File No. 1410-22
Form C28-0309

IBM Systems Reference Library

IBM 1410 Autocoder

This publication describes the Autocoder programming system for IBM 1410 Data Processing Systems. The Autocoder language and the method of preparing and using macro-instructions and subroutines are discussed. Also included is a detailed description of the pseudo-macro-instructions that provide flexibility in coding the library routines for macro-instructions.

Revision (April, 1963)

This publication supersedes the IBM 1410 Data Processing System Bulletin, *Autocoder – Preliminary Specifications,* Form J24-1433-2, and related Technical Newsletters N28-1006, N28-1015, N28-1024, N28-1032, N28-1042, and N28-1047.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the content of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D91, PO Box 390, Poughkeepsie, N.Y.

Contents

Introduction

AUTOCODER is an advanced symbolic programming system for the IBM 1410 Data Processing System. It supplements and extends, but does not replace, the Basic Autocoder for the IBM 1410.

A more powerful language than the Basic Autocoder, the IBM 1410 Autocoder can process macro-instructions, and reduces card handling by using magnetic tape or disk storage for program manipulation during assembly. The Autocoder processor can assemble programs designed to operate on all IBM 1410 systems. The macro-instructions described in the *IBM 1410 Input/Output Control System for Card and Tape Systems: Preliminary Specifications,* Form J28-1432, can also be used when coding in Autocoder language.

With Autocoder, the user can provide library routines for operations that are common to many source programs. These routines are extracted from the library and tailored automatically by the processor to satisfy particular requirements outlined in the source program by the programmer.

IBM 1410 Autocoder consists of two major parts: the symbolic language used by the programmer, and the processor program that translates this symbolic language into actual machine language and assembles the object program automatically. This publication contains the language specifications for IBM 1410 Autocoder.

The Autocoder language is composed of mnemonic operation codes used to perform operations falling in the following categories.

1. Declarative operations
2. Processor Control operations
3. Imperative operations
4. Macro operations

The Declarative and Processor Control operation mnemonic codes direct the processor program. These codes are not translated into machine language, *and do not appear in the object program.* Imperative and macro operation mnemonic codes are translated by the processor program into machine-language operation codes that constitute the object program.

This publication contains sections treating each of these categories of operations in detail.

Prerequisites

To use Autocoder, the programmer should be familiar with the material contained in the manual, *IBM 1410 Principles of Operation,* Form A22-0526.

Machine Requirements

The Autocoder processor can assemble object programs for all IBM 1410 systems. The Bulletin, *IBM 1410 Processor Operating System using Magnetic Tape and IBM 1301 Disk Storage; Preliminary Specifications,* Form J28-0243, specifies the machine requirements for assembling programs written in the Autocoder language.

Coding Sheet

The 1401/1410 Autocoder coding sheet (Figure 1) is free-form (the operand portion of each line is not subdivided into fields), thus allowing the programmer increased coding flexibility.

All Autocoder entries are entered on the Autocoder coding sheet. Column numbers on the coding sheet indicate the punching format for all input cards in the source deck. Each line of the coding sheet is punched into a separate card. If the source program is entered by magnetic tape, the contents of the cards prepared from the coding sheet must be written in one-card-per-tape-record format. The function of each portion of the coding sheet is explained in the following paragraphs.

PAGE NUMBER (COLUMNS 1 AND 2)

This two-character entry provides sequencing for coding sheets. Any alphamerical characters may be used. Follow standard collating sequence for the IBM 1410 when sequencing pages.

LINE NUMBER (COLUMNS 3-5)

A three-character line number sequences entries on each coding sheet. The first 25 lines are prenumbered 01-25. The third position can be left blank. (Blank is the lowest character in the collating sequence.) The five unnumbered lines at the bottom of each sheet can be used to continue line numbering or to make insertions between entries elsewhere on the sheet. Use the units position of the line number to indicate the sequence of inserts. Any alphamerical character may be used, but standard collating sequence should be used. For example, if an insert is to be made between lines 02 and 03, it could be numbered 021. Line numbers do not necessarily have to be consecutive, but the deck should be in collating sequence for sorting purposes.

The programmer should note that insertions can affect address adjustment. An insertion might make it necessary to change the adjustment factor in the operand of one or more entries. See "Address Adjustment."

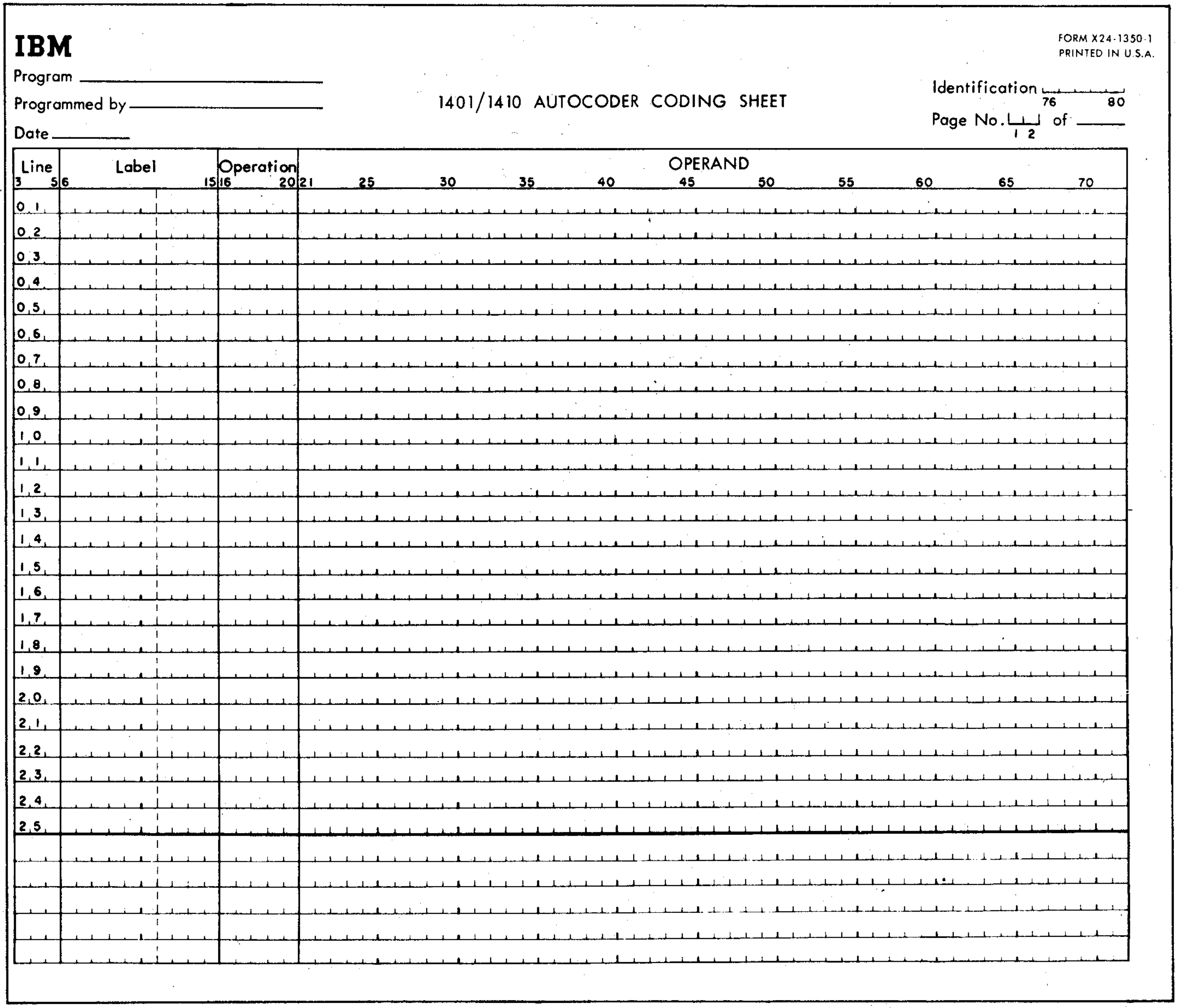
LABEL (COLUMNS 6-15)

Labeling is a method of providing meaningful alphamerical symbols for storage locations, constants, and instructions used in a program. All labels are assigned actual core-storage addresses during the assembly of an object program. When an entry is assigned a label, the programmer can refer to that entry symbolically by putting the label in the operand portion of a subsequent source program statement. Thus, the programmer need not concern himself with actual addresses of data and instructions, but must remember only the symbol which represents that address. Labels should be assigned only if subsequent reference to the items they represent is needed, because unnecessary labels delay the assembly process.

Autocoder labels can be symbolic or actual. A symbolic label can have as many as ten alphamerical characters, but the first character must be alphabetic. Special characters are not permitted in the label field.

Symbolic labels are written left-justified in the label field except as described in "DC" or "DCW."

Actual labels are always written left-justified in the label field. This actual address refers to the high-order position of the instruction, constant, or defined field. Actual labels have no effect on the address assignment counters.



IBM

FORM X24-1350-1
PRINTED IN U.S.A.

Program ____________________

Programmed by ____________________

Date ________

1401/1410 AUTOCODER CODING SHEET

Identification ______ (76 80)

Page No. __ (1 2) of ______

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50 55 60 65 70
01			
02			
03			
04			
05			
06			
07			
08			
09			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 1. IBM 1401/1410 Autocoder Coding Sheet

OPERATION (COLUMNS 16-20)

The operation field contains the mnemonic (easily remembered) operation code for an actual machine-language operation code. (See "Mnemonic Operation Codes.")

Several machine-language operation codes require operation modifiers (d-characters). With a few exceptions, these d-characters are incorporated into Autocoder mnemonics and do not have to be coded on the coding sheet. Thus, a single machine-language operation code may have two or more mnemonic equivalents. For example, the machine-language op code V *(test for word-mark or zone and branch)* has three mnemonic equivalents: BW (BRANCH IF WORD-MARK), BZN (BRANCH IF ZONE), and BWZ (BRANCH IF WORD-MARK AND/OR ZONE).

OPERAND (COLUMNS 21-72)

The operand field in an imperative instruction contains the actual or symbolic addresses of the data, literals, or address constants to be acted upon by the command in the operation field. Address adjustment and indexing can be used in conjunction with these constants.

The Autocoder coding sheet has a free-form operand field. The A-operand, the B-operand, and the d-character must be separated by commas. If address adjustment or indexing or both are to be performed, these notations must immediately follow the address being modified. Figures 3 and 4 show typical Autocoder entries.

COMMENTS

A comment can be included in the operand field of an Autocoder statement. At least two spaces must separate it from the last character of the operand.

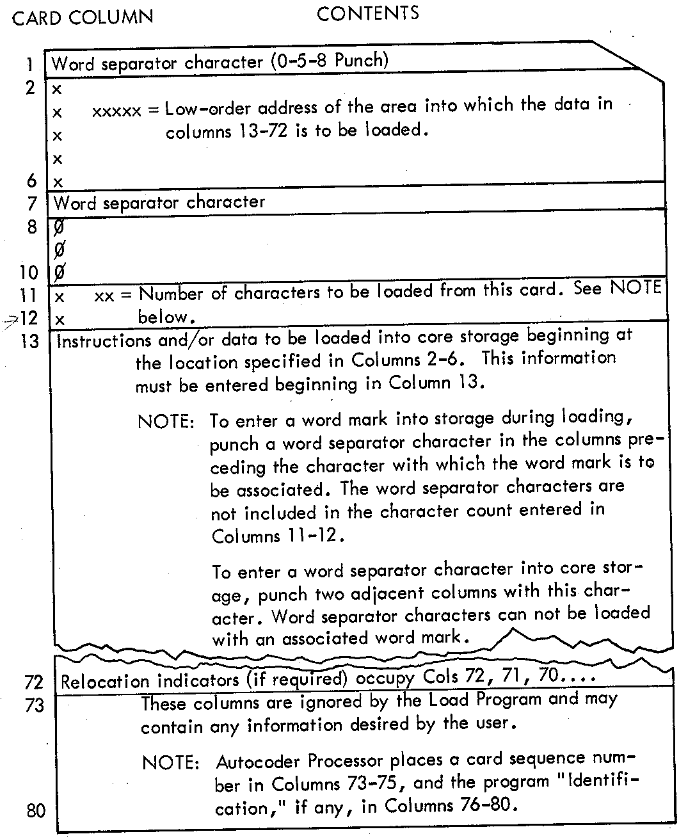
Entire lines of information can be included anywhere in the program by using a comments card. In such a card, containing comments only, the programmer must put an asterisk in column 6. Columns 7-72 can then be used for the comment itself. Comments inserted in this way appear in the symbolic listing but produce no entry in the object program.

IDENTIFICATION (COLUMNS 76-80)

This entry identifies a program or program section. This identification number, when used, appears in the object deck as described in "JOB." The areas labeled *Program, Programmed By,* and *Date* are for the convenience of the user, but are never punched.

Condensed Card Format

The machine language program produced by the Autocoder processor is punched into cards or placed on magnetic tape in card-image form, following the format shown in Figure 2.



CARD COLUMN	CONTENTS
1	Word separator character (0-5-8 Punch)
2 6	x x xxxxx = Low-order address of the area into which the data in columns 13-72 is to be loaded. x x x
7	Word separator character
8 10	ø ø ø
11 12	x xx = Number of characters to be loaded from this card. See NOTE x below.
13	Instructions and/or data to be loaded into core storage beginning at the location specified in Columns 2-6. This information must be entered beginning in Column 13. NOTE: To enter a word mark into storage during loading, punch a word separator character in the columns preceding the character with which the word mark is to be associated. The word separator characters are not included in the character count entered in Columns 11-12. To enter a word separator character into core storage, punch two adjacent columns with this character. Word separator characters can not be loaded with an associated word mark.
72	Relocation indicators (if required) occupy Cols 72, 71, 70....
73 80	These columns are ignored by the Load Program and may contain any information desired by the user. NOTE: Autocoder Processor places a card sequence number in Columns 73-75, and the program "Identification," if any, in Columns 76-80.

Figure 2. Condensed Card Format

Address Types

Six kinds of address types are valid in the operand field of an Autocoder statement: blank, actual, symbolic, asterisk, literals, and address constants.

Blank

A blank operand field is valid:

1. In an instruction that does not require an operand.
2. In instructions where valid A- and/or B-addresses are supplied by the chaining method. For example,

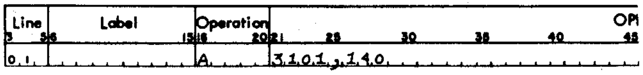
   MLCA A, B
   MLCA

NOTE: If an instruction is to have addresses stored by other instructions, the operand or operands affected must not be left blank. For example, B̌ 0 is recommended if the address of the branch instruction is to be supplied during the running of the object program.

Actual

The actual core-storage address of a data field is valid in the operand field. High-order zeros in actual addresses can be omitted as shown in Figure 3. Thus, an actual address can consist of from one to five digits.

Figure 3 shows an imperative instruction that causes the contents of core-storage location 3101 to be added algebraically to the contents of location 140. This entry will be assembled as a machine-language instruction: Ǎ 03101 00140. Note that high-order zeros can be eliminated when coding actual addresses for Autocoder.


Line
Label
Operation
OPERAND
3 5 6 15 16 20 21 25 30 35 40 45

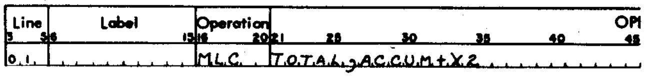
```
0 1  A     3101,140
```

Figure 3. Autocoder Instruction with Actual Addresses

Symbolic

A symbolic address can consist of as few as one or as many as ten alphamerical characters. Special characters are not permitted. Blanks may not be written within a symbolic address. Figure 4 shows how symbolic addresses are used.

Figure 4 shows an indexed imperative instruction that causes the contents of the location labeled TOTAL to be placed in an area labeled ACCUM as modified by the contents of index location 2. An address to be indexed is followed by a plus sign (+), an X to indicate indexing, and a number from 1 to 15 to specify which index location is to be used. TOTAL is the label for location 3101 and ACCUM is the label for location 140.


Line
Label
Operation
OPERAND
3 5 6 15 16 20 21 25 30 35 40 45

```
0 1  MLC   TOTAL,ACCUM+X2
```

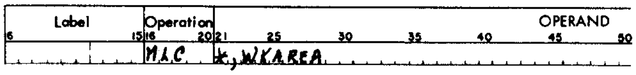
Figure 4. Autocoder Instruction with Symbolic Addresses

The assembled machine-language instruction for this entry is: Ď 03101 001M0 C. The M in the tens position of the B-address is a 4-punch with an 11-overpunch. The 11-overpunch is the B-bit tag for index location 2.

Asterisk (*)

If an asterisk (*) appears as an operand in the source program, the processor will replace it with the actual core-storage address of the last character of the instruction in which it appears (except in EQU, ORG, or LTORG statements, where the asterisk refers to the current position of the processor address assignment counter). For example, the instruction shown in Figure 5 is assigned core-storage locations 00340-00351. The actual address of WKAREA is 00598. The assembled instruction is Ď 00351 00598 C. When the instruction is executed in the object program, Ď 00351 00598 C will be placed in WKAREA.

Asterisk operands can have address adjustment and indexing.


Label
Operation
OPERAND
6 15 16 20 21 25 30 35 40 45 50

```
MLC   *,WKAREA
```

Figure 5. Asterisk Operand in Autocoder Instruction

Literals

The IBM 1410 Autocoder permits the user to put in the operand field of a source program statement the actual data to be operated on by an instruction. This data is called a "literal." The processor allocates storage for literals and inserts their addresses in the operand or operands of the instructions in which they appear. The processor produces a DCW card that puts a word mark in the high-order position of a literal when it is stored at program load time. Literals are permitted only in the operand field of an Autocoder statement and can be numerical or alphamerical. A literal can be up to 52 characters in length, including the sign; i.e., it must be contained in one line of the coding sheet, and it must not extend beyond column 72. Literal addresses may make use of address-adjustment and/or indexing.

