩

⟨single character⟩ ::= ⟨visible string character⟩ | ⟨single space⟩

⟨visible string character⟩ ::= . |) |] | ⟨ | ≤ | & | $ | * |] | ≥ | # | - |
/ | , | : | ; | ← | x | (| = | ⟩ | + | A | B |
C | D | E | F | G | H | I | @ | J | K | L | M |
N | O | P | Q | R | ≠ | S | T | U | V | W | X |
Y | Z | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

⟨single space⟩ ::= {a single unit of horizontal spacing which does not contain a visible string character}

⟨string bracket character⟩ ::= "

SEMANTICS.

3-28. The general format for the ⟨destination string statement⟩ is:

⟨destination string statement⟩ ::= DS ← ⟨transfer part⟩

where DS represents the destination string and the ⟨transfer part⟩ represents one of the two major types of transfer available for use with this statement. Information is transferred to the destination string in the manner specified by the ⟨transfer part⟩. The limit of ⟨ri⟩ is 63.

SOURCE STRING TRANSFER.

3-29. The first type of ⟨transfer part⟩ of the ⟨source string transfer⟩ transfers a specific portion of the source string to the destination string. The ⟨source string transfer⟩ format is:

⟨source string transfer⟩ ::= ⟨ri⟩ ⟨transfer type⟩

where ⟨ri⟩ is a repetitive indicator and is formed by an unsigned integer or a simple stream variable. The ⟨transfer type⟩ is a general representation which may take on several different forms. These types are as follows:

a. Transfer words.
b. Transfer characters.
c. Transfer and convert.
d. Transfer and add.
e. Transfer and subtract.
f. Transfer portions of characters.

Each of these types will be considered in detail in the paragraphs that follow.

3-30. TRANSFER WORDS (WDS). This ⟨transfer type⟩ transfers a specific number of words from the source string to the destination string. The number of words transferred depends upon the value of the repetitive indicator. When this ⟨transfer type⟩ is used, the general format for the ⟨destination string statement⟩ is:

DS ← ⟨ri⟩ WDS

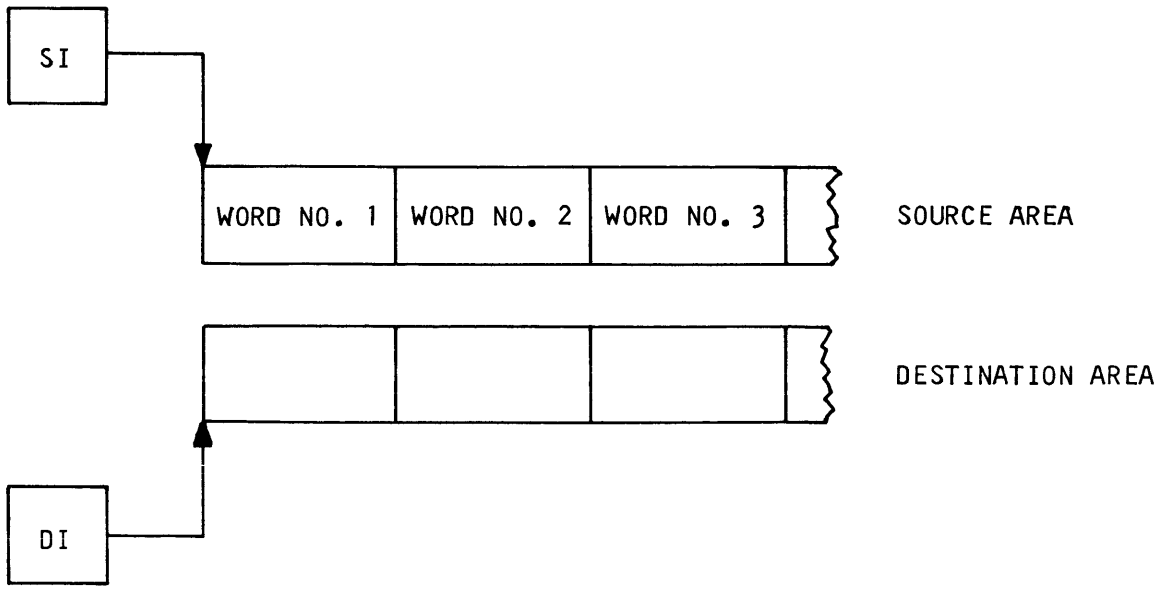
The source and destination indexes are adjusted according to the conventions described previously in paragraphs 3-4 through 3-9. After the address indexes have been adjusted, ⟨ri⟩ source words are transferred to the destination string. The transfer affects SI and DI as follows:

$$SI_w \leftarrow SI_w + ri$$

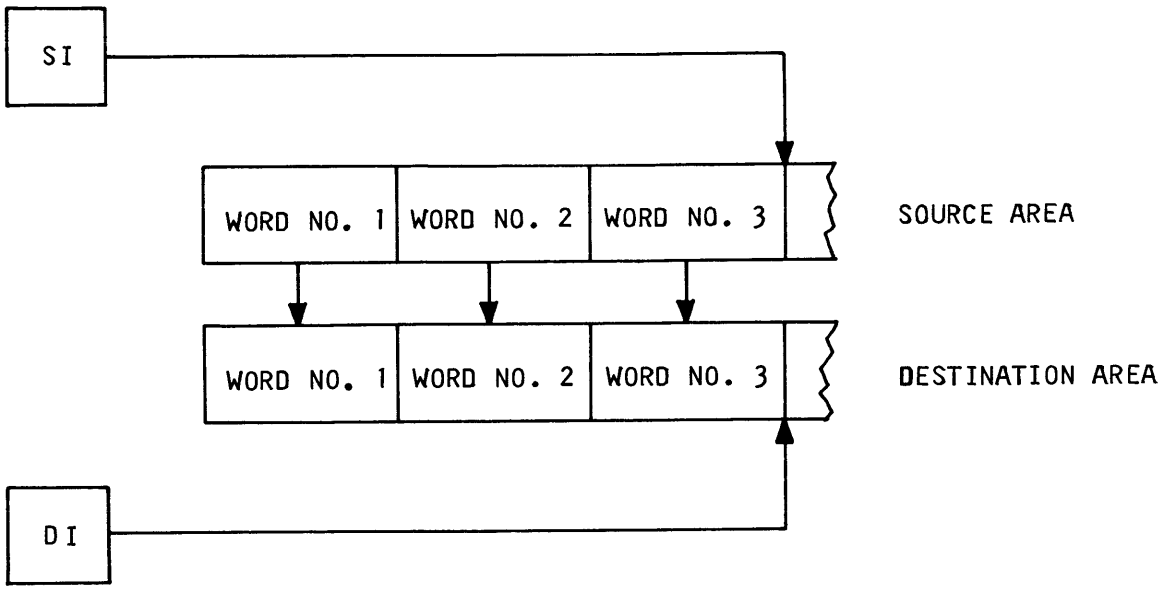
$$DI_w \leftarrow DI_w + ri$$

The operation of this form of the ⟨destination string statement⟩ is illustrated below.

3-31. Assume the statement DS ← 3 WDS is executed and the configuration of the source and destination areas is:



After execution of the above statement, the configuration of the source and destination areas would be:



3-32. TRANSFER CHARACTER (CHR). When used with the transfer character option, the ⟨destination string statement⟩ places a specified number of characters in the destination string, where the number of characters transferred depends upon the value of the repetitive indicator. When this option is used, the general format for the ⟨destination string statement⟩ is as follows:

DS ← ⟨ri⟩ CHR

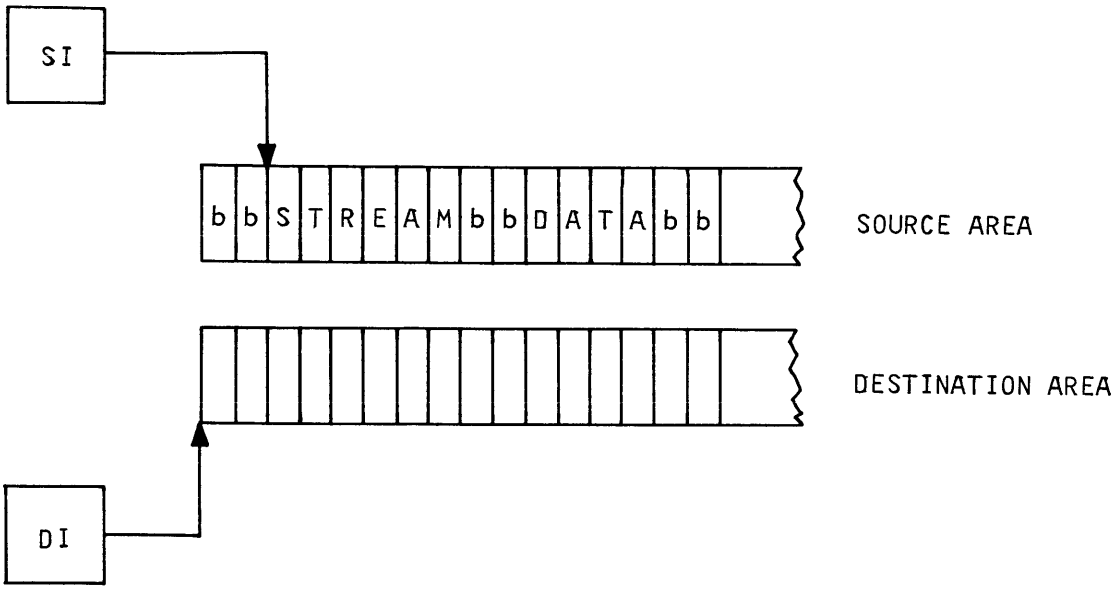
The source and destination indexes are adjusted according to the conventions described for character transfer in paragraphs 3-4 through 3-9. After the address indexes have been adjusted, ⟨ri⟩ source characters are transferred to the destination string. The transfer affects SI and DI as follows:

