SOS

Reference Manual

Distribution No. 1

This is the first in a series of distributions of material which, when complete, will constitute the reference manual for the SHARE 709 System. Each distribution will contain pages to be inserted into a loose-leaf binder, and may consist of basic material for one or more of the sections of the manual and/or replacement pages which correct previously published material. In addition, each distribution will contain a list of current pages of the manual, a list of the form numbers of all previous distributions, and, if necessary because of new included sections or extensive changes, an updated table of contents.

A three-ring loose-leaf binder, SOS Reference Manual — SHARE System for the IBM 709, form X28-1213, will be supplied to 709 installations to hold the pages issued in these distributions. A set of index tabs will also be provided with the binder to facilitate reference to a particular part of the system. The set of tabs will be printed with titles and section numbers of the various parts of the system, as follows:

01 Introduction
02 SCAT Language
03 Compiler
04 Lister
05 Modify and Load
06 Debugging System
07 Input/Output System
08 IB Monitor
09 SHARE Monitor
10 Programming and Operating Notes
11 Glossary
12 Appendices
13 Index

This, then, will be the arrangement of the manual. In connection with the above outline of the SOS reference manual, it should be noted that Section 10, Programming and Operating Notes, was included as a means whereby programming techniques, operating methods, etc., which have been found useful by one user of SOS, could be conveniently included in this manual and thereby communicated to the other users of the system. Material of this sort, which is intended for inclusion in the manual, should be addressed to:

SOS Group
704/709 Applied Programming
International Business Machines Corporation
590 Madison Avenue
New York 22, New York

It is also anticipated that a companion manual, the SHARE-709-System General Information Manual, will be published in the future. That manual will approach SOS on a more basic level and will be intended primarily for persons who are unacquainted with SOS.

In order to facilitate updating of the manual, all pages will specify the distribution number and will include a date, consisting of the month and year published, and a page number. All page numbers, except those for the appendices, will be six digits separated into three groups of two digits by a decimal point. The first two digits will be the section number, the next two digits the chapter number within that section, and the last two digits will be the number of the page within the chapter. For example, 05. 02. 04 will be the number of the fourth page of the second chapter in the Modify and Load Section (05). The page numbers of the appendices will be eight digits long. The first two will be 12 (the section number of the appendices), the next three groups of two digits will be (in order), appendix number, chapter number, and page number.

SOS

Reference Manual

Distribution No. 2

This distribution contains additional reference material for the SHARE 709 System. The attached pages include both new material and pages to replace some previously published. The new material consists of: Section 01: Introduction; Section 08: IB Monitor; and four appendices (Section 12). It will be noted that the appendices included in this distribution are not consecutively numbered. The spaces in the numbering have been left so that the series of appendices presently planned will be numbered in an orderly fashion. The replacement pages include an updated Table of Contents and listing of current pages.

Users of SOS are invited to contribute material, for inclusion in Section 10: Programming and Operating Notes, so that programming techniques, operating procedures, etc. which have been found useful by them may be communicated to other users of the system. Such material should be addressed to:

SOS Group
704/709/7090 Applied Programming
International Business Machines Corporation
1271 Avenue of the Americas
New York 20, New York

SOS

Reference Manual

Distribution No. 3

This distribution contains additional reference material for the SHARE 709 System. The attached pages include both new material and pages to replace some previously published. The new material consists of Section 06: Debugging System; one chapter of Section 07: Input/Output System; and initial pages for Section 13: Index. The replacement pages include an updated Table of Contents and listing of current pages.

Users of SOS are invited to contribute material, for inclusion in Section 10: Programming and Operating Notes, so that programming techniques, operating procedures, etc., which have been found useful by them may be communicated to other users of the system. Such material should be addressed to:

SOS Group
704/709/7090 Applied Programming
International Business Machines Corporation
1271 Avenue of the Americas
New York 20, New York

SOS

Reference Manual

Distribution No.4

This distribution contains additional reference material for the SHARE 709 System. The attached pages include both new material and pages to replace some previously published. The new material consists of five chapters of Section 07: Input/Output System; and Section 09: SHARE Monitor. The replacement pages include an updated Table of Contents and Index, and a listing of current pages.

Users of SOS are invited to contribute material, for inclusion in Section 10: Programming and Operating Notes, so that programming techniques, operating procedures, etc., which have been found useful by them may be communicated to other users of the system. Such material should be addressed to:

SOS Group
704/709/7090 Applied Programming
International Business Machines Corporation
1271 Avenue of the Americas
New York 20, New York

SOS

Reference Manual

Distribution No.5

This distribution contains material for inclusion in the SHARE 709 System Reference Manual. The attached pages include both new material and pages to replace some previously published. The new material consists of Section 02: SCAT Language; Section 03: Compiler; two chapters of Section 07: Input/Output System; and two appendices. The replacement pages include an updated Table of Contents and listing of current pages, and index pages.

Users of SOS are invited to contribute material, for inclusion in Section 10: Programming and Operating Notes, so that programming techniques, operating procedures, etc., which have been found useful can be communicated to other users of the system. Such material should be addressed to:

SOS Group
704/709/7090 Applied Programming
International Business Machines Corporation
1271 Avenue of the Americas
New York 20, New York

ACKNOWLEDGEMENT TO THE SHARE ORGANIZATION

The SHARE 709 System described in this manual was developed under the auspices of the SHARE organization of 704 and 709 users. The specifications for the various components of SOS were developed over a period of eighteen months by the SHARE 709 System Committee which was established in December 1956. This committee originally consisted of:

Chairman:

Mr. Donald L. Shell	General Electric Company, Cincinnati, Ohio

Members:

Miss Elaine M. Boehm	IBM, New York
Mr. Ira Boldt	Douglas Aircraft Corp., Santa Monica, Calif.
Mr. Harvey Bratman	Lockheed Aircraft Corp., Los Angeles, Calif.
Mr. Vincent DiGri	IBM, New York
Mr. Irwin D. Greenwald	Rand Corporation, Santa Monica, Calif.
Miss Maureen E. Kane	IBM, Poughkeepsie
Miss Jane E. King	General Electric Co., Schenectady, New York
Mr. Owen R. Mock	North American Aviation, Los Angeles, Calif.
Mr. Stanley Poley	Service Bureau Corp., New York
Mr. Thomas B. Steel	System Development Corp., Santa Monica, Calif.
Mr. Charles J. Swift	Convair, San Diego, Calif.

Subsequently, during implementation of SOS by IBM, the numerous modifications and clarifications which have become necessary and desirable have been worked out through liaison with the current SHARE 709 System Committee.

PREFACE

In writing this manual, it has been assumed that readers are familiar with the material contained in two IBM manuals:

General Information Manual: IBM 709-7090 Data Processing System, form D22-6508

Reference Manual: IBM 709-7090 Data Processing System, form A22-6503.

In particular, a knowledge of methods of symbolic programming is assumed. Those readers who are unfamiliar with symbolic programming are referred to the sections of the above manuals which deal with that subject. It is anticipated that a primer on symbolic programming and assembly programs in general and on SOS in particular will become available in the near future. At such time as it is available, notice will be given in this manual.

TABLE OF CONTENTS

SECTION 07: INPUT/OUTPUT SYSTEM

00.00.01	- 00.00.04	2	2/60
00.00.05	- 00.00.17	5	6/61
01.00.01	- 01.00.04	2	2/60
02.00.01	- 02.00.11	5	6/61
03.00.01	- 03.00.47	5	6/61
04.01.01	- 04.01.02	3	1/61
04.01.03	- 04.01.04	2	2/60
04.01.05		1	11/59
05.01.01	- 05.01.02	1	11/59
05.02.01	- 05.02.02	2	2/60
05.02.03	- 05.02.04	4	3/61
05.02.05	- 05.02.16	1	11/59
06.01.01	- 06.01.02	5	6/61
06.02.01	- 06.02.02	5	6/61
06.02.03	- 06.02.04	3	1/61
06.02.05	- 06.02.06	5	6/61
06.02.07	- 06.02.10	3	1/61
06.03.01	- 06.03.08	3	1/61
06.04.01	- 06.04.04	3	1/61
06.04.05	- 06.04.06	5	6/61
06.04.07	- 06.04.08	3	1/61
06.04.09	- 06.04.10	5	6/61
06.05.01	- 06.05.04	5	6/61
07.01.01	- 07.01.57	5	6/61
07.02.01	- 07.02.62	5	6/61
07.03.01	- 07.03.11	4	3/61
07.04.01	- 07.04.02	4	3/61
07.04.03	- 07.04.04	5	6/61
07.04.05	- 07.04.11	4	3/61
07.05.01	- 07.05.04	4	3/61
07.05.05	- 07.05.06	5	6/61
07.05.07	- 07.05.08	4	3/61
07.06.01	- 07.06.24	4	3/61
07.07.01	- 07.07.10	4	3/61
07.08.01	- 07.08.03	4	3/61
08.01.01		5	6/61
08.02.01	- 08.02.02	5	6/61
08.02.03	- 08.02.04	3	1/61
08.02.05	- 08.02.06	5	6/61
08.02.07		3	1/61
08.03.01	- 08.03.02	3	1/61
08.03.03		5	6/61
09.01.01	- 09.01.04	4	3/61
09.02.01	- 09.02.02	4	3/61

09.02.03	- 09.02.04	5	6/61
09.02.05	- 09.02.07	4	3/61
09.03.01		4	3/61
09.04.01	- 09.04.04	4	3/61
09.05.01	- 09.05.03	4	3/61
09.06.01	- 09.06.02	4	3/61
11.01.01		5	6/61
11.02.01		5	6/61
11.03.01		5	6/61
11.04.01		5	6/61
11.05.01		5	6/61
11.06.01		5	6/61
11.07.01		5	6/61
11.09.01		5	6/61
11.12.01		5	6/61
11.13.01		5	6/61
11.15.01		5	6/61
11.16.01		5	6/61
11.18.01		5	6/61
11.19.01		5	6/61
11.20.01		5	6/61
11.21.01		5	6/61
12.01.00.01		2	2/60
12.02.00.01	- 12.02.00.02	2	2/60
12.03.01.01		2	2/60
12.03.02.01	- 12.03.02.02	2	2/60
12.03.03.01		2	2/60
12.03.04.01		2	2/60
12.03.05.01		2	2/60
12.03.06.01	- 12.03.06.03	2	2/60
12.03.07.01		2	2/60
12.03.08.01	- 12.03.08.04	5	6/61
12.03.09.01	- 12.03.09.03	5	6/61
12.03.10.01	- 12.03.10.08	5	6/61
12.10.01.01		5	6/61
12.10.02.01	- 12.10.02.02	5	6/61
12.12.00.01	- 12.12.00.05	5	6/61
12.13.01.01	- 12.13.01.02	5	6/61
12.13.02.01		5	6/61
12.13.03.01	- 12.13.03.02	5	6/61
12.13.04.01	- 12.13.04.05	5	6/61
12.13.05.01		5	6/61
13.01.01		5	6/61
13.02.01	- 13.02.02	5	6/61
13.03.01	- 13.03.03	5	6/61

13.04.01	- 13.04.03	5	6/61
13.05.01	- 13.05.03	5	6/61
13.06.01		5	6/61
13.07.01		5	6/61
13.08.01		5	6/61
13.09.01	- 13.09.04	5	6/61
13.10.01		4	3/61
13.12.01	- 13.12.02	5	6/61
13.13.01	- 13.13.02	5	6/61
13.14.01		5	6/61
13.15.01	- 13.15.04	5	6/61
13.16.01	- 13.16.02	5	6/61
13.18.01	- 13.18.02	5	6/61
13.19.01	- 13.19.05	5	6/61
13.20.01		5	6/61
13.21.01		5	6/61
13.22.01		5	6/61
13.23.01		5	6/61
13.24.01		4	3/61
13.26.01		5	6/61

Total number of current pages: 461

INTRODUCTION

The SHARE 709 System is designed to provide all the advantages of symbolic assembly, and, at the same time, eliminate most of the disadvantages associated with other symbolic assembly systems. For example, the use of most assembly programs permit only two options for making changes to an assembled program:

A. Changes may be made in symbolic form, and inserted into the symbolic source deck, which must then be reassembled. Thus, each time changes are made in a program there is a resulting loss of machine time.

B. The changes may be made in machine language and "patched" into a program. This method, while conserving machine time, does require tedious record keeping to relate machine language patches to the symbolic listing.

The SHARE 709 System provides the advantages of making changes in symbolic form with little increase of machine time over the loading of binary punched cards. The method by which this is accomplished is described in the various parts of this manual as the need arises.

An additional feature of SOS is the facility for listing debugging information in symbolic form, rather than actual or machine language as was previously required.

The SHARE 709 System also includes provisions for:

A. The use of mnemonic operation codes (including a large group of pseudo-operations).

B. Arbitrarily chosen location symbols.

C. Relative and complex addressing.

D. The definition of special purpose macro-instructions for use in a given program.

Further details are given in the discussion which follows of the various parts of SOS.

Although SOS is in reality an integrated system, it has for convenience and easy reference been divided into the following subsystems:

A. The SHARE-Compiler-Assembler-Translator (SCAT). This subsystem has also, for convenience of discussion, been subdivided into three parts:
   1. Compiler

2. Lister
3. Modify and Load

B. The Debugging System

C. The Input/Output System

D. Monitor. (Two monitor programs are described in the manual: the IB Monitor and the SHARE Monitor.)

SCAT

As indicated above, this subsystem will be described in three different parts: Compiler, Lister, and Modify and Load. These three parts together perform all the functions associated with symbolic assembly. In addition, SCAT produces symbolic listings, performs all the mechanics of incorporating modifications into a program, and loads programs for execution.

A. Compiler

The Compiler performs the first part of the assembly of a symbolic source program. This consists of reading symbolic cards, translating the information contained in them into, and producing, a compact binary-coded-symbolic (SQUOZE) form of the program. This SQUOZE form of the program contains all the information supplied in the source program, including remarks cards, and comments from instruction cards. (For detailed information concerning the composition and form of the SQUOZE program, the reader is referred to the appropriate appendix.)

The SQUOZE deck produced by the Compiler may be used in either of two ways:

1. It may be used with a symbolic deck and other SQUOZE decks as input to subsequent Compiler passes, and incorporated with the symbolic deck to form one SQUOZE program as output. This feature makes it possible to write a program in parts and debug each part before combining them.
2. It may be used as input to Modify and Load, which completes assembly and loads the program for execution.

B. Lister

The SCAT Lister is in reality a part of the Modify and Load program. However, since the Lister is used by the Compiler as well as by Modify and Load, and because knowledge of certain features of the listing produced

by SCAT are required for the understanding of the discussion of Modify and Load, the Lister is considered separately in this manual.

The Lister provides the counterpart of an assembly listing of a program. The listings produced include all the symbolic information, including remarks and comments, from the original source program deck as modified by subsequent changes, and the machine language program generated.

C. Modify and Load

Input to Modify and Load is a SQUOZE program and, when necessary, symbolic cards which indicate changes to be made in the program. Modify and Load completes the assembly of the input, incorporates symbolic modifications (if included with the input) and loads the program into storage for execution.

Modify and Load also provides the following features:

1. A new SQUOZE program, which incorporates symbolic changes, can be prepared when desired. (A new listing of the program will also be prepared.)
2. An absolute binary deck can be punched from a SQUOZE program.
3. A new listing of a program in SQUOZE form can be prepared when required.

DEBUGGING SYSTEM

The Debugging System consists of a group of closed subroutines and their associated macro-instructions, which may be written into a program at strategic points, or included as program changes through Modify and Load. These subroutines provide the instructions necessary to print-out symbolic information which will aid in debugging.

INPUT/OUTPUT SYSTEM

The Input/Output System consists of a set of macro-instructions which cause the generation in a program of the instructions necessary for input and output conversion of several types. These macro-instructions are a general purpose type and are intended to be interspersed with machine instructions as necessary to achieve special purpose input/output for a given job.

MONITOR

The input to the Monitor program consists of one or more "job decks." A job deck, as the term is used in this manual, is a program deck to be processed by SOS (symbolic, SQUOZE, or a combination of the two), together with control cards to indicate the functions to be performed on the program, i.e., compile, list, load, etc. The processing of each program is controlled by the Monitor as specified in the control cards included in a job deck.

When a job deck is used as input to the Monitor, the Monitor reads the control card(s) included in the deck, determines the part of SOS required for the processing of the deck and loads the required part. Control is then transferred to the processor loaded by the Monitor. That program then processes input until the end of the job deck is reached, or a new control card is encountered, or an error occurs. When the end of the deck is reached or a new control card is encountered, the Monitor is reloaded into storage and the process is repeated. If an error occurs, the Monitor will print a message indicating the error and will, if possible, continue the processing of the job. If it is not possible for the Monitor to continue, it will skip to the next job.

This manual describes two Monitor programs which can be used with SOS; for detailed information on each, see the appropriate sections of the manual.

SCAT LANGUAGE

A symbolic program consists of an ordered set of symbolic instructions. These instructions are punched into IBM cards, one per card, keeping the same order. The resulting (ordered) deck is then used as input for the Compiler.

An instruction consists of an ordered string of Hollerith characters. (For a list of the allowable characters, see Appendix 1.) An instruction is divided into four fields. From left to right they are:

a. the location field (always 6 columns)
b. the operation field (1 to 7 columns)
c. the variable field (variable length)
d. the comment field (variable length)

The fields are separated by the character "blank."

Since only 72 of the 80 columns of an IBM card can be used for an instruction, the length of an instruction is limited to 72 characters (except for three cases; see page 03.00.44).

Every instruction should specify an operation in the operation field. However, it will often happen, depending on the operation, that the location field and/or the variable field may be left blank. The comment field is provided merely as a means for improving the readability of program listings, and may always be omitted. All comments are retained and included in program listings. Each will appear with its associated instruction(s). For further provisions for writing comments, see page 02.00.11 concerning the use of "*" for remarks.

Operation Codes

Each operation which SOS recognizes, including all of the 709/7090 machine operations, is abbreviated by a mnemonic operation code placed in the operation field. (A blank operation field is equivalent to PZE; see page 03.00.01.) This code is a string of from one to six alphabetic characters. Indirect addressing of an operation, where permitted, is indicated by placing an asterisk (*) at the end of the code. The asterisk is then considered part of the operation code. The operation code may be as long as seven characters when indirectly addressed.

Symbols

A symbol is any string of from one to six non-blank Hollerith characters. At least one character must be non-numeric, and none may be:

+ - * / $ = ,

For example, "A", "A1", "(1)", "12345X" are all legal symbols. However, "123456A", "AbB" where b represents a blank, "A = B", "123", "A,B" are not symbols.

"=" is not allowed because it is reserved for a special purpose. Leading zeros are considered legitimate characters of a symbol.

It is important to distinguish between operation codes and symbols. An operation code must be taken from a fixed set of codes which is the code-vocabulary of SOS. This vocabulary may be expanded, within a given program, by means of the operation MACRO (see page 03.00.27). Symbols, on the other hand, are chosen arbitrarily by the programmer. An operation code is recognized as such by the fact that it appears in the operation field. Symbols cannot appear in the operation field, but may appear either in the location field or the variable field.

Integers

An integer is a string of numeric characters. Integers will usually be interpreted as decimal (base 10), but sometimes as octal (base 8), depending on the operation code in the instruction where they appear.

By this definition, integers are always positive. There are certain restrictions on the maximum size of an integer. These depend on its use, and are described on pages 02.00.05, 02.00.06, and 03.00.18.

The Location Field

The location field of an instruction should either be blank or else contain a symbol. The use of an integer in the location field is an error. It is ignored and flagged as a possible error in the output listing (see page 04.01.02).

The normal purpose of a location symbol is to give a name to an instruction, so that the instruction may be referred to by this name in other instructions of the program. However, for the location symbol of some "pseudo-instructions," this is not true (see page 03.00.06). In fact, placing a given symbol in the location field of some instruction is the only way of establishing this symbol as a name.

Every symbol used in a program should appear exactly once in the location field of some instruction in the program. If it does not appear as a location symbol, the symbol is said to be undefined. If it appears more than once, it is said to be multiply defined and of course is ambiguous as a name. The listing produced by the assembly process will contain, for a given source program, a list of all undefined symbols and also a list of all multiply defined symbols (see page 04.01.04).

Although there is nothing logically wrong in naming an instruction without ever using the name, it is generally desirable to use a location symbol for an instruction only if a symbol is needed to refer to the instruction elsewhere in the program. The reason for this is that the Compiler, processing the source program, constructs a "dictionary" of location symbols in core storage. The amount of core storage which can be allotted for this purpose, although reasonably large, is limited and the extra symbols may cause compilation to be stopped. An error will be flagged on the output listing if this occurs.

The Location Counter

Each entry in the dictionary contains a location symbol and the "value" of the location symbol. This value is an absolute binary number denoting an actual machine cell. The instruction with the given location symbol will finally occupy this cell when the object program is later loaded by Modify and Load (however, see page 04.02.01).

In order to assign the proper value to each location symbol used in the source program, the Compiler uses a special cell called the location counter. The location counter can be initially set to an arbitrary value by the source program (see page 03.00.02).

For each machine instruction processed in the source program, the location counter is increased by 1. Certain pseudo-instructions, on the other hand, may result in no increase or an increase of more than 1 (see page 03.00.04).

Whenever a location symbol occurs with an instruction being processed, the symbol is entered in the dictionary with the current value in the location counter as the value of the symbol. For certain pseudo-instructions, a dictionary entry is not made (see page 03.00.20 and following).

Arithmetic Expressions

A reference by one instruction to another instruction of a program may be made by using the symbolic name (location symbol) of the instruction. For example, suppose that the following instruction appears in a program:

Location	Operation	Variable Field
START	CLA	L(1)

Control may be transferred to this instruction by:

Location	Operation	Variable Field
	TRA	START

However, sometimes a programmer must refer to an instruction that does not have a name. If he wishes, he may go back and give a name to the instruction. This, however, is not necessary. Suppose he wishes to transfer control to the instruction CLA GAMMA in the following sequence.

Location	Operation	Variable Field
ALPHA	TRA	BETA
	CLA	GAMMA
	SUB	L(1)
STGAM	STO	GAMMA
	TPL	DELTA

This may be done by either of the following instructions:

	TRA	ALPHA+1
	TRA	STGAM-2

Thus, an unnamed instruction may be referred to by using the name of an instruction somewhere in its vicinity and adding or subtracting an integer or symbol.

The combination "ALPHA+1" or "STGAM-2" is called a relative expression. A relative expression is the sum or difference of not more than two symbols or constants. The presence in an expression of a $, *, /, or more than one plus or minus sign defines the expression as complex. A negative symbol or constant in an address field is treated as relative, i. e., is treated as zero minus the symbol or constant.

Relative expressions should be used with care, since a later insertion or deletion of instructions between X and X+n (or X-n) changes the instruction to which "X+n" (or "X-n") refers. This is the outstanding disadvantage of a so-called "relative coding" system. It is theoretically possible, though rarely advisable, to use only one name in an entire program and make all references relative to that name.

Occasionally it may be found useful to combine symbols and integers in more complicated ways than in a relative expression. For example:

```
A*B
A/B
A*B/C+D*2-E
```

The Compiler recognizes and correctly interprets, according to the ordinary rules of arithmetic, any meaningful arithmetic expression which can be constructed from symbols, integers, and the four arithmetic operations:

+ (addition)
- (subtraction)
* (multiplication)
/ (division)

Since left and right parentheses can occur as legitimate characters in a symbol, they cannot be used as grouping marks in an expression. Thus, "A multiplied by (B+C)" must be written as "A*B+A*C". Most, but not all, expressions using parentheses can be written without parentheses. Note that A/(B+C) cannot be written without using parentheses, and hence cannot be used.

The evaluation of an arithmetic expression is carried out as follows: First, all symbols must be defined and all integers appearing in the expression are taken as decimal. Integers must be less than 2^{35}. The whole expression will be indicated as an error on the output listing for either of the following violations:

a. Any of the symbols in the expression are not defined.
b. An integer exceeding $2^{35}-1$ occurs in the expression.

The evaluation proceeds by first scanning the expression from left to right and performing all multiplications and divisions. (In division, only the integral part of the quotient is retained; the remainder is discarded.) Then another left-to-right scan is made and all additions and subtractions are performed.

All arithmetic is carried out using 35 binary bits and a sign. If, at any point in these operations, the numeric part of the result exceeds $2^{35}-1$, only the rightmost 35 binary bits are kept, i.e., the number is reduced modulo 2^{35}. If the result, R, after the second scan is negative, R is replaced by the 2s complement of the absolute value of R, i.e., R becomes $(2^{35}-|R|)$. When the expression has been completely processed, the value taken for the expression is the rightmost 15 bits of R, i.e., R is reduced modulo 2^{15}.

Ordinarily, none of the computations should result in more than 15 bits, but the expression is still considered meaningful if 15 bits is exceeded.

Note the term expression is meant to include not only relative and complex expressions, but also simple expressions consisting of a single symbol or integer. Thus "A" and "7" are expressions, as are "-A" and "-7".

According to the rules given above, "-7" would yield the 15-bit value $(77771)_8$.

The Use of "*" as a Term

An important and extremely useful convention is another meaning for the asterisk (*) in an expression. When the Compiler encounters an "*" as a term in an expression (i.e., in that part of the expression where a symbol or integer should logically be), it interprets the "*" as the current value of the location counter. For example, assume the location counter contents are 155 when the following instruction is processed:

Location	Operation	Variable Field
	TRA	*+2

Then the relative expression "*+2" is evaluated as 155+2=157, so that a transfer to the second instruction after the TRA instruction is indicated. An "*" employed in this way represents a kind of "floating address," and by it judicious use in a program one can often avoid introducing superfluous names. For instance, TRA *-2 always means "transfer control to the second instruction preceding this instruction" and that instruction need not be named. There is no confusion between the use of "*" as a term and its use to indicate multiplication in an expression, e.g., the expression "***" means "the current value of the location counter multiplied by the current value of the location counter."

Sense Indicator Instructions

Special provisions are made in the SCAT language for dealing with sense indicator instructions. Unlike the 15-bit address of ordinary instructions, a sense indicator instruction has a "mask" of 18 bits, which is really a string of 18 independent logical (Boolean) bits.

The mask field (which corresponds to the address field of an ordinary symbolic instruction) of a sense indicator instruction written in the SCAT language must contain a single octal number or a single symbol. If this condition is violated, the mask will be evaluated as zero and an error will be indicated on the output listing. The same treatment is given to an integer mask for any of the following violations:

a. The integer representation contains the character "8" or "9" so that it is clearly not octal.

b. The integer value exceeds $2^{18}-1$.

c. The integer representation uses more than 12 characters.

Boolean Symbols and Expressions

If a symbol is used in the mask field of a sense indicator instruction, this symbol should be defined by means of a special pseudo-operation whose sole purpose is to define such symbols. Such symbols are called "Boolean" symbols and have an 18-bit range, as distinct from "ordinary" symbols with a 15-bit range.

An expression involving Boolean symbols and/or octal integers is called a "Boolean" expression. A Boolean expression which does not consist simply of an octal number or a (Boolean) symbol must occur only in a BOOL pseudo-instruction (see description of BOOL, page 03.00.08). An expression should never be "mixed," i.e., if one of the symbols in an expression is Boolean or one of the integers is octal, then all symbols should be Boolean and all integers should be octal in this expression. Similarly, if there is one ordinary symbol or decimal integer in an expression, then all symbols should be ordinary and all integers decimal.

The rules for constructing a Boolean expression resemble those for an arithmetic (ordinary) expression. However, the meanings of the four operations, "+", "-", "*", and "/" are Boolean rather than arithmetic. They are simply:

"+" ("or", "inclusive or", "union")

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1

"*" ("and", "intersection")

0 * 0 = 0
0 * 1 = 0
1 * 0 = 0
1 * 1 = 1

"-" ("exclusive or", "symmetric difference")

0 - 0 = 0
0 - 1 = 1
1 - 0 = 1
1 - 1 = 0

"/" ("ones complement", "complement", "not")

/ 0 = 1
/ 1 = 0

Note that the operations "+", "-", and "*" are ordinarily operations connecting two terms, whereas the operation "/" ordinarily involves one term. However, by convention, "A/B" is taken to mean "A*/B." Thus the table for "/" as a two-term operator is:

0/0 = 0
0/1 = 0
1/0 = 1
1/1 = 0

Other conventions are:

+A = A+ = A -A = A- = A *A = A* = A A/ = 0	one operand missing
+ = 0 - = 0 * = $(777777)_8$ / = 0	both operands missing

The above tables completely define the four Boolean operations for one-bit quantities and hence for the 18-bit Boolean quantities in SCAT. For if A and B are 18-bit Boolean quantities, each can be regarded as a string of 18 independent one-bit quantities. Thus C = A*B (for example) is simply obtained by 18 parallel, independent "AND" operations, where each "AND" is performed between one bit of A and the corresponding bit of B.

For example,

$$(123456)_8 * (234567)_8 = (020446)_8$$
$$(123456)_8 + (234567)_8 = (337577)_8$$
$$(123456)_8 - (234567)_8 = (317131)_8$$
$$/ (123456)_8 = (654321)_8$$

The evaluation of a Boolean expression proceeds as described for an arithmetic expression, but the four operations are interpreted as Boolean in the sense defined above, rather than arithmetic. First, the operations "*" and "/" are carried out from left to right, and then the operations "+" and "-". Eighteen-bit Boolean arithmetic is used in all stages, and the final value of the expression is 18-bit Boolean.

For restrictions on integers in a Boolean expression, see page 03.00.10.

The Variable Field

In order to specify a 709/7090 machine instruction completely, the programmer can, and sometimes must, specify a certain combination of address, tag, and decrement (or count), depending on the operation used in the instruction. For example, a TIX instruction requires an address, tag, and decrement; LXD requires an address and tag but must not have a decrement; CLA requires an address and may have a (operative) tag, but must not have a decrement; PXD requires a tag, must not have a decrement, but may have an (inoperative) address; CLM must not have any address, tag, or decrement; etc.

The complete details for all 709/7090 operations can be found in the 709 and 7090 reference manuals.

The address A, and/or the tag T, and/or the decrement D, of an instruction are specified in the instruction's variable field, in that order (i.e., A, T, D). The subfields A, T, D are separated by commas. For example, the following instruction specifies an address ALPHA, tag of 4, decrement of 1.

Location	Operation	Variable Field
	TIX	ALPHA, 4, 1

The end of the variable field is signalled by the occurrence of the first blank character in scanning from left to right. Hence, there must be no blanks left between the subfields of the variable field, nor within the subfields themselves. The sole exception to this is the pseudo-operation BCI (see page 03.00.16). For those operations which require a tag but no address, the address zero should be used, e.g.,

Location	Operation	Variable Field
	PXD	0, 4

Two very useful conventions in specifying variable fields are provided:

a. If one or more of the subfields of a variable field is to be zero, the programmer may omit writing the "0" character and use only the separating comma. For example:

	Operation	Variable Field
	TXL	ALPHA, 0, 5
can be written:	TXL	ALPHA, , 5
	TXH	0, 0, 5
can be written:	TXH	, , 5
	PXD	0, 4
can be written:	PXD	, 4

Notes: 1. In a subfield which is not the last subfield of the variable field, never replace the "0" with a blank, since the blank signals the end of the variable field.

2. If zero subfields are omitted, messages are printed to indicate that they are possible errors, and zeros are inserted in the subfields.

b. If the programmer wishes to specify the value 0 in the last subfield, or subfields of the variable field, he may do so by omitting these fields along with their separating commas. For example:

	Operation	Variable Field
	TXL	ALPHA, 4, 0
can be written:	TXL	ALPHA, 4
	TXL	ALPHA, 0, 0
can be written:	TXL	ALPHA
	TXH	0, 0, 0
can be written:	TXH	
	PXD	0, 0
can be written:	PXD	

Certain pseudo-instructions in SOS require more than three subfields in the variable field. The same convention applies to these; i. e., if the last n subfields are to contain zeros, they may all be omitted along with their separating commas. The restrictions on the address (mask) field of sense indicator instructions have already been stated. With the exception of this special case, the subfields of the variable field of a 709/7090 machine instruction may contain any arithmetic expression. For instance:

Location	Operation	Variable Field
	TIX	A*B+C, D/E-F, G*29+H

is perfectly legal, so long as the symbols A, B, ..., H are all defined and are arithmetic.

The use of Boolean symbols in other than a sense indicator instruction is not strictly prohibited but can result in errors which will not be flagged in the output listing. A Boolean symbol used by itself as an address or decrement, can change the tag or operation field, respectively. For example, if A and B are Boolean symbols with value $(777777)_8$, then the instruction TIX A, 1, B will result in the absolute machine word $(7\ 77777\ 7\ 77777)_8$ which has an undesired prefix and tag of 7. Such an error will not be detected and indicated to the programmer. However, if A and B (in the example) did not exceed 15 bits (i. e., were less than 2^{15}), the

correct prefix and tag would result. A Boolean symbol occurring in a relative or complex expression will be detected and indicated as a possible error on the output listing. The expressions in the address and decrement subfields will be evaluated as previously described, i. e., the rightmost 15 bits of the results will be placed in the address or decrement. However, only the rightmost three bits of a result will be placed in the tag, i. e., the result is reduced modulo 2^3. Certain pseudo-operations in SOS will be described later which require variable fields in different forms. In an instruction using such operations, the subfields of the variable field have special restrictions. The rules for specifying the variable field depend on the given pseudo-operation. These rules are set down in the following sections with the description of the pseudo-operation.

Comments Field

Any non-blank characters found after the blank that signals the end of the variable field will be regarded as comments and will appear unaltered in the output listing.

The start of this field must be separated from the end of a preceding non-blank variable field by at least one blank. However, if the variable field is blank, the comments field must not start to the left of column 17. It ends in or before column 72. This field may contain blanks. It does not affect execution of the instruction, but it is retained by SOS for inclusion in program listings.

Remarks

Any card with "*" in column 1 is called a "remarks" card. When such a card is encountered, columns 2 through 72 are treated as commentary. This commentary is saved and printed out as a single line on the output listing, exactly as it is written. Such a card has no other effect on the processing of the source program.

Remarks cards can be extremely useful in producing a readable output listing. One or more such cards might be placed at the beginning of the program for different descriptive purposes, or inside the program to include pertinent information for the reader of the program.

THE COMPILER

Classification of SOS Operations

Every operation in the SOS language belongs to one of two classes: 709 machine operations (e.g., CLA, LXD, RDS, or IOCD) and non-machine operations. A non-machine operation is called a "pseudo-operation." Instructions using pseudo-operations are called "pseudo-instructions."

The 15 operations associated with the Data Synchronizer Channels are put in the same class as ordinary operations and the "commands" in which these operations appear are specified in the same way as ordinary machine instructions.

Note that SOS provides the following 12 codes, which can be used in a convenient mnemonic way to specify only the prefix (leftmost three bits) of the instruction, accompanied by the usual variable field pattern of address, tag, decrement.

PZE	(Plus zero)	PTH	(Plus three)
MZE	(Minus zero)	MTH	(Minus three)
PON	(Plus one)	FOR	(Four)
MON	(Minus one)	FVE	(Five)
PTW	(Plus two)	SIX	(Six)
MTW	(Minus two)	SVN	(Seven)

Machine Operations

A machine instruction (i.e., an instruction using a machine operation) always generates one 36-bit binary machine word in the object program. The rules for specifying the location field and the variable field of a machine instruction have already been given in Section 02.

Pseudo-Operations

Unlike machine instructions, some pseudo-instructions may generate more than one machine word in an object program or may generate no words at all. The pseudo-operations of SOS have a variety of functions which will be seen in the following pages.

Pseudo-Operations Which Control the Location Counter

The function of the following three pseudo-operations is principally to control the contents of the location counter (see page 02.00.03).

A. ORG (Origin)

If a programmer wishes the origin of his program (i.e., the location of the first word in his object program) to be $(3490)_{10}$, he may simply preface his source program with the pseudo-instruction:

Location	Operation	Variable Field
	ORG	3490

No word is generated in the object program by this instruction. Its effect is to cause the Compiler to set the location counter to the value $(3490)_{10}$. If the instruction immediately following the ORG is ALPHA CLA BETA, then the symbol ALPHA will receive the value $(3490)_{10}$ when placed in the dictionary. The binary word which results from the CLA BETA part will be ear-marked for location $(3490)_{10}$, and subsequently assigned to this location by Modify and Load. (However, see page 04.02.01.)

ORG instructions may appear anywhere in the program. Moreover, the expression in the variable field need not be an integer as in the above example. It may be any arithmetic expression. For example:

Location	Operation	Variable Field
	ORG	ALPHA+BETA*GAMMA-1

Thus, the variable field of an ORG instruction consists of a single subfield. If more than one subfield is used (e.g., ORG A, B), only the first, in this case A, will be used. The remaining subfields will be ignored, and an error flagged in the program listing. The effect of the above instruction (and, in general, any ORG instruction) is to cause the location counter to be set to the value of the expression in the variable field. Of course, any symbols used in the variable field expression of the ORG must be eventually capable of receiving values, i.e., they must be defined in the sense given on page 02.00.02. However, they need not have been assigned values before they are used. In the above example, ALPHA and/or BETA and/or GAMMA need not have appeared as location symbols before the ORG instruction itself.

A location symbol can meaningfully appear in an ORG instruction, for example.

Location	Operation	Variable Field
ALPHA	ORG	3490

This instruction will cause the symbol ALPHA to be entered in the dictionary of symbols with the associated value of $(3490)_{10}$. If the variable field had been a symbol or some non-simple expression, then the value of ALPHA in the dictionary would have been the value of that symbol or expression. Note that, in this example, if the next instruction were

Location	Operation	Variable Field
	CLA	BETA

then the same effect could also have been obtained by writing

Location	Operation	Variable Field
	ORG	3490
ALPHA	CLA	BETA

On the other hand, if the programmer were to write

Location	Operation	Variable Field
ALPHA	ORG	3490
GAMMA	CLA	BETA

Then ALPHA and GAMMA would both be entered into the dictionary, with the value $(3490)_{10}$. Another way of achieving such an effect will be seen in the pseudo-operation EQU.

The Compiler does not require the presence of an ORG at the beginning of the source program, nor anywhere within the source program. If the programmer fails to use an ORG instruction to set the location counter to an initial value, the Compiler will assume that the program is to begin at a location to be determined later.

Lower core storage will ordinarily contain a part of the SHARE Monitor. For this reason, a source program with no initial ORG instruction will not be started at location $(00000)_8$. Instead, the initial location is assigned by the monitor as the lowest available location.

Since a certain part of lower core storage is normally required for the functioning of SOS, the programmer should not specify a program origin which is so low that the object program will conflict with this required part. An error will be indicated on the output listing if a program origin which is too low is specified.

B. BSS (Block Started by Symbol)

A programmer will often need to reserve a block of one or more words of core storage for such purposes as "erasable storage," input and output buffers, etc. If, for example, in writing his source program, he needs to reserve the next 50 words, he may write:

Location	Operation	Variable Field
	BSS	50

When encountered, this instruction will cause the location counter to be increased by $(50)_{10}$.

If it is desired to given the name ALPHA to the first word of the block, the instruction can be written:

Location	Operation	Variable Field
ALPHA	BSS	50

With BSS, it is not possible to associate location symbols with any words of the reserved block except the first word. This, however, can be accomplished by EQU.

Like ORG, the variable field of a BSS instruction may contain any arithmetic expression, for example:

Location	Operation	Variable Field
ALPHA	BSS	BETA/GAMMA+4*DELTA-3

The effect of the above instruction is to enter ALPHA in the dictionary with the current value of the location counter, and then to increase the location counter by the value of the arithmetic expression in the variable field. The general comments about ORG also apply to BSS.

Unlike ORG, the variable field of a BSS instruction may have a second subfield which can be used to provide information for later use by the Debugging System. The programmer can specify this information by placing, in the second subfield, an alphabetic format code which he can choose from one of the following list of seven codes:

Code	Format Intended
C	Command (DSC control word)
F	Floating point number
H	Hollerith (binary coded decimal) information
O	Octal integer
S	Symbolic instruction
V	Variable Format
X	Fixed point number

For example, suppose a programmer writes the following instruction:

Location	Operation	Variable Field
ALPHA	BSS	50, F

By use of the F, he specifies that the 50 words in the block beginning at ALPHA are to be interpreted as floating point numbers. Subsequently, whenever the Debugging System is to dump information from this block, the words will appear in the output as floating point numbers.

If the programmer does not intend to use the Debugging System, he will, of course, have no need for specifying a second subfield in a BSS instruction.

For further information on the meaning and use of the seven format codes, see page 06.01.02 and following.

C. BES (Block Ended by Symbol)

This pseudo-operation has exactly the same properties as BSS, except that when it is used with a location symbol, the symbol is associated with the first word following the reserved block (rather than with the first word of the block). For example, suppose a programmer writes

Location	Operation	Variable Field
ALPHA	BES	50
	CLA	BETA

Then the symbol ALPHA becomes associated with the instruction CLA BETA. Thus, the programmer could with equivalent results, have written

Location	Operation	Variable Field
	BES	50
ALPHA	CLA	BETA

On the other hand, if the programmer writes

Location	Operation	Variable Field
ALPHA	BES	50
GAMMA	CLA	BETA

then ALPHA and GAMMA would both be entered into the dictionary, each with the value of the location counter at the time the CLA instruction is processed.

Thus, the effect of the instruction ALPHA BES 50 is to increase the location counter by $(50)_{10}$ and then to enter ALPHA into the dictionary with the resulting value in the location counter. Note that ALPHA BES 50 is equivalent to ALPHA ORG *+50.

As with BSS, a second subfield can be used in the variable field of a BES instruction to specify one of the seven possible formats for the reserved block.

Pseudo-Operations for Relating Symbols

The following three pseudo-operations serve the sole function of equating two or more symbols, or of assigning a value to a symbol.

A. EQU (Equals)

When writing a source program, a programmer may want to use a name for something, the precise nature of which he does not yet know. For example, he might wish to refer to some instruction which he has not written down, and does not yet want to decide on the name of this instruction. For example, he may be at a point such as:

Location	Operation	Variable Field
	CLA	ALPHA
	SUB	BETA
	TZE	

where he knows what he wants to do next if c(AC) is not zero but he would like to leave the address of the TZE instruction unspecified until later. He may, of course, leave this address blank temporarily. He may want to write something arbitrary such as TZE X1, just to make the instruction complete (especially if his program is being punched in batches). Later when he decides what the instruction X1 is to be, he may be satisfied to use the name (location symbol) X1 for this instruction. If he is not satisfied with the name X1, he may go back and replace it with the symbol he has decided on wherever he has used it. However, for one reason or another, this replacement may be impractical, e.g., if instruction cards referring to X1 have already been punched.

In the above example, suppose the programmer has actually written X1 and later decides that what he has called X1 should be called NOGOOD. Then he could simply write

Location	Operation	Variable Field
	CLA	ALPHA
	SUB	BETA
	TZE	X1
	.	.
	.	.
	.	.
NOGOOD	PXA	0,1
X1	EQU	NOGOOD
	AXT	4,1
	.	.
	.	.
	.	.
	.	.

where the PXA and AXT instructions are two instructions of the NOGOOD subroutine. X1 EQU NOGOOD specifies that the symbols NOGOOD and X1 are to be equivalent.

The above situation is one of many examples where the pseudo-operation EQU can be used very conveniently and effectively. Even if, in this example, the name X1 had occurred in many places, before or after the EQU instruction, X1 would still be equated to the symbol NOGOOD.

Moreover, the EQU instruction could have been put anywhere in the program, before or after the instruction named NOGOOD. The general comments about ORG also apply to EQU. As with ORG, the variable field of an EQU instruction can contain any arithmetic expression, subject to the restrictions described under ORG. For example, one can write

Location	Operation	Variable Field
ALPHA	EQU	BETA*GAMMA-DELTA/9+17

The effect of the above instruction is to enter ALPHA in the dictionary with the value of the arithmetic expression in the variable field. Unlike ORG, BSS, and BES, the pseudo-operation EQU does not affect the value in the location counter.

An EQU instruction is meaningless if it does not have a location symbol. An EQU instruction without a location symbol will have no effect and an error will be indicated on the output listing.

Like ORG, the variable field of an EQU instruction should contain only one subfield, namely an arithmetic expression. If more than one subfield appears in the variable field of an EQU instruction, only the first will be used. The remaining subfields will be ignored and an error indicated on the output listing.

In the example given above, the variable field of the EQU instruction in each case contains a symbolic expression. The variable field expression may also be completely numeric. For example:

Location	Operation	Variable Field
	LLS	SHIFT
	.	.
	.	.
	.	.
SHIFT	EQU	35

Here, the symbol SHIFT receives the value 35_{10} by virtue of the EQU instruction.

B. SYN (Synonym)

In the SCAT language, SYN is simply another code for EQU, and they may be used interchangeably. The reason for providing two codes is purely historical.

C. BOOL (Boolean Equals)

If the programmer uses the Sense Indicators in a program, he may often need to write instructions in which the 18-bit address ("mask") corresponds to the 18 leftmost or 18 rightmost bits of this special register. If he cannot conveniently predetermine what particular sense indicator positions he would like to use, he might write, for example:

Location	Operation	Variable Field
	RIR	SENSX

Later, when he has decided that SENSX should be, say, the rightmost four positions (i. e., positions 32, 33, 34, and 35 of the Sense Indicator register), he can write:

Location	Operation	Variable Field
SENSX	BOOL	17

where the 17 is interpreted as an octal number equivalent to $000\ 000\ 000\ 000\ 001\ 111_2$. The effect of the instruction SENSX BOOL 17 is similar to the effect of SENSX EQU 17. However, they differ in two important respects:

1. For EQU, the 17 would be interpreted as decimal, while for BOOL, the 17 is taken as octal.

2. For BOOL, the symbol SENSX would be entered into the dictionary with a special indication that this symbol is "Boolean," while with EQU, the symbol SENSX receives no such special indication.

In the above example, the variable field of the BOOL instruction contains an octal integer, which is a special case of a Boolean expression (see Section 02). The variable field of a BOOL instruction can in general contain any Boolean expression. For example, one might write:

Location	Operation	Variable Field
A	BOOL	B*C+D-707070

where the variable field expression is Boolean, i.e., B, C, D are Boolean symbols, 707070 is an octal integer, and the operations (*, +, and -) are Boolean (see Section 02).

It is not necessary that the symbols used in the variable field expression should already have received values when the BOOL instruction is first encountered. However, the symbols should all be defined in the sense that each symbol in the expression should occur once in the location field of some other BOOL instruction, since all symbols used must be Boolean and a Boolean symbol can be defined by a BOOL instruction.

As in ORG and EQU, the variable field of a BOOL instruction should contain only one subfield. If more than one subfield appears, only the first will be used. The remaining subfields will be ignored and an error indicated on the output listing.

The variable field expression of a BOOL instruction will be evaluated as zero, and an error will be indicated on the output listing, for any of the following reasons:

1. An integer appears in the expression using the character 8 or 9, so that the integer is clearly not octal.

2. An integer appears in the expression, and the value of the integer exceeds $2^{35}-1$.

3. An integer appears in the expression and the representation of the integer uses more than 12 numeric characters.

If the variable field of a BOOL instruction uses some non-Boolean (i. e., ordinary) symbol, this symbol will be treated as though it were Boolean and a possible error will be indicated on the output listing.

Pseudo-Operations for the Introduction of Data

The following four pseudo-operations can be used to introduce decimal, octal, binary-coded-decimal, or mixed data from the source program into the object program.

A. DEC (Decimal Data)

This pseudo-operation causes the decimal numbers specified in the variable field to be converted to binary, and assigned to successive locations beginning with the current value of the location counter. If there is a location symbol, it is entered in the dictionary with the current value of the location counter. The first (i. e., leftmost) decimal number specified in the variable field can be referred to by this location symbol.

Example:

Suppose the value in the location counter is $(3900)_{10}$, when the following instruction is encountered:

Location	Operation	Variable Field
CONST	DEC	1,-3,5,-7,9

The effect of this instruction is to enter the symbol CONST in the dictionary with the value 3900. The five integers 1,-3,5,-7,9 are converted to binary and assigned to locations 3900, 3901, 3902, 3903, 3904, respectively. The value of the location counter upon completion will be 3905_{10}.

If the programmer desires to add the integer 9 to the contents of the AC, he may now write:

Location	Operation	Variable Field
	ADD	CONST+4

Every decimal number must be represented by a string of characters from the following set of 15 characters:

0	(numeric characters)	+	(plus sign)
1		-	(minus sign)
2		.	(decimal point)
3		E	(exponent)
4		B	(binary point)
5			
6			
7			
8			
9			

In order to represent a valid decimal number, the composition of the string must satisfy the rules given below.

There are three types of decimal numbers which can be specified in the variable field of a DEC instruction:

1. Integers
2. Fixed point numbers
3. Floating point numbers.

A given decimal number is recognized as belonging to one of these three types by the representation of the number itself.

The sign of any decimal number is always specified by the first character, "+" or "-". If no initial "+" or "-" is given, the sign is assumed to be "+".

Integers are represented by a string of numeric characters only (with possibly a leading sign character). For example, -31 is an integer, but -31. is not. An integer is converted to a 35-bit binary number with sign and stored in positions S, 1-35 of the 709 binary word cell, the position of the binary point in the cell being at the right-hand end of the word. For instance, -31 would convert to $(400\ 000\ 000\ 037)_8$.

The term "integer" as used here differs from the term "integer" defined in Section 02, where it essentially means a non-negative whole number to be dealt with according to the rules for evaluating a symbolic expression. For example, the instruction PZE -1 converts to $(000\ 000\ 077\ 777)_8$, while DEC -1 converts to $(400\ 000\ 000\ 001)_8$. On the other hand, PZE 1 and DEC 1 both convert to $(000\ 000\ 000\ 001)_8$. The reader can always determine which meaning of integer is intended by considering the context.

A fixed point number or a floating point number may have a decimal point and/or a signed decimal scale factor. The scale factor is indicated by the character E and is placed after the principal part of the number. If the decimal point is not present, it is assumed to be at the right-hand end. For example, the strings 314159. E-5 and 314159E-5 each represent the number $(3.14159)_{10}$ (i.e., 314159.0×10^{-5}). The sign of the scale factor may be omitted if it is +. Thus, for example, the number $(3.14159)_{10}$ might be presented by .314159E+1 or .314159E1. The character E, which indicates a decimal scale factor, may also be omitted, but only if the scale factor is signed. Thus, $(3.14159)_{10}$ could be represented by 314159-5 or .314159+1 (but not, of course, by .3141591).

A number is recognized as being floating point and will be converted to a normalized floating point binary number if and only if its principal part contains a decimal point and/or it has a decimal scale factor, but not if it uses the character B. Thus, all the examples given in the previous paragraph (i.e., 314159. E-5, 314159E-5, .314159E+1, .314159E1, 314159-5, .314159+1) are floating point numbers. Another representation of 3.14159_{10} in floating point form would be simply the string 3.14159.

A number is recognized as being fixed point if and only if the string representing it contains the character B. The B must be followed by a signed integer (as usual, if the sign is +, it may be omitted). This integer specifies the position of the binary point in the cell in which the fixed point binary number resulting from the conversion is to be stored. The B-integer is used to count from the left-hand end of the binary word cell from left to right. Thus, B = 0 specifies a binary point between positions S and 1, and B = 35 specifies a binary point immediately to the right of position 35. The B-integer can thus be thought of as the number of integral places.

It is not necessary for the B-integer to be positive. A negative value means the binary point is positioned outside of the left-hand end of the cell. The B-integer may also exceed 35, e.g., 2.0B+36 would convert to $(000\ 000\ 000\ 001)_8$. Here the binary point is one position outside the right-hand end of the word cell.

Note that it is possible to lose bits on the left-hand end of the number if the B-integer is improperly chosen. For example, 1.5B0 results in the loss of the integral bit of the converted result $(1.1)_2$. If such a loss on the left occurs, the number will be taken as zero and an error indicated on the output listing.

It is also possible, and generally unavoidable, to lose bits on the right for a fixed point decimal number. For example, 3.0B+36 would cause the rightmost 1-bit in $(11)_2$ to be lost, because the word cell has only 35 bits. This loss could have been avoided by specifying a B-integer of 35 instead

of 36. However, numbers like $(0.4)_{10}$ which equals $(.314631463146....)_8$ do not have a finite binary representation, so that 0.4B0 will result in an unavoidable loss of bits on the right. If bits are lost from the right-hand end, no error will be indicated, and the best possible result will be obtained.

A fixed point number may have a decimal point and/or a decimal scale factor; e.g., 314159E-5B2, 314159.E-5B2, and 3.14159B2 are all fixed point numbers. However, the presence of the B alone is sufficient to define a number as fixed point; e.g., 32B6 is fixed point.

Note that an integer can always be represented equivalently by using a fixed point representation with a B-integer of 35, e.g., -31 is equivalent to -31B35.

If the string representing a fixed point number contains both B and E, the B-part and the E-part should both be placed after the principal part of the number, but their relative order is unimportant. For example, 314159B2E-5 is equally as acceptable as 314159E-5B2.

A 709/7090 word cannot accommodate integers whose absolute value exceeds $2^{35}-1$ or floating point numbers whose absolute value exceeds approximately 10^{38}. Hence no decimal numbers outside of these ranges should be specified in a DEC instruction.

If an integer exceeding $2^{35}-1$ or a floating point number exceeding approximately 10^{38} (in absolute value) is specified in a DEC instruction, the number will be taken as zero and an error indicated on the output listing. There is no restriction on the number of numeric characters which may be used in representing a number, so long as these rules are followed.

The number of decimal numbers which may be specified in a single DEC instruction is limited only by the number of subfields which can be written in the variable field (this depends on the length of the variable field and the lengths of the strings representing the numbers).

Any combination of types of numbers is allowable in a single DEC instruction; for example, CONST DEC 2,3,1,0.9B5. However, it is generally preferable for all numbers in a given DEC instruction to be the same type.

The reason for this is that a DEC instruction location symbol has an associated format code which is entered into the dictionary along with the location symbol itself. This code is retained and later used by the Debugging System to determine how numbers are to be interpreted, whether as fixed point (format code X) or as floating point (format code F). The format code used is determined by the type (format) of the first decimal number appearing in the variable

field of the DEC instruction (in the above example, this would be X). The remaining decimal numbers in the variable field will be assumed to be of the same format, and if they are not, they will not appear on the debugging output list as originally written.

B. OCT (Octal Data)

This pseudo-operation causes octal integers in the variable field to be converted to binary, and assigned to successive locations beginning with the current value of the location counter. If there is a location symbol, the symbol is entered in the dictionary with the current value of the location counter. Thus, the octal integer specified in the leftmost subfield of the variable field can be referred to by this symbol.

For example, suppose the value of the location counter is 3900_{10} when the following instruction is encountered:

Location	Operation	Variable Field
OCTDAT	OCT	777777777777, -77,66,-55,44

The effect of this instruction is to enter the symbol OCTDAT in the dictionary and to convert the five octal integers in the variable field to binary. The numbers are assigned to locations 3900, 3901, 3902, 3903, 3904, respectively, leaving the value of the location counter at 3905_{10}. Thus, the symbol OCTDAT+2, for example, may be used to refer to the number 66_8 specified by this instruction.

Every octal integer must be represented by a string of characters from the following set of 10 characters:

0 1 2 3 4 5 6 7	numeric characters
+	(plus sign)
-	(minus sign)

An octal number may consist of up to 12 numeric characters and may be preceded by a sign.

If more than 12 digits are used in representing the number, or if an 8 or 9 is included, then the number is converted as zero and an error is indicated in the output listing.

The sign of the octal number provides an easy way to specify the sign of the binary result. + and - specify a 0-bit or 1-bit, respectively, in the sign position. An alternative way of specifying the leftmost bit of the binary result is by using twelve octal digits in representing the number. If the leftmost octal digit is 4, 5, 6, or 7, this implies that the leftmost binary bit of the result is 1 (i.e., a - sign). For small octal integers such as -77_8, it is easier to write -77 than to write 400000000077 (these two strings are equivalent and convert to $(400\ 000\ 000\ 077)_8$). However, the 12-digit, signless representation can always be used for any 36-bit binary number and is preferable if 12 octal digits must be used in any case.

If a sign and 12 octal digits are used to represent an octal number, redundancies or inconsistencies arise (unless the leftmost octal digit is regarded as base 4 instead of base 8). For example, +700000000000 and -300000000000 are inconsistent, while -700000000000 and +300000000000 are both redundant. If an octal number is represented by 12 octal digits and an explicit sign, and the leftmost digit is 4, 5, 6, or 7, then an error is indicated in the output listing. The following conventions are then used in conversion:

sign and leftmost octal digit	binary result (bits S, 1, 2)
+0	000
-0	100
+1	001
-1	101
+2	010
-2	110
+3	011
-3	111
+4	100
-4	100
+5	101
-5	101
+6	110
-6	110
+7	111
-7	111

C. BCI (Binary Coded Information)

One to ten words of binary coded information can be provided in the object program by means of a BCI instruction. The variable field of a BCI instruction has two subfields.

The first subfield specifies the number of words of information. This first subfield must contain a number from 1 through 9, or else consist only of the comma which is used to separate the two subfields. In this case the number of words of information is taken as ten.

The second subfield specifies the BCI information and must consist of a continuous string of Hollerith characters, including comma, blank, etc. The number of characters taken from the second subfield is six times n, where n is the specification of the first subfield.

If a BCI instruction has a location symbol, the symbol is entered in the dictionary with the current value of the location counter. The n words of Hollerith information are converted to n words, consisting of six binary-coded-decimal characters each. These n words are assigned to the n successive locations beginning with the current value of the location counter. Thus, the first word of BCD information can be referred to by the location symbol name.

For example, suppose the location counter value is 3900_{10} when the following instruction is encountered:

Location	Operation	Variable Field
IDENT	BCI	3, THISbISbAbBCI

(where "b" indicates the character, blank)

The effect of this instruction would be to enter the symbol IDENT into the dictionary with the value $(3900)_{10}$, and to store the BCD representation of the three Hollerith words. The words would be stored as follows:

3900	THISbI
3901	SbAbBC
3902	Ibbbbb

where b represents a blank, i. e., the core storage BCD character 110000_2.

The value of the location counter would be 3903_{10}. The programmer could then refer to the first word in, say, an IOCD command by means of the symbol IDENT.

D. VFD (Variable Field-Definition)

The preceding three pseudo-operations describe means of introducing decimal, octal, or Hollerith information into the object program in units of words. However, it will often be found desirable to prescribe information in smaller units of a word.

It is, of course, possible (though sometimes not convenient) to use the prefix, decrement, tag, address format of a word by specifying a proper machine instruction, e.g., SVN 9,1,127 results in a prefix (bits 0-2) of 7, a decrement (bits 3-17) of 127, a tag (bits 18-20) of 1, and an address (bits 21-35) of 9. Even in such cases, the use of VFD may be found easier, because its basic unit of information is a bit, instead of a word. It allows the programmer to specify a continuous string of bits, from left to right, starting at the left-hand end (0-bit) of a word.

As an example, suppose the programmer would like to break up a single 36-bit word into four subfields: positions 0-9, positions 10-14, positions 15-29, and positions 30-35. Suppose also that he would like to have placed in these four subfields, respectively, the following four pieces of information:

1. The binary equivalent of the decimal integer 895.
2. The binary equivalent of the octal integer 37.
3. The binary value of the symbol ALPHA.
4. The binary coded value for the character C.

Then he may simply write:

Location	Operation	Variable Field
	VFD	10/895,O5/37,15/ALPHA,H6/C

The four subfields of the variable field of the above instruction are, as usual, separated by commas and the variable field itself is terminated by a blank.

The term "variable field" of the symbolic instruction is not to be confused with the name VFD meaning "variable field-definition;" this latter term is meant to indicate that the binary words of the converted result have been broken up into the "variable field" format.

From the above sample instruction would be generated a machine word which contains (assuming ALPHA is location 18563_8):

Specification	Bits	Contents	
10/895	0-9	1577_8	(895_{10})
O5/37	10-14	37_8	
15/ALPHA	15-29	18563_8	
H6/C	30-35	23_8	

Note that the number preceding the "/" in the instruction subfield defines the length (number of bits) of the binary subfield. If the letter O precedes this number, this indicates that the information to follow the "/" is an octal integer, while the letter H indicates Hollerith information.

The absence of any letter indicates that the information following the "/" is to be regarded as an ordinary (arithmetic) expression, and is to be evaluated according to the standard rules, except that only the rightmost n bits of the result are to be used, where n is the length of the subfield.

Hence, an integer is treated as in an arithmetic expression, not as an integer in the sense of DEC, where it is converted to a signed binary number. For example, VFD 18/-1, 18/1 converts to $(777\ 777\ 000\ 001)_8$. The restrictions on the arithmetic expression here are those given in Section 02. Hence, no integer should exceed $2^{35}-1$. Integers larger than $2^{15}-1$, but less than 2^{35}, are allowable and will be properly converted.

If the binary result requires less than n bits, the result will be placed in the right-hand end of the binary subfield, and zeros will be filled in on the left. This is also true of octal fields. For example, VFD 36/8 and VFD O36/10 both convert to $(000\ 000\ 000\ 010)_8$.

The examples given thus far specify information to be packed into a single binary word, and the sum of the lengths of the binary subfield is 36. This is not a requirement. Any number of binary words can be specified, using any number of binary subfields. (See the description of ETC.) Moreover, subfields may overlap the binary words. For example;

Location	Operation	Variable Field
PACKIN	VFD	30/1,O12/777

is allowable and converts to the two binary words:

$000\ 000\ 000\ 107_8$
$770\ 000\ 000\ 000_8$

Note in the above example that the sum of the two subfield lengths is 42, which is not a multiple of 36. As implied, this results in the unspecified bits of the last-used word being taken as 0's. (In the example, there are 30 such bits.) Thus, the example could equally well have been written PACKIN VFD 30/1,O12/777,30/0. The number of words generated by a VFD instruction is, of course, always the smallest integer greater than or equal to the sum of the lengths of all the subfields divided by 36.

Note in the above example that the location symbol PACKIN appears in the instruction. As in the three previously described pseudo-operations, such a symbol is entered into the dictionary with the current value of the location counter, so that the first word of information generated by the VFD instruction can be referred to by this symbol. The value of the location counter is then increased by the number of words generated. In the above example, if the location counter value had been 3900_{10} when the instruction was encountered, the symbol PACKIN would have received the value 3900_{10} and the final value of the location counter would have been 3902_{10}.

The length of a subfield must not exceed 63_{10}. If a binary subfield length exceeding 63_{10} is specified, this length is taken as 63_{10} and an error is indicated in the output listing.

Although decimal integers (i. e., integers in an ordinary subfield) should not exceed $2^{35}-1$, there is no limit on the number of octal digits that can be specified in an octal (O) subfield, beyond the limitation mentioned in the previous paragraph and the limitation imposed by the length of the instruction card.

The length specified in a Hollerith (H) subfield should, in general, be a multiple of 6, since one Hollerith character converts to six binary bits. However, it is not required. If the bit-length specified is too small to accommodate the characters specified, the resulting string of bits is truncated, on the left, to that length. If the bit-length is too large, the string is right-justified, i. e., placed in the right-hand end of the binary subfield. However, the left-hand unused bits in this case are filled out with the binary code for blank, i. e., 110000_2. If the Hollerith subfield bit length is too large and is not a multiple of 6, a terminal segment of this six-bit code is used to fill out the subfield on the left.

Any Hollerith characters, except blanks and commas, can be used in an H subfield. Blanks and commas are, of course, not permitted because they are used to terminate the variable field and to separate subfields of the variable field, respectively.

DUP (Duplicate)

The instruction DUP M,N causes the next M instructions to be duplicated N times. For example, suppose that the current value of the location counter is 3900_{10} when the following instructions are encountered:

Location	Operation	Variable Field
	DUP	3,30
	DEC	1
	DEC	2
	DEC	3

The effect of these four instructions will be to assign the 90 integers 1, 2, 3, 1, 2, 3, . . . , 1, 2, 3 to the 90 locations 3900, 3901, 3902, 3903, . . . 3988, 3989, respectively. The value of the location counter will be left at 3990_{10}.

Exactly the same effect could have been obtained by writing

Location	Operation	Variable Field
	DUP	1,30
	DEC	1,2,3

Note, as in this example, that some or all of the instructions to be duplicated may be pseudo-instructions, which may in turn generate more than one word in the object program. DUP M,N increases the location by the quantity: N times (the number of machine instructions plus the number of words reserved by principal pseudo-operations).

No more than ten principal pseudo-operations (BSS, BES, TCD, HEAD, ORG) may appear within the range of a DUP instruction.

If a location symbol is used with a DUP instruction, this symbol will be entered in the dictionary with the value of the location counter at the time the DUP instruction is encountered. Thus, this symbol can be used to refer to the first word generated by the DUP instruction (in the above example, this would be location 3900_{10}).

If an instruction to be duplicated has a location symbol, this symbol is associated with the first, and only the first, occurrence of the duplicated instruction.

No instruction which a DUP instruction specifies is to be duplicated may itself be a DUP instruction or an END instruction.

If; in the instruction DUP M,N; M and/or N are missing, zero, non-numeric, or complex, an error will be flagged on the output listing and M will be replaced by 1 and/or N will be replaced by 2.

LBR (Library Program)

Normally, each installation using SOS will have available, during the execution of the Compiler, one (or possibly more than one) tape containing a set of library programs. Each of these library programs will normally conform to SHARE standards and will exist on the tape in SQUOZE (binary-coded-symbolic) form. Moreover, each library program on the tape will have an identification recognizable by the Compiler, so that the Compiler can search the library tape for a specified program, and incorporate it into the source program being processed.

The identification label is not to be thought of as a symbol. It is simply an identifying string of alphanumeric characters.

The library program itself, being in SQUOZE form, will ordinarily use certain location symbols. The programmer is provided with means for incorporating all, some, or none of these symbols in the object program. Usually he will probably prefer to eliminate all symbols except possibly the first one used in the library program by specifying that the library program is to be "relativized."

A program is said to be "completely relativized" if only one location symbol is used and all references are made relative to this symbol. For example, consider the following "unrelativized" program:

Location	Operation	Variable Field
USEP1	TZE	EXIT
	FAD	UNITY
	FDP	P1
EXIT	TRA	1,4
UNITY	DEC	1.0
P1	DEC	3.14159

The above program will be completely relativized with respect to the symbol "USEP1" if it appears in the form:

Location	Operation	Variable Field
USEP1	TZE	USEP1+3
	FAD	USEP1+4
	FDP	USEP1+5
	TRA	1,4
	DEC	1.0
	DEC	3.14159

A program may also be "partially relativized" in the sense that more than one, but not all symbols are retained.

Now suppose that the programmer is at a point in writing his source program where he would like to specify incorporation of a certain library program. Suppose that his program is on the standard library tape and that the program has identification IDENT. Suppose further that the programmer would like to have the program relativized with respect to the first symbol of the program. Then he can simply write:

Location	Operation	Variable Field
	LBR	IDENT

When the instruction is encountered, the Compiler will search the standard library tape for the program labeled IDENT, relativize this program, and incorporate the resulting program into the source program. The first word of the generated program will be assigned the value in the location counter at the time the LBR instruction is encountered. The number of words generated by an LBR instruction will be the total number of words generated by all the instructions of the library program itself, and the value in the location counter will have been increased by this number when the Compiler has finished processing the LBR instruction. Thus the effect is just as if the original symbolic instructions for the library program had been inserted into the source program in the place of the single LBR instruction (except for the relativization which will result in the omission of symbols as illustrated above). Of course none of the omitted symbols will be entered into the dictionary. All retained symbols, like USEP1 in the example above, will be entered into the dictionary.

It will often be the case that the programmer, in using a library program, will not want to retain any of the symbols in the library program in his final object program dictionary, because of possible conflicts with his own source program symbols. He may also not want to know (nor be forced to refer to the library program description) the symbol to which he must refer when he calls for the library program in his main program (normally, he does this with a TSX to the library program). All he really needs to know is the identification and the tape. In the above example, "LBR IDENT", he did not even have to specify the tape, since he knew it was the standard library tape. The identification label IDENT in this case may or may not be the same as the name of the program, i.e., the symbol of the starting instruction of the program (to which the main program must transfer). In this case, he may construct a symbol, say SUBRI, and write:

Location	Operation	Variable Field
SUBRI	LBR	IDENT

The effect of this instruction will be to enter the symbol SUBRI into the dictionary. The value used for SUBRI will be the same as the value assigned to the first word generated by the LBR instruction, i. e., the value of the location counter when the LBR instruction is encountered. If this first word already has an associated symbol because such a symbol appears within the library program itself, then this symbol will be omitted and relativization will occur with respect to the programmer's symbol SUBRI, unless the symbol is specifically "exempted from relativization" (see the description of EXEMPT following). Thus the programmer, by specifying relativization and by using his own location symbol SUBRI, has effectively omitted all the symbols used in the library program. He may then enter the library program by the instruction:

Location	Operation	Variable Field
	TSX	SUBRI, 4

The programmer may not want to relativize the library program. In this case, he can write:

Location	Operation	Variable Field
SUBRI	LBR	IDENT, U

where the second subfield, U, means "unrelativized." In this case, all of the original symbols used in the library program are entered into the dictionary, and the programmer must be careful not to use these symbols in his own program as location symbols. Even in this case, the symbol SUBRI might be used to enter the library program, since it will still be in the dictionary with the value assigned the first word generated by the LBR instruction. If the first word already has an associated location symbol because such a symbol appears in the library program itself, then either this symbol or SUBRI can be used, since both have the same value.

If the first subfield of the variable field of a LBR instruction (i. e., the identification subfield) is zero or blank, then the instruction must have a location symbol, and this location symbol will also be used to serve as the identification label. Thus, SUBRI LBR is equivalent to SUBRI LBR SUBRI, and SUBRI LBR 0, U is equivalent to SUBRI LBR SUBRI, U.

If the second subfield of the variable field is omitted, or contains anything except the single character U, then the program will be relativized.

EXEMPT (Exempt from Relativization)

This pseudo-operation has meaning only when used in a source program which is intended to be a library program. It is used to specify which symbols in the library program are to be exempted from relativization.

Since the Compiler has no means of distinguishing a source program intended to be a library program from a source program which is not, the Compiler always generates a list of exempt symbols specified by an EXEMPT instruction, even though this list will have no meaning unless it is to be part of a program on a library tape.

The variable field of an EXEMPT instruction may have one or more subfields. Each subfield must contain a symbol which occurs as a location symbol somewhere in the library program in which the EXEMPT instruction occurs. When the instruction occurs the symbols given in the variable field are arranged in a list of exempt symbols. If more than one EXEMPT instruction is used, the additional symbols are also added to this list. The order of the symbols is unimportant. The list is retained with the compiled library program when the program is placed on the library tape. When the library program containing this "exempt list" is called in by an LBR instruction which specifies relativization (i. e., does not use the character U in the second subfield of the variable field) all exempt symbols are entered into the dictionary. That is, they will be exempt from relativization and will not disappear from the library program when it is incorporated into the main program. (See page 03.00.21.)

Actually, a special list is not used, an exempt symbol in the library program is indicated as exempt by the use of a bit in the dictionary entry for this symbol.

Of course, if the U (specifying "unrelativized") is used in the LBR instruction, all symbols are retained and the exempt list is ignored.

As an example, suppose that a library program whose identification is "SINCOS" is planned to be used in two ways, one to evaluate the Sine function where the user must enter the program by TSX SIN,4 and the other to evaluate the Cosine function where the user must write TSX COS,4. Suppose that the library program is arranged as follows:

Location	Operation	Variable Field
SIN	xxx	xxxx
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
COS	xxx	xxxx
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
ERROR	xxx	xxxx
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
EXIT	xxx	xxxx
.	.	.
.	.	.
.	.	.
	EXEMPT	SIN, COS

where the two symbols SIN and COS have been exempted.

Now if LBR SINCOS is included in a source program, the library program, when incorporated into the source program, will be relativized using the symbols SIN and COS. Thus, both of these symbols will be retained, but all other location symbols (ERROR, EXIT, etc.) will be removed.

If the instruction SINE LBR SINCOS is used, both the symbols SIN and SINE would be entered into the dictionary with the same value, namely the location value assigned to the first word generated by the library program. In such a case, the internal library symbol, SIN, not the external symbol, SINE, will be used for relativization. However, since SIN and SINE both have the same value, the programmer could enter the subroutine by either TSX SIN, 4 or TSX SINE, 4.