Four types of literals are discussed in the following paragraphs.

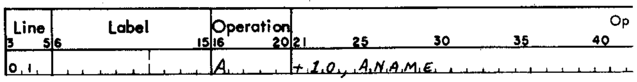
NUMERICAL LITERALS

Numerical literals are written according to the following specifications:

1. A plus or minus sign must precede a numerical literal. The processor puts the sign over the units position of the number when it is assigned a storage location. NOTE: To store an unsigned number, use an alphamerical literal.

2. When a numerical literal does not exceed nine digits plus sign (blanks are not allowed), it is assigned a storage location only once per program or *program section*, no matter how many times it appears in the source program or program section. NOTE: A program section is defined as the source program entries that precede a Literal Origin, End or Execute Statement. In some programs several program sections are needed because the entire object program exceeds the total available storage capacity of the object machine. In these cases individual program sections are loaded into storage from cards, tapes, or random access storage and are executed as they are needed. Program sections are sometimes called "overlays."

Figure 6 shows how a numerical literal can be used in an imperative instruction. Assume the literal (+ 10) is assigned a storage location of 00584 and 00585 and ANAME is assigned 00682. The symbolic instruction will cause the processor to produce a machine-language instruction (Ǎ 00585 00682) that causes + 10 to be added to the contents of ANAME.



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 Op
01		A	+10, ANAME

Figure 6. Numerical Literal

ALPHAMERICAL LITERALS

Alphamerical literals are written according to the following specifications:

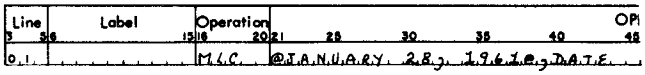
1. An alphamerical literal must be preceded and followed by the @ symbol. The literal itself can contain blanks, alphabetic, numerical, and special characters (including the @ symbol). However, a comment on the same line as an alphamerical literal must not contain the @ symbol.

Upon encountering an alphamerical literal, the processor proceeds to column 72 of the card and searches right to left for the terminal @ symbol. If it encounters any @ symbol, it will assume this is the legitimate terminal.

2. An alphamerical literal of from one to nine characters with preceding and following @ symbols is assigned a storage location only once per program or program section, no matter how many times it is used in the source program.

3. Longer alphamerical literals are assigned a storage location each time they are encountered in the source program. To save storage space in cases where multiple use of long literals is necessary, use a DCW statement.

Figure 7 shows how an alphamerical literal can be used in an imperative instruction. Assume that the literal JANUARY 28, 1961 is assigned a storage location of 00906, and DATE is assigned 00230. The machine-language instruction (Ď 00906 00230 C) causes the literal JANUARY 28, 1961 to be moved to DATE.



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 45 OP
01		MLC	@JANUARY 28, 1961@,DATE

Figure 7. Alphamerical Literal

AREA-DEFINING LITERAL

The 1410 Autocoder allows the user to define an area to be reserved by placing an area-defining literal in the operand field of a symbolic program entry as follows:

1. An area of any size may be defined in any instruction which has as an operand the symbol which references it; for example, WKAREA#6.

2. A # symbol (8-3 punch) must precede the number that specifies how many core-storage locations are needed for the work area. Note that the # symbol is represented in the FORTRAN character set as an = symbol.

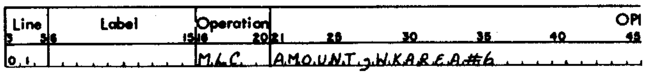
3. A word mark is placed over the high-order position of the area.

4. If the user refers to a portion of the same defined area, such as WKAREA#2, he will be given a multiple definition flag in his output listing. However, he may refer to the defined area (WKAREA#6) more than once.

5. Area-defining literals must be redefined after each LTORG entry. (See "Processor Control Operations.")

6. Address adjustment and indexing are permitted when using area-defining literals.

Figure 8 shows an imperative instruction with an area-defining literal. This entry causes the processor to allocate six storage locations for WKAREA. Six blanks will be loaded in storage at object program load time by a DCW card automatically produced by the processor. Assuming that AMOUNT is in storage location 00796 and WKAREA is in 00596, the assembled machine-language instruction that moves AMOUNT to WKAREA is Ď 00796 00596 C.



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 45 OP
01		MLC	AMOUNT,WKAREA#6

Figure 8. Area-Defining Literal

ADDRESS CONSTANT LITERALS

The actual 5-character machine address which is assigned to a label by the processor can be defined as an *address constant.* Autocoder permits address constants to be coded symbolically in the instructions that require them:

1. The symbol for an address constant can contain as many as ten characters.
2. A plus sign must precede the symbol. The address constant is the actual address which was assigned to the label by the processor.
3. The address constant being defined must appear elsewhere as a symbol in the symbolic program.
4. The address constant is assigned a core-storage address, as are all constants, and a DCW card is created automatically by the processor. The address constant literal is unsigned in core storage.

NOTE: If address adjustment and/or indexing occur, they modify the address of the literal, not the literal itself. If the literal itself is to be modified, it must be defined by a DCW statement.

Figure 9 shows how an address constant literal can be used. Assume that CASH is used as a label elsewhere in the program and has been assigned a machine address of 00600. The address constant (00600) has been assigned storage location 00797. The first character in the second instruction is in core storage at address 00401. Thus, the address of INST + 5 is 00406.

Label 6–15	Operation 16–20	OPERAND 21–50
	MLC	+CASH,INST+5
INST	MLC	0,WORK

Figure 9. Address Constant

The assembled machine-language instruction for the first symbolic instruction in Figure 9 is $\overset{\vee}{\text{D}}$ 00797 00406 C.

WORK is in storage location 00729. The assembled machine-language instruction for the second symbolic program entry is $\overset{\vee}{\text{D}}$ 00000 00729 C. When the first instruction is executed in the object program, the constant 00600 is moved to 00406 and the second instruction becomes $\overset{\vee}{\text{D}}$ 00600 00729 C. When the second instruction is executed, the contents of CASH are moved to WORK.

Thus, the programmer can write an instruction that will move a machine address into the operand of another instruction at program execution time, even though he does not know what that address is.

Address Adjustment

Address adjustment is valid in the operand field on all symbolic addresses, including the asterisk. It enables the programmer to refer to an entry in his source program that is a specified number of locations away from a symbolic address. Its usage reduces the number of symbolic labels required. Address adjustment is indicated by writing after the symbolic address a plus or minus sign followed by one to five digits (Figure 10).

When the label MANNO is assigned location 05000 and TOTAL is assigned the location 00075, the assembled instruction is $\overset{\vee}{\text{A}}$ 05012 00075.

If the instruction in Figure 11 is assigned the address 05000, the assembled instruction is $\overset{\vee}{\text{?}}$ 04998 00075, because * refers to the rightmost position of the instruction (05010). When using address adjustment, the programmer should remember that insertions or deletions in the source program can affect adjustment addresses.

Label 6–15	Operation 16–20	OPERAND 21–50
	A	MANNO+12,TOTAL

Figure 10. Address Adjustment

Label 6–15	Operation 16–20	OPERAND 21–50
	ZA	*-12,TOTAL

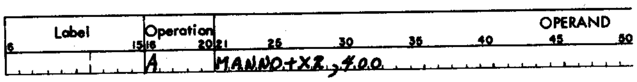
Figure 11. Address Adjustment with an Asterisk Operand

Index Registers

Indexing is accomplished by tagging an address in the operand field with an indicator telling the processor which index register is to be used. The IBM 1410 system has 15 index registers that can be referred to in Autocoder language by placing an X before their number. Thus, X10 denotes index register 10. The X enables the processor to distinguish between address adjustment and indexing.

An index register can also be referred to symbolically. X0 through X15 are not acceptable as symbolic names. The index label must be preceded by a plus. It follows the operand address and the address adjustment, if any. Figure 12 shows an example of indexing.

The contents of the location whose address is the address of MANNO *plus the contents of index register 2* are algebraically added to the contents of location 00400. For example, if the label MANNO is assigned location 05000 and index register 2 contains 500, then the preceding instruction causes the contents of location 05500 to be added to the contents of location 00400. Indexing is not acceptable in DS, ORG, or LTORG declarative operations or control operations.



Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
	A	MANNO+X2,400

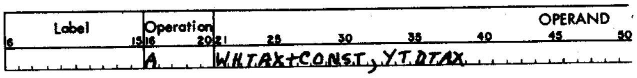
Figure 12. Indexing

An index register can be specified in the operand field for other than indexing purposes. For example, a numerical value can be added to the contents of an index register. In this case, the index register may be referred to by its actual label (X1, X2, etc.) or its symbolic label (see "EQU").

Index Register Reservation

The processor assigns index registers referred to in the symbolic program.

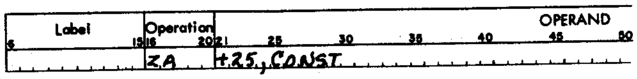
Those index registers that are coded in actual notation (X1, X2, etc.) and those equated to a symbolic address by an EQU statement are assigned first. Then the remaining index registers are assigned to symbols the programmer has used to represent index registers. For example, the programmer may use the symbolic instruction shown in Figure 13.



Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
	A	WHTAX+CONST,YTDTAX

Figure 13. Symbolic Label for an Index Register

In this case, CONST is the symbolic label for an index register. Its contents will modify the address assigned to the label (WHTAX). The instruction in Figure 13 may be followed by the instruction shown in Figure 14. This instruction puts the numerical value 25 in the index register which the processor assigns to CONST.

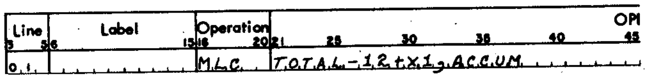


Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
	ZA	+25,CONST

Figure 14. Using the Symbolic Label

Autocoder Coding Examples

Figure 15 shows an imperative instruction with address adjustment and indexing on a symbolic address. The processor will subtract 12 from the address assigned the label TOTAL. The effective address of the A-operand is the sum of TOTAL −12 plus the contents of index location 1. The assembled instruction Ď 030Y9 00140 C will cause the contents of the effective address of TOTAL −12 +X1 to be placed in the location labeled ACCUM (assuming again that TOTAL is the label for location 3101 and ACCUM is the label for location 140). The Y in the tens position of the A-address is an 8-punch with a zero overpunch. The zero overpunch is a tag for index location 1.



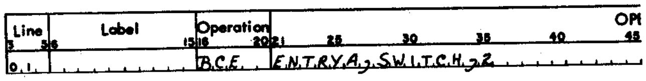
Line	Label	Operation	OP
3 5	6 15	16 20	21 25 30 35 40 45
01		MLC	TOTAL-12+X1,ACCUM

Figure 15. Autocoder Instruction with Address Adjustment and Indexing

NOTE: Address adjustment and indexing are permitted in the same operand. Multiple address adjustment causes the algebraic sum of the factors to be used. With multiple indexing, only the rightmost index notation is effective. For example:

A TOTAL +3 +X1 −12 +X2, ACCUM −5 +X2 +35
will be interpreted as:
A TOTAL −9 +X2, ACCUM +30 +X2
which is equivalent to:
A TOTAL +X2 −9, ACCUM +X2 +30

Figure 16 is an imperative instruction with two symbolic operands and a d-character. Although many of the augmented operation codes available for use with Autocoder eliminate the need to write the d-character in a symbolic instruction, sometimes the d-character must be specified by the programmer. If an instruction requires such a specified d-character, it is written following the A- and B-operands and is separated from the remainder of the instruction by a comma. The assembled machine-language instruction is: B̌ 00392 00498 2. It branches to ENTRY A (00392) if the location labeled SWITCH contains a 2.



Line	Label	Operation	OP
3 5	6 15	16 20	21 25 30 35 40 45
01		BCE	ENTRYA,SWITCH,2

Figure 16. Autocoder Instruction with a d-Character

Declarative Operations

A program for the 1410 usually requires the use of *work areas* and *constants.* A work area is a portion of storage into which data is transferred for processing. It can be used for the accumulation of totals or for the assembling of data to be printed out or punched into cards. A constant is a fixed quantity or item of information that is required again and again or that must remain the same throughout the course of the program. For example, a date can be considered a constant.

Autocoder enables the programmer to refer to work areas and constants by their descriptive names without regard to their actual location in core storage. For example, assume that the programmer wants to reserve twenty consecutive core locations for accumulating a final sales total. A declarative operation enables the programmer to reserve such an area and to refer to it by a symbolic label without concerning himself with the actual address of the field.

Declarative operations are definitions rather than instructions. As such they are acted upon during assembly but are not executed during the running of the object program. For this reason the programmer should keep declaratives separate from imperatives (machine instructions) when writing the symbolic program. If they are placed in the body of the program, care must be taken to branch around them so they will not be treated as instructions.

The IBM 1410 Autocoder provides five different declarative operations for reserving work areas and storing constants:

OP CODE	PURPOSE
DCW	Define Constant with Word Mark
DC	Define Constant (no word mark)
DS	Define Symbol
DA	Define Area
EQU	Equate

DCW — Define Constant with Word Mark

A DCW statement is used to enter a numerical, alphamerical, or address constant with a word mark into a core-storage area. Symbolic labels address the low-order position of the constant. Word marks are set in the high-order positions of all constants. If a symbolic label is indented one position, the address of the high-order position of the constant will be assigned to the symbol. Actual labels always refer to the high-order position of the defined constant.

The programmer:

1. Writes the operation code (DCW) in the operation field.

2. May write an actual or symbolic label in the label field. The programmer may refer to the constant later by writing this label in the operand portion of subsequent instructions.

3. Writes the constant in the operand field beginning in column 21.

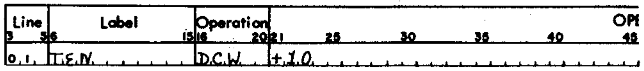
NUMERICAL CONSTANTS

1. A numerical constant can be preceded by a plus or minus sign. A plus sign causes AB-bits to be placed over the units position of the constant; a minus sign causes a B-bit to be put there. If a numerical constant is unsigned in the DCW statement, it will be stored as an unsigned field.

2. The first blank column appearing in the operand field terminates a numerical constant.

3. The maximum size of a numerical constant is 51 digits and a sign, or 52 digits with no sign.

Example: Figure 17 shows the number +10 defined as a numerical constant. The address of the constant will be inserted in the object instruction wherever TEN appears in the operand field of another symbolic instruction.



Line	Label	Operation	OPE
3 5	6 15	16 20	21 25 30 35 40 45
01	TEN	DCW	+10

Figure 17. Numerical Constant Defined in a DCW Statement

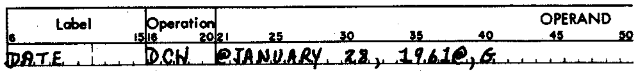
ALPHAMERICAL CONSTANTS

1. An alphamerical constant must be preceded and followed by the @ symbol. Blanks and the @ symbol can appear within an alphamerical constant, but the @ symbol cannot appear in a comment on the same line as an alphamerical constant.

2. The alphamerical constant itself can be as large as 50 characters.

3. If no terminal @ is present, a 51-character constant will be produced.

Example: Figure 18 shows the alphamerical constant, JANUARY 28, 1961, defined in a DCW statement. The address of the constant will be inserted in the object pro-



Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
DATE	DCW	@JANUARY 28, 1961@,G

Figure 18. Alphamerical Constant Defined in a DCW Statement

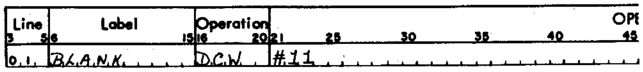
gram instruction wherever DATE appears in the operand field of another symbolic program entry.

NOTE: A comma G following the trailing @ symbol of an alphamerical constant causes the processor to put a group-mark word-mark in storage following the last character of the constant. The associated label, if any, will refer to the last character of the constant, not the group-mark word-mark.

BLANK CONSTANTS

A # symbol precedes a number indicating how many blank storage positions are to be defined. This permits the programmer to reserve a field of blanks with a word mark in the high-order position of the field. Maximum size of this field is limited only by the available storage capacity.

Example: Figure 19 shows an 11-character blank field defined by a DCW statement. The address of this blank field will be inserted in an object program instruction whenever the symbol BLANK appears as the operand of another symbolic program entry.



Line	Label	Operation	OPE
3 5	6 15	16 20	21 25 30 35 40 45
01	BLANK	DCW	#11

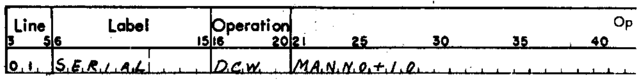
Figure 19. Blank Constant Defined by a DCW Statement

ADDRESS CONSTANTS

An address constant can be preceded by a plus or a minus sign, or it can be left unsigned. The constant is the actual machine-language address of the field whose associated label is included in the operand. The units position of the constant will have the sign which the user placed before the operand.

NOTE: Address constants may be address adjusted and indexed.

Example: Figure 20 shows an address constant (the address of MANNO) defined by a DCW statement. The address of the address constant MANNO will be inserted in an object program instruction whenever SERIAL appears as the operand of another symbolic program entry. This example shows the DCW defining the address of MANNO plus 10. Therefore, if MANNO is located at 01000, the value in the DCW will be 01010.