$SI_c \leftarrow SI_c$ + ⟨ri⟩ (overflow into SI_w may occur)

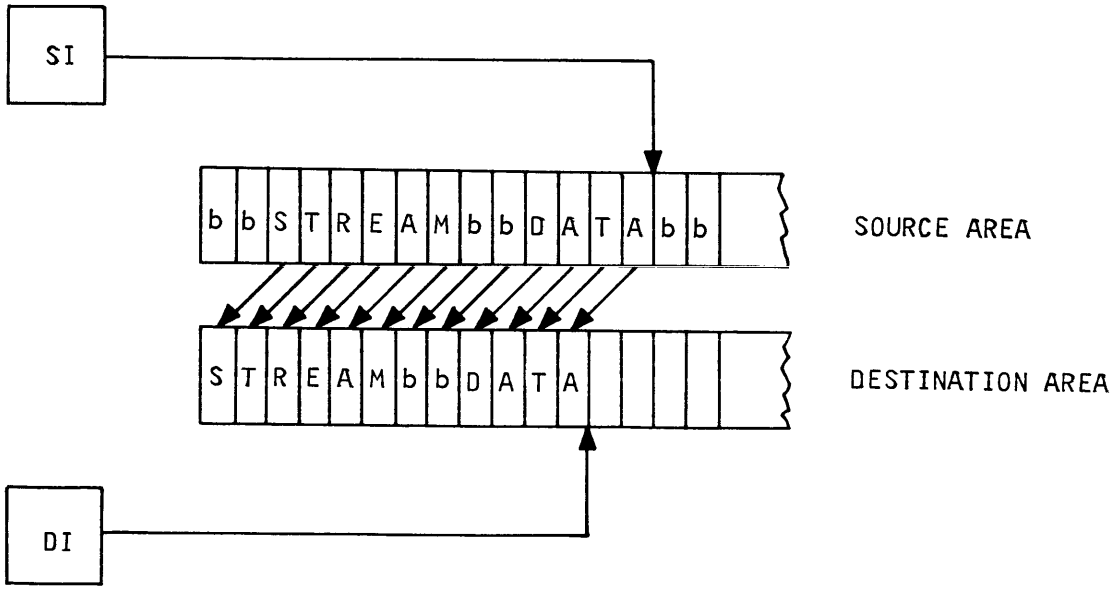
$DI_c \leftarrow DI_c$ + ⟨ri⟩ (overflow into DI_w may occur)

The operation of this form of the ⟨destination string statement⟩ is illustrated below.

3-33. Assume the statement DS ← 12 CHR is executed and the configuration of the source and destination areas is:



After execution of the above statement, the configuration of the source and destination areas would be:



3-34. TRANSFER AND CONVERT OPERATION. There are two forms of the ⟨transfer type⟩s for transfer and convert operations. These two forms have been specified to handle both input and output conversions.

Input Convert (OCT).

3-35. When used with this ⟨transfer type⟩, the ⟨destination string statement⟩ converts ⟨ri⟩ characters of input data from decimal to octal and transfers this data to the destination string in units of words. The general format for this statement is:

DS ← ⟨ri⟩ OCT

The adjustment of the SI is as shown for character transfer operations, and the adjustment needed for the DI is as shown for word transfer operations, both of which have been covered previously in paragraphs 3-4 through 3-9. After these adjustments have been made, ⟨ri⟩ decimal source characters are transferred and converted to an octal word in the destination string. The value of ⟨ri⟩ must be less than or equal to 8.

3-36. If the value of the source field is zero, the sign of the destination field is set to plus; otherwise, the sign is obtained from the zone bits of the least-significant character of the source field. These zone bits are interpreted as follows: If BA=10, the sign is minus; otherwise, the sign is plus. All other zone bits of the source characters are ignored.

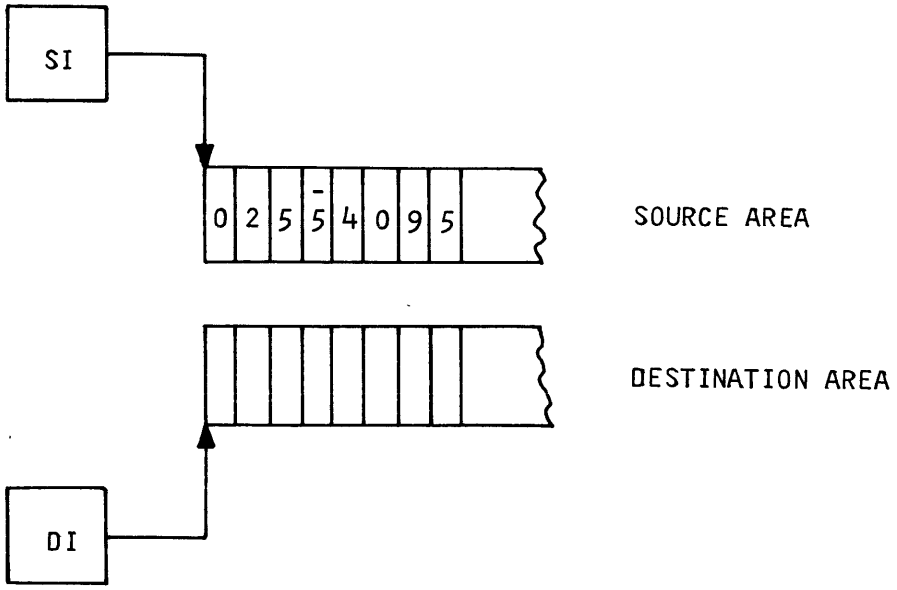
3-37. The decimal source field is treated as an integer, and the resulting destination octal word is an integer. The flag bit, exponent sign, and exponent of the octal word are each set to zero. The transfer affects the SI and DI in the following manner:

$SI_C \leftarrow SI_C + \langle ri \rangle$ (overflow into SI_W may occur)

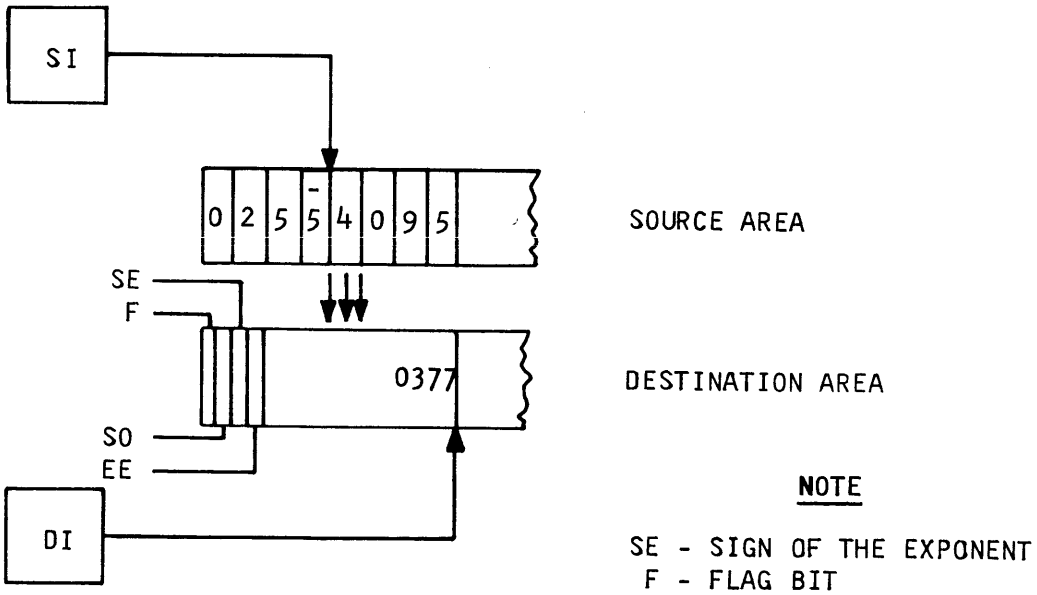
$DI_W \leftarrow DI_W + 1$

The operation of the above form is illustrated below.

3-38. Assume that the statement DS ← 4 OCT is to be executed and the configuration of the source and destination areas is:



After execution of the above statement, the configuration of the source and destination areas is:



Output Convert (DEC).

3-39. When used in this form, the ⟨destination string statement⟩ operates in a manner exactly opposite to that of the Input Convert (OCT) statement. The general format for the statement is:

DS ← ⟨ri⟩ DEC

The adjustment of the SI is as shown for word transfer and the adjustment of the DI is as shown for character transfer, listed previously in paragraphs 3-4 through 3-9. After these adjustments have been made, an octal source word is transferred and converted to ⟨ri⟩ destination characters. The value of ⟨ri⟩ must be less than or equal to 8.

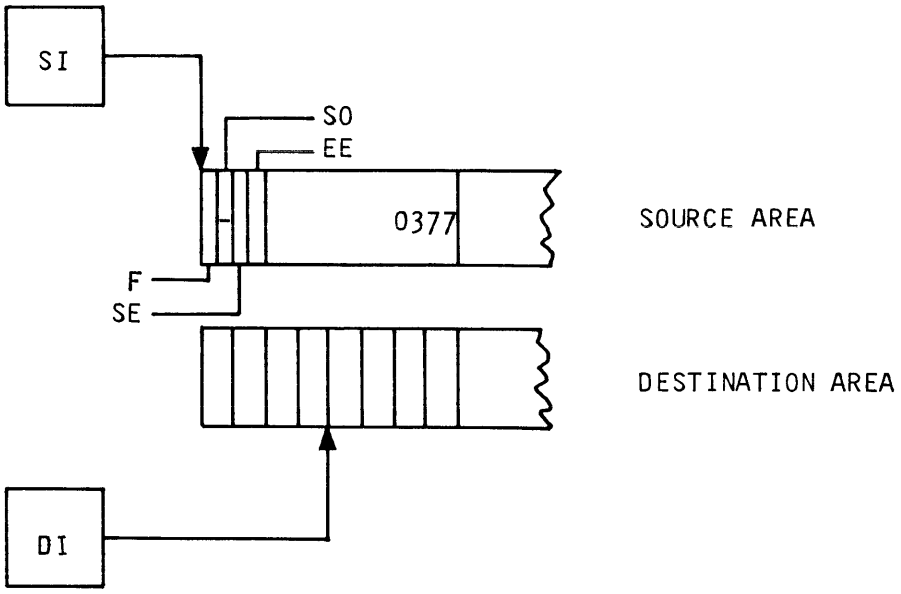
3-40. If the value of the octal source word is zero, the sign of the word is stored as plus in the zone bits of the least-significant destination character (BA=10 is minus). All other zone bits in the destination field are ignored.