To illustrate relativization with respect to more than one symbol, suppose that the following library program has been processed by the Compiler, and placed on the library tape with identification IDENT:

Location	Operation	Variable Field
USEPI	TZE	EXIT
	FAD	UNITY
	FDP	PI
EXIT	TRA	1,4
UNITY	DEC	1.0
PI	DEC	3.14159
	EXEMPT	USEPI, UNITY

When the instruction LBR IDENT is encountered during processing some program, the following will be incorporated:

Location	Operation	Variable Field
USEPI	TZE	USEPI+3
	FAD	UNITY
	FDP	UNITY+1
	TRA	1,4
UNITY	DEC	1.0
	DEC	3.14159

If no symbols in a library program are specifically exempted from relativization and an LBR instruction is used to incorporate the program into a source program, where the LBR instruction specifies relativization but has no location symbol, then it is necessary that the first instruction of the library program which generates a machine word have a location symbol. If this condition is violated, an error will be indicated on the output listing, the first symbol will be considered to be exempt, and complete relativization will be done using this symbol.

A location symbol should not be used with an EXEMPT instruction itself, since it would be logically meaningless because EXEMPT generates no machine words. If a location symbol is used in an EXEMPT instruction, it will be ignored and an error indicated on the output listing.

Macro-Operations

A macro-operation is a special type of pseudo-operation. An instruction whose operation is a macro-operation is called a macro-instruction. The most significant property of a macro-instruction is that it generates n machine words where n is greater than or equal to 1 (ordinarily, n is greater than 1, hence the term "macro").

The instructions generated by a macro-instruction are what is often called an "open subroutine". Unlike a "closed subroutine", whose use is ordinarily independent of its location in storage, the instructions generated by a macro are executed "in-line", i. e., serially with the rest of the main program.

A macro can be regarded as an abbreviation for a block of instructions (or words resembling instructions). The block of instructions generated by a macro is determined by its definition and the items used in the variable field. The definition consists of a "skeletal pattern" of instructions. This skeleton is filled in with the items in the variable field of the macro.

Two classes of macro-operations are provided for by SOS. The first class consists of macros which the programmer himself can define, these are called "programmer macros." The second class consists of a large set of permanent macros comprising the vocabularies of the Input/Output System, Debugging System and SHARE Monitor routines. In addition, there are two special macros: BEGIN and RETURN (see below). This set is called "system macros."

A. MACRO (Macro-Instruction Definition)

This pseudo-operation is not itself a macro-operation and does not generate any machine words. It is used to define programmer macros.

Suppose that the programmer has written a source program with the following structure:

Location	Operation	Variable Field
.	.	.
.	.	.
.	.	.
.	.	.
	CLA	FEDTAX
	ADD	STATAX
	STO	TOTAX
.	.	.
.	.	.
.	.	.
.	.	.
	CLA	XSUB1
	ADD	YSUB1
	STO	ZSUB1
.	.	.
.	.	.
.	.	.
.	.	.

```
                CLA             PART1
                ADD             PART2
                STO             TOTAL
.               .               .
.               .               .
.               .               .
.               .               .
```

The pattern of three instructions (which in this particular example appears three times in the source program) might be abbreviated by some alphabetic name, e.g., QSUM. This abbreviation could then be defined as follows:

```
Location        Operation       Variable Field

QSUM            MACRO           V1, V2, V3
                CLA             V1
                ADD             V2
                STO             V3
                END
```

The above sequence of five instructions itself generates no words in the object program - the sequence constitutes the definition of the programmer macro called QSUM. This definition is then inserted in the source program before the first place the three instructions are required. The programmer could replace each of the three blocks in his source program as follows:

```
Location        Operation       Variable Field

.               .               .
.               .               .
.               .               .
(5-instruction definition occurs somewhere here)
.               .               .
.               .               .
.               .               .
.               .               .
                QSUM            FEDTAX, STATAX, TOTAX
.               .               .
.               .               .
.               .               .
.               .               .
                QSUM            XSUB1, YSUB1, ZSUB1
.               .               .
.               .               .
.               .               .
.               .               .
```

```
                    QSUM            PART1, PART2, TOTAL
     .              .               .
     .              .               .
     .              .               .
     .              .               .
```

Note, as in the above simple example, the following points:

1. QSUM, which appears in the location field of the instruction whose operation is MACRO, is not a symbol. It is a name for the programmer macro-operation defined. This name is not entered in the dictionary as a symbol. Instead it is retained in a special table of programmer macros along with its skeletal definition. When the name appears in the operation field of an instruction, the definition in the table of programmer macros is generated in the program. Since the name is used as an operation code, it must be no more than six alphabetic characters long, and must not coincide with any other operation code.

   Since the macro-name is not a symbol, it may be identical to a true location symbol (appearing elsewhere in the source program) without confusion between the one and the other.

   A useful fact to remember is that none of the operation codes in SOS (machine or pseudo, including system macros) begin with the character Q. Thus, as in the example, the use of Q as the first character in the name of a programmer macro insures non-ambiguity.

2. The variables (V1, V2, and V3 in the example) which appear in the variable field of the instruction whose operation is MACRO, and also in the skeleton which immediately follows, only indicate the order items must be specified and the places each is to be inserted. For example, suppose a programmer defines the macro QSUM as above and then writes:

Location	Operation	Variable Field
	QSUM	TOTAX, FEDTAX, STATAX

   Then the following block is generated in the source program:

Location	Operation	Variable Field
	CLA	TOTAX
	ADD	FEDTAX
	STO	STATAX

On the other hand, if his definition had been written QSUM MACRO V3, V1, V2 with the skeleton the same as in the original definition, then the instruction QSUM TOTAX, FEDTAX, STATAX would expand into the same instructions as before.

3. The end of the macro-definition must always be indicated by the use of an END instruction immediately following the last instruction in the skeleton. The location and the variable fields of the END instruction should be blank. (For another meaning of END, see description following.)

As in the case of LBR, DEC, etc., a location symbol used in a programmer macro-instruction will be given the same value as the value assigned to the first word generated by the instruction. For example, using the above definition of the macro QSUM, the instruction

Location	Operation	Variable Field
TAX	QSUM	FEDTAX, STATAX, TOTAX

would generate the instructions

Location	Operation	Variable Field
TAX	CLA	FEDTAX
	ADD	STATAX
	STO	TOTAX

In the example given above, the elements of the skeleton required to be filled-in were simply address subfields of the skeleton instructions, and in the example using the defined macro, these address subfields were filled in by symbols. In general, a variable used in any skeleton instruction may appear in the location field, the operation field, or in any of the subfields of the variable field. Moreover, the variables may take on values which are expressions (symbols, integers, or non-simple expressions) or Hollerith characters. For example, the following definition is possible.

Location	Operation	Variable Field
QPOLY	MACRO	COEFF, LOOP, DEG, T, OP
	AXT	DEG, T
	LDQ	COEFF
LOOP	FMP	GAMMA
	OP	COEFF+DEG+1, T
	XCA	
	TIX	LOOP, T, 1
	END	

Note that the variable LOOP appears in a location field, OP appears in an operation field, and COEFF and DEG appear combined and uncombined in address subfields. Note also that GAMMA is a symbol, not a variable, and presumably will be defined (in the sense of being assigned a location value) elsewhere in the program. Of course, any use of the operation QPOLY in a macro-instruction must be accompanied by appropriate specifications for values of the variables. For example, LOOP must be given a value which is a pure symbol, and this symbol, being a location symbol, can be used only once in the program - otherwise it would be multiply defined. OP must be some legitimate operation code, like FSB. (The "value" of the operation code in such a case as this might also contain a final * to specify indirect addressing in the instructions in which it appears, e.g., FSB*.)

Having written the above definition of QPOLY, suppose that the programmer now writes:

Location	Operation	Variable Field
	QPOLY	C1-4, FIRST, 5, 4, FAD

Then this macro-instruction will expand into the following block of six instructions:

Location	Operation	Variable Field
	AXT	5, 4
	LDQ	C1-4
FIRST	FMP	GAMMA
	FAD	C1+2, 4
	XCA	
	TIX	FIRST, 4, 1

Notice that these instructions are the result of simple replacement in the definitional skeleton, except for the expression "C1+2", which results from the skeleton address "COEFF+DEG+1", where COEFF=C1-4 and DEG=5. Unlike the examples given thus far, the definitional skeleton of a programmer macro may also use pseudo-operations which generate one or more machine words, i.e., "VFD", "BCI", system macros, etc.

Moreover, the definition of a programmer macro may itself contain one or more programmer macros, which in turn may be defined using programmer macros, etc., with no restriction as to depth. Note, however, that a programmer macro may not contain itself since this would lead to the generation of infinitely many instructions. For example, suppose we define:

```
Location    Operation    Variable Field

QADD        MACRO        A,B,C
            CLA          A
            ADD          B
            STO          C
            QADD         A,B,C
            END
```

Now if the programmer were to write

```
            QADD         ONE,TWO,X
```

this would expand to

```
            CLA          ONE
            ADD          TWO
            STO          X
            QADD         ONE,TWO,X
```

which would in turn expand into

```
            CLA          ONE
            ADD          TWO
            STO          X
            CLA          ONE
            ADD          TWO
            STO          X
            QADD         ONE,TWO,X
```

and so on.

The pseudo-operation LBR could be used in the definitional skeleton, but this will generally make sense only if the identification (and location symbol, if any) is variable, and if different uses of the macro-operation specify different library programs to be incorporated. Otherwise each use would result in the same library program being incorporated more than once.

Suppose that the following definition is given:

```
Location    Operation    Variable Field

QCSEQ       MACRO        SUBR,T,LABEL,SCALE
            TSX          SUBR,T
            VFD          H36/LABEL, 36/SCALE
            END
```

Then if the programmer writes:

Location	Operation	Variable Field
	QCSEQ	PRINT, 4, SECOND, 35

he will effectively obtain the following block of instructions:

Location	Operation	Variable Field
	TSX	PRINT, 4
	VFD	H36/SECOND, 36/35

The last example illustrates how, by using an appropriate definition, the programmer can write a calling sequence in a single instruction. There may be many places in his program where he requires a calling sequence of this particular form, and in each place, he can use his established definition of the operation QCSEQ. By thoughtful use of programmer macros, much writing can be saved and the readability of the source program itself may be improved.

The maximum number of definitions (i. e., number of different programmer macro-operations) which can be used in a single source program is 100. However, there is no limit on the length of any single definition, i. e., the number of instructions in the skeleton, nor to the sum of the lengths of all definitions used.

Because of internal space considerations, the maximum number of variables which can be used in the definition, and hence the use, of a programmer macro-operation is 32. However, any instruction whose operation is MACRO and any instruction using a defined macro-operation can be extended to more than one symbolic card by means of the pseudo-operation ETC.

The use of * (current contents of the location counter) as the "value" for a parameter used as an address in a macro-skeleton is permissible and will be interpreted in the natural way; e. g., given the definition:

Location	Operation	Variable Field
QLDTR	MACRO	X, Y
	CLA	X
	TRA	Y
	END	

then the instruction

```
QLDTR     ALPHA,*+1
```

would generate the instructions

```
CLA       ALPHA
TRA       *+1
```

where *+1 would refer to the instruction following the TRA instruction.

The use of a programmer macro-operation with an indirect address indication e.g., SUMMAC* will result in the use of an indirect address in the first instruction of the expansion.

In addition to symbols, integers, non-simple expressions, etc., another kind of "value" that a variable in a macro-definition can take on is a heading character. (See description of HEAD following.)

If a programmer macro which uses a given name is defined in the source program, and if a new definition using the same name occurs later in the program, then any use of this name as a macro-operation which appears after the new definition is referred to the new definition, not the old one.

B. BEGIN (Begin Subroutine)

BEGIN and RETURN (see below) are related special-purpose system macros. They have been provided in SOS as an aid in specifying basic linkage within a closed subroutine, particularly within library programs.

A standard SHARE library program requires a calling sequence with the following structure:

Operation	Variable Field	
TSX	LIBPG,4	Transfer to library program.
.	.	N words of information deter-
.	.	mined by LIBPG specifications.
.	.	
(transfer to error routine)		(may or may not be required)
		Place of normal return.

The library program itself must provide for a normal return to the first instruction following the calling sequence by means of, say, "TRA K, 4" where K is the total number of words in the calling sequence, including the TSX instruction and the error transfer which may appear as the last word of the calling sequence.

Since index register 4 (XR4) is used in the return transfer, the library program must initially save and finally restore the contents of XR4 before TRA K,4 is executed if it employs XR4 in another connection. Another SHARE convention requires that the contents of XR4 be saved regardless of whether it is used. (This is for debugging purposes.) As will be seen, BEGIN does not require that this convention be followed, but the use of RETURN does require it.

The general form of a BEGIN instruction is:

Location	Operation	Variable Field
FIRST	BEGIN	K,T,I,E

where FIRST is a location symbol associated with the first instruction generated by BEGIN and K, T, I, and E have the following meanings:

1. K is as defined above, i.e., the location of the normal return relative to the TSX instruction in the calling sequence.

2. T specifies the set of index registers which are to be initially saved and finally restored. T is interpreted just as a tag in an ordinary machine instruction. That is, T = 0 specifies saving no XR's, T = 1 specifies XR1, T = 2 specifies XR2, etc.

   If the SHARE convention of always saving XR4 is adhered to, T must be 4, 5, 6, or 7. On the other hand, T = 0, 1, 2, or 3 are acceptable, and each of these values results in the desired instruction expansion. Note, however, that RETURN cannot be used in this case.

3. I specifies whether the contents of the Sense Indicators are to be saved. If I = 0, then the indicators are <u>not</u> saved and restored; if I = 1, then the contents of the indicators are saved and restored. I = 0 is the normal case, and, of course, can be indicated by simply omitting the I-subfield.

4. E is used to indicate whether to save and restore the conditions for data channel trapping. If E = 0, or is omitted then the conditions are not saved and restored. If E = 1 the conditions will be saved and restored.

Trapping will be disabled within the closed subroutine unless the routine enables its own trapping conditions. *

The number of machine instructions into which a BEGIN macro-instruction expands is a function of the values of T, I, and E. The maximum number is 20, obtained by using T = 7, I = 1 and E = 1. For example

Location	Operation	Variable Field
LIBPG	BEGIN	2,7,1,1

expands into

Location	Operation	Variable Field	
LIBPG	TXL	*+11	
	AXT	**,4	
	AXT	**,2	
	AXT	**,1	
	LDI	*+5	
	STI*	SYSENB	Second set executed
	LDI	*+4	
	XEC	SYSENB	
	TRA	2,4	
	PZE		
	PZE		
	XEC	SYSDSB	
	SXA	*-11,4	
	SXA	*-11,2	First set executed
	SXA	*-11,1	
	STI	*-5	
	LDI*	SYSENB	
	STI	*-8	
	STZ*	SYSENB	
	LDI	*-9	

The first set of instructions ("first" from the standpoint of execution) saves the index registers and the sense indicator register, and disables all channel traps. The second set restores these special registers and then returns (by TRA 2,4) to the main program. The first instruction of the expansion, TXL *+11, skips over this second set, and the first instruction of the second set, AXT 0,4, will presumably be transferred to from some later instruction in the subroutine. It will be seen that this transfer can be affected by the macro-operation RETURN.

* This option is available only with the SHARE Monitor.

If the SHARE convention of saving XR4 is followed, the smallest number of instructions obtainable from a BEGIN macro-instruction is four. For example:

Location	Operation	Variable Field
LIBPG	BEGIN	3,4

expands into:

Location	Operation	Variable Field
LIBPG	TXL	*+3
	AXT	0,4
	TRA	3,4
	SXA	*-2,4

The logical minimum is two instructions, LIBPG BEGIN 1 expands into:

Location	Operation	Variable Field
LIBPG	TXL	*+2
	TRA	1,4

C. RETURN (Return)

Often a library program will be written to detect one or more possible kinds of errors during its execution, e.g., unacceptable input, overflow, etc. As mentioned under BEGIN, the SHARE convention, in case of an error, will be to cause a return to the main program one location before the place of normal return. In order to distinguish between different kinds of errors in the same library program, the standard SHARE procedure will involve placing an error code (by means of instructions executed within the library program) in the decrement part of the first instruction of the library program, which, as seen above, is a TXL instruction with a zero tag. The programmer who writes the library program chooses these error codes for his particular program as he pleases. In the calling sequence for this library program, he can write a transfer, as the last instruction, to a subroutine to analyze the kind of error, making use of the error code stored within the library program.

The macro-operation RETURN is designed to provide the programmer with means for specifying, within the subroutine:

1. instructions to effect a normal return
2. instructions to effect an error return using a certain error code.

In both cases 1 and 2, it is assumed that a BEGIN instruction, or machine instructions equivalent to the expansion of a BEGIN instruction, with an associated location symbol of, say, "LIBPG", has been written.

Then, as an illustration of case 2, the macro-instruction:

Location	Operation	Variable Field
BACK	RETURN	LIBPG

will expand simply into:

Location	Operation	Variable Field
BACK	TRA	LIBPG+1

The specified transfer to LIBPG+1 is the desired link to the restoring set of instructions associated with the BEGIN instruction.

Since this use of RETURN results in a single instruction, the programmer could as well write BACK TRA LIBPG+1, instead of using the macro,

As an illustration of an error return, suppose that the error code is 8. Then the programmer may write this error code in a second subfield of the variable field, as follows:

Location	Operation	Variable Field
ERRI	RETURN	LIBPG, 8

The specification, as in this example, of any expression except zero (or blank) in the second subfield of the variable field always results in a four-word expansion. The expansion for this example would be:

Location	Operation	Variable Field
ERRI	AXT	8,4
	SXD	LIBPG,4
	LXA	LIBPG+1,4
	TXI	LIBPG+2,4,1

As can be seen, the effect of these four instructions will be to:

1. place the error code, "8", in the decrement part of the instruction at LIBPG
2. restore XR4
3. increase the contents of XR4 by 1

4. transfer to the second instruction after LIBPG.

Note that the above instructions assume that XR4 has been saved in the manner prescribed by a BEGIN instruction, and also that the exit instruction is TRA K,4, which when executed will result in a return to the location one less than the normal return (since the contents of XR4 have been increased by 1 by the TXI instruction).

As an illustration of the joint use of BEGIN and RETURN, consider the following sequence:

Location	Operation	Variable Field
SR	BEGIN	2,7,1
	TPL	SR2
SR1	RETURN	SR,1
SR2	DVP	ALPHA
	STQ	BETA
SR3	RETURN	SR

The instruction named SR1 specifies an error return with an error code of 1, while the instruction named SR3 specifies a normal return. This six-word sequence would be expanded into the following 19-word set of instructions:

Location	Operation	Variable Field
SR	TXL	*+7
	AXT	0,4
	AXT	0,2
	AXT	0,1
	LDI	*+2
	TRA	2,4
	PZE	
	STI	*-1
	SXA	*-5,1
	SXA	*-7,2
	SXA	*-9,4
	TPL	SR2
SR1	AXT	1,4
	SXD	SR,4
	LXA	SR+1,4
	TXI	SR+2,4,1
SR2	DVP	ALPHA
	STQ	BETA
SR3	TRA	SR+1

HEAD (Heading)

It is frequently convenient, and sometimes necessary, to construct a source program by writing it in pieces, where these pieces are finally put together and compiled as a total program. Different pieces of the program may be written by different programmers, or by the same programmer with considerable time elapsing between the writing of the pieces.

Suppose, in such a situation, that a program block, say B1, has been written, that another program block, B2, is in the course of being written, and that B1 and B2 eventually are to be joined together into a single program. Certain location symbols have already been used in writing block B1, and certain other location symbols, different from the location symbols used in B1, must now be used in writing block B2. Otherwise, such symbols would be multiply defined. This seems to mean that the programmer who is writing the block B2 must be concerned with the symbols used in B1, even though B2 might be quite independent of B1.

If B1 and B2 are completely independent of each other in the sense that neither makes reference to the symbols defined in the other, then it might be convenient to have B1 and B2 processed independently. On the other hand, such a procedure might be more time-consuming and even very inconvenient in certain situations. For example, if B2 is eventually to be loaded into core storage immediately following B1, and the size of the B1 object program is not easy to compute.

However, if the symbols he is using in block B2 which might conflict with the symbols in B1 are all less than six characters in length, the programmer can completely ignore the symbols in B1 by prefacing B2 with the following pseudo-instructions:

Location	Operation	Variable Field
	HEAD	X

where the single character X may be any non-blank character allowable in symbols (see Appendix 1).

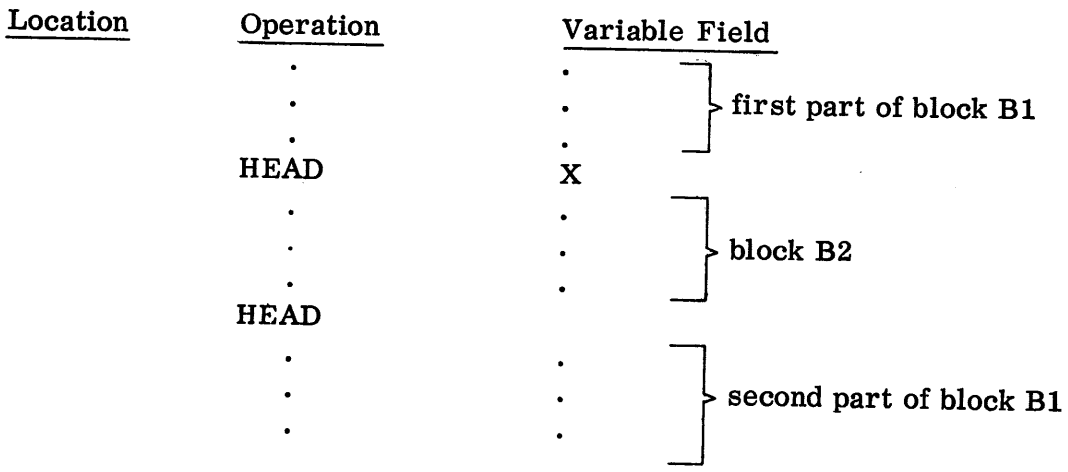
This pseudo-instruction, HEAD X, generates no words in the object program. When such an instruction is encountered, the symbols of all following instructions, until another HEAD is encountered, whether they appear in the location field or in the variable field, are treated as though they were "headed" by the character X provided that these symbols are five or less characters in length. Thus the symbols used in block B1 which are less than six characters long cannot possibly conflict with the symbols used in block B2. Six-character symbols are not affected, i.e., they are immune from "heading".

A symbol, ALPHA, headed by a non-blank heading character, X, is not to be thought of as identical with the symbol XALPHA. On the contrary, the heading character is essentially on a different level from the characters which make up the string representing the symbol. The property, belonging exclusively to six-character symbols, of being immune from heading, will also be seen to be useful.

A symbol is said to be "unheaded" if and only if its representation uses exactly six characters. For example, the symbol COMMON is unheaded. Every symbol, e.g., ALPHA, whose length is less than six characters, is considered to be headed, whether or not it is under control of a HEAD pseudo-instruction as described above. If ALPHA is under control of HEAD X (where X is some non-blank heading character), then ALPHA is said to be "headed by X". If ALPHA is under control of a HEAD instruction with a blank variable field or is not under control of any HEAD instruction, then ALPHA is said to be "headed by blank". Hence, ALPHA headed by a blank should be regarded as identical to the symbol ALPHA.

Of course, if a HEAD instruction with a non-blank variable field does not occur in the entire source program, all considerations of heading can be ignored. This is the reason for not introducing the concept of headed symbols earlier.

A HEAD instruction with a blank variable field must be used if the programmer desires to discontinue the heading process. For example, note that the instruction HEAD and the instruction HEAD 0 are quite different. "0" is an allowable heading character and must be distinguished from the character blank. In the case assumed above, where the blocks B1 and B2 are joined together in one program, suppose that B2 must be put somewhere in the middle of B1, as follows:



Here, the second HEAD instruction effectively serves the purpose of nullifying the instruction HEAD X.

In this example, the entire program might have been prefaced by a HEAD with a blank variable field. As implied above, however, such a HEAD instruction would be superfluous, since the symbols in the first part of block B1 are automatically headed by blank, being under the control of no HEAD instruction at all.

As a concrete illustration, consider the following sequence:

	Location	Operation	Variable Field
		ORG	3000
(3000)	ALPHA	CLA	GAMMA1 (3001)
(3001)	GAMMA1	CLA	ALPHA (3000)
		HEAD	X
(3002)	ALPHA	CLA	DELTA1 (3003)
(3003)	DELTA1	CLA	ALPHA (3002)
(3004)	BETA	CLA	GAMMA1 (3001)
		HEAD	
(3005)	BETA	CLA	DELTA1 (3003)

The value assigned to each location symbol is given by the parenthesized number to the left of the instruction in which this location symbol appears, and the value corresponding to the address symbol is given by the number to the right of the instruction.

Notice in this example that the symbols ALPHA and DELTA each appear twice in a location field, once headed by blank, and once headed by X, and that each different appearance results in a different value. ALPHA headed by blank is entered into the dictionary with value 3000, and ALPHA headed by X, and BETA headed by blank receive values 3004 and 3005, respectively.

Because of the different headings, these two appearances constitute two <u>different</u> symbols, so that this is <u>not</u> a case of multiple definition. On the contrary, the address symbol in the instruction GAMMA1 CLA ALPHA, not being under control of a HEAD instruction, is headed by blank and is given the value 3000, whereas the address symbol in the instruction DELTA1 CLA ALPHA is under control of HEAD X, and is given the value 3002.

Notice also in the above example that GAMMA1 and DELTA1, being six-character symbols, are unheaded, and can be used for reference either inside or outside the headed region in which they appear as location symbols.

The device of referring in one headed region to a symbol defined in another headed region by making this symbol six characters long, is often inconvenient. In order to facilitate cross-referencing between headed blocks, the following convention can be used:

Suppose that a headed symbol, say ALPHA headed by X, has been defined by some instruction. Suppose further that this symbol is to be referred to in an instruction under the control of the instruction HEAD Y. Then the desired reference can be made by writing the string X$ALPHA.

In general, if the two-character string "C$", where C is any allowable heading character, is placed in front of the headed reference symbol ALPHA, then the result is ALPHA headed by C. To specify ALPHA headed by blank, one simply writes $ALPHA, with no character preceding the $ character. (A blank must not be used ahead of the $, since this always indicates the end of the variable field.)

Note that location symbols cannot be headed by means of "$".

As an illustration of the use of $, consider the following:

Location	Operation	Variable Field
	ORG	3000
(3000) ALPHA	CLA	BETA (3001)
(3001) BETA	CLA	X$ALPHA (3002)
	HEAD	X
(3002) ALPHA	CLA	BETA (3003)
(3003) BETA	CLA	$ALPHA (3000)
	CLA	Y$ALPHA (3005)
	HEAD	Y
(3005) ALPHA	CLA	X$BETA (3003)
	CLA	$GAMMA (3007)
	HEAD	
(3007) GAMMA	CLA	ALPHA (3000)
	CLA	Y$ALPHA (3005)

where the value assigned to each location symbol is given by the parenthesized number to the left of the instruction in which this location symbol appears, and the value corresponding to the address symbol is given by the number to the right of the instruction.

All symbols which occur in the instructions generated by a macro-instruction are affected by a controlling HEAD instruction in the same way as symbols in ordinary instructions are affected. This is also true of the symbols in instructions generated by a library program if the library program is called for incorporation into the source program by means of an LBR instruction which specifies relativization. On the other hand, if the program is called for by an LBR that specifies no relativization, all headings internal to the library program will be retained and

will take precedence over any previous external HEAD instructions. These internal HEAD instructions, however, will be effective only in the scope of the incorporated library program. Thus, an external heading control preceding the LBR instruction will be nullified only temporarily by internal HEAD instructions. Since every library program is considered to be headed by blank the programmer must refer to the beginning of the library program by $ whenever the reference is within a headed region. For example, suppose the programmer wants to use the library program LIBPG by entering it via a TSX. If he is within a region headed, say, by X, he must enter the program by a TSX $LIBPG,4.

In defining a programmer macro (see MACRO), it is permissible to use a variable which can take on a heading character as a value. For example:

Location	Operation	Variable Field
QMOVE	MACRO	V1,V2,V3
	CLA	V1$V2
	STO	V3
	END	

defines a macro-operation where V1 is a heading character. With this definition, if one writes:

Location	Operation	Variable Field
	QMOVE	X,ALPHA+BETA*GAMMA+1,DELTA

then one obtains the expansion:

Location	Operation	Variable Field
	CLA	X$ALPHA+X$BETA*X$GAMMA+1
	STO	DELTA

Note that a heading character applied to a non-simple expression (e.g., ALPHA+BETA*GAMMA+1) affects all the symbols used in the expression.

A location symbol should not be used in a HEAD instruction. If a location symbol is used in a HEAD instruction, it is ignored and an error is indicated on the output listing.

ETC (Et Cetera)

With three exceptions, all instructions in SOS must be limited to a single instruction card, i.e., 72 columns. The purpose of ETC is to provide for extending this 72-character limit in the following three cases:

1. a VFD instruction
2. a MACRO instruction (used to define a programmer macro)
3. a system or programmer (defined) macro-instruction.

Suppose that the variable field of an instruction whose operation is VFD, MACRO, or any system or programmer macro-operation, is too long to fit on a single card. Then the variable field of this instruction can be broken off after some subfield, say the nth subfield, and continued in the variable field of a second card, beginning with the n+1th subfield of the instruction, by writing ETC in the operation field of the second card. The comma which ordinarily separates the nth from the n+1th subfield must appear as the last character in the variable field of the first card, not the first character in the variable field of the second card.

The omission, on the first card, of the separating comma between the two subfields will cause an error to be indicated on the output listing, but the comma is assumed to be present.

The location field in an ETC card should be left blank. If a location symbol is used in an ETC card, this symbol will be ignored and an error will be indicated on the output listing.

If one ETC card is still not sufficient to specify the variable field of the instruction, additional ETC cards can be used, without limit.

As an illustration, consider:

Location	Operation	Variable Field
	VFD	18/1,H18/ABC,18/2,
	ETC	H18/DEF,18/3,
	ETC	H18/GHI

Note that this is an example of a single pseudo-instruction (which generates three words in the object program) although it requires three cards.

SQZ (SQUOZE)

SQUOZE decks produced by SOS can be combined with symbolic coding during compilation. The place at which a SQUOZE deck is to be included is indicated by an SQZ instruction. If the symbolic input component and SQZ input component are the same, the SQZ deck is inserted in the symbolic deck following the SQZ card. If they are different, the SQZ deck must be immediately available at the input component when the SQZ card is encountered.

The location symbol of an SQZ card has the value it would receive if the SQZ card were an instruction. An SQZ deck loaded on-line is assumed to be column binary unless the variable field contains the symbol RB, in which case it is treated as a row binary deck.

The deck cannot contain symbolic modification cards, cannot be preceded or followed by blank cards and must be in the form produced by the system.

In the 32K IB Monitor system the programmer macros in the deck (see page 03.00.27) are combined and may be used thereafter. In the 8K IB system programmer macros are not combined. All symbols in the SQZ deck retain their original heading. No heading characters introduced within the SQUOZE deck affect symbols used after the point where it is inserted. If both texts are in the SQUOZE deck, commentary text is combined (if present) and non-commentary text is ignored. If only non-commentary text is present, it is combined.

Cards are checked for checksum agreement, sequence number, and presence of the SQZ punch (minus sign for control word).

Note: A symbolic deck may consist solely of SQZ cards with SQZ decks and an END card.

Additional details are given in Chapter 3, Section 08.

END (End)

This pseudo-operation has two distinct applications:

1. END is used to indicate the end of a skeleton for a programmer macro. This application has already been mentioned under MACRO. In this case, the operation END should appear by itself, with a blank location field and blank variable field.

2. END must also be used at the physical end of every source program, to indicate the end of the source program. Here, the variable field is pertinent and should consist of a single subfield. This subfield should contain the address which constitutes the starting-point of the program. This address, as usual, may be any legal arithmetic expression, but will ordinarily be an integer or the symbol used to name the first executed instruction of his source program, e.g., START.

Suppose that the Compiler encounters the pseudo-instruction:

Location	Operation	Variable Field
	END	ALPHA

This instruction will produce two effects:

1. The Compiler assumes that there are no more instructions to be processed.

2. The symbol ALPHA, which presumably appears with its assigned value in the dictionary, is saved in a special way. Later, when the Modify and Load program has finished loading the object program, the information is used to transfer control to the core storage location corresponding to ALPHA to begin execution of the object program.

In case the programmer has requested his object program in absolute binary form, the result of END ALPHA will be a transfer card containing the absolute address corresponding to ALPHA.

Although an END instruction generates no words in the object program, a location symbol used in an END instruction is processed in the usual way, i.e., entered into the dictionary with the current value in the location counter. This value will be, of course, one greater than the value corresponding to the final word generated by the source program.

TCD (Transfer Card)

This pseudo-operation will have the same effect as the second application of the pseudo-operation END described above, except that it will not cause the Compiler to assume that the source program is at an end. That is, the pseudo-instruction:

Location	Operation	Variable Field
	TCD	ALPHA

causes ALPHA to be saved as control information for later use by the Modify and Load program, as in the case of END ALPHA. After processing the TCD instruction, processing continues with the next instruction.

The rules governing the location field and the variable field of a TCD instruction are the same as for an END instruction.

In case the programmer requests his object program in absolute binary form, the result of TCD ALPHA will be simply an ordinary transfer card containing the absolute address corresponding to ALPHA. This is the reason for the code "TCD". This pseudo-operation is provided, for example, to allow programs which are too large to fit in core storage to be loaded in pieces, each piece being terminated by a transfer card.

LISTER

CHAPTER 1: SCAT LISTINGS

Symbolic listings of a program reproduce, with some exceptions, the symbolic source program deck. The exceptions which are never reproduced are:

1. Invalid operation codes, which are replaced in the listing by ///.

2. Invalid symbols, such as those longer than six characters, which are replaced by //////.

3. The shortened forms of extended operation codes, which are changed to, and listed in their extended forms, e.g., the instruction WRS 1169 is listed as WTBB 1.

In addition to the above, words generated by the BCI, DEC, DUP, LBR and OCT instructions or by macro-instructions are not normally listed in detail (see Chapter 3). Instead, only a "title line" and the first word generated by these instructions are printed. (The "title line" reproduces the card containing the BCI, DEC, DUP, LBR, OCT, or macro-instruction.) All generated words may, however, be listed by exercising one or more of the options available.

When a SQUOZE deck is listed, comments are aligned with the first comment in the program, and therefore may not be aligned as in the source deck.

Symbolic listings produced by SCAT may consist of four main parts:

A. The first part may be either of two items depending on whether the program is being processed by the Compiler or by Modify and Load.

   1. During Compiler processing, this part is a listing of errors found in the program (see "Compiler Error Listing" below).

   2. During processing by Modify and Load, this part is a listing of the modification cards (if any) for the program and any errors found in those cards (see "Modifications Listing" below).

B. The next part produced is a listing of all inconsistently defined principal pseudo-operations and of all undefined and doubly-defined symbols in the program. (See "Symbol and Pseudo-Operation Error Listing.")

C. The third part of the listing will be the symbolic program with absolute (octal) equivalents. (See "Program Listing.")

D. The final part is a (dictionary) listing of all symbols in the program which have been defined at least once. (See "Symbol Listing.")

Each of these four parts of the listing is described in more detail in the following paragraphs.

COMPILER ERROR LISTING

The list of errors detected during the processing of a program will be headed by the statement

COMPILER ERROR LIST XX/XX/XX

where XX/XX/XX is the date of compilation, if supplied to the Compiler; otherwise it will be 0/ 0/00. This statement will be followed by a list of all instructions, including comments and alter numbers (see Chapter 2) in which errors were found. Messages which indicate the type of errors found will be given following each instruction. If no errors are found by the Compiler, this part will consist of merely the statement

NO ERRORS FOUND BY COMPILER

MODIFICATIONS LISTING

A listing of all modification cards is always produced during processing of a SQUOZE deck, if the deck includes such cards. If no modification cards are included, this listing will, of course, be omitted.

The modifications listing may also include messages indicating errors found in modification cards. These messages will follow immediately after the cards in which the errors were found. Error messages (when using 32K SOS only) will be indicated by a row of asterisks to make them easily distinguishable.

A sample of a modifications listing is given below.

```
                    CHANGE   CRT1+16,CRT1+17
                    CLM
                    LLS      35
                    ALTER    45,46
                    CLM
                    LLS      35
                    CHANGE   START+14
                    TIX
MISSING ADDRESS FIELD
TAG AND/OR DECR EXPECTD
                    CHANGE   CRT1+21,CRT1+22
                    CLM
                    LLS      35
```

SYMBOL AND PSEUDO-OPERATION ERROR LISTING

This part of the listing is always given during processing of a program if errors of the types described below are detected.

The symbol and pseudo-operation error listing may consist of up to five parts as indicated below. Any one or more of these parts will be omitted when no items are found which fall into the given category.

1. The first list which may appear in this part will be labelled "INCONSISTENTLY DEFINED PRIN PSEUDO OPS" and will be a list of all the principal pseudo-operations (BES, BOOL, BSS, END, EQU, HEAD, ORG, SYN, TCD), which have caused a circular definition of a symbol(s) in the program. A given principal pseudo-operation may appear more than once in the list if that pseudo-operation occurs in more than one instruction which has caused a circular definition. All items in the list are given in the order in which they are encountered.

2. The next list which may occur in this part is a list of all undefined symbols which occur in the program in connection with a principal pseudo-operation, i.e., in the variable field of a principal pseudo-instruction. This part is labelled "UNDEF SYMBOLS IN PRIN PSEUDO OPS."

   This list will also show the value which has been arbitrarily assigned to each symbol in the list.

3. The third list which may appear in this part is a list of all symbols which appear in the program in connection with principal pseudo-operations and

are defined more than once in the program. Each symbol listed will also show the arbitrary value assigned to it.

This list will be labelled "DOUBLY DEF SYMBOLS IN PRIN PSEUDO OPS."

4. Next may appear a list of all symbols in the program, other than those described in 2 above, which are undefined. The symbols given in this part also will be shown with the arbitrary values assigned them.

   This list will be labelled "UNDEF SYMBOLS IN TEXT."

5. The final list which may be given in this part is labelled "DOUBLY DEF SYMBOLS IN TEXT." This will be a list of all symbols, other than those described in 3 above, which are defined more than once in the program. The arbitrary values assigned each symbol will also be shown.

PROGRAM LISTING

The third part of listings produced by SCAT will be a listing of the program itself. This part will occupy as many pages as is necessary.

Each page of this part will show the job identification, page number, and date in the upper right-hand corner. Page numbers will be assigned beginning with the first page of this part. The job identification and date will be given only if supplied with the program. (See the Monitor sector for information on how these are supplied.) If no date is supplied, 0/ 0/00 will be printed where the date would appear.

Up to fifty lines of the program will be printed on each page, and will include the symbolic program instructions and octal equivalents. These instructions will be given numbers from two reference systems (i.e., relative and alter numbers) which are assigned as described in Chapter 2.

SYMBOL LISTING

The last part of listings produced by SCAT will be a list of all defined symbols in the program, whether they are defined once or more than once. Six-character symbols will be listed first, followed by symbols less than six characters long which are not explicitly headed, followed by symbols which are explicitly headed. The symbols will be listed in five columns and will be arranged alphabetically within each category. Symbols which are doubly-defined will be listed at the end in the order in which they were encountered, and will appear once for every appearance in a location field in the program.

Any symbol in this list which is defined (whether once or more than once) but not used in the variable field of any instruction is indicated by an "*".

The number of the page on which a symbol appears in the location field of the program listing will be shown with the symbol. Thus, since doubly-defined symbols appear in the list once for each definition, all page numbers for doubly-defined symbols are shown.

A sample symbol listing is shown below.

```
PRCOMM  0001        TSTBIT  0002        EXIT    0002        WRITE   0002        ZERO   *0002
RESTOR  0002        WKAREA  0001        IMAGE   0002        ZNUM    0002        ZERO   *0002
STRTWD  0001         CLEAR  0001        MASK    0002      NUMBER *0002
SWITCH  0002         CON6   0001        PRINT   0002      NUMBER *0002
```

CHAPTER 2: REFERENCE SYSTEMS

The two numbering systems previously mentioned, relative numbering and alter numbering, are used by SCAT in order to facilitate references to words in a program. These numbers are initially assigned by the Compiler, and are changed, if necessary, by Modify and Load only when a new SQUOZE deck is punched.

RELATIVE NUMBERING

A relative number is an integer used to indicate the position of a machine word, not assigned a location symbol, relative to the last preceding word in the program with which a location symbol is associated. The positions thus indicated are the relative positions of instructions the last time a SQUOZE deck was punched. However, if unaffected by modifications to the current SQUOZE deck, relative numbers also indicate the relative storage locations to be occupied by the machine words of the program.

Since relative numbers, in a sense, indicate storage locations occupied by machine words, they are assigned only to those instructions which will occupy locations when loaded for execution. Thus, relative numbers are never assigned to principal pseudo-instructions (BES, BOOL, BSS, END, EQU, HEAD, ORG, SYN, TCD), generative pseudo-instructions (BCI, DEC, DUP, LBR, OCT), or programmer macro-instruction definitions. However, the words generated by the generative pseudo-instructions and by macro-instructions <u>are</u> assigned relative numbers.

Relative numbering is begun when the first location symbol of a program is encountered. The word associated with this symbol is numbered +0 (although not shown on listings), the next word is numbered +1, and so forth until either another word with a location symbol, or an instruction with a principal pseudo-operation, is encountered. When a new symbol is encountered the process is begun again. If, however, relative numbering is suspended by one of the pseudo-operations, it is not begun again until a new symbol is encountered. Words for which a positive relative number cannot be computed, will be given a negative relative number, i. e., a number relative to a succeeding symbol, if that can be computed. If neither can be computed no relative number will be shown.

Although only one relative number is shown on the listing for a given word, there exists, in general, many other equivalent relative numbers, both positive

and negative, any one of which may be used when referring to that word. For instance, in the list

```
82    RESTOR    AXT     **0,1          RESTORE
83        +1    AXT     **0,2          INDEX REGISTERS
84        +2    AXT     **0,4          CONTENTS
85        +3    AXT  &  TRA 2,4        RETURN
86        +4    SLN     1              TURN SENSE LIGHT 1 ON
87        +5    TRA     PRINT
88    MASK      OCT     373737373737
89        +1    OCT     377737773777
90    WRITE     PZE     WKAREA,,24
91    IMAGE     BSS     NUMBER,0
92    NUMBER    EQU     24
93    ZERO      EQU     0
94    TSTBIT    PZE                    STORAGE FOR TEST BIT
95              END     PRCOMM
```

the word numbered +1 relative to the symbol MASK, has the equivalent numbers:

+7 Relative to RESTOR
-1 Relative to WRITE

and so forth. Note, however, that there is no number for a word relative to a symbol which is separated from that word by a principal pseudo-operation. For example, in the listing, the words preceding the BSS with the location symbol IMAGE, have no numbers relative to the symbol TSTBIT.

ALTER NUMBERING

Alter numbers are essentially numbers for the symbolic cards in a source program deck. Alter numbers are assigned to remarks cards and to all cards except those which:

1. contain ETC, EXEMPT, and MACRO instructions
2. define programmer macro-instructions
3. fall within the range of a DUP instruction.

Generative pseudo-instructions, including defined programmer macro-instructions, are assigned alter numbers although the words generated by the instructions are not. Modify and Load pseudo-instructions are never listed and, therefore, not assigned alter numbers.

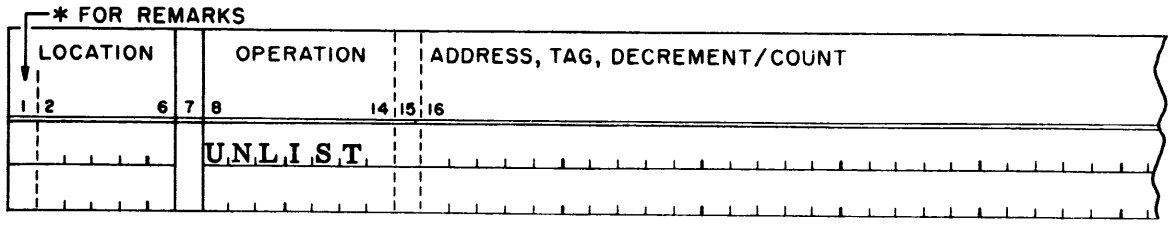
CHAPTER 3: PSEUDO-OPERATIONS

As was mentioned in Chapter 1, SCAT listings normally include only a title line and the first word generated by BCI, DEC, DUP, LBR and OCT instructions and by macro-instructions. This mode of printing is called the normal or Title mode. It was also indicated that all words generated by these instructions could be listed, if desired. This is accomplished by using one of the pseudo-operations described in this chapter. The remaining instructions, of which there are five, are used to control the format of a listing. The features provided permit:

A. suppression of printing for part or all of the listing, if desired
B. inclusion of extra spacing at strategic points
C. printing of various parts of a program on separate pages.

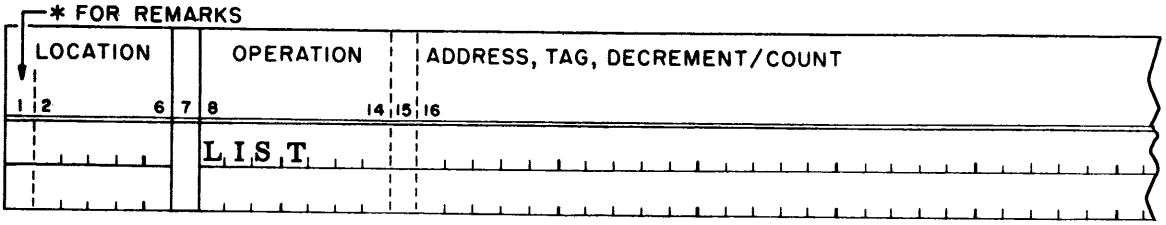
The six listing pseudo-operations may be inserted in a source deck or incorporated subsequently by Modify and Load. A location symbol used in connection with any of the pseudo-operations is meaningless and should be omitted.

UNLIST



The UNLIST pseudo-instruction causes printing to be suppressed until a LIST pseudo-instruction is encountered. This pseudo-operation makes it possible to skip over parts of a program for which a listing is not desired, such as portions which have been completely debugged and no longer require attention until a final listing of the program is produced.

LIST



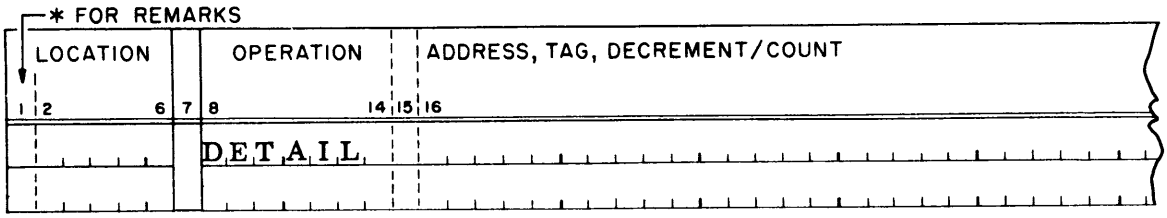
This pseudo-instruction causes printing to be resumed in the normal mode.

The LIST pseudo-instruction is effective until an UNLIST or DETAIL instruction is encountered. LIST has no effect unless an UNLIST or DETAIL instruction appears prior to it in the program.

Note: A LIST pseudo-instruction at the beginning of a program is redundant, since the listing program operates as though a LIST were present. Thus, if the first part of a program is not to be listed, an UNLIST must be included.

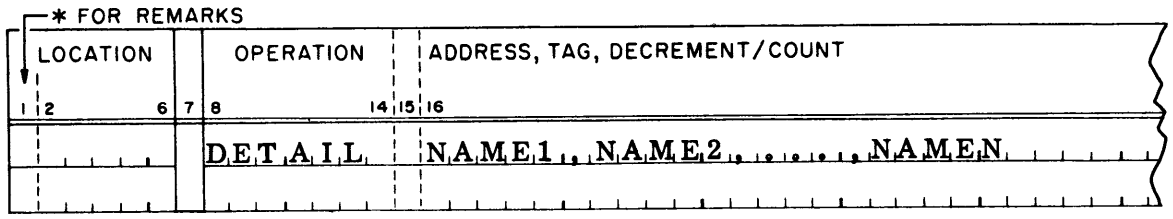
DETAIL

The DETAIL pseudo-instruction may be used in either of two forms. The first is:



This form causes printing, if in progress, to be continued in complete detail, i.e., all machine words generated from macro-instructions and BCI, DEC, DUP, LBR, and OCT instructions will be listed.

The second form which is permitted is

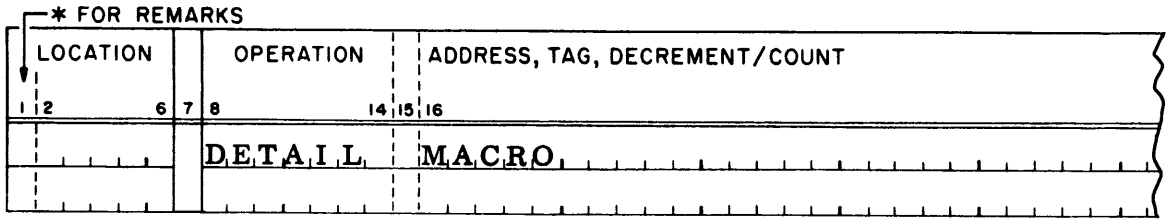


where NAME1, NAME2, . . . , NAMEN may be one or more of the symbols BCI, DEC, DUP, LBR, MACRO, and OCT.

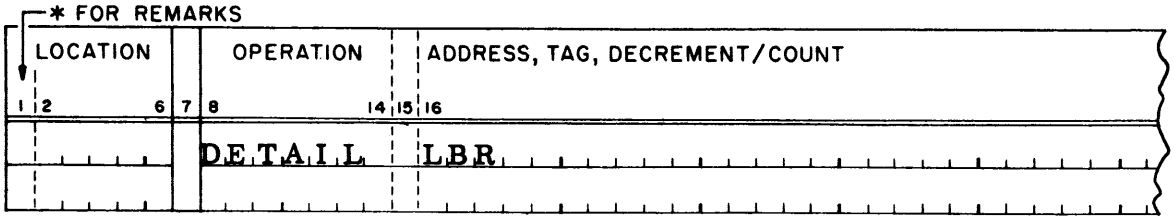
This form causes printing to be continued in partial detail. The appearance of MACRO in the variable field causes detail printing of macro-instructions; the appearance of BCI, DEC, DUP, LBR, or OCT causes detail printing of those

types of pseudo-instructions. If all the permissible symbols appear in the variable field of a DETAIL instruction, that instruction has the same effect as the first form above.

A DETAIL instruction does not cancel the effect of a previous DETAIL which has not been cancelled by a TITLE, LIST or UNLIST instruction; rather, it adds to the effect of that DETAIL instruction. For example,



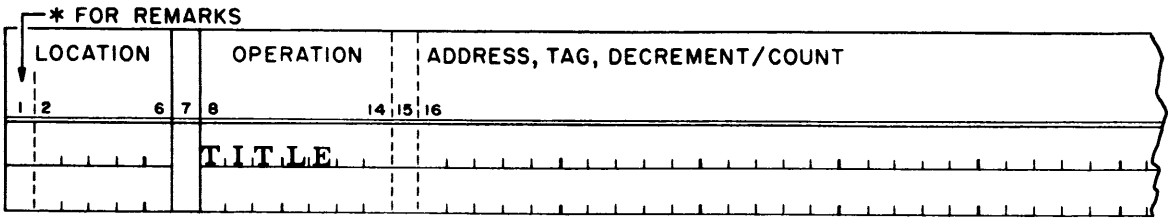
might appear at the beginning of a program to cause macro-instruction to be printed in detail. A subsequent



will then cause detail printing of LBR instructions as well.

The DETAIL pseudo-operation has no effect unless printing is in progress when it is encountered.

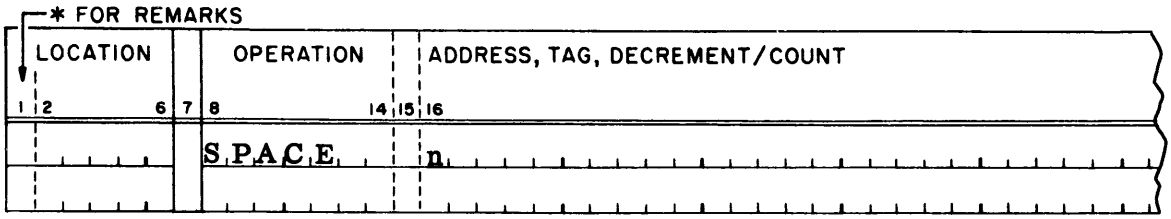
TITLE



A TITLE instruction causes printing in the Title mode. The instruction will have no effect if printing has been suspended, or if printing in the Title mode is already in progress.

TITLE is useful when it is desired to change printing from the Detail to the Title mode if printing is in progress, and printing is not to be resumed if it has been suspended.

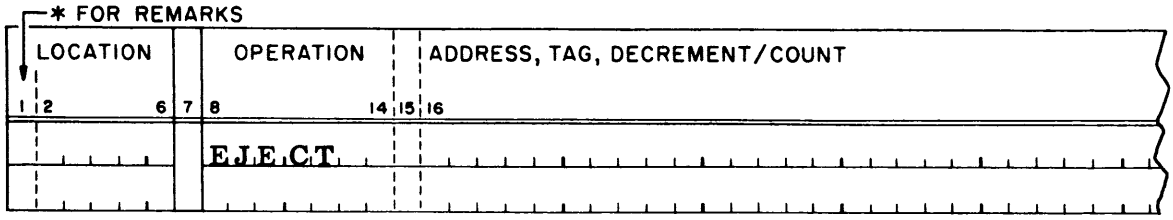
SPACE



where n is any valid character.

This pseudo-instruction will cause the printer to space a number of lines equal to the decimal equivalent of the octal representation of n in storage, e.g., if n is A, the printer will space 17 lines since A is represented in storage as 21_8. However, if n is a blank or is zero, it will be taken to be 1. SPACE is effective only when printing is in progress.

EJECT



The EJECT pseudo-instruction causes the printer to skip to the beginning of the next form. EJECT is not effective if encountered while printing is suspended.

MODIFY AND LOAD

CHAPTER 1: MAIN FEATURES

The SCAT Compiler uses as input a symbolic source program. From this input the Compiler produces a compact binary-coded-symbolic (SQUOZE) program which contains all of the information supplied in the source program, including remarks cards and comments from instruction cards.

SQUOZE decks produced by the Compiler may be used with symbolic decks as input to subsequent Compiler passes, and incorporated with the symbolic deck to form one SQUOZE output deck. Thus, a program can be written in parts and each part debugged before all are combined.

Detailed information concerning the composition and form of the SQUOZE deck is included in the appendices.

SQUOZE decks produced by the Compiler are also used as input to Modify and Load. Since all symbolic information is available to Modify and Load, three major advantages over previous assembly systems are provided:

A. Changes can be specified in symbolic form for incorporation into the program by Modify and Load.

B. Symbolic changes do not require the source deck to be reprocessed by the Compiler.

C. Symbolic information is available and may be retained for printing during debugging runs, thus making debugging easier.

The main functions performed by Modify and Load are:

A. Modification of a SQUOZE program on the basis of symbolic information supplied with the SQUOZE deck.

B. Loading the modified version of a program into storage in preparation for execution of the program.

In addition to the above, Modify and Load also provides the following features:

A. When desirable, a new SQUOZE deck, incorporating symbolic modifications, may be prepared. (A new SQUOZE deck is automatically prepared when a modification affects a heading card.) Generally, it is desirable to exercise this option when the number of modification cards is approximately equal to

the number of cards in the SQUOZE deck, since many symbolic cards will add a significant amount of time to loading and assembly.

B. A symbolic listing of a program can be prepared from a SQUOZE deck which includes no modifications. (A new symbolic listing is automatically prepared when a new SQUOZE deck is punched.)

C. An absolute binary version of a program may be obtained from a SQUOZE deck. Although this option is available to the user, little benefit is derived by exercising the option until a program has been completely debugged, because the debugging and modification features of SOS can only be used with SQUOZE program decks.

CHAPTER 2: PSEUDO-OPERATIONS

The SCAT language includes five pseudo-operations by which changes can be effected through Modify and Load in a program in SQUOZE form; the use and effect of these pseudo-operations are described below.

To accomplish modifications, the modification instructions and any words to be inserted into a program are punched in symbolic form and used as input with the SQUOZE deck. (See the appropriate Monitor description for operational details.) The changes indicated in these cards are made in the program before it is loaded into storage but do not effect the SQUOZE deck until a new deck is punched. At that time, the changes are physically incorporated into the SQUOZE deck.

The effects of the modification pseudo-operations when loading a program into storage and when preparing a new SQUOZE deck are equivalent to, and could be accomplished by, making the required changes in the original symbolic source program, reprocessing with the Compiler, and then loading the new SQUOZE deck. Therefore, in the discussion that follows only the effect which the pseudo-operations have on the SQUOZE deck will be indicated.

It should be remembered throughout the discussion that each change must be indicated as though it were the only one affecting the program, regardless of the actual number. That is, all changes must be indicated in terms of the current deck and the associated listing.

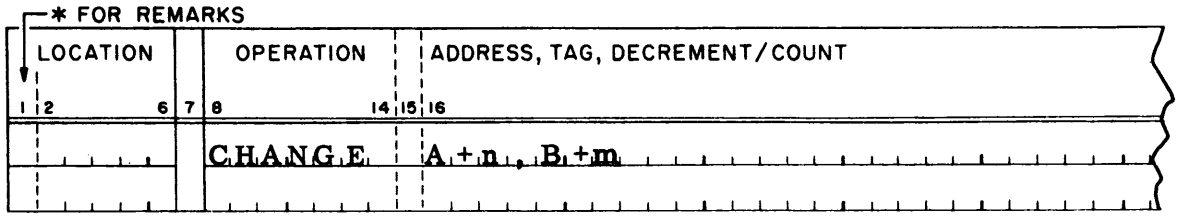
CHANGE

The CHANGE pseudo-operation can be used to delete words from a program, insert additional words in a program, or both depending on the form of the instruction. When using CHANGE, the modifications are specified in terms of relative numbers.

CHANGE instructions may be used to delete or insert words with which location symbols are associated, in which case, the location symbol is also deleted or inserted. When a word which has a location symbol is deleted, the symbol is deleted from the Dictionary, and may, therefore, be used subsequently as a location symbol for another word.

No location symbol is required with CHANGE. If one is present, it will be ignored.

Two forms of the CHANGE instruction are permissible. The first is:

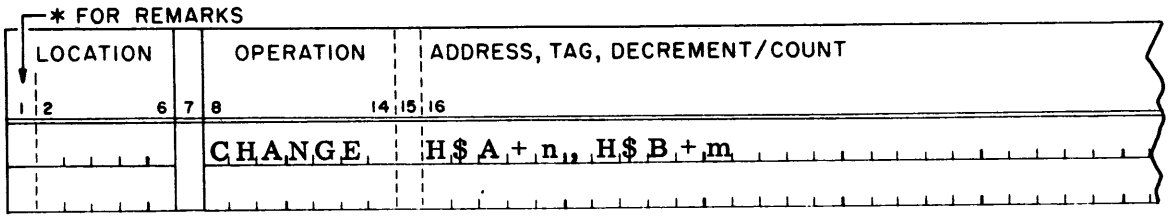


where, A + n and B + m represent relative expressions, i.e., A and B are symbols and m and n are integers which may be positive, negative, or zero.

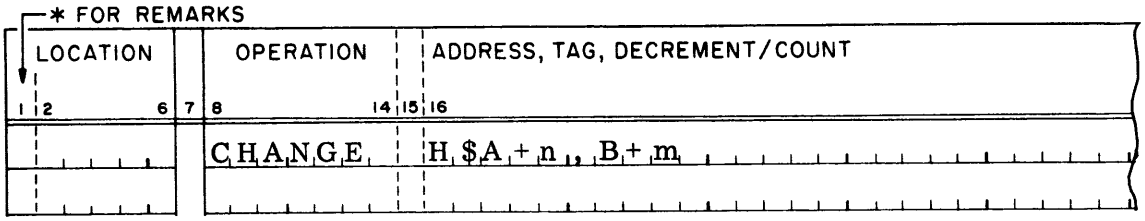
This form indicates that all words in locations A + n to B + m, inclusive, are to be deleted from the program. If, in addition, symbolic instruction cards immediately follow an instruction in this form, the instruction also indicates that the words in the symbolic cards are to be inserted beginning with location A + n. Since insertions are made as in an assembly, the words following location B + m are automatically adjusted and the number of insertions and of deletions need not be equal.

When any, but not all, of the words generated by either BCI, DEC, DUP, LBR, MACRO or OCT are deleted by a CHANGE, each of the subfields remaining from the original instruction is carried as a separate word and assigned a separate alter number. In the listing, however, only the absolute word, and relative and alter numbers are shown. No symbolic information is shown in the operation, variable and comments fields. In all other cases to which a CHANGE can apply, the comments associated with deleted words are deleted from the SQUOZE deck; remarks cards falling within the range of deletion by a CHANGE are not deleted from the program.

When a CHANGE instruction of the form shown above affects a headed area, it must be written as follows:



or alternatively

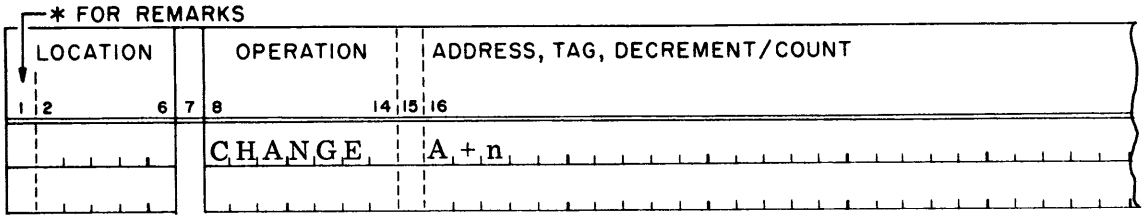


where H represents a heading character (the Hs must not be different) and A + n and B + m are as previously described.

When this latter is used, B is assumed to be also headed by H.

Note: The heading character H is required even when the symbol(s) are six characters long.

The second form permitted is:

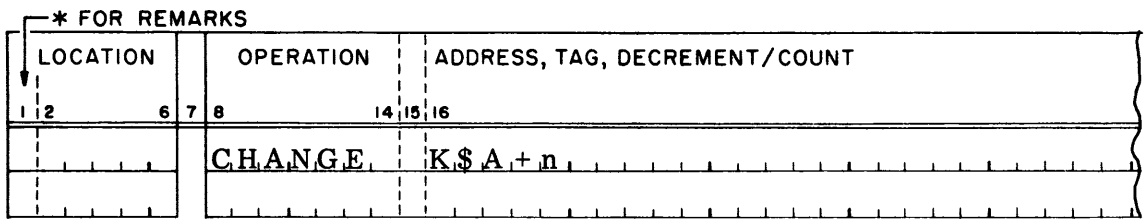


where A is a symbol and n is an integer which may be positive, negative, or zero.

This form of a CHANGE instruction indicates that the symbolic instruction cards which immediately follow it are to be inserted between the words in locations A + n and A + n + 1. No deletions are caused by this form. If no symbolic cards follow an instruction in this form the instruction is ignored.

When a generative pseudo-operation is inserted into a program by means of a CHANGE instruction, the individual terms are not assigned separate alter numbers.

When insertions are to be made in a headed area, the second form of the CHANGE instruction must be written:



where K represents a heading character and A + n is as previously described.

Here also K is required even if A is a six character symbol.

In the following list of restrictions all statements are made in terms of the headed forms of CHANGE. These restrictions can be applied to the unheaded forms by considering an unheaded symbol to be headed by the character "blank."

Restrictions:

1. In a CHANGE instruction of the first form, H$A + n must be either less than or equal to H$B + m; otherwise the CHANGE and the symbolic cards following it will be ignored.

2. No principal pseudo-operation (BES, BOOL, BSS, END, EQU, HEAD, ORG, SYN, TCD) may appear within the range of the lesser and the greater of HA, HA + n, H$B, and H$B + m. If this restriction is violated an instruction in the first form shown will cause deletion of only those words in the range H$A + n to H$B + m, if any, which precede the pseudo-operation. In no case will any insertion be made.

3. No principal pseudo-instruction, listing pseudo-instruction, or remarks card may appear as an insertion by means of a CHANGE. Any insertion which violates this restriction will be ignored.

4. Remarks cards and listing pseudo-operations cannot be deleted by a CHANGE. When remarks cards or listing pseudo-operations appear between H$A + n and H$B + m, inclusive, they will not be affected by the CHANGE.

5. No CHANGE instruction should specify the deletion of only part of the words generated by a VFD pseudo-operation.

6. If a programmer macro-instruction is inserted by means of a CHANGE, the definition must also be included with the group of modifications. This does not mean that the definition must be included with the same CHANGE that is to insert the macro-instruction. Instead, it may be included by an ALTER or by another CHANGE. The definition may also be placed in front of the group of modifications and need not be preceded by a Modify and Load pseudo-operation.

7. A modification by a CHANGE instruction must not overlap another modification by an ALTER (see below) or by a CHANGE.

Examples

1. Assume that in the following listing the instructions with alter numbers 79 and 80 are indicated to be in error.

```
78        +1   TRA    CLEAR-4        RETURN
79   EXIT      AXT    ,1
80        +1   ///     1             IF SENSE LIGHT 1 IS ON
81*                                  DONT RESTORE IR 1
```

In order to remove the error indication by means of CHANGE, the following instructions are necessary:

* FOR REMARKS

LOCATION (1, 2–6)	7	OPERATION (8–14)	15	ADDRESS, TAG, DECREMENT/COUNT (16)
		CHANGE		EXIT,EXIT+1
EXIT		AXT		**0,1
		SLT		1

Note: "**0" was arbitrarily selected to indicate modified addresses.

Assuming there were no modifications which affected the alter numbering of previous instructions in the listing, the instructions would appear in a listing of the modified deck as:

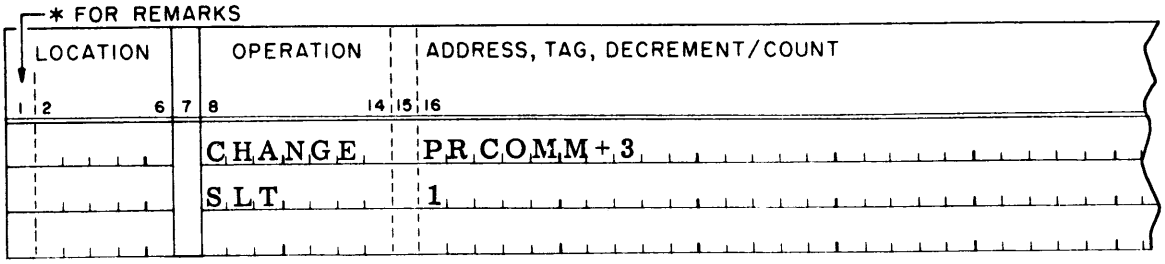
```
78        +1   TRA    CLEAR-4        RETURN
79   EXIT      AXT    **0,1
80        +1   SLT    1
81*                                  DONT RESTORE IR 1
```

(The octal absolute has been omitted for the sake of clarity; however, the absolute equivalents would also be changed.)

2. Assume that it is desired to insert the instruction SLT 1 following the instruction in the list below which has alter number 9, without deleting any instructions.

```
 6    PRCOMM    CLA     1,4             GET PRINTER CONTROL WORD
 7        +1    TMI     *+3
 8        +2    WPDA                    DOUBLE SPACE PRINTER IF
 9        +3    WPDA                    CONTROL NEGATIVE, SINGLE IF +
10        +4    SXA     RESTOR,1        SAVE INDEX
11        +5    SXA     RESTOR+1,2      REGISTER
```

The required modification cards are:



After this change is made, the listing will appear as

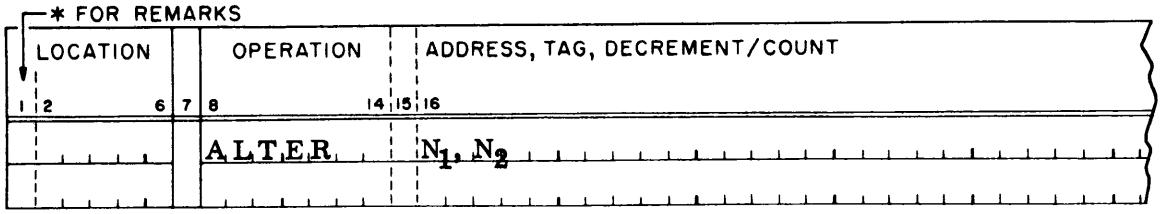
```
 6    PRCOMM    CLA     1,4             GET PRINTER CONTROL WORD
 7        +1    TMI     *+3
 8        +2    WPDA                    DOUBLE SPACE PRINTER IF
 9        +3    WPDA                    CONTROL NEGATIVE, SINGLE IF +
10        +4    SLT     1
11        +5    SXA     RESTOR,1        SAVE INDEX
12        +6    SXA     RESTOR+1,2      REGISTER
```

assuming that there are no changes which affect previous instructions.

ALTER

The ALTER pseudo-operation is analogous to CHANGE, in that it may occur in two forms similar to those of CHANGE and may be used to make insertions, deletions, or both. ALTER, however, inserts and/or deletes the equivalents of symbolic source program cards, instead of machine words.

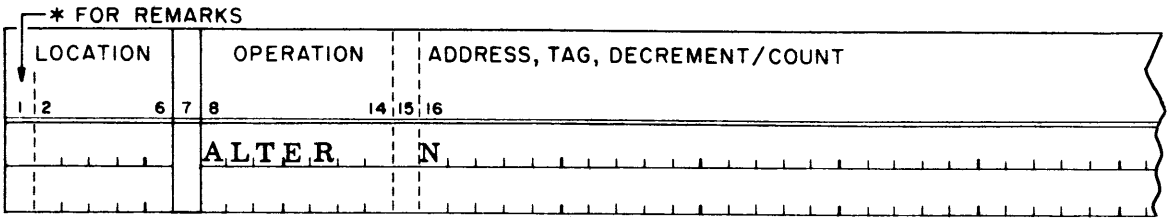
There are two permissible forms for ALTER. The first is:



where N_1 and N_2 represent alter numbers.

This form indicates that the information corresponding to alter numbers N_1 through N_2, inclusive, is to be deleted from the program. If symbolic cards are associated with an ALTER instruction in this form, the instruction also indicates that the cards are to be inserted into the program between $N_1 - 1$ and $N_2 + 1$. As with CHANGE, the number of insertions need not be equal to the number of deletions since the words following N_2 are automatically adjusted.

The second form is:



where N is an alter number.

This form indicates that no deletions are to be made, and that the associated program modification cards are to be inserted between the symbolic instructions numbered N and N + 1.

Restrictions:

1. In an ALTER instruction in the first form, N_1 must be less than or equal to N_2; otherwise the instruction and the symbolic cards to be inserted will be ignored.

2. Remarks cards, and DETAIL, EJECT, LIST, SPACE, TITLE, and UNLIST pseudo-instructions cannot be deleted by an ALTER. When an ALTER specifies alter numbers which include one of these in their range, the ALTER does not affect the remarks cards or listing pseudo-instructions.

3. An ALTER instruction cannot delete an END card without also inserting an END card.

4. An ALTER cannot insert an END instruction without deleting an END instruction. If an ALTER includes an END and does not specify the deletion of an END, the END to be inserted is ignored.

5. If a programmer macro-instruction is inserted by means of an ALTER, the definition must also be included with the group of modifications. This does

not mean, however, that the definition must be included with the same ALTER that is to insert the macro-instruction. Instead, it may be included by a CHANGE or by another ALTER. The definition may also be placed in front of the group of modifications and need not be preceded by a Modify and Load pseudo-instruction.

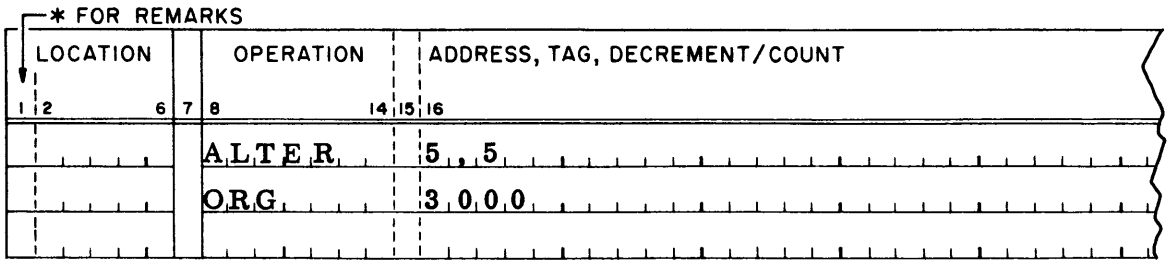
6. A modification by an ALTER must not overlap a modification either by another ALTER or by a CHANGE.