Line	Label	Operation	Op
3 5	6 15	16 20	21 25 30 35 40
01	SERIAL	DCW	MANNO+10

Figure 20. Address Constant (address adjusted) Defined by a DCW Statement

DC — Define Constant (No Word Mark)

This statement has the same characteristics as the DCW statement. The only difference is that the processor does not cause a word mark to be set at the high-order position of the constant when the constant is produced in the object deck.

DS — Define Symbol

A DS statement reserves and labels an area of core storage. It differs from a DCW or DC statement in that no information (constant) is loaded into this area at program load time.

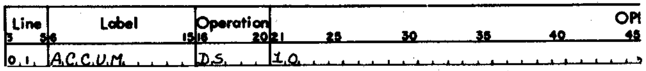
The programmer:

1. Writes the operation code (DS) in the operation field.
2. May write a symbolic address in the label field.
3. Writes a number in the operand field to indicate how many storage positions are to be reserved.

The processor:

1. Assigns an actual address to the low-order position of the reserved area.
2. Inserts this address in the instruction wherever the symbol in the label field appears in the operand field of another symbolic program entry.

Example: Figure 21 shows how a 10-position core-storage area can be reserved. The programmer can refer to the label by putting ACCUM in the operand field of another symbolic program entry.



Line	Label	Operation	OPE
3 5	6 15	16 20	21 25 30 35 40 45
01	ACCUM	DS	10

Figure 21. DS Statement

DA — Define Area

DA statements reserve and define portions of core storage, such as input or output or work areas. They can also define more than one area, if all these areas are identical in format. A DA statement differs from a DCW statement in that a DA statement can, in addition to defining the large area, also define several fields within it. The DA statement furnishes the processor with the lengths, names, and relative positions of fields within the defined area.

HEADER LINE

The programmer constructs a header line for the DA entry as follows:

1. Writes the operation code (DA) in the operation field.
2. May write an actual or symbolic address in the label field. This address represents the high-order position of the entire area defined by the DA statement.

3. Indicates in the operand field the required size of the area in the form B X L. B is the number of identical areas to be defined and L is the length of each area. For example, if four identical areas, each 100 characters long, are to be defined, the first entry in the operand field is 4 X 100 as shown in Figure 22. If only one area is to be defined, the first entry is 1 X 100.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 OPI 45
0 1	TAPEAR	DA	4X100

Figure 22. Four Areas Defined

Indexing: To index a DA statement, place a comma and the number of the index location (X1, X2, X3, etc.) after the B X L indication. All labels in the entries following the header line will be indexed by the specified index register when they appear in instructions, *unless the instruction referring to the field is itself indexed.* For example, if IN AREA is defined by the statement shown in Figure 23, ACCUM is indexed by index location 1. If the entry shown in Figure 24 appears as an instruction elsewhere in the program, ACCUM (for this instruction only) will be indexed by the contents of index location 2. Because the instruction in Figure 24 has indexing, this indexing overrides the indexing prescribed by the DA statement.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 OPI 45
0 1	INAREA	DA	3X60,X1
0 2	ACCUM		35,40

Figure 23. Indexing a DA Statement

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 10 OPI 45
0 1		ZA	GROSS,ACCUM+X2

Figure 24. Overriding Indexing in a DA Statement

NOTE: The programmer can negate the effect of indexing on a field or subfield by putting an X0 in the operand field of each instruction in which indexing is not wanted. Symbolic names for index registers may be specified in the heading line of a DA statement only if previously defined by an EQU statement.

Record Marks: Record marks can be inserted to separate records in the defined area. The processor will cause a ǂ to be placed in storage immediately following each identically defined area if a comma ǂ follows the B X L entry in the operand field. B X L does not include an allowance for the record mark. For example, 2 X 100 will cause 200 positions to be reserved for the defined area, but 2 X 100, ǂ will cause 202 positions to be reserved.

Group Mark with Word Mark: The user can cause the processor to put a group mark with a word mark one position to the right of the entire defined area by writing a G, preceded by a comma in the operand field.

Relative to Zero Addressing: By writing a comma zero after the B X L entry, the user can cause the processor to assign addresses to the labels of fields and subfields as though the high-order position of the defined area was core-storage location zero. The label of the DA statement is assigned the address of the high-order position of the area actually reserved by the processor.

NOTE 1: A user of 1410 IOCS must define areas to be used for blocked records using indexing, with relative to zero addressing.

NOTE 2: The programmer may write the ǂ, index code, G and 0 entries in any order in the operand field of the DA header statement provided that they follow the B X L entry.

OTHER DA ENTRIES

The programmer constructs the balance of the DA statement which defines fields and subfields for each area as follows:

1. Leaves the operation field blank.

2. Writes a symbolic label in the label field if one is desired.

3. Specifies the relative location of defined fields within the area by putting two numbers in the operand field. The first location of the defined area is considered location 1. The high-order and low-order positions of the field are written beginning in column 21. These two numbers must be separated by a comma.

4. A subfield is a field within a defined field and is defined by putting the number representing the low-order position in the operand field.

NOTES: The processor causes word marks to be set in the high-order position of each defined field, but does not so identify subfields. If a word mark is desired in a one-position field, the relative position number must be written twice with the two numbers separated by a comma.

Fields defined in a DA statement can be listed in any order, and all positions within the defined area do *not* have to be included in the defined fields.

The processor:

1. Allocates an area in core storage equal to B X L plus positions for record marks and a group mark if they are specified in the heading line of the DA entry and assigns actual addresses to the defined fields and subfields.

2. Inserts the assigned address of the high-order position of the entire defined area wherever the contents of the heading line label field appear as the operand of another symbolic program entry.

3. Inserts the assigned addresses of the low-order positions of defined fields and subfields in the place of symbols corresponding to the labels of the field-defining entries.

Result: At object program load time:

1. Word marks are set for field definition as noted previously.
2. A group mark and record marks are loaded as specified in the heading line.

Example

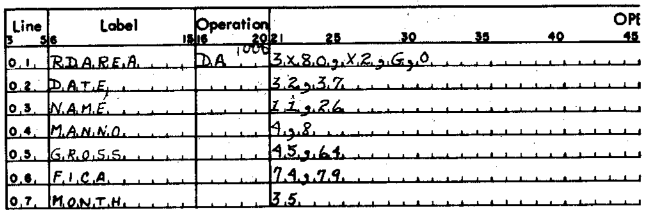
In this example, data is to be read from magnetic tape into an area of storage where it is to be processed. It is a payroll operation, and each record refers to a different employee. The records are written on tape in blocks of three. Each record is eighty characters long and has the following format:

Positions 4-8	Man Number
Positions 11-26	Employee Name
Positions 32-37	Date
Positions 45-64	Gross Wages
Positions 66-71	Withholding Tax
Positions 74-79	FICA Deduction

Remaining positions contain data not used in this operation. A group mark with a word mark is to be placed in storage immediately following the third area.

The DA statement in Figure 25 defines three adjacent identical areas into which each block of three records will be read. It also defines the fields and subfields that are to receive the data listed. The notation 3 X 80 in the header line indicates that three consecutive areas of eighty locations each are to be reserved. The entire 240-location area can be referred to by its high-order label, RDAREA. The G in the header line will cause a group mark with a word mark to be placed in the 241st position. The reference to index location 2 in the header line indicates that the labels NAME, MANNO, DATE, GROSS, FICA, and MONTH, when referred to in symbolic instructions, will be indexed by index location 2.

The IOCS will give an instruction to read data from tape into a storage area labeled RDAREA. This causes a block of three data records to be placed in the 240 reserved core locations. As a result, the significant data is read into the appropriately labeled fields. This data can now be referred to via the labels DATE, MANNO, FICA, etc., and the user need not concern himself with actual machine addresses. In this example, the IOCS begins by setting index location 2 to the address of the input area. The user then processes the significant data in the first record. The subsequent GET macro will increment index location 2 by eighty, and the user can branch back to the first instruction of the particular routine. Because all labels defined by this DA statement are incremented by the contents of index location 2, the program will now be processing the second record read into storage. When this routine is performed three times, the user has processed three input records and is ready to read three more records into storage. This has all been performed without any reference to actual machine addresses.



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 OPE 45
01	RDAREA	DA 1000	3X80,X2,G,0
02	DATE		32,37
03	NAME		11,26
04	MANNO		4,8
05	GROSS		45,64
06	FICA		74,79
07	MONTH		35

Figure 25. DA Statement

NOTE 1: An area can be reserved for a record with variable fields by defining all possible fields as subfields. In this case, no word marks will be set in an individual area, but the programmer can control data transfer by setting word marks in the receiving fields.

NOTE 2: If the length of the whole record can also vary, the programmer should reserve an area equal to the largest possible record size.

EQU — Equate

An EQU statement assigns a symbolic label to an actual or symbolic address. Thus, the user can assign different labels to the same storage location in different parts of his source program.

The programmer:

1. Writes the operation code (EQU) in the operation field.
2. Writes an actual or symbolic address in the operand field. This address can have indexing and address adjustment.
3. Writes a new label in the label field for the symbolic address defined in the operand field.

The processor:

1. Assigns to the label of the equate statement the same actual address that is assigned to the symbol in the operand field (with appropriate alteration if indexing and address adjustment are indicated).
2. Inserts this actual address wherever the label appears as the operand of another symbolic program entry.

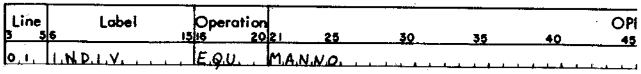
Result: The programmer can now refer to a storage location by using either name.

Examples

Figure 26 shows the label INDIV equated to MANNO which has been assigned storage location 01976. Whenever either MANNO or INDIV appear in a symbolic program, 01976 will be used as the actual address.

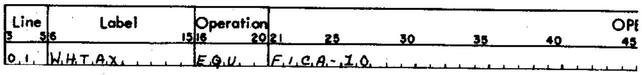
Figure 27 shows an equate statement with address adjustment. If FICA is assigned location 00890, WHTAX will be equated to FICA−10 (00880). WHTAX now refers to a field whose units position is 00880.

Figure 28 shows a label assigned to an actual address. Assume that an input card contains NETPAY in card columns 76-80. When this card is read into storage, the area locations 01076-01080 contain net pay (if the read area is 01001-01080). This field can be referred to as NETPAY if the EQU statement in Figure 28 is written in the source program.



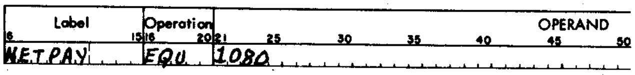
Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 OPI 45
01	INDIV	EQU	MANNO

Figure 26. EQU Statement



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 OPI 45
01	WHTAX	EQU	FICA-10

Figure 27. Address Adjustment in an EQU Statement



Label 6 15	Operation 16 20	21 25 30 35 40 OPERAND 45 50
NETPAY	EQU	1080

Figure 28. Assigning a Label to an Actual Address

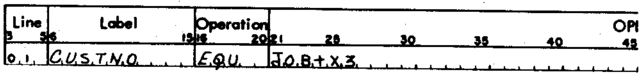
Figure 29 shows how an equate statement can be indexed. With indexing, the label is indexed by the index location specified in the EQU statement, whenever it appears as an operand in a symbolic program entry, unless the operand in which it appears is itself indexed. In Figure 29, the address assigned the symbolic label CUSTNO is equated to the actual address of JOB + the contents of index location 3. However, if CUSTNO + X2 or CUSTNO + X1 appear as the operand of another symbolic program entry, the actual address of JOB will be added to the contents of index location 2 or 1. Thus, the indexing in an instruction takes precedence, and index register 3 is ignored.

Figure 30 shows the symbol FIELDA equated to an asterisk address. The asterisk refers to the current position of the processor assignment counter. (This will be the first position of the instruction or data to be next assigned.) Assume that this address is 00698. FIELDA is now equal to 00698.

Figure 31 shows how a label can be assigned to an index location. The operand contains a number from 1 to 15, followed by a comma, followed by the letter X to indicate the specific index register. INDEX 1 is now equal to 00029. Figure 31 also shows an alternative method for equating a label to an index register.

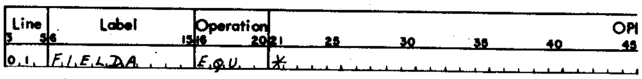
Figure 32 shows how a tape unit can be assigned a label. In this case, the programmer wishes to refer to tape 4 on channel 1 as INPUT.

A tape unit may also be equated to a symbolic name by using the actual X-control field (for example % U 4) as the operand, as shown in Figure 33.



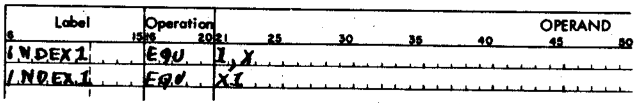
Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 OPI 45
01	CUSTNO	EQU	JOB+X3

Figure 29. Indexing an EQU Statement



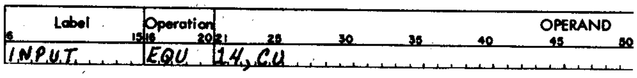
Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40 OPI 45
01	FIELDA	EQU	*

Figure 30. Equating with an * Operand



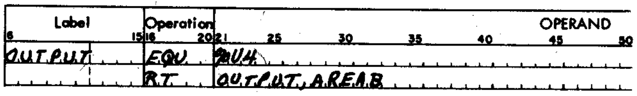
Label 6 15	Operation 16 20	21 25 30 35 40 OPERAND 45 50
INDEX1	EQU	1,X
INDEX1	EQU	X1

Figure 31. Assigning a Label to an Index Location



Label 6 15	Operation 16 20	21 25 30 35 40 OPERAND 45 50
INPUT	EQU	14,CU

Figure 32. Assigning a Label to a Tape Unit



Label 6 15	Operation 16 20	21 25 30 35 40 OPERAND 45 50
OUTPUT	EQU	%U4
	RT	OUTPUT,AREAB

Figure 33. Actual X-Control Field

Processor Control Operations

Autocoder has several control operations that enable the user to exercise some control over the assembly process:

OP CODE	PURPOSE
JOB	Job Card
EJECT	Eject
RESEQ	Resequence
LOAD	Load
RUN	Run
CTL	Control Card
ORG	Origin
LTORG	Literal Origin
EX	Execute
XFR	Transfer
SFX	Suffix
PST	Print Symbol Table
END	End

JOB — Job

This card in the user's source program deck prints a heading line on each page of the output listing from the assembly process and identifies the object deck or tape.

The programmer:

1. Writes the mnemonic operation code (JOB) in the operation field.
2. Writes in the operand field the indicative information to be printed in the heading line. This information may be any combination of valid 1410 characters and appears in columns 21-72.
3. Writes in the identification field (columns 76 to 80), the information to be contained in the object deck or tape.

The processor:

1. Prints the information, the identification from columns 76-80, and a page number from the JOB card on each page of the output listing. If there is no JOB card, the processor will generate one. In this case, nothing will be printed in the heading line except the page number.
2. Punches the identification number (columns 76-80) in all condensed cards produced for the object program. If another JOB or RESEQ card (or cards) appear elsewhere in the source program, the new identification number will be punched in subsequent condensed cards.

Result: The programmer can identify a job or parts of a job in the output listing.

EJECT — Eject

An EJECT control card may be placed in the symbolic source program by the user to cause the carriage to restore at any point in the output listing.

The programmer may now have separate routines or sequences in the output listing.

RESEQ — Resequence

The RESEQ control card resets the card sequence count to 001 in the object program deck and the identification number from columns 76 to 80 will replace the former identification number.

This will allow the user to separate his object deck into logical groups or blocks.

LOAD — Load

The LOAD control card is used to signal the processor that a load program should precede the object deck.

RUN — Run

This is the first card in the user's source program deck. It tells the processor which type of run is desired. There are two types: Autocoder and Systems.

AUTOCODER

By placing the label AUTOCODER in the label field of a run control card, the user signifies that he desires a compilation of a given deck by the processor. The user's source program deck is placed immediately behind this card.

SYSTEMS

By placing the label SYSTEMS in the label field of a run control card, the user signifies that he desires an updating run. This updating run pertains only to the library entries or routines on the systems tape.

CTL — Control

The control statement is normally the second entry (card) in the source program deck.

The programmer:

1. Writes the operation code (CTL) in the operation field.

2. Writes codes in the operand field as follows:

Column 21: This column is unused and is not interpreted by the processor.

Column 22: Indicates the core-storage size of the 1410 system that will be used to process the object program. Use the following codes:

STORAGE SIZE	CODE
10,000	1
20,000	2
40,000	3
60,000	4
80,000	5
	6

Column 23: Indicates if punch or print operations are to be suppressed. The codes are as follows:

CODE	
1	Suppress Punch
2	Suppress Print

The processor interprets the codes and processes the source program accordingly.

NOTE: If the CTL card is missing, the processor assumes that both the processing machine and the object machine have 20,000 positions of core storage.

ORG — Origin

An origin statement can be used by the programmer to specify a storage address at which the processor should begin assigning locations to instructions, constants and work areas in the symbolic program.

The programmer:

1. Writes the mnemonic operation code (ORG) in the operation field.

2. Writes the symbolic, actual, blank, or asterisk address in the operand field. Addresses can have address adjustment, but indexing is *not* permitted in ORG statements.

3. If a symbolic label appears in the operand field of an ORG statement, it must appear in the label field in an entry preceding the ORG statement in the program sequence.

The processor:

1. Assigns addresses to subsequent instructions, constants and work areas starting with the address specified in the operand field of the ORG statement.

2. If there is no ORG statement preceding the first symbolic program entry, the processor automatically begins assigning storage locations at 00500.

NOTE: In the absence of the IOCS entries, normal origin will be 00500.