3-41. The mantissa of the octal source word is treated as an integer, with the flag bit, exponent sign, and the exponent of the word being ignored. If the value of the octal mantissa is larger than can be converted in ⟨ri⟩ characters, the most-significant characters in excess of ⟨ri⟩ will be lost. The true/false toggle is set to false if any characters are lost; otherwise, it is set to true. The transfer affects the SI and DI as follows:

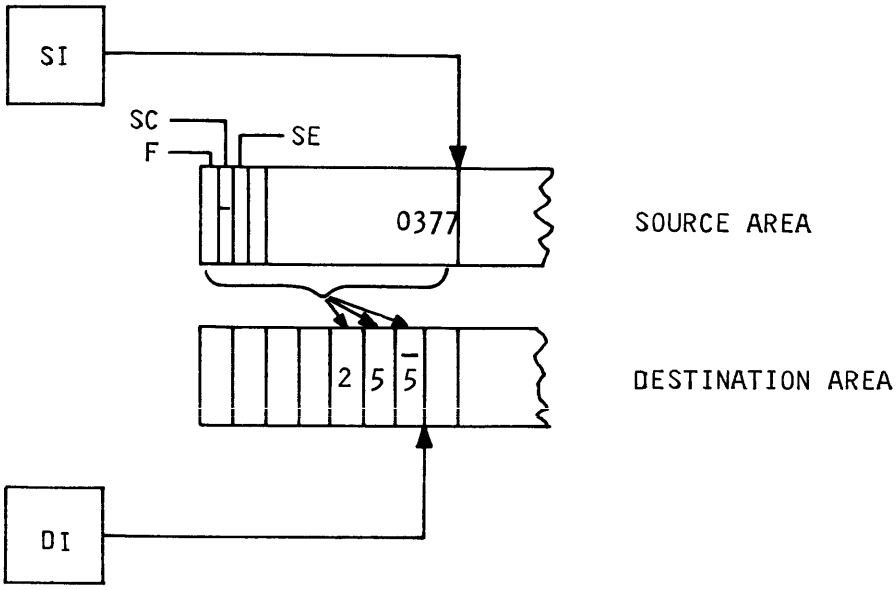
$SI_W \leftarrow SI_W + 1$

$DI_C \leftarrow DI_C$ + ⟨ri⟩ (overflow into DI_W may occur)

3-42. Assume that the statement DS ← 3 DEC is to be executed and that the source and destination areas have the following configuration:



After execution of the above statement, the configuration of the source and destination areas is:



3-43. TRANSFER AND ADD (ADD). When used in the ⟨destination string statement⟩, this ⟨transfer type⟩ algebraically adds a specified number of characters of the source string to the same number of characters in the destination string. The general format of this statement is:

DS ← ⟨ri⟩ ADD

where ⟨ri⟩ has the same meaning as before. The adjustment of the SI and DI is as shown for character transfer, discussed previously in paragraphs 3-4 through 3-9. After the appropriate adjustment of the address indexes has been made, a source field of ⟨ri⟩ characters is transferred and algebraically added to a destination field of ⟨ri⟩ characters.

3-44. The signs of the two fields are the zone bits of their respective least-significant characters. (If BA=10, the sign is minus; otherwise, the sign is plus.) All other source zone bits are ignored. All other destination zone bits are set to zero. The sign of the result is stored in the zone bits of the least-significant destination character.

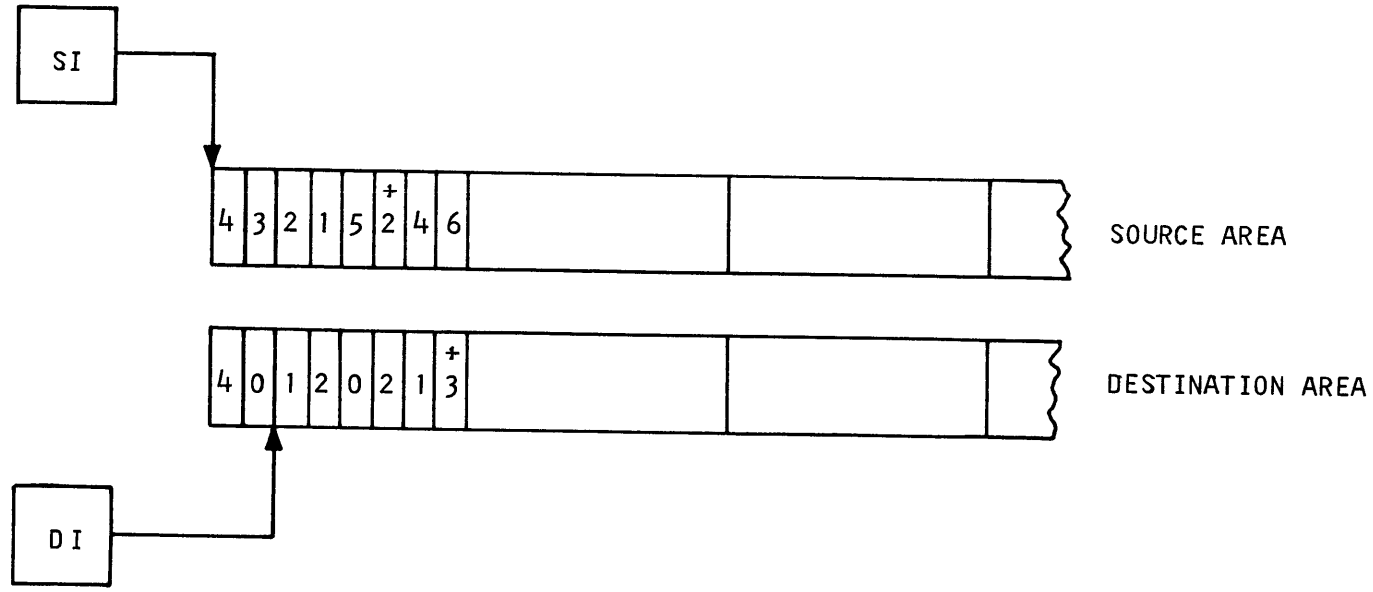
3-45. A result of zero is given a sign of plus except when the destination field is minus zero and the source field is zero with either sign. If overflow occurs in the destination field, the true/false toggle is set to true; otherwise, it is set to false. The transfer affects the SI and DI as follows:

$SI_C \leftarrow SI_C$ + ⟨ri⟩ (overflow into SI_W may occur)

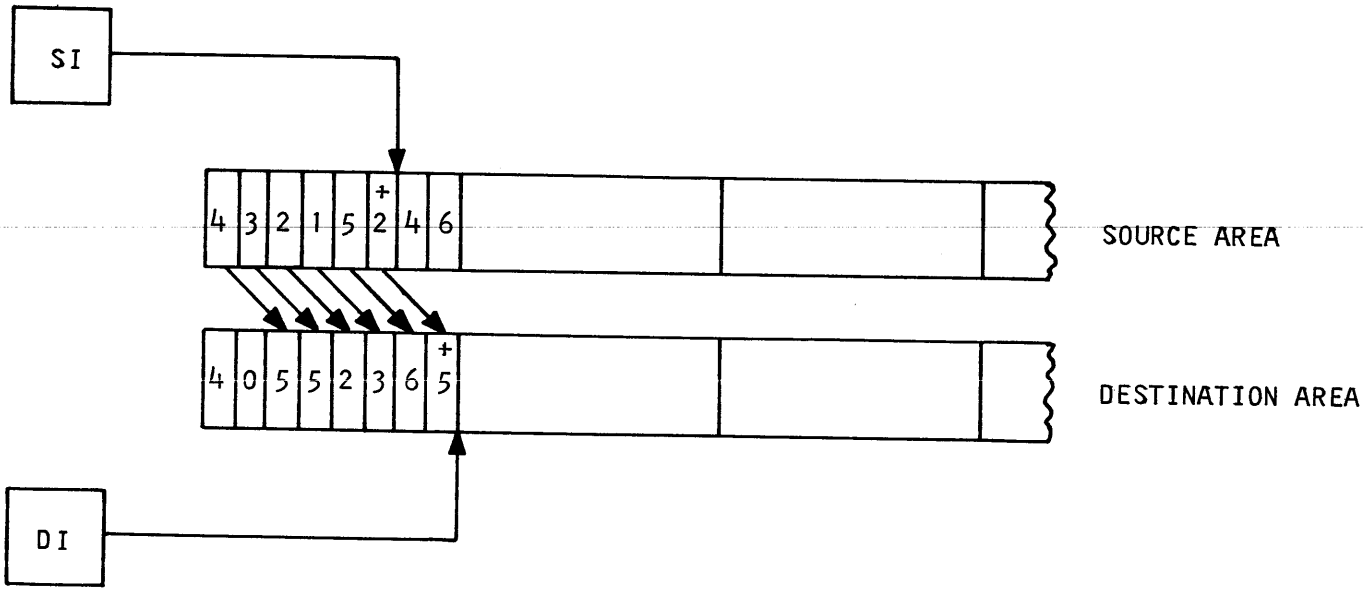
$DI_C \leftarrow DI_C$ + ⟨ri⟩ (overflow into DI_W may occur)

The operation of this form of the ⟨destination string statement⟩ is illustrated below.