Examples

1. Assume that it is desired to correct the instruction with alter number 5 in the following listing.

```
4*
5X              ORG     START
6     PRCOMM    CLA     1,4             GET PRINTER CONTROL WORD
```

The instructions necessary to accomplish the correction are:



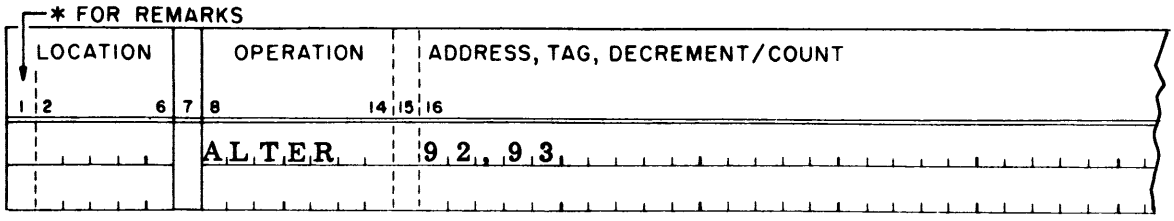
After this correction is incorporated, the listing, assuming no changes affecting the preceding remarks cards, will appear as follows:

```
4*
5               ORG     3000
6     PRCOMM    CLA     1,4             GET PRINTER CONTROL WORD
```

2. Assume that in the following listing the instructions with alter numbers 92 and 93 are to be deleted.

```
91    NUMBER    EQU     24
92    NUMBER    EQU     12
93    ZERO      EQU     0
94    TSTBIT    PZE                     STORAGE FOR TEST BIT
```

The required instruction is:

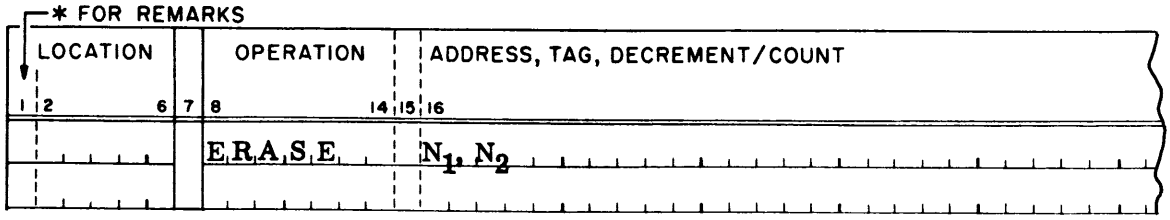


The listing after this change is made will appear (assuming no modifications affecting preceding instructions) as:

```
91    NUMBER    EQU     24
92    TSTBIT    PZE                    STORAGE FOR TEST BIT
```

ERASE

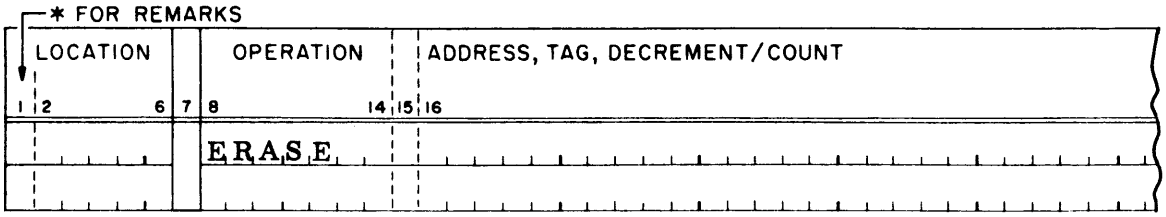
The ERASE pseudo-operation is provided in order that commentary information, i.e., comments, remarks cards, and listing pseudo-instructions, may be deleted from a SQUOZE deck. No insertions can be made by means of this operation. An ERASE modification may overlap those of CHANGE and ALTER without error. Four forms of ERASE are permissible. The first is:



where N_1 and N_2 represent alter numbers and $N_1 \leq N_2$. (Note that two alter numbers must appear in the variable field.)

This form causes the deletion of all commentary corresponding to alter numbers N_1 through N_2, inclusive.

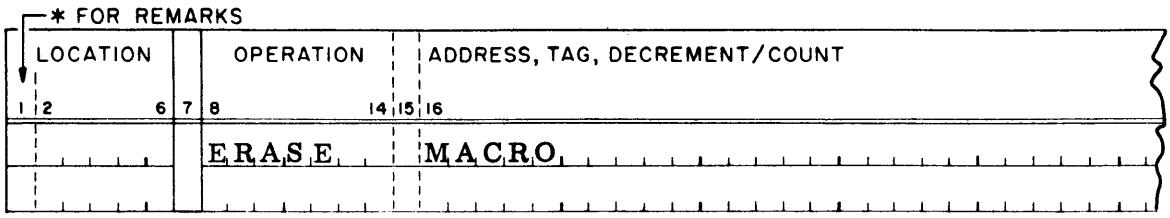
The second form of ERASE is:



and specifies the deletion of the Macro-instruction Name Table and the Macro-instruction Skeletons (see the appendix which describes the SQUOZE deck format) and of all information in the SQUOZE deck which is not essential to the execution of the program.

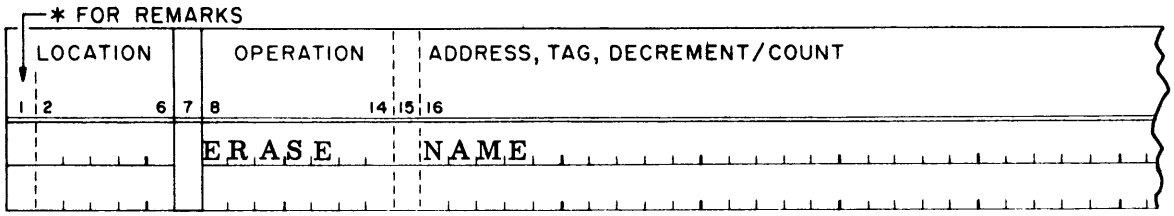
Since this form of ERASE does the work of all other forms, its presence will cause any other ERASE instruction to be ignored.

The third form is:



This form causes the deletion of the entire Macro-instruction Name Table and all Macro-instruction Skeletons from the SQUOZE deck.

Finally, the fourth form for ERASE is:



where NAME represents the name of a programmer macro-instruction.

This form causes the name of the specified macro-instruction to be deleted from the Macro-instruction Name Table, and the definition of the macro-instruction to be deleted from the Macro-instruction Skeletons.

The first two forms shown above provide the only means of deleting remarks cards and listing pseudo-operations from a SQUOZE deck.

When the commentary of generative pseudo-instructions is deleted, each word generated by such instructions is assigned a separate alter number. The words generated by BCI, DEC, and OCT are shown in the absolute portion of the listing and the alter and relative numbers for the words are shown in the symbolic portion. The operation, variable, and comments fields are left blank.

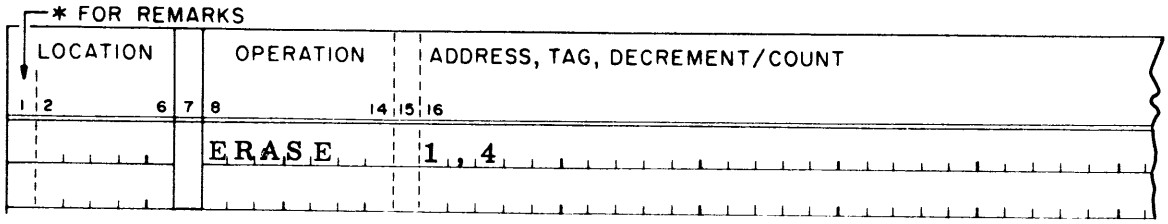
The third and fourth forms cause deletions only in the Macro-instruction Name Table and the Macro-instruction Skeletons of the SQUOZE deck. The information in the Text remains in the SQUOZE deck; however, the deletions made prevent further use in the program of the macro-instructions deleted.

Examples:

1. Assume that it is desired to delete the remarks cards at the beginning of the following listing.

```
1*   THIS IS ONLY A SAMPLE LISTING INTENDED TO ILLUSTRATE
2*     THE FORMAT OF LISTINGS PRODUCED BY SOS.
3*
4*
5X              ORG     START
6    PRCOMM     CLA     1,4             GET PRINTER CONTROL WORD
```

The instruction for doing this is:



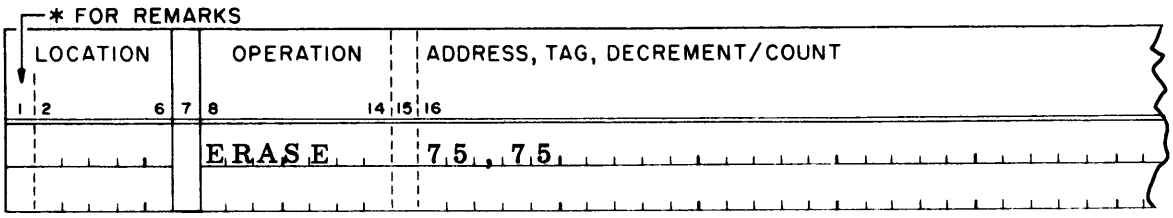
The listing of the program would then appear as follows:

```
1X              ORG     START
2    PRCOMM     CLA     1,4             GET PRINTER CONTROL WORD
```

2. Assume that it is desired to delete the comments associated with the instruction with alter number 75 in the following listing.

```
74          +7    TRA     *+2
75          +8    TRA     STRTWD          NEW WORD
76          +9    TIX     CLEAR+3,4,1     TRANSFER IF HALF DONE
```

The following instruction is used.

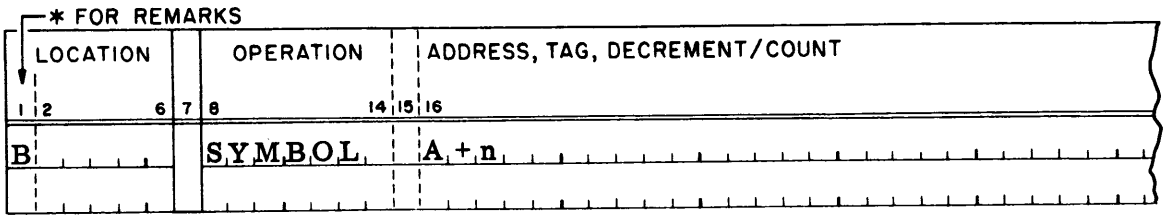


The instruction would appear in subsequent listings as:

```
74          +7    TRA     *+2
75          +8    TRA     STRTWD
76          +9    TIX     CLEAR+3,4,1     TRANSFER IF HALF DONE
```

SYMBOL

The SYMBOL instruetion permits the assignment of a location symbol to a word without requiring the deletion and subsequent insertion of the word. There is one form for a SYMBOL instruction:

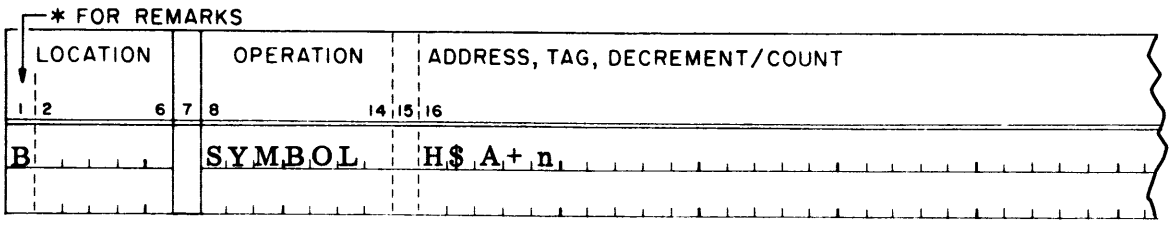


where B represents a symbol of 1-6 characters which is to become associated with the word previously assigned the relative location expression A + n.

If SYMBOL is used to associate a location symbol with a word which already has a location symbol, the new symbol does not replace the old; instead, the two are made synonymous by an EQU instruction. However, if the symbol in the location field of the SYMBOL instruction has been previously defined in the program, it is defined again with the new value and becomes a doubly-defined symbol.

It should be noted that if the location field or the variable field of a SYMBOL instruction is blank, the instruction is ignored.

When a SYMBOL instruction is to assign a symbol to a word in a headed area, i.e., when A is headed, the instruction is written



where H is the character by which A is headed, and B and A + n are as described previously.

Restrictions:

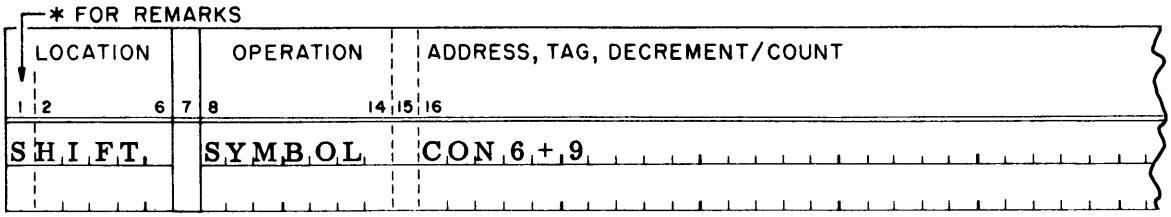
If a principal pseudo-operation appears in the range H$A and H$A + n, inclusive, (or, if A is unheaded, A and A + n, inclusive) the SYMBOL pseudo-instruction has no effect on the program.

Example

Assume that a symbol must, for convenience, be assigned the instruction with alter number 25 in the following listing.

```
16    CON6      PDX     6,2
17         +1   LGR     18              COMPUTE # INSERT WORDS +
18         +2   ADD     1,4             START ADDRESS AND
19X        +3   STO     //////          STORE.
20         +4   CLA     CON6            INITIALIZE FOR OCTAL IF TAG
21         +5   TQP     *+2             OF PRINT CONTROL IS 4.
22         +6   ARS     1               IF OUTPUT IS OCTAL STORE
23         +7   STA     STRTWD+2        3 IN CONVERSION ADDRESS
24         +8   CLA     SWITCH
25         +9   LLS     0
26        +10   STO     SWITCH
27        +11   AXT     24,1
28        +12   TCOA    *               DELAY UNTIL CHANNEL AVAILABLE
29X       +13   NOP     WKAREA+23,1
30X       +14   STZ     WKAREA+24,1     CLEAR WORK AREA
31    CLEAR     TIX     *-1,1,1         FOR CONVERSION
```

The instruction:

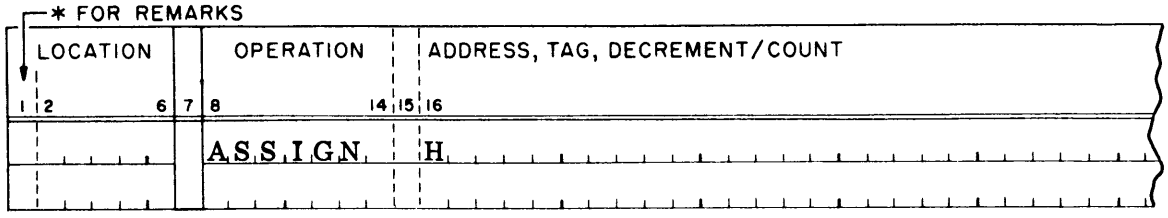


would cause the instruction to appear in a subsequent listing (assuming no other changes) as:

```
16    CON6      PDX     6,2
17         +1   LGR     18              COMPUTE # INSERT WORDS +
18         +2   ADD     1,4             START ADDRESS AND
19X        +3   STO     //////          STORE.
20         +4   CLA     CON6            INITIALIZE FOR OCTAL IF TAG
21         +5   TQP     *+2             OF PRINT CONTROL IS 4.
22         +6   ARS     1               IF OUTPUT IS OCTAL STORE
23         +7   STA     STRTWD+2        3 IN CONVERSION ADDRESS
24         +8   CLA     SWITCH
25    SHIFT     LLS     0
26         +1   STO     SWITCH
27         +2   AXT     24,1
28         +3   TCOA    *               DELAY UNTIL CHANNEL AVAILABLE
29X        +4   NOP     WKAREA+23,1
30X        +5   STZ     WKAREA+24,1     CLEAR WORK AREA
31    CLEAR     TIX     *-1,1,1         FOR CONVERSION
```

ASSIGN

The ASSIGN pseudo-instruction is provided in order that symbols may be defined or redefined by insertion of EQU, SYN, or BOOL cards. The form of an ASSIGN instruction is:



where H represents a heading character which may be a blank.

This instruction must be followed by at least one EQU, SYN, or BOOL instruction to perform one of the following functions:

1. To define new symbols and undefined symbols in a program.

2. To redefine symbols originally defined in a program by EQU, SYN, or BOOL instructions.

An ASSIGN instruction may not be immediately followed by any instruction other than EQU, SYN, BOOL, or SYMBOL. (Note that a SYMBOL following an ASSIGN does not terminate the effect of the ASSIGN.)

If an ASSIGN instruction specifies a non-blank heading character, all the symbols used in the following EQU, SYN, and BOOL instructions are headed by that character. (In addition, the only EQU, SYN, and BOOL instructions processed are those for which the location symbol has been previously defined by an EQU, SYN, or BOOL card and is not doubly-defined. Under these conditions the new definition replaces the old one.)

When an ASSIGN instruction specifies a blank heading character, the EQU, SYN, and BOOL instructions are treated as follows:

1. If the symbol in the location field of a SYN, EQU, or BOOL instruction is undefined, or is new to the program, the symbol becomes defined as usual. The EQU, SYN, or BOOL instruction defining the symbol is inserted at the beginning of the program, preceded only by remarks included at the beginning of the source program deck.

2. If the symbol in the location field of a SYN, EQU, or BOOL instruction is defined in the program by a SYN, EQU, or BOOL and is not doubly-defined, the new definition replaces the old one at the same point in the program.

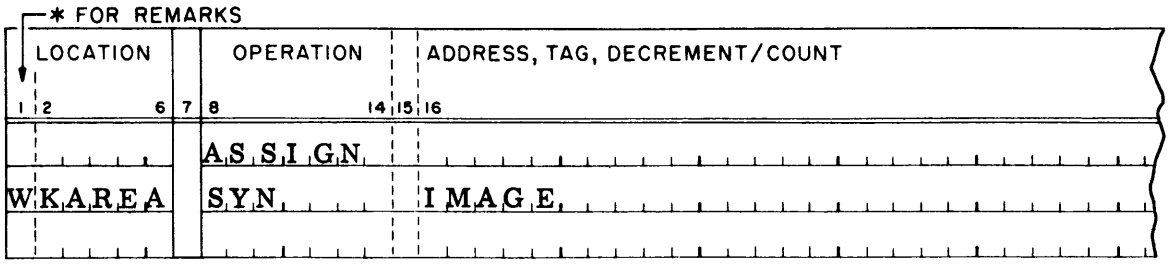
3. In all other cases, the symbol in the location field of a SYN, EQU, or BOOL instruction will be doubly-defined in the program.

When a SYMBOL card follows an ASSIGN, the location symbol will be headed by the heading character of the ASSIGN, providing the location symbol is less than six characters long.

The symbol in the variable field of the SYMBOL will also be considered headed under the same condition.

Example

Assume that the symbols WKAREA and IMAGE are to be equated in a program. The instructions necessary are:



The listing might then appear as follows (assuming no modifications affecting four remarks cards at the beginning of the source program):

```
4X
5    WKAREA    EQU     IMAGE
6              ORG     START          GET PRINTER CONTROL WORD
```

DEBUGGING SYSTEM

CHAPTER 1: GENERAL FEATURES

The Debugging System provides a set of system macros which will produce, in the terms of the source program, the status of the machine at any point in the object program, with or without specific intermediate results. The execution of the debugging macros does not affect program continuity or operation in any way; therefore, they may be ignored when considering operation of the program. The macros may be inserted by the Compiler or by Modify and Load. They can, and probably should, be deleted when the object program operates correctly.

The debugging macros fall into the following general categories:

1. Information. These are macros which cause the actual dumping of information. The macros are PANEL, CORE, TAPE, DSC, and TRAP with its corresponding UNTRAP. Of these, TRAP and UNTRAP are included as part of the IB Monitor System only.

   All information is written in its most compressed form on an intermediate tape. Execution of an object program must be terminated by a TRA SYSTEM (or, in the IB System, TRA 110) for the monitor to resume control. In its output phase, the debugging translator expands and prints the information.

2. Modal. These are macros which permit a program to specify the mode to be used in interpreting subsequent information, or to reset certain parameters used by the Debugging System.

   The modal macros are USE, FORMAT, NUCASE, BUFFER, and POINT. (The IB System also provides the ON and OFF macros to designate printing on- or off-line.)

3. Conditional. These macros may be used to control the execution of the information and modal macros.

   The conditional macros are WHEN, UNLESS, AND, OR, and EVERY. These macros set up and examine the conditions under which information and modal macros will be executed.

Although the expansion of a macro may not be changed during the running of the object program, the programmer is not necessarily confined to the specific parameters in the variable field of the macro. In every macro where feasible, provision has been made for indexing and indirect addressing which permits a program to change the parameters of a macro without changing its expansion.

The information produced by the Debugging System will normally be printed under the format controls generated in the dictionary by the Compiler or Modify and Load.

Since the dictionary contains only those locations which have attached symbols, it behooves a programmer to give a location symbol to any pseudo-op where a particular format is desired.

In the information macros (see Chapter 2), the format codes listed below may be specified and will override any dictionary information. Such a specification is effective only for execution of the macro in which it is coded.

The alphabetic codes are:

BH — BCI information, written in the binary mode. This format may only be used in the TAPE macro.

C — Data Channel command.

F — Floating point decimal number.

A number is printed as a signed fraction followed by a signed characteristic.

H — Hollerith (BCI) information.

O — Octal number.

S — Symbolic instruction with symbolic location.

The variable field will be in symbolic where applicable.

X — Fixed point decimal number.

A number is printed as a signed integer in which the fractional part is normally zero and not printed. That is, in conversion from binary, the binary point is assumed to follow the last bit. The binary point may be moved by the POINT macro, and returned to normal by the NUCASE macro.

V — A mixed format defined by a FORMAT macro (see page 06.03.06). V may only be used in a CORE macro which is preceded by the defining FORMAT macro.

DEBUGGING SYSTEM

CHAPTER 2: INFORMATION MACROS:
Share Monitor — PANEL, CORE, TAPE, DSC
IB Monitor — PANEL, CORE, TAPE, DSC, TRAP, UNTRAP

The information macros cause the Debugging System to save information for subsequent printing. PANEL, DSC, TRAP, and UNTRAP do not require or use information in the variable field. In the remaining macros, CORE and TAPE, the variable field controls the amount of information and its form. The coding of these fields will be discussed with each macro.

A PANEL macro is executed automatically prior to the execution of any other information macro, and is not repeated in any given set.

PANEL

This macro causes the following information to be printed:

1. Contents of the P and Q bits of the AC.

2. Contents of the AC, excluding P and Q, in octal and in floating point decimal.

3. Contents of the MQ in octal and in floating point decimal.

4. Status of the sense indicators, as an octal number.

5. Status of the AC overflow, divide check and I/O check. "ON" or "OFF" is printed for each.

6. Contents of each index register, in octal and decimal.

7. Status of sense lights, as a binary number in which 0 indicates that the corresponding light is Off, and 1 indicates that it is On.

8. Settings of sense switches, as a binary number in which 0 indicates that the corresponding switch is Up, and 1 indicates that it is Down.

9. SHARE Monitor: The location at which the debugging macro was encountered.
   IB Monitor: The setting of the entry keys, as an octal number.

Example of PANEL output (IB Monitor):

```
    DT8A+1        PANEL

AC  00 + 002000 001511  +.00000000+00  MQ - 002200 000800  -.29387359-38  SI  000000 300000  OVFL OFF  DC OFF  IOC OFF
IR1  77772  32762   IR2  77400  32512   IR4  76570  32120                 SL 0000   SW 100100   EK + 077200 001001
```

CORE LOC1, LOC2, Q, IT1, IT2

This macro dumps a section of memory according to the subfields which are interpreted as follows:

LOC1 and LOC2 — These subfields designate the first and last locations of the area to be dumped. They may be symbolic references or decimal numbers, and are subject to conventional address modification by the IT1 and IT2 subfields, if coded (see below).

If LOC1 and/or LOC2 is omitted, the range of the dump will be interpreted as follows:

LOC1 = system origin
LOC2 = top of memory

LOC1 must be less than or equal to LOC2.

Q — This represents the format code to be used in printing the dump. This field should be coded only if the dictionary format is inadequate. The CORE macro may use the entire range of formats, X, F, S, O, C, H and V. V, of course, must be preset by a FORMAT macro (page 06. 03. 06).

IT1 and IT2 — These subfields apply to LOC1 and LOC2 respectively. In each, I may be coded before or after T, and either or both may be omitted. T1, for example, is coded as the index register modifying LOC1, and I, coded as "I", causes the effective location to be indirectly addressed.

Examples of the CORE macro, and their results are:

CORE

system origin through top of memory, in dictionary format

CORE TINKER

location TINKER through top of memory, in dictionary format

CORE , EVERS

system origin through location EVERS, in dictionary format

CORE EVERS, CHANCE, X

location EVERS through location CHANCE, as fixed point decimal numbers

CORE TINKER, TINKER, F

word in location TINKER, as a floating point decimal number

CORE EVERS, EVERS+99, X

100 words, beginning at location EVERS, as fixed point decimal numbers

CORE 0, 1, S

locations 0 and 1, as symbolic instructions

CORE TINKER, EVERS, , I

location TINKER, indirectly addressed, through location EVERS, in dictionary format

CORE EVERS, CHANCE, , 2, 4

location EVERS, modified by index register 2, through location CHANCE, modified by index register 4, in dictionary format

CORE TINKER, EVERS, X, I2, 4I

location TINKER, modified by index register 2 and indirectly addressed, through location EVERS, modified by index register 4 and indirectly addressed, as fixed point decimal numbers

TAPE

SHARE System: TAPE T, M, R, W, Q, N
IB System: TAPE T, R, W, Q, N

This macro records the contents of a tape record, or records, in whole or in part. The subfields specify the tape from which information is to be dumped, the amount of information to be dumped, how it is to be printed, and whether the tape should be repositioned prior to return to the object program. Variable length records do not affect the operation of this macro. The uppermost 256_{10} words of memory will be used as temporary storage, unless a BUFFER macro has changed the length and location of available storage. The storage is not restored.

This macro will re-read a record which results in a redundancy check. A second failure will cause a comment to be printed before the printout of the record.

Note that there are incompatibilities between the requirements of this macro for the SHARE System and for the IB System.

The initial subfields are interpreted as follows

in the SHARE System:

T This is the symbolic name of the tape which has been assigned by the SHARE Monitor. See Section 09 for tape names and assignments.

M This field specifies the mode in which the symbolic tape was written.

The field is coded:

D for decimal, that is, BCD
B for binary

Omission of the field is interpreted as binary.

in the IB System:

T This field is coded as a decimal number which specifies both the physical unit and the mode in which the tape was written. For example:

657_{10} (1221_8) = tape A1, in the binary mode
1154_{10} (2202_8) = tape B2, in the BCD mode

The remaining subfields are common to both systems.

R This field specifies the number of tape records to be processed by this macro. The number is ignored and processing terminates if an end-of-file, or beginning-of-tape is encountered.

The number will be interpreted as follows:

n	the n records following the initial position.
-n	the n records which precede the initial position. Records are printed in inverse order relative to their position on tape. For example, record 3 is printed before record 2.
0	all records following the initial position until an end-of-file is encountered.
-0	all records which precede the initial position until an end-of-file or beginning-of-tape is encountered.

If this field is omitted, it will be interpreted as 0, that is, read forward until end-of-file.

W This field specifies the number of words to be dumped from each record. If this field is omitted, is coded 0 or -0, or if it exceeds the number of words in a record, the entire record will be printed. Otherwise, the number is interpreted as follows:

n	the first n words in a record.
-n	the last n words in a record, in their proper sequence.

Q This specifies the format to be used in printing the record. The TAPE macro may only specify F, X, H, BH, or O. Any other format, or omission of this field, will be interpreted as octal.

N This field indicates whether or not the tape is to be returned to its initial position after execution of this macro. The normal case is to reposition, so, while this field may be coded 0, it is usually omitted.

If no repositioning is desired, the field is coded 1.

Examples:

SHARE Monitor:

```
TAPE    SYSAR1,D,2,,X
```

All words in next 2 records on tape SYSAR1 are read in the decimal mode and the tape is repositioned. The information is printed as fixed point numbers.

IB Monitor:

```
TAPE      658,-0,2
```

The tape is backspaced a record at a time and the first 2 words are saved from each record. Processing is terminated by beginning-of-tape signal, or by end-of-file. The tape is repositioned. The information is printed in octal.

DSC

This macro records the register contents of each Data Synchronizer Channel. The information is expanded and printed to show, for every channel:

1. location of the last command processed
2. octal representation of the command
3. symbolic location of the command
4. symbolic operation code of the command
5. symbolic address, which is 1 greater than the address of the last word processed by the command
6. channel designation

Example of printed results:

```
DSC

 C   06231   +2 00000 0 05670        C1      IORP    TP1              CHANNEL A
 C   02032   +0 00000 0 02036        -1050   IOCD    1054             CHANNEL B
 C   77777   +0 00000 0 00000       +29502   IOCD    0                CHANNEL C
 C   77777   +0 00000 0 00000       +29502   IOCD    0                CHANNEL D
 C   77777   +0 00000 0 00000       +29502   IOCD    0                CHANNEL E
 C   77777   +0 00000 0 00000       +29502   IOCD    0                CHANNEL F
 C   77777   +0 00000 0 00000       +29502   IOCD    0                CHANNEL G
 C   77777   +0 00000 0 00000       +29502   IOCD    0                CHANNEL H
```

Note that the information clearly indicates that channels C-H are not attached to the 709 on which the problem was run. There is, at present, no way to suppress printing of this information.

TRAP

This is the only macro which causes the printing of information but does not also produce a PANEL, and which requires another macro, the terminating UNTRAP.

The 709 manual specifies fully the operation of the machine in the trapping mode. This macro keeps a table of any transfer instruction that the machine is able to trap, with the exception of transfers within the Debugging System. The entrance into, and the exit from, the Debugging System are included in the table.

The table is automatically dumped on a system intermediate tape only when TRAP is not able to make the next entry. Therefore, a complete transfer trace can only be obtained by using the UNTRAP macro to dump remaining entries.

The trapped instructions are printed as a 50-entry table which shows the location at which an instruction was trapped, the octal representation of the instruction and the instruction in symbolic. The table is printed after any information printed by those debugging macros which lie within the range of that table.

The TRAP macro offers another device which can aid in locating errors in the flow of a program. Except at the initial entry into the TRAP macro, all debugging macros which immediately follow a TRAP are executed every time an instruction in the object program is trapped by the machine. For example:

```
TRAP
PANEL
```

would produce every branch of flow in a program, and the status of the machine at that point.

UNTRAP

This macro causes the 709 to revert to its normal mode of operation; the remaining entries in the table of trapped instructions are written on the intermediate tape for subsequent printing.

DEBUGGING SYSTEM

CHAPTER 3: MODAL MACROS:
SHARE System: USE, FORMAT, POINT, NUCASE, and BUFFER
IB System: USE, FORMAT, POINT, NUCASE, BUFFER, ON, and OFF

The modal macros reset parameters used by the Dubugging System either during execution of the object program or during translation of debugging data. During the execution phase, BUFFER allocates storage to the TAPE macro and NUCASE nullifies prior modal macros and resets the counts of conditional macros. The other modal macros, USE, FORMAT, and POINT, have no effect during the running of the object program, but rather are used by the debugging translator in expanding data recovered by the information macros. These macros modify certain aspects of that data which cannot be controlled by a format indication in the dictionary, for instance, choice of symbols for locations common to several sections of a program; refinement of format control to apply to groups of bits rather than whole words; and rescaling of the fraction in interpretation of fixed point numbers.

Only NUCASE does not require or use information in the variable field.

USE A, B, C, ...

This macro designates those symbols in the dictionary which are to be associated with locations common to two or more segments of a program. The macro influences information in all subsequent printing until another USE supercedes it.

The variable field should contain at least one symbol which is unique to the common segment currently being dealt with by the debugging macros. Usually, a common segment should have location symbols at the first and last locations, and if these two symbols are given in a USE macro, there will be no ambiguity of symbols in the printing.

For instance, the program which deals with the storage block JONES to TINKER, and then reads over it the block EVERS to CHANCE, should have:

USE EVERS, CHANCE

coded before any information macros are permitted to record information from the last block.

POINT n

In the information macros which follow a POINT macro, the words to be converted to fixed point decimal are first considered as fractions in which the binary point lies between the nth and the n+1th bits.

n is a decimal number ranging from 0 to 35; the sign of the word is not considered. The effect of this macro may be reset by another POINT. NUCASE will also return it to normal conversion which is effectively POINT 35.

BUFFER LOC1, LOC2, IT1, IT2

The subfields of this macro specify the temporary storage to be used by a subsequent TAPE macro. Temporary storage is not restored. This macro must be used when it is necessary to prevent TAPE from reading into the uppermost 256_{10} words of memory. For records exceeding standard buffer size (256_{10} words), considerable time will be saved if the block reserved by BUFFER is at least as long as the maximum record being recovered by TAPE.

This macro affects allocation of storage until another BUFFER macro is encountered. The NUCASE macro will reset the allocation to the top 256_{10} words of memory.

The subfields are interpreted as follows:

LOC1 and LOC2	These subfields designate the first and last locations of a temporary storage buffer. They may be symbolic references or decimal numbers, and are subject to conventional address modification by the IT1 and IT2 subfields, if coded.
IT1 and IT2	These subfields apply to LOC1 and LOC2, respectively. In each, I may be coded before or after T, and either or both may be omitted. T1, for example, is coded as the index register modifying LOC1, and I (coded as "I"), causes the effective location to be indirectly addressed.

NUCASE

This macro initializes parameters of certain debugging macros. POINT n is restored to POINT 35; BUFFER allocates the uppermost 256_{10} words of memory as temporary storage for any TAPE macro; and the counts generated by the count type conditional macros (see Chapter 4) are all reset to zero.

In the IB System only, execution of NUCASE terminates any on-line printing initiated by the ON macro.

FORMAT n1, Q1, n2, Q2, . . . , ni, Qi

This macro defines the pattern to be used in printing the blocks of words to be dumped by a subsequent CORE macro which has been coded with V in its format subfield.

The block may have been compiled by a VFD pseudo-op, or in a heterogeneous format, and its length is limited to the number of words in memory which will just contain the number of bits being specified.

In this macro, n1 specifies the number of bits at the beginning of the block which are to be printed in format Q1 (see below), n2 specifies the next number of bits to be printed in format Q2, etc.

No subfields may extend into the next word and unspecified bits in the terminal word of the block will not be printed. The pattern will be repeated for the next block of words beginning with the first n1 bits of the first word of the block. The formats which may be specified by Q in this macro are limited to S, X, O, and H. These are further limited as follows:

(1) X will always cause POINT 35 format regardless of a previous POINT macro

(2) if any Q is H, the corresponding n should be a multiple of 6

(3) S will only be interpreted as a location symbol or relative to a location symbol.

ON

This macro causes subsequent debugging output to be printed on-line when finally converted to the output form.

The effect of ON is terminated by an OFF or a NUCASE macro-instruction.

OFF

OFF is used to terminate the effect of an ON without executing a NUCASE and thereby affecting other macros which it is not desired to change.

CHAPTER 4: CONDITIONAL MACROS: WHEN, UNLESS, EVERY AUXILIARY CONDITIONAL MACROS: AND, OR

The conditional macros are a means of restricting the conditions under which modal or information macros will be executed, and thus of ensuring the relevance limiting the quantity, and controlling the form of debugging output.

A conditional macro proper (WHEN, UNLESS, and EVERY) is effective only if the condition specified by its variable field is satisfied. When effective, WHEN and EVERY permit the execution of the next macro in sequence; UNLESS prevents the execution of all subsequent macros in the set. When the condition is not satisfied, the effect of a conditional macro is reversed. AND and OR are auxiliary to, and governed by, a preceding WHEN or UNLESS (not EVERY). They specify, respectively, an additional or an alternative condition for the WHEN or UNLESS. AND and OR are logically a part of the preceding WHEN or UNLESS and should not be considered independent macros.

Thus, for instance, a WHEN followed by ANDs and ORs must be considered a WHEN with a complex condition, and the "next macro in sequence" would then be the UNLESS, EVERY, modal or information macros immediately following the last AND or OR attached to the WHEN.

The control exercised by a conditional macro extends to the end of the continuous set of debugging macros in which it occurs. The control is terminated by any instruction which is not a debugging macro.

The condition specified by the variable field may be a simple count expressed by a decimal number. This number is compared with the contents of a counter which is maintained for each macro and incremented on each execution of the macro. The interpretation of the result of this comparison depends on the macro concerned and is discussed below under the heading for the particular macro.

A second form of the variable field is provided for all conditional macros except EVERY. This form uses five subfields to specify two values and the relationship which must exist between them in order for the macro to be effective.

This second form is:

COMP1, R, COMP2, IT1, IT2

The variable subfields are interpreted:

COMP1 and COMP2

These fields designate the items to be compared. The description which follows for COMP1 applies equally to COMP2.

1. If COMP1 is zero or is omitted, or is a symbol whose value is zero, and IT1 is omitted, the designated value is +0.

2. If COMP1 is any number from 1 to 7, or a symbol with such a value, and IT1 is omitted, then the designated value is obtained from the index register(s) specified. "or" of index registers is possible. The contents of the index register is treated, for purposes of comparison, as a positive 36-bit fixed-point number less than 2^{15}.

3. If neither of the two conditions above are met, then COMP1 denotes the location in memory from which the value is to be obtained, and this subfield is subject to conventional address modification by the subfield IT1.

R

This field states the relationship which must exist between the values specified by COMP1 and COMP2 in order for the condition to be satisfied.

This subfield is coded:

L - less than
E - equal to
G - greater than
LL - logically less than
LE - logically equal to
LG - logically greater than

IT1 and IT2

These subfields apply to COMP1 and COMP2, respectively. I may be coded before or after T, and either, or both, may be omitted. T1 is coded as a number from 1 to 7 and denotes the index register modifying the address shown in COMP1, and I, coded simply as I, causes indirect addressing.

In the examples below, C signifies "the contents of" and IC "the indirect contents of." Thus, IC(X, 2) is that number obtained in the accumulator by the execution of the instruction CLA* X, 2 (or CAL* X, 2 for a logical comparison).

1. 4, E,

   The condition is: C(IR4) equal to zero.

2. 2, L, 2, I

   The condition is: IC(2) less than C(IR2)

3. A, LG, B, 2, 4I

   The condition is: C(A, 2) logically greater than IC(B, 4)

4. 1000, E, X, , 4

   The condition is: C(1750_8) equal to C(X, 4)

WHEN

A WHEN specifies a condition which must be satisfied to allow execution of subsequent macros. Thus, WHEN (C1) may be read as "proceed to execute the next macro when the condition (C1) is satisfied but not otherwise." In other words, if the condition is satisfied, the WHEN is effective and the next macro in sequence is executed; otherwise all subsequent macros in the set are bypassed.

WHEN n where n is a decimal number

The condition is satisfied, and the WHEN is effective (in the absence of qualifying ANDs and ORs), on the nth execution of the WHEN and on every execution thereafter.

WHEN COMP1, R, COMP2, IT1, IT2

If the condition is satisfied, as described on page 06.04.02, then (in the absence of qualifying ANDs and ORs) the WHEN is effective.

UNLESS

An UNLESS specifies a condition which must be unsatisfied to allow execution of subsequent macros. Thus, UNLESS (C1) may be read as "proceed to execute the next macro unless condition (C1) is satisfied."

In other words, if the condition is satisfied, the UNLESS is effective and all subsequent macros in the set are bypassed; otherwise, the next macro in sequence is executed.

UNLESS n where n is a decimal number

The count for the UNLESS is increased by one each time ONE of the set of information macros following it is executed. When the count reaches n, the set of information macros then being executed is completed, and the UNLESS becomes effective.

UNLESS COMP1, R, COMP2, IT1, IT2

If the condition is satisfied, then (in the absence of qualifying ANDs and ORs) the UNLESS is effective.

AND

An AND specifies a condition, in addition to all previous conditions for the governing WHEN or UNLESS, which must be satisfied before the governing macro becomes effective.

AND n where n is a decimal number

The condition is tested according to the rule for the governing macro (WHEN, UNLESS). Thus, for instance, the condition specified by an AND n following a WHEN is satisfied on the nth execution of the WHEN and on all subsequent executions.

AND COMP1, R, COMP2, IT1, IT2

The condition is tested as described on page 06. 04. 02.

If the condition specified by an AND is satisfied and, in addition, the governing macro is effective in the absence of the AND, then the governing macro remains effective. In all other cases, the governing macro is ineffective.

OR

An OR specifies a condition which is alternative to all conditions previously specified for the governing WHEN or UNLESS. The satisfaction of the OR makes the governing macro effective, whatever its previous status.

OR n where n is a decimal number

The condition is tested according to the rule for the governing macro.

OR COMP1, R, COMP2, IT1, IT2

The condition is tested as described on page 06. 04. 02.

If the condition specified by an OR is not satisfied, and, in addition, the governing macro is ineffective in the absence of OR, then the governing macro remains ineffective. In all other cases the governing macro is effective.

EVERY

The variable field of EVERY has only one form. The macro provides a means of preventing output, or modal control, except at predetermined intervals.

EVERY n where n is a decimal number

The condition is satisfied, the EVERY is effective, and the next macro in sequence is executed, only on the first execution of the EVERY and on every nth execution thereafter; that is, on the 1st, (n+1)th, (2n+1)th, etc.

On all other executions, all subsequent macros in the set are bypassed.

COMBINATIONS OF CONDITIONAL MACROS

1. An UNLESS may precede or follow a WHEN (in fact, pairs of these macros may be interchanged without affecting output), but no conditional macro may follow an EVERY.

2. The counter for a macro is incremented on every execution of that macro (except as noted for the case of an UNLESS macro), and not necessarily on every entry to the set of macros.

3. Any number of ANDs and ORs may follow WHEN or UNLESS. Every time the governing WHEN or UNLESS is executed all the conditions thus specified are tested, in sequence, and the "effective" status of the governing macro is modified continually in accordance with the results of the tests. For example, consider the sequence:

```
WHEN    (C1)
AND     (C2)
OR      (C3)
OR      (C4)
AND     (C5)
```

where C_i is a condition to be tested.

Let B_i equal 1 if C_i is satisfied; and otherwise zero. Then the WHEN is effective if $((((B_1 \times B_2)+B_3)+B_4) \times B_5)$ is equal to 1 (where the algebraic operators are Boolean).

It will be seen that the limitations of the language prevent the representation of certain combinations of conditions; for instance (C_1 AND C_2) OR (C_3 AND C_4).

Examples of combinations of conditional macros.

1.
```
WHEN    3
UNLESS  3
CORE    A,B
```

The WHEN is effective, and the UNLESS is executed, on the third execution of the WHEN and thereafter. Thus, the CORE is executed on the third, fourth, and fifth entries to the sequence.

2.
```
UNLESS  3
WHEN    3
CORE    A,B
CORE    C,D
```

The CORE macros are executed only on the third and fourth entries to the sequence.

3.
```
FORMAT    (F1)
WHEN      (C1)
UNLESS    5
FORMAT    (F2)
NOP
CORE      A,A,V
```

The contents of location A are always printed. On the first five times condition (C1) is satisfied, format (F2) is used; on all other occasions format (F1) is used.

4.
```
WHEN      10
UNLESS    2
EVERY     10
PANEL
```

The PANEL is executed only on the 10th and 20th entries to the sequence.

5.
```
WHEN      (C1)
OR        (C2)
AND       (C3)
EVERY     5
CORE      A,B
```

The CORE is executed on the 1st, 6th, 11th, etc., times that condition (C3) and either condition (C1) or condition (C2) are satisfied.

6.
```
UNLESS    (C1)
AND       (C2)
OR        (C3)
WHEN      4
PANEL
```

The PANEL is executed every time, after the third, that neither condition (C3) nor both conditions (C1) and (C2) are satisfied.

7.
```
UNLESS    8
CORE      A,B
CORE      C,D
CORE      E,F
```

The CORE macros are executed on the first, second, and third entries to the sequence.

CHAPTER 5: EXPANSIONS OF DEBUGGING MACROS

The expansions of the Debugging Macros given below are those produced when the SHARE Monitor version of SOS is used. * In the expansions produced by the IB Monitor version, the symbol SYSDB1 is replaced by 2169_{10}, and SYSDB2 is replaced by 1819_{10}.

(1) WHEN L_1, R, L_2, IT_1, IT_2

```
STL     SYSDB1
TXL     SYSDB2,0,1
PZE     L1,0,L2
VFD     H12/R,H12/IT1,H12/IT2
```

(2) UNLESS L_1, R, L_2, IT_1, IT_2

```
STL     SYSDB1
TXL     SYSDB2,0,2
PZE     L1,0,L2
VFD     H12/R,H12/IT1,H12/IT2
```

(3) AND L_1, R, L_2, IT_1, IT_2

```
STL     SYSDB1
TXL     SYSDB2,0,3
PZE     L1,0,L2
VFD     H12/R,H12/IT1,H12/IT2
```

(4) OR L_1, R, L_2, IT_1, IT_2

```
STL     SYSDB1
TXL     SYSDB2,0,4
PZE     L1,0,L2
VFD     H12/R,H12/IT1,H12/IT2
```

(5) EVERY N

```
STL     SYSDB1
TXL     SYSDB2,0,5
PZE     N
```

* In some instances, PZE will be replaced by HTR.

(6) CORE L_1, L_2, F, IT_1, IT_2

```
STL     SYSDB1
TXL     SYSDB2, 0, 7
PZE     L1, 0, L2
VFD     H12/F, H12/IT1, H12/IT2
```

(7) TAPE T, M, R, W, F, N (SHARE Monitor)

```
STL     SYSDB1
TXL     SYSDB2, 0, 8
VFD     H3/M, 15/T, 6/0, H12/F
VFD     1/N, 2/0, 15/W, 3/0, 15/R
BCI     1, T
```

(8) TAPE T, R, W, F, N (IB Monitor)

```
STL     LOC
TXL     STPAN, 0, 8
VFD     6/0, 12/T, 6/0, H12/F
VFD     1/N, 2/0, 15/W, 3/0, 15/R
```

(9) DRUM D, L_1, L_2, F, IT_1, IT_2 (IB Monitor)

```
STL     LOC
TXL     STPAN, 0, 9
VFD     3/D, 15/L2, 3/0, 15/L1
VFD     H12/F, H12/IT1, H12/IT2
```

(10) PANEL

```
STL     SYSDB1
TXL     SYSDB2, 0, 10
```

(11) DSC

```
STL     SYSDB1
TXL     SYSDB2, 0, 11
```

(12) USE $A_1, A_2, . . , A_n$

```
STL     SYSDB1
TXL     SYSDB2, 0, 12
PZE     2n+3
BCI     1, A1
PZE     A1
BCI     1, A2
PZE     A2
  .
  .
  .
BCI     1, An
PZE     An
```

(13) FORMAT $B_1, F_1, B_2, F_2, \ldots, B_n, F_n$

```
STL     SYSDB1
TXL     SYSDB2, 0, 13
PZE     (no. words in this calling seq.)
VFD     6/B1, H6/F1, . . . , 6/Bn, H6/Fn
```

(14) POINT N

```
STL     SYSDB1
TXL     SYSDB2, 0, 14
PZE     N
```

(15) BUFFER L_1, L_2, IT_1, IT_2

```
STL     SYSDB1
TXL     SYSDB2, 0, 15
PZE     L1, 0, L2
VFD     12/0, H12/IT1, H12/IT2
```

(16) ON (IB Monitor)

```
STL     LOC
TXL     STPAN, 0, 16
```

(17) OFF (IB Monitor)

```
STL     LOC
TXL     STPAN, 0, 17
```

(18) NUCASE

```
    STL     SYSDB1
    TXL     SYSDB2, 0, 18
```

(19) TRAP (IB Monitor)

```
    STL     LOC
    TXL     STPAN, 0, 19
```

(20) UNTRAP (IB Monitor)

```
    STL     LOC
    TXL     STPAN, 0, 20
```

INPUT/OUTPUT SYSTEM

CHAPTER 1: THE INPUT SYSTEM — INTRAN

This chapter deals with the INTRAN vocabulary of SOS. This vocabulary consists entirely of system macros.

INTRAN provides the programmer with two tools:

1. A large set of fundamental subroutines, each of which performs one of the basic input functions required for a general class of external information.

2. An easy means of specifying that one of these fundamental subroutines is to be used, i.e., by a single macro.

These basic subroutines may in turn be used as building blocks to construct higher-level subroutines. Such a higher-level subroutine might be designed to cover part or all of the input processing required in a very large class of problems, and thus be made a standard input program. The nature and range of these input programs will vary widely and may be chosen to suit the particular needs of the installation. By using the INTRAN macros the programmer will find that the task of constructing such programs is greatly simplified.

On the other hand, for a given production problem with certain input requirements (perhaps peculiar to the problem), the programmer may choose to write his own subprograms, as part of his total source program. In such a case, he will find that the direct use of the INTRAN macros in his program will be of great value.

RULES FOR SPECIFYING INTRAN MACROS

The general rules for specifying any SOS instruction (see Section 02) apply to all INTRAN macros. For example, the location field of a macro may contain a symbol, the variable field is divided into subfields separated by commas, etc.

As is the case of all macro-instructions, a location symbol of an INTRAN macro will be associated with the first word generated by the instruction, i.e., the location symbol will be entered into the dictionary of symbols with the value assigned to the first word generated by the macro-instruction.

A list of all the INTRAN macro-operations and their expansions is given on page 07.01.53. Those macro-operations for which indirect addressing is permissible are so indicated. Indirect addressing is, as usual, specified by placing the character "*" at the end of the operation code.

The list of macros also indicates the pattern of the variable field of each macro. As is evident, the number of subfields which must be specified in the variable field is fixed for each macro, but among all the macros, this number varies from zero to at most six. The programmer may, of course, specify zero values for the last n subfields, along with their separating commas. The roles played by these various subfields and the rules for specifying the subfields depend, in general, on the operation, and are discussed below.

By definition, a macro-instruction always generates, or "expands into", one or more machine words. The number of words in the expansion of any macro depends only on the macro-operation used in the instruction, and not on the values of the expressions in the variable field. In fact, every SOS macro (whether a programmer or a system macro) has this property, i.e., its use in a macro-instruction always results in a fixed number of generated words, where this number is determined solely by the macro-operation itself.

The expansions for the different Input macros vary in size from two to at most five words. For example, the INTRAN macro will always generate two words, whereas the IFLOAT macro will always generate five words. The number of words generated by a given INTRAN macro and the number of sub-fields required in the variable field of such an instruction are directly related. The relation between these two numbers is due to the fact that the set of words generated by a macro-instruction is simply a calling sequence, which, of course, must contain all of the information specified in the variable field of the macro-instruction.

It can be seen that the first two words generated by any INTRAN macro are an STL instruction and a TXL instruction. When these two instructions are later executed, their effect will be to save the contents of the instruction location counter and to transfer control to the INTRAN program. The INTRAN program consists of a set of subroutines which are executed at object program time. These subroutines carry out the functions prescribed by the particular macro, using the information which it finds in the words, if any, immediately following the TXL instruction in the macro expansion. Finally, control will be transferred back to the first word immediately following the expansion, and execution of the object program will continue. Note: For certain macro-operations, and under certain conditions (such as error conditions, etc.) which will be discussed later, control may be transferred to some special location.

SPECIAL REGISTERS AND INDICATORS

Whenever the INTRAN program is entered by a calling sequence generated by a macro, the contents of the AC and MQ, and the status of the AC Overflow indicator are destroyed. However, the INTRAN program will not disturb the contents of any of the three index registers, the Sense Indicator register, the status of the Sense lights, or any of the special indicators on the 709/7090. However, it is possible that the status of the Tape Check indicator, the End-of-Tape indicator, or the End-of-File indicator associated with a given Data Synchronizer Channel will be affected by an ISCRIB or IREADY instruction which uses that channel for tape operations.

PURPOSE OF THE INPUT SYSTEM

The Input System was designed to simplify the problem of obtaining information to be processed by an object program. This information will usually originate from Hollerith punched cards, and processing of the information will usually require conversion, particularly from decimal to binary.

The external information which can be processed by the Input System must consist of one or more "unit records." A unit record may take on any of the four following physical forms:

1. A card punched with 72 columns of Hollerith information to be read on-line by a 709 card reader. (Hollerith cards, when read, are automatically converted to BCD form.)

2. A BCD tape record consisting of n words (i.e., 6n characters), where n is a positive integer. The usual case will be n = 14 (i.e., 84 characters), arising from an 80-column card written on tape by off-line card-to-tape operation.

3. A 72-column column binary (rather than row binary) card to be read on-line by a 709 card reader.*

4. A binary tape record consisting of 28 words. Such a record normally originates from a column binary card which has been written on tape by off-line card-to-tape equipment.

---

* By convention, a card punched in column binary must contain 7- and 9-punches in column 1 to distinguish it from a Hollerith card. This simultaneous occurrence of 7- and 9-punches, which may be accompanied by punches in any of the other rows of column 1, is of course impossible in a card punched in Hollerith code.

The binary information contained in a unit record of form 3 or 4, above, can be quite arbitrary. Since this information is already represented in binary, there is no conversion problem. The only concern of the Input System with such a record will be to read it into core storage.

The major part of the Input System is devoted to the problem of extracting strings of BCD characters (six-bit groups) from a record of the form 1 or 2, and converting these strings (which may represent floating point decimal numbers, fixed point decimal numbers, decimal integers, octal integers, or binary integers) to an appropriate binary representation.

The rules for representing these various kinds of numbers by forming such character strings conform to the rules for DEC and OCT (see Section 03), but are more inclusive. These rules are described below with the appropriate macro.

Hollerith cards or BCD tape records might also contain information which requires no conversion when read. Such information may consist of any of the set of 50 allowable Hollerith characters, including the character "blank." Two macros (IBCC and IBCW) are provided to treat such information.

The INTRAN language is designed around the principle of processing a single unit record at a time.

There are two stages which, in general, the programmer must specify for the processing of every unit record:

1. Read-in Stage:

   The unit record is read, from a card reader or a tape unit, into a region of core storage in its external form. No conversion will have been done at the end of this stage, except for a standard Hollerith-to-BCD conversion if the record is a Hollerith card, i. e., the conversion which would be performed automatically by the card-to-tape peripheral equipment when used to produce a 12-word BCD tape record from a 72-column Hollerith card.

2. Internal Processing Stage:

   The information, after passing through the read-in stage, is processed according to the format specified by the programmer. If the information is from a column binary card or a binary tape, it is simply moved to some other region in core storage. If the information is from a Hollerith card or a BCD tape, it will generally be broken up into sub-strings which are converted to binary and moved to specified core storage locations.

The programmer is provided with five macros which apply to the read-in stage. With the exception of the two macros IIMAGE and INTRAN, which apply to both stages, all remaining macros are directly concerned with the internal processing stage.

The remainder of this chapter is devoted to detailed descriptions of the INTRAN macros. Note that, with a few exceptions, the subfields of the variable field can be any arithmetic expression, just as in an ordinary symbolic instruction. The exceptions will be specifically stated in the descriptions.

IIMAGE Y,T,C

The I-region of core (so-called because it contains the "image" of the external record) is essentially a "buffer" area which links the read-in stage with the internal processing stage. A fixed part of the storage occupied by INTRAN is automatically set aside for the I-region, and normally the programmer need not make any reference to it. The I-region consists of two adjacently located regions, I_1 and I_2, each of length 28 words. The reason for the dual regions will be seen under the description of the ISCRIB macro.

28 words is sufficient to accommodate a unit record of any of the four possible forms described above, with the exception of a BCD tape record consisting of more than 28 words. In this case, the programmer must specify a non-normal I-region of his own. To do this, he may simply write an instruction of the form IIMAGE Y,T,C. The effective address Y-c(T) is the location of the first word of the I-region, and C is one-half the length of the total I-region (i. e., the length of either one of the two buffers I_1 and I_2 making up the I-region). For example, the instruction

Operation	Variable Field
IIMAGE	ALPHA, 0, 100

specifies that the I-region is to begin at cell ALPHA and end at cell ALPHA+199, with I_1 comprising cells ALPHA through ALPHA+99 and I_2 comprising cells ALPHA+100 through ALPHA+199.

All macros* which require an I-region and are executed after the above instruction will automatically use the 100-plus-100 word region specified, rather than the normal I-region, until another IIMAGE instruction, or the INTRAN instruction (see below), is executed.

* In particular, the read-in macros ISCRIB and IREADY, and the internal processing macros IBCC, IBCW, IOCTAL, etc.

If a non-normal I-region has been specified, and the programmer wishes to switch back to the normal buffer,* he may specify this by writing the instruction:

Operation	Variable Field
IIMAGE	0

The treatment of large BCD tape records is not the only application for IIMAGE. For example, if the programmer wishes to apply some of the internal processing macros to a region in his own program, he may write an IIMAGE instruction specifying this region. Any subsequent instructions using an internal processing macro will operate on the region defined, until the I-region is redefined (by IIMAGE or INTRAN).

MODAL I-MACROS

Ten of the I-macros** to be described below are active macros in that they are used to specify some dynamic operation, e.g., the operation of reading a unit record into the I-region, or of converting a number from decimal to binary. Eleven other of the I-macros*** have the passive function of specifying the mode of operation of the active macros, and hence are called "modal" macros. For example, the macro IIMAGE is modal since it is used to specify the destination region (in the case of ISCRIB and IREADY) or source region (in the case of the active internal processing macros, like IINT, etc.) and hence controls the operation of all the active macros.

For a given modal macro, there is a certain set of active macros controlled by it. This set may consist of a single macro (for instance, the modal macro IPOINT controls only IFIX), or of more than one macro (e.g., IIMAGE controls all ten of the active I-macros). In the following descriptions of the modal macros, a list of all the macros controlled by each will be included. Conversely, the description of each active macro will list all modal macros which control it.

---

* See also the macro INTRAN.

** IBCC
IBCW
IBIN
IFLOAT
IFIX
IINT
IOCTAL
IREADY
ISCAN
ISCRIB

*** IBRNCH
ICHAR
IEOR
IFILE
IIMAGE
IMASK
IOVPCH
IPOINT
IREDUN
ISCALE
ISPILL

Just as the modal macro IIMAGE has a normal mode (i.e., the use of the standard 28-plus-28 word buffer), every modal macro has a unique normal mode which in each case will be defined in the description of that macro.

The device of setting the mode associated with IIMAGE to normal (mentioned above) by using the instruction IIMAGE 0 can be used for any modal macro. That is, if the operation code of a modal macro is OPCOD, its mode may be returned to normal by execution of the instruction

Operation	Variable Field
OPCOD	0

INTRAN

This macro sets the modes of all 11 modal I-macros to normal.

The variable field of INTRAN should be left blank. Thus, the instruction

Operation	Variable Field
INTRAN	

is equivalent to a complete set of modal macros with a zero variable field.

Every program using the Input macros should include INTRAN as its first Input macro to insure normalization of all the modal I-macros.

THE READ-IN MACROS

The following five macros are concerned with the read-in stage. The first two are active macros and the remaining three are modal macros.

ISCRIB Y, T, C, L

ISCRIB can be used to read a unit record in any of the four permissible forms into the I-region. Y-c(T) is the effective address of a cell which contains the standard 709/7090 "address" of the desired tape unit or card reader (i.e., the 709/7090 unit address, including channel number).* In the case of a tape unit, the address used must indicate BCD mode, not binary mode, regardless of whether the tape record to be read is in BCD or binary form.

* See 709/7090 Reference Manual for standard unit addresses.

For the SHARE Monitor System, Y would normally contain the symbolic unit name, and T would be zero. These symbolic unit names specify locations containing the standard 709/7090 designation and do not require defining by the programmer.

The L-parameter indicates the presence or absence of look-ahead information. (See 709 reference manual, form A22-6536 and 7090 Operator's Guide, form A22-6535.)

If a card reader is to be used, the C-subfield of the variable field is ignored by the ISCRIB subroutine, and may be coded as zero, or be omitted. Thus, if the address of the word in location UNIT is $(01321)_8$, the following instruction will cause a card to be read by the Channel A card reader:

Operation	Variable Field
ISCRIB	UNIT

A card read by an ISCRIB instruction may be punched either in Hollerith code or in column binary code. If the information on the card is Hollerith, it will undergo a standard Hollerith-to-BCD conversion and produce 12 words (72 characters) in the I-region. If the card is column binary, it will produce 24 words in the I-region, in row binary form.

As in the case of a card, a tape record read by ISCRIB may also be either BCD or binary.

All binary tape records will be assumed to have the constant length of 28 words. If an ISCRIB reads such a tape record, the C-subfield is again ignored by the ISCRIB subroutine* and may be coded as zero or omitted.

A BCD tape record, on the other hand, may be of any length, and in this case, the C-subfield is used to specify the number of words of the record to be read into the I-region. If the C-value is coded as zero, or is omitted, in the case of a BCD tape record, C will be taken as 14 (the usual length of a BCD record). Thus, the following two instructions are equivalent:

Operation	Variable Field
ISCRIB	UNIT
ISCRIB	UNIT, 0, 14

If the address of the word in location UNIT is 02203_8, the execution of either of the above instructions would cause a record to be read from tape B3, producing 14 words in the I-region, if the record is BCD, or 28 words, if the record is binary.

* However, as described below, in the event that "look-ahead" information is not available for this record, the ISCRIB subroutine will first attempt to read the record in BCD. In this case, it will use the C-value in the manner described subsequently.

If the record were 100 BCD words, the programmer must give the following instruction in order to read the full 100 words into the I-region. *

Operation	Variable Field
ISCRIB	UNIT, 0, 100

Thus, the number of words, N, which a given ISCRIB instruction prescribes are to be read from a given tape record into the I-region can be characterized as follows:

1. If the record is binary, N = 28 (regardless of the C-subfield).
2. If the record is BCD and
   a. C is 0, N = 14.
   b. C is not zero, N = C.

The number of words read, N, should ordinarily be the same as the length, M, of the tape record. However, no error will be indicated if these two numbers are not equal. If N is less than M, only the first N words of the record will be used; if N exceeds M, then only M words (one record) will be read into the I-region. In no case will an ISCRIB cause more than one tape record to be read.

An error condition will occur if the number of words to be read exceeds one-half the total length of the I-region. For example, the execution of the following instruction will result in an error unless an associated I-region of at least 100-plus-100 words has been provided.

Operation	Variable Field
ISCRIB	UNIT, 0, 100

where the address part of UNIT contains a tape address and the tape record to be read is BCD.

Such an error condition can never occur if all ISCRIB instructions have an $N \leq 28$ and if the normal I-region is always used. However, if an ISCRIB instruction causes this kind of error, the error will be detected by the ISCRIB subroutine before anything else is done. Upon detection of this error, the subroutine will not initiate the prescribed data transmission; instead it will

1. load the number 1 into the MQ

* A sufficiently large I-region (at least 100-plus-100 words) must have been previously specified by an IIMAGE instruction.

2. load the input unit address for the current ISCRIB into the decrement of the AC

3. load the input unit address for the last ISCRIB executed into the address of the AC

4. transfer to the location determined by the current mode of IREDUN.

Note: The "1" placed in the MQ can be used to distinguish the kind of error , since there are other error conditions which will cause a transfer to the same location.

Example:

Assume that the instruction ISCRIB UNIT, 0, 100 is currently being executed with $c(UNIT)_{21-35} = (02203)_8$ and that the last ISCRIB executed was ISCRIB UNIT1 with $c(UNIT1)_{21-35} = (01321)_8$. Suppose also that the normal I-region is currently in use, and that the record to be read is in BCD. Then, because the current ISCRIB instruction specifies reading of 100 words, the error condition will occur. In this case, before transferring control to the location determined by the current mode of IREDUN, the ISCRIB routine will set the AC and MQ as follows:

$$c(MQ) = 1$$
$$c(AC)_{3-17} = (02203)_8$$
$$c(AC)_{21-35} = (01321)_8$$

Note that the mode (BCD or binary) of the tape record is not specified in the ISCRIB instruction. The mode is determined by the ISCRIB subroutine from the information on the tape itself. The procedure is described below.

The tape to be read may contain some records in BCD and others in binary, and records may or may not include look-ahead information to specify the mode of the next physical record on the tape. The presence of look-ahead information is specified in the L-parameter. A non-zero value indicates the information is provided. A zero or blank L indicates that it is absent.

If the look-ahead information has not been provided in the tape records, * an attempt is first made to read the records in the BCD mode. If the attempt is unsuccessful because of a tape redundancy error indication, two additional attempts will be made. If a redundancy error persists after three attempts, up to three attempts are made to read the record in the binary mode (which is presumably the correct

* Even if the look-ahead information has been provided elsewhere, it can, of course, never exist for the first record of any tape, since this record has no predecessor.

mode, since the BCD mode did not work). If a redundancy error persists after these three attempts, the number "0" is loaded into the MQ, the input unit addresses used by the current and last executed ISCRIB instructions are saved as described above and control is transferred to the location determined by the current mode of IREDUN. *

If the look-ahead information exists, an attempt is immediately made to read in the specified mode. If the read is unsuccessful, up to two more attempts are made. If all three attempts are unsuccessful, control is transferred to the location determined by the mode of IREDUN (as before, with c(MQ) = 0, etc.).

When a successful binary mode transmission occurs (whether or not preceded by attempts to read in BCD mode), the tape record image is checked for the characteristic marking of a column binary record (bits 9 and 11 of word 1 both 1). If bits 9 and 11 are not both 1, the ISCRIB subroutine transfers control to the location determined by IREDUN. When the transfer occurs, c(MQ) = 2, and the decrement and address of the AC will contain the current and previous input unit addresses.

If an ISCRIB instruction specifies that a record is to be read from an unassigned unit (the contents of the address of Y-c(T) equal zero), control will be transferred to the location determined by IREDUN. When this transfer occurs the address of the AC will contain the octal location of the ISCRIB instruction referring to the unassigned unit, and the contents of the MQ will be 3.

If an ISCRIB specifies reading from a card reader, the ISCRIB subroutine will first transmit the image of the card in the standard row fashion into a special 24-word card buffer region. During the execution of the next ISCRIB (or IREADY) instruction, this image will be examined for the characteristics of a column binary card. If bits 9 and 11 of the first word are both 1, the 24-word image will be moved to the I-region (after a column-to-row-binary conversion). If the card is found to be a Hollerith card, a standard Hollerith-to-BCD conversion is used to produce 12 BCD words in the I-region.

When an attempted transmission of either a card or tape record results in an end-of-file condition, the ISCRIB subroutine will detect this condition** and transfer control to the location determined by the current mode of IFILE.

---

* As described below, the data transmission specified by a given ISCRIB instruction is merely initiated by the ISCRIB subroutine at the time the instruction is executed, and it is only at the time the next ISCRIB (or IREADY) instruction is executed that transmission is checked for completion and all of the processes are carried out which must necessarily follow the completion of the transmission, e.g., testing tape redundancy errors, backspacing and re-attempting the read in case of redundancy error.

** In the case of a tape record, the special end-of-file record will first undergo the usual redundancy check.

When the transfer occurs, the AC will contain the input unit addresses used by the current and previous ISCRIB instructions in the same way as for an IREDUN transfer. *

In addition to the transfer specified by IREDUN and IFILE, there is a transfer used with ISCRIB which is specified by IBRNCH (see below). When an ISCRIB specifies reading of a binary card or a binary tape record, and reading does not result in an error or end-of-file condition, control is transferred to the location specified by the current mode of IBRNCH. Before the transfer occurs, the current and previous input unit addresses are saved in the same way as for IREDUN or IFILE. *

As described above, the I-region consists of a pair of adjacently located regions, I_1 and I_2. For the sake of simplicity in what has been said so far about ISCRIB, the destination region, into which a given ISCRIB instruction causes a record to be read, has been characterized simply as "the I-region," and no mention has been made of I_1 and I_2.

To enable the programmer to make use of the simultaneous input and computing facilities of the 709/7090, these dual "buffers" work in conjunction with the ISCRIB subroutine as described below.

The destination region for a record read by a given ISCRIB instruction may be either I_1 or I_2, and successive ISCRIB instructions will use I_1 and I_2 alternately. ** For example, consider the following sequence:

Operation	Variable Field
ISCRIB	UNIT
ISCRIB	UNIT
ISCRIB	UNIT

and suppose that $c(UNIT)_{21-35} = (02203)_8$. Then if the first ISCRIB instruction reads the tape record into I_1, the second ISCRIB would read the next record into I_2, the third into I_1, and so on.

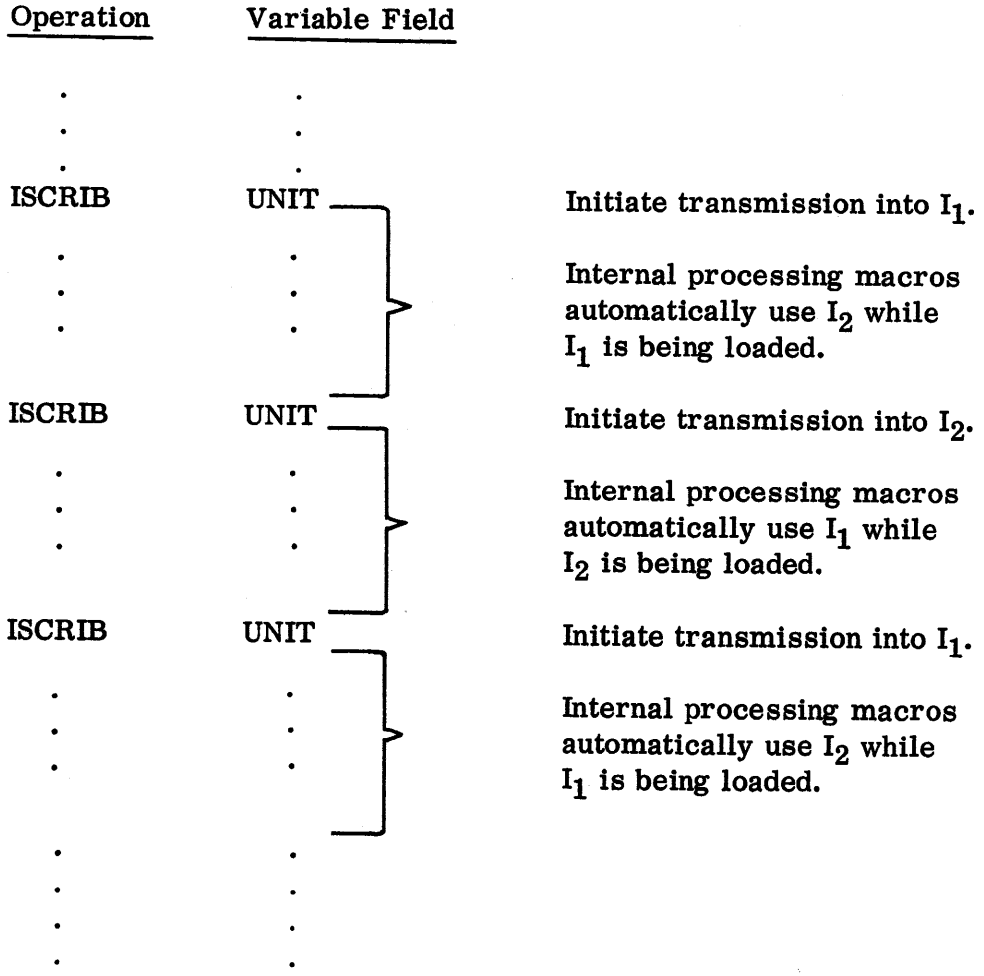
---

* As described below, the data transmission specified by a given ISCRIB instruction is merely initiated by the ISCRIB subroutine at the time the instruction is executed, and it is only at the time the next ISCRIB (or IREADY) instruction is executed that transmission is checked for completion and all of the processes are carried out which must necessarily follow the completion of the transmission, e. g., testing tape redundancy errors, backspacing and re-attempting the read in case of redundancy error.