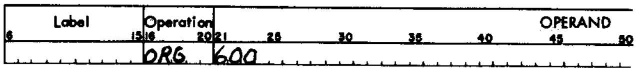
3. An ORG statement inserted at any point within the symbolic program causes the processor to assign subsequent addresses beginning at the address specified in the operand field of the new ORG statement. (Exception: see Figure 39.)

4. The processor maintains a high assignment counter which contains the highest assigned location at any given point of an assembly run.

Result: The programmer chooses the area of storage where the object program, defined constants, etc., will be located.

Examples

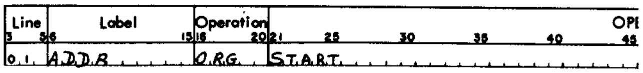
Figure 34 shows an ORG statement with an actual address. The first symbolic program entry following this ORG statement will be assigned with storage location 00600 as a reference point. If the first entry is an instruction, the op code position (I-address) of that instruction will be 00600; if the first entry is a 5-character DCW, it will be assigned address 00604, etc.



Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
	ORG	600

Figure 34. ORG Statement with an Actual Address

The ORG statement in Figure 35 shows how the programmer can direct the processor to save the address of the last storage location allocated. The label ADDR is the symbolic address of the next available location before re-origin occurs. The processor will continue to assign addresses beginning at the actual address of START.



Line	Label	Operation	OPI
3 5	6 15	16 20	21 25 30 35 40 45
01	ADDR	ORG	START

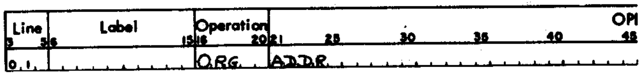
Figure 35. Saving the Address of Last Storage Allocated

The programmer can insert another ORG statement later in the source program to direct the processor to begin assigning storage at ADDR.

Figure 36 shows an ORG statement that directs the processor to start assigning addresses with the actual address assigned to ADDR.

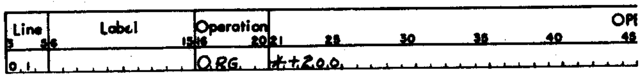
Figure 37 shows an ORG statement that directs the processor to bypass 200 positions of core storage when assigning addresses.

When the processor encounters the statement shown in Figure 38, it will assign subsequent addresses beginning with the next available storage location whose address is a multiple of 100. For example, if the last



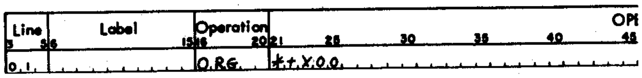
```
01        ORG  ADDR
```

Figure 36. ORG Statement with a Symbolic Address



```
01        ORG  *+200
```

Figure 37. ORG Statement with an Asterisk Operand and Address Adjustment

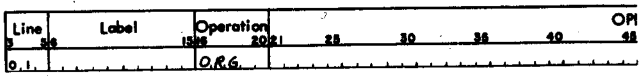


```
01        ORG  *+X00
```

Figure 38. ORG Statement Advancing Address Assignment to Next Available Multiple of 100

constant was assigned location 00725, the next instruction would have an address of 00800.

Figure 39 shows an ORG statement with a blank operand. The processor will assign addresses to subsequent entries beginning at the location designated by the high assignment counter plus one.



```
01        ORG
```

Figure 39. ORG Statement with a Blank Operand

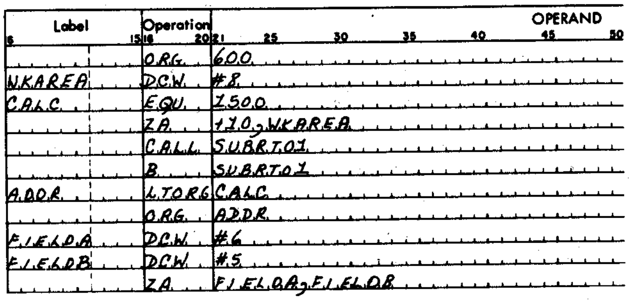
LTORG — Literal Origin

LTORG statements are coded in the same way as ORG statements. Their function is to direct the processor to assign storage locations to previously encountered literals and closed library routines, beginning with the address written in the operand field of the LTORG statement. LTORG statements can appear anywhere in the source program.

If no LTORG statement appears in the source program, the processor begins assigning addresses to literals and closed library routines when it encounters an EX or END statement.

Example: Figure 40 shows how the programmer can direct the processor to begin assigning the storage locations to literals and closed library routines.

The programmer has instructed the processor to begin storage allocation at 00600. All instructions, constants, and work areas (ending with B SUBRT 01) will be assigned storage. However, the literal (+10) in the statement ZA + 10, WKAREA, and the library routine (SUBRT 01) extracted by the CALL macro (see "Call") will not be assigned storage until the LTORG statement is encountered. The first instruction in the library routine (SUBRT 01) will be assigned address 01500 (because CALC has been equated to 01500). After all instructions in SUBRT 01 have been assigned storage locations, the literal + 10 will be assigned an address. The processor will begin assigning the rest of the instructions, constants, and work areas with the storage location immediately to the right of the area occupied by the instruction B SUBRT 01. Thus, if B SUBRT 01 (J̌ 01500) is assigned locations 00691-00697, FIELDA will be assigned storage locations 00698-00703.



```
          ORG   600
WKAREA    DCW   #8
CALC      EQU   1500
          ZA    +10,WKAREA
          CALL  SUBRT01
          B     SUBRT01
ADDR      LTORG CALC
          ORG   ADDR
FIELDA    DCW   #6
FIELDB    DCW   #5
          ZA    FIELDA,FIELDB
```

Figure 40. Using a LTORG Statement

EX — Execute

During the loading of the assembled machine-language program, the programmer may want to discontinue the loading process temporarily in order to execute portions of the program just loaded. The EX statement is used for this purpose.

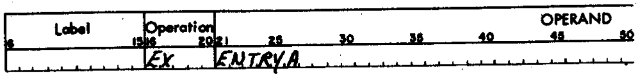
The programmer:

1. Writes the mnemonic operation code EX in the operation field.
2. Writes an actual or symbolic address in the operand field. This address must be the same symbol that appears in the label field of the first instruction to be executed.

The processor:

1. Incorporates closed library routines, literals, and address constants.
2. Assembles a branch instruction (an unconditional branch to the first instruction to be executed), the I-address of which is the address assigned to the instruction referenced by the symbol in the operand field. This instruction does not become part of the assembled machine-language program, but it causes the processor-produced loading routine to halt the loading process at the appropriate time and execute the branch instruction. NOTE: To continue the loading process after the desired portion of the program has been executed, the programmer must provide re-entry to the load routine. (The IBM standard re-entry point is 00281.)

Example: Figure 41 shows how an EX statement can be coded. When this statement is encountered in the loading data, the loading process halts and a branch to the instruction whose label is ENTRYA occurs.



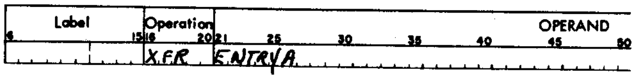
```
          EX    ENTRYA
```

Figure 41. EX Statement

XFR – Transfer

This entry has the same function as an EX statement except that literals, closed library routines, and address constants are not incorporated. An XFR statement transfers to and executes instructions which have been previously loaded.

Example: Figure 42 shows an XFR entry.



Label	Operation	OPERAND
	XFR	ENTRYA

Figure 42. XFR Statement

SFX – Suffix

This statement directs the processor to put a suffix code in the tenth position of all subsequent symbolic labels in a source program section which have less than ten characters until another SFX statement is encountered. In this way, the programmer can use the same label in different sections of the complete program.

The programmer:

1. Writes the mnemonic operation code (SFX) in the operation field.
2. Writes the character (which can be any valid 1410 alphabetic character) to be used for the suffix code in the operand field.

The processor:

1. Inserts the suffix code in the tenth position of all subsequent symbolic labels in the subsequent entries which have less than ten characters.
2. Changes the suffix code when a new SFX card is encountered.

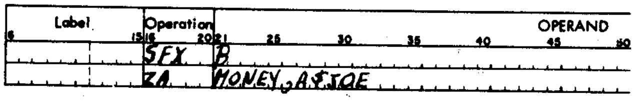
Cross Referencing with Suffixing: If the programmer wishes to cross reference to a previously used label which is in a section with a different suffix, he may do so by writing the suffix of the different section followed by a dollar sign before the label in the operand.

If JOE appeared as a label in a program section with a suffix A and the given statement is in a section with a suffix B, he may refer to JOE by cross referencing as indicated in Figure 43.

To cross reference a label contained in a section to which no suffix has been assigned, precede the label with a dollar sign (Figure 44).

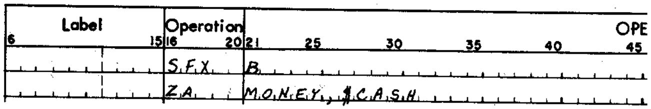
The programmer can instruct the processor to discontinue suffixing by using a SFX card with a blank operand.

Note: Labels beginning with the letters IOCS, and DTF file names, are never suffixed.



Label	Operation	OPERAND
	SFX	B
	ZA	MONEY,A$JOE

Figure 43. Cross Referencing Suffixing



Label	Operation	OPE
	SFX	B
	ZA	MONEY,$CASH

Figure 44. Cross Reference to Unsuffixed Label

PST – Print Symbol Table

This entry causes the processor to print out the symbol table ahead of the printed listing of the program.

The programmer writes the mnemonic operation code (PST) in the operation field.

The processor lists the symbol table. All labels used in the source program are printed with their assigned core-storage addresses. Note: This card can appear anywhere in the source program deck preceding the END card.

END – End

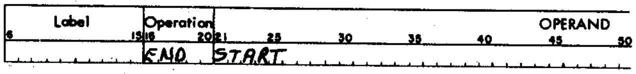
This is always the last card in the source deck. It signals the processor that all of the source program entries have been read, and provides the processor with the information necessary to create an execute card. This execute card causes a transfer to the first instruction to be executed after the program has been loaded into the machine at program load time. Thus, program execution begins automatically.

The programmer:

1. Writes the mnemonic operation code (END) in the operation field.
2. Writes, in the operand field, the symbolic or actual address of the first instruction to be executed after the program has been loaded.

The processor creates an unconditional branch instruction which is used as part of the loading data. Other processor functions are the same as for an EX statement.

Example: Figure 45 shows an END card.



Label	Operation	OPERAND
	END	START

Figure 45. END Card

Diagnostic Aids

Error flags appear in the listing of Autocoder source statements obtained during the assembly run.

The flags and their meaning are as follows:

F indicates a format error.

U indicates an undefined label.

M indicates a label that has been defined more than once.

O indicates an invalid operation code.

Autocoder imperative operations are the symbolic statements for the machine language instructions to be executed in the object program. Most of the statements written in a source program will be imperative instructions. Although the Autocoder processor can assemble instructions with all the imperative operation code mnemonics, the programmer must remember the particular special features and devices that will be included in the object machine that will be used to execute the program he is writing.

The imperative operation mnemonic codes are listed in the "Mnemonic Operation Codes" section of this publication.

The programmer:

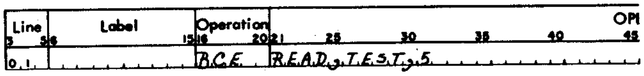
1. Writes the mnemonic operation code for the instruction in the operation field.
2. If the instruction is an entry point for a branch instruction elsewhere in the program or if the programmer wishes to make other reference to it, it should have a label. This label will be assigned an actual address equal to the address of the operation code of the assembled machine-language instruction. Thus, the programmer can use this label as the symbolic I-address of a branch instruction elsewhere in the program (see Figure 47).
3. Writes the symbolic address of the data, devices, or constants in the operand field. The first symbol will be used as the A- or I-address of the imperative instruction. If the instruction also requires a B-address, a comma is written following the first symbol and its address adjustment and/or indexing codes (if any), then the symbol for the B-address is written. If the instruction requires that a d-character be specified, a comma and the actual d-character follow the symbolic entries for the B-address or an I-address if the B-address is not needed.

The processor assembles the object instruction as follows:

1. Substitutes the actual machine-language operation code for the mnemonic written in the operation field.
2. Substitutes the actual addresses of symbols used in the operand field to specify the X-control field A or I, and B-addresses of the instructions. If address adjustment or indexing is indicated, the substituted address will reflect these notations (tag bits will be inserted for indexing and addresses will be altered by adding or subtracting the adjustment factor if address adjustment is specified). The d-character will be supplied automatically for unique mnemonics, or will be taken from the operand field if the programmer has supplied it.
3. Assigns the actual machine-language instruction an area in storage. The address of this area is the position which the operation code occupies in object machine core storage. This address is assigned to the label if one appears in the label field.

Result: This instruction will be placed in the object program deck. A word mark will be set in the operation code position at program load time.

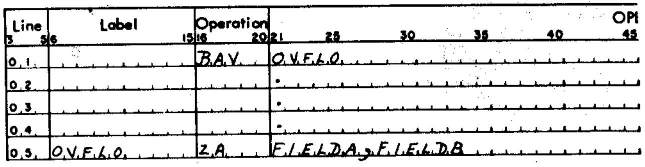
Examples
Figure 46 shows an imperative instruction with I- and B-operands and a mnemonic which requires that the programmer include the d-character. A branch to a location labeled READ will occur if the location labeled TEST has a 5 in it. Assuming that the address of READ is 00596 and TEST is in 00782, the assembled instruction is $\check{\text{B}}$ 00596 00782 5.



Line 3 5	Label 6 15	Operation 16 20	OPI 21 25 30 35 40 45
0 1		BCE	READ,TEST,5

Figure 46. Branch-if-Character-Equal

Figure 47 shows an imperative instruction with a unique mnemonic. A branch to a location labeled OVFLO will occur if an arithmetic overflow has occurred. Assuming that the address of OVFLO is 00896 the assembled machine-language instruction is $\check{\text{J}}$ 00896 Z.



Line 3 5	Label 6 15	Operation 16 20	OPI 21 25 30 35 40 45
0 1		BAV	OVFLO
0 2			.
0 3			.
0 4			.
0 5	OVFLO	ZA	FIELDA,FIELDB

Figure 47. Branch-if-Arithmetic-Overflow

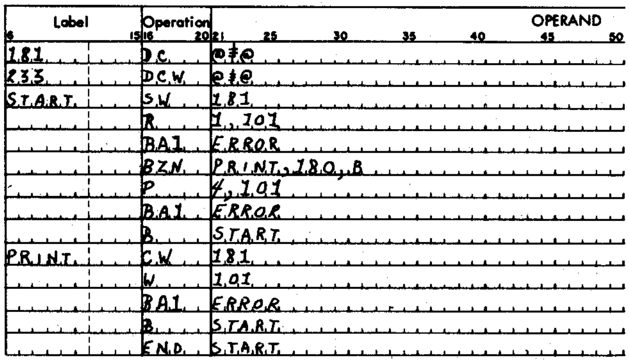
NOTE: Unique Mnemonics. Several mnemonic operation codes have been developed to relieve the programmer of coding the d-character in the operand field of symbolic imperative instructions. However, some operation codes have so many valid d-characters that it is impractical to provide a separate mnemonic for each.

In these cases, the programmer supplies the d-character as previously described. In the listing of mnemonic operation codes for imperative instructions all mnemonics which require that the d-character be included in the operand field are indicated by a D in the operand column and in the d-modifier position of the assembled machine-language instruction.

Coding

Figure 48 shows a brief routine illustrating a section of Autocoder coding.

Several imperative operations are governed by special rules, and care must be taken when coding with these instructions. The special cases are described in the following paragraphs.



Label	Operation	OPERAND
181	DC	@‡@
233	DCW	@‡@
START	SW	181
	R	1,101
	BA1	ERROR
	BZN	PRINT,180,B
	P	4,101
	BA1	ERROR
	B	START
PRINT	CW	181
	W	101
	BA1	ERROR
	B	START
	END	START

Figure 48. Autocoder Coding

Data-Move Instructions

The data-move command is controlled in machine language by the op code D. The actual conditions of the various types of data-move instructions are regulated by the d-character. To make the Autocoder language more meaningful, each of these move and scan instructions has a different mnemonic op code. Each of these mnemonics specifies the type of operation, the direction of the move or scan, the nature of the data to be moved, and what terminates the operation. The following rules apply in constructing the mnemonics for data-move commands:

MOVE MNEMONICS

1. The first character of the mnemonic is M.

2. The second character specifies the direction of data movement.

   L — Right-to-left movement.

   R — Left-to-right movement.

3. The third section of the mnemonic specifies the portion of data moved.

   N — Move numerical portion of data.

   Z — Move zone portion of data.

   C — Move whole characters.

   W — Move word marks.

   NW — Move numerical portion and word marks.

   ZW — Move zone portion and word marks.

   CW — Move whole characters and word marks.

4. The final character of the mnemonic specifies what terminates the move.

   a. To terminate right-to-left move:

   A — Word mark in A-field.

   B — Word mark in B-field.

   (Blank) — Word mark in either field.

   S — Move single location only.

   b. To terminate left-to-right move:

   R — Record mark in A-field.

   G — Group mark with a word mark in A-field.

   M — Record mark or group mark with a word mark in A-field.

   (Blank) — Word mark in either field.

SCAN MNEMONICS

1. The first three characters are always SCN.

2. The fourth character specifies the direction of scan.

   L — Right-to-left scan.

   R — Left-to-right scan.

3. The fifth character specifies what terminates the scan.

   a. To terminate right-to-left scan:

   A — Word mark in A-field.

   B — Word mark in B-field.

   (Blank) — Word mark in either field.

   S — Scan left single position.

   b. To terminate left-to-right scan:

   R — Record mark in A-field.

   G — Group mark with a word mark in A-field.

   M — Record mark or group mark with a word mark in A-field.