3-46. Assume that the statement DS ← 6 ADD is to be executed and the configuration of the source and destination areas is as shown below:



After execution of the above statement, the configuration of the source and destination areas would be:



3-47. TRANSFER AND SUBTRACT (SUB). When used in the ⟨destination string statement⟩, this ⟨transfer type⟩ algebraically subtracts a certain number of characters in the source string from the same number of characters in the destination string. The general format of this statement is:

DS ← ⟨ri⟩ SUB

The adjustment of the SI and DI follows the conventions established for character transfer, listed previously in paragraphs 3-4 through 3-9. After adjustment of the address indexes has taken place, a source field of ⟨ri⟩ characters is transferred and algebraically subtracted from a destination field of ⟨ri⟩ characters. The signs of the two fields are the zone bits of their respective least-significant characters. (If BA=10, the sign is minus; otherwise, the sign is plus.) All other source zone bits are ignored. All other destination zone bits are set to zero. The sign of the result is stored in the zone bits of the least-significant destination character.

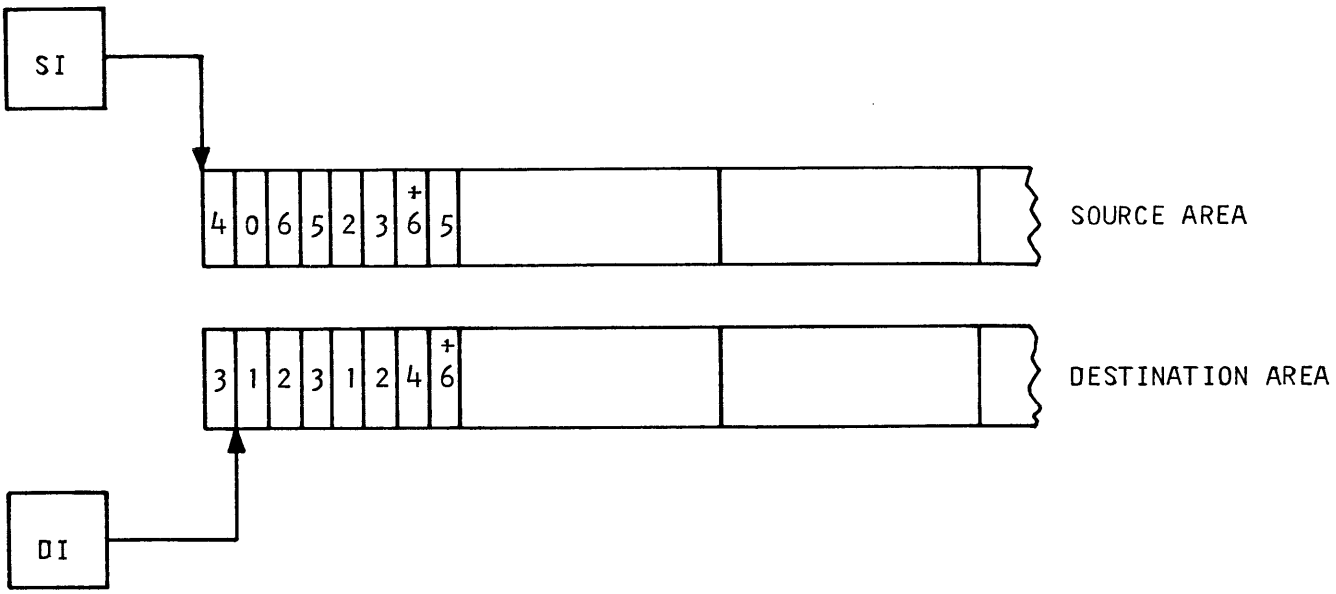
3-48. A result of zero is given a sign of plus except when the destination field is minus zero and the source field is zero with either sign. If overflow occurs in the destination field, the true/false toggle is set to true; otherwise, it is set to false. The transfer affects the SI and DI as follows:

$SI_C \leftarrow SI_C + \langle ri \rangle$ (overflow into SI_W may occur)

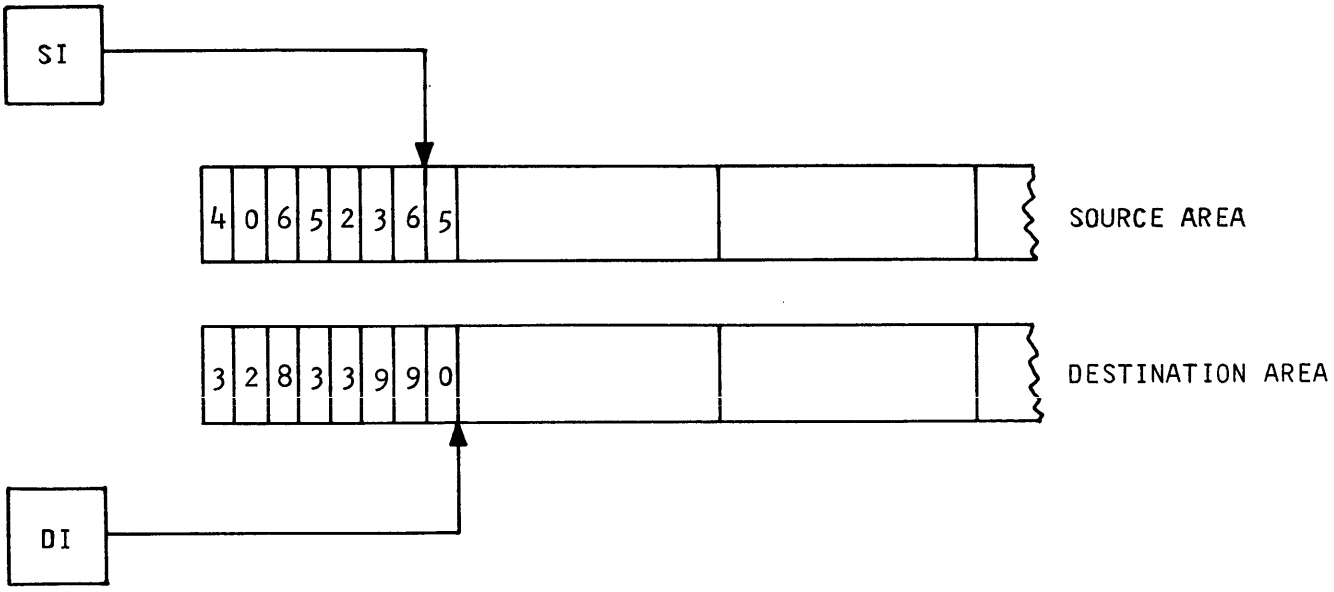
$DI_C \leftarrow DI_C + \langle ri \rangle$ (overflow into DI_W may occur)

The operation of this form of the ⟨destination string statement⟩ is illustrated below.

3-49. Assume that the statement DS ← 7 SUB is to be executed and the configuration of the source and destination areas is:



After execution of the above statement, the configuration of the source and destination areas would be:



3-50. TRANSFER CHARACTER PORTION. This transfer type has two forms. The first form transfers the zone bits from the source string to the destination string. The second form transfers the numeric bits from the source string to the destination string. Each of these forms is discussed in detail below.

Transfer Character Portion (ZON).

3-51. The adjustment of the SI and DI is as shown for character transfer, discussed in paragraphs 3-4 through 3-9. After the adjustment of the address indexes has taken place, the zone bits of ⟨ri⟩ source characters are transferred to the zone portions of ⟨ri⟩ destination characters. The numeric portion of the destination characters is not affected. When this ⟨transfer type⟩ is used, the general format of the ⟨destination string statement⟩ is:

DS ← ⟨ri⟩ ZON

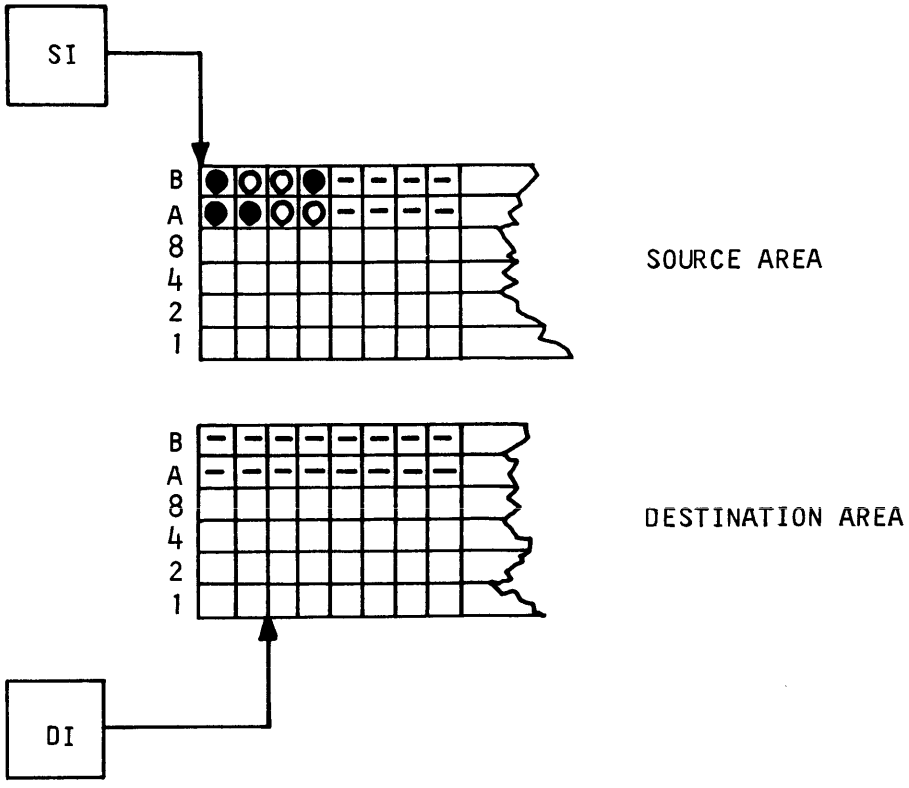
The transfer affects the SI and DI as follows:

$SI_C \leftarrow SI_C + \langle ri \rangle$ (overflow into SI_W may occur)

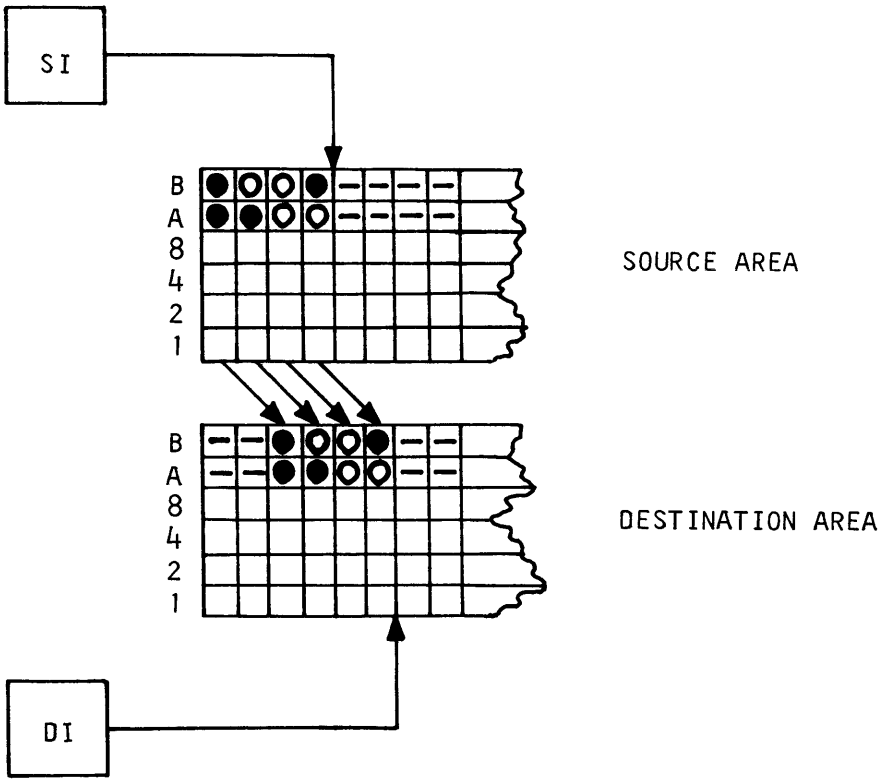
$DI_C \leftarrow DI_C + \langle ri \rangle$ (overflow into DI_W may occur)

The operation of this form of the ⟨destination string statement⟩ is shown below.

3-52. Assume that the statement DS ← 4 ZON is to be executed and the configuration of the source and destination areas appears as shown below:



After execution of the above statement, the configuration of the source and destination areas is as shown below:



All numeric bits remain unchanged.

Transfer Character Portion (NUM).

3-53. The adjustment of the SI and DI is as shown for character transfer, discussed in paragraphs 3-4 through 3-9. After the address indexes have been adjusted, the numeric portion of ⟨ri⟩ source characters are transferred to the numeric portion of ⟨ri⟩ destination characters. The zone bits of the destination characters are set to zero. If the zone portion of the least significant source character is minus (BA=10), the true/false toggle is set to true; otherwise, it is set to false. When this ⟨transfer type⟩ is used, the general format of the ⟨destination string statement⟩ is:

DS ← ⟨ri⟩ NUM

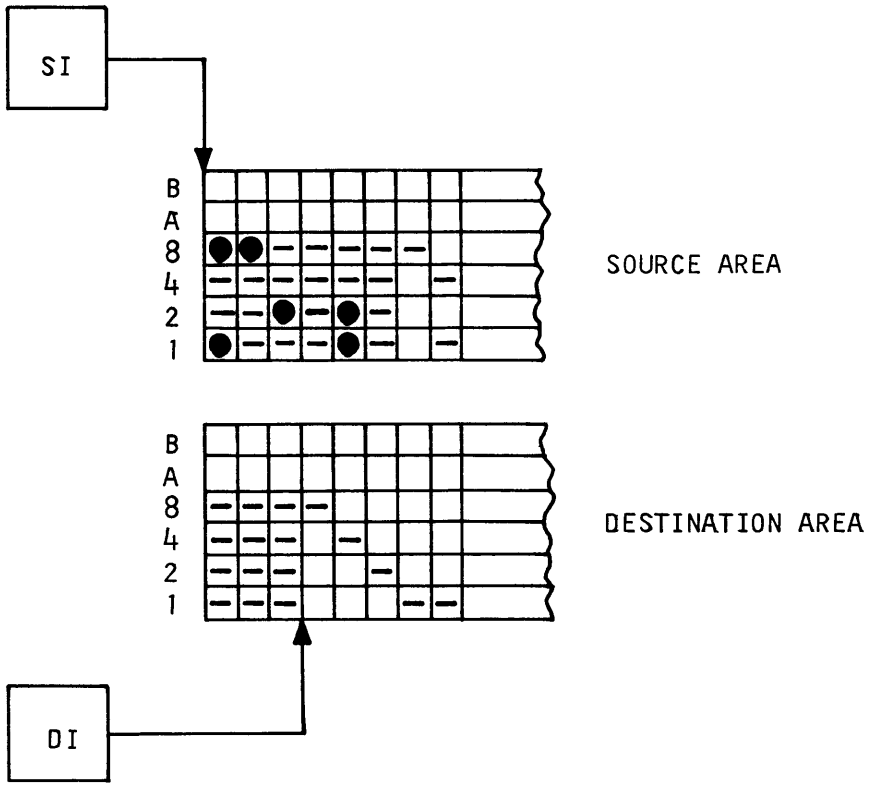
The transfer affects the SI and DI as follows:

$SI_c \leftarrow SI_c + \langle ri \rangle$ (overflow into SI_w may occur)

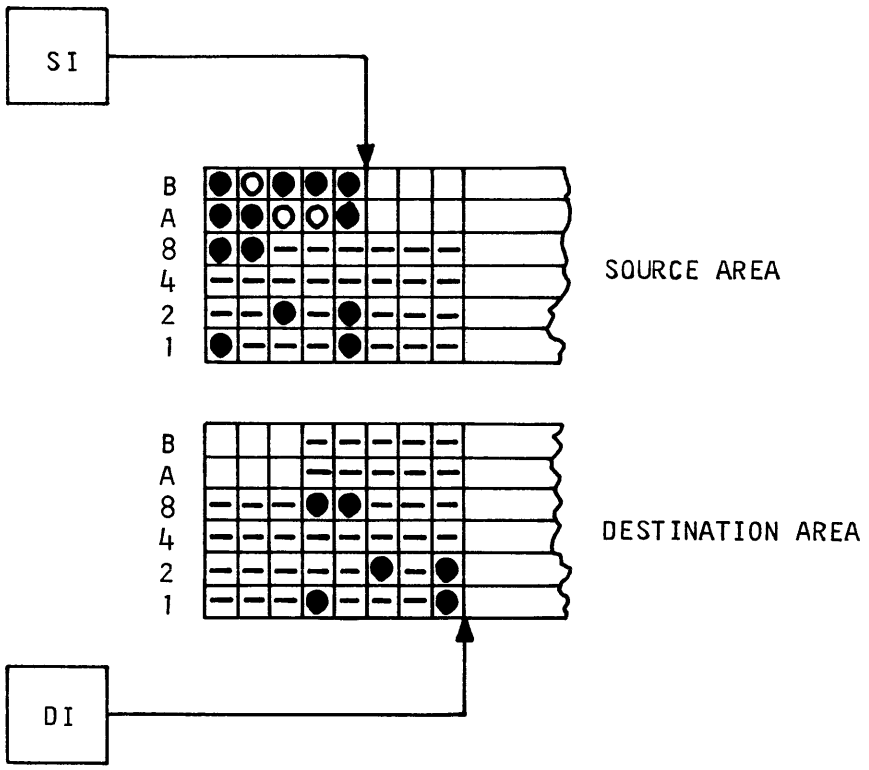
$DI_c \leftarrow DI_c + \langle ri \rangle$ (overflow into DI_w may occur)

The operation of this form of the ⟨destination string statement⟩ is illustrated below.

3-54. Assume that the statement DS ← 5 NUM is to be executed and that the configuration of the source and destination areas is as shown below:



After the above statement has been executed, the configuration of the source and destination areas appears as shown on the next page.



LITERAL TRANSFER

3-55. The second type of transfer defined for the ⟨destination string statement⟩ is the literal transfer. This ⟨transfer type⟩ allows the placement of two types of literals in the destination string: characters or bits.

3-56. LITERAL CHARACTERS. The adjustment of the DI is as shown for character transfer, discussed in paragraphs 3-4 through 3-9. The SI does not need adjustment because the characters being placed in the destination string are contained in the STREAM PROCEDURE code, not in the source string. When this ⟨transfer type⟩ is used, the general format for the ⟨destination string statement⟩ is:

DS ← ⟨ri⟩ LIT "string"

where the ⟨ri⟩ specifies how many characters of the ⟨"string"⟩ following the reserved word LIT will be placed in the destination string.