** Unless IREADY or IIMAGE instructions intervene (see below).

When an ISCRIB instruction, which specifies reading of a certain record, is executed, the actual "reading" (i.e., the physical transmission) of the record into the I_1 or I_2 region is not completely carried out. It is merely initiated by the ISCRIB subroutine. Nothing further is done for this current record until the execution of the next ISCRIB (or IREADY) instruction. At that time, the ISCRIB subroutine checks the transmission, and if the transmission is not completed, waits until it is completed. Meanwhile, the ISCRIB subroutine initiates the transmission required by this next ISCRIB instruction, as soon as possible, into the alternate buffer.

Thus, immediately after the execution of any ISCRIB instruction (i.e., after the ISCRIB subroutine has completed the previous transmission, initiated the present transmission, and transferred control back to the main program) one buffer is being loaded and the other is available for use. In particular, the latter buffer is available for use by all the active internal processing macros. For example, suppose in the illustration above that each ISCRIB instruction is followed by some internal processing macro-instructions, as shown below:



Hence, by this method of automatic alternation in the ISCRIB subroutine, one buffer can be loaded simultaneously with the processing of the other, without specific mention of either buffer by the programmer.

The above example states, in the comment following the first ISCRIB instruction, that I_2 is to be processed, and therefore assumes that I_2 has been loaded by some previously executed ISCRIB instruction which is not shown. At the initial stage of any input processing using the simultaneous feature, it is of course necessary first to load one of the buffers before the simultaneous activities (i.e., reading in record n+1 while processing record n) can begin. This means that two ISCRIB instructions should be executed at the beginning of the program before the execution of any of the internal processing macro-instructions.

In using ISCRIB, the programmer must keep in mind that the alternation of the buffers I_1 and I_2 will always take place whenever the transmission specified by an ISCRIB instruction is initiated. * However, if no ISCRIB (or IREADY) instructions are used in the source program, i.e., if the programmer uses some means other than ISCRIB to supply a source region for the internal processing macros, the internal processing macros will always take I_1 as their source region. This can be seen to be a consequence of the following rules which determine the alternation procedure:

1. Prior to execution of the first ISCRIB (or IREADY) instructions, the source region, S, for any internal processing macro-instructions is set to I_1.

2. If S is I_1 just before the execution of an ISCRIB instruction, then S is I_2 just after the normal execution of the ISCRIB instruction. "Normal execution" here implies that the ISCRIB subroutine either has returned control to the next instruction after the ISCRIB instruction or has transferred control to the location specified by IBRNCH. Transfers to the locations specified by IREDUN or IFILE are specifically excluded. The alternation procedure in these special cases is discussed below.

3. If S is I_1 just before the execution of an IREADY instruction, then S is I_2 just after the execution of the IREADY instruction. (No exceptions.)

4. The execution of an IIMAGE instruction always sets S to I_1, regardless of its previous state.

   Of course, the execution of an IIMAGE instruction usually will change the definition of the I-region, and this rule means that of the two new buffers, I_1 and I_2, the former will be used first.

* There are some circumstances (see page 07.01.15) in which the ISCRIB subroutine will not perform the alternation because the transmission specified by the instruction will not be initiated.

An INTRAN instruction will also set S to I_1 (using the normal I-region).

5. The destination region for a record read by a given ISCRIB instruction is the same as the source region (S) which was in effect just prior to the execution of the ISCRIB instruction.

The ISCRIB subroutine will initiate the transmission required by the present ISCRIB instruction (and hence will alternate I_1 and I_2) whenever possible, even if the previous transmission results in an error or end-of-file condition, provided in this latter case that the input unit used by the current ISCRIB instruction is different from the input unit used by the previous transmission. The only conditions under which the present transmission will not be initiated, and the buffers I_1 and I_2 will not be alternated are:

1. The I-region is too small to accommodate the size of the record prescribed by the current ISCRIB instruction. This error will cause an immediate transfer of control according to the current mode of IREDUN with c(MQ) = 1. The ISCRIB subroutine will not check the previous transmission.

2. The input unit used by the present and previous transmission are the same, and a redundancy error in the previous transmission is detected. This error causes a transfer of control according to the current mode of IREDUN with c(MQ) = 0.

3. The input units used by the present and previous transmissions are the same tape unit, the previously transmitted record was a binary record, but the standard indication for column binary does not appear. This error condition causes a transfer of control according to the current mode of IREDUN with c(MQ) = 2.

4. The input units used by the present and previous transmissions are the same, and an end-of-file condition is detected for the previous transmission. This condition causes a transfer of control according to the modal IFILE.

5. An unassigned unit has been specified by ISCRIB. This causes a transfer of control according to the current mode of IREDUN with c(MQ) = 3.

One further point concerning the logical treatment of I_1 and I_2 by the ISCRIB subroutine should be noted:

As already indicated, the ISCRIB subroutine must wait until the appearance of the next ISCRIB (or IREADY) instruction to check the transmission for the current ISCRIB instruction. When the transmission is checked (by either the ISCRIB or IREADY subroutine), it will be assumed that the definition of the I-region has

not meanwhile been changed. Therefore, if an IIMAGE instruction is executed at any time after the execution of an ISCRIB instruction, then the execution of an IREADY instruction must occur between the ISCRIB and IIMAGE instructions.

Normally, a programmer need not be concerned with the locations of I_1 and I_2. However, if it is found necessary to be aware of which buffer is current (available for use by the internal processing macros), this information can be found as follows:

> The decrement of SYSIT2 contains the address of a word which in turn contains the location of the first word of the current buffer, e.g., if $c(SYSIT2)_{3-17} = (27000)_8$ and $c(27000)_{21-35} = (27777)_8$, then the current buffer starts at $(27777)_8$. When the current buffer is changed by an ISCRIB or IREADY, this address is also changed to give the location of the alternate buffer.

IREADY

The variable field of any IREADY instruction should be left blank.

The programmer will ordinarily use an IREADY instruction at any point in his program where he does not wish to initiate any further transmission (by means of ISCRIB) but where he wishes to make use of the information for which transmission was already initiated by a previous ISCRIB instruction. Thus, an IREADY instruction will usually be used after a series of ISCRIB instructions to check the completion of the transmission caused by the last executed ISCRIB instruction, and make the transmitted information available for use.

For example, suppose that it is required to process the first 100 records on tape unit $(01201)_8$, each record is a normal 14-word BCD record, and no special error or end-of-file routines are to be used. A sample program might be as follows:

```
Location   Operation   Variable Field
START      REWA        1            Rewind tape 1, channel A.
           INTRAN                   Set all modes to normal.
           ISCRIB      START        Initiate reading record n=1.
           AXT         99,1
READ       ISCRIB      START        Initiate reading record
*                                     n=2,3,4,...,100
             .           .          Process record n (using internal
             .           .          processing macros) while reading
             .           .          in record n+1
             .           .          (n = 1,2,3,...,99)
```

```
TIX          READ,1,1     Count records.
IREADY                    Check transmission of record 100
 .            .
 .            .           Process record 100 (using
 .            .              internal processing macros)
 .            .
 .            .
TRA          SYSTEM       Finish.
```

Note that the first ISCRIB instruction is used to initialize the simultaneous process of reading in record n+1 while processing record n. The IREADY instruction checks the transmission of the final record n = 100. This final record is processed without initiating further transmission.

As indicated in the description of ISCRIB, the execution of an IREADY instruction will automatically produce an alternation of the buffers I_1 and I_2. As in the above sample program, this necessary alternation will cause the internal processing macros to use that buffer which was loaded by virtue of the last ISCRIB instruction.

In the above example, the buffer which was loaded by the execution of the last (100th) ISCRIB instruction was I_2. Since the execution of the IREADY instruction caused an intervening alternation of I_1 and I_2, it follows that if another ISCRIB instruction were then executed, the buffer to be loaded would again be I_2.

If the programmer so chooses, he may avoid the alternation and use only the single buffer I_1 by the simple device of following the execution of every ISCRIB instruction by the execution of an IREADY instruction, before the execution of any internal processing macro-instructions preceding the next ISCRIB instruction. In general, of course, the use of this device will defeat the purpose of the alternation logic, i.e., the overlapping of reading and processing. For example, the job done by the sample program above might have been done as follows:

```
Location   Operation   Variable Field
START      REWA        1              Rewind tape 1, channel A.
           INTRAN                     Set all modes to normal.
           AXT         100,1
READ       ISCRIB      START          Initiate reading record
*                                        n = 1, 2, 3, . . . , 100
           IREADY                     and check transmission of
*                                     this record.
  .          .           .
  .          .           .           Process record n (using
  .          .           .              internal processing macros)
           TIX         READ,1,1       Count records.
           TRA         SYSTEM         Finish.
```

In this program, the transmission initiated by each of the 100 executions of the ISCRIB instruction is checked at once by the IREADY instruction. The double alternation of the buffers I_1 and I_2 (one arising from ISCRIB and one from IREADY) results in the continual use of I_1. The logic of this latter program is of course simpler than that of the former one, but the relative loss of time entailed by the serial transmission and processing (rather than parallel, as in the former case) may be considerable.

The IREADY subroutine can best be characterized by noting that it performs the same work as the ISCRIB subroutine (see description of ISCRIB) except the initiation of a new transmission (with which IREADY is not concerned).

In particular, the execution of an IREADY instruction, as with an ISCRIB, may result in:

1. A transfer to the instruction following the IREADY. This is the normal case, i.e., when the previously initiated transmission, having been checked by the IREADY subroutine, does not result in an error or end-of-file condition and where the record transmitted is not binary.

2. A transfer according to the current mode of IBRNCH. This is the case when the previously initiated transmission, having been checked by the IREADY routine, does not result in an error or end-of-file condition, and where the record transmitted is binary. *

3. A transfer to the location specified by the current mode of IFILE. This is the case when the previously initiated transmission results in an end-of-file condition. *

4. A transfer to the location specified by the last IREDUN, with c(MQ) = 0. This is the case when the previously initiated transmission is unsuccessful because of a persistent tape redundancy error. *

5. A transfer to the location specified by the last IREDUN, with c(MQ) = 2. This is the case when the previously initiated transmission is successful and involves a binary tape record, but the test for column binary is not satisfied. *

However, unlike the ISCRIB subroutine, there are <u>no</u> circumstances under which the IREADY subroutine will fail to alternate the buffers I_1 and I_2 (even in the eventuality of a special transfer of control according to IREDUN or IFILE).

* In the description of ISCRIB, it is noted that when the transfer of control occurs, the address part of the AC will contain the address of the input unit for which the condition occurred. In the case of IREADY, this is still true, but the decrement part of the AC (which for ISCRIB contains the address of the "present" input unit) is irrelevant, since IREADY does not initiate a new transmission.

If an IREADY instruction occurs but there is no "previous" transmission, e.g., before an ISCRIB instruction has ever been executed or immediately after the execution of another IREADY instruction, the only action taken will be to alternate the buffers I_1 and I_2. Control will be returned to the instruction following the IREADY instruction.

IBRNCH Y,T

IBRNCH is used to establish a mode which relates to ISCRIB and IREADY, and which has already been discussed in the description of these macros.

When an instruction of the form IBRNCH Y,T is executed, the IBRNCH subroutine computes the effective address Y-c(T). Later, if an ISCRIB instruction initiates the transmission of a binary (card or tape) record, then, after this transmission is checked by a subsequent ISCRIB or IREADY instruction, control will automatically be transferred to Y-c(T). The established address (mode) will continue to be used until the execution of another IBRNCH instruction, or INTRAN instruction, establishes a new mode.

As indicated under ISCRIB and IREADY, the situation when an IBRNCH type of control transfer occurs is as follows:

> The binary record in question has already been transmitted into the I-region and, in case of tape, this transmission has been checked. Moreover, the control transfer occurred in the final stage of the execution of the next ISCRIB (or IREADY) instruction following the ISCRIB instruction which initiated the transmission of the binary record. If this next instruction was an ISCRIB instruction, the transmission prescribed by it has already been initiated; and whether the next instruction was an ISCRIB or IREADY instruction, the usual alternation of buffers I_1 and I_2 has already been performed. In addition, the address and decrement parts of the AC, at the time of the transfer of control, contain the input unit addresses for the previous and present transmission, respectively.

The normal mode associated with IBRNCH results in a message being printed on the debugging output unit. The message specifies which macro in the object program encountered the IBRNCH condition. The contents of the AC and the MQ are also printed out for analysis. The object program is terminated at this point and control transferred to the monitor. *

As an aid to writing the special routine which may be required to provide for an IBRNCH type of control transfer, attention is called to the following:**

* With the IB Monitor, the normal mode results in a recognizable stop.

** This fact applies equally in programming routines for the IREDUN and IFILE control transfers.

The macro-expansion of an ISCRIB instruction is a 4-word calling sequence beginning with the instruction STL SYSIT1 followed by a TXL instruction which transfers control to the ISCRIB subroutine. Thus, when a special transfer of control takes place in the ISCRIB subroutine, the location of this TXL instruction is available for use by the programmer in location SYSIT1, i.e., the location of the next instruction following the ISCRIB instruction expansion would be $c(SYSIT1)_{21-35} + 3$.

IFILE Y,T

This modal macro is used to establish a mode which relates to ISCRIB and IREADY, and which has already been discussed in the description of these macros.

When an instruction of the form IFILE Y,T is executed, the IFILE subroutine computes and establishes the effective address Y-c(T). If an ISCRIB instruction subsequently initiates a transmission of either a card or tape record which results in an end-of-file condition, this condition is detected when the next ISCRIB or IREADY instruction occurs and control will automatically be transferred to this established address. The established address (mode) will be used until the execution of another IFILE instruction (or INTRAN instruction) establishes a new mode.

As indicated under ISCRIB and IREADY, the situation when an IFILE type of control transfer occurs is as follows:

If the end-of-file condition is for a tape unit, the redundancy check was satisfactorily made. Moreover, the control transfer occurs during the execution of the ISCRIB (or IREADY) following the ISCRIB which initiated the transmission leading to an end of file. If this next instruction was an ISCRIB, the transmission prescribed by it has already been initiated (and the buffers I_1 and I_2 alternated) if and only if the input unit used is different from that unit on which the end-of-file condition occurred. If the next instruction was an IREADY, the usual alternation of the buffers I_1 and I_2 has already been performed. In addition, at the time of the transfer of control, the address and decrement of the AC contain the input unit addresses for the previous and present transmission, respectively.

The normal mode associated with IFILE results in a message being printed on the debugging output unit. The message will specify which macro in the object program encountered the IFILE condition. The contents of the AC and the MQ are also printed out for analysis. The object program is terminated at this point and control given to the monitor. *

---

* With the IB Monitor, the normal mode results in a recognizable stop.

The final remarks for IBRNCH, page 07.01.19, should also be noted.

IREDUN Y,T

This modal macro is used to establish a mode which relates to ISCRIB and IREADY, and which has already been discussed in the description of these macros.

When an instruction of the form IREDUN Y,T is executed, the IREDUN subroutine computes and establishes the effective address Y-c(T). Control is automatically transferred to this established address in any one of the following three situations:

1. During the execution of an ISCRIB or IREADY instruction, when the transmission initiated by the previous ISCRIB instruction was for a tape record and a persistent tape redundancy error is detected. In this case, c(MQ) = 0.

2. During the execution of an ISCRIB instruction which prescribes the transmission of a record which is too large to be accommodated by the I-region. In this case, c(MQ) = 1.

3. During the execution of an ISCRIB or IREADY instruction, when the transmission initiated by the previous ISCRIB instruction is successful and involves a binary tape record, but the test for column binary is not satisfied. In this case, c(MQ) = 2.

4. During the execution of an ISCRIB instruction which attempts to read an unassigned unit. In this case c(MQ) = 3.

The established address (mode) will continue to be used until the execution of another IREDUN instruction (or INTRAN instruction) establishes a new mode.

The normal mode associated with IREDUN results in the printing of an error message on the debugging output unit. This message will specify the macro which encountered the error, and will include the contents of the AC and MQ. Control is transferred to the monitor. *

As indicated under ISCRIB and IREADY, the situation when an IREDUN type of control transfer occurs is as follows:

If c(MQ) = 0 or c(MQ) = 2: The control transfer occurred during the execution of the next ISCRIB or IREADY instruction following the ISCRIB which initiated the transmission that led to the error condition. If this next instruction was an ISCRIB, then the new transmission prescribed by it has already been

* With the IB Monitor, the normal mode results in a recognizable system halt.

initiated (and the buffers I_1 and I_2 alternated) if and only if the input unit used is not the one on which the error condition occurred. If this next instruction was an IREADY instruction, the usual alternation of the buffers I_1 and I_2 has already been performed.

If c(MQ) = 1 or c(MQ) = 3: The transfer occurred in the initial stage of the execution of the ISCRIB instruction specifying the illegal (large) record. The present (illegal) transmission has not been initiated (consequently no buffer alternation has been performed) and the previous transmission (if any) has not been examined.

In any case, the address and decrement parts of the AC, at the time of the transfer of control, contain the input unit addresses for the previous and present transmissions, respectively.

THE INTERNAL PROCESSING I-MACROS

As indicated above, the internal processing macros are provided to operate on the record image, i. e., the information in the I-region. (The information is normally placed in the I-region by an ISCRIB instruction.) All of the rest of the I-macros to be described below are of this type.

There are eight active internal processing macros. Six of these, which involve conversion to binary, share certain common properties and will be called "conversion macros." The other two (IBCC and IBCW) involve no conversion.

Associated with the eight active macros, are seven modal macros whose description follows that of the active macros.

In addition, there are three other internal processing macros: Two of these, ICOLR and ICOLIN, are discussed immediately below; the third, IRPT, is discussed at the end of this section.

All eight of the active internal processing macros require a variable field of the form Y, T, C, N. Y and T together specify a core storage address, and C and N together specify a subfield of the I-region. The I-region here is always thought of as divided into six-bit groups called "columns" (or BCD character positions) which are numbered 1, 2, 3, . . . , M, where M (the number of characters in the I-region) is determined by the current mode of the macro IIMAGE. The normal value of M is 6 x 28 = 168.

The C-value defines the beginning column of the subfield and the N-value defines the length in characters (except with IBCW, for which N is the length in words) of the subfield. Thus, for example, C = 49, N = 10 would refer to the subfield of the I-region consisting of columns 49, 50, 51, . . . , 58.

As will be seen below, C = 0 or N = 0 are given a special interpretation, and are equivalent to specifying some positive C-value or N-value, respectively.

The Column Counter

A record image is processed, one subfield at a time, from left to right. Each subfield is processed by the execution of a single active (internal processing) macro-instruction. For example, if an 80-column record image is divided into eight subfields of ten columns each, then eight successive macro-instructions would ordinarily be executed. Such a sequence would appear as follows:

Operation	Variable Field	
OPCOD	Y,T,1,10	Process 1st subfield.
OPCOD	Y,T,11,10	Process 2nd subfield.
OPCOD	Y,T,21,10	Process 3rd subfield.
.	.	.
.	.	.
.	.	.
OPCOD	Y,T,71,10	Process 8th subfield.

where OPCOD stands for one of the active internal processing macros.

In such a case, to avoid the necessity of always specifying the beginning column in the macro-instruction, an automatic feature called the "column counter," with the following properties, is provided:

1. If C = 0 in the macro-instruction, then the contents of the column counter will be used as the C-value for this instruction.

2. If C $\neq$ 0 in the macro-instruction, then the column counter will be set to C.

3. After either 1 or 2 has taken place, the column counter will be increased by the number of characters in the image subfield processed.

Thus, in the above example, the sequence could be simply:

Operation	Variable Field	
OPCOD	Y,T,1,10	Process 1st subfield.
OPCOD	Y,T,0,10	Process 2nd subfield.
OPCOD	Y,T,0,10	Process 3rd subfield.
.	.	.
.	.	.
.	.	.
OPCOD	Y,T,0,10	Process 8th subfield.

ICOLR Y,T

This macro is provided to set the column counter to the value Y-c(T). For example, suppose index register 4 contains 10 when the following instruction is executed.

Operation	Variable Field
ICOLR	71,4

The effect would be to set the column counter to the value 61. Note that the value Y-c(T) is always non-negative since, like an effective address, the twos complement is used if Y-c(T) is negative. In general, Y-c(T) should not be zero, since the columns of the image are numbered beginning with 1.

As explained above, the execution of an active macro-instruction will also serve to set the column counter (e.g., the instruction IINT Y,T,1,10 would set the column counter to 1 + 10 = 11) in addition to carrying out the processing which the instruction specifies.

ICOLIN Y,T

The purpose of this macro is to increase the contents of the column counter by the value Y-c(T). For example, suppose the column counter contains 20 and index register 4 contains 10 when the following instruction is executed.

Operation	Variable Field
ICOLIN	15,4

The effect would be to change the contents of the column counter to 20 + 5 = 25.

This macro is especially useful in cases where the programmer desires to skip over subfields of the record image during internal processing.

IBCC Y,T,C,N

This active macro-instruction causes N characters (N six-bit groups) to be extracted from the record image beginning with column C. This is inserted into the destination region beginning, normally,* with the first (leftmost) character position of the word whose effective address is Y-c(T). Note that the modal macro IMASK may be used to change positioning of the string inserted.

* "Normally," as used here, means that the current mode for IMASK is normal.

The I-region is left undisturbed, as are all positions of the destination region which do not receive transmitted characters.

Example:

Suppose that the record image contains, in columns 21 through 28, the (BCD-coded) string ABCDEFGH, and that c(3000) = c(3001) = $(777\ 777\ 777\ 777)_8$. Then the execution of the instruction

Operation	Variable Field
IBCC	3000, 0, 21, 8

would result in:

$c(3000) = (ABCDEF)_{BCD}$
$c(3001)_{0-11} = (GH)_{BCD}$
$c(3001)_{12-35} = (77\ 777\ 777)_8$
$c(\text{column counter}) = (29)_{10}$

It is not necessary that the record image contain legal BCD characters. The IBCC macro will move any binary information (in groups of six bits) from the I-region to an arbitrary region. This property makes IBCC particularly suitable for use with binary records.

The execution of IBCC Y, T, C, N with N = 0 will lead to a transfer of control to the location determined by the current mode of ISPILL, with c(MQ) = 6.

Besides IMASK and ISPILL, the modal macro IEOR (see below) also applies to IBCC.

IBCW Y, T, C, N

This macro has exactly the same properties as IBCC except for the meaning of N, which is the number of words, rather than the number of characters as in IBCC. Thus, the two instructions

Operation	Variable Field
IBCW	Y, T, C, N
IBCC	Y, T, C, 6*N

are equivalent. (The column counter is, of course, increased by 6*N, rather than by N.)

IBCW is provided only for convenience, and cannot be used if the number of characters to be moved is not a multiple of six.

The modal macros IMASK, ISPILL, and IEOR apply to IBCW.

Rules for the Use of N in the Conversion Macros

For the six conversion macros to be described below, the programmer is not required to specify an N-value (image subfield length). Instead, he may specify N = 0, in which case the subfield used will begin, as usual, at the column specified by the C-value, and extend to the right until a field-terminating character is encountered. The characters which will terminate the field are, in general, different for each of the conversion macros, but will always include the following:

1. blank
2. comma
3. all non-numeric characters other than plus (+), minus (-), decimal point (.), E, and B.

The exact rules for field termination are given for each macro.

A conversion macro-instruction with N = 0 is said to use a "variable length field." After the successful execution of such an instruction (by the conversion subroutine), the terminating character, in BCD, and the column number are loaded into the address and decrement of the AC, respectively. These two items are then available for possible use by the object program on the normal return to the next instruction. Since the terminating character is regarded as belonging to the subfield, the column counter will then contain the column number of the character following the terminating character (i.e., $1 + c(AC)_{3-17}$).

Example:

Suppose the record image contains +0, +1 in columns 21-25, and the following instruction is executed

Operation	Variable Field
OPCOD	Y, T, 21, 0

where OPCOD stands for one of the conversion macros.

The number in columns 21 and 22 (+0) would then be processed, and return made to the next instruction with:

$c(AC)_{21-35}$ = BCD code for ","
$c(AC)_{3-17} = (23)_{10}$
$c(\text{column counter}) = (24)_{10}$

Thus, the N-value specified by a conversion macro-instruction is defined as follows:

If N = 0, then the N-value = the specified N.
If N = 0, then the N-value = $\lceil$(column number of first terminating character) - (C-value) +1$\rceil$.

In no case should the N-value (in a conversion macro-instruction) exceed $(31)_{10}$. The violation of this condition will lead to generally unpredictable results.

If a conversion macro-instruction is defined with N = 0, and the scan goes past the last column of the I-region as established by IEOR (see IEOR — the normal value for the last column is 120), it will terminate with the last column number in the decrement and $(77)_8$ in the address of the AC. The number will be converted. Another conversion macro-instruction following with N = 0 will cause a transfer according to IEOR.

IOCTAL Y,T,C,N

N characters of the I-region, beginning with column c are assumed to contain a BCD-coded octal integer. The execution of the instruction causes this integer to be converted to binary and stored in the cell Y-c(T).

An octal integer must be represented by a string of characters from the set $\lceil 0, 1, 2, 3, 4, 5, 6, 7 \rceil$, and may or may not be preceded by a + or -. The appearance of an irregular character in the subfield will, if N ≠ 0, cause control to be transferred according to the current mode of the modal macro ICHAR (see below). When N = 0, an irregular character is treated as a field-terminating character.

Thus, if columns 21, 22, 23 of the image contain -77, the execution of the instruction

Operation	Variable Field
IOCTAL	3000, 0, 21, 3

would cause:

$c(3000) = (400\ 000\ 000\ 077)_8$
$c(\text{column counter}) = (24)_{10}$

However, if columns 21-23 contain -78, then, because of the illegal 8, the execution of the above instruction would result in a transfer of control according to the mode of ICHAR, without disturbing c(3000). For further information, see page 07.01.42.

On the other hand, if columns 21-23 contain -78, and the instruction

Operation	Variable Field
IOCTAL	3000, 0, 21, 0

is executed, then the result would be:

$c(3000) = (400\ 000\ 000\ 007)_8$
$c(\text{column counter}) = (24)_{10}$
$c(AC)_{21-35} = \text{BCD code for } 8$
$c(AC)_{3-17} = (23)_{10}$

If the representation of the octal integer to be processed by an IOCTAL instruction begins with a - sign, then the numeric part of the integer must not exceed $2^{35}-1$ (i.e., $377\ 777\ 777\ 777_8$). In no case may the numeric part exceed $2^{36}-1$. If either of these conditions is violated, then the attempted execution of an IOCTAL instruction will result in a transfer of control to the location determined by the current mode of ISPILL, with c(MQ) = 0.

The "0" in the MQ can then be used to distinguish the kind of spill error, since there are other conditions which will cause a transfer to the same location.

To illustrate:

1. The strings -400000000000 and 1000000000000 are illegal and result in a transfer of control according to ISPILL.

2. The strings -377777777777, 777777777777, +777777777777, 00777777777777 are all legal and all convert to $(777\ 777\ 777\ 777)_8$.

Besides ICHAR and ISPILL, the modal macros IMASK, IEOR, and IOVPCH also apply to IOCTAL.

IBIN Y, T, C, N

The image subfield defined by C, N in this macro-instruction is assumed to contain a BCD-coded binary integer. The execution of the instruction causes this integer to be converted to binary and stored in location Y-c(T).

A binary integer must be represented by a string of characters from the set [0, 1], and may or may not be preceded by a + or -. Unlike the case of IOCTAL, the use of a - sign with an integer to be processed by an IBIN instruction does not specify that the sign bit of the result is to be a 1. Instead it indicates that the ones complement of the integer is to be taken with respect to a 35-bit binary field; the sign bit of the converted result will always be a "0". The absolute value of the integer in any case cannot exceed 31 bits (or 30 bits, if a "-" sign is used), because of the restriction that the N-value of an IBIN instruction must be less than 32 (see above).

The appearance of an irregular character in the image subfield will, if $N \neq 0$, cause a transfer according to the current mode of ICHAR. If N = 0, the irregular character is treated as a field-terminating character.

Examples:

Assume that columns 21 through 24 of the image contain -101. Then the execution of the instruction

Operation	Variable Field
IBIN	3000, 0, 21, 4

would cause:

$c(3000) = (377\ 777\ 777\ 772)_8$
$c(\text{column counter}) = (25)_{10}$

However, if columns 21 through 24 contain +112, the execution of the above instruction, because of the illegal 2, would result in a transfer of control according to the mode of ICHAR. The contents of location 3000 would be unchanged.

On the other hand, if columns 21 through 24 contain +112, then the execution of the instruction

Operation	Variable Field
IBIN	3000, 0, 21, 0

would cause:

$c(3000) = (000\ 000\ 000\ 003)_8$
$c(\text{column counter}) = (25)_{10}$
$c(AC)_{21-35} =$ BCD code for 2
$c(AC)_{3-17} = (24)_{10}$

In addition to ICHAR, the modal macros IMASK, ISPILL, IEOR, and IOVPCH also apply to IBIN.

IINT Y, T, C, N

The image subfield defined by C, N in this macro-instruction is assumed to contain a BCD-coded decimal integer. The execution of the instruction causes this integer to be converted to binary and stored in location Y-c(T).

A decimal integer must be represented by a string of characters from the set [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] and may or may not be preceded by a + or -. The appearance of an irregular character in the subfield will, if $N \neq 0$, cause a transfer according to the current mode of ICHAR. If N = 0, the irregular character is treated as a field-terminating character.

Example:

If columns 21, 22, 23 of the image contain -31, the execution of the instruction

Operation	Variable Field
IINT	3000, 0, 21, 3

would cause:

$c(3000) = (400\ 000\ 000\ 037)_8$
$c(\text{column counter}) = (24)_{10}$

However, if columns 21-23 contain -9+, the execution of the above instruction, because of the illegal +, would result in a transfer of control according to the mode of ICHAR. The contents of location 3000 are unchanged.

On the other hand, if columns 21-23 contain -9+, the execution of the instruction

Operation	Variable Field
IINT	3000, 0, 21, 0

would cause:

$c(3000) = (400\ 000\ 000\ 011)_8$
$c(\text{column counter}) = (24)_{10}$
$c(AC)_{21-35} =$ BCD code for +.
$c(AC)_{3-17} = (23)_{10}$

If the absolute value of a decimal integer to be processed by an IINT instruction exceeds $2^{35}-1$, an attempt to execute the instruction will result in a transfer of control to a location determined by the current mode of the modal macro ISPILL, with c(MQ) = 1.

In addition to ICHAR and ISPILL, the modal macros IMASK, IEOR, and IOVPCH also apply to IINT.

IFLOAT Y,T,C,N,D*

The image subfield defined by C,N in this macro-instruction is assumed to contain a BCD-coded decimal number, possibly specified with a decimal point and/or an exponent. The execution of the instruction causes this number to be converted to a normalized floating point binary number (i.e., 1-bit sign, 8-bit characteristic, 27-bit fraction) and stored at location Y-c(T).

A decimal number to be processed by an IFLOAT instruction must consist of a string of characters from the set [0,1,2,3,4,5,6,7,8,9,.,+,-,E]. This string must begin with a "principal part" composed of a possible + or - sign followed by a string of numeric characters, possibly containing a decimal point. If the sign is omitted, a + is understood; if the decimal point is omitted, it is understood to be at the right-hand end. The principal part may be followed by an "exponent part" consisting of the character E followed by a + or - sign, and a string of not more than four numeric characters. The + or the E of the exponent part may be omitted, but not both.

For example, the strings 3.14159, +.314159E1, .314159+1, 314159-5, +314159E-5 all are permissible representations of numbers for processing by an IFLOAT instruction and each will convert to the floating point binary representation of the number 3.14159_{10}.

The rules given above conform to the rules for representing a floating point decimal number as required by DEC (see page 03.00.11) except that, for IFLOAT, an integer is also acceptable.

The D-parameter* specifies a nominal decimal scale factor. It is employed to alter the data, for a given IFLOAT macro, by a decimal scale factor equal to D. It will be ignored if there is a decimal point in the data.

* The D-parameter is not recognized when used with the IB Monitor. Use of the IFLOAT subroutine with the IB Monitor assumes a D-value (nominal decimal scale factor) of zero. Therefore, all comments here pertaining to the D-parameter may be ignored.

For example, if the string +7090 starts in column 40 and is operated upon by the instruction

Operation	Variable Field
IFLOAT	3000, 0, 40, 5, 2

it would cause:

c(3000) = the floating point binary representation of $(709000)_{10}$

If the data contains an exponent part, the D-value is additive to this exponent part.

The normal value for the D-parameter is zero, thus not affecting the data value as given.

It is important to note that an additional decimal scale factor may be introduced by the modal macro ISCALE (see below). This value is additive to the decimal scale factors specified by an exponent part of the data and/or the D-parameter (unless there is a decimal point in the data, in which case a D-parameter is ignored). Thus, the effective scale factor is the sum of the exponent part in the data, the D-value (if given), and the current ISCALE value (normal case is zero).

For example, if columns 45 through 52 contain +7090E-3 when operated upon by the instruction

Operation	Variable Field
IFLOAT	3001, 0, 45, 8

would cause:

c(3001) = the floating point binary representation of $(7.090)_{10}$

but for the same string, the instruction with a D-parameter

Operation	Variable Field
IFLOAT	3002, 0, 45, 8, -2

would cause:

c(3002) = the floating point binary representation of (.07090)

The appearance of an irregular character in the image subfield to be processed by an IFLOAT instruction will, if N ≠ 0, cause a transfer according to the current mode of the modal macro ICHAR. If N = 0, the irregular character is treated as a field-terminating character.

Thus, if columns 21 through 27 of the image contain +.25E+1, then the execution of the instruction

Operation	Variable Field
IFLOAT	3000, 0, 21, 7

would cause:

$c(3000) = (202\ 500\ 000\ 000)_8$ (i.e., the floating point binary representation of $(2.5)_{10}$)

$c(\text{column counter}) = (28)_{10}$

However, if columns 21 through 27 contain +.25Z+1, the execution of the above instruction would, because of the illegal Z, result in a transfer of control according to the mode of ICHAR, without disturbing c(3000).

On the other hand, if columns 21 through 27 contain +.25Z+1, the execution of the instruction

Operation	Variable Field
IFLOAT	3000, 0, 21, 0

would cause:

$c(3000) = (177\ 400\ 000\ 000)_8$ (i.e., the floating point binary representation of $(0.25)_{10}$)

$c(\text{column counter}) = (26)_{10}$
$c(AC)_{21-35}$ = BCD code for Z
$c(AC)_{3-17} = (25)_{10}$

There are four restrictions with respect to size on a decimal number to be processed by an IFLOAT instruction:

1. The string of numeric digits in the principal part of the number, considered as a decimal integer (i.e., disregarding the decimal point), must not exceed $2^{35}-1$, e.g., the number -090.000000000 is illegal since the corresponding integer $(90\ 000\ 000\ 000)_{10}$ is too large. Note that the number -090.00000000E1 would be acceptable. If such an illegal number is encountered by the IFLOAT subroutine, control will be transferred to the location determined by the current mode of ISPILL, with c(MQ) = 2.

2. The exponent part of the number (if any) should use no more than four numeric characters. Thus, 1.5E00004 is illegal. If this restriction is violated, the IFLOAT subroutine will transfer control according to the mode of ISPILL, with c(MQ) = 3.

3. The absolute value of the number must not exceed the limit of approximately 10^{38} (the maximum size representable in standard floating point form). If the IFLOAT subroutine encounters a number exceeding this limit, control will be transferred according to the current mode of ISPILL with c(MQ) = 8, indicating floating point overflow.

4. If the IFLOAT subroutine encounters a number smaller than approximately 10^{-38}, a floating point underflow condition will result and control will transfer according to the current mode of ISPILL with c(MQ) = 7.

Note that in 3 and 4 above, the possible effect of the D-parameter (not applicable with IB Monitor) and a decimal scale factor introduced by the modal macro ISCALE (described below) must also be included in the value of the number.

In addition to ICHAR and ISPILL, the modal macros ISCALE, IEOR, and IOVPCH also apply to IFLOAT (see below).

IFIX Y,T,C,N,D,P*

The image subfield defined by C,N in this macro-instruction is assumed to contain the same kind of number as for IFLOAT (i.e., a decimal number, possibly specified with a decimal point and/or a decimal scale factor), except that an additional part may be present, called the "B-part." The execution of the instruction causes the number to be converted to a fixed point binary number (i.e., 1-bit sign, 35-bit numeric part) and stored in location Y-c(T). The position

* The D- and P-parameters are not recognized when used with the IB Monitor. Use of the IFIX subroutine with the IB Monitor assumes a D-value (nominal decimal scale factor) and P-value (nominal binary scale factor) of zero. Therefore all comments here about D and P can be ignored if INTRAN is being used with the IB Monitor.

of the binary point in Y-c(T) is defined by the B-part (as possibly modified by the P-parameter or IPOINT).

The B-part is a data string consisting of the character B followed by a signed integer using at most four numeric characters. If the sign is +, it may be omitted, but the B must always be present. The B-integer is used to count from the left-hand end of the binary word (which consists, of course, of a sign position and positions 1, 2, 3, . . . , 35). Thus, a B-integer of 0 indicates that the binary point is to be regarded as immediately to the left of position 1, while a B-integer of 35 indicates that the binary point is to be regarded as immediately to the right of position 35.

Ordinarily, the binary point will be specified to lie inside the 35-bit word cell, corresponding to a B-integer of 0, 1, 2, . . . , 35. However, the B-integer can be negative, in which case it implies counting to the left (instead of to the right) so that, for example, . 25B-1 would convert to $(200\ 000\ 000\ 000)_8$. The actual number is, of course, $(0.01)_2$ and the binary point is one position outside the left-hand end of the word.

The B-integer may also exceed 35; for example, 2. 0B+36 would convert to $(000\ 000\ 000\ 001)_8$. Here, the binary point is one position outside the right-hand end of the word cell.

In addition to the B-part discussed above, there are two other methods of specifying the location of the binary point. They are:

1. The modal macro IPOINT (see below).
2. The value of the P-parameter in the IFIX instruction.

The following table shows the relationships between the three methods of binary point specification. *

	IN EFFECT	RESULT
1.	Only one of the three modes	That mode will be effective
2.	Combination of IPOINT and B in the data, or of IPOINT and the P-parameter	Result is the sum of the two values
3.	B in data and P-parameter	The P-parameter is ignored
4.	All three modes are specified	Result is the sum of the B-part (in the data) and the IPOINT value. The P-parameter is ignored.

* For use with the IB Monitor, the P-parameter is not available. The two methods of binary point location are IPOINT and a B-value in the data. If the two are used together, the IPOINT value is ignored.

Example:

If columns 21 through 25 contain 14B10 and are being processed by

Operation	Variable Field
IFIX	3000, 0, 21, 5, 0, 25

the position of the binary point will be following bit 10. However, if the IPOINT modal macro established a current non-zero value for the binary point location, the effective location would be the sum of this IPOINT value and the 10 from the B-part. In both cases, the P-value of 25 from the IFIX expansion would be ignored because of the presence of the B10 in the data.

Further, if the instruction were given:

Operation	Variable Field
IFIX	3000, 0, 21, 2, 0, 25

(thus converting only the 14, and not examining the B10), the position of the binary point will be after bit 25. If the IPOINT modal macro established a current non-zero value, the effective binary point location would be the sum of the IPOINT value and 25 from the P-parameter.

The D-parameter represents a nominal decimal scale factor and behaves exactly as the D-parameter described for IFLOAT. (Note again that D is ignored if the data has a decimal point.)

The normal value of both D and P is zero, which is the equivalent of no scale factor at all.

If the number to be converted by an IFIX instruction uses both a B-part and E-part ("exponent part"), both parts should be placed after the principal part of the number, but their relative order is unimportant. For example, 314159B2E-5 is equally as acceptable as 314159E-5B2.

The appearance of an irregular character in the image subfield to be processed by an IFIX instruction will, if $N \neq 0$, cause control to be transferred according to the current mode of ICHAR. If $N = 0$, the irregular character is treated as a field-terminating character.

Example:

Suppose that columns 21 through 27 of the image contain 25B5E-1. Then the execution of the instruction

Operation	Variable Field
IFIX	3000, 0, 21, 7

would cause:

c(3000) = $(024\ 000\ 000\ 000)_8$ (i. e., the fixed point binary representation of $(2.5)_{10}$ regarding the binary point between bits 5 and 6)

c(column counter) = $(28)_{10}$

However, if columns 21 through 27 contain 25B5Z-1, the execution of the above instruction would, because of the illegal Z, result in a transfer of control according to the mode of ICHAR, without disturbing c(3000).

On the other hand, if columns 21 through 27 contain 25B5Z-1, then the execution of the instruction

Operation	Variable Field
IFIX	3000, 0, 21, 0

would cause:

c(3000) = $(310\ 000\ 000\ 000)_8$ (i. e., the fixed point binary representation of $(25.0)_{10}$ regarding the binary point between bits 5 and 6)

c(column counter) = $(26)_{10}$
$c(AC)_{21-35}$ = BCD code for Z
$c(AC)_{3-17} = (25)_{10}$

There are four restrictions with respect to size on a decimal number to be processed by an IFIX instruction:

1. The string of numeric digits in the principal part of the number, considered as a decimal integer (i. e., disregarding the decimal point) must not exceed $2^{35}-1$, e. g., the number -090. 000000000 is illegal since the corresponding integer $(90\ 000\ 000\ 000)_{10}$ is too large. Note that the number -090. 00000000E1 would be acceptable. If such an illegal number is encountered by the IFIX subroutine, control will be transferred to the location determined by the current mode of the modal macro, ISPILL, with c(MQ) = 2.

2. The exponent and B-parts of the number (if any) should use no more than four numeric characters each. Thus, 1.5E00004 and 1.5B00001 are illegal. If this restriction is violated, the IFIX subroutine will transfer control according to the mode of ISPILL, with c(MQ) = 3.

3. The specified B-integer must lie in the range: $-128 \leq$ B-integer ≤ 127. The violation of this restriction will lead to generally unpredictable results.

4. The position of the binary point must not be such as to cause the loss of high-order (leftmost) 1-bits in the converted result of the number. (Note that the possible effect of a decimal scale factor introduced by the modal macro ISCALE and the D-parameter must also be considered.) For example, 1.5B0 results in the loss of the integral 1-bit of the converted result $(1.1)_2$. However, 1.5B1 is acceptable. If this restriction is violated, the IFIX subroutine will transfer control according to ISPILL, with c(MQ) = 4.

In addition to ICHAR, ISPILL, and IPOINT, the modal macros ISCALE, IEOR, and IOVPCH also apply to IFIX.

ISCAN Y,T,C,N

The image subfield defined by C,N is assumed to contain any number which is composed according to the rules for IINT, IFLOAT, or IFIX. Thus, the rules of composition are exactly the same as for decimal numbers specified in a DEC instruction (see page 03.00.11). Moreover, the manner of treatment (integer, fixed point, or floating point) is determined by the same rules as for DEC so that:

1. If the number has no exponent part, no B-part, and no decimal point, it is treated as though the ISCAN instruction were IINT Y,T,C,N except that IMASK does not apply.

2. If the number has a B-part, it is treated as though the ISCAN instruction were IFIX Y,T,C,N.

3. If the number has no B-part, but has a decimal point or exponent part, it is treated as though the ISCAN instruction were IFLOAT Y,T,C,N.

Just as for IINT, IFLOAT, and IFIX, the appearance of an irregular character in the image subfield to be processed by an ISCAN instruction will, if $N \neq 0$, cause control to behave according to the current mode of ICHAR. If N = 0, the irregular character is treated as a field-terminating character.

Here, the term "irregular character" means a character which does not conform to the rules of representation of any of the three possible forms. For example, if columns 21-28 contain "+5E-1B0,", then the execution of the instruction

Operation	Variable Field
ISCAN	3000, 0, 21, 0

would cause the comma in column 28 to be taken as the terminating character, and the number to be treated as fixed point, producing $(200\ 000\ 000\ 000)_8$ in location 3000. With the IB Monitor, the modal macro IPOINT is not applicable to ISCAN, since the explicit appearance of the character B is necessary to ISCAN to indicate a fixed point number.

The restrictions on the size of a decimal number to be processed using IINT, IFLOAT, and IFIX also hold as restrictions for numbers to be processed by ISCAN. The violation of any one of these restrictions will, in each case, lead to action indicated for the appropriate macro.

The modal macros applying to IINT, IFLOAT, and IFIX apply equally to ISCAN except for the modal IMASK which applies to IINT only.

IMASK Y, T, C, N

This modal macro is used to establish a mode of execution for the five macros:

IOCTAL
IBIN
IINT
IBCC
IBCW

If the mode associated with IMASK is normal, the above five macros will function as already described. (As mentioned previously, either of the instructions IMASK 0 or INTRAN will set this mode to normal.)

The execution of the instruction IMASK Y, T, C, N defines a binary subfield, where Y-c(T) is the length in bits of the subfield, and C specifies the number of the first bit of the subfield. (N, which is normally specified as zero, will be described below.) The subfield here is taken with respect to a binary word whose bits are numbered 1, 2, 3, . . . , 36 (not 0, 1, 2, . . . , 35). Thus, if index register 4 contains 1 when the instruction IMASK 16, 4, 22 is executed, then the subfield is the address part of the word (i.e., bits 22 through 36).

After the instruction IMASK Y, T, C is executed, any subsequent instructions of one of the forms

Operation	Variable Field
IOCTAL	ALPHA, TAG, COL, LENGTH
IBIN	ALPHA, TAG, COL, LENGTH
IINT	ALPHA, TAG, COL, LENGTH

will have the same effect as usual, except that, in each case, the 36-bit converted result will not be stored in the cell ALPHA-c(TAG). Instead, the converted result, or the twos complement of its rightmost 35 bits in the case the sign bit is 1 (negative), will be stored in the subfield defined by the controlling IMASK. If the result (or its twos complement) is too large to fit in the subfield, only the rightmost Y-c(T) bits will be used. No other positions in core storage except the specified subfield positions will be disturbed.

For example, if columns 21, 22 of the image contain -2, then the execution of

Operation	Variable Field
IMASK	15, 0, 4
IINT	3000, 0, 21, 2

would set the decrement of c(3000) = $(77776)_8$, without disturbing the other bit positions in cell 3000.

If IMASK is to be applied to an IOCTAL, IBIN, or IINT instruction, the subfield defined by the IMASK should lie entirely within a single cell. For example, the instruction IMASK 15, 0, 31 would not be allowable. If this restriction is violated, an attempt to execute the IOCTAL, IBIN, or IINT will lead to a transfer of control to the location determined by the current mode of ISPILL, with c(MQ) = 5.

However, the instruction

Operation	Variable Field
IMASK	15, 0, 58

would be legal. Suppose it were followed by the instruction

Operation	Variable Field
IOCTAL	3000, 0, 21, 7

and columns 21 through 27 of the image contained +654321. The result would be

$$c(3001)_{address} = (54321)_8$$

Note, as implied in the above example, that the C of IMASK may exceed 36. Thus, the specified subfield, although defined in this case with respect to location 3000, is not in location 3000, but in location 3001.

After the instruction IMASK Y, T, C is executed, any subsequent instructions of one of the forms

Operation	Variable Field
IBCC	ALPHA, TAG, COL, LENGTH
IBCW	ALPHA, TAG, COL, LENGTH

will have the same effect as usual. However, the destination region for the string of characters to be moved by the IBCC or IBCW instruction does not begin with the first character position in ALPHA-c(TAG). Instead, the region begins with bit position C (considering C = 1 as the leftmost bit position of ALPHA-c(TAG)). Thus, if columns 21 through 27 of the image contain the string ABCDEFG, the execution of the instructions

Operation	Variable Field
IMASK	0, 0, 31
IBCC	3000, 0, 21, 7

would cause:

$$c(3000)_{30-35} = (A)_{BCD}$$
$$c(3001) = (BCDEFG)_{BCD}$$

with $c(3000)_{0-29}$ undisturbed.

In the above example, C = 31 happens to correspond to the leading bit of a character position (of which there are six: 1, 7, 13, 19, 25, 31). This is not required. C is quite arbitrary and a character (six-bit group) may overlap two words.

Note that Y and T in the instruction IMASK Y, T, C are irrelevant when applied to IBCC or IBCW.

In the instruction IMASK Y, T, C, N; N must have one of the two values 0 or 1. The specification of N = 1 indicates a "floating mask," so that each time the IMASK instruction is applied (by the execution of an instruction using one of the

five macros controlled by IMASK), the C-value is increased by the length, Y-c(T), of the subfield.

Example:

Suppose that columns 21 through 28 of the image contain +1+2+3+4. Then the execution of the instructions

Operation	Variable Field
IMASK	12, 0, 1, 1
IINT	3000, 0, 21, 2
IINT	3000, 0, 0, 2
IINT	3000, 0, 0, 2
IINT	3000, 0, 0, 2

would result in:

$c(3000) = (0001\ 0002\ 0003)_8$
$c(3001)_{0-11} = (0004)_8$

$c(3001)_{12-35}$ would be undisturbed.

Note that the same effect could be achieved (at more cost in space) by the following sequence of instructions:

Operation	Variable Field
IMASK	12, 0, 1
IINT	3000, 0, 21, 2
IMASK	12, 0, 13
IINT	3000, 0, 0, 2
IMASK	12, 0, 25
IINT	3000, 0, 0, 2
IMASK	12, 0, 37
IINT	3000, 0, 0, 2

A floating mask can also be applied to IBCC and IBCW. For example, suppose columns 21 through 23 of the image contain the string ABC. Then the execution of the instructions

Operation	Variable Field
IMASK	36, 0, 31, 1
IBCC	3000, 0, 21, 1
IBCC	3000, 0, 0, 1
IBCC	3000, 0, 0, 1

would result in storing the BCD codes for A, B, and C in the rightmost six bit positions of cells 3000, 3001, and 3002, respectively, without disturbing any other bit positions in these cells.

ICHAR Y, T, C

This modal macro, which applies to the six conversion macros, actually embodies four different independent modal macros:

ICHAR	Y, T, 1
ICHAR	Y, T, 2
ICHAR	Y, T, 4
ICHAR	Y, T, 8

The C-value in ICHAR Y, T, C can be any non-zero sum of the set $[1, 2, 4, 8]$.

Thus, $1 \leq C \leq 15$, and provides a convenient abbreviation for the corresponding combination of ICHAR instructions, e.g., executing the single instruction

Operation	Variable Field
ICHAR	Y, T, 11

is equivalent to executing the three instructions

Operation	Variable Field
ICHAR	Y, T, 1
ICHAR	Y, T, 2
ICHAR	Y, T, 8

Consequently, it suffices to discuss ICHAR Y, T, C for C = 1, 2, 4, and 8.

1. ICHAR Y, T, 1

When an instruction of the form ICHAR Y, T, 1 is executed, the ICHAR subroutine computes and establishes the effective address Y-c(T). Later, if a conversion macro-instruction with a non-zero N-field specifies conversion of an image subfield which begins with one or more blank characters and contains at least one non-blank character, control will be transferred automatically to location Y-c(T). The core location specified by the conversion macro-instruction will remain undisturbed, although the column counter will be increased as usual.

Example:

Suppose the instruction

Operation	Variable Field
ICHAR	5000, 0, 1

has been executed, and columns 21 through 24 of the image contain the string bb+1, then the execution of the instruction

Operation	Variable Field
IINT	3000, 0, 21, 4

will result in a transfer of control to location 5000, with c(column counter) = 25. Location 3000 is unaffected.

The normal mode for ICHAR Y, T, 1 is to ignore the leading blanks and continue processing as usual. Thus, in the above example, the IINT instruction would set c(3000) = $(000\ 000\ 000\ 001)_8$.

2. ICHAR Y, T, 2

The remarks for C = 1 hold here also, except that the condition which will cause the control transfer is a completely blank image subfield.

The normal mode here is to treat the completely blank subfield as though it contained all zeros, thus yielding the value 0 for the converted result in all cases.

3. ICHAR Y, T, 4

The remarks for C = 1 hold here also, except that the condition which will cause the control transfer is a non-blank subfield which ends with one or more blanks, e.g., +1bb.

The normal mode here is to ignore the trailing blanks and continue processing as usual.

4. ICHAR Y, T, 8

The remarks for C = 1 hold here also, except that the condition which will cause the control transfer is a subfield which contains an irregular character, i.e., a character which does not conform to the rules of representation for the conversion macro in question.

Examples:

If the string 1. 2 is referred to by IINT, then the ". " is irregular.
If the string 1B1 is referred to by IFLOAT, then the "B" is irregular.
If the string 1B1Eb-1 is referred to by IFIX, then the blank (b) is irregular.

In addition, before the transfer, the irregular character (in BCD code) and its column number are stored, by the conversion macro subroutine, in the address and decrement of the AC, respectively. Thus, these two items are available for possible use by an error routine which the programmer may write.

Example:

Suppose the instruction

Operation	Variable Field
ICHAR	5000, 0, 8

has been executed, and columns 21 through 24 contain +787. Then the execution of the instruction

Operation	Variable Field
IOCTAL	3000, 0, 21, 4

would cause a transfer to the location 5000, with

$c(AC)_{21-35}$ = BCD code for 8
$c(AC)_{3-17} = (23)_{10}$
c(column counter) = $(25)_{10}$
c(3000) unchanged

The normal mode* of ICHAR is a transfer to the system and results in the printing of an error message on the debugging output unit. The location of the conversion macro which encountered the illegal character is included in the message. The contents of the AC and the MQ are printed out for analysis. The remainder of the job is deleted, and control is then transferred to the monitor.

An error return has been provided in the event the programmer desires to return to his program when an irregular character is encountered; i. e., an ICHAR 8 condition (see below).

* With the IB Monitor, the normal mode results in a recognizable stop.

The four modes associated with the four different ICHAR conditions can be set to normal simultaneously by using the instruction ICHAR 0 (or by INTRAN). Combinations of the four can be set to normal by using ICHAR 0, 0, C with a proper choice of C. Thus, the execution of the instruction

Operation	Variable Field
ICHAR	0, 0, 7

will cause leading and trailing blanks in a subfield to be ignored, and a completely blank subfield to be treated as zero, until a subsequent ICHAR instruction changes one or more of these three modes.

ISPILL Y, T

This modal macro is used to establish a mode which relates to the six conversion macros, and which has already been mentioned in the description of each of these macros.

When an instruction of the form ISPILL Y, T is executed, the ISPILL subroutine computes and establishes the address Y-c(T). Subsequently, control is transferred to this address in any one of the following situations:

1. When the numeric part of an octal integer being processed by an IOCTAL instruction exceeds $2^{36}-1$, or the octal integer has a "-" sign and its numeric part exceeds $2^{35}-1$.

   Control is transferred with c(MQ) = 0.

2. When the absolute value of a decimal integer being processed by an IINT or ISCAN instruction exceeds $2^{35}-1$.

   Control is transferred with c(MQ) = 1.

3. When a floating point or fixed point decimal number being processed by an IFLOAT, IFIX, or ISCAN instruction is represented in such a way that the string of numeric digits in its principal part, considered as an integer, exceeds $2^{35}-1$.

   Control is transferred with c(MQ) = 2.

4. When a floating point or fixed point decimal number being processed by an IFLOAT, IFIX, or ISCAN instruction is represented with an E-field or a B-field using more than four numeric characters.

   Control is transferred with c(MQ) = 3.

5. When a fixed point decimal number being processed by an IFIX or ISCAN instruction is such that the indicated position of the binary point causes the loss of high-order 1-bits in the converted result.

   Control is transferred with c(MQ) = 4.

6. When an IOCTAL, IBIN, or IINT instruction is executed under a controlling IMASK instruction which specifies a binary subfield extending into more than one binary word.

   Control is transferred with c(MQ) = 5.

7. When IBCC Y,T,C,N or IBCW Y,T,C,N is executed with N = 0.

   Control is transferred with c(MQ) = 6.

8. When an IFLOAT instruction results in an underflow condition.

   Control is transferred with c(MQ) = 7.

9. When an IFLOAT instruction results in an overflow condition.

   Here, c(MQ) = 8.

Whenever the execution of a conversion macro; e.g., IINT Y,T,C,N, leads to a transfer of control according to ISPILL, the column counter will be increased, as usual, by the N-value specified in the conversion macro-instruction, but the contents of location Y-c(T) will be unaffected.

The normal mode associated with ISPILL* is a transfer to the system which results in the printing of an error message on the debugging output unit indicating which macro caused the spill. The contents of the AC and MQ are printed for analysis. Control is then transferred to the monitor. The remainder of the job is deleted.

ERROR RETURN: ICHAR 8; ISPILL

If an error condition occurs which transfers control to an ICHAR 8 or ISPILL error routine, the programmer may want to determine where the error occurred in order to return to the program at the proper location. This task is performed automatically. In order to return to the next sequential instruction after the macro which causes the error, it is only necessary to transfer to location SYSIT1.**

* With the IB Monitor, the normal mode is a recognizable stop within the system.

** With the IB Monitor, SYSIT1 must be referred to as an absolute location, and can be determined from the expansion of any input macro-instruction.

Example:

Location	Operation	Variable Field	
	ISPILL	ERROR	
LOC1	IFIX	Y,T,C,N	Convert
LOC2	IFIX	Y,T,C,N	"
LOC3	IFIX	Y,T,C,N	"
	.	.	
	.	.	Continuation of the program
	.	.	
ERROR	PANEL		Error routine — look at AC and
	TRA	SYSIT1	MQ for analysis, then return to object program at next instruction.

If a spill occurred in the IFIX at LOC1, the error routine would give a PANEL (see Section 06) and return control to LOC2.

When returning via this error return, two options are available:

1. A zero placed in the decrement of SYSIT1 causes zeros to be stored in the word where the converted result was to go.

2. A non-zero value in the decrement of SYSIT1 causes the location where the converted result was to go to remain unaltered.

The normal condition is a zero in the decrement of SYSIT1.

If the index registers are altered in the error routine, their values when the transfer to SYSIT1 is executed, will be in effect when control is returned to the next instruction of the object program.

IPOINT Y,T

This modal macro applies only to IFIX and is used to supply a B-value, by means of the literal value Y-c(T), for any fixed point decimal numbers (processed by later IFIX instructions).

Example:

Suppose c(index register 4) = 1 when the instruction

Operation	Variable Field
IPOINT	36,4

is executed, and a later IFIX instruction is to process an image subfield containing 1. 2E3. The result would be the same as if the subfield had contained 1. 2E3B35. All subsequent fixed point decimal numbers would be so treated until a new IPOINT mode is established.

The value established by the IPOINT mode is additive when used with one of the other two methods of locating the binary point (the B-part in the data and the P-parameter of IFIX).* If all three methods specify a non-normal value, the effective binary point location is the sum of the B-part in the data and the IPOINT value, the P-parameter is ignored.

The normal mode for IPOINT is the use of 0 for the B-value. That is, fixed point numbers represented without a B-part and with P = 0 for IFIX are normally treated as proper fractions.

A negative B-integer (e.g., IPOINT -2) is allowable and will be properly interpreted. Even though the -2 would become internally converted, as usual, to its twos complement, the presence of a complemented number is later detected and the number is effectively translated back to its algebraic (signed) value.

ISCALE Y,T

This modal macro applies to all IFLOAT and IFIX instructions, and to all ISCAN instructions which process floating point or fixed point decimal numbers (not integers). ISCALE Y,T can be used for such numbers to supply a decimal scale factor, Y-c(T), in addition to the possible E-part which may also be present and the possible D-parameter of the IFIX or IFLOAT macros.

Example:

Suppose c(index register 4) = 1 when the instruction

Operation	Variable Field
ISCALE	6,4

is executed, and a later IFLOAT, IFIX, or ISCAN instruction is processing an image subfield containing 1. 2E-3. Then the result would be the same as though this number had been multiplied by 10^5, i.e., as though the string were 1. 2E2. This scale factor, 10^5, would continue to be so used until a new ISCALE mode is established.

* With the IB Monitor, the IPOINT value will be effective only if there is no B-part in the data. The P-parameter is not recognized when used with the IB Monitor.

If an IFLOAT or IFIX instruction with a D-parameter* of -1 is processing the same image subfield as above, then the number 1. 2 is multiplied by 10^{-3}, by 10^{5}, and by 10^{-1}, i. e., as if the number were 1. 2E1. The E-value, the ISCALE value and the D-parameter are all additive, unless there is a decimal point in the data. In the latter case, the D-parameter is ignored.

The normal mode for ISCALE is the use of the scale factor $10^{0} = 1$. This is, of course, equivalent to applying no scale factor at all.

The specification of a negative power of 10, e. g., ISCALE -2, is allowable and will be properly interpreted.

IOVPCH Y, T, C

In normal practice, which has been assumed thus far, all signs used in the numbers processed by conversion macros appear as separate characters ahead of the numeric string with which each sign is associated.

IOVPCH, which applies to all conversion macros, provides for the treatment of "overpunched" signs, a space-saving device which combines the sign with one of the characters in the numeric string to which this sign applies. For example, +123 and -123 are represented, respectively, by $12\overset{+}{3}$ and $12\overline{3}$. $\overset{+}{3}$ and $\overline{3}$ indicate a 3-punch, in the input card, "overpunched" with a 12-punch or an 11-punch, respectively. These codes are identical to the card code for the characters C and L (see Appendix 1).

If a sign is overpunched, it must occur with one and only one of the digits in the string to which it relates. The overpunch may occur anywhere in this string, except over a decimal point.

In the general form IOVPCH Y, T, C; C must have one of the values 1, 2, or 4.

1. IOVPCH Y, T, 1

   When an instruction of the form IOVPCH Y, T, 1 is executed, the IOVPCH subroutine computes and establishes the effective value Y-c(T). Subsequent to this, any image subfield processed by any of the conversion macros will be assumed to contain a number which originated from a card where the sign of the principal part is punched over the nth character of the subfield, where n = Y-c(T). Note that the nth character must be a numeric character in the principal part, but that possible leading blanks and decimal point are included in the character count.

* With the IB Monitor the D-parameter is not recognized, and therefore this paragraph does not apply.

Example:

The subfield bb1. 23^{+} would be treated using the mode:

Operation	Variable Field
IOVPCH	6, 0, 1

If a separate sign is punched in addition to the overpunched sign, e.g., b+1. 23^{+}, it will be treated as an irregular character. Moreover, if an IOVPCH mode has been specified, but the designated numeric character does not actually have an overpunch, a + overpunch will be assumed. Of course, overpunching in a card column which does not agree with the IOVPCH mode will result either in an irregular character or in a misinterpretation, e.g., the code for an overpunched +5 is the same as the Hollerith code for E.

2. IOVPCH Y, T, 2

This macro-instruction has the same effect as IOVPCH Y, T, 1; except that it refers only to overpunched signs in the E-field (exponent field) of a decimal floating or fixed point number. Consequently, it applies only to IFIX, IFLOAT, and ISCAN. The character E is not included in the count. Thus, 1. 2E$1\overline{0}^{+}$ would be treated by the mode:

Operation	Variable Field
IOVPCH	2, 0, 2

3. IOVPCH Y, T, 4

This macro-instruction has the same effect as IOVPCH Y, T, 1; except that it refers only to overpunched signs in the B-field of a decimal fixed point number. Consequently, this macro applies only to IFIX and ISCAN. The character B is not included in the count. Thus, the string 1. 2B$1\overline{0}$ would be treated by the mode:

Operation	Variable Field
IOVPCH	2, 0, 4

The normal mode associated with IOVPCH for all three types (C = 1, 2, and 4) is no overpunching. All three can be simultaneously set to normal by using the instruction IOVPCH 0 (or INTRAN). Each can be set independently to normal by using IOVPCH 0, 0, C with C = 1, 2, or 4.

IEOR Y,T,C

When an instruction of the form IEOR Y,T,C is executed, the IEOR subroutine computes and establishes the effective address Y-c(T), and saves the specified C-value. Later, if a conversion macro, or an IBCC or IBCW instruction is executed, and specifies an image subfield whose rightmost column number exceeds the saved C-value, then control will be transferred to location Y-c(T).

For example, if the instruction

Operation	Variable Field
IEOR	5000,0,72

has been executed, then the execution of the instruction

Operation	Variable Field
IBCC	3000,0,71,3

would cause a transfer of control to location 5000, without disturbing c(3000).

The contents of the column counter after such an IEOR transfer of control is in general meaningless.

The normal mode associated with IEOR is a transfer of control to the system, whenever the rightmost column number of the specified image subfield exceeds 120.* This results in an error message being printed on the debugging output unit which indicates the location of the macro which caused the error. Control then is transferred to the monitor and the remainder of the object program is deleted.

IRPT R,I,J

This special control macro can be used to repeat the execution of any active internal processing macro-instruction, i.e., any conversion macro-instruction or IBCC or IBCW macro-instruction.

When an instruction of the form IRPT R,I,J is executed, the first active internal processing macro-instruction thereafter will be executed a total of R times, instead of only once. The first time the macro is executed using the Y,T,C,N fields as specified, the second time with Y increased by I and C increased by J, etc.

* With the IB Monitor, the normal mode results in a recognizable stop within the system.

Note that if the macro-instruction which is repeated uses indirect addressing, it is the direct address, Y, and not the indirect address which is increased by I.

For example, the execution of the two instructions

Operation	Variable Field
IRPT	5, 1, 6
IBCC	3000, 0, 21, 2

is equivalent to the execution of the five instructions

Operation	Variable Field
IBCC	3000, 0, 21, 2
IBCC	3001, 0, 27, 2
IBCC	3002, 0, 33, 2
IBCC	3003, 0, 39, 2
IBCC	3004, 0, 45, 2

In IRPT R, I, J; the value of R must be non-zero. The specification of a negative value for I or J, e.g., IRPT 5, -1, -6, is allowable and will have the effect of decreasing the value of Y and C, respectively, in the subsequent repeated executions of the active macro-instruction.

EXPANSIONS OF INTRAN MACROS*

(1) IBCC [*] Y, T, C, N

```
STL      SYSIT1
TXL      SYSIT2, 0, 1
PZE [*]  Y, T
PZE      C, 0, N
```

(2) IBCW [*] Y, T, C, N

```
STL      SYSIT1
TXL      SYSIT2, 0, 2
PZE [*]  Y, T
PZE      C, 0, N
```

(3) IBIN [*] Y, T, C, N

```
STL      SYSIT1
TXL      SYSIT2, 0, 5
STO [*]  Y, T
PZE      C, 0, N
```

(4) IBRNCH [*] Y, T

```
STL      SYSIT1
TXL      SYSIT2, 0, 16
PZE [*]  Y, T
```

(5) ICHAR [*] Y, T, C

```
STL      SYSIT1
TXL      SYSIT2, 0, 12
PZE [*]  Y, T
PZE      C
```

(6) ICOLIN Y, T

```
STL      SYSIT1
TXL      SYSIT2, 0, 21
PZE      Y, T
```

---

* With the IB Monitor, SYSIT1 is replaced by 43_{10} and SYSIT2 by 44_{10}. With both systems, PZE is, in some instances, replaced by HTR.

(7) ICOLR Y,T

```
STL     SYSIT1
TXL     SYSIT2,0,20
PZE     Y,T
```

(8) IEOR[*] Y,T,C

```
STL     SYSIT1
TXL     SYSIT2,0,11
PZE[*]  Y,T
PZE     C
```

(9) IFILE[*] Y,T

```
STL     SYSIT1
TXL     SYSIT2,0,14
PZE[*]  Y,T
```

(10) IFIX[*] Y,T,C,N,D,B

```
STL     SYSIT1
TXL     SYSIT2,0,7
STO[*]  Y,T
PZE     C,0,N
PZE     D,0,B
```

(11) IFLOAT[*] Y,T,C,N,D

```
STL     SYSIT1
TXL     SYSIT2,0,6
STO[*]  Y,T
PZE     C,0,N
PZE     D
```

The D and B parameters of IFIX and the D parameter of IFLOAT are not available when using the IB Monitor, and the respective expansions are four words rather than five.

(12) IIMAGE[*] Y,T,C

```
STL     SYSIT1
TXL     SYSIT2,0,22
PZE[*]  Y,T
PZE     C
```

(13) IINT [*] Y, T, C, N

```
STL     SYSIT1
TXL     SYSIT2, 0, 4
STO [*] Y, T
PZE     C, 0, N
```

(14) IMASK Y, T, C, N

```
STL     SYSIT1
TXL     SYSIT2, 0, 10
PZE     Y, T
PZE     C, 0, N
```

(15) INTRAN

```
STL     SYSIT1
TXL     SYSIT2, 0, 0
```

(16) IOCTAL [*] Y, T, C, N

```
STL     SYSIT1
TXL     SYSIT2, 0, 3
SLW [*] Y, T
PZE     C, 0, N
```

(17) IOVPCH Y, T, C

```
STL     SYSIT1
TXL     SYSIT2, 0, 17
PZE     Y, T, C
```

(18) IPOINT Y, T

```
STL     SYSIT1
TXL     SYSIT2, 0, 18
PZE     Y, T
```

(19) IREADY

```
STL     SYSIT1
TXL     SYSIT2, 0, 24
```

(20) IREDUN [*] Y, T

```
STL       SYSIT1
TXL       SYSIT2, 0, 15
PZE [*]   Y, T
```

(21) IRPT R, I, J

```
STL       SYSIT1
TXL       SYSIT2, 0, 9
PZE       R-1
PZE       I, 0, J
```

(22) ISCALE Y, T

```
STL       SYSIT1
TXL       SYSIT2, 0, 19
PZE       Y, T
```

(23) ISCAN [*] Y, T, C, N

```
STL       SYSIT1
TXL       SYSIT2, 0, 8
STO [*]   Y, T
PZE       C, 0, N
```

(24) ISCRIB [*] Y, T, C, L

The L parameter is not available when using the IB Monitor.

```
STL       SYSIT1
TXL       SYSIT2, 0, 23
LDQ [*]   Y, T
PZE       C, 0, L
```

(25) ISPILL [*] Y, T

```
STL       SYSIT1
TXL       SYSIT2, 0, 13
PZE [*]   Y, T
```

CHAPTER 2: OUTPUT SYSTEM — OUTRAN*

This chapter deals with the OUTRAN vocabulary of SOS. This vocabulary consists entirely of system macros.

OUTRAN provides the programmer with two tools:

1. A large set of fundamental subroutines, each of which performs one of the basic output functions required for a general class of information.

2. An easy means of specifying that one of these fundamental subroutines is to be used, i.e., by a single macro.

These basic subroutines may in turn be used to construct higher-level subroutines. These higher-level subroutines might be designed to cover part or all of the output processing required in a very large class of problems, and thus be made into a standard output program. The nature and range of output programs will vary widely and may be chosen to suit the particular needs of the installation. By using the Output macros, the programmer will find that the task of constructing such programs is extremely simplified.

On the other hand, for a given production problem with certain output requirements (perhaps peculiar to the problem), the programmer may choose to write his own subprograms as part of his total source program. In such a case, he will find that the direct use of the OUTRAN macros in his program will be of great value.

RULES FOR SPECIFYING OUTRAN MACROS

The general rules for specifying any SOS instruction (see Section 02) apply to all OUTRAN macros. For example, the location field of a macro may contain a symbol, the variable field is divided into subfields separated by commas, etc.

As is the case with all macro-instructions, a location symbol of an OUTRAN macro will be associated with the first word generated by the instruction, i.e., the location symbol will be entered into the dictionary with the value assigned to the first word generated by the macro-instruction.

---

* There are currently two versions of OUTRAN, one which is used with the SHARE Monitor and the other with the IB Monitor. The differences in the two versions are generally relatively minor and are indicated by footnotes. However, significant differences occur in the actual output transmission routine (OSCRIB). The routine is, therefore, described separately for each version.

A list of all the OUTRAN macro-operations and their expansions is given on page 07.02.58. The macro-operations for which indirect addressing is permissible are so indicated. Indirect addressing is, as usual, specified by placing the character "*" at the end of the operation code.

The list of macros also indicates the pattern of the variable field of each macro. As is evident, the number of subfields which must be specified in the variable field is fixed for each macro, but among all the macros, this number varies from zero to at most six. The programmer may, of course, specify zero values for the last n subfields of the variable field by simply omitting the subfields along with their separating commas. The roles played by these various subfields and the rules for specifying subfields depend, in general, on the operation, and are discussed below.

By definition, a macro-instruction always generates, or "expands into," one or more machine words. The number of words in the expansion of any macro depends only on the macro-operation used in the instruction and not on the values of the expressions in the variable field. In fact, every SOS macro (whether a programmer or system macro) results in generation of a fixed number of words.

The expansions for the different Output macros vary in size from at least two to at most five words. For example, the macro OUTRAN will always generate two words, whereas the OFLOAT macro will always generate five words. The number of words generated by a given macro and the number of subfields required in the variable field of such an instruction are directly related. The relation between these two numbers is due to the fact that the set of words generated by a macro-instruction is simply a calling sequence, which of course must contain all of the information specified in the variable field of the macro-instruction.