   (Blank) — Word mark in A- or B-field.

For example, when whole characters and word marks are to be moved from right to left, terminating the move on a word mark in the A-field, the Autocoder mnemonic op code is MLCWA.

SSF — Select Stacker and Feed

This instruction causes the last card transferred to storage to be selected into the stacker specified in the operand field of the instruction. A blank operand causes the card to be selected into the zero read pocket. A 1 in the operand field causes the card to be selected into stacker 1. A 2 in the operand field causes the card to be selected into stacker 8/2.

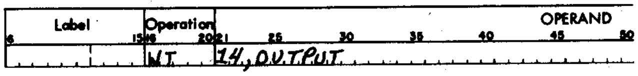
Magnetic Tape Commands

Mnemonics referring to magnetic tape do not require d-characters. However, it is necessary to specify, in the operand, the number of the tape unit and channel needed for the operation. This can be done in one of three ways. The programmer can:

1. Assign a label to the channel and tape unit as described in EQU and use it as the A-operand of a tape instruction.

2. Write the number of the channel and tape unit in columns 21 and 22 of the tape instruction. The assembled instruction for the symbolic entry shown in Figure 49 will cause a record to be written on tape unit 4 using the data beginning in a storage area labeled OUTPUT.

3. Write the X-control field as the A-operand of the tape instruction.



Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50
	WT	14,OUTPUT

Figure 49. Write Tape

Disk Commands

All input-output commands involving disk units must specify the channel (1 or 2) as the first entry of the operand field. If an address is used in the operand, it follows the channel designation and is separated from it by a comma as shown in Figure 50.

BZN — Branch on Zone

The form of this command is:

BZN i, addr, ch

A branch to i will occur if the character at addr has the zone-bit configuration specified by ch.

Permissible operands are as follows:

i — May be any symbolic or absolute address (indexing and address adjustment are permitted).

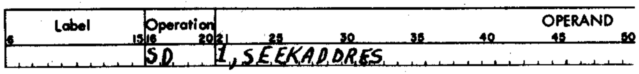
addr — May be any symbolic or absolute address (indexing and address adjustment are permitted).

ch — May be:

A or ¢ specifying an A zone bit.
B or − specifying a B zone bit.
AB or + specifying A and B zone bits.
b specifying absence of zone bits.

The address for i, or addr, or both, may be omitted if the operation is chained. Acceptable forms of this operation are:

BZN i, addr, ch
BZN i
BZN



Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50
	SD	1,SEEKADDRES

Figure 50. Disk Storage Instruction

BWZ — Branch if Word Mark, Zone, or Both

This operation is the same as BZN except that a branch also takes place if a word mark is present.

BCE — Branch if Character Equal

The operand of this command takes the form i, addr, ch where:

ch = Character to be matched.
addr = Address of character to be compared.
i = Address to be branched to.

Permissible forms of these operations are:

BCE i, addr, ch
BCE i
BCE

BBE — Branch if Bit Equal

The operand of this command will take the form i, ADDR, ch where:

ch = Character containing bit(s) to be tested for.
ADDR = Address of character to be tested.
i = Address to be branched to.

Permissible forms of this operation are:

BBE i, addr, ch
BBE i
BBE

CC — Control Carriage

The forms control character must be written in the operand field of this instruction. Standard forms control characters are to be used.

P — Punch

The pocket into which the punched card will be selected must be specified as the first entry of the operand field of this instruction. The address from which data will be punched is specified following the stacker specifications and is separated from it by a comma. A 0-punch selects punched cards into stacker pocket 0; a 4-punch selects punched cards into stacker pocket 4; an 8-punch selects cards into stacker pocket 8/2.

R — Read

A read command must have as the first entry in its operand either the number of the stacker into which the card is to be selected after reading, or an indication that a select stacker command will follow the read command. A 0-punch selects cards into stacker pocket 0; a 1-punch, into stacker pocket 1; and a 2-punch, into stacker pocket 8/2. A 9-punch indicates that a select stacker command will follow. The address (symbolic or actual) of the storage area into which the data from the card is to be read must be the second entry in the operand of a read command.

Input/Output Commands

Valid forms of the mnemonic operation codes for input/output devices are listed by device in the "Mnemonic Operation Codes" section of this publication.

PRIORITY PROCESSING

IBM 1410 Data Processing Systems equipped with the priority-processing feature can process I/O no-op commands. To code these in Autocoder language, write an N as the first character of the I/O mnemonic. For example, the instruction shown in Figure 51 will be assembled as M̌ %U1 00100 Q.

The instruction shown in Figure 52 will be assembled as M̌ %U1 00100 V.

The I/O no-op instruction will set the appropriate I/O external indicators, but no data movement takes place.

NOTE: Like any other I/O instruction, the I/O no-op instruction *always* sets the I/O interlock latch on. This latch must be set off before another I/O instruction can be executed on the same channel.

The I/O interlock latch can be set off by one of the following *classes* of instructions:

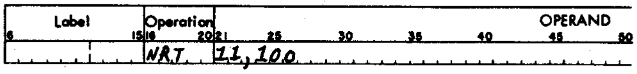
1. Branch Any External Indicator – Channel (1 or 2) (i.e., BA1 or BA2) or
2. Branch on External Indicator – Channel (1 or 2) (i.e., BEX1 d or BEX2 d, where d ≠ ‡), *provided the branch is executed.* (If the branch is *not* executed, the I/O interlock latch is *not* turned off.)

NOPWM – No Operation Word Mark

The 1410 Autocoder permits the programmer to set programmed no-op switches easily. If the statement shown in Figure 53 is written in the source program, the processor will insert in the object program the operation code Ň (no-op) with a word mark, followed by the branch instruction (JXXXXX) without a word mark in the operation code position. Subsequent instructions in the object program can then be used to set and clear the word mark in the operation code position of the branch instruction as needed. If there is no word mark, the branch instruction will be ignored, and if the word mark is present, the branch instruction will be executed. The assembled instructions produced by the entries shown in Figure 53 are Ň J 00500 (assuming start is in location 500).

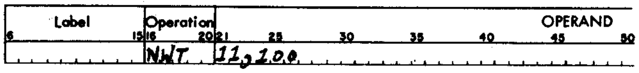
NOTE 1: Any instruction (not only branch instructions) may be bypassed by this method.

NOTE 2: Use of a regular NOP-instruction in place of a NOPWM instruction will cause an opposite setting of the switch.


6
15
16
20
21
25
30
35
40
45
50

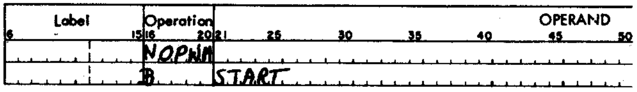
Label	Operation	OPERAND
	NRT	11,100

Figure 51. I/O No-Op Input Command


6
15
16
20
21
25
30
35
40
45
50

Label	Operation	OPERAND
	NWT	11,100

Figure 52. I/O No-Op Output Command


6
15
16
20
21
25
30
35
40
45
50

Label	Operation	OPERAND
	NOPWM	
	B	START

Figure 53. No-Op Word Mark

BEX1 or BEX2 – Branch on External Indicator

These mnemonics are used for the machine-language op codes R and X. BEX1 is equal to R; BEX2 is equal to X. One of the two, depending on channel, is used when testing for a combination of external indicator conditions for which there is no mnemonic. The symbolic operand must take the form addr, d where:

addr = Address to be branched to, if any of the external indicators specified in the d-character have been set as a result of executing an I/O command.

d = the actual character formed by the combination of d-character-control bits of the individual external condition tests.

For example, a branch to a location labeled EXIT is desired if the channel-busy indicator, the not-ready indicator, or the wrong length record indicator has been set following an I/O command. The appropriate Autocoder instruction has BEX1 as the operation code, EXIT as the addr and L as the d-character. The d-character L results from a combination of the 2-bit, 1-bit, and B-bit required to interrogate the three indicators just mentioned.

The Macro System

Many of the routines that must be incorporated in programs written for the IBM 1410 are general in nature and can be used repeatedly with little or no alteration. The IBM 1410 Autocoder makes it possible for the user to write a single symbolic instruction (a *macro-instruction*) that causes a series of machine-language instructions to be inserted automatically in the object program. Thus, the ability of Autocoder to process macro-instructions relieves the programmer of much repetitive coding. With a macro-instruction, the programmer can extract, from a library of routines, a sequence of instructions tailored by the processor to fit his particular program.

Definitions of Terms

Several programming terms are used to describe the requirements and operational characteristics of the macro system. These terms are explained here as they are applied in the following discussions.

Object Routine. The specific machine-language instructions needed to perform the functions specified by the macro-instruction. If the object routine is inserted directly in a larger routine (for example, the main routine) without a linkage or calling sequence, it is called an *open routine* or in-line routine. If the routine is not inserted as a block of instructions within a larger routine, but is entered by basic linkage from the main routine, it is called a *closed routine,* or out-of-line routine.

Model Statement. A general outline of a symbolic program entry. Model statements are used only in flexible library routines.

Library Routine. The complete set of instructions or model statements from which the object routine is developed. If the library routine cannot be altered, it is *inflexible*. It is *flexible* if the library routine is designed so that symbolic program entries can be deleted from certain object routines (at the discretion of the programmer) or if parameters can be inserted.

Library. The complete set of library routines stored on magnetic tape with an identifying label for each routine that can be extracted by a macro-instruction. Several macro-instructions and library routines are provided by IBM. Others are designed by the user to suit particular processing requirements.

Librarian. The phase of the processor that creates the library tape from card input. After the original writing of the library tape, this phase is used to insert additional library routines and their identifying labels, as well as to update routines. This phase is omitted during program assembly.

Parameters. The symbolic addresses of data fields, control names, or information to be inserted in the symbolic program entries outlined by the model statements. By placing parameters in the operand field of a macro-instruction, the programmer can specify symbolically the data to be operated on. The actual addresses of the data (or other information) are inserted in the object routine by the processor during assembly. Also, literals and actual addresses can be used.

Pseudo-macro. A macro-instruction that is used internally by the processor to control the production of a series of machine-language instructions. The difference between a pseudo-macro and a macro is that the pseudo-macro is not written in the source program but appears only in a flexible library routine.

Macro Operations

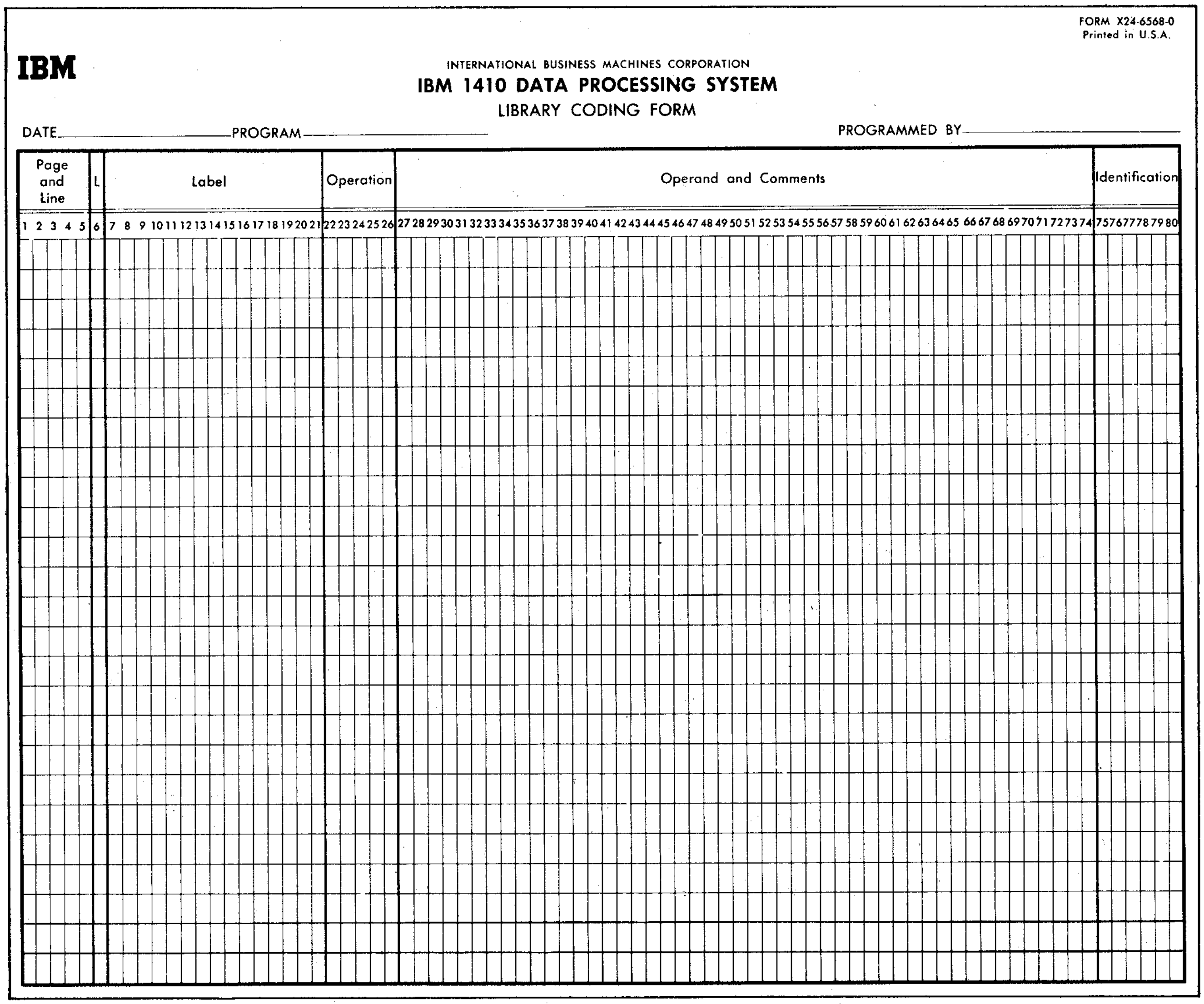
To illustrate the basic operation of the macro system, a hypothetical macro called CHECK with a simple flexible library routine is used. The routine is designed to compare two fields, and test the compare indicator for a high, low, or equal condition.

Figure 54 shows the library coding form which is used to write library routines.

Figure 55 shows the library entry, a macro-instruction specifying that all instructions in the library routine appear in the object program, and the symbolic program entries created during the macro phase of Autocoder. The symbolic program entries are inserted in the source program following the macro-instruction. During assembly of the object program, the symbolic program entries will be translated to actual machine-language instructions with the actual addresses of the parameters inserted in the label, operation, and operand fields.

The Library Entry

The library entry for the CHECK macro-instruction consists of the four model statements shown in Figure 55. This entry is placed on the library tape and identified by an INSER statement. (Refer to Figure 95 for an example of this use of INSER.)



FORM X24-6568-0
Printed in U.S.A.

IBM

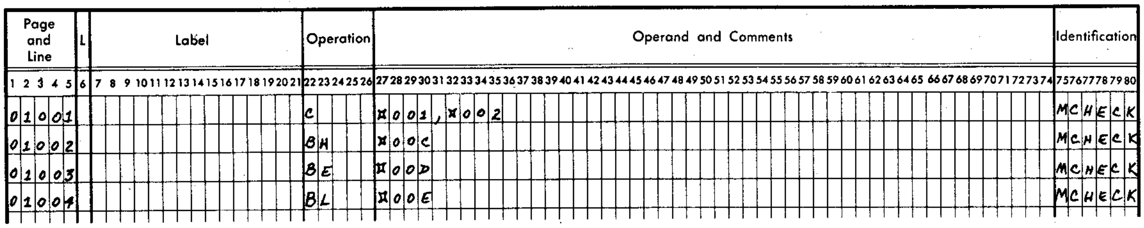
INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1410 DATA PROCESSING SYSTEM
LIBRARY CODING FORM

DATE________________PROGRAM________________ PROGRAMMED BY________________

Page and Line	L	Label	Operation	Operand and Comments	Identification
1 2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74	75 76 77 78 79 80

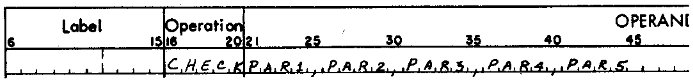
Figure 54. Library Coding Form

Library Entry



Page and Line	L	Label	Operation	Operand and Comments	Identification
1 2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74	75 76 77 78 79 80
01001			C	⌑001,⌑002	MCHECK
01002			BH	⌑00C	MCHECK
01003			BE	⌑00D	MCHECK
01004			BL	⌑00E	MCHECK

Macro Instruction



Label	Operation	OPERANI
6 15	16 20	21 25 30 35 40 45
	CHECK	PAR1,PAR2,PAR3,PAR4,PAR5

Assembled Symbolic Program Entry

```
C     PAR1,PAR2
BH    PAR3
BE    PAR4
BL    PAR5
```

Figure 55. Macro Operations

INSER — Insert

An INSER statement identifies a library routine. The INSER statement causes an identification record to be generated and placed on the library tape preceding the library routine it identifies.

The programmer:

1. Writes the operation code INSER in the operation field of the Autocoder coding sheet.
2. Writes the five-character label for the library routine in the label field. The label will be the same as the name that appears in the operation field of the associated macro-instruction except when either the CALL or INCLD macro is used.
3. Writes an M in column 21 of the operand field to indicate a flexible library routine, or an S in column 21 to indicate a CALL or INCLD type library routine.

The processor puts the indicative information ahead of the model statements in the library tape during the librarian phase of Autocoder.

Result: During assembly, the header label is matched with the macro name in the operation field of the macro-instruction. The model statements following the header label in the library tape are used to assemble the symbolic program entries as specified by the macro-instruction.