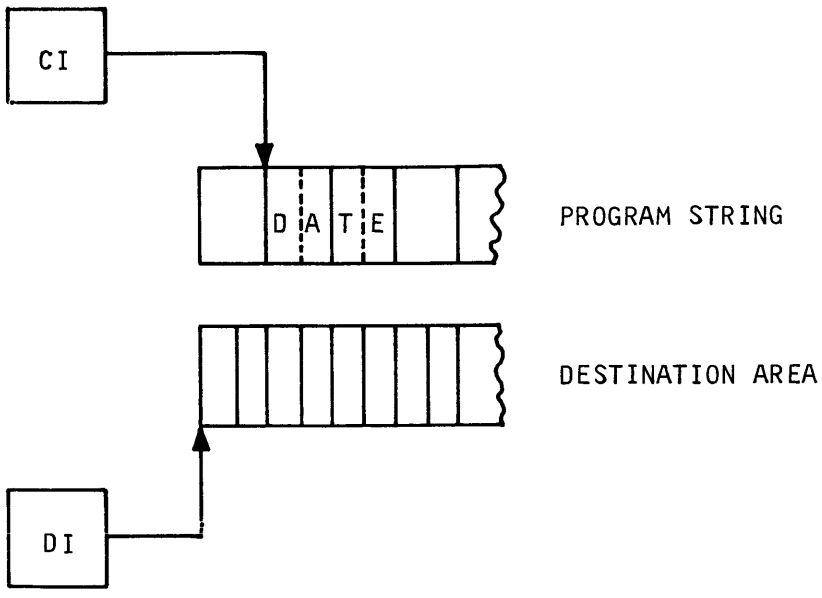
3-57. After adjustment of the DI, ⟨ri⟩ literal characters from ⟨"string"⟩ are transferred to the destination string. The ⟨ri⟩ should be an integer in value and should be equal to the number of characters in the ⟨"string"⟩. If ⟨ri⟩ is greater than the number of literal characters, repetitive left-to-right use is made of the ⟨"string"⟩ until the designated number of destination characters is filled. If ⟨ri⟩ is less than the number of characters in the ⟨"string"⟩, the rightmost characters of the ⟨"string"⟩ are ignored. The transfer affects SI and DI as follows:

$SI \leftarrow SI$ (no change)

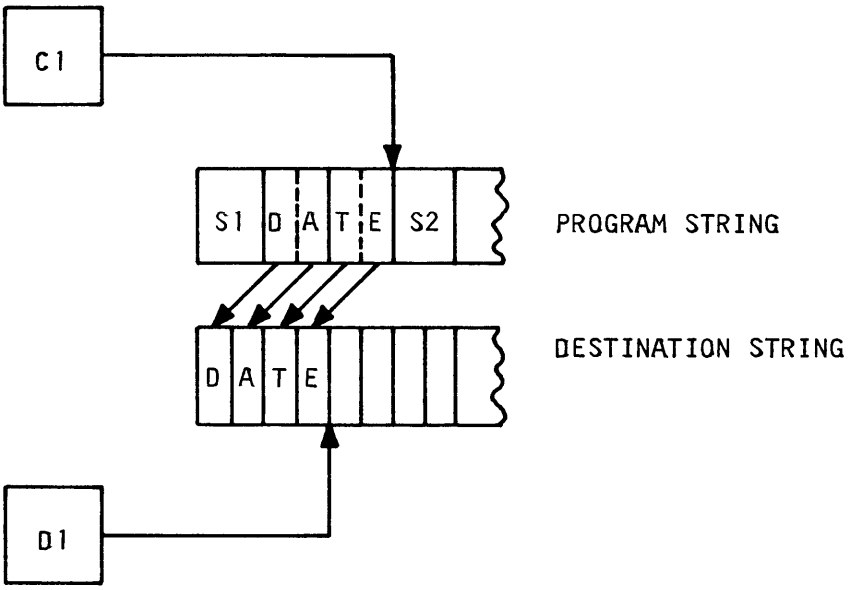
$DI_C \leftarrow DI_C + \langle ri \rangle$ (overflow may occur into DI_W)

The operation of this form of the ⟨destination string statement⟩ is illustrated below.

3-58. Assume that the statement DS ← 4 LIT "DATE" is to be executed and that the configuration of the program string and the destination area is as shown below:



After execution of the above statement, the configuration of the program string and the destination area is:



This statement does not affect the SI. It should be noted that the literal string was placed in the program string during compilation.

3-59. LITERAL BITS. When used in combination with this ⟨transfer part⟩, the destination string statement turns a specified number of bits on or off. The general format for this statement is as follows:

DS ← ⟨ri⟩ SET

DS ← ⟨ri⟩ RESET

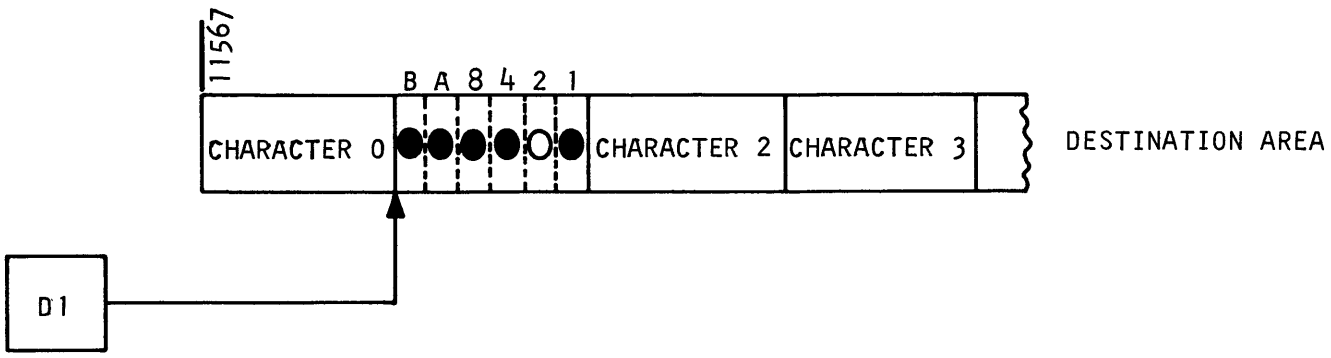
The number of bits to be operated upon is indicated by ⟨ri⟩. The reserved word SET causes a bit to be turned on, and the reserved word RESET causes a bit to be turned off. The effect of this operation upon the SI and DI is as follows:

SI ← SI

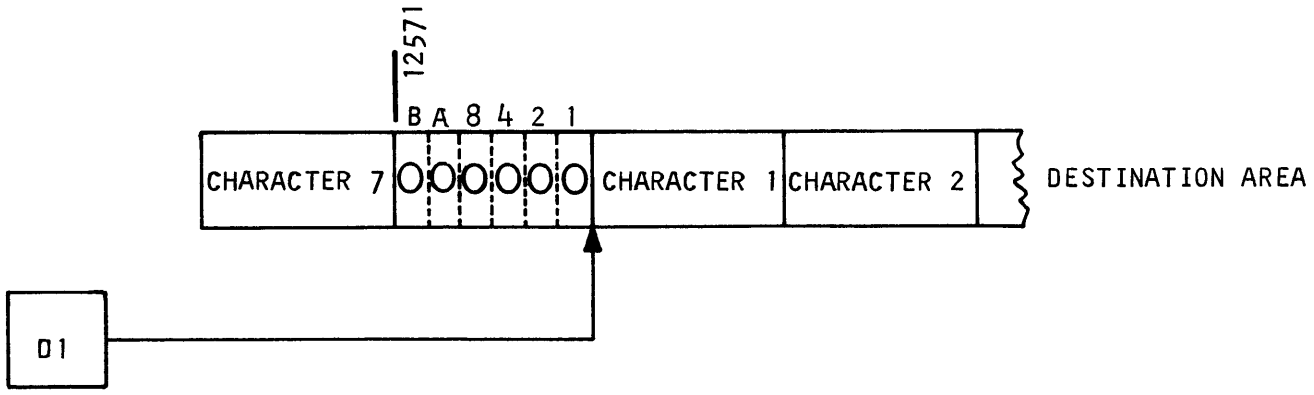
DI ← DI + ⟨ri⟩ (overflow into DI_C may occur, as well as overflow into DI_W)

The operation of this form of the ⟨destination string statement⟩ is illustrated below.

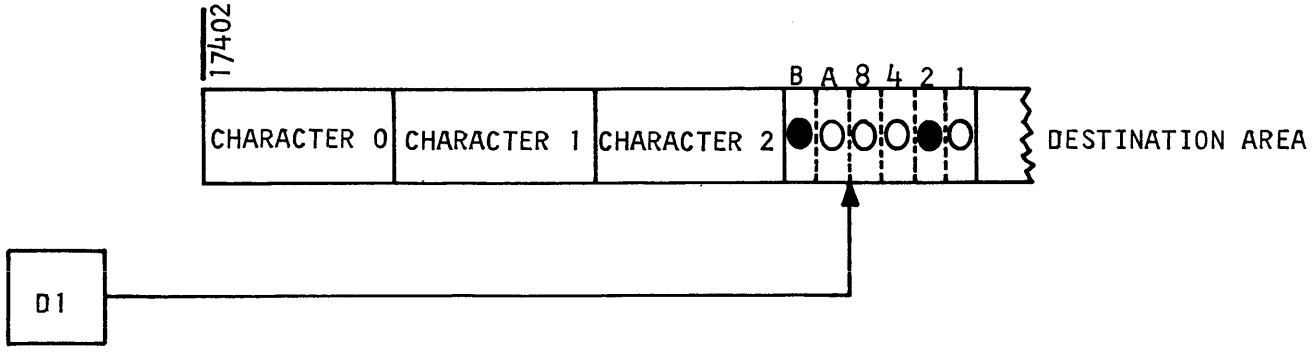
3-60. Assume that the statement DS ← 6 RESET is to be executed and the configuration of the destination area is as shown below:



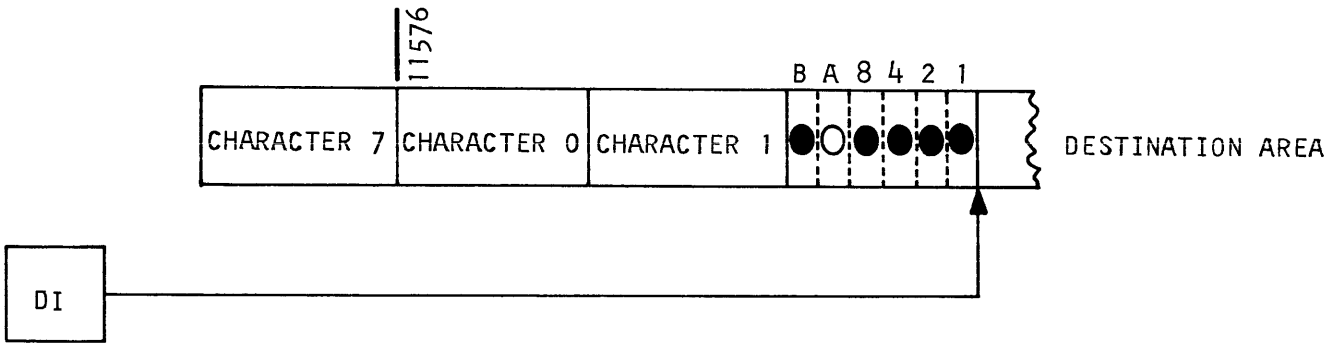
After execution of the statement, the configuration of the destination area is:



This statement does not affect SI. Now assume that the statement DS ← 4 SET is to be executed and the configuration of the destination area is as shown below:



After execution of the statement, the configuration of the destination area is:



This statement does not affect SI.

STREAM GO TO STATEMENT.

SYNTAX.

3-61. The syntax used with the ⟨stream go to statement⟩ is:

⟨stream go to statement⟩ ::= GO TO ⟨label⟩

SEMANTICS.

3-62. This statement causes an unconditional transfer within the STREAM PROCEDURE. It is directly analogous to the GO TO statement defined in ALGOL, except that the label must be LOCAL to the STREAM PROCEDURE in which the ⟨stream go to statement⟩ appears. An example of this statement is:

GO TO START

When the above statement is executed, control is transferred to the point in the program indicated by the label START.

SKIP BIT STATEMENT.

SYNTAX.

3-63. The following is the syntax used with the ⟨skip bit statement⟩.

⟨skip bit statement⟩ ::= SKIP ⟨repetitive indicator⟩ ⟨source or destination bit⟩

⟨source or destination bit⟩ ::= SB | DB

SEMANTICS.

3-64. The ⟨skip bit statement⟩ allows the programmer to skip specified bits in the source string or the destination string. The general formats for this statement are as follows:

⟨skip bit statement⟩ ::= SKIP ⟨ri⟩ SB

::= SKIP ⟨ri⟩ DB

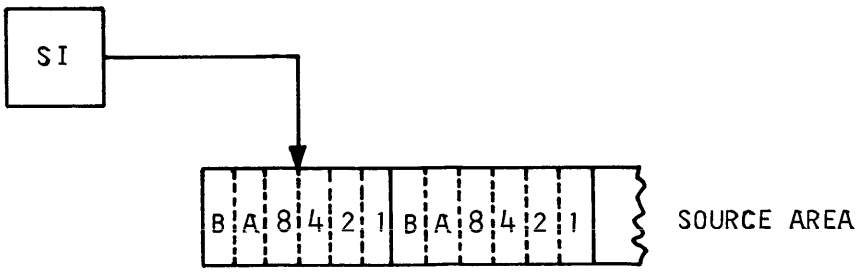
where ⟨ri⟩ is the repetitive indicator, SB specifies the ⟨ri⟩ source bits which will be skipped in the source string, and DB indicates the same action for the destination string. This statement affects the SI and DI as follows:

(SB) SI ← SI + ⟨ri⟩ (overflow into SI_C and SI_W may occur)

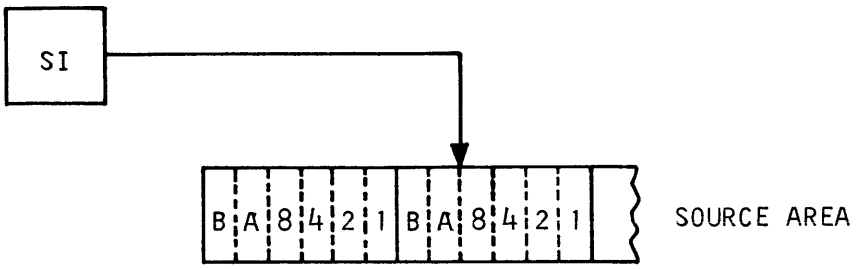
(DB) DI ← DI + ⟨ri⟩ (overflow into DI_C and DI_W may occur)

The operation of this statement is illustrated below.

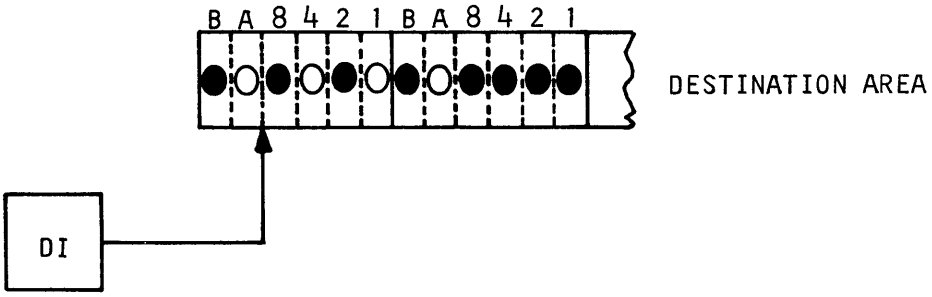
3-65. Assume the statement SKIP 5 SB is to be executed and the configuration of the source area is as shown below:



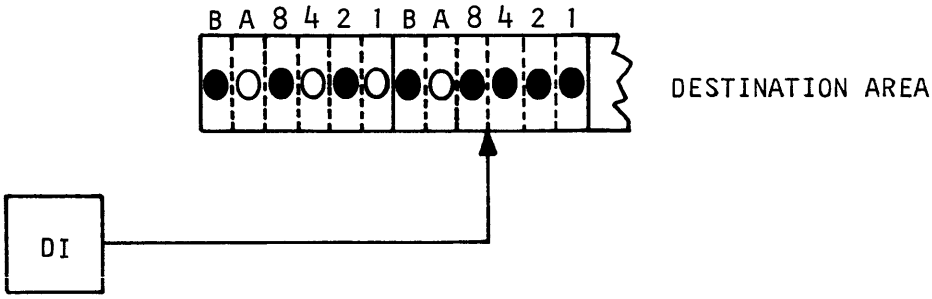
After execution of the statement, the configuration of the source area is:



DI is not affected by this statement. Now assume the statement SKIP 7 DB is to be executed and that the configuration of the destination area is as shown below:



After execution of the statement, the configuration of the destination area is:



SI is not affected by this statement.

STREAM TALLY STATEMENT.

SYNTAX.

3-66. The following is the syntax used with the ⟨stream tally statement⟩.

⟨stream tally statement⟩ ::= TALLY ← ⟨stream primary⟩ |
TALLY ← TALLY + ⟨stream primary⟩ |
⟨stream simple variable⟩ ← TALLY

SEMANTICS.

3-67. The STREAM PROCEDURE is designed with character manipulation operations in mind, but it is sometimes necessary to perform arithmetic operations within the STREAM PROCEDURE. Therefore, the tally statements have been designed to handle such operations. Local variables can be initialized and incremented with these statements. TALLY has a value which is modulo-64; all overflows are lost.

3-68. The operation of this statement is illustrated by the following examples:

```
TALLY ← ABLE
TALLY ← TALLY + 1
TALLY ← TALLY + BETA
GAMMA ← TALLY
```

STREAM NEST STATEMENT.

SYNTAX.

3-69. The following is the syntax used with the ⟨stream nest statement⟩.

⟨stream nest statement⟩ ::= ⟨repetitive indicator⟩ (⟨compound nest⟩)

⟨compound nest⟩ ::= ⟨nest⟩ | ⟨nest⟩ ; ⟨compound nest⟩

⟨nest⟩ ::= ⟨stream statement⟩ | ⟨jump out statement⟩ | ⟨label⟩ :
⟨jump out statement⟩

⟨jump out statement⟩ ::= JUMP OUT | JUMP OUT ⟨number of nests⟩ TO ⟨label⟩

⟨number of nests⟩ ::= ⟨empty⟩ | ⟨unsigned integer⟩

SEMANTICS.

3-70. This statement serves as a repetitive control statement where loops can be described, and the number of passes through these loops is indicated by the ⟨ri⟩. The general format for this statement is as follows:

⟨stream nest statement⟩ ::= ⟨ri⟩ (⟨stream statements⟩)

The ⟨stream nest statement⟩ may be formed from ⟨stream statement⟩s, ⟨compound stream statement⟩s, and a statement called the ⟨jump out statement⟩. A ⟨jump out statement⟩ may only appear in a ⟨stream nest statement⟩. It transfers control to the statement immediately beyond the next right parenthesis. In the case of nested ⟨stream nest statement⟩s, the ⟨jump out statement⟩ escapes from the innermost statement only. An example of the ⟨stream nest statement⟩ containing the ⟨jump out statement⟩ is as follows:

⟨stream nest statement⟩ ::= ⟨ri⟩ (BEGIN SNS; JUMP OUT END);

The SNS refers to other ⟨stream nest statement⟩s. The ⟨jump out statement⟩ may be labeled in the same manner as any other ⟨stream statement⟩.