It can be seen that the first two words generated by any OUTRAN macro are an STL instruction and a TXL instruction. When these two instructions are executed, they store the contents of the instruction location counter and transfer control to the OUTRAN program. The OUTRAN program consists of a set of subroutines which carry out the functions prescribed by the particular macro, using the information which it finds in those words of the macro expansion, if any, immediately following the TXL instruction. Finally, control is returned to the first word following the expansion. Execution of the object program then continues. Note: For certain macro-operations, and under certain conditions (such as error conditions, etc.) which will be discussed below, control may be transferred to some special location.

SPECIAL REGISTERS AND INDICATORS

Whenever the OUTRAN program is entered by a calling sequence generated by a macro, the contents of the AC and MQ, and the status of the AC Overflow indicator

are lost. The contents of the three index registers, the Sense Indicator register, the status of the Sense Lights, or the special indicators on the 709/7090 are not disturbed. However, it is possible that the status of the tape check indicator, the end-of-tape indicator, or the end-of-file indicator associated with a given data synchronizer channel will be affected by an OSCRIB or OREADY instruction which uses that channel for tape operations.

PURPOSE OF THE OUTPUT SYSTEM

The Output System was designed to simplify the problem of providing a suitable external form for output. The external representation will generally be the result of an intermediate conversion (usually from binary to decimal), and will be in the form of either on-line printing, cards punched on-line, or magnetic tapes which may be used to produce printing or punched cards off-line.

OUTRAN is designed to treat a single unit record at a time.

The external information produced by OUTRAN consists of one or more unit records. A unit record may take on any of the following forms:

1. A card punched with 72 columns of Hollerith information (produced on-line).

2. A BCD tape record consisting of n words (i.e., 6n characters). Such a record might be used for off-line punching (BCD mode) or printing, or be intended merely for storage on tape.

3. A BCD tape record specially prepared with spacing characters, to be used for an off-line tape-to-printer operation. The number of characters which are printed in this case can be any multiple of 6 up to 114, or it can be 119 (not 120). (During the actual off-line printing, the Carriage Control switch must be set to Program.)

4. A printed line of Hollerith characters (produced on-line).

5. A card punched with 72 columns of binary information, in column binary form (produced on-line). *

6. A binary tape record consisting of 28 words. Such a record will normally be used to produce an 80-column column binary card by means of an off-line tape-to-card operation. *

---

* The conventional column binary card contains 7- and 9- punches in column 1. The macro OSCRIB, when used for punching a column binary card, will not automatically produce 7- and 9- punches in column 1, and their presence will be at the option of the programmer.

The binary information in a unit record of either of the last two forms can be quite arbitrary. Since this information is merely a copy of the internal binary information there is no conversion problem. The only concern of OUTRAN in producing such a record is to provide a means for writing it on tape, or punching it into cards.

The major part of OUTRAN is devoted to converting binary words, or portions of words, to BCD characters (six-bit groups) representing numbers in any of the following forms: floating point decimal, fixed point decimal, decimal integers, octal integers, and binary integers; and to assembling these character strings to form a record in one of forms 1, 2, 3, or 4. The choice of the external form produced, i.e., position of decimal point, use of a decimal exponent field, etc., is quite general and conforms to the rules of representation for INTRAN.

Internal information in BCD form, thus requiring no conversion except the automatic conversion from BCD to Hollerith, can also be treated to produce a record of form 1, 2, 3, or 4. (See OBCC and OBCW below.)

There are two stages which the programmer must specify for the output production of every unit record:

1. Internal Processing Stage:

   Internal binary information to be produced as an output record is processed to form a string of six-bit groups occupying a region, I, * of core storage.

   "Processed" here may mean either converted to a BCD representation of decimal, octal, or binary data, or else merely moved without conversion. If the unit record is to be of form 1, 2, 3, or 4, then the six bit groups must be legal BCD characters. If the unit record is to be of form 5 or 6, then the six-bit groups are arbitrary.

2. Write-out Stage:

   Information placed in the I-region by the internal processing stage is written in one of the unit record forms (punched cards, printed sheets, or tape records). No conversion is necessary in this stage except the standard BCD-to-Hollerith conversion required when the record is to be a Hollerith card or a printed line.

---

* The I-region (image region) used by the Output System is independent of, and must not be confused with, the I-region used by the Input System. See OIMAGE below.

The programmer is provided with six macros, called "write-out" macros, to specify the write-out stage. With the exception of the two macros OIMAGE and OUTRAN, described immediately below, which apply to both stages, the remaining output macros are directly concerned with the internal processing stage, and are called "internal processing" macros.

OIMAGE Y, T, C

An I-region is used by the Output System as a buffer area to link the internal (output) processing stage with the write-out stage, just as the I-region used by INTRAN is used to link reading and internal processing. This output I-region has no logical relation to the input I-region, and is determined, completely independently, by the mode of the macro OIMAGE in the same way as IIMAGE determines the input I-region.

OIMAGE has the same properties as IIMAGE. The normal output I-region is a fixed region inside that part of core storage occupied by the Output System, and consists of dual regions I_1 and I_2, each 28 words long. This size is adequate to treat any of the permissible unit record forms except a BCD tape record of more than 28 words. OIMAGE Y, T, C specifies a non-normal I-region, 2xC words long, beginning at the effective location Y-c(T). OIMAGE 0 sets the output I-region to normal.

In the remainder of this chapter, such terms as "I-region", "record image", "image subfield", etc., all refer to the output I-region (as distinct from the input I-region). Of course, if the Input and Output Systems are used together, it is possible to identify the input I-region with the output I-region by using an IIMAGE and an OIMAGE instruction with the same variable field.

OUTRAN

This macro is the exact counterpart of the macro INTRAN and serves to set the 12 Output System modes to normal. The OUTRAN macro should ordinarily be used before the execution of any other Output System macro, to insure normalization of all 12 modes.

MACRO CLASSIFICATIONS

The following chart shows the classification of the OUTRAN macros.

	Internal processing macros		Write-out macros	
ACTIVE	OBCC	OFLFIX	OREADY	OSCRIB
	OBCW	OFLOAT		
	OBIN	OFXFLO*		
	OBLANK	OINT		
	OFIX	OOCT		
	*SHARE Monitor only			
MODAL	OEOR	ORPT	OHEAD	OSPACE
	OMASK	OSCALE	OREDUN	OTPEND
	OOVPCH	OSPILL		
	OPOINT	OZERO		
Other	OIMAGE	OUTRAN	OCOLC*	OIMAGE
			OCOLIN	OUTRAN
			OCOLR	
			*SHARE Monitor only	

Note that as with the INTRAN modal macros, the mode associated with any OUTRAN modal macro can be set to normal by using the instruction OPCODE 0; where OPCODE is the modal macro to be reset to normal.

THE INTERNAL PROCESSING MACROS

The internal processing macros include ten active macros (nine when using IB Monitor). One of these, "OBLANK", is special and will be discussed first. Seven involve conversion (from binary) and, hence, are called "conversion macros." The other two (OBCC and OBCW) do not involve conversion. Associated with the active macros there are seven modal macros, whose description follows that of the active macros.

The variable field of an active internal processing macro (except OBLANK) must be of the form Y, T, C, N; Y, T, C, N, K; or Y, T, C, N, K, B.* Here, as in the corresponding INTRAN macros, Y and T together specify a core storage address, and C defines the beginning column of an image subfield, i.e., a subfield of the output I-region, where the I-region is again considered to be composed of column positions (six-bit groups) numbered 1, 2, 3, . . . , M, with M = the character length of the I-region. For each active macro the length of this image subfield depends on N and K (if K is relevant) as described for each macro.

The Column Counter

There is a column counter for OUTRAN (independent of the Input System column counter) with exactly the same properties as the INTRAN column counter.

Thus, for the execution of any active internal processing macro-instruction (including an OBLANK instruction):

1. If C = 0, the contents of the column counter will be used as the C-value for the instruction.

2. If C = 0, the column counter will be set to C.

3. After 1 or 2 is completed the column counter will be increased by the character length specified in the instruction.

OCOLR Y, T

This macro is the exact counterpart of ICOLR and is used to set the column counter to the value specified literally by Y-c(T)

OCOLIN Y, T

This macro is the exact counterpart of ICOLIN and is used to increase the contents of the column counter by the value specified literally by Y-c(T).

* The latter form is not available with the IB Monitor.

OCOLC Y, T*

The execution of this macro will cause the current value of the column counter to be placed in location Y-c(T).

OBLANK Y, T, C

The execution of this active macro-instruction will cause the BCD code for the character "blank" to be placed in every column of that image subfield whose first column is specified by the C-value and whose length is specified literally by Y-c(T). The length should never be zero. If it is, the result is generally unpredictable.

Note that the rules given above under "The Column Counter" will apply to OBLANK.

Example: Suppose index register 4 contains 10 when the following instruction is executed:

Operation	Variable Field
OBLANK	15, 4, 21

The result would be:

c(image columns 21 through 25) = $(\text{bbbbb})_{\text{BCD}}$ (where "b" indicates the character, blank)

c(column counter) = $(26)_{10}$

On the other hand, if c(column counter) = 29 when the instruction

Operation	Variable Field
OBLANK	2

is executed, the result (since C = 0 in the instruction) would be:

c(image columns 29 and 30) = $(\text{bb})_{\text{BCD}}$

c(column counter) = $(31)_{10}$

---

* The OCOLC macro is not available with the IB Monitor.

In constructing a record image, it will often be desirable to begin by clearing the image region so that it will contain all blanks. This can be done most conveniently by using an OBLANK instruction. For example, if the length of the image region is 120 character positions, the programmer can simply write:

Operation	Variable Field
OBLANK	120, 0, 1

This instruction will not set the entire I-region to blanks, only the first 120 columns of either I_1 or I_2 whichever is currently in use (see OIMAGE and OSCRIB which describes the alternation of I_1 and I_2).

Note that the modal macro OEOR (see below) applies to OBLANK.

OBCC Y, T, C, N

The execution of this active macro-instruction causes the string of N characters (six-bit groups) beginning, normally*, with the first (leftmost) character position of the word in location Y-c(T) to be moved to the record image subfield defined by C, N. The core storage region which is the source of the string remains undisturbed, as do all positions of the image region which do not receive transmitted characters.

Example:

Suppose that

c(3000) = (ABCDEF)
BCD
c(3001) = (GHIJKL)
BCD

Then the execution of the instruction

Operation	Variable Field
OBCC	3000, 0, 21, 8

*"Normally," as used here, means that the current mode for OMASK is normal.

would result in:

c(image columns 21 through 28) = $(ABCDEFGH)_{BCD}$

c(column counter) = $(29)_{10}$

Note that the modal macro OMASK can be used to change the position of the string to be inserted.

OBCC is not restricted to the treatment of legal BCD characters. Any binary information, in multiples of six bits, can be moved to the image region.

The execution of OBCC Y, T, C, N when the N-value is zero will lead to a transfer of control to a location determined by the current mode of OSPILL. When the transfer occurs c(MQ) = 0. The "0" in the MQ can be used to distinguish the kind of OSPILL error, since there are other conditions which will cause a transfer to the same location.

Besides OMASK and OSPILL, the modal macro OEOR also applies to OBCC.

OBCW Y, T, C, N

This macro has exactly the same properties as OBCC except for the meaning of N. Here N is the number of words, rather than the number of characters as in OBCC. The column counter is, of course, increased by 6*N, rather than N. Thus, the following two instructions are equivalent.

Operation	Variable Field
OBCW	Y, T, C, N
OBCC	Y, T, C, 6*N

OBCW is provided only for convenience, and cannot be used if the number of characters to be moved is not a multiple of six.

Note that the modal macros OMASK, OSPILL, and OEOR apply to OBCW.

OOCTAL Y, T, C, N

The execution of this macro-instruction causes the contents of cell Y-c(T) to be considered a binary integer and to be converted to an octal integer. The converted number is stored in the image subfield beginning with column C. The length of the subfield is N characters.

If N is written as zero, or is omitted, the value of N will be taken as 12. N should not exceed 13. If N>13, control will be transferred to the location determined by the current mode of the modal macro OSPILL with c(MQ) = 3.

If $1 \leq N \leq 12$ (which is the ordinary case), then the contents of location Y-c(T) are considered as a 36-bit, signless integer, and converted to the corresponding string of 12 octal digits. The rightmost N digits are stored in the specified image subfield.

Thus, if c(3000) = $(000\ 000\ 000\ 077)_8$, then the execution of the instruction

Operation	Variable Field
OOCTAL	3000, 0, 21, 3

would set c(image columns 21 through 23) = (077)$_{10}$ BCD

c(column counter) = (24)$_{10}$

If the specified image subfield length is too small to accommodate the number (i.e., if N < 12 and the (12-N) leftmost digits in the 12-octal-digit converted result are not all zeros), then control will be transferred to the location determined by the current mode of OSPILL with c(MQ) = 9.

For example, if the instruction in the illustration above had been OOCTAL 3000, 0, 21, 1, then such an OSPILL transfer would occur because of the loss of the leftmost 7 digit.

If N = 13, then the contents of location Y-c(T) will be converted as a 35-bit, signed integer to produce a 13-character string consisting of a sign followed by 12 octal digits (the leading digit cannot exceed 3). If the sign is +, it will be suppressed, i.e., stored as a blank character.

Example:

If c(3000) = $(000\ 000\ 000\ 077)_8$, then the execution of

Operation	Variable Field
OOCTAL	3000, 0, 21, 13

would cause c(image columns 21 through 33) = (b000000000077)$_{10}$ BCD

c(column counter) = (34)$_{10}$

Note that leading zeros are never suppressed.

In addition to OSPILL, the modal macros OMASK and OEOR also apply to OOCTAL.

OBIN Y, T, C, N

The execution of this macro-instruction causes the contents of cell Y-c(T), considered as a 36-bit signless integer, to be converted to a binary integer; i.e., a 36-character string of 0-digits and 1-digits. The rightmost N digits of this string are then stored in the N-character image subfield beginning at column C. If N is 0, or is omitted, the value of N will be taken as 36. The value of N should not exceed 36. If N >36, then control will be transferred to the location determined by the current mode of OSPILL with c(MQ)= 4.

Thus, if c(3000) = $(000\ 000\ 000\ 077)_8$, then the execution of the instruction

Operation	Variable Field
OBIN	3000, 0, 21, 7

would set c(image columns 21 through 27) = $(0111111)_{BCD}$

c(column counter) = $(28)_{10}$

If the specified image subfield length is too small to accommodate the number, i.e., if N<36 and the (36-N) leftmost digits in the converted result are not all zeros, control will be transferred to the location determined by the current mode of OSPILL with c(MQ) = 10.

For example, if the instruction in the illustration above had been OBIN 3000, 0, 21, 5, then an OSPILL transfer would occur because of the loss of the leftmost 1 digit.

Note that the results of using OBIN never involve the characters + or -, and that leading zeros are not suppressed.

In addition to OSPILL, the modal macros OMASK and OEOR also apply to OBIN.

OINT Y,T,C,N

The execution of this macro-instruction causes the contents of cell Y-c(T), considered as a 35-bit, signed binary integer, to be converted to a decimal integer, thus producing a sign and a string of 11 decimal digits. The rightmost N digits of this string, with leading zeros replaced by blanks and with an extra blank character attached to the beginning of the N-digit string, is then stored in the N+1th character image subfield beginning at column C. Finally, if the sign of the integer is negative, a - character replaces the rightmost blank character (i.e., the - immediately precedes the most significant digit). If the absolute value of the integer is 0, so that the 11-digit converted result is all zeros, then the above procedure is not used. Special rules, detailed in the description of the modal macro OZERO (see below), apply in this case.

Leading zeros and + signs are suppressed, but the length of the image subfield is always N+1, not N. *

If N is 0, or is omitted, the value of N will be taken as 11. The value of N should not exceed 11. If N>11, then control will be transferred to the location determined by the current mode of OSPILL with c(MQ) = 2.

Thus, if c(3000) = $(400\ 000\ 000\ 017)_8$, then the execution of the instruction

Operation	Variable Field
OINT	3000,0,21,3

would set c(image columns 21 through 24) = $(b-15)_{BCD}$

c(column counter) = $(25)_{10}$

If the specified image subfield length is too small to accommodate the number, i.e., if N<11 and the (11-N) leftmost digits in the 11-digit converted result are not all zeros, then control will be transferred to the location determined by the current mode of OSPILL with c(MQ) = 8.

For example, if the instruction in the illustration above had been OINT 3000,0, 21,1, then an OSPILL transfer would occur because of the loss of the leftmost digit.

---

* Unless a non-normal OOVPCH mode is in effect (in which case the length is N; see OOVPCH below).

In addition to OSPILL, the modal macros OMASK, OEOR, OZERO, and OOVPCH also apply to OINT.

OFLOAT Y,T,C,N,K

The execution of this macro-instruction causes the contents of location Y-c(T) to be regarded as a floating point binary number (i.e., 1-bit sign, 8-bit characteristic, 27-bit fraction), possibly not normalized. This number is first normalized and then converted to floating point decimal form, i.e., to a string composed of a principal part ("mantissa") followed by an exponent part (decimal scale factor). This string is then stored in the image subfield beginning at the column number specified by C.

The number of characters in the string (and hence the length of the image subfield used) depends on N and K, as follows:

> The principal part of the number will consist of a sign (+ will be suppressed) followed by N+K decimal digits, with the leftmost digit non-zero. If $K \neq 0$, a decimal point will be used to separate the N leftmost and the K rightmost digits, so that the principal part is composed of N+K+2 characters. In this case, N = 0 is allowable and will cause the point to occur between the sign and the leftmost digit. If K = 0, a decimal point will not be used, so that the principal part is composed of N+1 characters. When K = 0, an N = 0 is not allowable and will cause a meaningless result.

If the converted result is zero then special rules apply which are given under the description of the modal macro OZERO (see below).

An exponent part will always appear following the principal part. This exponent part will not use the character E*, but always commences with a sign (+ is not suppressed). The sign is followed by exactly two decimal digits, unless the absolute value of the decimal exponent exceeds 99, in which case three decimal digits will be used. Leading zeros are not suppressed.

Ordinarily, the absolute value of the decimal exponent will not exceed 38, but the use of an additional decimal scale factor by means of OSCALE (see below) may cause it to exceed 99.

---

* Unless the current OOVPCH mode is non-normal; see description of OOVPCH below.

Thus, the number of characters in the generated string is normally*:

N + 4	if	K = 0	(sign and exponent)
N + K + 5	if	K ≠ 0	(sign, decimal point, and exponent)

When the exponent requires three decimal digits this number of generated characters is increased by 1.

As an illustration, suppose that c(3000) = $(601\ 500\ 000\ 000)_8$, i.e. the standard floating point binary representation of $(-1.25)_{10}$. Then the execution of the instruction

Operation	Variable Field
OFLOAT	3000, 0, 21, 3, 0

would set c(image columns 21 through 27) = $(-125\text{-}02)_{BCD}$

c(column counter) = $(28)_{10}$.

On the other hand, the instruction

Operation	Variable Field
OFLOAT	3000, 0, 21, 0, 3

would result in c(image columns 21 through 28) = $(-.125{+}01)_{BCD}$

c(column counter) = $(29)_{10}$

Whereas the instruction

Operation	Variable Field
OFLOAT	3000, 0, 21, 1, 3

would result in c(image columns 21 through 29) = $(-1.250{+}00)_{BCD}$

c(column counter) = $(30)_{10}$.

---

* Unless the current OOVPCH mode is non-normal; see description of OOVPCH below.

There are three restrictions with respect to OFLOAT whose violation will cause a transfer of control to the location determined by the current mode of OSPILL. These are:

1. N+K must not exceed 8. Violation of this restriction leads to a control transfer with c(MQ) = 5.

2. The absolute value of the decimal exponent of the converted result must not exceed 999. Violation of this restriction leads to a control transfer with c(MQ) = 15.

3. The floating point binary number in location Y-c(T), if not normalized, should not be so small as to cause a floating point underflow when it is normalized. Violation of this restriction leads to a control transfer with c(MQ) = 13.

In addition to OSPILL and OSCALE, the modal macros OEOR, OZERO, and OOVPCH, also apply to OFLOAT.

OFLFIX Y,T,C,N,K

The execution of this macro-instruction has the following effect:

The contents of location Y-c(T) is regarded as a floating point binary number possibly not normalized. * This number is first normalized and then converted to fixed point decimal form which consists of a signed string of decimal digits, possibly with a decimal point. This string is then stored in the image subfield beginning at the column number specified by C.

If K = 0, the string which is developed will consist of N+K+2 characters; one character for the sign, N digits representing the integral part of the number (N may be 0), followed by a decimal point and K digits representing the fractional part of the number.

If the current OOVPCH mode (see below) is non-normal, then the number of characters generated is 1 less than indicated above.

If K = 0, no decimal point will be used, and the string will consist of N+1 characters; one character for the sign and N digits representing the integral part of the number. In this case, the specification of N = 0 is not allowable and will cause a meaningless result.

---

* If bits 9-35 of the contents of location Y-c(T) are all zeros, special rules apply. These are given under the description of the modal macro OZERO (see below).

Leading zeros in the integral part of the number and plus signs are suppressed. Minus signs will appear in the position immediately to the left of the high-order non-zero digit in the integral part, or immediately to the left of the decimal point.

For example, suppose that c(3000) = $(601\ 500\ 000\ 000)_8$, i.e., is the standard floating point binary representation of $(-1.25)_{10}$. Then the execution of the instruction

Operation	Variable Field
OFLFIX	3000, 0, 21, 3, 0

would set c(image columns 21 through 28) = $(bb\text{-}1.250)_{BCD}$

c(column counter) = $(29)_{10}$.

However, the instruction

Operation	Variable Field
OFLFIX	3000, 0, 21, 3, 0

would cause c(image columns 21 through 24) = $(bb\text{-}1)_{BCD}$

c(column counter) = $(25)_{10}$.

OFLFIX can convert to any fixed point number up to 16 characters long. * However, since a normalized floating point binary number can be accurate only to eight decimal integers, OFLFIX will pad trailing zeros on numbers with N+K greater than eight (or in some cases, nine). If K> 8, fractional numbers (without integers) will have significant zeros inserted between the decimal point and the eight converted characters. Note, however, that accuracy is only as great as the eight (or nine) non-zero characters, if an integer part is present.

For instance, with the following converted results,

512,064,470,000	The last 4 zeros are padding and not accurate.
512.06447000	The last 3 zeros are padding and not accurate.

* In the IB Monitor version of OUTRAN, OFLFIX converts up to 8 characters only, and this paragraph does not apply.

.00000005126447	The first 7 zeros were inserted but each character is significant.
.00512064470000	The last 4 zeros are padding and not accurate.

There are three restrictions with respect to OFLFIX whose violation will cause a transfer of control to the location determined by the current mode of OSPILL. These are:

1. N+K must not exceed 16 (8 for the IB Monitor version). Violation of this restriction leads to a control transfer with c(MQ) = 6.

2. N must be large enough to accommodate the integral part of the converted result, e.g., if c(3000) = $(-1.25)_{10}$ as in the above example the specification of N = 0 is inadequate.* If this restriction is violated, an attempt will be made to give a result wherever possible by placing the number, in floating point form, within the space provided.** If this attempt fails, control is transferred to OSPILL with c(MQ) = 11.

3. If the floating point binary number in location Y-c(T), is not normalized, it should not be so small as to cause a floating point underflow when normalized. Violation of this restriction leads to a control transfer with c(MQ) = 14.

In addition to OSPILL, the modal macros OEOR, OSCALE, OZERO, and OOVPCH apply to OFLFIX.

OFIX Y,T,C,N,K,B***

The execution of this macro-instruction causes the contents of location Y-c(T) to be regarded as a fixed point binary number with the position of the binary point determined by the value of the B-parameter*** and by the current mode of the modal macro OPOINT. This number is converted to fixed point decimal form which consists only of a signed string of decimal digits possibly with a decimal point. This string is then stored in the image subfield beginning at the column number specified by C.

---

* Note here that the effect of any decimal scale factor introduced by the modal macro OSCALE must also be provided for in the converted result. Thus, in this example, if OSCALE -1 is in effect, N=0 would be adequate since the converted result would have the value $(-0.125)_{10}$.

** The attempt to place the number in the image in floating point form will not occur when using the IB Monitor version of OUTRAN. Control will always transfer to OSPILL with c(MQ) = 11.

***The B-parameter cannot be handled by the version of OUTRAN used with the IB Monitor and is thus always assumed to have a zero value.

The B-parameter* modifies the location of the binary point in the fixed point number to be converted. It supplements the OPOINT modal value (see below) and is effective only for the OFIX in which it appears. For example, if the mode of OPOINT is normal, i.e., the binary point is between bit positions S and 1, a B value of 35 will place the binary point to the right of bit position 35. The converted number will then be an integer with no fractional part.

The rules for specifying N and K, and the make-up of the generated BCD string are the same as for OFLFIX, except that N+K may be no larger than 11. Thus, the length of the image subfield used will be N+K+2 if K $\neq$ 0, and N+ 1 if K = 0.

Example:

Suppose that c(3000) = $(000\ 000\ 000\ 005)_8$ and the binary point is between positions 33 and 34.** This is a fixed point binary representation of $(1.01)_2 = (1.25)_{10}$. Then the execution of the instruction

Operation	Variable Field
OFIX	3000, 0, 21, 3, 3

would set c(image columns 21 through 28) = $(\text{bbb1.250})_{BCD}$

c(column counter) = $(29)_{10}$

However, the instruction

Operation	Variable Field
OFIX	3000, 0, 21, 3, 0

would result in c(image columns 21 through 24) = $(\text{bbb1})_{BCD}$

c(column counter) = $(25)_{10}$

There are three restrictions with respect to OFIX whose violations will cause a transfer of control to the location determined by the current mode of OSPILL (described below). These are:

1. N+K must not exceed 11. Violation of this restriction leads to a transfer with c(MQ) = 7.

* The B-parameter cannot be handled by the version of OUTRAN used with the IB Monitor and is thus always assumed to have a zero value.

**The combined effect of OPOINT and the B-parameter is assumed to be 33.

2. N must be large enough to accommodate the integral part of the converted results, e.g., if c(3000) = $(1.25)_{10}$, as in the above example, the specification of N = 0 is inadequate. * Violation of this restriction leads to a transfer with c(MQ) = 12.

3. If A denotes the absolute value of a number, as originally represented in fixed point binary in location Y-c(T) and by the current OPOINT mode and B, ** then A x 10^K must not exceed $2^{35}-1$. Violation of this restriction leads to a transfer with c(MQ) = 16.

   For example, such a violation would occur on the execution of "OFIX 3000, 0, 21, 0, 11" where c(3000)= $(200\ 000\ 000\ 000)_8$ with normal OPOINT mode. Here, A = $(0.1)_2 = (0.5)_{10}$, so that A x $10^K = (0.5) \times 10^{11} > 2^{35}$.

   However, if OSCALE -1 were in effect, then A = $(0.5) \times 10^{-1} = (0.05)$, so that A x $10^K = (0.05) \times 10^{11} = 5 \times 10^9 < 2^{35}$, which is not a violation.

In addition to OPOINT and OSPILL, the modal macros OEOR, OSCALE, OZERO and OOVPCH also apply to OFIX.

OFXFLO Y, T, C, N, K, B***

The execution of this macro-instruction causes the contents of location Y-c(T) to be regarded as a fixed point binary number with the position of the binary point determined by B and the current mode of OPOINT. This number is converted to a normalized floating point binary number and stored in location Y-c(T). Control is then transferred to OFLOAT. All parameters, except B, are then interpreted by the OFLOAT program and the number is converted to floating point decimal form. This string is then stored in the image subfield beginning at the column specified by C.

The parameters, except B, must meet the OFLOAT specifications and the number of characters in a string, depending on N and K, is calculated exactly as for the OFLOAT routine.

---

* Note that the possible extra effect of a decimal scale factor introduced by the modal macro OSCALE (described below) must also be included in the converted result. Thus, in the example, if OSCALE -1 is in effect, the specification of N = 0 would be adequate since the converted result would be the value $(0.125)_{10}$.

** The possible effect of a decimal scale factor introduced by the modal macro OSCALE (described below) must also be included here in the value of the number.

***OFXFLO is not available with the IB Monitor.

Example:

Assume c(3000) = $(000\ 012\ 400\ 000)_8$ and the OPOINT mode is normal. Then the instruction

Operation	Variable Field
OFXFLO	3000,,10,2,2,17

will set c(image columns 10 through 18)= $(b.10.50+00)_{BCD}$

c(column counter) = $(19)_{10}$

Note that since OPOINT was normal, the B-parameter established the binary point to the right of bit position 17. The B-parameter is additive to the OPOINT modal value (see OPOINT below) and applies only to the macro-instruction in which it is specified.

OMASK Y,T,C,N

This modal macro is used to establish a mode of execution for instructions using one of the five macro-operations:

OOCTAL
OBIN
OINT
OBCC
OBCW

If the mode associated with OMASK is normal, the above five macros will function as already described. As mentioned previously, either of the instructions OMASK 0 or OUTRAN will set this mode to normal.

The execution of the instruction OMASK Y, T, C, N defines a binary subfield, where Y-c(T) specifies literally the length (in bits) of the subfield, and C specifies the number of the first bit of the subfield. N, which is normally 0, is described below. The defined subfield is taken with respect to a 36-bit word with bits numbered 1 - 36; not 0, 1 - 35. Thus, if index register 4 contains 1 when the instruction OMASK 16, 4, 22 is executed, then the subfield so defined will be the address part of the cell, i.e., bits 22 through 36.

After the instruction OMASK Y, T, C is executed, any subsequent instructions of the forms

Operation	Variable Field
OOCTAL	ALPHA, TAG, COL, LENGTH
OBIN	ALPHA, TAG, COL, LENGTH
OINT	ALPHA, TAG, COL, LENGTH

will have the same effect as usual, except that, in each case, the number used (i.e., converted and stored in the record image) will not be simply the contents of cell ALPHA-c(TAG). Instead, the subfield defined by the controlling OMASK instruction, will be extracted and converted. It is assumed here that the subfield length Y-c(T) does not exceed 35. For other cases, see below.

Example:

If c(3001) = $(700\ 011\ 777\ 777)_8$, then the execution of the instructions

Operation	Variable Field
OMASK	15, 0, 40
OINT	3000, 0, 21, 1

would set c(image columns 21 and 22) = $(b9)_{BCD}$

The extracted subfield in this case is the 15-bit string $(00011)_8$, and the word used is $(000\ 000\ 000\ 011)_8$.

Note, as implied in the above example, that the C of the OMASK instruction may exceed 36. Thus, the specified subfield, although defined in this case with respect to location 3000, does not lie in location 3000. It is in fact the decrement part of location 3001. However, any OMASK instruction (as applied to an OOCTAL, OBIN, or OINT instruction) must specify a subfield which lies entirely in a single cell. For example, the instruction OMASK 15, 0, 31, or any OMASK instruction with Y-c(T) exceeding 36, would not be allowable. If this restriction is violated, the attempted execution of the OOCTAL, OBIN or OINT instruction will lead to a transfer of control to the location determined by the current mode of OSPILL with c(MQ) = 1.

In the special case where the length Y-c(T), of the extracted subfield is specified as 36 which requires the specification of C as 1 or 1 plus some multiple of 36, the leftmost bit of this subfield, the sign bit, will be unconditionally changed to zero. For example, if c(3000) = $(400\ 000\ 000\ 005)_8$, then the execution of the instructions

Operation	Variable Field
OMASK	36, 0, 1
OINT	3000, 0, 21, 1

would result in c(image columns 21 and 22) = $(b5)_{BCD}$ (not $(-5)_{BDC}$).

Thus, the sign of the 36-bit number used by an OOCTAL, OBIN, or OINT instruction and controlled by a non-normal OMASK mode is always positive.

Note, however, that a sign bit included in an extracted subfield of length less than 36 is treated as an ordinary numeric bit. For example, if c(3000) = $(400\ 000\ 000\ 000)_8$, then the execution of the instructions

Operation	Variable Field
OMASK	3, 0, 1
OINT	3000, 0, 21, 1

would set c(image columns 21 and 22) = $(b4)_{BCD}$

After the instruction OMASK Y, T, C is executed, any subsequent instructions of the forms

Operation	Variable Field
OBCC	ALPHA, TAG, COL, LENGTH
OBCW	ALPHA, TAG, COL, LENGTH

will have the usual effect, except that the source region for the string of characters to be moved by the OBCC or OBCW instruction to the image region does not begin with the first character position in location ALPHA - c(TAG). Instead the region begins with bit position C, considering C = 1 as the leftmost bit of ALPHA - c(TAG). Thus, if c(3000) = $(ABCDEF)_{BCD}$ and c(3001)= $(GHIJKL)_{BCD}$, then the execution of the instructions

Operation	Variable Field
OMASK	0, 0, 31
OBCC	3000, 0, 21, 7

would cause c(image columns 21 through 27) = $(FGHIJKL)_{BCD}$.

In the above example, C = 31 happens to correspond to the leading bit in a character position (of which there are six: C = 1, 7, 13, 19, 25, 31). This is not required. C is arbitrary, and a character (six-bit group) may overlap two words.

Note that Y and T in the instruction OMASK Y, T, C, when applied to OBCC and OBCW, are irrelevant.

In the instruction OMASK Y, T, C, N; N must have one of the two values 0 or 1. The specification of N = 1 indicates a "floating mask." Thus, each time the OMASK instruction is applied by the execution of an instruction using one of the five macros subject to OMASK control, the number of the leading bit of the subfield is increased by the length, Y-c(T), of the subfield.

Example:

Suppose, that c(3000) = $(0001\ 0002\ 0003)_8$ and c(3001) = $(0004\ 0005\ 0006)_8$.

Then the execution of the instructions

Operation	Variable Field
OMASK	12, 0, 1, 1
OINT	3000, 0, 21, 1
OINT	3000, 0, 0, 1
OINT	3000, 0, 0, 1
OINT	3000, 0, 0, 1

would result in c(image columns 21 through 28) = $(b1b2b3b4)_{BCD}$

Note that the effect of the above example could be achieved (at more cost in space) by the instructions:

Operation	Variable Field
OMASK	12, 0, 1
OINT	3000, 0, 21, 1
OMASK	12, 0, 13
OINT	3000, 0, 0, 1
OMASK	12, 0, 25
OINT	3000, 0, 0, 1
OMASK	12, 0, 37
OINT	3000, 0, 0, 1

A floating mask can also be applied to OBCC and OBCW. For example, if

$$c(3000) = (ZZZZZA)_{BCD}$$
$$c(3001) = (ZZZZZB)_{BCD}$$
$$c(3002) = (ZZZZZC)_{BCD}$$

then the execution of the instructions

Operation	Variable Field
OMASK	36, 0, 31, 1
OBCC	3000, 0, 21, 1
OBCC	3000, 0, 0, 1
OBCC	3000, 0, 0, 1

would result in c(image columns 21 through 23) = $(ABC)_{BCD}$

OSPILL Y, T

This modal macro is used to establish a mode which relates to the six conversion macros, and which has already been mentioned in the description of these macros.

When an instruction of the form OSPILL Y, T is executed, the OSPILL subroutine computes and establishes the effective address Y-c(T). Later, control will be transferred automatically to this established address in any one of the situations given below. The contents of the MQ at the time of the control transfer to location Y-c(T) can be used to distinguish the different conditions.

Contents of MQ	Condition
0	OBCC Y, T, C, N or OBCW Y, T, C, N executed with N=0.
1	An OOCTAL, OBIN, or OINT instruction executed under control of an OMASK instruction specifying a binary subfield extending into more than one binary word (see description of OMASK).
2	OINT Y, T, C, N executed with N>11.
3	OOCTAL Y, T, C, N executed with N>13.
4	OBIN Y, T, C, N executed with N>36.

Contents of MQ	Condition
5	OFLOAT Y, T, C, N, K or OFXFLO Y, T, C, N, K, B executed with N+K > 8.
6	OFLFIX Y, T, C, N, K executed with N+K > 16. *
7	OFIX Y, T, C, N, K executed with N+K > 11.
8	OINT Y, T, C, N executed with N too small to accommodate the converted number (see OINT).
9	OOCTAL Y, T, C, N executed with N too small to accommodate the converted number (see OOCTAL).
10	OBIN Y, T, C, N executed with N too small to accommodate the converted number (see OBIN).
11	OFLFIX Y, T, C, N, K executed with N too small to accommodate the integral part of the converted result (see OFLFIX).
12	OFIX Y, T, C, N, K executed with N too small to accommodate the integral part of the converted result (see OFIX).
13	An OFLOAT instruction executed which operated on a binary floating point number that caused a floating point underflow when normalized (see OFLOAT).
14	An OFLFIX instruction executed which operated on a binary floating point number that caused a floating point underflow when normalized (see OFLFIX).
15	The execution of an OFLOAT or OFXFLO instruction gives a converted result with a decimal exponent greater than 999 in absolute value (see OFLOAT). (Note below the exception in this case with respect to the column counter.)
16	OFIX Y, T, C, N, K executed, which operated on a fixed point binary number whose value A, considering the binary point as defined by the current OPOINT mode and including the effect of a possible decimal scale factor introduced by OSCALE, is such that $A \times 10^{K} > 2^{35} - 1$. (See OFIX.)

* With the IB Monitor, when N + K > 8.

With the exception of the case where c(MQ) = 15, the execution of a conversion macro-instruction leading to any one of the above types of OSPILL control, transfer will not cause any change in the contents of the column counter or of the record image.

However, if the execution of OFLOAT Y,T,C,N,K leads to an OSPILL transfer with c(MQ) = 15, then the OFLOAT subroutine, before transferring control, will first store the principal part (omitting the exponent part) of the converted result in the specified image subfield. Then the column counter is increased by either N+1 (K=0) or N+K+2 (K $\neq$ 0), the length of the principal part. Thus, the contents of the column counter will be the number of the column where the exponent part would have commenced if it had not been too large.

The normal mode associated with OSPILL is a transfer to an address inside the SOS program. The routine beginning at this location causes printing of a message, on the debugging output unit, describing where the error occurred and giving the contents of the MQ and AC for analysis. *

It is possible to return to the object program when an OSPILL error occurs, by setting the OSPILL mode to the location of a section of coding which terminates with a TRA SYSOTI.** This will return control to the main program at the location immediately following the macro which caused the error condition.

Normally, the field specified by the macro that spilled will be filled with Xs. If Xs are not desired, a non-zero value should be placed in the decrement of SYSOTI before transferring to it. This will advance the column counter beyond the field without inserting anything.

At the time of the spill, the registers contain:

AC	decrement: column counter
MQ	address: location of macro causing spill
	type of spill

Analysis, after the spill, can be made before returning to the object program, but an Output macro cannot be used. The use of such a macro would destroy preset conditions within OUTRAN and a later return via SYSOTI would fail.

---

* When used with the IB Monitor, the normal mode of OSPILL is a recognizable stop.

**This error return feature is not available when using the IB Monitor.

OPOINT Y,T

This modal macro applies only to OFIX and OFXFLO and is used to define, by means of the literal value Y-c(T), the position of the binary point in any fixed point binary number processed by a later OFIX or OFXFLO instruction.

Example:

Suppose that c(index register 4) = 3 when the following instruction is executed:

Operation	Variable Field
OPOINT	35,4

Then, until a new OPOINT mode is established, any binary word converted by a subsequent OFIX or OFXFLO instruction would be treated as though its binary point were placed between positions 32 and 33. Thus, the 36-bit binary string $(000\ 000\ 000\ 034)_8$ would be regarded as representing the number $(11.100)_2 = (3.5)_{10}$.

The B-parameter* of an OFIX or OFXFLO instruction is additive to the current OPOINT value. Thus when an OPOINT 10 is followed by the instruction

Operation	Variable Field
OFIX	DATA,,,,,25

the B value of 25 to be added (algebraically) to the OPOINT value of 10, making the effective location of the binary point to be to the right of bit position 35.

The normal mode for OPOINT defines the binary point as being between bit positions 0 and 1. Thus, the execution of OPOINT 0 or OUTRAN causes all fixed point numbers to be treated as proper fractions.

The specification of a negative value Y-c(T); e.g., OPOINT -2; is allowable and will be properly interpreted.

OSCALE Y,T

The execution of this modal macro-instruction defines a decimal scale factor 10^S, where S is specified literally by the value Y-c(T). This scale factor is applied to all decimal numbers resulting from execution of any OFLOAT, OFIX, OFLFIX, or OFXFLO instruction, until a new OSCALE mode is established.

* With the IB Monitor, the B-parameter is not recognized.

Example:

Suppose c(index register 4) = 1 when the following instruction is executed:

Operation	Variable Field
OSCALE	6,4

Then the scale factor 10^5 is established. Then, if an OFLOAT instruction is executed which would normally produce the BCD string b123-01, the string b123+04 would be produced instead.

The normal mode for OSCALE is the scale factor $10^0 = 1$. This is, of course equivalent to applying no scale factor at all.

The specification of a negative power of 10 (e.g., OSCALE -2) is allowable.

OZERO Y,T

This modal macro applies only to the execution of OINT, OFLOAT, OFLFIX, OFXFLO and OFIX instructions. It provides a means for controlling the manner of representing results when they are zero.

There are only two modes possible for OZERO: the normal mode and the non-normal mode, specified respectively by a zero and a non-zero value for Y-c(T).

For the normal mode, the following rules hold:

1. When an OINT, OFLFIX, or OFIX instruction is executed and the decimal digits in the converted result are all zeros, the resulting BCD string which is stored in the record image will consist of either b0 (if the sign is plus) or -0 (if the sign is minus) in the two rightmost BCD positions. Blanks occupy all other positions.

2. If an OFLOAT or OFXFLO instruction is executed with $c(Y\text{-}c(T))_{9-35} = (000\ 000\ 000)_8$ (i.e., if the binary number being processed has a zero mantissa) then, regardless of the mode of OSCALE, the resulting BCD string which is stored in the record image will be either b0+00 (if the number is positive) or -0+00 (if the number is negative) in the five rightmost character positions. Blanks will occupy all other positions.

For the non-normal mode, the following rules hold:

1. Same as rule 1 above except that, if the number is positive, the string will consist entirely of blank characters.

2. Same as rule 2 above except that, if the number is positive, the string (including the positions in the exponent part) will consist entirely of blank characters.

Note that the rules for determining the number of characters stored in the record image and the consequent amount by which the column counter is increased still hold, regardless of the OZERO mode and whether the converted result is zero. Thus, if c(3000) = $(000\ 000\ 000\ 000)_8$, and if the OZERO mode is normal, then the execution of

Operation	Variable Field
OFLOAT	3000, 0, 21, 5, 3

would result in c(image columns 21 through 33) =$(bbbbbbbbb0+00)_{BCD}$

c(column counter) = $(34)_{10}$.

OOVPCH Y, T, C

This macro, which applies only to OINT, OFLOAT, OFLFIX, OFXFLO, and OFIX instructions, can be used to cause the "overpunching" of signs in the decimal number representation resulting from the execution of any of these instructions. Ordinarily, OOVPCH, if employed at all, will be used only in the construction of record images which are to be punched into cards.

In the general form OOVPCH Y, T, C the C-value must be either 1 or 2.

1. OOVPCH Y, T, 1

   When an instruction of the form OOVPCH Y, T, 1 is executed, the OOVPCH subroutine computes and establishes the literal value Y-c(T). Subsequent to this, any BCD character string resulting from the execution of an OINT, OFLOAT, OFLFIX, OFXFLO, or OFIX instruction will be altered as follows before being stored in the record image:

   The sign character of the number (in the case of OFLOAT and OFXFLO, the sign character of the principal part of the number), will be removed from the string and combined with (in the sense of overpunching) the nth character of the resulting string where n = Y-c(T). The character with which the sign is combined must be an actual numeric character, not a decimal point or a blank character resulting from a suppressed zero, although such non-numeric characters are included in the count. Thus the length of the character string, and consequently the length of the image subfield used, is reduced by 1 from the lengths as previously stated.

For example, suppose that the following OOVPCH mode has been given:

Operation	Variable Field
OOVPCH	7, 0, 1

Then, if c(3000) is such that the subsequently executed instruction OFLFIX 3000, 0, 21, 3, 3 would normally cause

$$c(\text{image columns 21 through 28}) = (\text{bbb1.250})_{BCD}$$

$$c(\text{column counter}) = (29)_{10}$$

the result which would be produced instead, because of the given OOVPCH mode, would be:

$$c(\text{image columns 21 through 27}) = (\text{bb1.25}\overset{+}{0})_{BCD}$$

$$c(\text{column counter}) = (28)_{10}.$$

Note that + overpunch is not suppressed, but actually appears as the conventional 12-punch combined with the numeric character.

2. OOVPCH Y, T, 2

A macro-instruction of this form applies only to OFLOAT and OFXFLO instructions and is used to cause overpunching with respect to the exponent part of the number only. However, since the exponent sign also plays the role of a separator its removal is accompanied by the insertion of the character E in its place. Hence, unlike the case C = 1, there is no reduction in the number of character positions used. The position Y-c(T) in this case is counted beginning in the first numeric position of the exponent part. Of course, since the exponent can consist of at most three numeric characters, the value of Y-c(T) is restricted to 1, 2, or 3.

Example:

Thus, if the result of an OFLOAT instruction is normally, say, "-125-01", then the use of the non-normal mode

Operation	Variable Field
OOVPCH	2, 0, 2

would produce the result -125E0J (i.e., -125E01) instead.

The normal mode associated with OOVPCH for both types (C = 1 and 2) is no overpunching. Both can be simultaneously set to normal by using the instruction OOVPCH 0 or OUTRAN. Either can be set independently to normal by using OOVPCH 0, 0, C with C = 1 or 2.

OEOR Y, T, C

When an instruction of the form OEOR Y, T, C is executed, the OEOR subroutine compues and establishes the effective address Y-c(T) and saves the specified C-value. Later, if a conversion macro-instruction or an OBCC, OBCW, or OBLANK instruction is executed and specifies an image subfield in which the number of the rightmost column exceeds this saved C-value, then control will be transferred to the address Y-c(T).

For example, if the instruction

Operation	Variable Field
OEOR	5000, 0, 72

has been executed, then the execution of the instruction

Operation	Variable Field
OBCC	3000, 0, 71, 3

would cause a transfer of control to location 5000.

After the attempted execution of a macro-instruction which leads to an OEOR transfer of control, the contents of the column counter will in general be meaningless. In some cases the contents of the illegal image subfield specified by the instruction will have been changed by the attempted execution.

The normal mode associated with OEOR* is a transfer of control whenever the rightmost column of the specified image subfield exceeds 120. The transfer will be to a location within the SOS program where a message will be written on the debugging output unit indicating in which macro the error occurred.

ORPT R, I, J

This special control macro can be used to repeat the execution of any active internal processing macro-instruction, i.e., of any conversion macro-instruction or OBCC, OBCW, or OBLANK instruction.

* With the IB Monitor, the normal mode of OEOR leads to a recognizable stop.

When an instruction of the form ORPT R, I, J is executed, the first active internal processing macro-instruction which is executed thereafter will be executed a total of R times (instead of only once), the first time using the Y, T, and C fields as specified, the second time with Y increased by I, and C increased by J; etc. Note that if the macro-instruction which is repeated uses indirect addressing, it is the direct address, Y, and not the indirect address, $c(Y)_{21-35}$, which is increased by I.

For example, the execution of the two instructions

Operation	Variable Field
ORPT	5, 1, 6
OBCC	3000, 0, 21, 2

is equivalent to the execution of the five instructions

Operation	Variable Field
OBCC	3000, 0, 21, 2
OBCC	3001, 0, 27, 2
OBCC	3002, 0, 33, 2
OBCC	3003, 0, 39, 2
OBCC	3004, 0, 45, 2

In ORPT R, I, J; the value of R must be non-zero. The specification of a negative value for I or J (e.g., ORPT 5, -1, -6) is allowable and will have the effect of decreasing the value of Y and C, respectively, in the subsequent repeated executions of the active macro-instruction.

THE WRITE-OUT MACROS

The following six macros are concerned with the write-out stage. The first two are active macros and the last four are modal macros.

OSCRIB Y, T, C, N (SHARE Monitor System)*

OSCRIB is used to initiate writing of a tape record, a printed line, or a punched card.

Y-c(T) specifies the location of a cell whose address contains the standard 709/7090 code for the desired output unit. C is used to specify the number of words to be transmitted from the I-region. N makes possible the distinction between the eight modes of possible output (see Table 1).

MODE	Function	709/7090 Unit Code (octal)	Required N	C	W
STH	Storage to BCD tape	X201 through X210	0	C = 0 C ≠ 0	W = 14 W = C
STHB	Storage to Binary tape BCD for columnar binary punching off-line. (Results in Hollerith cards) cards)	X221 through X230	3	ineffective	24
STHP	Storage to BCD tape. Carriage control by OHEAD and OSPACE (for off-line printing)	X201 through X210	1	C = 0 C ≤ 20 C > 20	W = 20 W = C W = 20
STB	Storage to Binary tape	X221 through X230	2	ineffective	28
SPH	Storage to on-line printer (First character in buffer is for carriage control)	Y361 (SYSPRT)	0	C = 0 C ≤ 20 C > 20	W = 20 W = C W = 20
SPHP	Storage to on-line printer (Carriage control by OHEAD and OSPACE)	Y361 (SYSPRT)	1	C = 0 C ≤ 20 C > 20	W = 20 W = C W = 20
SCH	Storage to on-line punch (BCD)	Y341 (SYSPCH)	0	ineffective	12
SCB	Storage to on-line punch (Columnar binary)	Y341 (SYSPCH)	1	ineffective	24

Notes:

X and Y are the channel numbers required. A programmer need not be concerned with the numbers if symbolic tape references are used.

W is the number of words, in the I-region, that are actually used.

If the 709/7090 code given is not acceptable, or if W is too large, an error condiction occurs. This is controlled by OREDUN.

* See page 07.02.40 for description of OSCRIB for IB Monitor System.

Output Modes

STH:Tape-BCD-Normal

The first W words in the I-region (see Table 1) are written on tape as a BCD record.

If W is more than 28 words, then the standard I-region cannot be used, and a larger I-region must be defined by means of OIMAGE.

This type of output is usually used for off-line Hollerith card punching, or for off-line printing under single space or double space control. To punch a full 80-column card, W should be 14, in which case the last four characters are not relevant.

STHB: Tape-Binary-Hollerith Image

The first 12 words in the current buffer are converted from BCD to a 24-word column binary image and written on tape as a binary record.

This type of output allows the off-line punching of Hollerith cards with the off-line punch in the binary mode. Thus, mixed information (BCD and binary) can be punched from the same tape with one setting of the punch control switch (binary mode).

STHP:Tape-BCD-Special

This type of output is used for off-line printing with the printer Carriage Control switch set to Program.

The first W words in the I-region are shifted right one character (6 bits), and a spacing character is inserted in the first position. If W = 20, the last character is lost. If W < 20, then W+1 words are transmitted, and the last 5 characters are blanks.

For other than standard single spacing, OSPACE can be used to control spacing. Headings and line counts can be controlled by OHEAD.

Automatic page overflow does not take place; however, OHEAD can be used to control page overflow.

STB:Tape-Binary

The first 28 words in the I-region are written on tape as a binary record.

This is normally used to punch column binary cards off-line. *

SPH:Print-on-line.

The first W BCD words in the current buffer (I_1 or I_2) are printed on-line (except the first character in the buffer - see below).

If W is 12 or less, a 72-character line will be printed (any trailing characters beyond W up to character 72 will be automatically blank).

If W is greater than 12, a 120-character line will be printed. Note that the speed of printing for 120-character lines is 75 lines per minute, and that the OSCRIB routine must wait a full print cycle for the printing of the right hand portion of the line.

The first character of the buffer is assumed to be for carriage control. (See IBM 700-7000 Series Auxiliary Operations manual, form A22-6502.) It should be placed there by the programmer for his own spacing requirements. Since the first character is for carriage control, the maximum number of BCD characters that can be printed on line is 119, not 120.

SPHP:Print-on-line-Special

The first W BCD words in the current buffer are printed on-line.

The type of line to be printed is similar to that described above for the SPH mode. However, in this mode, the first character is a spacing character placed there by the OHEAD or OSPACE routines. All other characters are moved one character space to the right so that only (6*W)-1 characters will be printed; the maximum for one line being 119 characters.

For other than standard single spacing, OSPACE can be used to control spacing. Headings and line counts can be controlled by "OHEAD."

SCH:Punch-Hollerith

The first 12 BCD words in the I-region are punched on-line as the first 72-columns of a Hollerith card.

---

* The first column should contain 7- and 9-punches, which must be supplied by the programmer.

SCB:Punch Binary

The first 24 words of the current buffer are converted from row binary to column binary form and punched on-line in the first 72 columns on a card.

The OSCRIB routine will not automatically insert the characteristic 7- and 9-punches into the column binary card. They must be supplied, if desired, by the programmer.

Special Conditions

A special condition can occur during the execution of an OSCRIB instruction. In this case, instead of returning control to the instruction following the OSCRIB instruction, control is transferred to a special location.

Transfer of control might occur to the location specified by OHEAD, if effective. If an end-of-tape mark is encountered, or an error condition occurs, control is transferred to a standard error message routine, or to the location specified by OTPEND or OREDUN, respectively, if they are effective. The MQ and AC will in general contain some pertinent information. Transmission of the current record may or may not take place. Table 2 gives a summary of the different special conditions that can occur.

SPECIAL CONDITION		Modal Macro Effective	Does Transmission Take Place?	MQ Register	Accumulator	
					Decrement	Address
Headings		OHEAD	NO	-	—	CUNIT
End-of-Tape Mark	CTUNIT = PTUNIT	OTPEND	NO	-	CUNIT	PUNIT
	CTUNIT ≠ PTUNIT	OTPEND	YES	-	CUNIT	PUNIT
Redundancy Error	CTUNIT = PTUNIT	OREDUN	NO	0	CUNIT	PUNIT
	CTUNIT ≠ PTUNIT	OREDUN	YES	0	CUNIT	PUNIT
I-Region Too Small		OREDUN	NO	1	Δ C	CUNIT
Unit Not Assigned		OREDUN	NO	2	CUNIT	PUNIT
Illegal N or Unit Name		OREDUN	NO	3	CUNIT	PUNIT

CUNIT and PUNIT are the standard machine codes for the current and previous transmission units, respectively. CTUNIT and PTUNIT mean the current and previous tape units, respectively, regardless of mode (BCD or Binary). When there is no previous transmission PUNIT = 0. ΔC is the difference between W for the OSCRIB instruction and the actual size of the I-region.

Headings.

This applies only when current output type is Print-on-line (SPHP) or Tape-BCD-Special (STHP) and the current mode of OHEAD is non-normal. Whenever the number of lines, including spaces, that have been transmitted is equal to or greater than the number specified by the OHEAD mode, then control is transferred to the location specified by the OHEAD mode. Transmission does <u>not</u> take place.

End-of-Tape Mark.

Whenever the previous transmission involved a tape, then, during the execution of the current OSCRIB instruction, the transmission which was initiated by the last OSCRIB instruction is tested for end of tape. If an end-of-tape condition occurred, control is transferred to the location determined by the current mode of OTPEND.

Transmission does not take place if the current and previous tape units are the same. If the tape units are different, the current transmission is initiated.

Redundancy Error.

Whenever the previous transmission involved a tape, then during the execution of the current OSCRIB instruction, the transmission initiated by the last OSCRIB is tested for tape redundancy.* If the indicator is On, the SHARE Monitor Bad Spot Routine (SYSBAD) attempts to rewrite the previous record.** If the redundancy persists, control is transferred to the location determined by the current mode of OREDUN. Transmission does not take place if the current and previous units are the same. If the units are different, the current transmission is initiated.

---

* Only if the End-of-Tape indicator was not On, since the end-of-tape test takes precedence over the redundancy test.

** In the case of Tape-BCD-Special (SPHP), if extra one-word spacing records were written on the tape, these records are not rewritten. If the error occurred on such a record, only the last information record is rewritten, and the error in the spacing record is not tested. This type of error can be recognized only during the actual off-line printing.

Example:

Assume that the instructions OSCRIB SYSAR1 followed by OSCRIB UNIT2 are given, that OREDUN Z is effective, $c(SYSAR1)_{21-35}=(01201)_8$, and $c(UNIT2)_{21-35}=(01361)_8$. Suppose that the record on A1 cannot be correctly written because of a bad spot on the tape. The first OSCRIB instruction initiates writing on A1, and the second OSCRIB instruction, before initiating the printer output, tests A1 and tries to rewrite the record ten times (under control of the SHARE Monitor Bad Spot Routine). Then the printer output is initiated, and control is transferred to Location Z, with c(MQ)=0, $c(AC)_{3-17}=(01361)_8$, and $c(AC)_{21-35}=(01201)_8$. If $c(UNIT2)_{21-35}$ had been $(01201)_8$ or $(01221)_8$, then the current transmission would not have been initiated. If $c(UNIT2)_{21-35}$ had been $(02201)_8$, then the current transmission would have been initiated before any testing.

I-Region Too Small

Whenever the W of an OSCRIB instruction (see Table 1) exceeds the length of the I-region, control is transferred to the location determined by the current mode of OREDUN. In this case, the current transmission does not take place. If the standard 28-plus-28 word I-region is used, this condition can occur only if the output type is Tape-BCD-Normal (STH) with C exceeding 28. For example, the execution of the following instruction will result in this type of error unless an associated I-region of at least 50-plus-50 words has been provided by an OIMAGE instruction:

Operation	Variable Field
OSCRIB	SYSARI, 0, 50, 0

where the address part of SYSARI contains a standard code for BCD tape.

Unit Not Assigned.

If the address portion of Y-c(T) is zero, control is transferred according to the current mode of OREDUN. In this case, the current transmission does <u>not</u> take place.

The contents of Y-c(T) might be zero if a system tape symbol, e.g., SYSARI, were used and that symbolic tape were not assigned by an ASSIGN control card or by SYSTAS.

Illegal N or Unit Name.

The choice of mode by OSCRIB is based upon the decoding of bits from the output unit address, together with N in cases of ambiguity. When an absolute unit address is not compatible with the N given, and OSCRIB cannot decode the information, control is transferred to the location specified by the current mode of OREDUN.

OSCRIB Y,T,C,N (IB Monitor System)*

OSCRIB is used to initiate writing of a tape record, a printed line, or a punched card from the I-region.

Y-c(T) specifies the location of a cell whose address contains the standard 709/7090 code for the desired output unit. C is used to specify the number of words to be transmitted from the I-region. In Table 3, below, the actual number of words transmitted is denoted by W. N makes possible the distinction between two types of BCD tape outputs, and two types of card punching.

OUTPUT TYPE	709 UNIT CODE (octal)	N	C	W
Punch-Hollerith	Y341	0	ineffective	12
Tape-BCD-Normal	X201 through X210	0	C=0 C≠0	14 C
Tape-BCD-Special	X201 through X210	1	C= 0 or C≥20 1≤ C ≤ 19	20* C
Print-On-Line	Y361	ineffective	0≤C≤12 13≤C	12 20
Punch-Column-Binary	Y341	1	ineffective	24
Tape-Binary	X221 through X230	ineffective	ineffective	28

NOTES:

X and Y are the channel numbers required.
X can be 1,2,3,4,5, or 6. Y should be 1, 3, or 5.
* indicates that the last character is lost.
W is the number of words in the I-region that are actually used. The number of words that make up the actual output record is not necessarily W. However, this does not, in general, concern the programmer. If the unit address given is not acceptable, or if W is too large, then an error condition occurs. OREDUN controls this.

---

* See page 07.02.34 for description of OSCRIB for SHARE Monitor System.

Four of the six output types require that OSCRIB perform a special conversion before transmission is initiated. For this purpose, a special buffer is set aside for the final output image. The programmer does not have to be concerned with the special buffers.

Output Types

Punch-Hollerith.

The first 12 BCD words in the I-region are punched on-line in Hollerith as the first 72 columns of a card.

Tape-BCD-Normal.

The first W words in the I-region are written on tape as a BCD record. If W is more than 28 words, the standard I-region cannot be used and a larger I-region must be defined by means of OIMAGE.

This type of output is usually used for off-line Hollerith card punching, or for off-line printing under single space or double space control. To punch a full 80-column card, W should be 14. In this case, the last four characters will be irrelevant.

Tape-BCD-Special.

This type of output is used for off-line printing, under program control.

The contents of the first W words in the I-region are shifted right one character position and a spacing character, for the off-line printer, is inserted in the first position. If W = 20, the last character is lost. If W < 20, then W + 1 words are transmitted and the last 5 characters of the last word are blanks.

If non-standard spacing is desired, OSPACE can be used to control spacing. Headings and line counts can be controlled by OHEAD. Automatic page overflow does not take place. However, OHEAD could be used to control overflow.

Print-On-Line.

The first 12 or 20 BCD words in the I-region are printed as a 72 or 120 character line, respectively, on the on-line printer.

If non-standard spacing is desired, OSPACE can be used to control spacing. Headings and line counts can be controlled by OHEAD.

For maximum speed (150 lines per minute) lines should consist of 72 characters only. If 120 characters are required, printing speed will be at most 75 lines per minute, because the OSCRIB subroutine must wait a full print cycle, during the left-half transmission, before the right-half transmission can be initiated.

Punch-Column-Binary.

The first 24 words in the I-region are converted to column binary and punched on-line as the first 72 columns of a card.

Tape-Binary.

The first 28 words in the I-region are written on tape as a binary record.

This is normally used to punch off-line column binary cards from the tape.

Special Conditions

A special condition can occur during the execution of an OSCRIB instruction. In this case, instead of returning control to the instruction following the OSCRIB instruction, control is transferred to some special location.

Transfer of control might occur to the location specified by OHEAD, if effective. If a tape end mark is encountered, or an error condition occurs, control is transferred to a standard debugging routine, * or to the location specified by OTPEND or OREDUN, respectively, if they are effective. Transmission of the current record may or may not take place.

Table 4 gives a summary of the different special conditions that can occur. The MQ and the AC will in general contain pertinent information.

SPECIAL CONDITION		Modal Macro Effective	Does Transmission Take Place ?	MQ Register	Accumulator	
					Decrement	Address
Headings		OHEAD	NO	-	—	CUNIT
End-of-Tape Mark	CTUNIT = PTUNIT	OTPEND	NO	-	CUNIT	PUNIT
	CTUNIT ≠ PTUNIT	OTPEND	YES	-	CUNIT	PUNIT
Redundancy Error	CTUNIT = PTUNIT	OREDUN	NO	0	CUNIT	PUNIT
	CTUNIT ≠ PTUNIT	OREDUN	YES	0	CUNIT	PUNIT
I-Region Too Small		OREDUN	NO	1	ΔC	CUNIT
709 Code Non-Acceptable		OREDUN	NO	2	CUNIT	PUNIT

CUNIT and PUNIT are standard codes for the current and previous transmission units, respectively. When there is no previous transmission PUNIT = 0.

CTUNIT and PTUNIT are the current and previous tape units, respectively, regardless of mode (BCD or Binary).

ΔC is the difference between W for the OSCRIB instruction and the actual size of the I-region.

* At present, program halts.

Headings.

This applies only when the current output type is Print-On-Line or Tape-BCD-Special and the current mode of OHEAD is non-normal. Whenever the number of spaces plus the number of lines transmitted is equal to or greater than the number specified by the OHEAD mode, control is transferred to the location specified by the OHEAD mode. Transmission does <u>not</u> take place.

End of Tape.

Whenever the previous transmission involved a tape, the transmission initiated by the last OSCRIB instruction is tested for end-of-tape condition during execution of the current OSCRIB. If the End-of-Tape indicator is On, control is transferred to the location specified by the current mode of OTPEND. If the tape units are different, the current transmission is initiated before the transfer occurs. Transmission does not take place if the current and previous tape units are the same.

Redundancy Error.

Whenever the previous transmission involved a tape, the transmission which was initiated by the last OSCRIB instruction is tested for the tape redundancy** during the execution of the current OSCRIB.

If the Tape Redundancy indicator is On, up to three attempts are made to rewrite the previous record.*** If the error still persists, control is transferred to the location specified by the current mode of OREDUN. If the tape units are different, the current transmission is initiated before the transfer occurs.

Example:

Assume that the instructions OSCRIB UNIT1 and OSCRIB UNIT2 were given, when OREDUN Z is effective. Suppose also that $c(UNIT1)_{21-35} = (01201)_8$, $c(UNIT2)_{21-35} = (01361)_8$, and that the record on A1 cannot be correctly written because of a bad spot on the tape.

** Only if the End-of-Tape indicator was not On, since the end-of-tape test takes precedence over the redundancy test.

***In the case of Tape-BCD-Special, if extra one-word spacing records were written on the tape, these records are not rewritten. If the error occurred on such a record, only the last information record is rewritten, probably once, and the error in the spacing record is not tested. This type of error can be recognized only during the actual off-line printing.

The first OSCRIB instruction initiates writing on A1, and the second OSCRIB instruction, before initiating the printer output, tests A1 and rewrites the record three times (including three backspaces). Then transmission of printer output is initiated, and control is transferred to location Z, with c(MQ) = 0, $c(AC)_{3-17} = (01361)_8$, and $c(AC)_{21-35} = (01201)_8$.

If $c(UNIT2)_{21-35}$ had been $(01201)_8$ or $(01221)_8$, then the current transmission would not have been initiated. On the other hand, if $c(UNIT2)_{21-35}$ had been $(02201)_8$, the current transmission would have been initiated before any testing.

I-Region too Small

Whenever the W of the OSCRIB instruction (see Table 3) exceeds the length of the I-region, control is transferred according to the current mode of OREDUN. In this case, the current transmission does <u>not</u> take place. When the standard 28-plus-28 word I-region is used, this error condition can occur only if the output type is Tape-BCD-Normal with C exceeding 28.

For example, the execution of the following instruction will cause this type of error unless an output I-region of at least 50-plus-50 words has been provided by an OIMAGE instruction:

Operation	Variable Field
OSCRIB	UNIT, 0, 50, 0

where the address part of UNIT contains a 709/7090 code for a BCD tape.

709/7090 Code Not Acceptable

Whenever the 709/7090 unit code is not acceptable to OSCRIB, control is transferred to the location specified by the current mode of OREDUN. In this case, the current transmission does not take place.

The following codes are acceptable:

Channels 1, 2, 3, 4, 5, 6. (Even if some channels are not operative.)
Unit codes (in octal): 341, 361, 200 through 237.
Channels 0 and 7, and all other unit codes are considered as an error.

USE OF THE BUFFER AREA

In order to permit use of the simultaneous output and computing features provided on the 709/7090, the I-region is split into two buffers, I_1 and I_2, which operate as described below. The source region for a record written by a given OSCRIB instruction may be either I_1 or I_2, and, normally, successive executions of OSCRIB instructions will use I_1 and I_2 alternately. * For example, consider the following sequence:

Operation	Variable Field
OSCRIB	UNIT
OSCRIB	UNIT
OSCRIB	UNIT

If the first OSCRIB instruction writes a record from I_2, the second OSCRIB instruction would write the next record from I_1, the third from I_2, and so on.

When an OSCRIB instruction is executed, the actual writing of the record from I_1 or I_2, is not completely carried out, but merely initiated, and nothing further is done for this current record, until the next OSCRIB (or OREADY) is executed. At that time, the OSCRIB (or OREADY) subroutine delays until the previously initiated transmission is completed, and checks for the end of tape and redundancy. The transmission specified by the current OSCRIB instruction is initiated, as soon as possible, from the alternate buffer, e.g., from I_1 if I_2 was previously used.

If the channels are different, the current transmission is initiated before the delay and test of the previous transmission. However, if the channels are the same, the current transmission is initiated after the delay and test.

If an end of tape or redundancy condition is detected, and the units are the same, the current transmission is not initiated.

Thus, immediately after the execution of an OSCRIB instruction, one buffer is being written, and the other is available for use. In particular, this buffer is available for use by all the active internal processing macros which will ordinarily be employed to prepare the buffer for the next OSCRIB instruction. The programmer is never required to specify which of the two buffers (I_1 or I_2) is to be used, provided the internal processing macros refer to the next OSCRIB to be executed.

* Unless OREADY or OIMAGE instructions intervene, and if the current transmission takes place.

In using OSCRIB, the alternation of the buffers I_1 and I_2 takes place whenever the transmission specified is initiated. * However, if no OSCRIB (or OREADY) instructions are used in the source program, the internal processing macros will always take I_1 as their destination region. By means of OREADY or OIMAGE, the buffer I_1 can be used at all times.

Example:

Suppose that a line of information is to be both printed on-line and written on tape for off-line printing. For off-line printing, one tape is to be prepared for printing with single space control, and another for printing under program control. Four sequences which will accomplish the same end result are shown below. It is assumed that:

$c(SYSPRT)_{21-35} = (01361)_8$ (on-line printer)

$c(SYSAR3)_{21-35} = (02205)_8$ (tape B5)

$c(SYSAU4)_{21-35} = (01206)_8$ (tape A6)

Operation	Variable Field	
.	.	internal processing
.	.	macro-instructions
.	.	
OSCRIB	SYSPRT, 0, 20	
OREADY		
OSCRIB	SYSAR3, 0, 20, 0	
OREADY		
OSCRIB	SYSAU4, 0, 20, 1	

In this sequence, the OREADY instructions are somewhat wasteful of time, since each causes a delay until the previous OSCRIB transmission is completed. Either the I_1 or the I_2 region is used at all times, depending on the state of the alternation at the beginning of the program.

Operation	Variable Field	
OREADY		
OIMAGE		set to I_1
.	.	internal processing
.	.	macro-instructions
.	.	
OSCRIB	SYSPRT, 0, 20	
OIMAGE		set to I_1
OSCRIB	SYSAR3, 0, 20, 0	
OIMAGE		set to I_1
OSCRIB	SYSAU4, 0, 20, 1	

* Under some special circumstances, the OSCRIB subroutine will not perform the buffer alternation because the current transmission was not initiated. The macro OHEAD or some error condition (already mentioned) can cause this.

Here the OIMAGE instruction is used instead of OREADY, so that no excessive time is wasted. However, I_1 is always used here since the execution of the OIMAGE instruction always sets the alternation to I_1 (see below). The OREADY instruction appearing at the beginning should be used to insure the completion of the previous transmission, if any, since such a transmission might be writing out I_1, which would, of course, be disturbed by the execution of the indicated internal processing macro-instructions.

Operation	Variable Field	
.	.	internal processing
.	.	macro-instructions
.	.	
OSCRIB	SYSPRT, 0, 20	
.	.	internal processing
.	.	macro-instructions
.	.	
OSCRIB	SYSAR3, 0, 20, 0	
OSCRIB	SYSAU4, 0, 20, 1	

In this sequence, both buffers are filled with the same information by using the same internal processing macro-instructions on the two different alternations. Such a procedure might waste some time if the internal processing time is great.

Operation	Variable Field	
OIMAGE	X, 0, 20	
.	.	internal processing
.	.	macro-instructions
.	.	
OSCRIB	SYSPRT, 0, 20	
OBCW	X, 0, 20	
OSCRIB	SYSAR3, 0, 20, 0	
OSCRIB	SYSAU4, 0, 20, 1	

In this sequence, a non-standard I-region beginning at location X is defined, and the OBCW instruction is used to set up the buffer I_2 (beginning at location X + 20).

The above four examples should serve as some indication of how the buffers I_1 and I_2 function.

The following rules govern the alternation procedure:

1. Prior to execution of any OSCRIB (or OREADY) instruction, the destination region, D, for use by any internal processing macro-instruction is set to I_1.

2. The execution of OIMAGE 0 or the instruction OUTRAN always sets D to I_1.

3. The destination region used by an internal processing macro-instruction becomes the source region for the next OSCRIB instruction if no OREADY, OIMAGE, or OUTRAN instruction intervenes.

4. If D is I_1 just before the execution of an OSCRIB instruction, then D is I_2 just after the normal execution of the OSCRIB instruction. ("Normal execution" here means whenever current transmission is initiated - see Tables 2 and 4 above.)

5. If D is I_1 just before the execution of an OREADY instruction, then D is I_2 just after the execution of the OREADY instruction. (No exceptions.)