Model Statements

Model statements establish the conditions for inserting parameters in the object routine and define the basic structure of the symbolic program entries.

The programmer:

1. Designs a general routine to perform many specific functions (depending upon the parameters supplied) when it is executed in the object program.
2. Writes the model statement as follows:
   a. If the entry is complete, it is written exactly the same as though it were an entry in a source program. This entry will be included in all object routines unless a bypass condition exists (see "BOOL").

   Example: Figure 56.

   b. If the entry is incomplete, the programmer writes a special four-character code to indicate that a certain parameter from the macro-instruction operand field *must* be inserted in its place. This code is a ⌑ followed by a number from 001 to 199, the position of the parameter in the macro-instruction. This entry will be inserted in all object routines.

L	Label	Operation	Operand and Comments
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
		C	FIELDA,FIELDB

Figure 56. Model Statement for a Complete Instruction

L	Label	Operation	Operand
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
		C	⌑001,⌑002

Figure 57. Model Statement for an Incomplete Instruction with Required Parameters

   Example: Insert parameters 001 and 002 specified by the CHECK macro-instruction shown in Figure 57.

   c. If the entry is incomplete the programmer writes a ⌑ followed by a number from 001-199 with AB-bits over the units position (parameter 001 is ⌑ 00A, parameter 002 is ⌑ 00B, etc.) to indicate that the entry is to be included in the object routine only if the parameter is specified by the macro-instruction.

   Example: Insert parameter 003 in the following instruction if it is specified by the macro-instruction. If parameter 003 does not appear in the macro-instruction, the instruction shown in Figure 58 will be deleted from the object routine.

L	Label	Operation	Operand
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
		B	⌑00C

Figure 58. Model Statement for an Incomplete Instruction with Conditional Parameters

NOTE: Substitution codes can also be used to substitute a parameter in any part of a model statement. For example, it is possible to substitute an operation code, any part of a literal, a label, etc.

Labeling: If the model statement represents an instruction entry point for a branch instruction elsewhere in the program, it should have a label. The label of the macro-instruction causes a generated label EQU* in the assembled object routine as shown in Figure 59.

If additional external labels are required and specified as parameters in the macro-instruction they can

Macro Instruction

Label	Operation	O
6 15	16 20	21 25 30 35 40
TEST2	INVER	START1

Model Statement

L	Label	Operation	Operand
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
		B	⌑001

Assembled Symbolic Program Entry

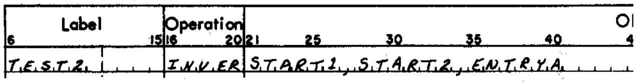
```
TEST2  EQU  *
       B    START1
```

Figure 59. Labeling

be inserted in the label field of the symbolic program entry by using □ 001-199 code.

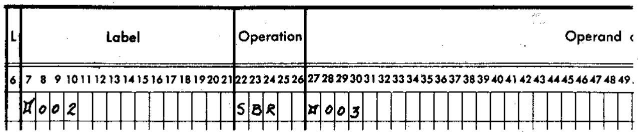
Example: Insert parameter 002 in the label field of the assembled symbolic program entry as shown in Figure 60.

Macro Instruction



Label	Operation	Operand
TEST2	INVER	START1,START2,ENTRYA

Model Statement



L	Label	Operation	Operand
	□002	SBR	□003

Assembled Symbolic Program Entry

```
START2      SBR   ENTRYA
```

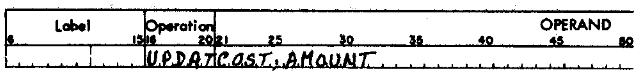
Figure 60. Additional External Labels

Symbolic Addressing within the Library Routine. To allow symbolic reference to other instructions in a flexible library routine a □ followed by a number from 001 to 199 with a B-bit over the units position (□ 00J = symbolic address 1, □ 00K = symbolic address 2, etc.) can be used. The processor generates the symbolic address if the code, for example, □ 00J is used as a label for one entry and as an operand of at least one other entry in the same library routine.

Internal labels within flexible routines are generated in the form □ nnnmmm, where nnn is the code (00J-19R), and mmm is the number of the macro within the source program. This is done to avoid duplicate address assignments for labels.

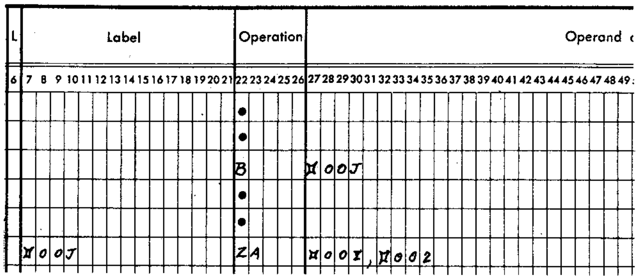
Example: Use the generated symbolic address of □ 00J as an operand for entry 3 and as the label for entry 6. UPDAT is the 23rd macro encountered in the source program (Figure 61).

Macro Instruction



Label	Operation	Operand
	UPDAT	COST,AMOUNT

Model Statement



L	Label	Operation	Operand
		●	
		●	
		B	□00J
		●	
		●	
	□00J	ZA	□001,□002

Assembled Symbolic Program Entry

```
              ●
              ●
              B       □00J023
              ●
              ●
□00J023       ZA      COST,AMOUNT
```

Figure 61. Internal Labels

Address Adjustment and Indexing: The parameters in a macro-instruction and the operands in partially complete instructions in a library routine can have address adjustment and indexing.

If address adjustment is used in both the parameter and the instruction, the assembled instruction will be adjusted to the algebraic sum of the two. For example, if the address adjustment of one is +7 and the other is −4, the assembled instruction will have address adjustment equal to +3.

Operands may be indexed in the library routine. If a parameter supplied by the macro-instruction is indexed, it will be cancelled by the indexing in the library routine.

Literals: Operands of instructions in flexible routines may use literals as required. However, these literals may not contain the @ symbol within an alphamerical literal.

NOTE 1: A model statement in the library routine for a macro-instruction may not be another macro-instruction, except the CALL or INCLD macro (see "Call").

NOTE 2: Literal Origin, Ex and End statements cannot be used in library routines.

The processor enters model statements in the library tape immediately following the header statement during the librarian phase of Autocoder.

Result: Any library routine can be extracted by writing the associated macro-instruction in the source program.

Figure 62 is a summary of the codes that can be used in the model statements of flexible library routines.

CODE	POSITION	FUNCTION
□001-□199	Statement	Substitute parameter (parameter must be present)
□00A-□191	Statement	Substitute parameter (if parameter is missing, delete statement)
□00J-□19R	Label Field and Operand Field	Assign internal label

Figure 62. Model Statement Codes

Macro-Instructions

A macro-instruction is the entry in the source program that causes a series of instructions to be inserted in a program.

The programmer:

1. Writes the name of the library routine in the operation field. This name must be the same five characters that appear in the label field of the INSER statement of the library entry.

2. Writes in the label field the label that is to be inserted in the label field of the first assembled model statement.

3. Writes in the operand field the parameters that are to be used by the model statements that are required for the particular object routine desired, as follows:

   a. Parameters must be written in the sequence in which they are to be used by the codes in the model statements. For example, if cost is parameter 001, it must be written first so that it will be substituted wherever a □001, or □00A appears as a label, operation code, or operand of a model statement.

   b. As many parameters may be used as can be contained in the operand fields of five or fewer coding sheet lines. If more than one line is needed for a macro-instruction, the label and operation fields of the additional lines must be left blank. Parameters must be separated by a comma. They cannot contain blanks or commas unless they appear between @ symbols. The @ symbol itself cannot appear between @ symbols. If parameters for a single macro-instruction require more than one coding sheet line, the last parameter in each line must be followed immediately by a comma. The last parameter in a macro-instruction need not be followed by a comma.

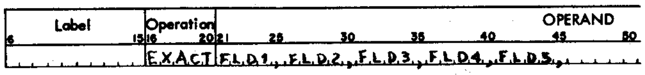
   c. Parameters that are not required for the particular object routine desired can be omitted from the operand field of the macro-instruction. However, if a parameter is omitted, the comma that would have followed the parameter must be included, unless the omitted parameter is behind the last parameter which is included in the macro-instruction. These commas are necessary to count parameters up to the last included parameter. All parameters between the last included parameter and parameter 199 are assumed by the processor to be absent.

Figures 63, 64, 65, and 66 show how parameters can be omitted. The hypothetical macro-instruction called EXACT is used. EXACT can have as many as nine parameters.

The processor:

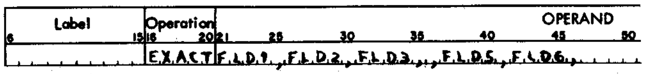
Extracts the library routine and selects the model statements required for the object routine as specified by the parameters in the macro-instructions and by the substitution and condition codes in the model statements.

Result: The resulting program entries are merged with the source program entries behind the macro-instruction.



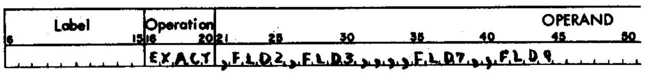
Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
	EXACT	FLD1,FLD2,FLD3,FLD4,FLD5,

Figure 63. Parameters for EXACT Included; Parameters 006-199 Missing



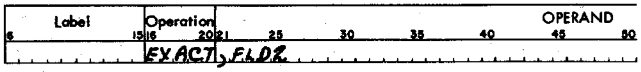
Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
	EXACT	FLD1,FLD2,FLD3,,FLD5,FLD6,

Figure 64. Parameters 004 and 007-199 Missing



Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
	EXACT	,FLD2,FLD3,,,,FLD7,,FLD9

Figure 65. Parameters 001, 004, 005, 006, 008 and 010-199 Missing



Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
	EXACT	,FLD2

Figure 66. Parameters 001, 003-199 Missing

Pseudo-Macro-Instructions

These statements never appear in a user's source program or in the output listing of an assembled 1410 Autocoder program. However, they are used in library routines to signal the processor that certain conditions exist that can affect the assembly of an object routine. For example, the presence of a pseudo-macro-instruction in a library routine can cause a group of model statements to be deleted. Thus, pseudo-macros provide the writer of library routines with a coding flexibility which exceeds the limitations of the substitution and condition codes described previously.

Pseudo-macro-instructions may be written anywhere in a library routine. The five pseudo-macros incorporated in the 1410 Autocoder processor are MATH, BOOL, COMP, NOTE, and MEND.

Permanent and Temporary Switches

The MATH, BOOL, and COMP pseudo-macros use internal indicators (switches) to signal the processor of existing status conditions.

There are 99 permanent and 199 temporary switches available for recording status conditions. Each switch occupies one core-storage position during the macro phase of Autocoder. If a storage position contains the character A (BA 1-bits), the switch is on; if it contains a ? (CBA 82-bits), the switch is off. At the beginning of assembly all switches are off.

PERMANENT SWITCHES

Permanent switches retain status conditions during the entire macro phase unless changed by a pseudo-macro. They are addressed by using a # symbol followed by the three-digit number of the switch to be set or tested. For example, # 001 addresses permanent switch 001; # 002 addresses switch 002; and # 099 addresses switch 099.

TEMPORARY SWITCHES

When the processor encounters a macro-instruction, the temporary switches are set to the condition (presence or absence) of the parameters in the operand of the macro field. If the parameter is present, the corresponding switch is set on. If the parameter is missing, the switch is set off. For example, if parameter 001 is present, temporary switch 001 is turned on. If parameter 002 is missing from the macro-instruction, temporary switch 002 is off. Temporary switches retain status throughout the processing of a macro-instruction unless changed by a pseudo-macro. After the macro-instruction has been completely processed, all temporary switches are set off. Temporary switches are addressed by using a ⌑ symbol followed by the three-digit number of the switch to be set or tested. For example, ⌑ 001 addresses temporary switch 001, ⌑ 002 addresses switch 002, and ⌑ 199 addresses switch 199.

For example, if a macro with a maximum of nine parameters is encountered, the processor sets the first nine temporary switches to indicate the presence or absence of these nine parameters. Temporary switches 010-199, which are off, can be used by the pseudo-macros to communicate conditions to the processor while it is working on this particular macro-instruction. This use of temporary switches is recommended because it reserves the permanent switches for communicating information from one macro to another.

MATH — For Solving Algebraic Expressions

A MATH pseudo-macro contains as operands: sum boxes, arithmetic expressions, and sign switches.

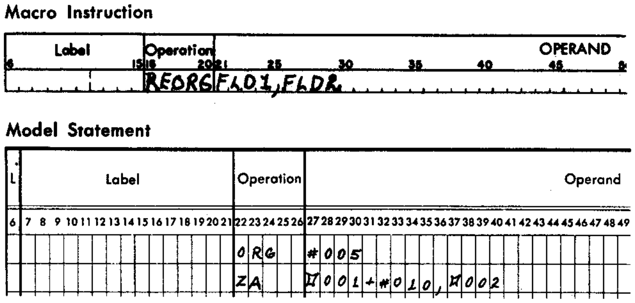
SUM BOXES

A sum box is a group of five core-storage positions used to store the result of an arithmetic expression. The 1410 Autocoder makes available 20 such sum boxes. A sum box is addressed by using a # symbol followed by the three-digit number (ending in zero or five) of the sum box to be referenced. For example, the address of the first sum box is # 005; the address of the second sum box is # 010; and the address of the twentieth sum box is # 000.

At the beginning of the macro phase, a sum box contains $0000\overset{+}{0}$. Any number may be placed in a sum box or added to its contents. The units position of the sum box always contains the sign of the result. Sum boxes retain information placed in them throughout the macro phase and their contents may be used and/or changed from one macro-instruction to another.

Sum boxes can be used by model statements as well as by a pseudo-macro. For example, in Figure 67, assume that sum box # 005 contains 12345 and sum box # 010 contains 00015.

NOTE: ZA FLD1 + 0001N, FLD2 is processed as ZA FLD1 – 15, FLD2.



Macro Instruction

Label	Operation	OPERAND
	REORG	FLD1,FLD2

Model Statement

L	Label	Operation	Operand
		ORG	#005
		ZA	⌑001+#010,⌑002

Assembled Symbolic Program Entry

```
ORG   1234E
ZA    FLD1+0001N,FLD2
```

Figure 67. Sum Boxes

ARITHMETIC EXPRESSIONS

Arithmetic expressions within the MATH pseudo-macro use add (+), subtract (–), multiply (*), and divide (/). An @ symbol represents both the left and right parentheses if they are required for the expression. For example, (001 + 12 – 5) 20 is written:

@ 001 + 12 – 5 @ * 20.

Arithmetic operations expressed in the operand field of the pseudo-macro are executed by the MATH pseudo-macro from left to right. The quotient resulting from the divide operation is *not* half-adjusted, and the remainder is lost. At the end of a multiplication operation the five low-order positions of the product are used for the result (the high-order digits are lost). An overflow is ignored.

The result of the arithmetic expression is produced and inserted with its sign in the designated sum box.

SIGN SWITCHES

Permanent and temporary switches may be used to store the sign of the result of an arithmetic expression. The first switch specified in the operand field of the pseudo-macro represents a positive result; the second represents a zero result, and the third represents a negative result. Consequently, one switch is on and the other two are off if the result is either positive or negative. A zero result causes both the zero and positive switches to be set on. It is not necessary to specify all three switches. However, if a switch code is omitted from the operand field, the comma that would have followed the switch code must be present. (This is the same rule that applies to missing parameters in a macro-instruction.)

The programmer:

1. Writes the name of the pseudo-macro (MATH) in the operation field.
2. Writes in the operand field:
   a. the code for the sum box in which the result of the arithmetic expression is to be stored.
   b. the arithmetic expression.
   c. the code for the switch in which the sign(s) of the result are to be stored.

NOTE: A comma must follow the sum box code, the arithmetic expression, and the individual sign-switch codes. Figure 68 shows the format for a MATH pseudo-macro.

The processor:

1. Produces the result of the arithmetic expression.
2. Stores the result in the sum box.
3. Sets the sign switches.

Example: The MATH pseudo-macro shown in Figure 69 multiplies parameter 007 by 401 and adds 12 to the result. The answer is stored in SUMBOX 6 (# 030). If the result is positive, permanent switch 004 is set on; if the result is zero, permanent switches 004 and 006 are set on; if the result is negative, temporary switch 009 is set on.

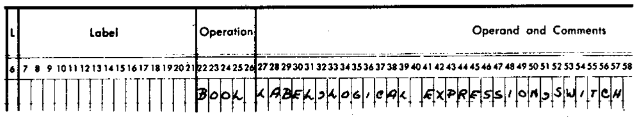
BOOL — For Solving Logical Expressions

The BOOL pseudo-macro can be used:

1. To set a permanent or temporary switch as the result of a logical expression.
2. To cause the processor to skip over certain model statements if the logical expression is false. If the statement is true, the processor goes to the next sequential model statement.

The programmer:

1. Writes the name of the pseudo-macro (BOOL) in the operation field.
2. May write a label, the logical expression (statement), and a switch code in the operand field in the format shown in Figure 70.