3-71. The operation of the ⟨stream nest statement⟩ is illustrated in the following examples:

```
25 (IF SC = "E" THEN JUMP OUT; SI ← SI + 1; TALLY ← TALLY + 1)
 6 (IF SB = 1 THEN BEGIN DS ← 8 DEC; V1 ← TALLY; JUMP OUT END TALLY ← TALLY + 1)
 L : 4 (IF SC = "0" THEN 4 (IF SC = "1" THEN JUMP OUT 2 TO L))
```

STREAM RELEASE STATEMENT.

SYNTAX.

3-72. The syntax for the ⟨stream release statement⟩ is:

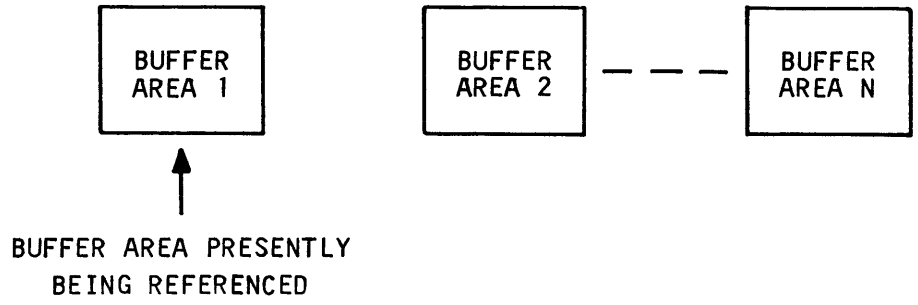
⟨stream release statement⟩ ::= RELEASE (⟨formal parameter⟩)

SEMANTICS.

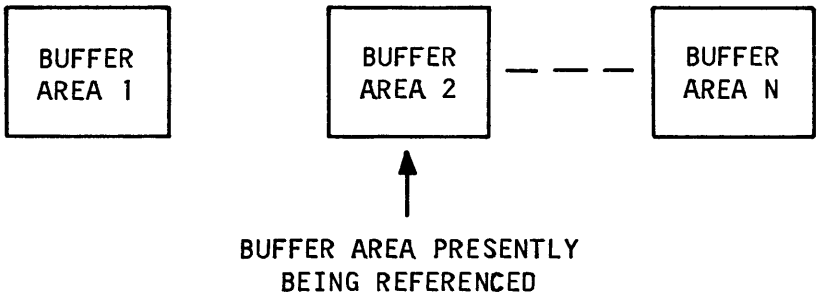
3-73. The formal parameter of a ⟨stream release statement⟩ must be replaced at program execution time by a file identifier, through the substitution of an actual parameter. The file identifier may represent either an input or an output file.

If the identifier represents an input file, the ⟨stream release statement⟩ causes one buffer of the file to be filled with new data. If the identifier represents an output file, the ⟨stream release statement⟩ causes the contents of one output buffer to be transferred to the appropriate output device. The operation of the ⟨stream release statement⟩ is illustrated below.

3-74. Assume that the statement RELEASE (CARD); is to be executed and that the buffer areas for the above-named file have the following configuration:



After execution of the statement, the configuration of the buffer areas is:



If the file pertains to input information, then buffer area 1 is refilled with the next record. If the file pertains to output information, then information from buffer area 1 is transferred to the appropriate output unit.

CONDITIONAL STREAM STATEMENT.

SYNTAX.

3-75. The following is the syntax used with the ⟨conditional stream statement⟩.

⟨conditional stream statement⟩ ::= ⟨stream if clause⟩
⟨unconditional stream statement⟩ |
⟨stream if clause⟩ |
⟨label⟩ : ⟨unconditional stream statement⟩ |
⟨conditional stream statement⟩ ELSE |
⟨stream statement⟩

⟨stream if clause⟩ ::= IF ⟨test⟩ THEN

⟨test⟩ ::= ⟨source with literal⟩ | ⟨source with destination⟩ | ⟨source bit⟩ |
⟨true-false toggle⟩ | ⟨source for alpha⟩

⟨source for alpha⟩ ::= SC = ALPHA

⟨source with literal⟩ ::= SC ⟨relational operator⟩ " ⟨single character⟩ " |
SC ⟨relational operator⟩ " ⟨string bracket character⟩"

⟨source with destination⟩ ::= ⟨repetitive indicator⟩ SC ⟨relational operator⟩ DC

⟨source bit⟩ ::= SB

⟨true-false toggle⟩ ::= TOGGLE

SEMANTICS.

3-76. The ⟨conditional stream statement⟩ allows the programmer to carry out various tests on the destination and source strings. There are five tests that may be made: compare source string with a literal, compare source string with the destination string, test for presence or absence of source bit, test setting of the true-false toggle, and test the source string for alpha content. The relational operators defined for the standard language also hold for the STREAM PROCEDURE. The general format of the ⟨conditional stream statement⟩ is as follows:

⟨conditional stream statement⟩ = IF test THEN ⟨stream statement⟩

This statement operates in the following manner. If the test is true, the statement following THEN is executed; otherwise, the statement is ignored. The five types of tests available to this statement are discussed in the paragraphs that follow.

3-77. SOURCE WITH LITERAL. This test causes one source character to be compared against one literal character. The general format of the ⟨conditional stream statement⟩ when this test type is used is:

⟨source with literal⟩ = IF SC ρ "literal" THEN ⟨stream statement⟩;

where ρ is a relational operator and the "literal" is any allowable ALGOL character. The adjustment of SI is the same as that shown for character transfer, discussed previously under TRANSFER OPERATIONS CONTROL. After the adjustment of the SI, one character from the source string is tested against one literal character. The true/false toggle reflects the logical value of the test. The test affects the SI and DI as follows:

SI ← SI (no change)

DI ← DI (no change)

The operation of this form of the ⟨conditional stream statement⟩ is illustrated in the following example:

IF SC = "E" THEN GO TO BLAZES;

3-78. SOURCE WITH DESTINATION. This test compares a specified number of source characters with the same number of destination characters. (ρ will stand for a relational operator.) The number of characters to be compared is specified with

a repetitive indicator, or ⟨ri⟩. When this type of test is used, the general format for the ⟨conditional stream statement⟩ is:

⟨source with destination⟩ = IF ⟨ri⟩ SC ℙ DC THEN ⟨stream statement⟩;

The adjustment of the SI and DI is the same as that shown for character transfer, discussed in paragraphs 3-4 through 3-9. After the adjustment takes place, ⟨ri⟩ source characters are compared with a like number of destination characters. The true/false toggle reflects the logical value of this comparison. The comparison affects SI and DI as follows:

$SI_C \leftarrow SI_C$ + ⟨ri⟩ (overflow into SI_W may occur)

$DI_C \leftarrow DI_C$ + ⟨ri⟩ (overflow into DI_W may occur)

The operation of this form of the ⟨conditional stream statement⟩ is illustrated in the following example:

```
IF 8 SC > DC THEN GO TO HOME;
```

3-79. SOURCE BIT. When this test is used, the ⟨conditional stream statement⟩ causes one source bit to be tested for equality with a literal value of 0 or 1. The general format for this statement is:

⟨source bit⟩ ::= IF SB THEN ⟨stream statement⟩;

The true/false toggle reflects the logical value of this test. This test does not affect DI or SI. The operation of this form of the ⟨conditional stream statement⟩ is illustrated in the following example:

```
IF SB THEN SI ← SI + 1;
```

3-80. TRUE/FALSE TOGGLE. When this test is used, the ⟨conditional stream statement⟩ tests the true/false toggle for equality with one of the logical values,

TRUE or FALSE. The general format of this statement is:

⟨true/false toggle⟩ ::= IF TOGGLE THEN ⟨stream statement⟩;

There is no adjustment of either the SI or DI. The value of TOGGLE is not affected by this test. The operation of this form of the ⟨conditional stream statement⟩ is illustrated in the following example:

```
IF TOGGLE THEN DS ← X ZON;
```

3-81. SOURCE FOR ALPHA. This form of the ⟨conditional stream statement⟩ tests one source character to determine if it is a letter or digit. The true/false toggle reflects the result of this test. The general format for this statement is as follows:

⟨source for alpha⟩ ::= IF SC = ALPHA THEN ⟨stream statement⟩;

The operation of this form of the ⟨conditional stream statement⟩ is illustrated in the following example:

```
IF SC = ALPHA THEN SI ← SI + 1;
```

The comparison affects SI as follows:

SI ← SI (no change)

APPENDIX A

STREAM PROCEDURE APPLICATIONS

The formation and application of a STREAM PROCEDURE can best be shown through the use of examples. Two examples of STREAM PROCEDURE formation and application are given in the following material.

EXAMPLE 1.

```
STREAM PROCEDURE MOVECHARACTERS(N,SOURCE,SSKIP,DEST,DSKIP);
        VALUE  N,SSKIP,DSKIP;
        BEGIN
        SI←SOURCE;    DI←DEST;
        SI←SI+SSKIP;  DI←DI+DSKIP;
        DS←N CHR;
        END;
```

Stream Procedure Movecharacters will move N characters from the sskip-th character in "SOURCE" to the dskip-th character in "DEST".

EXAMPLE 2.

```
REAL STREAM PROCEDURE COMPARE (WORD1,WORD2);
        VALUE WORD1,WORD2;
        BEGIN
           SI ← WORD1;  DI ← WORD2;
           IF 8 SC ≥ DC THEN
                BEGIN
                   SI ← SI - 8; DI ← DI - 8; TALLY ← 1;
                   IF 8 SC = DC THEN TALLY ← 2
                END;
```

```
                    COMPARE ← TALLY

                END COMPARE;
```

Stream Procedure Compare will compare the 8 chr word in word 1 to the 8 chr word in word 2 and sets the value of compare to:

0 if word 1 is less than word 2

1 if word 1 is greater than word 2

2 if word 1 is equal to word 2


Wherever There's
Business There's
Burroughs