The action of the OSCRIB subroutine proceeds chronologically as follows:

1. Test for unassigned output unit. *
2. Test for illegal N or unit name. *
3. Form W as in Tables 1 and 3.
4. Test for non-normal OHEAD mode.
5. Test for too small an I-region.
6. Prepare final output record, if any conversion is required. (The original I-region is not disturbed here.)
7. Compare previous and current transmissions.
   a. If there was no previous transmission, ** or if the current channel is different from the previous channel, then reset the End-of-Tape and Redundancy indicators for the current channel. Go to step 10.
   b. If the current channel is the same as the previous channel, wait until the previous transmission is completed.
8. If the previous transmission did not involve a tape, go to step 10.
9. Test the End-of-Tape indicator for previous transmission. If On, set end-of-tape exit:
   a. If CTUNIT = PTUNIT, go to step 15.
   b. If CTUNIT $\neq$ PTUNIT and step 10 has already been done, go to step 14. Otherwise, go to step 10.
10. Test for redundancy during previous transmission. If a redundancy occurred, attempt to rewrite record. If redundancy persists set up error exit and go to step 9(a) or 9(b).

---

* When using OUTRAN under IB Monitor control, steps 1 and 2 are replaced by a single test for non-acceptable output unit code.

** "No previous transmission" means that this is the first OSCRIB instruction executed, or an OREADY instruction was given between the last OSCRIB instruction and the present one. The macro OUTRAN does not affect this.

11. Modify the OHEAD counter if required (see OHEAD).
12. Initiate current transmission.
13. If there was a previous transmission, and if the current channel and the previous channel are different, wait until the previous transmission is completed. Otherwise, go to step 15.
14. If the previous transmission involved a tape, go to step 9.
15. Alternate the buffers I_1 and I_2.
16. Transfer control back to the program, or to the location determined by the current mode of OREDUN or of OTPEND.

OREADY

The programmer will ordinarily use an OREADY instruction to terminate a series of OSCRIB instructions. The OREADY subroutine delays until the last transmission initiated by an OSCRIB instruction is completed, and then checks for an end-of-tape condition and redundancy error. It will also alternate the buffers I_1 and I_2.

For example, suppose it is required to construct and write 50 records on tape B3 in BCD for later off-line punching. No special error routines are to be used. A sample program for accomplishing this is given below.

Location	Operation	Variable Field	Comments
START	REWB	3	Rewind tape B3.
	OUTRAN		Set all modes to normal.
	AXT	50, 1	Initialize loop.
LOOP			
	.	.	Use internal proc-
	.	.	essing macros to
	.	.	process record 1,
	.	.	then process record
	.	.	n while writing record
	.	.	n - 1, where n = 2, 3,
	.	.	. . . . . , 50.
	OSCRIB	START, 0, 14, 0	Initiate writing record
*			n=1, 2, ... 50 and check
*			records n=1, 2, ... , 49.
	TIX	LOOP, 1, 1	Count records.
	OREADY		Check record n=50.
	TRA	OUT	Finish.

The internal processing macro-instructions process the first record. Then, simultaneous writing and processing take place. While the last record is being written, no processing takes place. Finally the OREADY instruction delays until this last transmission is completed, and checks the transmission.

In the above example, the two buffers I_1 and I_2 were used alternately, starting with I_1. If the programmer so chooses, he may avoid the alternation and use only the single buffer I_1. This may be accomplished by execution of an OREADY instruction following every OSCRIB, since OREADY will alternate the buffer back to I_1. The use of this device will, in general, defeat the purpose of the alternation logic, i.e., simultaneous writing and processing. Thus, the function performed by the sample program above might have been accomplished as follows:

Location	Operation	Variable Field	Comments
START	REWB	3	Rewind tape B3.
	OUTRAN		Set all modes to normal.
	AXT	50, 1	Initialize loop.
LOOP	.	.	Use internal proc-
	.	.	essing macros to
	.	.	process record
	.	.	n=1, 2. . . , 50.
	OSCRIB	START, 0, 14, 0	Initiate writing record
*			n=1, 2, . . . , 50
	OREADY		Check record
*			n=1, 2, . . . , 50
	TIX	LOOP, 1, 1	Count records.
	TRA	SYSTEM	Finish

In this program, the transmission, initiated by each of the 50 executions of the OSCRIB instruction, is checked at once by the OREADY instruction. This double alternation of the buffers I_1 and I_2 (one alternation by the OSCRIB instruction and the other by the OREADY instruction) results in the continual use of I_1 only. The logic of this latter program is of course simpler than that of the former one, but the relative loss of time due to the serial processing and transmission (rather than parallel, as in the former case) may be considerable.

The OREADY subroutine can be characterized by noting that it performs functions similar to those of the OSCRIB subroutine, except initiation of a new transmission. In particular, the execution of an OREADY instruction, like that of an OSCRIB instruction may result in:

1. A transfer of control back to the instruction following the OREADY instruction. This is the normal case, i.e., when the previously initiated transmission, having been checked by the OREADY subroutine, does not result in an error or end-of-tape condition.

2. A transfer of control according to the mode of OTPEND. This is the case when the previously initiated transmission results in an end-of-tape condition. The decrement of the accumulator contains 0, and the address contains the unit code.

3. A transfer of control according to the mode of OREDUN, with c(MQ)=0. This is the case when the previously initiated transmission is unsuccessful because of a persistent tape redundancy error. The decrement of the accumulator contains 0, and the address contains the unit code.

Unlike OSCRIB, OREADY always alternates the buffers I_1 and I_2, even in the case of a special transfer of control according to OTPEND or OREDUN.

If an OREADY instruction is executed and there is no previous transmission (e.g., before execution of an OSCRIB or after execution of an OREADY), the only action taken will be to alternate the buffers I_1 and I_2.

The OREADY subroutine proceeds chronologically as follows:

1. Alternate the buffers I_1 and I_2.
2. Test whether OSCRIB or OREADY was executed last.
   a. If OREADY, or neither, operation is completed.
   b. If OSCRIB, go to step 3.
3. Wait until the previous transmission initiated by OSCRIB is completed.
4. Test whether the previous transmission involved a tape.
   a. If not a tape, operation is completed.
   b. If a tape, got to step 5.
5. Test for end-of-tape condition.
   a. If no, go to step 6.
   b. If yes, write end of file on the tape unit and transfer control according to OTPEND.
6. Test for redundancy.
   a. If no, go to step 7.
   b. If yes, the standard attempt is made to recover by the monitor bad spot routine.* If the error still persists, control is transferred according to the mode of OREDUN, with c(MQ) = 0.
7. The operation is completed.

---

* In the case of a tape prepared for off-line printing, if there were extra spacing records written on the tape, these records are not rewritten. If the error occurred on such a record, only the last record (the main one) is rewritten, probably once, and the error is not corrected. This type of error can be recognied only during the actual off-line printing.

OSPACE Y,T,C,N

This modal macro provides space control when the output unit is a printer or a tape prepared for off-line tape-to-printer operation with the carriage control switch set to "Program". The OSPACE mode is examined whenever an OSCRIB instruction specifies one of these two types of output, which are referred to below as "on-line" and "off-line" printing, respectively.

N specifies whether this OSPACE instruction applies to off-line or on-line. If N = 1, only off-line printing is affected. If N = 2, only on-line printing is affected. Hence, two independent OSPACE modes can be given. By specifying N = 3, one OSPACE instruction will affect both types of output; i.e., the instruction OSPACE Y,T,C,3 is equivalent to the two consecutive instructions OSPACE Y,T,C,1 and OSPACE Y,T,C,2.

The value Y-c(T) specifies literally the number of lines to be spaced before printing the next line, including the printed line itself. A Y-c(T) equal to 0 or 1 indicates single spacing (provided C = 0, see below).

C specifies whether the page is to be ejected before spacing takes place. If C = 0, normal spacing (no eject) is indicated. If C = 1, this indicates that the page is to be ejected before spacing. In this case only, there is a distinction between Y-c(T)= 0 and Y-c(T)= 1. Y-c(T)= 0 means eject and print, and Y-c(T)= 1 means eject, space one line, and then print.

The normal mode associated with OSPACE is no ejection and single spacing. Both modes (N = 1 and 2) can be set simultaneously to normal by using OSPACE 0, or each can be set independently to normal by using OSPACE 0,0,0,N where N= 1 or 2.

OHEAD Y,T,C,N,K

This modal macro controls line counts and headings when output is for on-line or off-line printing. The OHEAD mode is examined whenever on OSCRIB instruction specifies one of these two types of output.

K specifies whether this OHEAD instruction applies to off-line or on-line printing. The K-value should be either 1 or 2. K = 1 specifies off-line, and K = 2 specifies on-line. Hence, two independent OHEAD modes can be given. By specifying K = 3, one OHEAD instruction will affect both types of output. Thus, the instruction OHEAD Y,T,C,N, 3 is equivalent to the two consecutive instructions OHEAD Y,T,C,N, 1 and OHEAD Y,T,C,N,2.

There are two counters that are set aside; one for off-line and the other for on-line printing. Each counter contains the number of lines printed and spaced since the last execution of an OHEAD instruction. At the start, each counter is reset to zero. Whenever a line is printed, the total number of spaces used up is added to the corresponding counter. This means that 1 is added to the counter if the corresponding OSPACE mode is normal. If the OSPACE mode is non-normal, then the number of spaces, S, specified by the current OSPACE mode, is added to the counter.

N controls the number of lines, including spaces, which are normally to be printed before a transfer of control (to a heading program) occurs. Each time the OSCRIB subroutine attempts to transmit one of the two types of output, the contents of the appropriate counter is compared to N. If the counter contains a number less than N, the usual transmission takes place* and the value of S is added to the counter. However, if the contents of the counter is greater than or equal to N, then transmission does not take place. Instead, control is transferred to the location Y-c(T) specified in the controlling OHEAD instruction.

However, before this transfer of control, the following changes will take place: If C = 0 in the controlling OHEAD, the counter is reset to 0; if C is 1, the current page will be ejected, and the counter is reset to 1 (not 0) since the succeeding printing will be one line lower. (This is true because spacing always occur before printing a line.)

Example:

Suppose 500 lines, for "off-line" printing are to be written on tape A2, with each line consisting of 120 characters, single spaced, and that a new page, with a 48-character page heading line is to be started after every 50 lines.

The program might be:

Location	Operation	Variable Field	Comments
START	OUTRAN		Set all modes to normal.
	AXT	500, 1	Initialize counter for
*			total lines.
	OHEAD	A, 0, 1, 0, 1	Initialize first heading.
LOOP	.	.	Set up line image by
.	.	.	using internal proc-
.	.	.	essing macros.
.	.	.	
LOOP1	OSCRIB	TAPEA2, 0, 20, 1	Initiate transmission,
*			or transfer control
*			to A or B.

* Assuming no end-of-tape or redundancy error condition in the previous transmission is detected.

```
                TIX       LOOP,1,1          Count total lines.
                OREADY                      Check last record.
                HTR                         Finish
A               OHEAD     B,0,1,52,1        Initalize succeeding
*                                           headings.
B               OREADY                      Alternate buffers
*                                           I1 and I2.
                OBCW      PAGE,0,1,8        Set up alternate
*                                           buffer with heading
*                                           image.
                OSCRIB    TAPEA2,0,8,1      Write heading.
                TRA       LOOP1             Continue main program.
PAGE            BCI       8,(48 characters) Page heading.
TAPEA2          OCT       1202
```

Since the N-value is 0 in the instruction OHEAD A,0,1,0,1; the first execution of the following OSCRIB instruction which attempts to transmit a Tape-BCD-Special (STHP) suppresses this transmission, causes a page to be ejected, resets the counter to 1, and transfers control to location A. At A, the instruction OHEAD B,0,1,52,1 specifies that whenever 52 lines have been printed (50 standard lines, 1 heading line, and 1 extra space line due to the remote eject), the execution of a subsequent OSCRIB instruction specifying Tape-BCD-special (STHP) should result in ejecting a page, resetting the counter to 1, and transferring control to B. Otherwise, the required line is printed and is added to the counter. At B, the OREADY instruction alternates the buffers, so as not to destroy the line image already prepared by the internal processing macro-instructions but not yet transmitted. The OBCW instruction will then set up the alternate buffer with the 48-character page heading line. The OSCRIB instruction which follows then initiates the transmission of this heading line to the tape. The buffer is switched back to the one already prepared for the next line image, and control is transferred back to the usual OSCRIB instruction, which will initiate transmission of the required line.

Ordinarily, in a program such as the one given above, some modal macro-instructions would be used ahead of the main loop. If other than single-spacing is desired, it could be accomplished by using OSPACE.

There is one additional rule for OHEAD. When OHEAD Y,T,C,N,K is executed with N = 0, the corresponding heading counters, either the "off-line" counter, the "on-line" counter, or both, depending on whether K = 1, K = 2, or K = 3, are set to 0. Otherwise, the counters are unchanged.

The normal mode of OHEAD (in both cases K = 1 and K = 2) is for the OSCRIB subroutine to ignore the heading feature, so that no OHEAD transfer of control and no change in either of the heading counters will occur on the execution of an OSCRIB instruction.

The execution of OHEAD Y, T, C, N, K with Y-c(T)= 0 will set the OHEAD modes to normal. However, according to the rule stated above, this may or may not result in setting the counters to zero, depending on whether N= 0 or N $\neq$ 0.

As an application of the above, suppose that a Tape-BCD-Special record (STHP) is being written using the heading feature, and that it is desired, at the same time, to write another Tape-BCD-Special record without using the heading feature, where the unit code for this second tape is in, say, location TAPEB3. Then the following instructions might be used:

Operation	Variable Field	Comments
OHEAD	0, 0, 0, 1, 1	Suppress counting.
OSCRIB	TAPEB3, 0, C, 1	Write second tape.
OHEAD	X, 0, C, N, 1	Restore desired heading mode, continue counting.

Both of the OHEAD modes (K= 1, K= 2) can be simultaneously set to normal by using either OHEAD 0 or OHEAD 0, 0, 0, 0, 3. Each can be set independently to normal by using OHEAD 0, 0, 0, 0, K where K= 1 or 2. In this case, the counters will be set to zero. Note, that the execution of the macro OUTRAN also causes both counters to be set to zero.

Both of the OHEAD modes (K= 1 and 2) can also be set to normal by using "OHEAD 0, 0, 0, 1, 3", or each can be set independently to normal by using "OHEAD 0, 0, 0, 1, K", where K= 1 or 2. However, as indicated above, in this case, the counters will not be changed.

OREDUN Y, T

This modal macro is used to establish a mode which is examined during the execution of an OSCRIB or OREADY instruction in the event that a persistent tape redundancy error is detected. Control is transferred to the location Y-c(T) whenever one of the following situations occurs:

1. When the redundancy indicator remains on after the Bad Spot Routine attempts to rewrite the previous record on a tape unit. This can occur during the execution of an OSCRIB or OREADY instruction, when the tape transmission initiated by the previous OSCRIB instruction is tested. In this case, c(MQ) = 0; the decrement of the accumulator will contain the current unit code and the address will contain the previous unit code (involving the erroneous record). Three cases can be distinguished when examining the accumulator; the first two cases occur during the execution of an OSCRIB instruction and the third case during the execution of an OREADY instruction:

a. If the current and previous tape units are the same, i.e., the address for the two units, excluding the 5th bit from the right (which distinguishes BCD from binary tape mode), are the same, then the current transmission has not been initiated.

b. If the current and previous tape units are different, then the current transmission has been initiated. Here, only the previous unit is necessarily a tape; the current unit can be a tape, printer, or punch.

c. If the decrement part of the accumulator (the field for the current unit code) is zero, the redundancy test must have occurred during the execution of an OREADY instruction. Therefore, no current transmission was required.

2. When attempting to execute an OSCRIB instruction where the length of the I-region is smaller than the number of words, W, required to be transmitted. In this case, c(MQ)= 1, and the accumulator will contain the current unit code in the address and (W - word length of buffer) in the decrement. The transmission specified by the current OSCRIB instruction will not have been initiated.

3. When attempting to execute an OSCRIB instruction which specifies an output unit which has not been assigned. The current transmission does not take place. In this case, c(MQ)= 2; the address of the accumulator will contain the previous unit code and the decrement will contain the illegal current unit code.

4. When attempting to execute an OSCRIB instruction which specifies an illegal N or unit name. For example, with N = 2 (implying binary tape output) and the unit specified as the on-line printer, OSCRIB could not correctly determine its proper function.

   The current transmission does not take place and control transfer with c(MQ)= 3 and the accumulator as for 3 above. *

   The normal mode associated with OREDUN is a transfer of control to the SOS program initiating a message to the dugging output unit which gives the contents of the AC and MQ, together with the location of the current OSCRIB macro.**

---

* This situation does not apply if the IB Monitor is used.

** With the IB Monitor, the normal mode of OREDUN is a recognizable stop.

OTPEND Y,T

If an end-of-tape mark was encountered during the transmission of a tape record whose transmission was initiated by an OSCRIB instruction, then an end of file is written on that tape unit during the execution of the next OSCRIB or OREADY instruction, and control is transferred to location Y-c(T).

The decrement of the accumulator will contain the current unit code and the address will contain the previous unit code (denoting the tape for which the end-of-tape mark was detected).

If the Redundancy indicator was also turned On, it is neither tested nor reset. The OTPEND mode has priority over the OREDUN mode. However, during the execution of the next OSCRIB instruction, the Redundancy indicator is reset before transmission is initiated.

Transmission specified by the current OSCRIB instruction may or may not have been initiated.

1. If the current and previous tape units are the same, the current transmission has <u>not</u> been initiated.

2. If the current and previous units are different (the previous unit was necessarily a tape, but the current unit may be a tape, printer, or punch), the current transmission <u>has</u> been initiated.

3. If the decrement of the AC (the field for the current unit code) is zero, the end-of-tape test must have occurred during the execution of an OREADY instruction. Therefore, no current transmission was required.

The normal mode associated with OTPEND is a transfer of control to the SOS program. A message is written on the debugging output unit giving the contents of the AC and the location of the current OSCRIB instruction. *

---

* With the IB Monitor, the normal mode for OTPEND is a recognizable stop within the system.

EXPANSIONS OF OUTRAN MACROS*

(1) OBCC [*] Y, T, C, N

```
STL     SYSOT1
TXL     SYSOT2, 0, 7
LDQ[*]  Y, T
PZE     C, 0, N
```

(2) OBCW [*] Y, T, C, N

```
STL     SYSOT1
TXL     SYSOT2, 0, 8
LDQ[*]  Y, T
PZE     C, 0, N
```

(3) OBIN [*] Y, T, C, N

```
STL     SYSOT1
TXL     SYSOT2, 0, 1
LDQ[*]  Y, T
PZE     C, 0, N
```

(4) OBLANK Y, T, C

```
STL     SYSOT1
TXL     SYSOT2, 0, 9
PZE     Y, T, C
```

(5) OCOLIN Y, T

```
STL     SYSOT1
TXL     SYSOT2, 0, 13
PZE     Y, T
```

(6) OCOLC [*] Y, T (Not available when using the IB Monitor)

```
STL     SYSOT1
TXL     SYSOT2, 0, 28
STO[*]  Y, T
```

* With the IB Monitor, SYSOT1 is replaced by 22_{10}, and SYSOT2 by 23_{10}. With both systems, PZE is, in some instances, replaced by HTR.

(7) OCOLR Y,T

```
STL     SYSOT1
TXL     SYSOT2,0,14
PZE     Y,T
```

(8) OEOR [*] Y,T,C

```
STL     SYSOT1
TXL     SYSOT2,0,19
PZE[*]  Y,T
PZE     C
```

(9) OFIX [*] Y,T,C,N,K,B (B parameter is not available when using the IB Monitor)

```
STL     SYSOT1
TXL     SYSOT2,0,4
LDQ[*]  Y,T
PZE     C,0,N
PZE     K,0,B
```

(10) OFLFIX [*] Y,T,C,N,K

```
STL     SYSOT1
TXL     SYSOT2,0,5
LDQ[*]  Y,T
PZE     C,0,N
PZE     K
```

(11) OFLOAT [*] Y,T,C,N,K

```
STL     SYSOT1
TXL     SYSOT2,0,3
LDQ[*]  Y,T
PZE     C,0,N
PZE     K
```

(12) OFXFLO [*] Y,T,C,N,K,B (Not available when using the IB Monitor)

```
STL     SYSOT1
TXL     SYSOT2,0,6
LDQ[*]  Y,T
PZE     C,0,N
PZE     K,0,B
```

(13) OHEAD [*] Y,T,C,N,K

```
STL     SYSOT1
TXL     SYSOT2,0,24
PZE [*] Y,T
PZE     C,0,N
PZE     K
```

(14) OIMAGE [*] Y,T,C

```
STL     SYSOT1
TXL     SYSOT2,0,22
PZE [*] Y,T
PZE     C
```

(15) OINT [*] Y,T,C,N

```
STL     SYSOT1
TXL     SYSOT2,0,0
LDQ [*] Y,T
PZE     C,0,N
```

(16) OMASK Y,T,C,N

```
STL     SYSOT1
TXL     SYSOT2,0,12
PZE     Y,T
PZE     C,0,N
```

(17) OOCTAL [*] Y,T,C,N

```
STL     SYSOT1
TXL     SYSOT2,0,2
LDQ [*] Y,T
PZE     C,0,N
```

(18) OOVPCH Y,T,C

```
STL     SYSOT1
TXL     SYSOT2,0,11
PZE     Y,T,C
```

(19) OPOINT Y,T

```
STL     SYSOT1
TXL     SYSOT2,0,15
PZE     Y,T
```

(20) OREADY

```
STL     SYSOT1
TXL     SYSOT2,0,26
```

(21) OREDUN [*] Y,T

```
STL     SYSOT1
TXL     SYSOT2,0,20
PZE[*]  Y,T
```

(22) ORPT R,I,J

```
STL     SYSOT1
TXL     SYSOT2,0,23
PZE     R-1
PZE     I,0,J
```

(23) OSCALE Y,T

```
STL     SYSOT1
TXL     SYSOT2,0,16
PZE     Y,T
```

(24) OSCRIB [*] Y,T,C,N

```
STL     SYSOT1
TXL     SYSOT2,0,25
LDQ[*]  Y,T
PZE     C,0,N
```

(25) OSPACE Y,T,C,N

```
STL     SYSOT1
TXL     SYSOT2,0,17
PZE     Y,T
PZE     C,0,N
```

(26) OSPILL [*] Y,T

```
STL     SYSOT1
TXL     SYSOT2,0,18
PZE [*] Y,T
```

(27) OTPEND [*] Y,T

```
STL     SYSOT1
TXL     SYSOT2,0,21
PZE [*] Y,T
```

(28) OUTRAN

```
STL     SYSOT1
TXL     SYSOT2,0,27
```

(29) OZERO Y,T

```
STL     SYSOT1
TXL     SYSOT2,0,10
PZE     Y,T
```

INPUT/OUTPUT SYSTEM

CHAPTER 3: INPUT EDITOR

The SHARE System Input Editor provides for conversion of data during Phase 1. The binary results of the conversion are written on a mediary output tape, in a form suitable for later reading by means of the buffering routines SYSRTK and SYSWTK (see Chapter 6).

The editor uses INTRAN to convert data, and the Buffering routines to write the data on the mediary output tape. This is done internally and the programmer need not be concerned with the use of INTRAN and the writing of the mediary output tape. He must, however, utilize the Buffering routines (see page 07.06.01) to read the data from the mediary input tape during Phase 2.

Because conversion is handled during Phase 1, no storage space is required during execution of the program in Phase 2. When Phase 2 is begun data will have been converted and is ready for immediate use when read by SYSRTK or SYSWTK.

Cards containing data to be converted by the Input Editor will generally have a class code punched in column one. This code refers to the type of conversion to be performed on this data. Class codes 0, 1, ..., 9 are reserved for installation conversion routines; +0, A, B, ..., I are available for use by the individual programmer.

The conversion corresponding to a programmer class is described by a format statement. This statement must be inserted at some point prior to the data to be converted.

A special programmer class, $, permits the use of data cards which cannot be punched with a class code in column one. The $ FORMAT statement, which describes the conversion required for these cards, must immediately precede them.

INPUT DATA PACKAGE

The data to be processed by the Input Editor is arranged as follows:

1. †	DATA	A, B, C	(See page 09.02.06.)
2.	Blank Card		
3.	Data to be converted, including control cards and data cards.		
4. †	ENDATA		
5.	Blank Card		

More than one data package may be processed; each package must be arranged as described above.

† 7-, 8-, and 9-punches in column one.

CONTROL CARDS

The DATA and ENDATA cards are SHARE Monitor control cards; hence, they must have the combination of 7-, 8-, and 9-punches punched in column one. The control cards discussed below are, on the other hand, a part of the data. Therefore, column one must be blank, or be punched with a "$".

A. ENDRCD

This card will cause a Logical End of Record flag to be written on the output tape. (An 11-punch in column one of an installation class data card will also cause an end of record to be written following the data on that card.) Each group of successive data cards must be terminated by an end of record. ENDRCD cards not immediately following data are ignored.

B. ENDGRP

This card will cause a Logical End of Group flag to be written. An ENDGRP card will always have the same effect. However, the programmer may have two or more in succession. (A card with an "*" in column one, and otherwise blank, will also cause an end of group to be written.)

C. ENDFILE

This card will cause a Logical End of File flag to be written if the output tape is not SYSMOT. If SYSMOT has been specified, the ENDFILE card is an illegal control card.

It should be noted that it is not possible to read past a Logical End of File flag using the buffering routines SYSRTK and SYSWTK.

D. ENDTAPE

This card will cause a Logical End of Tape flag to be written. The restrictions for ENDFILE also apply to ENDTAPE.

E. NOMORG

A programmer will ordinarily use SYSRTK for input of converted data. The calling sequence for SYSRTK specifies the starting location into which the data is to be read. This starting location may, at the programmer's option, be incremented by the nominal origin associated with the data. The nominal origin may be specified in two ways: through the use of a NOMORG card, or on the data card itself.

The format of the NOMORG card is

NOMORG A, B

where A is the nominal origin in decimal (a sign is optional)
B is described below.

1. NOMORG A

This card may appear at any point in the data package. Its effect is to set the nominal origin to "A". A succeeding ENDRCD or its equivalent returns the mode of operation to normal. A nominal origin specified in a data card has precisely the same effect as "NOMORG A".

2. NOMORG A, RECORD

This card may appear only at the beginning of a record. Its effect is to set the nominal origin to "A" at the beginning of that and each succeeding record. An ENDGRP or its equivalent returns the mode of operation to normal.

3. NOMORG A, GROUP

This card may appear only at the beginning of a group. Its effect is to set the nominal origin to "A" at the beginning of that and each following group. An ENDFILE returns the mode of operation to normal.

The mode of operation is always either normal, or that prescribed by the latest NOMORG card or its equivalent. The normal mode of operation is

NOMORG 0

F. FORMAT

This card is used to describe the conversion of programmer class data cards. The format of the card is

FORMAT A, (format specifications)

where A is one of the programmer class codes. FORMAT cards must occur somewhere prior to the data cards whose format they describe.

A programmer class may be redefined at any time. For a complete discussion of format statements, see below.

G. ETC

This card is a continuation card for format statements which cannot be contained on one card. As many ETC cards as necessary may be used. The format of the card is

ETC (Continuation of format statement)

THE $ CLASS

The $ class is a special programmer class which permits the use of data cards which do not have a class code punched in column one. The "$ mode" is entered upon encountering a FORMAT card with a "$" punched in column one and column 16. This "$ FORMAT card" must immediately precede the data to be converted. All subsequent control cards must also have a "$" punched in column one. The only restriction on the data cards is that they must not have a "$" in column one.

Data may be punched in column one; therefore, the "$" format specification must define the first data field as beginning with column one, rather than with column two (the first data column of ordinary data cards).

A special control card, $ STOP, is required to release the Input Editor from the $ mode.

Example of $ FORMAT card:

```
$   FORMAT   $, 5I6, A6/ ...
```

FORMAT STATEMENTS

The format statement describes the arrangement of data in a card, and prescribes the types of conversion to be performed between this data and its binary equivalents. The format statement is examined by the Input Editor, and a conversion routine is generated, to be executed when data referring to the format statement is encountered.

The format statement describes a card by indicating, for each field in the card, starting from the leftmost column available for data:

1. The type of field.

2. The width "w" of the field, in columns.

3. Other modifiers, as necessitated by the field type.

A. Basic Field Specifications

1. Iw

   The field is assumed to contain a signed or unsigned decimal integer, which is converted to binary.

2. Ow

   The field is assumed to contain a signed or unsigned octal integer, which is converted to binary.

3. Ew.d, Fw.d

   The field is assumed to contain a signed or unsigned floating point decimal number, "d" must be specified. Unless a decimal point occurs in the data, the decimal point is assumed to be "d" places to the left of the rightmost digit of the principal part. If an exponent part does not occur, it is assumed to be zero. The exponent part must begin with an "E", or a sign, or both. The number is converted to floating point binary.

4. Ew.dBb, Fw.dBb

   The field is assumed to contain a signed or unsigned floating point decimal number. "d" must be specified. Unless a decimal point occurs in the data, the decimal point is assumed to be d places to the left of the rightmost digit of the principal part. If an exponent part does not occur, it is assumed to be zero. The exponent part must begin with an "E", or a sign, or both. The number is converted to fixed point binary. "b" must be specified. Unless a B part occurs in the data, the binary number is assumed to have its binary point b places to the right of the leftmost bit.

5. Aw

   The field is assumed to contain Hollerith data. The data is converted to BCD, and left adjusted; zeros are inserted in unused character positions.

6. wH

   The "w" columns following the "H" are skipped over in the scan of the format statement. These columns are converted as though the specification were "Aw". Note that all of the w characters and the specification must be on one FORMAT or one ETC card.

7. Nw

The field is assumed to contain a signed or unsigned decimal integer. The integer is converted to binary, and becomes the nominal origin for the following data. A blank field is ignored.

8. NwO

The field is assumed to contain a signed or unsigned octal integer. The integer is converted to binary, and becomes the nominal origin for the following data. A blank field is ignored.

9. wX

The field is ignored.

B. Other Specifications

1. pP

A decimal scale of 10^p is applied to all E and F fields following this specification, so that

Internal number = External number x 10^p

The decimal scale continues to apply until another "pP" is encountered, or another format statement is referred to, or an end of record is written.

2. Sx. y. z

This specification may follow any of the specifications for numerical fields and indicates the occurrence of overpunching in the field. The digits x, y, and z refer to the principal part, exponent part, and B part of the field, respectively. x is the number of the column within the field (numbered 1, 2, 3, . . . , w from left to right) in which the overpunched sign of the principal part occurs. y is the number of the column within the E part (with the column containing the "E" numbered 0) in which the overpunched sign of the exponent part occurs. z is the number of the column within the B part (with column containing the "B" numbered 0) in which the overpunched sign of the B part occurs. If the "E" or "B" do not occur in the field, the y or z are ignored. The absence of an overpunch implies a "+". A field which is blank except for an overpunch in the principal part is considered to be a blank field.

Any of the values "x," "y," "z" may be omitted from the specification, but the decimal points which define them must be specified. The following are correct specifications:

Ew.dSx, OwSx, Fw.dBbSx..z, Ew.dBbS..z, NwOSx

C. General

The order of the specifications must be followed precisely. Blanks in the format statement are ignored, except in the "w" columns following a "wH" specification. The numeric modifiers called for in the specifications may not, in general, be omitted. They may in some cases be negative; the sign must then precede the number.

Example: E10.-2B-20

Each specification in the format statement should be terminated by one of the following characters:

,	comma
/	slash
)	right parenthesis

or by the end of the format statement. The comma may be omitted after the specifications "wH" and "pP".

A basic field specification, except "wH" and "wX", may be preceded by a positive non-zero number "n". The effect of this modifier is to repeat the field specification "n" times. If "n" is not specified, it is taken to be one.

Example: 3E12.5 = E12.5, E12.5, E12.5

Parentheses may be used to enclose a group of specifications which are to be repeated. The modifier "n" precedes the left parenthesis, may not be omitted, and indicates that the group of specifications is to be repeated "n" times.

Example: 2(I3, O5) = I3, O5, I3, O5

Parentheses may be nested to any degree.

Example: 2(I2, 3(F6.1, A2)) = I2, F6.1, A2, F6.1, A2, F6.1, A2, I2, ...

More than one data card may be described in a FORMAT statement. Specifications of succeeding cards are separated by a slash.

Example: 11H FIRST CARD/12H SECOND CARD/11H THIRD CARD

Repeated slashes cause the indicated data cards to be ignored.

Example: 11H FIRST CARD//11H THIRD CARD

Repetition of a data card specification is indicated by "n*" at the beginning of the specification.

Example: .../2*6E11.4B35/... = .../6E11.4B35/6E11.4B35/...

Parentheses which enclose groups of field specifications must not enclose a slash. In other words, field specifications may be repeated only within a card. A single pair of parentheses, with the modifier "n" omitted before the left parenthesis, may be used. Normally, when the end of the format statement is reached and there is still data to be converted, the format statement is reentered at its beginning. However, if an unmodified left parenthesis was encountered, the format statement will be reentered at that point. This pair of parentheses may enclose slashes, but the left parenthesis must occur at the beginning of a data card specification, and the right parenthesis must be coincident with the end of the format statement.

Example: A60/(6I10/6O10) = A60/6I10/6O10/6I10/6O10/...
A60/1(6I10/6O10) = A60/6I10/6O10/A60/6I10/...

An unmodified left parenthesis occurring at the beginning of a format statement is obviously redundant and is ignored. In this case only, more than one unmodified left parenthesis may occur in a format statement.

It is particularly important that every left parenthesis be matched by a corresponding right parenthesis.

D. Data Conversion

Leading and trailing blanks in a numeric field are ignored in conversion. An imbedded blank is an error. A blank numeric field is ignored, and no corresponding data is written on the output tape, but the nominal origin is incremented. If SYSRTK is used to read in data from cards containing blank numeric fields, the calling sequence must include the MZE prefix (which causes SYSRTK to increment the starting location by the nominal origin). Any word which would ordinarily have been filled with data from a numeric field will be undisturbed, if the field was blank. If the PZE prefix is used, and blank numeric fields occurred in the data cards, the results are unpredictable.

ERROR ANALYSIS

The Input Editor performs an extensive error analysis on both data and control cards. Upon encountering an error, an indicative comment is written on the system output unit. When appropriate, the card number, card column, and card image are also written.

A. Type 1 Errors

Type 1 errors are those which would lead to loss of control by the Input Editor. The job is deleted immediately upon encountering a type 1 error, and control is returned to the system.

B. Type 2 Errors

Type 2 errors are those which would definitely prevent successful execution of the program. In order to locate as many errors as possible, processing of data continues. However, the job is deleted.

C. Type 3 Errors

Type 3 errors are those which might prevent successful execution of the program. If GO was punched in the variable field of the DATA card, the job is not deleted.

In general, a type 3 error involves bad data, such as illegal characters, loss of significant bits, or floating point spill. A bad data flag is written at the end of the logical record in which the bad data occurred. When this flag is encountered by SYSRTK, the error return is made. Bad data is replaced by zero, except in the case of a floating point overflow, when it is replaced by -377777777777_8.

ERROR MESSAGES

Error messages are enclosed in quotation marks below.

A. Type 1 Errors

1. "Illegal or blank control card."
2. "Binary card not a control card (7-8-9)."
3. "Binary record on tape not column binary in origin."
4. "More than ten persistent redundancy checks on input tape."
5. "Attempt made to process past end of record." This error can occur only if an installation class conversion routine is erroneously programmed.
6. "Physical end of file encountered on input unit." This is a type 3 error if the end of file is encountered after the ENDATA card is read.
7. "IREDUN return from INTRAN." This error can occur only if an installation class conversion routine is erroneously programmed.

B. Type 2 Errors

1. Errors encountered in scanning a format statement.

   a. "Parenthesis trouble, first noticed here."
   b. "The preceding specification is incorrect."
   c. "The specified Hollerith field was not complete on this card."
   d. "Illegal character."
   e. "A non-zero number must precede this character."
   f. "Specification not properly terminated."
   g. "Too many columns specified for tape (or card) record."
   h. "Character is illegal in this context."
   i. "A number must precede this character."
   j. "The conversion routines compiled for this job exceed the capacity of storage."
   k. "Illegal FORMAT class specified."

2. Any spill produced in converting the field of a NOMORG card, or a nominal origin field in a data card.

3. An illegal character in the nominal origin field of a data card.

4. "Improper specification in a NOMORG."

5. "Illegal class code."

6. "Legal but undefined class code."

7. "ETC not preceded by FORMAT."

8. "Persistent redundancy check on input tape."

C. Type 3 Errors

1. Spills produced in the process of data conversion.

   a. "Numeric part of OCT integer exceeds 2**36."
   b. "Absolute value of DEC integer exceeds 2**35."
   c. "String of numeric digits exceeds 2**35."
   d. "E or B field exceeds 4 characters."
   e. "Indicated binary pt. causes loss of left bits."
   f. "Binary subfield extends into more than 1 word." This error cannot occur in a conversion defined by a format statement.
   g. "IBCC or IBCW with zero field width." This error cannot occur in a conversion defined by a format statement.

h. "Floating pt. underflow occurred in conversion."
i. "Floating pt. overflow occurred in conversion."

2. "Character is illegal in data field."

3. "Field width exceeds 31 columns. Compilation of format statement will continue." The field is taken to consist of the rightmost 31 columns.

4. "NOMORG RCD/GRP not at beginning of rcd/grp." The required end of record or end of group is written, and the NOMORG card is then processed.

5. "ENDGRP or ENDATA not preceded by ENDRCD." The required end of record is written.

6. "$STOP or FORMAT not preceded by ENDRCD." The required end of record is written.

7. "STOP card in improper sequence." The card is ignored.

8. "Control card (7-8-9) not an ENDATA card." The card is treated as an ENDATA card.

INPUT/OUTPUT SYSTEM

CHAPTER 4: OUTPUT EDITOR

The Output Editor provides the programmer with a method of converting data and writing it on the standard system output unit in the form desired. The editor operates during Phase 3, which means that virtually no storage space is sacrificed by the programmer during execution of his program in Phase 2.

The editor uses OUTRAN to perform the necessary conversion and writing of information. If the editor is used, both it and OUTRAN will be brought in during Phase 3.

The language of the Output Editor consists of eight macros which the programmer may use in his program to punch Hollerith and to print Hollerith information with full control of page headings, footings, spacing, conversion specifications, etc. The macros (XFORM, XPRINT, XPUNCH, XHEAD, XFOOT, XSPACE, XEJECT, XCOUNT) are expanded by the Compiler as calling sequences to an Output Editor Supervisor. The supervisor, which always occupies 100 words of core during Phase 2, interprets the calling sequences, and writes information onto the mediary output tape. During the third phase, this information is read by the Output Editor, and editing proceeds.

The format of output can be specified precisely through the Output Editor macros and format statements. The format statement will define exactly what work is to be performed on each piece of data, together with spacing and control. The format statements are compatible with those of the Input Editor. The basic field specifications permissible in a format statement are described on page 07.04.06.

MACRO-INSTRUCTIONS

A. XFORM N

Format statements are introduced by the macro

XFORM N

where N is the number of flag-words to follow the macro (and is, in fact, the number of format statements to be introduced by the macro).

Each of the N flag-words has the form

pfx L, T, M

where L, T denotes the beginning of the M words comprising a single format statement.

The prefix is PZE if L, T is the direct address, MZE if indirect.

The M BCD words comprising a format statement appear in core as if (or in fact) a direct result of

```
(L, T)      BCI     m1, A, XXXXXXXX
            BCI     m2, XXXXX...XXX
             .        .       .
             .        .       .
             .        .       .
            BCI     mn, XXXXX...XXX
```

where $m_1+m_2+\ldots+m_n = M$.

A is a BCD character identifying the format statement.

Unlike the Input Editor's format statement, the character used to identify the statement may be any BCD alphabetic character or $ (0 through 9 will again be reserved for installation standard format conversion routines).

The basic field specifications permissible in a format statement are described on page 07.04.06.

Example:

To introduce two format statements

1. A, 2I5
2. B, 3E14.6, 2(E9.4B5, F4.3B5)

we use

```
XFORM 2
PZE     ALP, 0, 2
PZE     BET, 0, 5
```

and somewhere in our program we have

```
ALP   BCI   2, A, 2I5
BET   BCI   5, B, 3E14.6, 2(E9.4B5, F4.3B5)
```

B. XPRINT A, N, C

Conversion and printing of a block of data according to a given format is specified by the macro

XPRINT A, N, C

where A is the identifying character of the format statement describing the desired conversion;

N is the number of flag-words which follow the macro;
C (which may be omitted) specifies that if fewer than C lines are left on the output page, the page should be ejected before the macro is executed.

The N flag-words which follow the macro have the same form as those following the XFORM macro, that is pfx L, T, M. The location (L, T) specifies the first of the M words of data to be converted and printed according to the format statement.

Example:

```
XPRINT    A, 2
PZE       DATA1, 0, 2
PZE       DATA2, 0, 4
```

specifies conversion and printing of two words of data which begin at location DATA1 and four words of data which begin at location DATA2 according to format statement A (which is presumed to have been introduced earlier in the program by an XFORM statement).

C. XPUNCH A, N

The XPUNCH macro is parallel to the XPRINT macro. XPUNCH A, N instructs the Output Editor to create output on the system's punched output unit. Output will be in Hollerith if on-line or, more normally, in column-binary-coded Hollerith if off-line. The data indicated by the N flags following the macro is converted and punched according to the format statement whose identification is A.

D. XHEAD A, N, C

This macro instructs the Output Editor to convert, according to format statement A, the data specified in the following N flag-words and use it as the page heading for each page of print until suppressed by an XHEAD macro with a blank variable field or by an XHEAD specifying a new heading.

When the XHEAD macro is executed, the page is ejected if necessary (i.e., if the line count $\neq 0$).

If C is non-zero, the number of lines per page is reset to C. (This is provided since it is probable that when a new heading is begun, the programmer is setting up a new page format, and might at this time want to reset the number of lines printed per page.)

Spacing after the heading is normally suppressed. Therefore, if overprinting is not desired, spacing must be provided. Spacing following the heading may be specified in the format statement by multiple slashes. It may also be specified by a separate macro: XSPACE C,H where C is the number of spaces to follow the heading before any subsequent printing. (See below.)

E. XFOOT A,N,C

This macro is parallel to XHEAD, and causes the information desired to be printed at the foot of the page (before page ejection). Spacing before footing may be specified in the format statement or by an appropriate XSPACE.

Note: Page ejection is the normal mode of operation; if it is suppressed, heading and/or footing are also suppressed.

F. XSPACE C,K

This macro is used to specify that there are to be C spaces of type K, where:

K = H to specify spacing between the heading and the body of a print page
= B to specify spacing between the blocks of printing in the body (a "block" of printing is defined to be the printed output resulting from one XPRINT macro)
= F to specify spacing between the body and the footing
= blank, for all three types of spacing.

XSPACE C,K is modal and will apply until another XSPACE of the same type is executed, wherein C may define a different number of spaces or may be blank or zero to discontinue the mode established.

Note: The spacing specified by an XSPACE macro supersedes spacing specifications in a format statement.

G. XEJECT C

The XEJECT macro is used to direct immediate page ejection or to reset the standard number of lines to be printed previous to page ejection.

C = 0 or blank if the page is to be ejected immediately,
= a decimal integer if the number of lines per page is to be reset to C.

Interpretation of XEJECT with nonzero C will cause the page to be ejected immediately unless the line count is zero (that is, unless the page has just been ejected).

An XEJECT with C = -1 may be used to suppress page ejection altogether.

H. XCOUNT N, S, I, R

This macro is used to define a counter (as for page numbering) or a series of hierarchy of counters.

XCOUNT N, S, I, R

where N is the identifying number of the counter (may be 1 through 7)
S is the value to which counter N is to be set initially
I is the amount by which counter N should be incremented (under control of format statements; see below).
R is the identifying number of another counter which, when incremented will cause the counter N to be reset to its initial value (R ≠ N).

The contents of a particular counter are incremented and displayed under the control of format statements. Suppose, for example, that the macros

```
XCOUNT    r,l,k
XCOUNT    n,s,i,r
```

had been executed previously. Suppose, further, that execution of an XPRINT, XHEAD, XFOOT, or XPUNCH macro is in process, and the subfield rCw+ is encountered in the format statement. The current value of counter r would be printed (or punched) as a decimal integer in the next w columns of the output record, the value of counter r would be incremented by the amount k, and finally, the value of counter n would be reset to s.

As described on page 07. 04. 07:

rCw specifies that the value of counter r is to be printed but not incremented,
rC+ specifies that the value of counter r is not to be printed but is to be incremented (and thenee, in our example, reset the value of counter n).

The example on page 07. 04. 10 will serve to clarify the use of the XCOUNT macro in conjunction with format statements.

It should be noted that the counters are limited in capacity (similar to the index register limit) to modulo 32, 768.

FORMAT STATEMENT SPECIFICATIONS

As noted above, the parameters of the output macros (XPRINT, XPUNCH, XHEAD, and XFOOT), in addition to specifying the words in core to be transmitted, also specify the identifying name of a format statement describing the type of conversion to be performed between the internal machine language and the external notation as well as the physical format of the output record.

The format specification describes the line to be printed or punched by giving the type of conversion and the width (w) for each field in the line.

BASIC FIELD SPECIFICATIONS

Type	Description
Iw	The following w characters of output are to be the result of binary to decimal integer conversion. If w exceeds 11, only the 11 rightmost characters will be significant, the rest will be blanks. The result will be right-adjusted.
Ow	The following w characters of output are to be the result of binary to octal conversion. If w exceeds 13, the excess will be blanks and the result right-adjusted. Leading zeros will be converted to blanks.
Ew.d.i	The next w characters of output are to be the result of converting the internal binary floating point number to decimal floating point, and the result is to have d places to the right of the decimal point and i places to the left.
Ew.d.iBb	Internal binary fixed point number to external decimal floating number. The binary number is assumed to have its binary point b places to the right of the leftmost bit. The result is to have d places to the right of the decimal and i places to the left.
Fw.d	The next w characters of output are to be the result of conversion from an internal binary floating point number to an external decimal fixed point number, where d is as in the E-type field.
Fw.dBb	Internal binary fixed point number to be converted to external decimal fixed point number, d and b are as in the E-type field.
Aw	The next $\frac{w+5}{6}$ data words contain BCD characters, and w of these characters are to be moved into the next w columns of the output image (are to appear as Hollerith characters in the next w output columns).
wH	The w BCD characters following the H are to be moved directly into the output image (are to appear as Hollerith characters in the next w output columns).

wX	The next w columns of output are to be blank.
Nw	The next w columns of output are to be the result of conversion from the internal signed binary integer to an external (signed) decimal integer which is to be used as the nominal origin. Actually, the Output Editor simply treats this specification as if it were Iw. It is left to the programmer to so place it that it will serve him as a nominal origin.
NwO	See above specification. This type results in an octal number to be used as nominal origin.

The method for indicating repetition of a field specification, of a group of specifications, and of the specification for an entire format line, as well as the separation of lines, indications of blank lines, etc. are the same as for the Input Editor, as are the means of indicating overpunching and scale factors. See page 07.03.07.

The format statements for the Input and Output Editors are compatible. Parameters required in the statement for one editor and not the other will be ignored by the second editor, e.g., a format statement, for output, containing Ew.d.i (see above, floating point binary to floating decimal) could also be used intact as a format statement for the Input Editor. In this latter case, the i parameter is ignored and specification is treated as Ew.d (floating decimal to floating binary). (See page 07.03.05.)

LINE SPACING

If it is desired that a number of lines be spaced before printing a given line, the format specification of that line may be preceded by a decimal integer from 1 to 9 indicating literally the number of spaces desired. This integer must preceed the first field specification of the line and must be separated from it by a comma. In the absence of such a number, the Output Editor assumes a value of 1.

Example: FORMAT A, .../2,I5/3,3O6,...

COUNTER CONTROL BY FORMAT STATEMENTS

The basic field specification cC (c is the identifying number of a counter and may be 1 through 7) is unique to the format statement of the Output Editor. This specification governs the treatment of the value of a counter, c, where this counter has been previously defined by an XCOUNT macro.

cCw	specifies conversion of the value of counter c to decimal, and printing of it in the next w columns of the output record.

cCw+	specifies conversion and printing of the value of counter c, and incrementing the counter by the amount indicated in the increment parameter of the XCOUNT macro defining counter c.
cC+	specifies incrementing of the value of counter c, but not printing.

EXPANSIONS OF OUTPUT EDITOR MACROS

```
(1) XCOUNT      N,S,I,R
        STL         SYSOED
        TXL         SYSOED-1,,8
        VFD         15/0,3/N
        VFD         3/N,15/S,3/R,15/I

(2) XEJECT      C
        STL         SYSOED
        TXL         SYSOED-1,,1
        VFD         15/0,15/C,H6/

(3) XFOOT       A,N,C
        STL         SYSOED
        TXL         SYSOED-1,,7
        VFD         15/N,15/C,H6/A

(4) XFORM       N
        STL         SYSOED
        TXL         SYSOED-1,,3
        VFD         15/N,15/0,H6/

(5) XHEAD       A,N,C
        STL         SYSOED
        TXL         SYSOED-1,,6
        VFD         15/N,15/C,H6/A

(6) XPRINT      A,N,C
        STL         SYSOED
        TXL         SYSOED-1,,4
        VFD         15/N,15/C,H6/A

(7) XPUNCH      A,N
        STL         SYSOED
        TXL         SYSOED-1,,5
        VFD         15/N,15/0,H6/A

(8) XSPACE      C,K
        STL         SYSOED
        TXL         SYSOED-1,,2
        VFD         15/0,15/C,H6/K
```

OUTPUT EDITOR EXAMPLE


```
                                                1*            LET US ASSUME THAT WE HAVE A PROBLEM IN WHICH WE ARE
                                                2*       ITERATING ON A 4-BY-7 MATRIX AND THAT WE WANT TO DISPLAY THE
                                                3*       VALUES OF THE ELEMENTS AFTER EACH ITERATION.    IN THE SAMPLE
                                                4*       PROGRAM WHICH FOLLOWS, EFFECTIVE USE IS MADE OF THE COUNTER
                                                5*       DEFINING ABILITY OF THE EDITOR.

                                                7              XFORM   2                      INTRODUCE AS FORMAT STATEMENTS
13631 -0 62500 0 00104                       GO     +0  STL    SYSOED
13634  0 00010 0 13703                        8      +3  PZE    FORM1,0,8                        1. THE 8 WORDS BEGINNING AT FORM1
13635  0 00013 0 13713                        9      +4  PZE    FORM2,0,11                       2. THE 11 WORDS BEGINNING AT FORM2.
                                               10             XCOUNT  1,1,1            DEFINE COUNTER FOR PAGE NUMBER.
13636 -0 62500 0 00104                              +5  STL    SYSOED
                                               11             XCOUNT  2,1,1            DEFINE COUNTER FOR ITERATION NUMBER.
13641 -0 62500 0 00104                              +8  STL    SYSOED
                                               12             XCOUNT  3,1,1,2          DEFINE COUNTER FOR ROW OF ELEMENT.
13644 -0 62500 0 00104                              +11 STL    SYSOED
                                               13*               THUS, THE ROW COUNTER WILL BE RESET TO 1 EACH TIME THE
                                               14*               ITERATION COUNTER IS INCREMENTED.
                                               15             XCOUNT  4,1,1,3          DEFINE COUNTER FOR COLUMN NUMBER OF ELEMEN
13647 -0 62500 0 00104                              +14 STL    SYSOED
                                               16*               THE COLUMN COUNTER WILL BE RESET TO 1 EACH TIME THE ROW
                                               17*               COUNTER IS INCREMENTED.
                                               18             XHEAD   A                      PRINT AS THE OUTPUT HEADING THE
13652 -0 62500 0 00104                              +17 STL    SYSOED
                                               19*               INFORMATION SPECIFIED BY FORMAT A.
                                               20             XSPACE  10,H
13655 -0 62500 0 00104                              +20 STL    SYSOED
                                               21             XSPACE  6,B
13660 -0 62500 0 00104                              +23 STL    SYSOED


13663  0 60000 0 13727                       24     BEGIN   STZ    MTRX                     INITIALIZE THE PHONY MATRIX.
13664  0 77400 1 00024                       25         +1  AXT    20,1                     SET FOR 20 ITERATIONS
13665  0 50000 0 13726                       26     AGAIN   CLA    L(ONE)
13666  0 40000 0 13727                       27         +1  ADD    MTRX
13667  0 60100 0 13727                       28         +2  STO    MTRX
13670  0 77400 2 00033                       29         +3  AXT    27,2                     SET FOR A SINGLE ITERATION.
13671  0 50000 2 13762                       30     ITER    CLA    MTRX+27,2
13672  0 40000 0 13726                       31         +1  ADD    L(ONE)
13673  0 60100 2 13763                       32         +2  STO    MTRX+28,2
13674  2 00001 2 13671                       33         +3  TIX    ITER,2,1                 TEST ALL ELEMENTS IN MATRIX TREATED.
                                               34             XPRINT  B,1,12           YES, PRINT ACCORDING TO FORMAT B THE
13675 -0 62500 0 00104                              +4  STL    SYSOED
13700  0 00034 0 13727                       35         +7  PZE    MTRX,0,28                28 DATA WORDS BEGINNING AT MTRX.
13701  2 00001 1 13665                       36         +8  TIX    AGAIN,1,1
13702  0 07400 4 00042                       37         +9  TSX    SYSTEM,4
13703  2 17305 0 06773                       38     FORM1   BCI    8,A,50X,16HMATRIX ITERATION,40X,5HPAGE 1C4+
13713  2 27305 0 06773                       39     FORM2   BCI    9,B,50X,10HITERATION 2C3+////4*2,7(4H  X(3C2,1H,4C2+,2H)   =
13724  1 33103 7 30267                       40         +9  BCI    2,=13,2X)3C+
13726  0 00000 0 00001                       41     L(ONE)  OCT    1
                 13727                       42     MTRX    BSS    28,0
                 13631                       43             END    GO
```

ITERATION 13

X(1, 1)= 13	X(1, 2)= 14	X(1, 3)= 15	X(1, 4)= 16	X(1, 5)= 17	X(1, 6)= 18	X(1, 7)= 19
X(2, 1)= 20	X(2, 2)= 21	X(2, 3)= 22	X(2, 4)= 23	X(2, 5)= 24	X(2, 6)= 25	X(2, 7)= 26
X(3, 1)= 27	X(3, 2)= 28	X(3, 3)= 29	X(3, 4)= 30	X(3, 5)= 31	X(3, 6)= 32	X(3, 7)= 33
X(4, 1)= 34	X(4, 2)= 35	X(4, 3)= 36	X(4, 4)= 37	X(4, 5)= 38	X(4, 6)= 39	X(4, 7)= 40

ITERATION 14

X(1, 1)= 14	X(1, 2)= 15	X(1, 3)= 16	X(1, 4)= 17	X(1, 5)= 18	X(1, 6)= 19	X(1, 7)= 20
X(2, 1)= 21	X(2, 2)= 22	X(2, 3)= 23	X(2, 4)= 24	X(2, 5)= 25	X(2, 6)= 26	X(2, 7)= 27
X(3, 1)= 28	X(3, 2)= 29	X(3, 3)= 30	X(3, 4)= 31	X(3, 5)= 32	X(3, 6)= 33	X(3, 7)= 34
X(4, 1)= 35	X(4, 2)= 36	X(4, 3)= 37	X(4, 4)= 38	X(4, 5)= 39	X(4, 6)= 40	X(4, 7)= 41

ITERATION 15

X(1, 1)= 15	X(1, 2)= 16	X(1, 3)= 17	X(1, 4)= 18	X(1, 5)= 19	X(1, 6)= 20	X(1, 7)= 21
X(2, 1)= 22	X(2, 2)= 23	X(2, 3)= 24	X(2, 4)= 25	X(2, 5)= 26	X(2, 6)= 27	X(2, 7)= 28
X(3, 1)= 29	X(3, 2)= 30	X(3, 3)= 31	X(3, 4)= 32	X(3, 5)= 33	X(3, 6)= 34	X(3, 7)= 35
X(4, 1)= 36	X(4, 2)= 37	X(4, 3)= 38	X(4, 4)= 39	X(4, 5)= 40	X(4, 6)= 41	X(4, 7)= 42

SAMPLE OUTPUT

CHAPTER 5: SHARE MONITOR TRANSMISSION MACROS

The Transmission macros cause generation of a set of calling sequences to an input/output dispatching program. These macros provide for efficient use of the parallel input/output facility of the 709, while relieving the programmer of nearly all I/O timing considerations.

Data is moved directly between working storage and the specified I/O unit using DSC commands provided by the programmer. Buffers are not employed; therefore, no arbitrary limits are placed on the length of records being read or written. The status of any transmission may be determined at any time, and priority may be assigned to specific operations.

If an object program, through the use of a Transmission macro, attempts to initiate transmission on a channel that is in operation, an entry for the macro is made in the "stack table" for the channel involved. Control is then returned to the object program. Subsequent execution of a Transmission macro causes entry to the Dispatching program. The status of each channel is then checked and waiting operations (i.e., those previously entered into the stack table) are initiated.

Since the Transmission macros are not designed to read or write information in the SHARE Monitor buffered format, it is not possible to operate on the same tape with both the Transmission macros and the Monitor buffering routines.

In the following description of the Transmission macros, each element in the designated variable field is expressible as a SCAT symbolic expression. Each refers to a core address, except "T" which is the index register associated with the address "Y", and "N" which is a spacing count for the non-transmitting tape-moving macros. The location counter symbol (*) may not be used, and a blank or zero field may have special meaning.

Each of the macros except DISP specifies an input/output operation referring to a table located at the effective address Y—c(T), or indirectly to the table if the macro is indirectly addressed. The first word of an I/O table specifies the symbolic tape unit and the mode. The mode is indicated by the prefix, which is PZE for binary mode and MZE for decimal mode. In the case of a READ or WRITE macro, the remainder of the table consists of the DSC commands which control the actual transmission. The I/O table is required by the Dispatching program throughout the execution of the Transmission macro. Therefore, the table must not be altered until the operation is complete.

The following example illustrates the format of an I/O table:

```
X       PZE     SYSAR1
        IOCP    A,,3
        IOST    B,,1000
```

This table, when referred to by a READ macro, specifies reading in the binary mode from the tape assigned as SYSAR1 (PZE in the prefix of X indicates binary mode).

The first three words of a record read will be transmitted to locations A through A+2, and the remainder of the record (up to a maximum of 1,000 words) will be transmitted to location B and following cells.

Each of the Transmission macros, except DISP, has an ERROR return as one of its variable field parameters. With the exception of the IN and OUT macros, the Dispatching program will transfer control to the location specified by the ERROR parameter if either of two conditions occurs:

1. the stack table for the channel involved is full
2. the tape specified in the first word of the I/O table has not been assigned.

Upon return to ERROR, the sign of the MQ will be plus in the first instance and minus in the second. The accumulator will contain the complement of the macro location. If ERROR is zero or blank, the return will be to SYSERR and index register 4 will contain the complement of the macro location.

```
READ            Y,T,ERROR
```

This macro causes a read operation to be entered in the stack table for the specified channel. When the Dispatching program executes the read operation, the I/O table at location Y—c(T) is used to specify the unit and control the reading.

The status of the read operation is tested by an IN macro referring to the same I/O table.

```
STEPR           Y,T,N,ERROR
```

This macro enters a "step records" operation in the proper stack table. The I/O table at Y—c(T) need consist only of the symbolic tape address. That tape is spaced forward N records or until end of file or end of tape is encountered. Such an indication is not counted as a record.

The status of the STEPR operation is tested by an IN macro.

STEPF Y, T, N, ERROR

This macro enters a "step files" operation in the stack table. It causes the specified symbolic tape to be spaced forward N files.

The status of the STEPF operation is tested by an IN macro.

WRITE Y, T, ERROR

This macro enters a write operation in the proper stack table, exactly as described for READ.

The status of the write operation is tested by an OUT macro.

WEOF Y, T, ERROR

This macro enters a write-end-of-file operation in the stack table of the Dispatching program. The one-word I/O table at location Y—c(T) symbolically specifies the tape on which an end-of-file mark is to be written.

The write-end-of-file operation status is checked by an OUT macro.

BACKR Y, T, N, ERROR

This macro causes the tape specified in the I/O table to be backspaced N records, or until beginning-of-tape is encountered. This operation, because of its inherent inefficiency, should be used sparingly, if at all. Its status is checked by an OUT macro.

BACKF Y, T, N, ERROR

This macro enters a backspace-file operation in the stack table. When executed, the operation backspaces the specified tape N files. If load point is reached by fewer than N backspaces, an error is noted for subsequent reporting by an OUT macro. If load point is reached on the nth backspace, the operation is complete. If not at load point, the tape is moved forward over the end-of-file mark preceding the desired file.

The status of a BACKF operation is tested by an OUT macro.

BACKT Y, T, ERROR

This macro causes the tape specified symbolically at location Y—c(T) to be rewound.

The status of the operation is tested by an OUT macro.

IN Y, T, NI, ERROR, EOF

The execution of this macro determines the status of the earliest READ, STEPR, or STEPF operation relating to the I/O table at location Y—c(T). If the operation has been successfully completed, a return is made to the next program step. In the case of a READ, the contents of the DSC registers for the channel will be placed in the accumulator upon completion of the operation.

If the operation has not been completed, a "Not In" return is made to location NI unless NI is zero or blank, in which case the Dispatcher retains control until the operation is complete. When the "Not In" return is made, the accumulator is set to indicate the state of the transmission:

1. READ

   a. If the operation has not yet been started, the accumulator is set to zero.
   b. If the operation is in progress, the contents of the DSC registers involved in the transmission will appear in the accumulator.

2. STEPR or STEPF

   a. If operation has not started, the decrement of the accumulator is set to zero.
   b. If operation is in progress, the decrement of the accumulator contains the number of files or records already stepped, and the address contains the number remaining to be stepped.

If the operation was a READ or STEPR which was successful but terminated on an end-of-file, a return is made to location EOF. The contents of the DSC registers or the stepping counts as they were when the end of file was encountered will appear in the accumulator. In the case of these two macros, if EOF is zero or blank, return is made to ERROR.

If the operation was a STEPF and the specified number of files were successfully spaced over, the return is to EOF or, if EOF is zero or blank, to the next program step.

If the operation produced an error, a return is made to location ERROR. The contents of the DSC registers or the stepping counts as they were at completion of transmission will appear in the accumulator, and the MQ register will contain error bits in the following pattern:

Bit	Nature of Error
S	No READ, STEPR, or STEPF macro has been given for the specified I/O table.
1	Operation completed but unsuccessful. This is normally caused by an unrecoverable redundancy.
2	Not used
3	End of file was encountered during operation.

If no error return is specified, return will be made to SYSERR.

The IN macro removes its corresponding entry from the stack table on all returns except to NI. The object program must execute IN macros in such a way that each READ, STEPR,or STEPF entry in the stack tables is removed in order to preclude saturation of the stack tables and resultant "Too Full" error returns when subsequent macros are executed.

OUT Y, T, NO, ERROR

The execution of the OUT macro interrogates the status of the earliest WRITE, WEOF, BACKR, BACKF,or BACKT operation relating to the I/O table at the specified location. On all except the NO return, the corresponding entry is removed from the stack table. The program must execute OUT macros in such a way as to insure that each entry will be removed from the stack tables.

If the operation has been successfully completed, a return is made to the next program step. The contents of the DSC registers or the backspacing counts as they were at the end of the transmission will appear in the accumulator.

If the operation has not been completed, a "Not Out" return is made to location NO; the accumulator will indicate the status of transmission in the same format as that described for the IN macro. If NO is zero or blank, the Dispatching program retains control and delays until the operation is complete.

If the operation was completed but produced an error, the return is to ERROR. The contents of the DSC registers of the stepping counts as they were at completion of transmission will appear in the accumulator. The MQ will contain the appropriate error bits as follows:

Bit	Nature of Error
S	No WRITE, WEOF, BACKR, BACKF or BACKT macro has been executed relating to the specified I/O table.

1	Operation completed but unsuccessful. This is normally caused by an unrecoverable redundancy.
2	A BACK operation attempted to backspace beyond the beginning of tape. (The decrement of the accumulator contains the number of records or files actually spaced.)
4	End of tape encountered by WRITE or WEOF. If no error return is specified, return will be made to SYSERR

RUSH Y,T,ERROR

The execution of this macro causes priority to be given to the unit specified in the contents of location Y—c(T); i.e., all previous operations which relate to this unit are advanced in the stack table of the designated channel so that these operations will be executed as soon as possible. RUSH does not interrupt a transmission already in progress, nor does it apply to operations on that channel which are initiated by macros following the RUSH macro. If more than one RUSH macro is executed, the order of priority is the order of execution of the RUSH macros.

DISP

The execution of this macro causes the Dispatching program to update the operation of all channels. The updating function is automatically performed each time any Transmission macros is executed. However, it may be necessary for certain types of programs to execute DISP occasionally to avoid undue idle time on channels.

EXPANSIONS OF SHARE MONITOR TRANSMISSION MACROS

(1) READ Y,T,ERROR

```
CAL          *
TXL          SYSTMA,,ERROR
PZE          Y,T
normal return
```

(2) STEPR Y,T,N,ERROR

```
CAL          *
TXL          SYSTMA+1,,ERROR
PZE          Y,T,N
normal return
```

(3) STEPF Y,T,N,ERROR

```
CAL          *
TXL          SYSTMA+2,,ERROR
PZE          Y,T,N
normal return
```

(4) WRITE Y,T,ERROR

```
CAL          *
TXL          SYSTMA+3,,ERROR
PZE          Y,T
normal return
```

(5) WEOF Y,T,ERROR

```
CAL          *
TXL          SYSTMA+4,,ERROR
PZE          Y,T
normal return
```

(6) BACKR Y,T,N,ERROR

```
CAL          *
TXL          SYSTMA+5,,ERROR
PZE          Y,T,N
normal return
```

(7) BACKF Y,T,N,ERROR

```
CAL             *
TXL             SYSTMA+6,,ERROR
PZE             Y,T,N
normal return
```

(8) BACKT Y,T,ERROR

```
CAL             *
TXL             SYSTMA+7,,ERROR
PZE             Y,T
normal return
```

(9) IN Y,T,NI,ERROR,EOF

```
CAL             *
TXL             SYSTMA+8,,ERROR
PZE             Y,T,NI
PZE             EOF
normal return
```

(10) OUT Y,T,NO,ERROR

```
CAL             *
TXL             SYSTMA+9,,ERROR
PZE             Y,T,NO
normal return
```

(11) RUSH Y,T,ERROR

```
CAL             *
TXL             SYSTMA+10,,ERROR
PZE             Y,T
normal return
```

(12) DISP

```
CAL             *
TXL             SYSTMA+11,0
normal return
```

CHAPTER 6: SHARE MONITOR BUFFERING ROUTINES

The Buffering routines provide a simplified means of reading, or writing, mediary tapes. These routines offer a programmer the advantages of buffered input/output without his having to be concerned with either timing considerations or the relation between buffer lengths and actual physical record sizes.

The reading and writing routines are controlled by special system flags which partition the data. Hence, only tapes produced by the Buffering routines can be read by the Buffering routines. Blocks of words can be formed into a logical record or a logical group. Termination of a block, or blocks, is controlled entirely by the programmer although normally a logical group will be composed of several logical records.

Core storage must be provided for the use by the Buffering routines. This storage will be divided into 256-word buffers. At no time will more than one buffer be attached to any I/O unit unless otherwise specified. (See Dispatching routines.)

Reading and writing of a given tape cannot be interspersed in a job without intervening rewinding of the tape. Provision has been made for backspacing a logical record only. Note that the logical properties of a tape written by the Buffering routines have no relation to the physical properties of that tape.

GENERAL PURPOSE ROUTINES

The following Buffering routines will perform all functions required for normal operation.

Routine	Name	Page
Add Buffer	SYSBFD	07.06.04
Write Logical Records	SYSNPT	07.06.05
Read Logical Records	SYSRTK	07.06.06
Backspace Logical Record	SYSBKS	07.06.08
Rewind Tape	SYSRWD	07.06.09

TITLE: ADD BUFFER: SYSBFD

PURPOSE: Prior to the use of the Buffering routines, the programmer must assign blocks of core for use as buffer areas. Each block will be divided into 256-word buffers. At least one buffer should be available for each tape used. Non-contiguous blocks may be furnished by multiple entries to SYSBFD.

CALLING SEQUENCE:

```
TSX     SYSBFD,4
PZE     Y,,N
error return
return
```

where Y is the location of the first word in the block
N is the number of words available in the block

NOTES: The error return will be made if the block specified is less than 256 words.

TITLE: WRITE LOGICAL RECORDS: SYSNPT

PURPOSE: This routine is used to write logical sets of information on the specified tape.

CALLING SEQUENCE:

```
        TSX     SYSNPT,4
        PZE     TAPE,,N
         .        .
         .        .      } N flags
         .        .
        return
```

where TAPE is the symbolic tape name
N is the number of flags which follow

EXAMPLE:

```
TSX   SYSNPT,4
PZE   SYSBU3,,7    WRITE THE NEXT 7 FLAGS AND THE
                       DATA SPECIFIED ON TAPE SYSBU3
IOCP  A,,10        10 WORDS BEGINNING WITH LOCATION A
TCH   SYSLER       END OF FIRST LOGICAL RECORD
IOCP  B,,5         5 WORDS BEGINNING WITH LOCATION B
IOCP  C,,8         8 WORDS BEGINNING WITH LOCATION C
TCH   SYSLER       END OF SECOND LOGICAL RECORD
TCH   SYSPEF,,0    END OF THIS LOGICAL GROUP
TCH   SYSPEF,,1    END OF DATA ON THIS TAPE
```

TITLE: READ LOGICAL RECORDS: SYSRTK

PURPOSE: This routine reads logical sets of information from a tape. Data will be read until terminated by a Logical Record, Logical Group, or Logical End flag. A return corresponding to the terminating flag will be made.

CALLING SEQUENCE:

```
TSX     SYSRTK,4
OP      TAPE,,Y
error return
logical end return
logical group return
logical record return
```

where OP is PZE to place data in storage, starting at Y

OP is MZE to place data in storage at the block flag address relative to Y, i.e., Y plus the address specified in the block flag when it was originally written on tape

TAPE is the symbolic tape name.

NOTES: If the logical end return is made, further attempts to read this tape will result in the same return.

The error return is used only with the Input Editor and signifies that the data now in storage was improperly converted by this editor.

Upon exit from SYSRTK the address field of the accumulator will contain Y plus the number of words in the logical record just read. This address can be used to preset the decrement of the parameter for the next entry to SYSRTK.

EXAMPLE: The following could be used to read the tape written by the SYSNPT example shown previously.

```
START   TSX   SYSRTK,4      READ LOGICAL RECORD
        PZE   SYSBU3,,Y     INTO Y FROM SYSBU3
        TSX   SYSERR,4      IMPROPER RETURN
        TRA   END           END RETURN
        TRA   EOG           GROUP RETURN
REC      .     .            RECORD RETURN-PROCESS
         .     .               LOGICAL RECORD IN Y
         .     .          ]  FIRST LOGICAL RECORD
         .     .          }     WILL GO INTO Y TO Y+9.
         .     .          ]  SECOND RECORD WILL GO
                             INTO Y TO Y+12.
        TRA   START         GO TO READ NEXT RECORD
EOG      .     .          }  PROCESS LOGICAL END
         .     .          ]     OF GROUP
END      .     .          }  PROCESS LOGICAL END
         .     .          ]
Y       BSS   20
```

TITLE: BACKSPACE LOGICAL RECORD: SYSBKS

PURPOSE: This routine backspaces to the previous logical end of record.

CALLING SEQUENCE:

```
TSX     SYSBKS,4
OP      TAPE
error return
return
```

where OP is PZE for an input tape
OP is MZE for an output tape
TAPE is the symbolic tape name.

NOTE: The error return is made if the beginning of tape is encountered.

The use of backspace does not allow both reading and writing of the same tape.

TITLE: REWIND TAPE: SYSRWD

PURPOSE: This routine rewinds a tape written, or read, in a buffered format. It must be used if a buffered tape is to be rewound as it causes the proper termination of processing.

CALLING SEQUENCE:

```
TSX     SYSRWD,4
PZE     TAPE
return
```

where TAPE is the symbolic tape name.

NOTE: Tapes being written must be rewound before being read, and vice versa.

BUFFERING ROUTINE FLAGS

A. General Purpose Flags

The following are the commonly used flags which will furnish all the control necessary for normal jobs.

1. Block Flag

```
IOCP  Y,,N
```

where Y is the first location of N consecutive words to be written on tape.

Any number of Block flags may be used to form the contents of a single logical section. The Block flag is written on tape immediately followed by the information referred to.