L	Label	Operation	Operand and Comments
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
		BOOL	LABEL,LOGICAL EXPRESSION,SWITCH

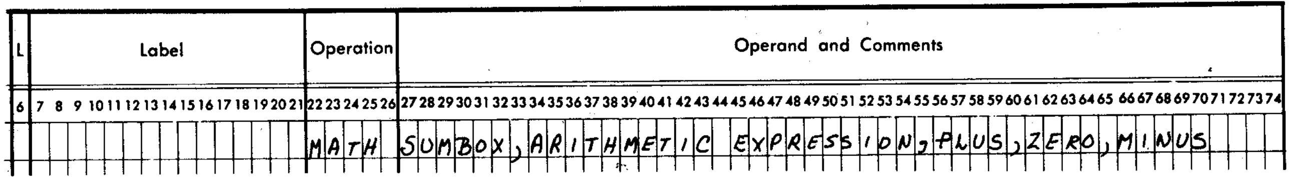
Figure 70. Format for the BOOL Pseudo-Macro

LABELING

A special one-character label permits skipping *forward* in the library routine as the object routine is being assembled by the processor. This one-character label is written in the first position of the operand field of the BOOL pseudo-macro and also in the label position (column 6 of the library coding form) of the first model statement (or command) to be examined after the skip has been initiated. Skipping occurs only if the logical statement is false. The label may be omitted if a skip is not desired, but the comma that would have followed the label must be written in the BOOL statement to indicate that the label is missing. The label can be any alphabetic or numerical character. Special characters are not permitted.

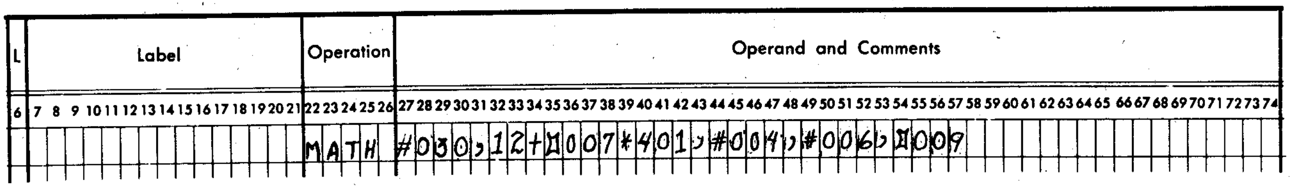
LOGICAL EXPRESSION

The BOOL pseudo-macro can have any combination of three logical operations: * (and), + (or), and − (not). The operators are defined in Figure 71. The combina-



L	Label	Operation	Operand and Comments
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
		MATH	SUMBOX,ARITHMETIC EXPRESSION,PLUS,ZERO,MINUS

Figure 68. Format for the MATH Pseudo-Macro

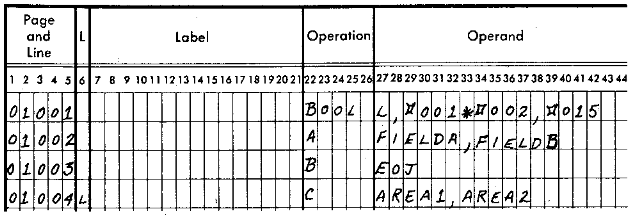


L	Label	Operation	Operand and Comments
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
		MATH	#030,12+⌑007*401,#004,#006,⌑009

Figure 69. MATH Pseudo-Macro

*	+	—
1 * 1 = 1 1 * 0 = 0 0 * 1 = 0 0 * 0 = 0	1 + 1 = 1 1 + 0 = 1 0 + 1 = 1 0 + 0 = 0	—1 = 0 —0 = 1

Figure 71. Table of Operators



Page and Line	L	Label	Operation	Operand
1 2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
01001			BOOL	L,¤001*¤002,¤015
01002			A	FIELDA,FIELDB
01003			B	EOJ
01004	L		C	AREA1,AREA2

Figure 72. Using the BOOL Pseudo-Macro

tion of these operators and the switches to be tested make up the logical expression (see example, Figure 72).

The @ symbol is used to represent both the left and right parentheses.

SWITCHES

Either a permanent or temporary switch may be used to store the result of the logical expression. If the expression is true, the specified switch will be set on. If the expression is false, the specified switch is set off. If no switch setting is desired, a comma must be used to indicate that the switch is missing.

The processor:

1. Examines the status switches to determine whether all conditions specified in the logical expression are satisfied. If they are, the expression is true. If the logical condition is not met, the expression is false.
2. Sets the specified status switch to ON or OFF to reflect the true or false condition.
3. If a false condition exists and a label appears in the BOOL operand, the processor skips forward to the command or model statement containing a corresponding label in its label position.

To determine if a logical expression is true or false:

a. call all ON conditions true and all OFF conditions false.
b. let 1 = true and 0 = false.
c. calculate the logical value of the expression.

If the logical value of the expression is zero, the expression is false. If the logical value is one, the expression is true. For example, if switches 001, 002, 003 and 004 are on, the expression

@ □ 001 * □ 002 @ + @ □ 003 * □ 004 @

is true because:

$$\begin{aligned}(ON * ON) + (ON * ON) &= \\ (1 * 1) + (1 * 1) &= \\ 1 + 1 &= 1\end{aligned}$$

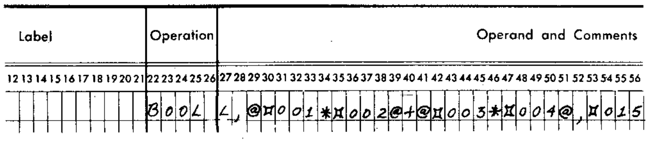
Examples:

Figure 72 shows how the BOOL pseudo-macro can be used. The BOOL entry states:

1. If temporary switches 001 and 002 are on, the statement is true. Therefore, set temporary switch 015 on.
2. However, if either temporary switch 001 or 002 is off, the statement is false. Therefore, set temporary switch 015 off and skip to statement 004.

The example shown in Figure 73 states:

1. If (both temporary switches 001 and 002) or (both temporary switches 003 and 004) are on, the statement is true. Therefore, set temporary switch 015 on.
2. However, if (either temporary switch 001 or 002) and (either temporary switch 003 or 004) is off, the statement is false. Therefore, set temporary switch 015 off and skip to the model statement whose label is L.



Label	Operation	Operand and Comments
12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
	BOOL	L,@¤001*¤002@+@¤003*¤004@,¤015

Figure 73. BOOL Pseudo-Macro

Figure 74 is a table showing all conditions that will cause the BOOL statement shown in Figure 73 to be true.

SWITCHES

	001	*	002	+	003	*	004		LOGICAL VALUE	
CONDITIONS	ON 1	*	ON 1	+	OFF 0	*	OFF 0	=	1	TRUE
	OFF 0	*	OFF 0	+	ON 1	*	ON 1	=	1	
	ON 1	*	ON 1	+	ON 1	*	ON 1	=	1	
	ON 1	*	ON 1	+	ON 1	*	OFF 0	=	1	
	OFF 0	*	ON 1	+	ON 1	*	ON 1	=	1	
	ON 1	*	ON 1	+	OFF 0	*	ON 1	=	1	
	ON 1	*	OFF 0	+	ON 1	*	ON 1	=	1	

Figure 74. True Conditions

Figure 75 is a table showing all conditions that will cause the BOOL statement shown in Figure 73 to be false.

	SWITCHES									
	001	*	002	+	003	*	004		LOGICAL VALUE	
CONDITIONS	OFF 0	*	OFF 0	+	OFF 0	*	OFF 0	=	0	FALSE
	ON 1	*	OFF 0	+	OFF 0	*	OFF 0	=	0	
	OFF 0	*	ON 1	+	OFF 0	*	OFF 0	=	0	
	OFF 0	*	OFF 0	+	ON 1	*	OFF 0	=	0	
	OFF 0	*	OFF 0	+	OFF 0	*	ON 1	=	0	
	OFF 0	*	ON 1	+	OFF 0	*	ON 1	=	0	
	ON 1	*	OFF 0	+	ON 1	*	OFF 0	=	0	
	OFF 0	*	ON 1	+	ON 1	*	OFF 0	=	0	
	ON 1	*	OFF 0	+	OFF 0	*	ON 1	=	0	

Figure 75. False Conditions

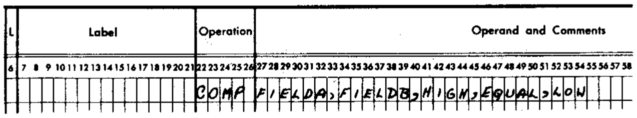
COMP — To Compare Two Fields

The COMP pseudo-macro compares an A-field to a B-field and sets permanent or temporary switches to indicate the result of the comparison.

The programmer:

1. Writes the name of the pseudo-macro (COMP) in the operation field.

2. Writes the operand field in the format shown in Figure 76. The first and second entries are the A- and B-fields. The A- and B-fields may be any of the parameters 001-199, sum boxes # 005-# 000, or literals. Note: For the COMP pseudo-macro, alphamerical literals are not enclosed by @ symbols. They cannot be switches. Entries 3, 4, and 5 are the high, equal, and low switches.



L	Label	Operation	Operand and Comments
		COMP	FIELDA,FIELDB,HIGH,EQUAL,LOW

Figure 76. Format for COMP Pseudo-Macro

NOTE: The codes for the two fields to be compared must be present in all COMP pseudo-macro-instructions. Codes for the switches may be omitted if they are not needed to store the result of the compare operation. However, if a switch is omitted, the comma that would have followed it must be included in the operand field.

The processor:

1. Compares the A-field to the B-field.

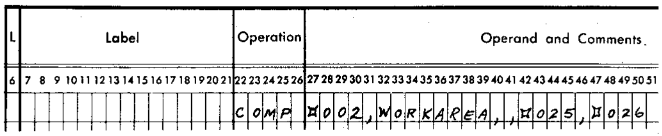
2. Sets the status switches to the result of the comparison:

a. The first switch is set on, if the value of the B-field is greater than that of the A-field.
b. The second switch is set on, if the B-field is equal to the A-field.
c. The third switch is set on, if the value of the B-field is less than that of the A-field.

Examples:

Figure 77 shows a COMP pseudo-macro which states:

1. Compare parameter 002 of the macro statement to WORKAREA.
2. If parameter 002 is equal to WORKAREA, turn on temporary switch 25.
3. If parameter 002 is less than WORKAREA, turn on temporary switch 26.

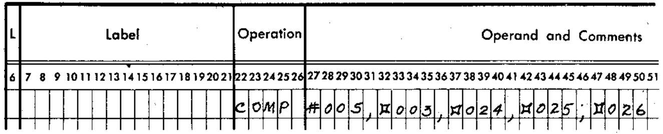


L	Label	Operation	Operand and Comments.
		COMP	⌑002,WORKAREA,,⌑025,⌑026

Figure 77. COMP Pseudo-Macro

Figure 78 shows a COMP pseudo-macro which states:

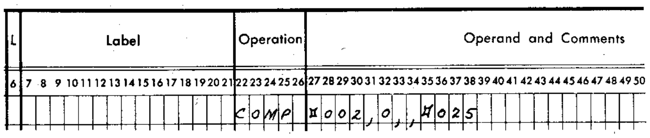
1. Compare the contents of sum box 005 to parameter 003 of the macro statement.
2. If the result is HIGH, set temporary switch 024 on.
3. If the result is EQUAL, set temporary switch 025 on.
4. If the result is LOW, set temporary switch 026 on.



L	Label	Operation	Operand and Comments
		COMP	#005,⌑003,⌑024,⌑025;⌑026

Figure 78. Comparing a Parameter to the Contents of a Sum Box

NOTE: The standard 1410 collating sequence determines HIGH, EQUAL, or LOW conditions. Comparisons are controlled by the B-field in the 1410. Thus, the statement shown in Figure 79 will cause temporary switch 025 to be set on if the low-order position of parameter 002 is an @ symbol (if parameter 002 is an alphamerical literal).



L	Label	Operation	Operand and Comments
		COMP	⌑002,0,,⌑025

Figure 79. Checking for an Alphamerical Literal

NOTE — To Produce a Message

The NOTE pseudo-macro writes messages concerning conditions that can arise during the processing of a macro-instruction.

The programmer:

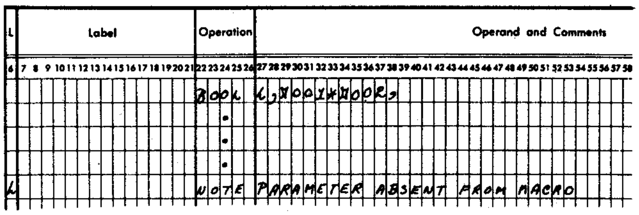
1. Writes the name of the pseudo-macro (NOTE) in the operation field.

2. Writes the message in the operand field. The page and line number of the macro statement that is being processed will precede the printed message.

The processor prints the message and its accompanying identification numbers on the console printer.

Example: Figure 80 shows how the NOTE pseudo-macro can be used in combination with the BOOL pseudo-macro. The BOOL pseudo-macro tests to insure that parameters 001 and 002 are present in the macro-instruction. If either parameter is missing, the processor skips to the NOTE pseudo-macro and prints:

1. The page and line number of the macro-instruction.
2. PARAMETER ABSENT FROM MACRO.



```
L | Label | Operation | Operand and Comments
  |       | BOOL      | L,¤001*¤002,
  |       |   .       |
  |       |   .       |
  |       |   .       |
L |       | NOTE      | PARAMETER ABSENT FROM MACRO
```

Figure 80. NOTE Pseudo-Macro

MEND — End of Routine

This pseudo-macro signals the end of generation for a macro-instruction. It may appear anywhere in a library routine.

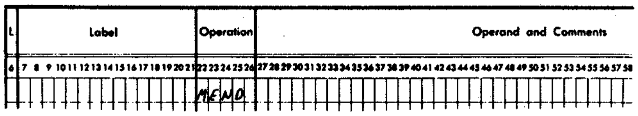
The programmer:

1. Writes the name of the pseudo-macro (MEND) in the operation field.
2. Leaves the operand field blank.

The processor stops processing the macro-instruction when it encounters a MEND statement.

NOTE: A BOOL pseudo-macro can be used to skip over a MEND pseudo-macro which appears within the library routine if conditions indicate that more model statements must be processed.

Example: Figure 81 shows a MEND pseudo-macro.



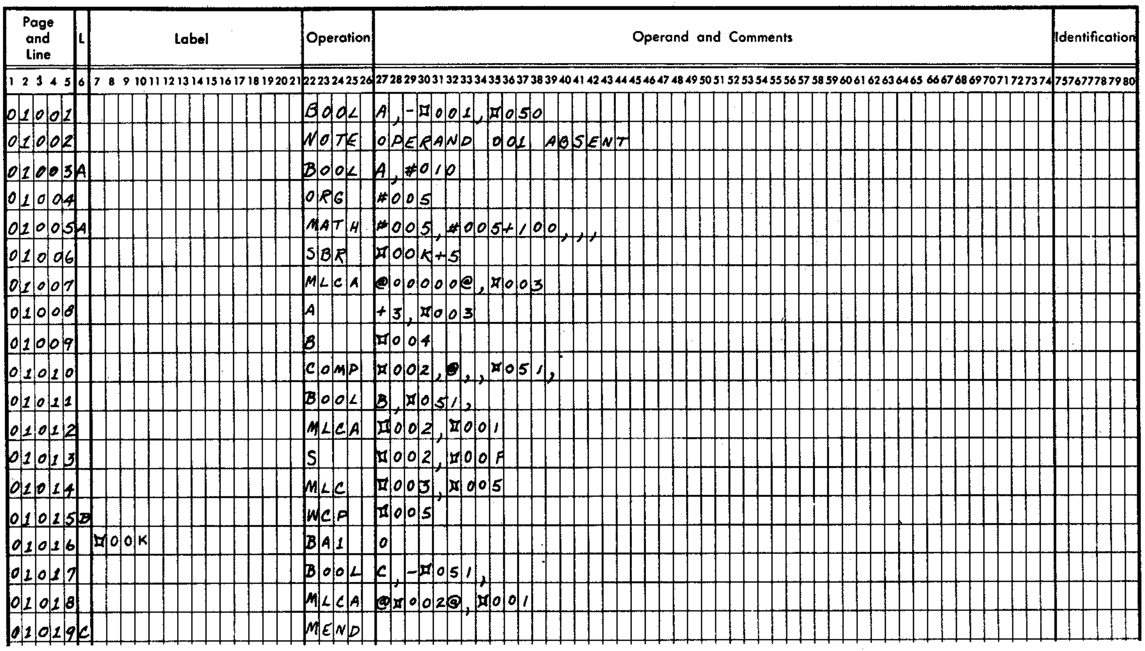
```
L | Label | Operation | Operand and Comments
  |       | MEND      |
```

Figure 81. MEND Pseudo-Macro

Pseudo-Macro Coding Example

Figure 82 shows the library entry for a hypothetical macro called PRLIT. This library routine uses all of the five pseudo-macros. It illustrates the effect of the pseudo-macros on the processing of a macro-instruction. The meaning of each line in the library routine is:

Entry 1: If parameter 001 is present, set temporary switch 050 off and go to entry 3. If parameter 001 is missing, go to entry 2.



```
Page and Line | L | Label | Operation | Operand and Comments       | Identification
01001         |   |       | BOOL      | A,-¤001,¤050
01002         |   |       | NOTE      | OPERAND 001 ABSENT
01003         | A |       | BOOL      | A,#010
01004         |   |       | ORG       | #005
01005         | A |       | MATH      | #005,#005+100,,,
01006         |   |       | SBR       | ¤00K+5
01007         |   |       | MLCA      | @00000@,¤003
01008         |   |       | A         | +3,¤003
01009         |   |       | B         | ¤004
01010         |   |       | COMP      | ¤002,@,,¤051,
01011         |   |       | BOOL      | B,¤051,
01012         |   |       | MLCA      | ¤002,¤001
01013         |   |       | S         | ¤002,¤00F
01014         |   |       | MLC       | ¤003,¤005
01015         | B |       | WCP       | ¤005
01016         |   | ¤00K  | BA1       | 0
01017         |   |       | BOOL      | C,-¤051,
01018         |   |       | MLCA      | @¤002@,¤001
01019         | C |       | MEND      |
```

Figure 82. PRLIT Library Routine

Entry 2: Print the note: OPERAND 001 ABSENT.