2. Logical End of Record Flag

```
TCH   SYSLER
```

This flag is written on tape in indicate the end of a logical record. A logical record consists of one or more Block flags together with the data to which the flags refer.

3. Logical End of Group Flag

```
TCH   SYSPEF,,0
```

This flag is normally used to separate groups of logical records.

4. Logical End Flag

```
TCH   SYSPEF,,1
```

An end-of-file mark is written on the tape followed by the Logical End flag. It is not possible to read past the Logical End flag.

B. Special Purpose Flags

The following flags are not necessary for normal use of the buffering routines but may be used for special purposes.

1. Nominal Origin Flag

```
IOCPN   Z
```

where Z is an address specified by the program.

Buffer read routines may be made to read data into locations dependent on addresses specified in block flags written on the tape. (See the MZE parameter in SYSRTK.) The Nominal Origin flag causes the write routines to process Block flags in a special way. The origin for output data specified in the Block flags which follow a Nominal Origin flag, is used only to designate the initial locations for writing the data on tape. The flags themselves are altered prior to being written on tape. The address field of the first Block flag following the Nominal Origin flag is replaced by the address contained in the Nominal Origin flag. The addresses of subsequent Block flags are replaced by the address specified in the Nominal Origin flag plus the word counts of the preceding Block flags. Any flag other than a Block flag terminates this method of output processing. The Nominal Origin flag is never written on tape. The use of this flag furnishes a means of relative addressing.

Example;

The logical record specified in a SYSNPT calling sequence by:

```
IOCPN   Z
IOCP    D,,4
IOCP    E,,11
TCH     SYSLER
```

would appear on tape as:

```
IOCP    Z,,4        }
                    } 4 words from location D
IOCP    Z+4,,11     }
                    } 11 words from location E
TCH     SYSLER      }
```

If this record were read by SYSRTK using the MZE parameter (i.e., MZE TAPE,,Y) then 15 words would be read into location Y+Z.

2. Immovable Block flag

```
IOSP    Y,,N
```

where Y is the location of N words to be written on tape.

This flag is like the Block flag except that when being read by SYSRTK, the data is put back into the locations beginning at Y no matter what mode of input control is being used by SYSRTK.

3. Symbol flag

IOSPN Y,,N

where Y is the location of N words to be written on tape.

This flag is normally used for record or tape labeling. The SYSRTK routine will simply pass over this flag and its data, and will interpret the next flag. The reading routine SYSWTK, permits the analysis of this flag and its data.

4. Sequence flag

IOSPN N,,0

where N is the sequence number assigned by the program, and is part of the flag itself.

This flag may be used for tape or record numbering.

SYSRTK will simply pass over the Sequence flag, and will interpret the next flag. The reading routine SYSRTK, permits analysis of the flag.

SPECIAL PURPOSE ROUTINES

The following routines may be used for special purposes:

TITLE: READ WORD: SYSWTK

PURPOSE: This routine is used when it is necessary to process each word or flag separately. It is the only reading routine which allows interpretation of symbol or sequence flags.

One of three returns informs a program that the routine is positioned at the beginning of a block of data; or that a word of data has been transmitted to the register specified; or that a terminating flag has been found.

CALLING SEQUENCE:

```
TSX     SYSWTK,4
OP      TAPE,TAG
PZE     A,,B
data return
```

where OP is CLA, CAL, LDQ, or LDI specifying the register into which a word of data is to be placed.

TAPE is the symbolic tape name

TAG is the index register which is available for us by this buffering routine. The index register must not be altered by the program until a terminating flag has been read.

A is the return to be made when a Block, Symbol, Sequence, or Immovable Block flag is encountered. The flag will be in the logical accumulator regardless of the specified data register. For this type of flag, the SYSWTK calling sequence is replaced before the A return is made. The next entry to SYSWTK should normally be to the former location of the PZE parameter to pick up the first word of data. The first data return will then be made with the first data word in the specified register. The remaining data words are picked up by entries to the former location of the TSX SYSWTK,4 instruction. The calling sequence is automatically restored when all the words in the block have been read.

If the word count is zero, as will always be the case with a Sequence flag, and may be the case with blindly formed Block flags, the first entry to SYSWTK following the A return must be to the former location of the OP parameter rather than the PZE parameter. SYSWTK will then make either the A or B return depending on what flag follows the one just read.

B is the return made when a Logical Record, Logical Group, or Logical End flag is encountered. The flag will be in the logical accumulator regardless of the specified data register. The next flag will be obtained if the program returns to the location of the TSX SYSWTK, 4.

The data return permits the program to process the word of data which has just been loaded into the specified data register. The next word or flag will be obtained if the program loops back to the location of the TSX SYSWTK, 4.

EXAMPLE:

```
Location    Operation    Address          Comments

WORD        TSX          SYSWTK, 4
SEQ         CLA          SYSBU3, 2
NUM         PZE          DAT, NODAT
              .            .
*             .            .              PROCESS DATA FOLLOWING
              .            .                BLOCK OR SYMBOL FLAG
              .            .
              .            .
            TRA          WORD             GO TO PICK UP NEXT DATA WORD
              .            .
              .            .
              .            .
DAT           .            .              PROCESS BLOCK SEQUENCE
              .            .                OR SYMBOL FLAG
            TRA          NUM              GO TO GET FIRST DATA WORD
              .            .
              .            .
              .            .
```

```
                TRA         SEQ         SEQUENCE FLAG OR BLOCK
*                                         FLAG WITH ZERO COUNT
                 .           .
                 .           .
                 .           .
NODAT                                     PROCESS LOGICAL END OF
*                                           RECORD, LOGICAL END OF
*                                           GROUP, LOGICAL END OR
*                                           SEQUENCE FLAGS
                 .           .
                 .           .
                 .           .
                TRA         WORD
```

TITLE: WRITE A BLOCK FLAG: SYSBLK

PURPOSE: This routine and the two which follow, SYSINF and SYSWHT, are used to write flags and data one word at a time. These routines are used principally in programs where non-contiguous data must be processed and written as a single logical section.

SYSBLK writes a single flag and initializes the routine SYSINF, which writes a word of data. SYSBLK must be used to write a Block, Immovable Block, Nominal Origin, or Symbol flag.

CALLING SEQUENCE:

```
TSX     SYSBLK,4
OP      TAPE,T
```

where OP is STO, SLW, STQ or STI specifying the register in which a word of data will be placed by the object program.

TAPE is the symbolic tape name

T is the index register which is available to the routine SYSINF, which places the data in the buffer. The index register must not be altered until the word count in the flag being written by SYSBLK has been reduced to zero by the appropriate number of entries to SYSINF

NOTE: Upon entry to the routine, the flag to be written must be in the logical accumulator.

If the Block, Immovable Block, or Symbol flag has a word count of zero, the SYSWHT routine should be used to output this flag.

TITLE: WRITE A DATA WORD: SYSINF

PURPOSE: This routine writes a word as a part of the block of data specified in the flag which was written by the SYSBLK routine. SYSINF closes out the internal processing initialized by SYSBLK when the number of words which have been stored is equal to the word count in the flag written by SYSBLK.

CALLING SEQUENCE:

```
TSX     SYSINF,4
```

NOTES: On entry, a word of data must be in the register specified in the OP parameter of the SYSBLK routine. The index register specified by SYSBLK will be initially 0 and will be decremented by 1 until the negative of the word count is reached.

TITLE: WRITE A TERMINATING OR NON-DATA FLAG: SYSWHT

PURPOSE: This routine writes a Logical End of Record, Logical End of Group, Logical End, or Sequence flag.

CALLING SEQUENCE:

```
TSX     SYSWHT,4
PZE     TAPE
```

where TAPE is the symbolic tape name

NOTES: Upon entry to the routine the terminating flag must be in the logical accumulator. If the terminating flag is a Logical End flag, the contents of the buffer are written on tape, followed by an end-of-file mark and the Logical End flag. In reading, it is not possible to read past a Logical End flag.

EXAMPLE:

To output a tape of the following form with the data words converted:

```
IOCP    J,,10
TCH     SYSLER
IOCP    K,,5
IOCP    L,,8
TCH     SYSLER
TCH     SYSPEF,,0
TCH     SYSPEF,,1
```

Location	Operation	Address	Comments
START	AXT	J,4	PRESET TO OUTPUT BLOCK J,,10
	SXA	RA,4	
	AXT	10,4	
	SXD	RA,4	
	CAL	RA	
	TSX	C,4	GO TO OUTPUT BLOCK J,,10
	CAL	SA	OUTPUT LOGICAL RECORD
	TSX	D,4	
	AXT	K,4	PRESET TO OUTPUT BLOCK K,,5
	SXA	RA,4	
	AXT	5,4	
	SXD	RA,4	
	CAL	RA	
	TSX	C,4	GO TO OUTPUT BLOCK K,,5

```
                AXT     L,4         PRESET TO OUTPUT BLOCK L,,8
                SXA     RA,4
                AXT     8,4
                SXD     RA,4
                CAL     RA
                TSX     C,4         GO TO OUTPUT BLOCK L,,8
                CAL     SA          LOGICAL RECORD
                TSX     D,4         GO TO OUTPUT LOGICAL RECORD
                CAL     TA          LOGICAL GROUP
                TSX     D,4         GO TO OUTPUT LOGICAL GROUP
                CAL     VA          LOGICAL END
                TSX     D,4         GO TO OUTPUT LOGICAL END
REST             .       .
                 .       .          REMAINDER OF PROGRAM
                 .       .

C               BEGIN   1,5
                STA     A
                STD     B
                TSX     SYSBLK,4
                STQ     SYSAR2,1
A               LDQ     **,1
                CAQ     X           CONVERT DATA
                TSX     SYSINF,4
B               TXH     A,1,**
                RETURN  C
                 .       .
                 .       .
                 .       .
D               BEGIN   1,4
                TSX     SYSWHT,4
                PZE     SYSAR2
                RETURN  D
                 .       .
                 .       .
                 .       .
RA              IOCP    **,,**      BLOCK
SA              TCH     SYSLER      LOGICAL RECORD
TA              TCH     SYSPEF,,0   LOGICAL GROUP
VA              TCH     SYSPEF,,1   LOGICAL END
```

DISPATCHING ROUTINES

The group of routines described below are Buffer Dispatching routines. They permit a program to exert some control over the functioning of the read or write Buffering routines.

A writing routine functions at maximum efficiency if two buffers are provided, one to be written on the tape as the other is filled from core storage. The program should request that the Dispatcher attach two buffers to an output tape. A number in excess of two will be ignored.

Reading may be made to keep any number of buffers ahead. Time may always be saved by having two buffers, rather than one, attached to an input tape. If logical sections are known to be longer than the buffer size, then it is to the programmer's advantage to supply space for more than two buffers and to use the dispatching routines to keep at least one logical section immediately available to the object program. That is, the program should request that as many buffers be attached as will accommodate two logical sections.

The Dispatching routines are:

TITLE: DISPATCHING INITIATION: SYSDPI

PURPOSE: This routine gives the Dispatcher the location of the programmer dispatching lists.

CALLING SEQUENCE:

```
TSX     SYSDPI,4
PFX     L,,N
return
```

where PFX = PZE to add to dispatching list
= MZE to delete a dispatching list
L is the location of a list
N is the number of entries in the list not including the terminating word of zero.

The dispatching list is of the form

```
L       PFX             TAPE,,M
          .               .
          .               .
L+N-1   PFX             TAPE,,M_N
L+N     PZE
```

where PFX = PZE for an input unit
= MZE for an output unit
TAPE is the symbolic tape name for which dispatching is to be performed.
M is the number of buffers to keep ahead.

A terminal word of zeros is always required.

NOTES: More than one dispatching list may be provided or deleted by multiple entries to SYSDPI. Dispatching will be in the same sequence as the order in the lists furnished.

TITLE: NORMAL DISPATCHING: SYSDIS

PURPOSE: Dispatching will occur whenever entry is made to a Buffering I/O routine. The Dispatcher scans the dispatching lists made available to it, initiates I/O operations and attaches buffers, if possible to I/O units based on the number requested in the dispatching lists. If insufficient buffers are available to meet dispatcher list requirements as many as available will be attached with preference given to those at the top of the lists.

In situations where large amounts of computation are accomplished without I/O, some increase in speed may be obtained by occasional entries to the Dispatcher to initiate I/O on quiet channels.

CALLING SEQUENCE:

```
TSX     SYSDIS,4
return
```

TITLE: DISPATCHER SUPPRESSION: SYSDPS

PURPOSE: To prevent further dispatching on a particular I/O unit. This routine will zero the buffer count on all dispatcher list entries referring to the specified I/O unit. This essentially releases those buffers for use on other I/O units.

CALLING SEQUENCE:

```
TSX     SYSDPS,4
PZE     TAPE
```

where TAPE is the symbolic name of the I/O unit.

CHAPTER 7: IB MONITOR TRANSMISSION MACROS

The IB Monitor Transmission macros provide a language by which the programmer can conveniently use the parallel operations of input, output, and computing. Transmission requests are stacked in tables corresponding to the channel specified. These requests are subsequently carried out by the Transmission macro routines. The status of a given transmission can be tested at any point in the program. The checking of all input/output indicators is handled automatically.

There are twelve IB Monitor Transmission macros which generate calling sequences for the Transmission macro routines.

Each element in the variable field of a Transmission macro can be a symbolic expression. Each refers to a core address except T (which is a tag for the address Y) and N (which is a count for backspacing).

The READ and WRITE macros specify transmission according to an I/O Table located at the effective address Y-c(T). The first word of the table specifies the mode (binary or BCD), channel, and unit in absolute. The first word is followed by the I/O control words to be executed for the transmission.

Example:

```
A71O        OCT         1227
            IOCP        CKSUM,,1
            IOCD        DATA,,100
```

This sequence specifies that tape 7, channel A, is to be read or written in the binary mode according to the two control words in A71O+1, and A71O+2.

The first word could also be written as:

```
A71O        PZE         663
              or
A71O        DEC         663
```

IN and OUT are used to test the status of the transmission specified for the I/O table at location Y—c(T). RUSH establishes a priority for all previous transmission macros for the unit specified by the word in location Y—c(T). BACK, REWIND, and WRITEF will backspace, rewind, and write end-of-file in the unit specified in location Y—c(T).

Whenever a READ, WRITE, WRITEF, BACK, or REWIND macro is executed, the corresponding operation may not be started immediately if the channel is performing a previous operation. In this case, the operation is automatically placed in the "stack table" and computation proceeds.

The stack table is an ordered list of operations for each channel which is kept and automatically updated by the Transmission routines. Entries are made in this table whenever a READ, WRITE, WRITEF, BACK, or REWIND is executed. IN and OUT interrogate the stack table as to the status of READ and WRITE operations, respectively. They determine whether the operations have been completed or not, and whether check indicators have been turned on.

In order to minimize idle time for a channel, it may be necessary to execute the macro DISP periodically during problem computation. This macro simply transfers to the Transmission routines and begins the next operation in the stack table for an idle channel. Dispatching also occurs automatically whenever any Transmission macro is executed.

OPERATION OF THE TRANSMISSION ROUTINES

Two cells in the IB Monitor communication region are used by the Transmission routines. These are:

$$\text{SYSTM1} = (61)_8 = (49)_{10}$$

$$\text{SYSTM2} = (62)_8 = (50)_{10}$$

The Transmission routines do not depend on any fixed location assignment for SYSTM1 and SYSTM2.

The IB Monitor automatically reads the Transmission routines from the System tape at the first attempt by an object program to execute a Transmission macro. The Transmission routines require approximately 700 locations.

At object time, core storage is allocated as follows:

$(0\text{-}2999)_{10}$	IB Monitor and Debugging System
$(3000\text{-}8999)_{10}$	INTRAN
$(9000\text{-}10499)_{10}$	Data Sentence Program
$(10500\text{-}14499)_{10}$	OUTRAN
$(14500\text{-}15200)_{10}$	Transmission routines

The last 256 words of memory will be used as a buffer by the Debugging System if the TAPE macro is executed without previously executing the BUFFER macro.

When the monitor reads in the Transmission routines from the system tape, it initializes routines via the calling sequence

```
TSX       **,4
PZE       SYSTM1,0,SYSTM2
return
```

where ** is obtained from the TCD card of the Transmission routines. The routines then initialize themselves to the value of SYSTM1 and place a transfer to the routines into SYSTM2. Control is then returned to the monitor.

When an object program causes loading of the Transmission routines, by execution of a Transmission macro, and later reads over the storage for the routines $(14500\text{-}15200)_{10}$, the object program must reload the routines by

```
TSX       SYSTM3,4
return
```

where SYSTM3 = $(63)_8 = (51)_{10}$.

The monitor will reload and initialize the Transmission routines.

The Transmission routines provide for 10 entries in each channel stack table.

TRANSMISSION MACROS

A. READ Y, T, ERROR

This macro causes a Read transmission to begin according to the mode, unit, and control words specified in the I/O table at the effective address Y-c(T). If the channel is active, or if other previously specified operations are waiting for the channel, this Read is stacked for subsequent execution.

If the Read is either executed or stacked, a return is made to the next program step without delay. If the Read is to be stacked and there is insufficient room in the stack table, the return is to ERROR and the entry is not stacked. An ERROR address of zero will cause a transfer to location SYSTEM when the stack table is full.

B. WRITE Y, T, ERROR

This macro causes a Write transmission to begin according to the mode, unit, and control words specified in the I/O table at the effective address Y-c(T). If the channel is active, or if other previously specified operations are waiting for the channel, the Write is stacked for subsequent execution.

If the Write is either executed or stacked, a return is made to the next program step without delay. If the Write is to be stacked and there is insufficient room in the stack table, the return is to ERROR and the entry is not stacked. An ERROR address of zero will cause a transfer to location SYSTEM when the stack table is full.

C. WRITEF Y, T, ERROR

This macro will cause an end-of-file to be written on the tape unit specified by the word in location Y-c(T). If the channel is active or if other previously specified operations are waiting for the channel, the WRITEF is stacked for subsequent execution.

If the WRITEF is either executed or stacked, a return is made to the next program step without delay. If the WRITEF is to be stacked and there is insufficient room in the stack table, the return is to ERROR and the entry is not stacked. An ERROR address of zero will cause a transfer to location SYSTEM when the stack table is full. WRITEF is interrogated by OUT in the same way as WRITE.

D. REWIND Y, T, ERROR

This macro will cause the unit specified in the word in location Y-c(T) to be rewound. If the channel is active, or if other operations are waiting for the channel, the REWIND is stacked for subsequent execution.

If the REWIND is either executed or stacked, a return is made to the next program step without delay. If the REWIND is to be stacked and there is insufficient room in the stack table, the return is to ERROR and the entry is not stacked. An ERROR address of zero will cause a transfer to location SYSTEM when the stack table is full.

REWIND should not be interrogated. Its entry is removed from the stack table as soon as the operation is completed on the channel.

E. BACK Y, T, N, ERROR

This macro will cause the tape unit specified in the word in location Y-c(T) to be backspaced N records. An end-of-file gap will be considered as a record. If N is zero, a backspace file will be executed. If the channel is active, or if other previously specified operations are waiting for the channel, the BACK is stacked for subsequent execution.

If the BACK is either executed or stacked, a return is made to the next program step without delay. If the BACK is to be stacked, and there is insufficient room in the stack table, then the return is to the ERROR address and the entry is not stacked. An ERROR return of zero will cause a transfer to location SYSTEM when the stack table is full.

BACK should be interrogated by the OUT macro to determine a beginning of tape error condition.

F. RUSH Y, T, ERROR

This macro will cause the unit specified by the word in location Y-c(T) to be given channel priority. Hence, all Transmission macros relating to a specific unit are placed (in their relative positions to each other) ahead of the macros referring to other units on the same channel.

RUSH will not interrupt a transmission already on the channel. If more than one RUSH macro is executed, the order of priority is the order of execution of the RUSH macros. A RUSH macro establishes priority between units rather than between macros for the same units.

After the execution of this macro, a return is made to the next program step. If there are no transmissions in the stack table relating to the specified unit, a return to ERROR is made. (Card punch and printer operations will be handled together; all other units are handled separately.)

G. IN Y, T, NI, ERROR, EOF

The status of the stack table of the READ operation relating to the I/O table at location Y-c(T) is interrogated. The Dispatcher searches the appropriate channel table for the first READ with the same Y, T entry and tests its status. If Y and T are zero, the Dispatcher will search for the last READ in any channel stack table and test its status.

If reading has been completed, and no check indicators have been turned on, a return is made to the next program step. When the return is made, the accumulator will contain the results of a Store Channel instruction executed at the end of the transmission.

If the reading has not been completed, a "Not In" return is made to location NI, and the accumulator will be set to indicate the status of transmission. The accumulator is set to zero if the macro has not been started. If the macro is in progress, the accumulator will contain the results of a Store Channel instruction executed at this time.

If the reading has not been completed, but an end of file was encountered, a return is made to location EOF. The accumulator will contain the results of a Store Channel instruction executed at this time.

If the operation has been completed but an error has been detected, a return is made to location ERROR. The accumulator will contain the results of a Store Channel instruction executed at the end of the transmission. The MQ will contain error indicator bits as follows:

Bits	Contents	Meaning
S	1	No READ macro has been given for the I/O table at location Y—c(T). No reading has taken place. A previous IN may have removed the macro the programmer is searching for.
1	1	Reading was completed, but a redundancy check occurred.
2	1	Reading was completed, but the beginning-of-tape indicator was on at the start of transmission.
3	1	The end-of-file indicator was turned on by the operation. Note that the end-of-file return to location EOF is made only if no error condition was encountered.

The IN macro removes its corresponding READ entry from the stack table on all returns except the NI return.

H. OUT Y, T, NO, ERROR, ETT

This macro permits the testing of the macros WRITE, WRITEF, and BACK relating to the I/O table at location Y-c(T). The Dispatcher searches the appropriate channel table for the first WRITE with the same Y, T entry and determines its status. If Y and T are zero, the Dispatcher searches for the last WRITE and tests its status.

If writing has been completed and no check indicators have been turned on, a return is made to the next program step. The accumulator will contain the results of a Store Channel instruction executed at the end of the transmission.

If the writing has not been completed, a "Not Out" return is made to location NO and the accumulator is set to indicate the status of transmission. The accumulator is set to zero if the macro has not been started. If the macro is in progress, the accumulator contains the results of a Store Channel executed at this time.

If the writing has been completed, but end-of-tape was encountered, a return is made to location ETT. The accumulator will contain the results of a Store Channel executed at the end of the transmission.

If the operation has been completed but an error has been detected, a return is made to the location ERROR. The accumulator will contain the results of a Store Channel instruction executed at the end of the transmission. The MQ will contain error indicator bits as follows:

Bits	Contents	Meaning
S	1	No WRITE macro has been given for the I/O table at location Y-c(T). No writing has taken place. A previous OUT may have removed the macro the programmer is searching for.
1	1	Writing completed but with a redundancy check.
2	1	The Beginning-of-Tape indicator was on when writing was started. Note that when backspacing, it is possible for records or long files to turn on the indicator after it has been checked. The indicator must then be checked by the object program.
3	1	Writing has been executed and has turned on the End-of-Tape indicator.

The OUT macro removes the corresponding WRITE from the stack table except upon the NO return.

I. DISP

This macro will cause a transfer to the Transmission routines which will initiate the next operation in the stack table for any idle channel. A return is made to the next program step.

Though this operation is performed automatically for all channels each time a Transmission macro is executed, it may be necessary in certain programs to execute DISP periodically to prevent excess idle time on channels.

J. CLEAR

This macro is used to initialize the stack tables. It will remove all entries from all tables regardless of status. Return is always made to the next program step.

K. CUT

This macro removes from the tables all entries that have not been completed, those that have not been initiated, and those in the process of execution. Return is always made to the next program step.

L. CSKIP

This macro removes from the tables all entries that have been executed. Return is always made to the next program step.

EXPANSIONS OF THE IB MONITOR TRANSMISSION MACROS

The twelve Transmission macros have the following expansions:

```
(1)  READ          Y,T,ERROR

          STL        SYSTM1
          TXL        SYSTM2,0,1
          PZE        Y,T,ERROR

(2)  WRITE         Y,T,ERROR

          STL        SYSTM1
          TXL        SYSTM2,0,8
          PZE        Y,T,ERROR

(3)  WRITEF        Y,T,ERROR

          STL        SYSTM1
          TXL        SYSTM2,0,9
          PZE        Y,T,ERROR

(4)  REWIND        Y,T,ERROR

          STL        SYSTM1
          TXL        SYSTM2,0,12
          PZE        Y,T,ERROR

(5)  RUSH          Y,T,ERROR

          STL        SYSTM1
          TXL        SYSTM2,0,5
          PZE        Y,T,ERROR

(6)  IN            Y,T,NI,ERROR,EOF

          STL        SYSTM1
          TXL        SYSTM2,0,6
          PZE        Y,T,NI
          PZE        ERROR,0,EOF
```

(7) OUT Y, T, NO, ERROR, ETT

```
        STL     SYSTM1
        TXL     SYSTM2, 0, 7
        PZE     Y, T, NO
        PZE     ERROR, 0, ETT
```

(8) BACK Y, T, N, ERROR

```
        STL     SYSTM1
        TXL     SYSTM2, 0, 10
        PZE     Y, T, N
        PZE     ERROR
```

(9) DISP

```
        STL     SYSTM1
        TXL     SYSTM2, 0, 11
```

(10) CUT

```
        STL     SYSTM1
        TXL     SYSTM2, 0, 3
```

(11) CSKIP

```
        STL     SYSTM1
        TXL     SYSTM2, 0, 4
```

(12) CLEAR

```
        STL     SYSTM1
        TXL     SYSTM2, 0, 2
```

Notes:

(1) SYSTM1 = $(49)_{10}$

SYSTM2 = $(50)_{10}$

(2) Omission of any part of a variable field is a possible error.

CHAPTER 8: DATA SENTENCES

The data sentence facilities, which may only be used with the IB Monitor, provide a means of entering data, which requires conversion, without using storage space for the input/output system during execution. This permits an object program to use all available storage except that required by the monitor.

Data sentences consist of a decimal address followed by an equal sign (=) which in turn is followed by data items to be loaded. Data items in a data sentence are separated by commas. All data sentences except the last in a set are terminated by an *. The last data sentence in a set, or "data sentence block," is terminated by the symbol, $. Following the $ which terminates a data sentence block there may be a decimal address. This address is used as indicated below.

Data items in a sentence may be decimal (integers, fixed point, and floating point) or octal. Decimal data is the normal case, and these items are written in the usual way for SOS, e.g., 32.1E5, -52, 69.7B12. Octal data on the other hand must be enclosed in parentheses which are preceded by the character "O". For example, O(56), O(-777,432601).

The following is an example of a data sentence block consisting of three data sentences.

```
5680 = 63,  45,4E12,52.8B22,O(-1),-50*
7456 = 25,  -25,10,O(-7,74)*
6584 = 100,25,50$2000
```

Data Sentence Processing

When the IB Monitor encounters a DS1 card, control is passed to the Data Sentence Program. This program reads the data sentence block following the DS1 card and using the Input system converts the data as indicated. The converted data is then written on tape B1 following the object program as a separate file. Control is then returned to the Monitor. If additional DS1 cards and data sentence blocks are encountered, the data is converted, and each block written as a separate file on tape B1.

In order to use a data sentence block during the execution of an object program, the following calling sequence must be used to read the required block.

```
TSX     82,4
PZE     A, ,B
Error return
```

where A is the number of the desired data sentence file, if A $\neq$ 0. If A = 0, the next file in the sequence is the desired block.

B is the location to which the monitor returns after reading the block, if B $\neq$ 0. If B = 0, the return to be used is that specified after the $ which terminates the block.

A and B can be either symbolic or absolute.

Punching Data Sentences

Data sentences are punched as follows:

1. Card columns 1-72 only may be used.

2. A data sentence may occupy more than one card, and more than one sentence may appear in a card.

3. A data sentence may start in any card column.

4. Data sentences must be punched continuously from the first character in a card to the last in the card.

5. If a data sentence is contained in more than one card, the last character on a card must be a comma.

6. A blank column subsequent to the first character in a card will cause the remainder of the card to be ignored.

Error Conditions

During conversion of data items, two types of errors are detected.

1. Overflow/Underflow.

2. Illegal characters.

Both of the conditions cause a normal zero to be stored, and an error message to be printed. Processing is then continued with the next data item. When the block has been completed, the error return address of the calling sequence is used for the return.

Example:

The data sentence

8964 = 14, 16.3E3, 14.6B23, O(4777, -1)$

causes the following items to be stored in the locations indicated.

Location	Contents (Octal)
8964	16
8965	216775300000
8966	164631
8967	4777
8968	-1

The manner in which data sentence cards are used is described on page 08.03.02.

CHAPTER 1: INPUT

The input to the IB Monitor consists of one or more job decks. These decks consist of a program (which may be either symbolic, SQUOZE, or a combination of the two) together with control cards which indicate the processing desired for the program. The types of processing which may be selected by control cards are:

A. Compilation. The input for a compilation is a symbolic program which may or may not include SQUOZE decks from previous compilations. The output from a compilation will be a SQUOZE deck, and a listing as described in Section 04.

B. List. The input when a listing is desired depends upon the manner in which a listing is specified (see Chapter 3 for details).

C. Punch a new SQUOZE deck. A new SQUOZE deck may be prepared from a SQUOZE deck which includes modifications. When a new SQUOZE deck is punched, a listing is also prepared.

D. Punch Absolute Deck. An option is available by which an absolute binary deck may be prepared from a SQUOZE deck. It is not, however, advisable to exercise this option until a program is completely tested and corrected, since the debugging features of SOS are not available with absolute binary decks.

E. Execution. The input for an execution job is a SQUOZE deck which may or may not include symbolic modifications. Sets of data sentences (see Chapter 8, Section 7) may also be included.

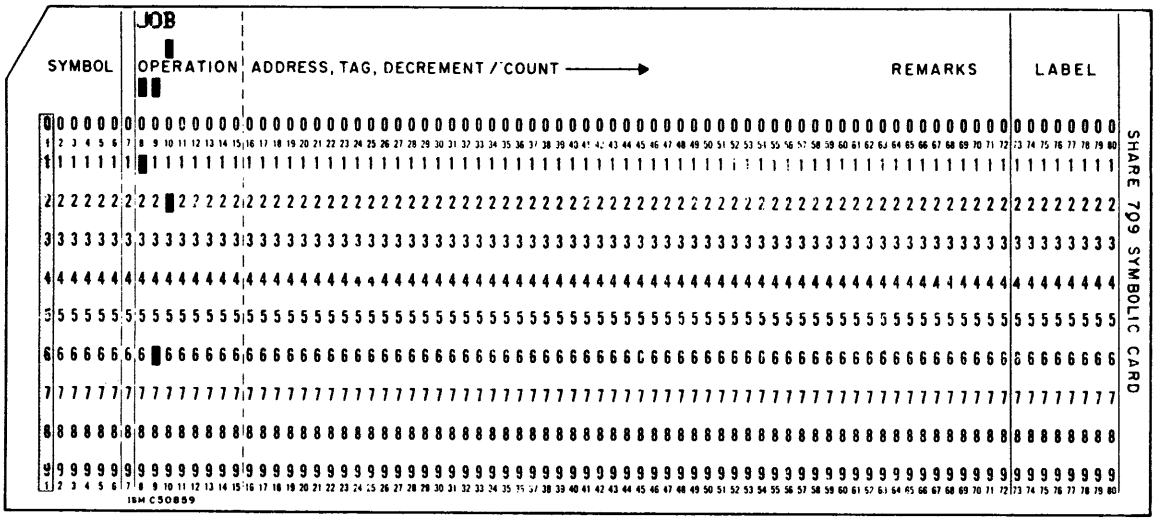
The control cards used for each of the above job types are discussed in Chapter 2, and the arrangement of job decks are given in Chapter 3.

CHAPTER 2: CONTROL CARDS

The IB Monitor uses fifteen control cards which indicate the processing required of the program, the beginning and end of a job deck, the date of processing, the format of the input, and the output desired. The effects of all the control cards are described below. In addition the Compiler pseudo-operation, SQZ, is described since it can be treated as control cards, and because it is mentioned in connection with deck arrangement. The manner in which they are used is discussed in Chapter 3.

It should be noted that information punched in all control cards except MOD will be listed by the system.

JOB

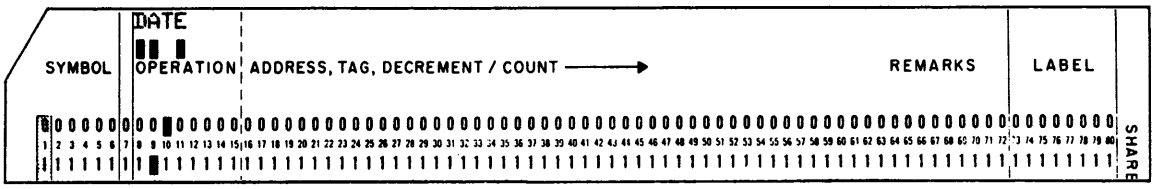


This card is used to indicate to the Monitor the beginning of a new job deck and is usually the first card of a deck. The JOB card may be the second card of a job deck, if it is preceded by a DATE card (see below).

The JOB card may be used as a means of introducing identification for a program, such as a program name. Columns 16-72 may be used for identification; however, only the characters in columns 16-27, if any, are included in the SQUOZE deck, and printed in the listings.

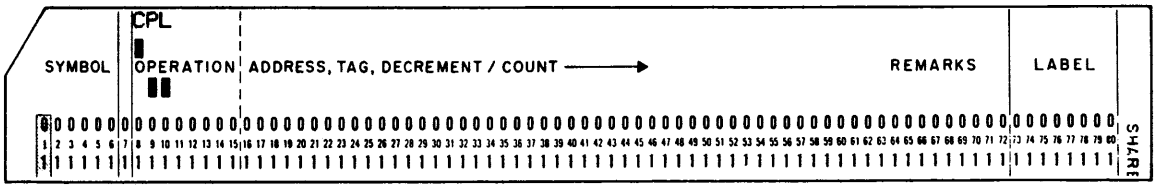
A JOB card will cause skipping to a new page during both off-line and on-line printing. The card will be punched preceding every output deck (both SQUOZE and absolute binary) and will be printed at the beginning of every listing.

DATE



This card is used whenever it is desired to supply a date to be incorporated when punching a SQUOZE deck. The date is punched in columns 16-21 as six decimal digits. For instance, July 4, 1976 would be punched 070476. This date would then appear on subsequent listings of the program until a new date is included when a new SQUOZE deck is punched. The new date will then replace the old one.

CPL



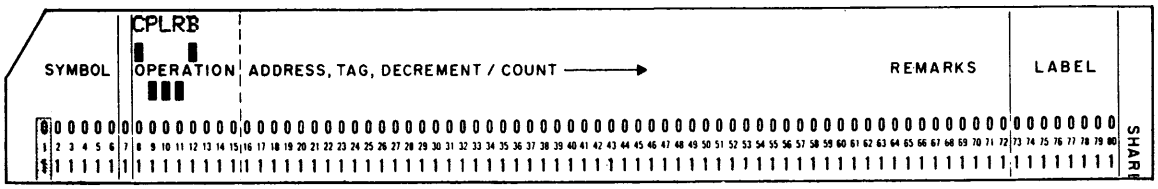
This card indicates:

1. The program which follows requires processing by the Compiler.

2. The SQUOZE output from the Compiler is to be punched in columnar binary form.

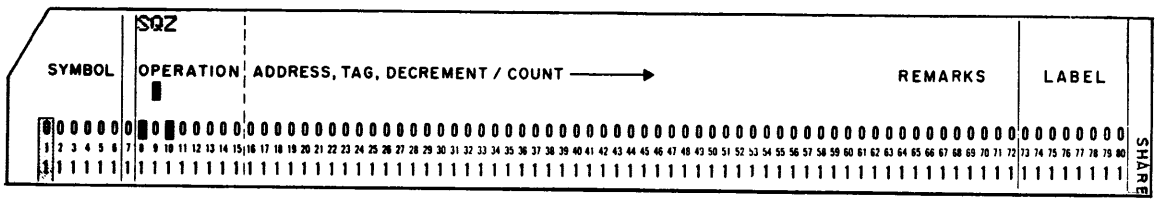
The program will also be listed as described in Section 04.

CPLRB



This card also indicates that the program which follows is to be processed by the Compiler. However, CPLRB indicates that SQUOZE decks are to be punched on-line in row binary form. A listing of the program will also be prepared.

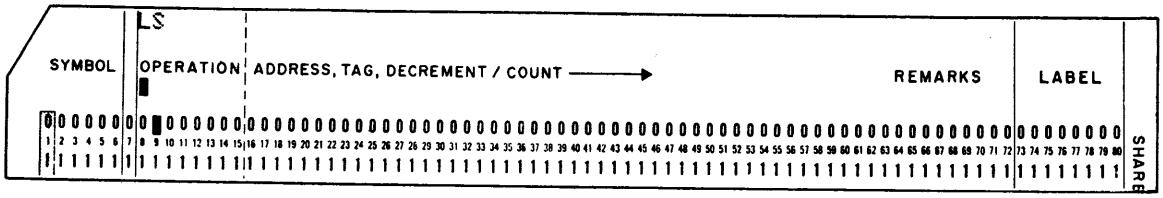
SQZ*



This card is inserted into a symbolic source program to indicate that a SQUOZE deck, which must be in columnar binary form, is to be incorporated into the program at the point where the SQZ card appears. The SQUOZE deck to be incorporated may be inserted into the symbolic deck following the SQZ card. If the source program is entered from tape, the SQUOZE deck may be entered from cards instead of from the tape (see page 08. 03. 02).

If the SQUOZE deck to be inserted is in row binary form, RB must appear in the variable field.

LS



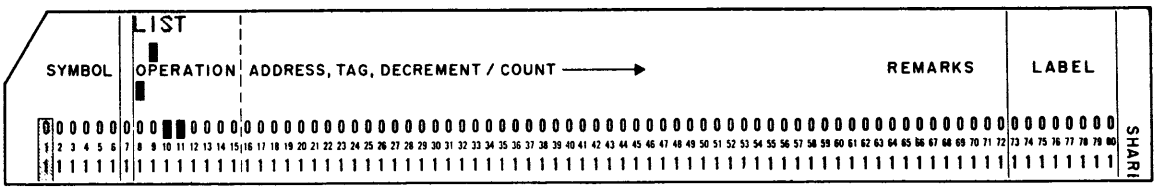
This card is used with a SQUOZE deck when it is desired to list a program. The LS card is used with SQUOZE decks which do not include symbolic modifications. The listing produced is identical with the one prepared when the SQUOZE deck was punched.

If the SQUOZE deck to be listed is in row binary form, this card must contain "RB" in columns 16-17. If the deck is columnar binary, columns 16-17 are left blank.

---

*This is a Compiler pseudo-instruction, rather than an IB Monitor control card, and is only included here for convenience.

LIST

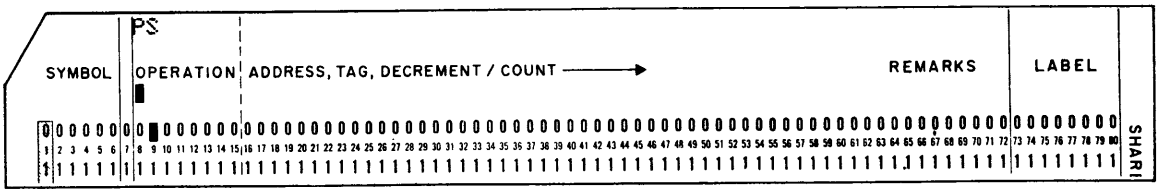


The LIST card will also cause a program listing to be prepared. LIST differs from LS in the following ways:

1. LIST does not require, as does LS, that the SQUOZE deck to be listed not include symbolic modifications.

2. The listing will be of the type prepared by the debugging macro-instruction CORE since, in fact, CORE is used to list the program. Thus, the listing produced by a LIST will be little more than a symbolic storage dump.

PS



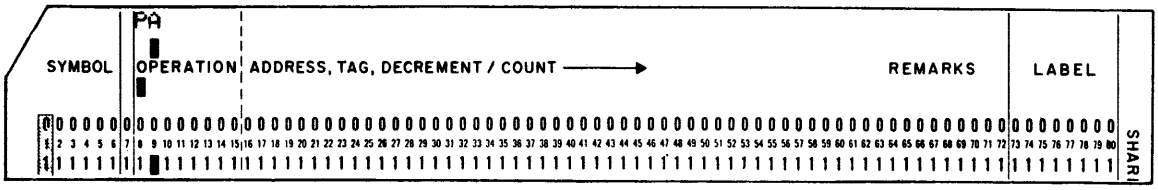
The PS card is used to cause punching of a new SQUOZE deck. In order to punch a new SQUOZE deck, symbolic modifications must be included. However, this requirement may be fulfilled by inserting a MOD and an ENDMOD card (see below) in the SQUOZE deck. When a new deck is punched a new listing of the program is prepared.

Columns 16-17 and 18-19 are also punched with "RB" when input and/or on-line output, respectively, are to be in row binary form. If input and/or output are to be columnar binary the appropriate pair of columns is left blank. Thus, if:

columns 16-19 contain:	input is:	and output will be:
RBRB	row binary	row binary
RBbb	row binary	columnar binary
bbRB	columnar binary	row binary
bbbb	columnar binary	columnar binary

where "b" represents a blank.

PA



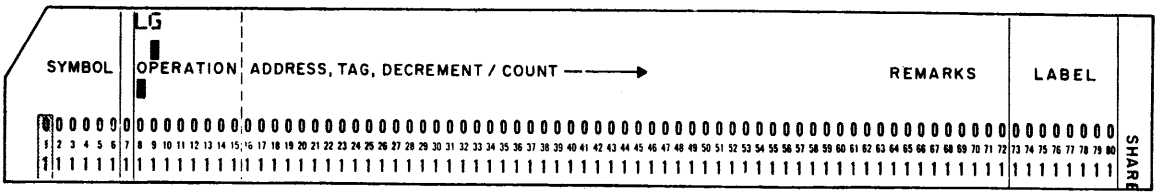
The PA card indicates that an absolute binary deck is to be prepared from the SQUOZE input deck. The SQUOZE deck may or may not contain symbolic modifications.

Columns 16-17 and 18-19 are also punched "RB" when input and on-line output, respectively are to be in row binary form. If input and/or on-line output are to be columnar binary, the appropriate pair of columns is left blank. Thus, if:

columns 16-19 contain:	input is:	and output will be:
RBRB	row binary	row binary
RBbb	row binary	columnar binary
bbRB	columnar binary	row binary
bbbb	columnar binary	columnar binary

where "b" represents a blank.

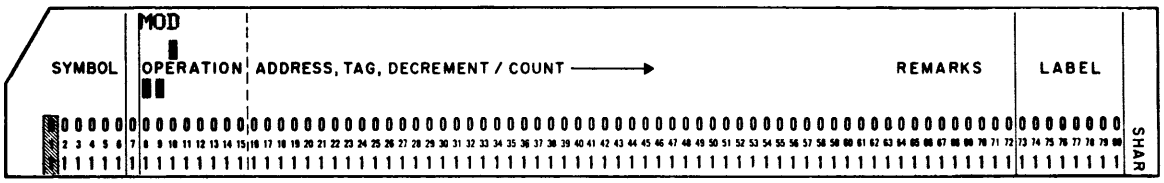
LG



This card indicates that the SQUOZE input is to be converted to absolute in preparation for listing (by LIST) or for execution. The absolute form of the program is written on tape B1 in n+1 files, where n TCD instructions are present in the program. These sections are written as the first, second, ..., n+1th files. The first n of these files are terminated by a TCD instruction and the n+1th file by an END (which, is required to be present). Thus, if no TCD instructions are used in the program, there will be only one file written on tape. (The subsequent use of the tape files written by LG is included with the descriptions of the DS1 and GO cards.)

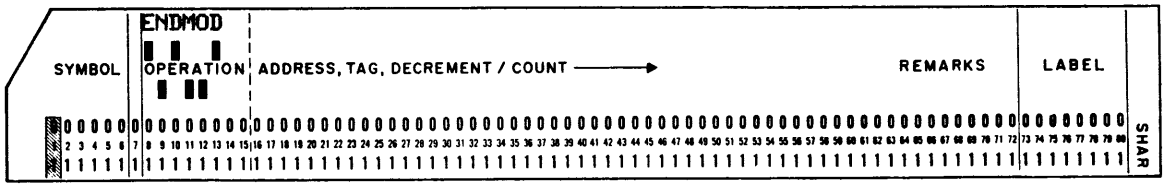
If the SQUOZE input deck is in row binary form, this card must contain "RB" in columns 16-17. If the deck is columnar binary, columns 16-17 are left blank.

MOD



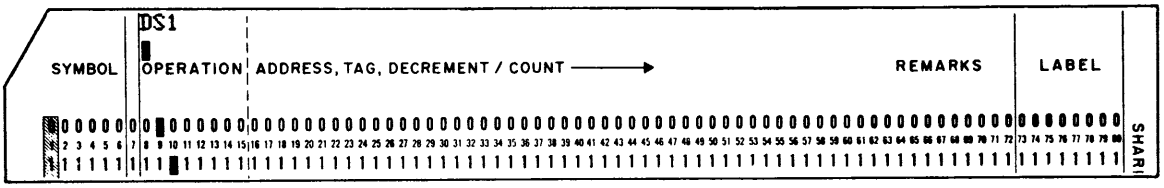
This card is used to indicate the start of symbolic modification cards. (It is never listed.)

ENDMOD



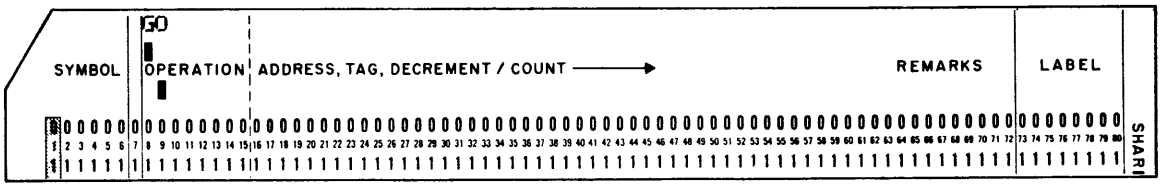
ENDMOD indicates the end of symbolic modification cards for a given program.

DS1



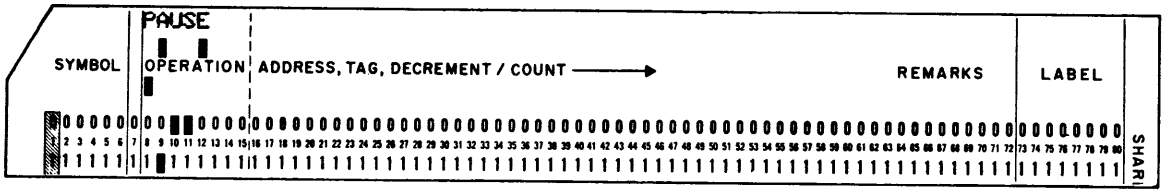
This card indicates that data sentences (see Chapter 8, Section 07) follow, and will cause the data sentence to be written in absolute on tape B1, following the absolute form of the program, as files n+2, n+3, . . . , n+m, where m is the number of data sentences. If errors are detected in the data sentences, error messages will be printed.

GO



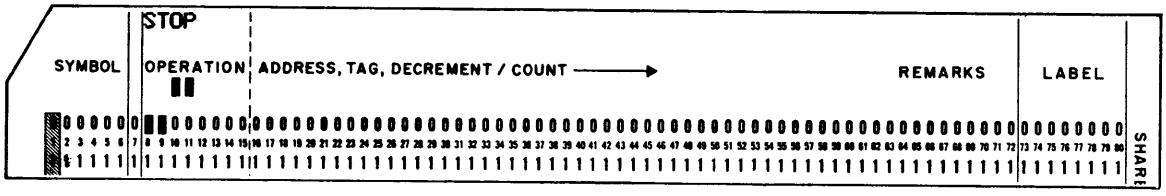
This card causes the debugging system to be loaded into core storage below location $(3000)_{10}$ and file 1 from tape B1 to be loaded into the locations assigned to the instructions in the record. Control is then transferred to the location specified by the TCD or END card of file 1.

PAUSE



PAUSE causes a halt in such a way that processing can be continued by depressing the Start key.

STOP



A STOP card causes a final halt, i.e., depressing the Start key will not continue processing. The card also causes an end-of-file mark to be written on the output tapes, if any, and the tapes to be rewound.

CHAPTER 3: JOB DECK ARRANGEMENT

As stated in Chapter 1, there are five types of processing which may be selected by IB Monitor Control Cards. These are:

A. Compilation
B. List
C. Punch a new SQUOZE deck
D. Punch an absolute binary deck
E. Execution

The arrangement and content of job decks for each of these types of processing is basically the same, and is given below.

1. JOB card

2. Pre-processing control card:

   a. CPL, CPLRB - input is a symbolic deck.

   b. LS - a SQUOZE input deck is to be listed.

   c. PS - a new SQUOZE deck is to be punched.

   d. PA - an absolute binary deck is to be punched.

   e. LG - a SQUOZE deck is to be loaded for execution or for listing by means of LIST.

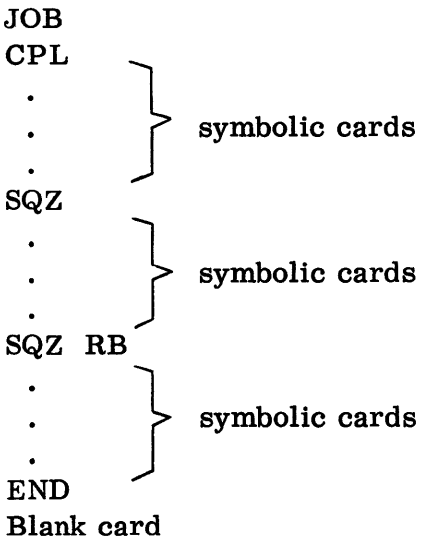
3. Input deck

   a. If the preceding card is CPL or CPLRB this is a symbolic deck. SQUOZE decks without modification may be included, but such decks must be preceded by either an SQZ (or SQZ RB) card. Alternatively, this type of job deck may be loaded in the following way:

      (1) All cards, except row binary SQUOZE decks, may be entered from tape. The SQZ (or SQZ RB) control card is included at the appropriate points in the symbolic program on tape. If columnar binary decks are to be entered from tape, they must appear where they are to be incorporated.

(2) The row binary SQUOZE decks and, if desired, columnar binary SQUOZE decks to be incorporated into the program, are stacked in the card reader in the order in which they are used in the program.

For example, the arrangement of the input tape would be:



The two SQUOZE decks to be incorporated into the program would be entered from the card reader. The columnar binary deck must, of course, be the first in the card reader and must be followed by one in row binary. The columnar binary SQUOZE deck may also be written on tape following the SQZ card and only the row binary deck read from cards.

b. In all other cases, this is a SQUOZE deck. Modification cards may be included in the deck. These cards must be preceded by a MOD card, followed by an ENDMOD card, and then inserted immediately preceding the blank card in the SQUOZE deck.

Modifications must be included, if the preceding control card is PS. However, this requirement is satisfied by the presence of a MOD and an ENDMOD card.

Modifications may not be included if the preceding card is LS. In all other cases, modifications are optional.

4. Blank card (required with CPL and CPLRB; optional for all others).

5. Data sentence deck, if desired. This deck may include any number of data sentence blocks. Each data sentence block must be preceded by a DS1 card and followed by a blank card.

6. Post Processing Package

   a. GO — The program is to be executed. This may be followed by input data if any.

   b. LIST — if the program is to be listed in the form of a symbolic core dump.

   c. PAUSE — if a halt is desired between jobs.

Operator Instructions

In addition to the job deck components listed above, a set of cards with operating instructions (columns 8-13 of such cards must be blank), and a PAUSE card may be inserted immediately following the JOB or DATE card. These cards will cause printing of instructions for the operator and a pause while the instructions are followed. Then, when the Start key is depressed processing will be continued.

STOP

A STOP card may be used to terminate input (i. e., a group of stacked jobs).

SHARE MONITOR

CHAPTER 1: INTRODUCTION

The SHARE Monitor operates as a three phase system. All jobs are processed through one phase before going into the next phase.

In general, Phase 1 converts a stack of jobs, in the form of SQUOZE or symbolic decks into machine language codes, and associated data packages into binary data. The results of this conversion are stacked on a system intermediate tape.

Symbolic and SQUOZE decks may also be converted by Phase 1 into new or revised SQUOZE decks, absolute decks, and listings. These are stacked on the appropriate system peripheral output units.

Phase 2 executes each of the stack of job codes formed in Phase 1. For each job the system assigns the required tapes, loads the necessary system input/output routines, loads the code and executes it. Jobs using the Debugging System or Output Editor write condensed binary output on a system intermediate tape.

Phase 3 converts the stacked output from Phase 2 and writes it on the appropriate peripheral output units.

Conversion and Input/Output Routines

Routine	Usage
Compiler; Modify and Load	Performs all conversion of symbolic and SQUOZE decks during Phase 1.
Input Editor	Converts and edits data packages during Phase 1 if requested by the programmer. The data package for each job is placed immediately behind the symbolic or SQUOZE deck. By using this routine, the programmer avoids having to furnish his own routines and core storage areas for editing of input data during execution of his job in Phase 2.
Buffering Routines	These routines are in core storage and are available to the programmer at all times. They must be used by an execution job to read data packages processed through Phase 1. They may also be used to read and write programmer intermediate tapes.

Transmission Macro Routines	These routines are read into core storage just prior to the execution of a job which specifies their use. They are used to read and write programmer tapes.
INTRAN Routines; OUTRAN Routines	These are brought into core storage just prior to the execution of a job which specifies their use. They allow the programmer to read, write, and edit information using his own input or output units. These routines occupy considerable core storage during execution of a job. Therefore, if this space is required for an object program, the Input and Output editors should be used.
Debugging Routines	These routines are brought into core storage just prior to the execution of a job which specifies their use. Information macros cause writing of condensed binary output on a system intermediate tape. Debugging information for all jobs is converted to peripheral form in Phase 3.
Output Editor Macro Routines	These routines may always be used by the programmer during execution of his job. The routines write editing information and unedited output data on the same intermediate tape used by the Debugging System. The Output Editor itself is used by the system in Phase 3 to edit and convert this information to peripheral form for all jobs. Hence, no space is required for output editing during execution of the job.

Tape Usage

The system normally uses the following tapes:

Phase	Tape Name	Contents or Use
1	SYSPOT (Peripheral Output Tape)	Listings converted from symbolic or SQUOZE decks for all jobs.
	SYSPPT (Peripheral Punch Tape)	SQUOZE or converted Absolute decks from Modify and Load or the Compiler for all jobs.
	SYSES1 SYSES2	Erase tapes used by Modify and Load or the Compiler.

Phase	Tape Name	Contents or Use
	SYSPIT (Peripheral Input Tape)	Original job decks.
	SYSMIT	System intermediate tape.
	SYSMOT	System intermediate tape.
	SYSTAP	Contains all system components
2	SYSMIT	See above.
	SYSMOT	See above.
	SYSTAP	See above.
	No peripheral tapes are assigned or used by the system during Phase 2.	
3	SYSDOT (Debugging Peripheral Output Tape)	Contains converted debugging output for all jobs.
	SYSPOT (Peripheral Output Tape)	Contains edited and converted printed output for all jobs using the Output Editor.
	SYSPPT (Peripheral Punch Tape)	Contains edited and converted punch output for all jobs using the Output Editor.
	SYSMIT	See above.
	SYSMOT	See above.
	SYSTAP	See above.

Non-standard system use:

1. The programmer may, if he desires, have his job executed in Phases 1 or 3 rather than in Phase 2. This will mean, of course, that fewer tapes are available to him; however, it does allow the programmer to write directly on the peripheral output units.

2. Special programmer tapes have names of the form SYSXRN or SYSXUN where X and N are the channel letter and unit number respectively. R and U denote reserved or utility tape.

Such a symbolic tape name serves as a location symbol for a cell containing the absolute tape address assigned to that symbolic tape. This absolute tape address is in the address field of the cell and is always a decimal mode address. Thus, to read tape SYSAR5 independent of system subroutines, in the binary mode, the following sequence could be used.

```
        CLA     SYSAR5
        ORA     BIN
        STA     SEL
         .       .
         .       .
         .       .
SEL     RDS     **
         .       .
         .       .
         .       .
BIN     OCT     20
```

SHARE MONITOR

CHAPTER 2: CONTROL CARDS

Control cards provide a compact and flexible method of communication between the programmer, the operator, and the system.

All SHARE Monitor control cards have the following common characteristics:

1. Column 1 contains 7-, 8-, and 9-punches to identify it as a control card.

2. The operation code of the control card is punched in the normal SCAT format, i.e., beginning in column 8.

3. The variable field must be separated from the operation code by at least one blank column, and must begin no later than column 16, i.e., in SCAT format.

4. If a normal case is defined for an item in the variable field of a control card and this normal case meets the programmer's needs, it is not necessary to punch the parameter in the control card. Parameters need be specified only in situations which require other than the normal case. If the variable field is entirely blank on a control card which has a normal case defined for all items, the System will employ the normal case for all items.

   Note: Only one comma should be inserted between the items actually punched on the control card.

The following sections describe the control cards.

JOB

The JOB card must be the first card of any deck to be processed by the system, since it serves to notify the system that a new job is about to commence.

JOB JN, RN, MN, ET, EMO, EPO

where JN = Job Number
RN = Run Number
MN = Man Number
ET = Estimated Running Time (Unit is 0.01 hrs.)
EMO = Estimated Mediary Output (Words)
EPO = Estimated Peripheral Output (Records)

The information contained in the variable field of the JOB card serves the following purposes:

1. It causes the system to enter the Accounting Initiation routine.

2. The identifying information (job-, run-, and man-numbers) is passed along from phase to phase to identify debugging and other peripheral output.

3. The estimated running time may be used by an Interval Trap routine or by the operator to determine whether the object program is using too much time in the execution phase.

4. The estimates of mediary and peripheral output will enable the System to discover and forestall such errors as excessive debugging output, etc.

Each installation may, with a minimum amount of modification to the Monitor, specify its own parameters for the JOB card.

LOAD

The LOAD card follows the JOB card in the input deck and indicates that a SQUOZE deck (with or without a modification package) is to follow. The card furnishes all the required information as to which parts of the system will be required, when they will be required, and how they are to operate on the associated SQUOZE deck.

LOAD A, B, C, D, E, F, G, H, J, K, L, M, N, P, Q

where

A	=	+ for input with only commentary text
	=	- for input with only non-commentary text
	=	B for input with both SQUOZE texts
B	=	R for row binary input
	=	C for columnar binary input
C	=	GOIF to execute if no errors
	=	GO to execute if no definite errors
	=	GOGOGO to execute provided the system was able to produce an executable code.
	=	NOGO to suppress execution
D	=	1-execute program in phase 1
	=	2-execute program in phase 2
	=	3-execute program in phase 3
E	=	SQZ to punch new SQUOZE deck
	=	NOSQZ to suppress punching SQUOZE deck
F	=	+ to output only commentary text
	=	- to output only non-commentary text
	=	B to output both texts

G	=	ABS to punch absolute deck
	=	NOABS to suppress punching absolute deck
H	=	R for row binary output
	=	C for columnar binary output
J	=	LIST to obtain listing
	=	NOLIST to suppress listing
K	=	DICT to write a dictionary as part of the program listing
	=	NODICT to suppress dictionary output
L	=	DEBUG to execute with the Debugging System
	=	NOBUG to execute without the Debugging System
M	=	SS to use System Symbols
	=	NS to suppress use of System Symbols
N	=	IN to execute with INTRAN
	=	NOIN to execute without INTRAN
P	=	OUT to execute with OUTRAN
	=	NOOUT to execute without OUTRAN
Q	=	NOMAC to execute without Transmission macros
	=	TRMAC to execute with Transmission macros

The normal case is:

LOAD B, C, GO, 2, NOSQZ, +, NOABS, C, NOLIST, DICT, DEBUG, SS, NOIN, NOOUT, NOMAC

SCAT

The SCAT card informs the System that a symbolic deck to be compiled follows. The card causes the presetting necessary to fulfill the demands of the various parameters.

SCAT C, D, E, F, G, H, J, K, L, M, N, P, Q

(The parameters C through Q are as described above for the LOAD card).

The normal case is:

SCAT NOGO, 2, SQZ, +, NOABS, C, LIST, DICT, DEBUG, SS, NOIN, NOOUT, NOMAC

SINGLE TEXT SQUOZE DECKS

The options for single text output on both the SCAT card and the LOAD card, and the options for single text input on the LOAD card, may be exercised. The admissible parameter forms are as follows:

B = both texts are present or required.
+ = text with commentary only is present or required.
- = text without commentary only is present or required.

No difficulty should be experienced with single text output. Use of this option will result in a SQUOZE deck which is numbered consecutively throughout, and has only one text section. The Preface card will contain the correct text word counts.

Single text input may consist of either a double text deck, with one text section manually removed; or a single text deck which was produced by a previous single text output run. In the former case, the Preface card, which must not be changed indicates that the SQUOZE deck is double text. The input parameter on the LOAD card must correspond to the actual state of the deck. In the latter case, the Preface card describes the deck completely and the input parameter on the LOAD card is ignored.

The following points should be noted:

1. Single text output may be obtained from double text input and vice versa. If, however, single text input has only text without commentary, then the two text sections of the double text output will, in the absence of modifications, be identical.

2. The normal case for output is single text with commentary. The normal case for input is double text. A single text input deck will load without difficulty even though double text input is specified or implied by the LOAD control card.

3. An absolute deck can be punched on the same run as single text SQUOZE deck.

4. Alter numbers and relative numbers are the same for text without commentary as for the corresponding text with commentary. Thus, for instance, it is possible to load with modifications using only text without commentary and referring alter numbers to a listing of text with commentary.

5. All operations previously possible with double text are possible with single text.

IDENT

The IDENT card is used, if desired, in conjunction with a SCAT or LOAD card and its associated deck. A and B, which are described below, are punched in the 9-right and 8-left word of the SQUOZE deck Preface card, and will appear in the upper right-hand corner of each page of the listing, if a listing is requested. C is punched in the stated columns of each card in the SQUOZE deck if the punch-sequencing device is installed on the on-line card punch.

The IDENT card (if used) must be positioned between the JOB card and the SCAT or LOAD card.

IDENT A, B, C

where A is the first BCI word for the Preface
B is the second BCI word for the Preface
C is the BCI field for columns 73-76

(Note: Commas and blanks are illegal BCI characters in the above fields since they serve as field terminators.)

Identification in the Preface card will follow the rules given below.

1. If an IDENT card exists, in a SCAT run, identification in the Preface comes from the first two subfields of the IDENT card.

2. If no IDENT card exists, the identification is taken from the first and third subfields of the JOB card. In a LOAD SQZ run the identification in the Preface of the output deck will be the same as that of the input deck unless an IDENT card is included whose first two subfields are non-zero or non-blank.

ASSIGN

The ASSIGN card causes a physical tape unit to be assigned to the symbolic reserve tape and utility tape names referred to by a job. The ASSIGN cards would normally be inserted in the input deck by a machine operator with knowledge concerning available tapes. However, certain installations may desire that the programmer know what physical tapes are available for his use. The ASSIGN card has the form:

ASSIGN XN=SYSXZN

where:

X = A through F, specifying a symbolic channel
N = 1 through 8, specifying a symbolic tape drive
Z = R for Reserved Tape assignment
Z = U for Utility Tape assigment

The assignment of utility and/or reserved tapes is accomplished by placing the appropriate ASSIGN cards after the JOB card and before the LOAD or SCAT card of the job for which the assignment is to be effected.

Examples:

a. ASSIGN B5 = SYSBR1

This card assigns unit 5, channel B as symbolic tape SYSBR1 for the job with which this ASSIGN card is associated.

b. ASSIGN C2 = SYSAU1

The above card will assign physical unit 2, channel C for use in all references to symbolic utility tape A1 during the execution of the object program.

At the conclusion of execution of the object program using the above ASSIGN cards, tape drives B5 and C2 would be rewound by the System, and the operator would be instructed as follows:

REMOVE B5 = SYSBR1

Thus, remove messages are given only for reserved tapes.

DATA

The DATA control card causes the following cards (or records) on the system input unit to be converted and/or transcribed on the specified tape unit in standard buffered format during Phase 1. Data processed through Phase 1, whether edited or not, must be read by the executing program using the Buffering routines SYSRTK or SYSWTK.

If the programmer wishes the data to be converted and edited, the parameters of the DATA card cause appropriate initialization prior to entry into the Input Editor (see page 07.03.01) for conversion and transcription. If data is to be only transcribed on the specified tape unit, the Input Editor is not required. BCD records are written as 12-word logical records and binary records are written as 24-word logical records. A subsequent DATA control record causes an Logical End of Group flag to be written; any other control record causes a Logical End flag to be written on SYSMOT or an End of Logical Tape flag to be written on SYSXRN. In the latter case, SYSXRN will be rewound and become unassigned.

The format of the DATA control card is:

DATA A,B,C

where A = EDIT if the Input Editor is to be used for conversion and transcription.

= NOEDIT if simple transcription is required.

B = GO to continue processing if bad data is encountered by the Input Editor.

= GOIF to discard the job if bad data is encountered.

C = SYSMOT if data is to be placed on the System Mediary Output Tape.

= SYSXRN if data is to be placed on reserved tape "N" on channel "X."

The normal case is:

DATA EDIT,GOIF,SYSMOT

SHARE MONITOR

CHAPTER 3: INPUT DECK ARRANGEMENT

Owing to the ordered sequence in which the various components of the System perform their functions on a given job, there are strict rules for the arrangement of the input deck. The first of these rules is that the JOB card must be the first card in a deck. The JOB card may be followed by an IDENT card, and/or ASSIGN cards. The next card must be either a LOAD or a SCAT control card, depending upon the type of operation to be executed. The composition of the remainder of the deck for each of these cases is described below:

A. LOAD control card

SQUOZE program deck (no blank card at end of SQUOZE deck)
Data package(s) (optional, see below)

B. SCAT control card

Symbolic deck (terminated by an END card)
Data package(s) (optional if "GO" requested, see below)

Data packages may be for Input Editor translation (EDIT) or for mere transcription (NOEDIT). Their arrangements are:

A. EDIT

DATA control card
Blank card
Data cards
ENDATA control card
Blank card

B. NOEDIT

DATA control card
Data cards

Multiple data packages may be used at the positions of the input decks as noted above, and in such a set, some can be EDIT packages while others are NOEDIT packages.

Note: The above are all of the allowable sequences of cards in an input deck for one job. Successive jobs are merely repetitions of these sequences.

CHAPTER 4: COMMUNICATION REGION TRANSFER POINTS AND ASSOCIATED STANDARD ROUTINES

Transfer points are designed to allow the programmer three options in dealing with errors discovered in the object program. Each transfer point consists of two instructions within the Communication Region, the first of which is labeled with a system symbol to which the programmer may refer. A transfer point is of the form

```
SYSXYZ   TXH   **,,**
         TXL   XYZ,,0
```

where SYSXYZ is the reference system symbol
XYZ is the System routine normally entered.

The option desired by the programmer is exercised through modification of the instruction at SYSXYZ. Under no circumstances may the instruction at SYSXYZ+1 be altered or replaced by the object program.

The options afforded the programmer are:

A. Execute the standard routine provided by the System with the return determined by the System routine entered.

Method

If the first instruction is not preset by the programmer, the TXH at SYSXYZ will fail. Entry will be made to the specified System routine via the TXL at SYSXYZ +1.

B. Execute the standard routine provided by the System and return to the address specified by the programmer.

Method

The decrement of SYSXYZ is preset by the object program with the special return. After the standard System routine has been entered and executed, control is transferred to the address specified in the decrement.

C. Execute a special routine provided in the object program.

Method

The instruction at SYSXYZ is replaced by a transfer to the object programs' special routine.

D. Execute a special programmer routine followed by the System routine.

Method

The instruction at SYSXYZ is replaced by a transfer to the object program's special routine. This routine then transfers to SYSXYZ+1 after performing its own functions.

E. Execute a special programmer routine followed by the System routine but making a return specified by the programmer.

Method

The instruction at SYSXYZ is replaced by an instruction such as TXL OWN1, OWN2. Transfer to SYSXYZ causes entry to OWN1. After OWN1 is completed, it transfers to SYSXYZ+1 to carry out the System routine. The System routine then makes the special return to OWN2.

There are nine transfer points and associated standard routines which the programmer may wish to use. Where applicable the setting of an error indication will result in a message which will be part of the debugging output for the job.

A. SYSERR: The standard Unexpected Error routine provides a message on SYSDOT saying 'THIS JOB HAS CAUSED A RETURN TO SYSERR FROM XXXX." If no special return is given in the decrement of SYSERR, the system will load SNAP into core and give a console scoop before returning to SYSTEM. The console scoop is suppressed for special returns to avoid the possibility of SNAP covering the object code.

The calling sequence to SYSERR is:

```
TSX     SYSERR, 4
  No return normally
```

B. SYSBAD:

```
TSX     SYSBAD, 4
X       TAPE, T, Y
```

where X = PZE for read
= MZE for write
TAPE = location of tape unit address
T = 0 for BCD mode
= 7 for binary mode
Y = location of the beginning of the I/O table

The Bad Spot routine will reposition the tape and read or write the records indicated in the object program's I/O list. If the routine is not able to accomplish this without encountering a condition listed below, the tape will be positioned as found and the program transfers to SYSTRC. That routine prints the number of the unit which cannot be used successfully and transfers back to the monitor to process the next job. The conditions which may cause this are:

1. record or end-of-file cannot be written, five attempts have been made to do so. Blank tape has been written prior to each attempt.

2. records in the I/O list cannot be read successfully. The tape was correctly repositioned, but 10 attempts to re-read have resulted in redundancy.

3. in repositioning the tape from an ambiguous command list (see below), the routine is not able to read the first five words without redundancy. 15 attempts have been made.

4. beginning-of-tape has been encountered while the routine is trying an extra-backspace and comparison of the 5 words (see below), in order to find the correct position for re-reading.

5. the tape cannot be correctly repositioned although more than 25 extra records have been backspaced and the first 5 words compared.

The I/O list is specifically restricted in that it may not contain more than 25 records with count type commands.

In writing a tape, there is no question about the number of records which must be backspaced before another attempt can be made to write. Read commands, unless they are exclusively IORP or IORT, force the Bad Spot routine to do a special search. It backspaces the minimum number of records, reads the first five words referred to in the command list and compares them to the first five words which must have been read into storage. A match, or a match except for a single bit, is recognized as a correct position for an attempt at re-reading. Therefore, an IORP or IOSP at the beginning of the list must specify a word count of at least 5, and refer to a tape record which contains five or more words.

The routine will not make the comparison unless the first five words, at least, can be read without redundancy; up to 15 attempts will be made to read the first five words of the record.

When the comparison shows that the tape is not yet in position, another backspace is executed and the comparison made again. A beginning-of-tape encountered at this stage will cause the routine to abandon the job after spacing the tape forward in order to leave it as found.

C. SYSTRC: The standard Tape Redundancy Check routine is entered by a TSX SYSTRC, 4 with the symbolic name of the tape unit in index register 2. An indication of the error is made in the problem status indicators and the routine exits to SYSERR.

D. SYSIOC: The standard Input/Output Check routine is entered by a TSX SYSIOC, 4. An indication of the error is made in the problem status indicators and the routine exits to SYSERR.

E. SYSTDC: The standard Divide Check routine is entered by a TSX SYSTDC, 4. Normally this routine goes to SYSERR after leaving an indication of the error.

F. SYSTUF: The standard Floating Point Underflow routine is entered when a floating point underflow is trapped. The register in which underflow occurred is set to zero and an error indication made. Return is normally to the object program at the instruction following the one causing underflow.

G. SYSTOF: The standard Floating Point Overflow routine is entered when a floating point overflow is trapped. Normally this routine exits to SYSERR after leaving an indication of the error.

H. SYSTRP: The standard Transfer Trap routine is entered if ETM is executed in a programmer's job and no entry to a programmer's routine has been placed in location SYSTRP. The standard routine will create a message on the debugging output unit indicating that no programmer routine was furnished and will go on with the next job.

I. SYSSTR: A store location and trap instruction normally causes entry to the Unexpected Error Routine SYSERR.

CHAPTER 5: EXECUTION COORDINATION UTILITY ROUTINES

Certain system routines are available to the programmer at execution time for the purpose of performing standard operations. These routines are:

TITLE: COMMENT ATTACHED PRINTER: SYSCAP

PURPOSE: SYSCAP may be used to print up to 12 words of information in a single line on an on-line printer. The routine should be used only for those messages to the operator which are vital to the operation of a job.

CALLING SEQUENCE:

```
TSX     SYSCAP,4
X       L,,N
return
```

where X = PZE to force a skip to channel 2 of the carriage control tape
= MZE to suppress skipping
L = the location of the first BCI word to be printed
N = the number of BCI words to be printed

The N BCI words are converted into a line image and the machine is delayed until the printer channel is free. The comment is then transmitted to the printer.

TITLE: MEDIARY TAPE LOADER: SYSMTL

PURPOSE: SYSMTL may be used by the programmer to load a program section, following a TCD card into core storage from SYSMIT, the mediary input tape.

CALLING SEQUENCE:

```
TSX     SYSMTL,4
X       SYSMIT,,R
```

where X = PZE to preset core to TSX SYSERR,4
= MZE to suppress presetting
R = is the return address
= 0 to use return on tape, i.e., the address specified on the END or TCD card

NOTES: Upon return, the sign of the accumulator is positive if loading is terminated by a TCD card. The sign is negative if loading is terminated by an END card. In either case, the address of the END or TCD card may be found in location SYSTRA.

SYSMTL assumes that SYSMIT is in correct logical position.

To alternate program files or load them in other than sequential order, the tape must be positioned properly by use of the Buffering routines SYSBKS, SYSWTK, and SYSRTK.

CHAPTER 6: AVAILABILITY OF MACHINE COMPONENTS

During Phase 2, the following machine components are available for use by the programmer:

A. Core storage above the system origin, which is the location of the first cell following those routines which must remain in core storage at all times, and those special System routines requested by the programmer for use in the execution of his job. This location is found in the address field of location SYSORG. Unless specified by an ORG card in the program, code will be automatically assigned above the system origin.

1. If INTRAN is requested, SYSORG will be at least 42,200 octal.

2. If OUTRAN is requested, SYSORG will be at least 26,500 octal.

3. If the Transmission macros are requested, SYSORG will be at least 13,700 octal.

4. If, the Debugging System is requested, SYSORG will be at least 12,600 octal.

5. If no System routines are requested SYSORG will be at least 10,500 octal.

B. All drum units.

C. The on-line printer may only be used to print special operator instructions by means of the System subroutine SYSCAP

D. All tapes except those designated as system tapes may be assigned as reserved or utility tapes. The system tape SYSMIT may be read to:

1. Load sections of a program following a TCD card, with the aid of the System Mediary Tape Loader, SYSMTL.

2. Load data converted by the System during Phase 1. In this case the Buffering routines SYSRTK (the routine which reads a logical record) and SYSWTK (the routine which reads a single word) are used.

During execution of a job's code in Phase 2 the following machine components are not available to the programmer:

A. Core storage from decimal location 00000 through the system origin.

B. The three System tapes SYSMIT, SYSMOT, and SYSTAP except as noted above.

C. Sense switches 1 through 6.

D. The MQ entry keys.

Accuracy	Correctness or freedom from error (as contrasted with precision).
Address	1. A label, name, or number identifying a register, location or unit where information is stored. 2. Loosely, the address field of a machine word.
Alphameric	A generic term for alphabetic letters, numerical digits.
Assembly Program	A program to translate a routine written in a symbolic machine language into absolute machine instructions, and to assign machine storage for those instructions and data.

BCD	Abbreviation for binary-coded-decimal.
Binary Cards	Cards containing up to twenty-three 36-bit binary words together with an origin, a word count, and an ACL checksum.
Binary Cards, Row	Binary cards in which successive bits are found by reading columns 1-36 and 37-72, alternately, row-by-row starting with row 9.
Binary Cards, Column	Binary cards in which successive bits are found by reading down the columns (starting with the leftmost, column 1).
Blocking	The combining of two or more data or item records into a tape record, or block. By thus reducing the number of inter-record gaps on tape, the acceleration or deceleration time per data record is reduced, and the number of data records which may be contained in a given length of tape is increased.
Block Length	The total number of words contained in one block.
Bootstrap	A technique or device designed to bring itself into a desired position by means of its own effort, e.g., a special machine routine to bring itself into the computer from an input device.
Buffer	An area assigned for use as an intermediate storage area for data to be transmitted between storage and input/output devices.

Call	To transfer control to a subroutine by means of a calling sequence.
Card Field	A fixed number of consecutive card columns assigned to a unit of information, e.g., card columns 15-20 can be assigned to identification.
Calling Sequence	An instruction that (1) records its own location and (2) transfers program-control to a closed subroutine, together with as many locations as are necessary to hold the information ("parameters" or "arguments") needed by the subroutine.
Character	A decimal digit, alphabetic letter, or special symbol such as $, %, etc.
Check	1. Parity check - one type of redundancy check. 2. Redundancy check - use of summation bits and redundant bits (check digits) to insure accuracy of tape information.
Clear (Verb)	To erase the contents of a storage location or register by replacing the contents with a pre-determined character, such as zeros, ones, nines.
Closed Subroutine	A routine which is not inserted as a block of instructions within a main routine but is entered by basic linkage from the main routine.
Column Binary	A form of binary card punching in which the first word on the card occupies columns 1-3 and, for a full card, the last word occupies columns 70-72. Bits 0-11 of each word go into positions 12-9 of the leftmost of the three allotted columns, etc.
Compare	To examine the representation of two groups of characters for the purpose of discovering relative magnitude.
Conditional Transfer	A transfer which occurs only when a certain condition exists at the time control passes to the transfer instruction.
Control Card	A card which contains input data or parameters for a specific application of a general routine.

Dump	To copy the contents of part or all of some storage medium onto another storage medium,e.g., to write the contents of core storage on a peripheral output tape through a translator.

Edit	To rearrange information for machine output or input.
Exponent	That portion of a floating point number which represents an integral power.

Field	A set of one or more contiguous bits or characters treated as a unit of information.
File	1. A collection of records, an organized collection of information. 2. On tape, a sequence of records terminated by an end-of-file mark and file gap.
Fixed Length Records	Records comprising a file in which every record is the same length.
Flag	A field which serves as a signal to a processor.
Format	The predetermined arrangement of characters, fields, lines, page numbers, punctuation marks, etc.

Grouping	Combining two or more records.

Initialization	Setting counters, switches, and instruction addresses at specified times in a program.
Instruction	1. Machine Instruction - An instruction directly recognizable by a machine. 2. Symbolic Instruction - In an assembly language, a group of symbols which can be translated directly into a machine code, i. e., there is a correspondence (usually one-to-one) between a symbolic instruction and a machine code (object language) instructions. 3. Pseudo-Instruction - A group of symbols which causes the Assembler to depart from the normal mode of translating symbolic instructions and to take some appropriate special action. 4. Macro-Instruction - A pseudo-instruction which calls for the insertion into the object routine of a sequence of instructions generated from a skeletal definition by the insertion of any parameters supplied.
Interrupt	A procedure by which the normal operation of the program is temporarily suspended by a special signal. The signal might be external to the computer, or might be caused by an error condition, or by the completion of an asynchronous operation. The machine branches to a routine appropriate to the cause of the interrupt, and at the completion of the routine, normal operation is resumed. Sometimes called trapping.
I/O	Abbreviation for input/output.

Library	A group of standard, proven routines which may be incorporated into larger routines.
Library Routine	A sequence of instructions which is used often enough to be identified and placed on file, and which is either the same sequence in all applications or else a sequence which is self-initializing through the use of parameters supplied at execution time.

Macro-Instruction	A pseudo-instruction which calls for the insertion into the object routine of a sequence of instructions generated from a skeletal definition by the insertion of any parameters supplied.
Mask	A machine word used with a logical instruction to eliminate undesired bits from another word.
Monitor	A routine which exercises supervisory control over some other program or collection of programs. When the collection of routines comprises all those normally used in the operation of a computer, the Monitor and the entire collection is called an operating system.

Object Program — The machine language program which is the final output of a coding system.

Off-Line — Pertaining to the operation of input/output devices or auxiliary equipment not under direct control of the central processing unit.

On-Line — Pertaining to the operation of input/output devices under direct control of the computer.

Open Subroutine — A separately coded sequence of instructions which is inserted into another instruction sequence directly in the line of flow of control. Sequences generated by macro-instructions are an example of open subroutines.

Origin

1. The absolute storage address of the beginning of a program or block.
2. In relative coding, the absolute storage address to which addresses in a region are referenced.

Overflow — In an arithmetic operation, the generation of a quantity beyond the capacity of the register.

1. For addition, generation of a sum greater than the capacity of a sum register.
2. For division, generation of a quotient greater than the capacity of the quotient register.

Precision	The number of significant digits in a quantity (see "Accuracy").

Record	A unit of information for computer input or output. A record may be of indefinite length, and need not be read in its entirety. A physical record is the smallest unit at which an input or output device may be positioned.
Redundancy Check	A check which uses extra bits.
Register	The hardware for temporarily storing information while or until it is used. (Usually not main storage.)
Reset	To set a field or switch back to an initial or standard condition.
Rewind	To return a magnetic tape to its beginning point.
Round-off Error	The error resulting from dropping the least significant digits of a number, and adjusting the most significant digits.
Routine	A sequence of machine instructions which carry out a well-defined function.
Row Binary	A binary card format in which the first word occupies the first 36 columns of the 9 row (hence, 9L), the second word occupies columns 37-72 of the 9 row (hence, 9R), the third word occupies 8L, and so on through 12R.

Self-loading	A term applied to a sequence of instructions which are so constructed that the first few instructions make the machine accept the following instructions automatically. (Sometimes referred to as "bootstrap.")
Simulator	1. A program or routine corresponding to a mathematical model or representing a physical model. 2. A routine which runs on one computer and imitates the operations of another.
Snap, Snapshot	A dynamic Dump, obtained by somehow interrupting the progress of a computation while a Monitor routine collects the desired information, after which the machine status at the time of interruption is restored and the computation resumed. The routine in SOS which performs such functions is called SNAP and the routine which converts the information for output is called SNAPTRAN.
Subroutine	See "Open Subroutine" and "Closed Subroutine."
Supervisor	See "Monitor."
Symbolic Coding	1. Broadly, any coding system in which symbols other than machine addresses are used. 2. A method of coding in which addresses are represented by arbitrary symbols bearing no absolute or relative relationship to actual memory locations (these symbols may be descriptive of the contents). In fact, the symbolic coding itself may bear little resemblance to machine language.
Symbolic Language	Any collection of symbols used in programming to represent operation codes, functions and/or addresses, with rules of usage.
Symbolic Modification	The insertion or deletion of coding in a SQUOZE deck by reference to symbolic locations in the original source program.

Tracing — An interpretive diagnostic technique to record executed instructions and results on an output device during execution.

Translate — To change information from one form of representation to another without significantly affecting the meaning.

Trapping

1. A hardware provision for interrupting the normal flow of control of a program while transfer is made to a known location. The trapping features are most commonly used by Monitor routines or for communication between input and output routines and the programs using them.

2. A technique of debugging for obtaining information during the execution of a routine.

Utility Routines	Standardized routines which perform a basic service.

TABLE OF PERMISSIBLE CHARACTERS

Character	Punched Card Code	BCD Code in Storage (octal)	BCD Code on Tape (octal)	SQUOZE Code (octal)	Character	Punched Card Code	BCD Code in Storage (octal)	BCD Code on Tape (octal)	SQUOZE Code (octal)	Character	Punched Card Code	BCD Code in Storage (octal)	BCD Code on Tape (octal)	SQUOZE Code (octal)
blank	blank	60	40	0	G	12 7	27	67	21	W	0 6	66	26	41
0	0	0	12	1	H	12 8	30	70	22	X	0 7	67	27	42
1	1	1	1	2	I	12 9	31	71	23	Y	0 8	70	30	43
2	2	2	2	3	J	11 1	41	41	24	Z	0 9	71	31	44
3	3	3	3	4	K	11 2	42	42	25	= † #	3-8	13	13	45
4	4	4	4	5	L	11 3	43	43	26	(%	0 4-8	74	34	46
5	5	5	5	6	M	11 4	44	44	27	) ¤	12 4-8	34	74	47
6	6	6	6	7	N	11 5	45	45	30	+ ✱ &	12 0	32	72	50
7	7	7	7	10	O	11 6	46	46	31	- ✱	11 0	52	52	51
8	8	10	10	11	P	11 7	47	47	32	- ✱ @	4-8	14	14	52
9	9	11	11	12	Q	11 8	50	50	33	+ † &	12	20	60	53
A	12 1	21	61	13	R	11 9	51	51	34	- †	11	40	40	54
B	12 2	22	62	14	S	0 2	62	22	35	* †	11 4-8	54	54	55
C	12 3	23	63	15	T	0 3	63	23	36	/ †	0 1	61	21	56
D	12 4	24	64	16	U	0 4	64	24	37	$ †	11 3-8	53	53	57
E	12 5	25	65	17	V	0 5	65	25	40	, †	0 3-8	73	33	60
F	12 6	26	66	20						.	12 3-8	33	73	61

† : Character may not be used in a symbol.

✱ : Character is not normally used. When it is used it will not be considered a sign.

SQUOZE OPERATION CODES

I. 5-bit Operation Codes

Octal	Mnemonic	Octal	Mnemonic	Octal	Mnemonic	Octal	Mnemonic
0	ACL	10	FMP	20	STO	30	TPL
1	ADD	11	FSB	21	STQ	31	TRA
2	AXT	12	LDQ	22	SXA	32	TSX
3	CAL	13	LXA	23	SXD	33	TXH
4	CLA	14	LXD	24	TIX	34	TXI
5	CLS	15	PSE†	25	TMI	35	TXL
6	FAD	16	PZE	26	TNX	36	TZE
7	FDP	17	STA	27	TNZ	37	XCA

†: Included in this category are all the 709 instructions which have +0760 in the upper twelve bits and use the address as part of the operation code, i.e., the instructions

BTT	ENK	PSE	SPR
CFF	ETM	RCT	SPT
CHS	FRN	RND	SPU
CLM	IOT	SLF	SSP
COM	LBT	SLN	SWT
DCT			

II. 9-bit Codes

Octal	Mnemonic	Octal	Mnemonic	Octal	Mnemonic	Octal	Mnemonic
40	ADM	64	FOR	110	LCHD	134	OAI
41	ALS	65	FSM	111	LCHE	135	OFT
42	ANA	66	FVE	112	LCHF	136	ONT
43	ANS	67	HPR	113	LDA	137	ORA
44	ARS	70	HTR	114	LDC	140	ORS
45	AXC	71	IIA	115	LDI	141	OSI
46	BSF	72	IIL	116	LFT	142	PAC
47	BSR	73	IIR	117	LGL	143	PAI
50	CAD	74	IIS	120	LGR	144	PAX
51	CAQ	75	IOCD(N)	121	LLS	145	PDC
52	CAS	76	IOCP(N)	122	LNT	146	PDX
53	CPY	77	IOCT(N)	123	LRS	147	PIA
54	CRQ	100	IORP(N)	124	MON	150	PON
55	CVR	101	IORT(N)	125	MPR	151	PTH
56	DVH	102	IOSP(N)	126	MPY	152	PTW
57	DVP	103	LAC	127	MTH	153	PXA
60	ERA	104	LAS	130	MTW	154	PXD
61	MSE @	105	LCHA	131	MZE	155	RDS #
62	FAM	106	LCHB	132	NOP	156	REW
63	FDH	107	LCHC	133	NZT	157	RFT

160	RIA	213	SVN	246	TQP	301	RCHF
161	RIL	214	TCH	247	TRCA	302	///(invalid code)
162	RIR	215	TCNA	250	TRCB	303	TQO
163	RIS	216	TCNB	251	TRCC	304	LCHG
164	RNT	217	TCNC	252	TRCD	305	LCHH
165	RQL	220	TCND	253	TRCE	306	RCHG
166	SBM	221	TCNE	254	TRCF	307	RCHH
167	SCHA	222	TCNF	255	TTR	310	SCHG
170	SCHB	223	TCOA	256	ENB	311	SCHH
171	SCHC	224	TCOB	257	UAM	312	TCNG
172	SCHD	225	TCOC	260	UFA	313	TCNH
173	SCHE	226	TCOD	261	UFM	314	TCOG
174	SCHF	227	TCOE	262	UFS	315	TCOH
175	SIL	230	TCOF	263	USM	316	TEFG
176	SIR	231	TEFA	264	VDH	317	TEFH
177	SIX	232	TEFB	265	VDP	320	TRCG
200	SLQ	233	TEFC	266	VLM	321	TRCH
201	SLW	234	TEFD	267	WEF	322	SDN
202	STD	235	TEFE	270	XCL	323	RUN
203	STI	236	TEFF	271	ZET	324	EAD
204	STL	237	TIF	272	IOST(N)	325	EDP
205	STP	240	TIO	273	blank	326	ELD
206	STR	241	TLQ	274	RCHA	327	EMP
207	WRS *	242	TNO	275	RCHB	330	ESB
210	STT	243	XEC	276	RCHC	331	EST
211	STZ	244	ESNT	277	RCHD	332	EUA
212	SUB	245	TOV	300	RCHE		

NOTES:

@ : This category includes all 709 instructions which have -0760 as their upper twelve bits and use the address as part of the operation. These are:

ECTM	LTM
EFTM	MSE
ESTM	PBT
ETT	SLT
LFTM	SSM
LSNM	

: This code is used for the following extended operations for Read Select:

RCD	RTB
RDR	RTD
RPR	

* : This code is used for the following extended operations for Write Select:

WDR	WTB
WPB	WTD
WPD	WTV
WPU	

SQUOZE DECK FORMAT

CHAPTER 1: GENERAL ARRANGEMENT

The SQUOZE deck produced by SOS is divided as follows and punched in the order given:

Preface
Heading Table
Macro-instruction Name Table
Blank Card
Macro-instruction Skeletons
Introduction
Dictionary
Footnotes
Text Without Commentary
Text With Commentary

This Appendix only describes the SQUOZE deck as produced by SOS, the actual structure of the deck when used is given in Section 08: IB Monitor or Section 09: SHARE Monitor.

Each card in the deck (except the blank card appearing between the Macro-instruction Name Table and the Macro-instruction Skeletons) has word 1 (9-left in row binary cards, and columns 1-3 in columnar binary cards) punched as follows:

Bits	Contents
S	1
1-5	Count of data words in this card.
6-8	High order bits of the sequence number of this card.
9-11	101
12-23	Low order bits of the sequence number of this card.
24-35	Logical check sum of all words (except bits 24-35 of word 1) contained in this card.

The remaining 23 words of the cards may be punched with up to 23 data words. The manner in which these words are punched varies from section to section and is described in the following chapters.

SQUOZE DECK FORMAT

CHAPTER 2: PREFACE

This section always consists on one and only one card. The card contains a summary of the sizes of the following sections and is used as a basis for the allocations of storage, etc.

The Preface card is punched:

Data Word	Bit Positions Used	Contents
1	all	First six BCD characters of the program identification (taken from the JOB card, see description of the Monitor program used).
2	all	Second six BCD characters of the program identification.
3	S, 1- 2	Unused.
	3-17	Number of words in the Introduction.
	18-20	Unused.
	21-35	Number of words in the Heading Table.
4	S, 1- 2	Unused.
	3-17	Number of words in the Macro-instruction Name Table.
	18-20	Unused.
	21-35	Number of words in the longest programmer macro-instruction skeleton.
5	all	Number of Dictionary entries.
6	S, 1- 2	Unused.
	3-17	Location of the END instruction relative to the first item in the Dictionary.
	18-20	Unused.
	21-35	Number of words in the Footnotes section.
7	S, 1- 2	Unused.
	3-17	Number of bits required for dictionary references.
	18-35	Number of words of Text With Commentary.

8	S, 1- 2	Unused.
	3-17	Highest alter number used in program.
	18-35	Number of words of Text Without Commentary.
9	all	Total number of words in all programmer macro-instruction skeletons.
10	all	Date of compilation (taken from the DATE card; see the section on the Monitor used).
11-23	all	Unused.

SQUOZE DECK FORMAT

CHAPTER 3: HEADING TABLE

This section is included only when heading characters are used in a program. When supplied with the SQUOZE deck, the table begins in a new card, and uses as many cards as are necessary to contain the table. The heading characters included in the Heading Table are punched in the data words of SQUOZE cards in the order of their appearances in the program. The first data word in the table, when supplied, is always punched with zeros; the remaining words required for the table are punched as follows:

Bit Positions Used	Contents
S, 1- 2	Unused.
3-17	Alter number of HEAD instruction.
18-20	Unused.
21-35	Base 50 representation of heading character (see Appendix 1).

SQUOZE DECK FORMAT

CHAPTER 4: MACRO-INSTRUCTION NAME TABLE

This section is only punched when the program uses programmer macro-instructions. The table is punched beginning on a new card, and uses as many cards as are necessary. Two consecutive data words are used for each name in the table. The first data word for each entry contains the name of the macro-instruction in BCD representation; the name is left-justified and the unused low order positions are filled in with zeros. The names in the table are arranged in the order in which they are encountered.

The second data word for each entry contains the following information:

Bit Positions Used	Contents
S, 1- 2 18-20	Number of parameters in the macro-instruction skeleton.
3-17	Number of words in the macro-instruction skeleton.
21-35	The relative position, in the Macro-instruction Skeleton Table, of the first word of the first macro-instruction skeleton. (The first word of the table is numbered zero, and the remaining words are numbered sequentially.)

SQUOZE DECK FORMAT

CHAPTER 5: BLANK CARD

The blank card included in the SQUOZE deck is always supplied by SOS. This card is unnumbered and no sequence number is reserved for it. The card is supplied to indicate the place at which modifications are inserted, must not be removed from the deck, and must follow modification cards.

SQUOZE DECK FORMAT

CHAPTER 6: MACRO-INSTRUCTION SKELETON TABLE

This section is punched only when programmer macro-instructions are used in the program. This section starts on a new card, and occupies as many cards as are necessary. Each macro-instruction skeleton uses a continuous string of bits, and each string begins in a new word. A skeleton is punched as follows:

Number of Bits Used	Contents	Comments
1	0	Beginning of new instruction.
1 or 2	0	No location symbol.
	10	Location symbol is present and is not a parameter of the macro-instruction.
	11	Location symbol is a parameter.
0, 5, or 36	Location symbol (36-bits)	This is the BCD representation of the symbol.
	Parameter number (5-bits)	Parameters are numbered beginning with 0.
1	0	Operation code is not a parameter.
	1	Operation code is a parameter.
5 or 6	Parameter number (5-bits)	Parameter number of operation code.
	Count of characters in operation code (6-bits)	The number of BCD characters in the operation code.
XX	Operation code	The number of bits used in six times the character count.

1	0	Instruction is indirectly addressed.
	1	Instruction is not indirectly addressed.
XX	Variable field	The variable field may contain any number of subfields, each of which has the form $T_1R_1T_2R_2 \ldots T_nR_nT_{n+1}$, where T and R are described below.
1	1	End of skeleton.
	0	Beginning of a new instruction. (This is the same bit as described first in this list.)
XX	Zeros	This field fills unused bits of the last word used for the skeleton.

Variable subfields:

4	T	This is a connector between two items in a subfield. All connectors are included, even leading plus signs which were omitted when the skeleton was coded. Leading plus signs are generally deleted when the SQUOZE deck is decoded.

If T is:	the connector is:
0000	+
0001	-
0010	,
0011	/ (VFD separator)
0100	*
0101	/ (division sign)
1001	$
1010	blank

The trailing T in a subfield is a blank, comma or VFD separator to terminate the subfield. Following a comma or a VFD separator, another subfield is begun. The blank terminates the instruction.

4, 6, or XX	R	This field is punched as indicated below.

R:

1 or 2	1	Entry is a parameter.
	00	Entry is a symbol or constant.
	01	Entry is a system macro-instruction name.

2, 4, or 5	Parameter number (5-bits)	(If preceding field is 1.) Omit next field.
	System macro-instruction name table reference (2-bits)	(If preceding field is 01.) Omit next field.
	Count of characters in symbol or constant (4-bits)	The number of BCD characters in the symbol or constant.
XX	Symbol or constant (BCD form)	The number of bits used in six times the character count.

No provisions are available for the inclusion of comments or remarks in the skeleton.

SQUOZE DECK FORMAT

CHAPTER 7: INTRODUCTION

The Introduction is only supplied with the SQUOZE deck when generative pseudo-instructions and/or commentary cards, i.e., remarks cards and listing pseudo-operations, are included in the program. The section starts on a new card and uses as many cards as are necessary.

Data words in this section contain entries in the order in which they are encountered.

The information for each entry in this section is punched as follows:

I. For generative pseudo-operations:

Bit Positions Used	Contents
S	0
1- 2	Unused.
3-17	Alter number of the instruction.
18-20	Count of the commentary cards which appear at the end of the set of machine words generated by the pseudo-instruction. As many as seven are permitted, and the number can be non-zero for macro-instructions only.
21-35	(n-1), where n is the number of words generated by the instruction.

II. For Commentary cards:

Bit Positions Used	Contents
S	1
1- 2	Unused.
3-17	Alter number of the commentary card or of the first of a consecutive set of such cards.
18-20	Unused.
21-35	Number of consecutive commentary cards.

SQUOZE DECK FORMAT

CHAPTER 8: DICTIONARY

A Dictionary is punched for all programs. The section occupies as many cards as necessary. The Dictionary is divided into two halves each beginning on a new card. A word in each half is used for every item. The words in the first half of the Dictionary are called "first word entries" and those in the second half "second word entries." The first word entries are arranged in ascending order according to bits 1-35; the second word entries are arranged in the same order as the corresponding first word entries.

The first two entries in each half of the Dictionary are used for special purposes. The first entry (number 0) is the reference for the "*" which means "the contents of the location counter." Bits 3-17 of the second word entry for this item contain the number of the item which was encountered first in the program (the remainder of the word is zero). The second entry (number 1) is the reference for invalid symbols. Subsequent entries are numbered beginning with 2.

First Word Entries

Two types of first word entries may appear in the Dictionary:

I. For location symbols and the pseudo-operations SYN, EQU, and BOOL:

Bit Positions Used	Contents
S	0 - Entry is a symbol which is the location symbol of a machine instruction or generative pseudo-instruction. 1 - Entry is for SYN, EQU, or BOOL instruction, or for a symbol associated with BES, BSS, END, ORG, or TCD.
1	0 - Symbol which follows is six characters long. 1 - Symbol is fewer than six characters in length.
2-35	Base 50 representation of the symbol with heading character. (For SYN, EQU, and BOOL entries this is the location symbol of the instruction.) The base 50 representation of a symbol is obtained as follows: a. If the symbol has fewer than five characters, it is headed (by blank if it is in an unheaded region). b. The symbol with it heading character is left-justified and any unused low-order positions are filled with blanks.

c. Each character in the symbol is replaced by it base 50 equivalent.
d. The result is then converted by the following: if the symbol, after each character is repaced by its base 50 equivalent, is ABCDEF, its base 50 representation is $(A*50^2+B*50+C)*2^{17}+(D*50^2+E*50+F)$.

II. For the pseudo-operations ORG, BSS, BES, HEAD, TCD, and END:

Bit Positions Used	Contents
S	1
1- 8	Type code: 372_8 ORG; 373_8 BES; 374_8 BSS; 375_8 HEAD; 376_8 TCD; 377_8 END
9-20	Unused.
21-22	00
23-35	These bits indicate the position in the Dictionary of the next item encountered during the scanning of the program. Thus, they indicate the order in which the Dictionary entries were made. However, the entry for END will have the base 50 representation of the current heading character in these bits.

Second Word Entries

There are five types of second word entries which may appear in the Dictionary.

I. For location symbols of machine instructions:

Bit Positions Used	Contents
S	0 - Symbol is defined. 1 - Symbol is undefined.
1	Unused.
2	0 - Symbol is not exempt. 1 - Symbol is exempt.

3- 4	Unused.
5-17	These bits indicate the position in the Dictionary of the entry for the next item for the program.
18-20	Debugging code.
21-35	Separation count, i. e., 1 plus the sum of the number of machine instructions, principal pseudo-operations, and instructions generated by generative pseudo-operations encountered in the program prior to this instruction.

II. For location symbols of principal pseudo-instructions ORG, BES, BSS, TCD, and END:

Bit Positions Used	Contents
S	0 - Symbol is defined. 1 - Symbol is undefined.
1- 4	0010
5-17	Location in the Dictionary of the corresponding pseudo-operation entry (see III below).
18-20	Debugging code.
21-35	Separation count.

III. For operations of principal pseudo-instructions ORG, BSS, BES, TCD and END:

Bit Positions Used	Contents
S	0 - Instruction not removed by a modification. 1 - Instruction removed by a modification.
1- 4	0000
5-17	The word of the Footnotes section containing the first 18-bit unit of the footnote for this item. (The words in the Footnotes are numbered consecutively beginning with 0.)
18-20	Unused.
21-35	Separation count.

IV. For HEAD pseudo-operation:

Bit Positions Used	Contents
S	0 - Instruction not removed. 1 - Instruction removed by a modification.
1-11	Unused.
12-17	Base 50 representation of heading character.
18-20	Unused.
21-35	Separation count.

V. For SYN, EQU, and BOOL pseudo-operations:

Bit Positions Used	Contents
S	0 - Defined symbol in variable field. 1 - Undefined symbol in variable field.
1	0 - Pseudo-operation is EQU or SYN. 1 - Pseudo-operation is BOOL.
2- 4	000
5-17	The position in the Footnotes section of the word containing the first 18-bit unit of the footnote for this item.
18-20	Debugging code.
21-35	Separation count.

VI. The second word entry for multiply defined symbols is always $(7000\ 000\ 700\ 000)_8$ or $(600\ 000\ 700\ 000)_8$.

VII. There are n+1 entries for a symbol defined n times in a program (n>1). The dummy entry (the one with the second word entry $700\ 000\ 700\ 000_8$ or $600\ 000\ 700\ 000_8$) will appear in the normal position, i. e., in the sorted list. Reference in the Text or Footnotes to a multiply defined symbol will be to this entry. The remaining n dictionary entries appear after the entry for the END card. The entries are referred to in the ordinary way within the dictionary. That is, the reference in one entry to a next item which is multiply defined will be to an item following the entry END, not to the dummy entry.

SQUOZE DECK FORMAT

CHAPTER 9: FOOTNOTES

This section begins on a new card and uses as many cards as are necessary. The footnotes are arranged in the order in which they were developed by SOS. Each footnote occupies as many bits as are required; and each new footnote begins in a new data word. Footnotes are punched as follows:

Bits S, 1-17 of the first data word used may be punched in either of two ways depending on the pseudo-operation which corresponds to the footnote:

I. ORG, BSS, BES, TCD, and END:

Bit Positions Used	Contents
S	1 - beginning of new footnote.
1	0 - no symbol is associated with the pseudo-operation. 1 - a symbol is associated with the pseudo-operation.
2- 4	000
5-17	If bit 1 is 0: Unused. If bit 1 is 1: The position in the Dictionary of the location symbol of the instruction.

II. SYN, EQU, and BOOL:

Bit Positions Used	Contents
S	1 - beginning of a new footnote.
1	Unused.
2	0 - pseudo-operation is SYN or EQU. 1 - pseudo-operation is BOOL.
3- 4	Unused.
5-17	The position of the next entry made in the Dictionary.

The remainder of the footnote is divided into 18-bit units which are punched as follows:

Bit Positions Used	Contents
0	0 - Continuation of footnote (carried at the beginning of a word only).
1- 3	Connector: 000 + 001 - 010 End of footnote 011 $ 100 * 101 / 110 111 Unused
	If the connector is 011 ($) bits 4-17 contain the base 50 representation of the heading character.
	If the connector is 010 (end of footnote) bits 4-17 contain zeros. Note that if the connector would fall in bit positions 1-3, the end of the footnote is indicated by the beginning of the next footnote.
	When the connector is +, -, *, or /, bits 4-17 are used as indicated below.
4	0 - Constant follows, succeeding bits are used as indicated below. 1 - Dictionary reference follows (in bits 5-17).
5	0 - 12-bit constant follows. 1 - 18- or 35-bit constant follows.
6	If bit 5 is 0, this is the high-order bit of a 12-bit constant. If bit 5 is 1: 0 - 18-bit constant follows 1 - 35-bit constant follows
7-17	If constant is 12 bits long, these are the 11 low-order bits. If constant is 35 bits long, these bits are unused.

When a constant is 18 or 35 bits long and the preceding connector is in bit positions 19-21, the constant is carried in the following word which is punched:

Bit Positions Used	Contents
S	0 - continuation of previous footnote.
1-18	18-bit constant or high-order bits of 35-bit constant
19-35	Low-order bits of 35-bit constant, or next connector (in this case the continuation bit is not used).

If the preceding connector is in bit positions 1-3, an 18-bit constant will appear in bit positions 18-35.

If a footnote requires more than 36 bits, succeeding 18-bit units are used.

SQUOZE DECK FORMAT

CHAPTER 10: TEXT

This section is supplied in two forms; one with commentary and one without commentary. Both sections are divided into groups of 230 data words. Each group appears in the order in which they appear in the program; however, within each group, the data words appear in inverse order.

The non-commentary text differs from the commentary text in the following respects:

1. where a single bit is used (see below) to indicate whether comments do or do not follow, the bit is always zero in the non-commentary text and the comment field is omitted.

2. when a character count is used instead of a single bit, the count will be zero; the comments are again omitted.

3. information in the variable and comments fields of DEC, DUP, LBR, SQZ, macro-instructions, ETC following a macro, and listing pseudo-operations are omitted.

4. remarks are omitted.

5. The operation fields of macro-instructions are omitted.

Note that in all cases, the alter numbers are the same in both commentary and non-commentary texts.

The number of data words required for the text of an instruction varies according to the information included with the instruction, so that the text can only be thought of as strings of bits. Within a string, the bits have the following significance in the order of their appearance.

Number of Bits Used	Contents	Comments
1	0	Instruction follows. (See I below.)
	1	Special item follows. (See II below.)
I. Instructions:		
1	0	Five bit operation code.
	1	Nine bit operation code.
1	0	Instruction not indirectly addressed.
	1	Instruction indirectly addressed.

5 or 9	Operation code	(SQUOZE code for operation)
2	00	Simple address; dictionary reference or constant follows. (See Note 1, page 12.03.10.07.)
	01	Complex address; expression follows. (See Note 2, page 12.03.10.07.)
	10	Relative address; expression follows. (See Note 3, page 12.03.10.07.)
	11	No address; next field is omitted.
0 or XX	Address of dictionary reference	Address sizes depend on the form, and the dictionary reference size. (See Notes 1-3, page 12.03.10.07.)
1 or 2	1	No tag; next field is omitted.
	00	Absolute tag; tag follows.
	01	Complex tag; expression follows. (See Note 2, page 12.03.10.07.)
0, 3 or XX	tag	
1 or 2	1	No decrement; next three fields are omitted.
	00	Simple decrement field; decrement follows. (See Note 1, page 12.03.10.07.)
	01	Complex decrement field; decrement follows; (See Note 2, page 12.03.10.07.)
0 or 1	0	Decrement field contains a count for variable length arithmetic or convert instruction.
	1	Decrement field is not a count.
0 or XX	Decrement or count	Decrement sizes depend on the form and the dictionary reference size. (See Notes 1-3, page 12.03.10.07.)
0 or 1	0	Sign of decrement is +.
	1	Sign of decrement is -.
1	0	No comments are associated with the instruction
	1	Comments are present, and follow.
0 or 1	0	Comment begins in the same column as the first comment in the program.
	1	Comment does not begin in the same column as the first comment in program.

	0 or 6	Count of Characters in comments	
	0 or XX	Comments (BCD form)	The number of bits used for this entry is six times the character count.

II Special Item:

2	00	Principal pseudo-operation. (See II. A. below.)
	01	Commentary. (See II. B. below.)
	11	Data. (See II. C. below.)
	10	Control (See II. D. below).

A. Principal Pseudo-operations:

1	0	Operation is ORG, BSS, or BES.
	1	Operation is EQU, SYN, or BOOL.
XX	Dictionary Reference	The number of bits used, XX, is specified in data word 7, bits 3-17 of the Preface.
1	0	No comments are associated with the item.
	1	Comments are present and follow.
0 or 1	0	Comment begins in the same column as the first comment in the program.
	1	Comment does not begin in the same column as the first comment in the program.
0 or 6	Count of characters in comments	
0 or XX	Comments (BCD form)	The number of bits used for this entry is six times the character count.

B. Commentary:

1	0	Pseudo-operation. (See II. B. (1) below.)
	1	Remarks. (See II. B. (2) below.)

(1) Pseudo-operations:

3	000	ETC
	001	DUP
	010	Macro-instruction
	011	LBR
	100	DEC
	101	OCT
	110	BCI
	111	SQZ.

7	Count of characters in variable field and/or comments	Number of BCD characters (not carried with OCT and BCI, see Note 3, Page 12.03.10.07). In the case of macro-instructions, the operation code is also included.
XX	Variable field and/or comments (BCD form)	The number of bits used for this entry is six times the character count.

(2) Remarks:

3	000	LIST
	001	UNLIST
	010	DETAIL
	011	TITLE
	100	SPACE
	101	EJECT
	110	Unused
	111	Remarks
7	Count of characters in variable field and/or comments	If entry is 000000, no variable field follows.
XX	Variable field and/or comments (BCD form)	The number of bits used for this entry is six times the character count.

C. Data:

36	Data item	One entry is made for pseudo-operation and each data word of an OCT, BCI and DEC instruction (See Note 4, page 12.03.10.07.)
1	0	No comments are associated with the item.
	1	Comments are present and follow.
0 or 1	0	Comment begins in the same column as the first comment in the program.
	1	Comment does not begin in the same column as the first comment in the program.
0 or 6	Character count of comments	This information is associated with only the last item from an OCT, BCI or DEC card.

0 or XX	Comments (BCD form)	The number of bits used for this entry is six times the character count.

D. Control:

2	00	End of modifications (used only by Modify and Load).
	10	TCD. (See II. D. (1) below.)
	11	END. (See II. D. (2) below.)
	01	Other. (See II. D. (3) below.)

(1) TCD:

XX	Dictionary reference	The number of bits used for this entry is specified in data word 7, bits 3-17 of the Preface.
1	0	No comments are associated with the item.
	1	Comments are present and follow.
0 or 1	0	Comment begins in the same column as the first comment in the program.
	1	Comment does not begin in the same column as the first comment in the program.
0 or 6	Character count of comments	
0 or XX	Comments (BCD form)	The number of bits used for this entry is six times the character count.

(2) END:

XX	Dictionary reference	The number of bits used for this entry is specified in data word 7, bits 3-17 of the Preface.
8	Residue of synchronization count	256-x, where x is the number of entries, including this one, which have been made since the last synchronization count. (See (3) below.)
1	0	No comments are associated with the item.
	1	Comments are present and follow.
0 or 1	0	Comment begins in the same column as the first comment in the program.
	1	Comment does not begin in the same column as the first comment in the program.

0 or 6	Character count of comments	
0 or XX	Comments (BCD form)	The number of bits used for this entry is six times the character count.

(3) Other:

3	000	Synchronization count (every 256th entry).
	001	Heading. (See II. D. (3)(a) below.)
	010	VFD. (See II. D. (3)(b) below.)
	011	Reserved for expansion of the system.
	100	
	101	
	110	
	111	

(a) HEAD:

6	Heading character	Base 50 representation of heading character.
1	0	No comments are associated with the item.
	1	Comments are present and follow.
0 or 1	0	Comment begins in the same column as the first comment in the program.
	1	Comment does not begin in the same column as the first comment in the program.
0 or 6	Character count of comments	
0 or XX	Comments	The number of bits used for this entry is six times the character count.

(b) VFD:

XX	Variable field	The number of bits used for this entry depends on the length of the variable field and the size required for dictionary references. The entry is subdivided into fields as shown in II. D. (3)(b) i. below. The format is: $K_1L_1F_1K_2L_2F_2 \ldots K_nL_nF_nK_{n+1}$

i. Variable field subdivisions:

2	K	Field type: 00 - Complex field follows. (See Note 1, below.) 01 - end of VFD instruction. 10 - Hollerith field follows. 11 - Octal field follows.
6	L	Bit count of field which follows.
L	F	VFD subfield.

Note 1: A simple field, R, is a dictionary reference if the first bit is 1. The number of bits used for the reference is specified in data word 7, bits 3-17 of the Preface. R is a constant if the first bit is 0. If the next two bits are:
00 - a four-bit constant follows.
01 - a twelve-bit constant follows.
10 - an eighteen-bit constant follows.
11 - a thirty--five-bit constant follows.

Note 2: Complex fields have the form $T_1R_1T_2R_2 \ldots T_nR_nT_{n+1}$; where T represents the type of connector used in a complex field; and R is used to represent dictionary references or constants shown in Note 1. Exception: If T = 011 ($), the R which follows is the 6-bit, base 50 heading character which preceded the $ in the original symbolic field.

If T is	000	the connector is +
	001	" -
	010	" End of field
	011	" $
	100	" *
	101	" /
	110	Unused
	111	

Note 3: Relative fields have the form R_1R_2S; where R is as shown above and S is used to represent a sign in a relative expression; S always occupies one bit only. If S is 0, the sign is plus (+); if S is 1 the sign is minus (-).

Note 4: For OCT, BCI, and DEC instructions, n+1 entries are made. For BCI, n is defined by the first subfield of the instruction. For OCT and DEC, n is the number of subfields in the variable field. The first entry is for the pseudo-operation, the remaining entries are for the n data words. If comments are also included with any one of these instructions, only the n+1 th entry will include the comments.

Examples

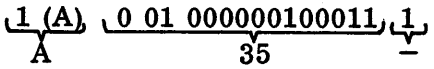
The complex field A+B/C*D+35, where (A), (B), (C), (D) represent the positions in the dictionary of the entry for A, B, C, and D, respectively, would appear in the text as:

000 1 (A) 000 1 (B) 101 1 (C) 100 1 (D) 000 0 01 000000100011 010

Taking the bits in order, they have the following significance:

000	+ (the signs are always explicitly given).
1	Dictionary reference follows.
(A)	n-bit Dictionary reference for A.
000	+
1	Dictionary reference follows.
(B)	n-bit Dictionary reference for B.
101	/
1	Dictionary reference follows.
(C)	n-bit Dictionary reference for C.
100	*
1	Dictionary reference follows.
(D)	n-bit Dictionary reference for D.
000	+
0	Constant follows.
01	Constant is 12-bits long.
000000100011	35
010	end of field

The relative field A-35, on the other hand, would appear as



32K IB MONITOR OPERATING NOTES

CHAPTER 1: EQUIPMENT REQUIREMENTS

The use of the 32K IB Monitor with SOS requires the availability of the following system components:

A. IBM 711 Card Reader

B. IBM 716 Printer

C. IBM 721 Card Punch

D. Three to seven tape units:

1. Unit A1. This unit is used for the SOS system tape and is, of course, always required.

2. Unit A2. Output for off-line listing is written on this tape. Hence, the unit need only be available when there is to be such output.

3. Unit A3 is used for two purposes:
   a. If Sense Switch 1 is Up, all input is assumed to have been written on tape by off-line card-to-tape operations, and is read from this tape.
   b. When a SQUOZE deck is to be read on-line in columnar binary form, this tape is used for intermediate storage of input if insufficient storage space is not otherwise available.

4. Unit A4. If the library tape is required for the processing of a job deck, it must be on this unit. When the library tape is not required, this unit need not be readied.

5. Unit A5 is used when SQUOZE or absolute binary output is to be punched off-line. The unit need only be readied when there is to be such output.

6. Unit B1 must always be readied and is used as a work tape for input by SOS.

7. Unit B2 must also always be readied. This tape is used by SOS for a working tape for output.

32K IB MONITOR OPERATING NOTES

CHAPTER 2: OPERATING INSTRUCTIONS AND PROGRAMMED HALTS

A. Ready the required tape units as outlined above (only tape A1 need be rewound).

B. Place either the SHARE 1 or SHARE 2 control panel in the printer.

C. If input is to be read from cards, ready job decks in the card reader.

D. Set Sense Switches for the desired options as follows:

Sense Switch	Setting	Option
1	UP	Input is to be read from tape (A3 must be readied).
	DOWN	Input is to be read from cards (card reader must be readied with input).
2	Presently unused.	
3	UP	Only monitor control cards and error statements are to be printed on-line.
	DOWN	All printing is to be done on-line.
4	UP	Normal procedure is to be used when loading the system.
	DOWN	After loading the system, the debugging output from the preceding job is to be printed. This feature makes it possible to recover debugging output when the previous job affected the operation of SOS and had to be terminated.
5	UP	All information for printing, including monitor control cards and error statements, is to be written on tape A2 for off-line tape-to-printer operations. (Tape A2 must, of course, be readied.)
	DOWN	No information is to be written on tape A2.

6	UP	All card output is to be written on tape for off-line tape-to-columnar-binary-card operation, regardless of whether control cards specify row binary or columnar binary output.
	DOWN	Card output is to be punched on-line in either row binary or columnar binary as specified in the control cards.

Note: If both Sense Switches 3 and 5 are Up during Modify and Load processing, information for printing will be written on tape A2 and not listed on-line.

E. Depress the Clear key and then the Load Tape key.

HALTS

There are four programmed halts which may occur during processing of a job deck:

Location counter (octal)	Explanation
1370	A STOP card has been encountered. If Sense Switch 1 is Up, tape A3 has been rewound. If printing is to be done off-line, an end-of-file has been written on tape A2 and the tape rewound. Similarly, if output decks are to be punched off-line, an end-of-file has been written on tape A5, and the tape rewound. Tape A1 is always rewound.
1402	A PAUSE card has been encountered. Depression of the Start key will cause processing to continue. OR An end-of-file condition has occurred while reading input from tape A3.
1746	An end-of-tape condition has occurred on tape A2.
2420	An end-of-tape condition has occurred on tape A5.

In each of the above cases, a message will be printed on-line indicating the condition which caused the halt. When any other condition arises, an error statement will be printed out (and, if Sense Switch 5 is UP, written on tape A2) indicating the condition, (see Appendix 11); when possible the Monitor will continue processing. If it is not possible to continue, the Monitor will terminate processing of the job deck and skip to the next deck.

SHARE MONITOR SYSTEM AND LIBRARY TAPE GENERATION AND UPDATING

In order to incorporate revisions to the various files of the System, the programmer responsible for the maintenance of SOS at his installation must be familiar with the use of the WST (Write System Tape) control card, the operation of the WST file, and the format of the System Tape.

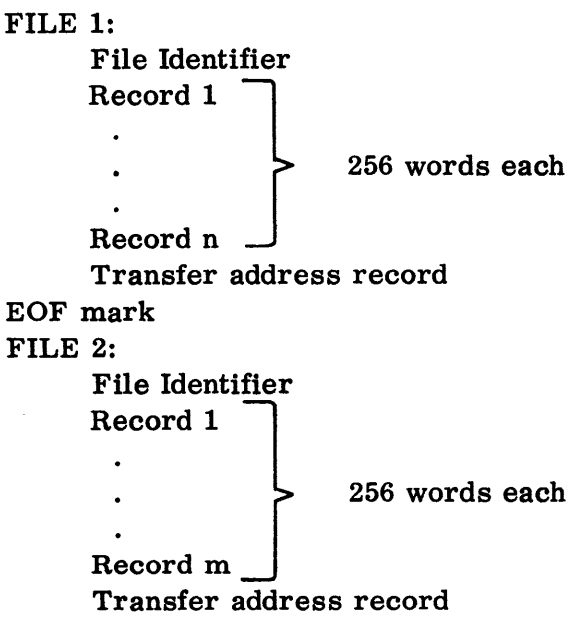
The SHARE Distribution Agency distributes changes either in the form of a modification package to the current SQUOZE deck of a particular file, or in the form of a new SQUOZE deck of that file. Therefore, installation-specific revisions should be accomplished by modifications to the current SQUOZE deck using CHANGE rather than ALTER in effecting such modifications.

The SQUOZE deck with its modifications must be converted to 709-type absolute binary before being written on the System Tape. Modify and Load can be used for this conversion.

System Tape Format

Each file on the System Tape begins with a one-word BCI file identifier for use by the System Tape Loader routine. The file identifier is followed by the absolute code comprising that file. The code is divided into 256-word records. The last record of a file is a one-word transfer address corresponding to the transfer card of the absolute binary deck, followed by a physical end-of-file mark.

The format of the System Tape is as follows:



EOF mark

. } files consisting of 256-word records
.
.

EOF mark

Terminal check sum file (File Identifier is $777\ 777\ 777\ 777_8$)

Use of System Tape Writer

The WST file is loaded into core storage whenever a WST control card is recognized, whether during Initiation or during Phase 1.

The following control cards are recognized by the System Tape Writer:

A. CHANGE X, Y (X and Y are BCI file identifiers)

The CHANGE card has two possible functions:

1. If the variable field is X, Y, the files on the System records from the beginning of X to the end of Y would be deleted.

   For example,

   CHANGE INTRAN, INTRAN

   would delete that file from SYSTAP. Any or all new files following that CHANGE card and preceding the next CHANGE or END card would be inserted at this point.

2. If the variable field contains only X, the new files following that CHANGE card and preceding the next CHANGE or END card will be inserted following the file identified by X.

B. CODE A

A is a file identifier of 1-6 BCI characters. The CODE control card is used to establish the file identifier (see above) for the subsequent absolute deck.

C. ROW

The ROW control card indicates that the absolute deck following is in row binary. The row binary deck must be followed by a blank card. No equivalent control card or blank card is required if the absolute deck is column binary.

D. LAST

The LAST control card causes the System Tape Writer to write an end-of-file mark on the new tape.

E. END

The END control card indicates that the last modification file has been read and that the remainder of the System Tape should be copied directly. The check sums written in the Terminal check sum file will be recomputed before being written.

Each of the five control cards described above is punched in the standard format. The exception is that the variable field, when required, must commence in column 16. In addition, column 1 of each card must contain 7-, 8-, 9-punches.

The System Tape Writer, as presently assembled, is capable of processing files of up to $44,000_8$ words in length. Files must be inserted and/or deleted in the order in which they appear on the input System Tape. The primary reason for this restriction is that duplicates of one or more files (using the same file identifier) may appear on the System Tape to reduce tape searching time. WST prints an ordered list of file identifiers for this purpose before the updating process is started.

Examples:

Three examples of the input deck format for System Tape modification are given below:

A. The purpose of this deck is to replace the current files of M1 and INTRAN with new versions:

```
WST
Blank
CHANGE M1, M1
CODE M1
ROW
New row binary absolute deck (with Transfer card)
Blank (used in conjunction with ROW. Note that this is omitted when
  the absolute deck is column binary.)
LAST
CHANGE INTRAN, INTRAN
CODE INTRAN
New column binary absolute deck
LAST
END
```

B. The purpose of this input deck is to replace the current file of M8, the Modify and Load Lister, with a new version and to insert the Output Editor on a tape that previously contained no version of the editor. For this example, it is assumed that the most logical position for the Output Editor is immediately following SNAPTRAN, the debugging translator.

```
WST
Blank
CHANGE M8, M8
CODE M8
Absolute deck of M8 in column binary
LAST
CHANGE SNPTRN
CODE OUTED  (OUTED is the file identifier of the Output Editor.)
ROW
Absolute deck of Output Editor in row binary
Blank
LAST
END
```

C. The following deck will reproduce a tape.

```
WST
Blank
END
```

Two methods can be used to perform a WST run from a peripheral input tape.

A. Arrange the deck as mentioned above.

B. Start the input tape with the CHANGE card of the first file to be changed (or inserted after), then CODE, etc. In this case the loader must have the GO card replaced with a WST card and a blank card following that. Sense Switch 2 will be Up.

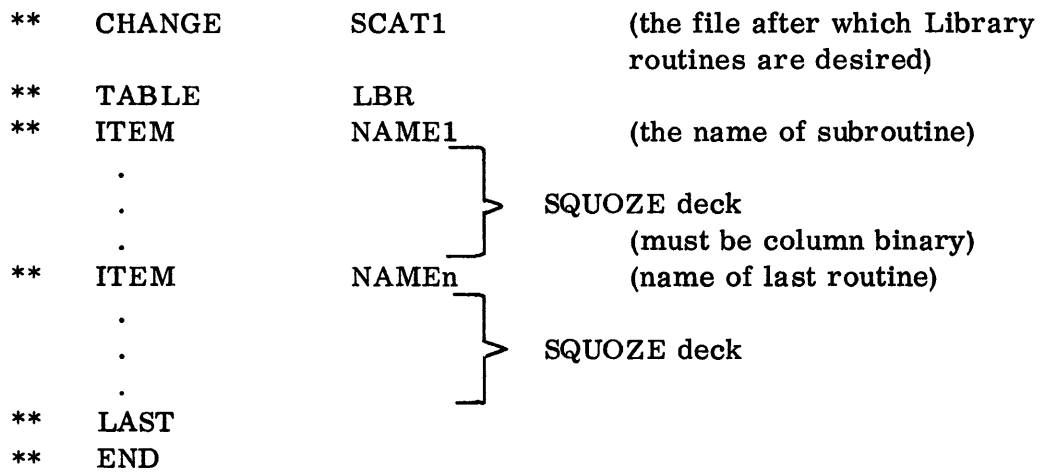
C. A JOB card can be used if the deck is to be placed on SYSPIT. Follow this with a WST card, a blank, and so on as prescribed. The standard loader should then be used with the GO card and Sense Switch 2 Up.

Naturally, only column binary cards can be read by the off-line card reader.

If the job is to be read on-line with Sense Switch 2 Down, the JOB card method should not be used. The proper method is to replace the GO card of the loader with a WST card, followed by a blank, CHANGE card, etc.

The following procedure is to be used to write a System Library tape as a file on SYSTAP or as a separate tape.

Set deck up as follows:



The procedure to write the Library file as a separate tape requires exactly the same deck setup as above, except that the CHANGE card is replaced by a NOTAPE card ** (NOTAPE is punched in columns 8-13). In this case, the library tape will be written on SYSES2, or unit B4 if there are no special assignments.

The job is run in the same way as a WST run, with the finished tape being denoted by an on-line message. In all cases, the message will indicate a completed SYSTAP, when in reality, if the Library tape is being written separately, the finished tape is not SYSTAP, but SYSLBR.

This may be run as mentioned above using either on-line or off-line input. The Library file must be in column binary form.

** 7-, 8-, and 9-punches required in column 1.

32K IB MONITOR OPERATING NOTES

CHAPTER 1: EQUIPMENT REQUIREMENTS

The use of the 32K IB Monitor with SOS requires the availability of the following system components:

A. IBM 711 Card Reader

B. IBM 716 Printer

C. IBM 721 Card Punch

D. Three to seven tape units:

1. Unit A1. This unit is used for the SOS system tape and is, of course, always required.

2. Unit A2. Output for off-line listing is written on this tape. Hence, the unit need only be available when there is to be such output.

3. Unit A3 is used for two purposes:
   a. If Sense Switch 1 is Up, all input is assumed to have been written on tape by off-line card-to-tape operations, and is read from this tape.
   b. When a SQUOZE deck is to be read on-line in columnar binary form, this tape is used for intermediate storage of input if insufficient storage space is not otherwise available.

4. Unit A4. If the library tape is required for the processing of a job deck, it must be on this unit. When the library tape is not required, this unit need not be readied.

5. Unit A5 is used when SQUOZE or absolute binary output is to be punched off-line. The unit need only be readied when there is to be such output.

6. Unit B1 must always be readied and is used as a work tape for input by SOS.

7. Unit B2 must also always be readied. This tape is used by SOS for a working tape for output.

32K IB MONITOR OPERATING NOTES

CHAPTER 2: OPERATING INSTRUCTIONS AND PROGRAMMED HALTS

A. Ready the required tape units as outlined above (only tape A1 need be rewound).

B. Place either the SHARE 1 or SHARE 2 control panel in the printer.

C. If input is to be read from cards, ready job decks in the card reader.

D. Set Sense Switches for the desired options as follows:

Sense Switch	Setting	Option
1	UP	Input is to be read from tape (A3 must be readied).
	DOWN	Input is to be read from cards (card reader must be readied with input).
2	Presently unused.	
3	UP	Only monitor control cards and error statements are to be printed on-line.
	DOWN	All printing is to be done on-line.
4	UP	Normal procedure is to be used when loading the system.
	DOWN	After loading the system, the debugging output from the preceding job is to be printed. This feature makes it possible to recover debugging output when the previous job affected the operation of SOS and had to be terminated.
5	UP	All information for printing, including monitor control cards and error statements, is to be written on tape A2 for off-line tape-to-printer operations. (Tape A2 must, of course, be readied.)
	DOWN	No information is to be written on tape A2.

6	UP	All card output is to be written on tape for off-line tape-to-columnar-binary-card operation, regardless of whether control cards specify row binary or columnar binary output.
	DOWN	Card output is to be punched on-line in either row binary or columnar binary as specified in the control cards.

Note: If both Sense Switches 3 and 5 are Up during Modify and Load processing, information for printing will be written on tape A2 and not listed on-line.

E. Depress the Clear key and then the Load Tape key.

HALTS

There are four programmed halts which may occur during processing of a job deck:

Location counter (octal)	Explanation
1370	A STOP card has been encountered. If Sense Switch 1 is Up, tape A3 has been rewound. If printing is to be done off-line, an end-of-file has been written on tape A2 and the tape rewound. Similarly, if output decks are to be punched off-line, an end-of-file has been written on tape A5, and the tape rewound. Tape A1 is always rewound.
1402	A PAUSE card has been encountered. Depression of the Start key will cause processing to continue. OR An end-of-file condition has occurred while reading input from tape A3.
1746	An end-of-tape condition has occurred on tape A2.
2420	An end-of-tape condition has occurred on tape A5.

In each of the above cases, a message will be printed on-line indicating the condition which caused the halt. When any other condition arises, an error statement will be printed out (and, if Sense Switch 5 is UP, written on tape A2) indicating the condition, (see Appendix 13); when possible the Monitor will continue processing. If it is not possible to continue, the Monitor will terminate processing of the job deck and skip to the next deck.

SHARE MONITOR OPERATING NOTES

CHAPTER 1: CONTROL CARDS

This chapter, and the remaining chapters of the appendix, are of interest to operators only. In the following material, knowledge of the information contained in Section 09, chapters 1, 2 and 3 is assumed. Therefore the reader of this appendix should have first read those chapters.

A. ASSIGN

An ASSIGN control card is employed to cause a change in the status or use of an I/O unit. It is placed before the GO control card (see below) in the loader. It may also be used between phases.

The format of an ASSIGN card is:

ASSIGN XN=Z

where X is an alphabetic channel designation (A through F)
N is a tape number (not used for card equipment)
Z is the assignment desired for the specified I/O unit.

Z = OFF to disconnect the unit and make it unavailable for use.
= ON to place the I/O unit in an "unassigned and available" status.
= SYSXXX, referring to the Communication Region control word for the symbolic unit named.

Examples:

1. ASSIGN C = SYSCRD

   causes the System to use the card reader on channel C for on-line input.

2. ASSIGN A5 = OFF

   will force the System to avoid the use of tape drive 5 on channel A for any purpose and to print an error message to the operator if an attempt is made to assign A5 for System or object program use.

3. ASSIGN A5 = ON

   will return the specified physical unit to available status.

4. ASSIGN B4 = SYSPOT

will cause tape B4 to be used as the Peripheral Output Tape during all subsequent phases which require it until reassignment or initialization.

Note: Reassignment of system tapes is permitted only at specific intervals. SYSTAP may be reassigned only during initiation, SYSMIT and SYSMOT between cycles (i. e., preceding execution of the input phase), and all other system tapes between phases.

B. DATE

A DATE control card may be placed before the GO card in the loader. The date is then available for use by the accounting routines, by the Compiler which places it in 4-left row of the Preface card of SQUOZE decks, and by the Lister which places it on each page of listings. The format of a DATE card is:

DATE M/D/Y

where M consists of 2 decimal digits specifying the month
D consists of 2 decimal digits specifying the day
Y consists of 2 decimal digits specifying the year.

C. GO

A GO card must be the last card in the loader. It indicates that all System control cards have been read, and causes the System to commence execution of Phase 1.

The format of this card is:

GO

No parameters are necessary.

D. END

An END card must be the last control card in a stack of jobs. It specifies that all the jobs to be executed in this cycle of operation have been read.

The format of this card is:

END

No parameters are necessary.

SHARE MONITOR OPERATING NOTES

CHAPTER 2: INPUT DECK ARRANGEMENT

The input deck consists of a stack of intermixed "Compiler" or "Modify and Load" jobs in any order. Each job deck will contain the program control cards necessary for that job. All program control cards must have 7-, 8-, and 9-punches in column one.

The Job Deck

Each job deck consists of a job card, a "Compiler" or "Modify and Load" deck, and possibly a data deck.

An END control card must follow the stack of jobs. This card must have 7-, 8-, and 9-punches in column one; and END punched in columns 8-10.

Note: If it appears that the end of the peripheral input tape will be reached, a symbolic deck for a SCAT job may be divided into two parts. The first part should have a card placed behind it which has ENDTAP punched in columns 8-13. No other punches should appear in the card. The remainder of the input decks may now be placed on another tape. This facility is not available for SQUOZE decks. The system will print a request to mount the second tape when it is needed.

The stack of jobs constituting an input deck is the same whether read on-line or off-line. However, row binary cards cannot be read by off-line card-to-tape equipment.

SHARE MONITOR OPERATING NOTES

CHAPTER 3: STARTING OPERATION

Operation of the SHARE Monitor System is started by means of a Loader deck. This deck may contain six or more cards. The last card of the deck is a GO card.

The system assumes a two channel machine with five tapes on each channel. Any variance in the number of tapes per channel will require the use of ASSIGN cards placed before the GO card in the Loader deck.

Before operation is begun, five tapes on each channel must be readied. The units should be dialed 1, 2, 3, 4, 5 on Channel A and 1, 2, 3, 4, 5, on Channel B.

The steps required for starting operation are given below. The standard addresses for the system tape and peripheral input tape are assumed.

A. Ready the System Tape on B1.
B. Ready the Peripheral Input Tape on B2.
C. Ready tapes on B3, B4, B5, A1, A2, A3, A4, A5.
D. Place Loader deck, containing any necessary ASSIGN cards, in card reader.
E. If input is from cards, place input deck following loader.
F. Ready card reader.
G. Set Sense Switches as indicated below.
H. Depress the Clear and the Load Cards keys.

After a period of initialization the printer will indicate that the system is entering Phase 1. A stop will then occur with the location counter containing 45_8. If no alteration of the tape assignments is necessary, the operator may continue by depressing the Start key.

As each job is started, the program control cards for that job will be printed. If Sense Switch 1 is Down, the identification of the file being read from the System Tape will also be printed on-line.

From time to time instructions concerning the readying or removal of tapes will also be printed. Operation may be continued, when these instructions have been carried out, by depressing the Start key.

If a request for tape assignment is made, the operator need only place the corresponding absolute address in the address portion of the Entry keys and depress the Start key. Tapes designated as free on the status list should be used to fill such requests. Any illegal assignments will result in a message and a stop for resetting the entry keys.

If any halt other than 45_8 occurs, transfer to location 46_8. If the system cannot continue, use the restart procedure (see page 12.13.05.01).

Sense Switch Settings

Sense Switch	Setting	Option
1	UP	Do not print file identifications on-line.
	DOWN	Print identifications on-line.
2	UP	Input is to be read from tape.
	DOWN	Input is to be read from cards.
3	UP	Write print-output on tape for off-line printing.
	DOWN	Write print-output on-line.
4	UP	Tape assignments changes are not required (see page 12.13.04.01).
	DOWN	Tape assignment changes are required.
5	UP	(Not used; must always be Up.)
6	UP	Write card output on tape for off-line punching.
	DOWN	Punch card output on-line.

SHARE MONITOR OPERATING NOTES

CHAPTER 4: SYSTEM TAPE REASSIGNMENT

The operator must be concerned with tape assignments at three different points in the processing of a stack of jobs.

A. During Initiation

ASSIGN cards may be placed in the loader to reassign the system tapes SYSPOT, SYSPPT, SYSPIT, SYSES2, SYSES1, SYSMIT, SYSMOT and SYSDOT. The assignments specified on these cards will hold throughout the entire cycle unless changed between phases.

B. Between Phases

The Monitor will halt after printing the necessary ready and remove messages and the tape status list. At this time, the assignment of some system tapes may be altered.

If the operator desires to change the assignment of a system tape from the card reader, he must:

1. Set Sense Switch 4 to Down.
2. Enter the prefix MTH (7) in the MQ entry keys.
3. Ready the appropriate ASSIGN cards in the reader.
4. Depress the Start key.

For example, to assign a new tape as SYSMOT the card would be:

```
ASSIGN    XN = SYSMOT
```

An end of file or an error will terminate processing of ASSIGN cards.

If the Monitor requests assignment of a specific tape, e.g.,

```
ASSIGN          SYSMOT
```

the operator may use the method outlined above, or the following:

1. Set Sense Switch 4 Down.
2. Enter the prefix PZE in the entry keys.
3. Enter the physical address of the tape to be assigned as SYSMOT, in the address position of the Entry keys.
4. Depress the Start key.

Note: SYSMIT may not be reassigned between phases.

If the operator wishes to change the output mode from on-line to off-line, or vice versa, he should:

1. Set Sense Switch 4 Down.
2. Place the prefix MON in the Entry keys.
3. Change the sense switches as desired.
4. Depress the Start key.

The necessary output changes will be made, a new tape status list will be printed, and the halt at location 45_8 will occur. Sense Switch 4 may now be set to Up and the Start key depressed to enter the phase.

C. Between Jobs

An ASSIGN card of the form ASSIGN XN = SYSXYN for each reserved or utility tape desired at execution time is placed between the JOB card and the SCAT or LOAD card.

At execution time, the system will print messages to ready the reserved and utility tapes for the next job to be executed. A halt will then occur. If the tape assignments described are insufficient or erroneous, the operator may correct them either by using ASSIGN cards in the card reader or by direct entry from the Entry keys. The type of change made will depend on the prefix in the Entry keys. In no case may the assignment of other than reserved or utility tapes be made at this time. The proper setting for the Entry keys is shown in the table below.

	KEY WORD			
FUNCTION	PREFIX	ADDRESS	DECREMENT	ANALOGOUS CARD
General Form	PFX	A (physical address)	B (symbolic designation as if it were physical)	-
Assign Reserved Tape	PON	A	B	ASSIGN XN = SYSYRM
Assign Utility Tape	PTW	A	B	ASSIGN XN = SYSYUM
*Assign System Tape	PZE	A	-	ASSIGN XN = SYSYYY
Make Physical unit available for new assignment.	MZE	A	-	ASSIGN XN = ON
Make Physical Unit Unavailable	PTH	A	-	ASSIGN XN = OFF
Process ASSIGN cards in card reader	MTH	-	-	-

* This entry is made only upon request by monitor.

Miscellaneous Notes

In some instances a program will use special tapes during execution. The person setting up the input deck must have information concerning the symbolic addresses of the tapes to be used and the phase in which the job will be executed. Any physical tape may be assigned for these symbolic addresses provided the tape is not already in use. There is no restriction concerning channel. This assignment is made with the use of an ASSIGN control card of the form

```
ASSIGN    XN = SYSYYY
```

where SYSYYY is the programmer's symbolic tape address such as SYSAR1, SYSBU4, etc.

XN is the tape to be used (X is the channel letter and N is the tape unit on that channel).

If the programmer is using a symbolic tape (i. e., a tape referred to by a symbolic name) as input, it will be necessary for the operator to mount that tape on the unit specified by the ASSIGN card.

For example, suppose that a programmer specifies that he needs tape SYSBR3. If tape A4 is to be used, the ASSIGN card would be

```
ASSIGN    A4 = SYSBR3
```

This card and any other required ASSIGN cards would be placed immediately after the JOB card for this job. An ASSIGN card is required for every symbolic tape unit used by the programmer.

Because of the importance of proper tape utilization to obtain maximum system efficiency it is suggested that a chart, similar to the one shown in the example below, be set up for each stack of jobs.

Example:

Suppose there is a stack of jobs with the following requirements:

Job	Symbolic Tapes	Execution Phase
1	SYSAR1, SYSBU1	3
2	SYSAR1	1
3	SYSAR4, SYSBR2, SYSAU8	2
4	SYSAU2	2
5	SYSBR2	1
6	SYSBR2, SYSAU4	2
7	SYSBU6	2
8	none	2

A table can then be formed on the basis of the phase in which the job will be executed. Normal system tape assignments are assumed. As shown in the table, the use of system tapes during Phases 1 and 3 is rather extensive if all operations are to be off-line. Thus, the majority of jobs should be executed in Phase 2.

After entering the tapes which remain the same during an entire phase, we then assign tapes as required for each job, so that the operator has time to mount and remove tapes where necessary.

If necessary, the tapes assigned as erase tapes (SYSESN) may be used as utility or reserved tapes for Phase 1 execution. The system will handle them automatically as though these were free and available tapes.

Prior to starting the next job, the system will stop if tapes normally used as erase tapes were used on previous jobs as reserved or utility tapes. Removal messages will be printed if the tapes were used as reserve tapes, and ready messages will be printed for those erase tapes involved.

PHASE	1		2					3
JOB NUMBER	2	5	3	4	6	7	8	1
TAPE NUMBER								
A1	SYSPOT	SYSPOT	-	-	-	-	-	SYSPOT
A2	SYSPPT	SYSPPT	-	-	-	-	-	SYSPPT
A3	SYSMIT	SYSMIT	SYSMOT	SYSMOT	SYSMOT	SYSMOT	SYSMOT	SYSMIT
A4*	SYSES1	SYSES1	SYSAU8	-	-	-	-	SYSDOT
A5	-	SYSBR3	SYSAR1	-	-	SYSBU6	-	-
B1	SYSTAP	SYSTAP	SYSTAP	SYSTAP	SYSTAP	SYSTAP	SYSTAP	SYSTAP
B2	SYSPIT	SYSPIT	-	SYSAU2	-	-	-	SYSAR1
B3	SYSMOT	SYSMOT	SYSMIT	SYSMIT	SYSMIT	SYSMIT	SYSMIT	SYSMOT
B4*	SYSES2	SYSES2	-	-	SYSBR2	-	-	-
B5	SYSAR1	-	SYSBR2	-	SYSAU4	-	-	SYSBU1

* ASSIGN cards for utility and reserved tapes may refer to the same tape unit used as SYSES1 and SYSES2 even though the job is to be executed in Phase 1.

The ASSIGN cards required for the above assignments would then be:

JOB 1

```
ASSIGN B2 = SYSAR1
ASSIGN B5 = SYSBU1
```

JOB 2

```
ASSIGN B5 = SYSAR1
```

JOB 3

```
ASSIGN A5 = SYSAR1
ASSIGN B5 = SYSBR2
ASSIGN A4 = SYSAU8
```

JOB 4

```
ASSIGN B2 = SYSAU2
```

JOB 5

```
ASSIGN A5 = SYSBR3
```

JOB 6

```
ASSIGN B4 = SYSBR2
ASSIGN B5 = SYSAU4
```

JOB 7

```
ASSIGN A5 = SYSBU6
```

JOB 8

none

SHARE MONITOR OPERATING NOTES

CHAPTER 5: RESTART PROCEDURE

If for some reason the monitor is written over during any phase of operation, it may be restored and operation begun with the next job to be processed. The procedure is as follows:

A. Note the tape assignments for SYSMIT and SYSMOT in the most recent tape status list. Make up an ASSIGN card for each of these.

B. Remove the first three cards of the loader deck and replace with the Recovery card.

C. Place the two ASSIGN cards in front of the GO card in the loader deck.

D. Depress the Clear key and the Load Cards key. Run all the cards in the revised loader deck through the card reader.

E. The system will be reloaded and initialized, and a stop will occur after a message is printed instructing the operator to place the phase number, in which to restart, in the entry keys. This phase number is determined from the tape status list. If input phase is indicated, this is Phase 1. The execution phase is Phase 2, and the output phase is Phase 3.

F. After setting the phase number in the keys, depress the Start key. The printer will indicate that restart has succeeded.

G. Restore the loader deck to its original form by removing the two ASSIGN cards and replacing the first three cards.

Note: The Recovery card is good only if the system tape is on B1. If non-standard assignments have previously been made for peripheral input, output and punch tapes, it will be necessary to include ASSIGN cards for those as well as for SYSMIT and SYSMOT.

current value of location counter	**02. 00. 06**
CUT	**07. 07. 08**

zero subfields 02. 00. 09

IBM
International Business Machines Corporation
Data Processing Division, 112 East Post Road, White Plains, N. Y.

Printed in U. S. A. 328-1219