Entry 3: If permanent switch 010 is off, go to entry 5. If permanent switch 10 is on, go to entry 4.

Entry 4: ORG at the contents of sum box #005.

Entry 5: Put the contents of sum box #005 plus 100 in sum box #005.

Entry 6: Store the contents of the B-address register in an address equal to the address assigned to the internal label (□ 00K) + 5.

Entry 7: Move five zeros to the field whose symbolic address is parameter 003 of the macro-instruction.

Entry 8: Add the literal + 3 to the field specified by the parameter 003.

Entry 9: Branch to parameter 004.

Entry 10: If parameter 002 is a literal, the EQUAL switch (□ 051) is set on.

Entry 11: If the EQUAL switch (temporary switch 051) is off, skip to entry 15. If the EQUAL switch is on, go to entry 12.

Entry 12: Move parameter 002 to parameter 001.

Entry 13: Subtract parameter 002 from parameter 006. (If parameter 006 is missing, this statement will be bypassed.)

Entry 14: Move parameter 003 to parameter 005.

Entry 15: On the typewriter print the field whose address is specified by parameter 005.

Entry 16: Branch to 0 if any of the I/O channel status indicators is on.

Entry 17: If temporary switch 051 is on, skip to entry 19. If temporary switch 051 is off, go to entry 18.

Entry 18: Insert parameter 002 as a literal, and move it to the field represented by parameter 001.

Entry 19: End-of-library routine. Assume that:

1. The macro shown in Figure 83 is encountered in the source program.
2. Permanent switch 010 is on.
3. Sum box #005 contains $1234\overset{+}{5}$.

Call Routines

The 1410 Autocoder processor permits the user to add inflexible routines to the library tape. These are commonly used sequences of instructions that can be extracted for an object program by the CALL macro. They differ from the routines processed by other macro-instructions in several ways:

1. All instructions must be complete; no parameters can be inserted.

2. All instructions in the routine are incorporated.

3. A CALL routine is not inserted at the point where the CALL macro was encountered in the source program. Instead, it is inserted only once as a closed routine elsewhere in the object program or program section. Linkage to the routine is provided automatically by the processor whenever its particular CALL macro is encountered in the source program. (The processor does not produce automatic linkage to the routines incorporated by other macro-instructions because these routines are inserted as open routines where the associated macro-instructions were encountered in the source program.)

4. Data needed by a CALL routine must be in the locations indicated by the symbols in the operand fields of its instructions.

Requirements: CALL routines have several specific requirements that must be considered when the routine is created:

1. Every entry point in a CALL routine must have a label. These labels (and all other symbols used in a CALL routine) must be at least five characters in length, and each of these labels must have the same first five characters.

Macro Instruction

Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50 55 60 65 70
	PRLIT	AREAA,@42AB@,FIELD1,EXIT1,WORKAREA

Assembled Symbolic Program Entry

```
        ORG  12345
        SBR  □00K023+5
        MLCA @00000@,FIELD1
        A    +3,FIELD1
        B    EXIT1
        MLCA @42AB@,AREAA
        MLC  FIELD1,WORKAREA
        WCP  WORKAREA
□00K023 BA1  0
```

Figure 83. Using the PRLIT Routine

CALL routines are stored as controlled by Literal Origin at the time and place where an END or EXECUTE processor control statement is encountered. Duplicate symbols can occur if a CALL routine is used in more than one program overlay (if the same CALL routine is named in CALL macros that are separated by a Literal Origin or Execute statement). To eliminate this possibility the Autocoder processor provides a suffix (see "SFX") operation. The programmer should use a suffix statement containing a new character in each program section.

2. The first instruction at each entry point in a CALL routine must store the contents of the B-address register SBR in an index location or in the last instruction executed in the CALL routine. This provides for re-entry at the proper place in the main routine after the CALL routine is executed.

3. All macro-instruction operation codes except CALL and INCLD are invalid in CALL routines. All other symbolic entries acceptable to Autocoder, except Literal Origin, Execute, and End, can be used. A CALL macro can be used:

a. to allow one CALL routine to be used at some point in another CALL routine, or

b. as a model statement in the library routine for a regular macro-instruction.

Call Macro

The CALL macro provides access to inflexible routines written by the user and stored in the library tape. It establishes linkage to a closed routine and stores that routine elsewhere in the program. The CALL macro is part of the Autocoder processor.

The programmer:

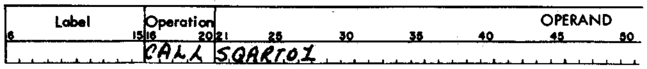
1. Writes the name of the macro (CALL) in the operation field.

2. Writes in the operand field the label of the library statement which is the desired entry point in the library routine. The first five characters of this label must be the same as the five characters in the label field of the INSER statement that was used to enter the routine in the library tape (see "INSER").

a. If the CALL routine is constructed so that all the data it requires must be taken from specifically-labeled areas of storage, the remainder of the operand field must be left blank. For example, a CALL routine whose entry point is SQART01 requires that the number whose square root is to be computed must be placed in a location labeled SQART02. The CALL macro is written as shown in Figure 84.

b. If the CALL routine is constructed so that the data it requires can be located in arbitrarily labeled areas of core storage, the symbols for these areas must be included immediately following the label in the operand field. These symbols must be entered in the order in which they are required by the CALL routine. This makes it possible to design CALL routines in which the required data can be placed in locations labeled in any way the programmer desires. This frees the source program writer from the restriction that he insert data in locations labeled according to the :equirements of the CALL routine. CALL routines to be used in this manner must be coded to utilize the address constants that will be created from the symbols in the operand field.

Call Macro


Label
Operation
OPERAND
6
15
16
20
21
25
30
35
40
45
50

CALL SQART01

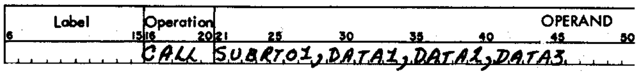
Assembled Symbolic Program Entry

```
B     SQART01
```

Figure 84. CALL Statement Specifying That Data Is in Specifically Labeled Areas of Storage

Example: Call a routine whose entry point is SUBRT01 (Figure 85). The addresses of DATA 1, DATA 2, and DATA 3 are needed by the CALL routine.

Call Macro


Label
Operation
OPERAND
6
15
16
20
21
25
30
35
40
45
50

CALL SUBRT01,DATA1,DATA2,DATA3

Assembled Symbolic Program Entry

```
B     SUBRT01
DCW   DATA1
      DATA2
      DATA3
```

Figure 85. CALL Statement for a Routine with Arbitrary Data-Storage Assignments

The processor:

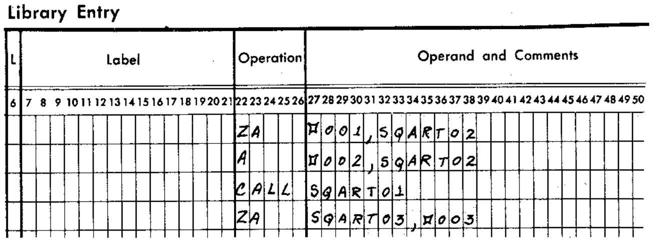
1. Establishes linkage from the main routine to the CALL routine by assembling a symbolic program entry for an unconditional branch instruction. The operand for this branch instruction is the entry point given in the operand field of the CALL macro as shown in Figures 84 and 85. The branch instruction follows the CALL macro.

2. Creates address constants for other symbols appearing in the operand field of the CALL macro, and inserts them following the unconditional branch instruction as shown in Figure 85. NOTE: These address constants are defined in the order in which the associated symbols appear in the CALL operand.

Result: A given CALL routine is inserted once per program or program section in a location determined by a processor control statement. Branch instructions are inserted as many times as an associated CALL macro is encountered in the source program. Thus, the CALL routine can be entered from several points in the main routine.

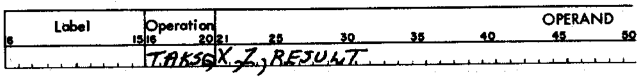
Example: Assume that a library routine to compute the value of X + Z is associated with a regular macro-instruction called TAKSQ. There is also a CALL routine in the library tape named SQART01 which calculates the square root of a number in a work area (SQART02) and places the answer in another work area (SQART03). The programmer can design a library entry for the TAKSQ macro that will provide linkage to the CALL routine as shown in Figure 86.

Library Entry



L	Label	Operation	Operand and Comments
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
		ZA	⌑001,SQART02
		A	⌑002,SQART02
		CALL	SQART01
		ZA	SQART03,⌑003

Macro Instruction



Label	Operation	OPERAND
	TAKSQ	X,Z,RESULT

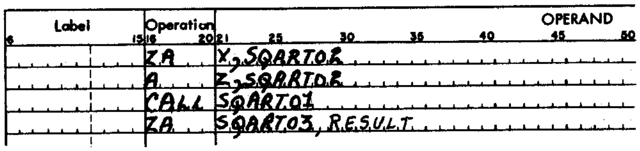
Assembled Symbolic Program Entry

```
ZA     X, SQART02
A      Z, SQART02
B      SQART01
ZA     SQART03, RESULT
```

Figure 86. CALL Statement within a Library Routine for a Macro-Instruction

When the object routine is executed, X + Z will be stored in SQART02. Then the program will branch to the CALL routine where the square root of X + Z will be calculated and the result stored in SQART03. The last instruction in the SQART01 routine will cause an unconditional branch to the last instruction in the TAKSQ routine which puts the answer in an area labeled RESULT. NOTE: This illustration shows the combination of a regular macro and the CALL macro. The same result could be achieved by writing entries in the source program as shown in Figure 87.

Source Program Entries



Label	Operation	OPERAND
	ZA	X,SQART02
	A	Z,SQART02
	CALL	SQART01
	ZA	SQART03,RESULT

Assembled Symbolic Program Entry

```
ZA     X, SQART02
A      Z, SQART02
B      SQART01
ZA     SQART03,RESULT
```

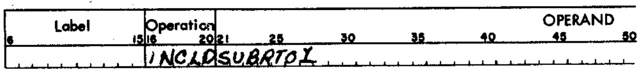
Figure 87. Alternative Source Program Entries

Incld Macro

This macro is used to extract an inflexible library routine from the library tape. However, the INCLD macro does not insert a branch instruction following the INCLD statement in the source program as does the CALL statement. The programmer establishes his own linkage to the closed routine. INCLD statements are constructed in the same manner as CALL statements.

Example: Figure 88 shows an INCLD statement that causes a library routine named SUBRT01 to be incorporated in the object program.

The processor does not produce a branch instruction. The programmer must insert a branch at the place in the main routine at which the exit to the closed routine is needed. Several INCLD statements can be written in a group in a source program to cause the associated library routines to be stored at LTORG, END, or EX time by the processor. Thus, one exit from the main routine can be used to cause several library routines to be executed at object time.

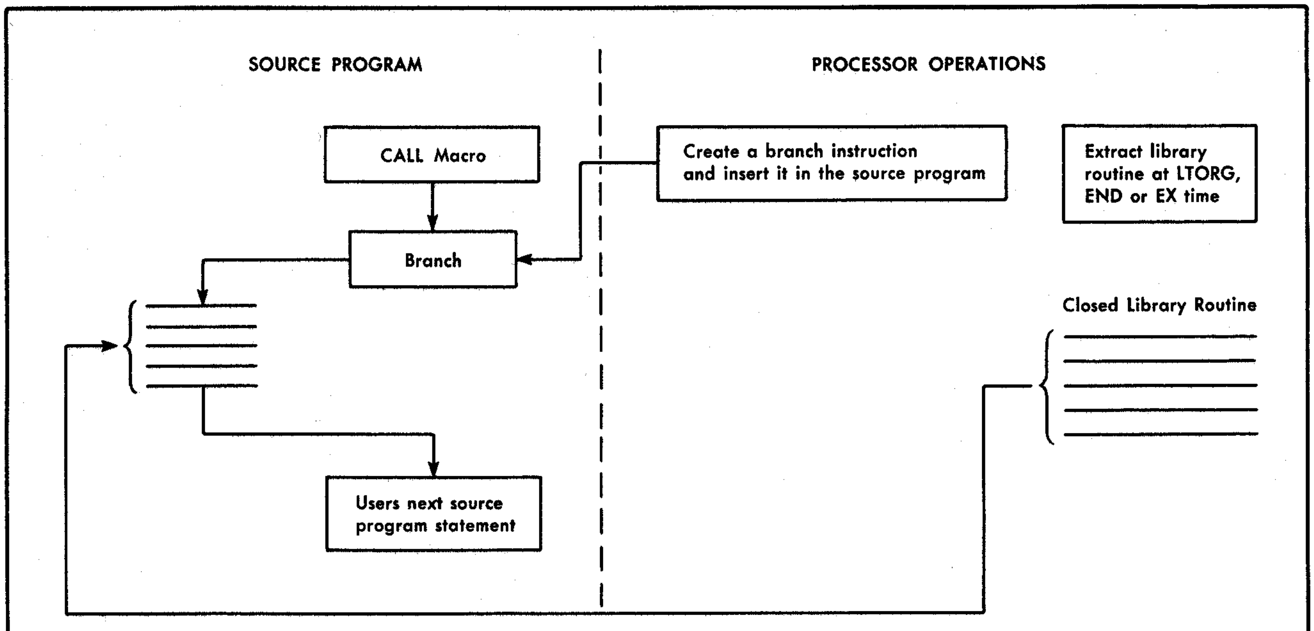


Label	Operation	OPERAND
	INCLD	SUBRT01

Figure 88. INCLD Statement

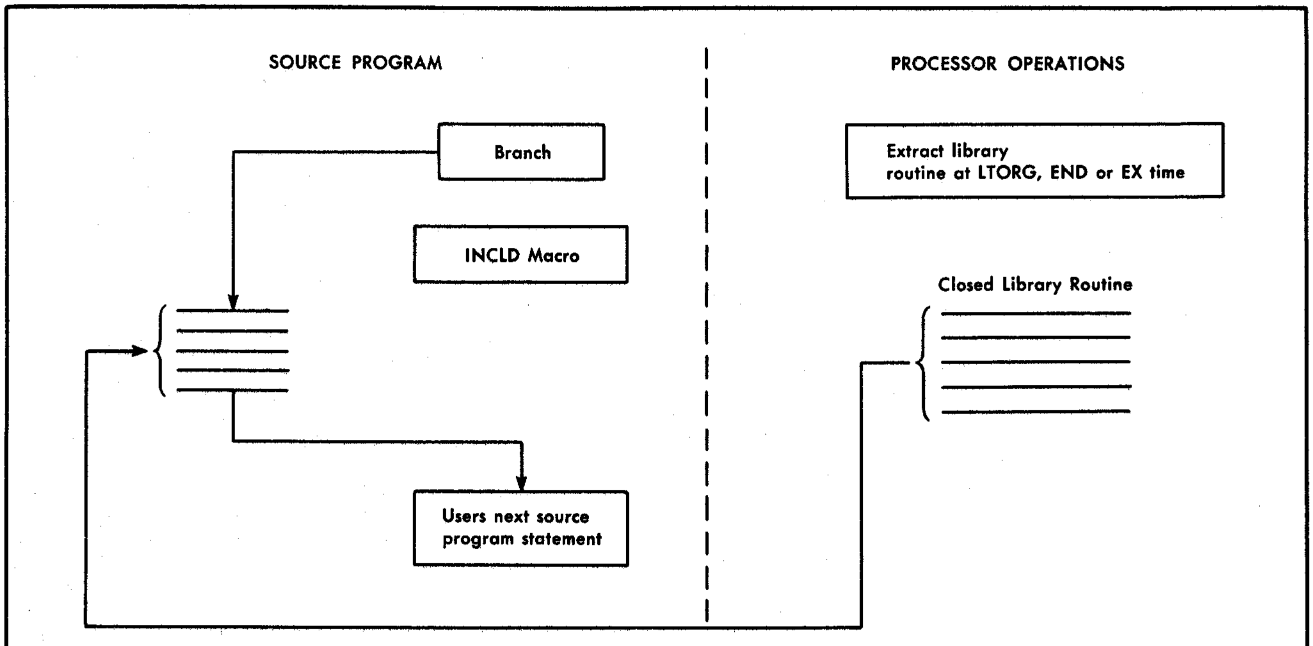
NOTE: CALL and INCLD statements may appear in either flexible or inflexible library routines. Also, an inflexible library routine may, in turn, have CALL or INCLD statements.

If CALL or INCLD are written *within* a library routine, only a single operand is permitted in the CALL or INCLD statement. This single operand is the name or entry point of the closed library routine. (See "Call Macro.")


SOURCE PROGRAM
PROCESSOR OPERATIONS
CALL Macro
Create a branch instruction and insert it in the source program
Extract library routine at LTORG, END or EX time
Branch
Closed Library Routine
Users next source program statement

When the processor encounters a CALL macro, it creates an unconditional branch instruction to link the main program to the library routine. The branch instruction is placed in the symbolic program immediately following the CALL macro statement. Later, when the processor encounters a LTORG, END or EX statement in the source program, it extracts all library routines specified by CALL macros and stores them as closed routines.

Figure 89. CALL Processing


SOURCE PROGRAM
PROCESSOR OPERATIONS
Branch
Extract library routine at LTORG, END or EX time
INCLD Macro
Closed Library Routine
Users next source program statement

When the processor encounters an INCLD macro, it incorporates the specified library routines when an LT ORG, END, or EX statement is encountered in the user's source program. Note that the branch instruction that links the main routine to the closed library routine is provided by the programmer.

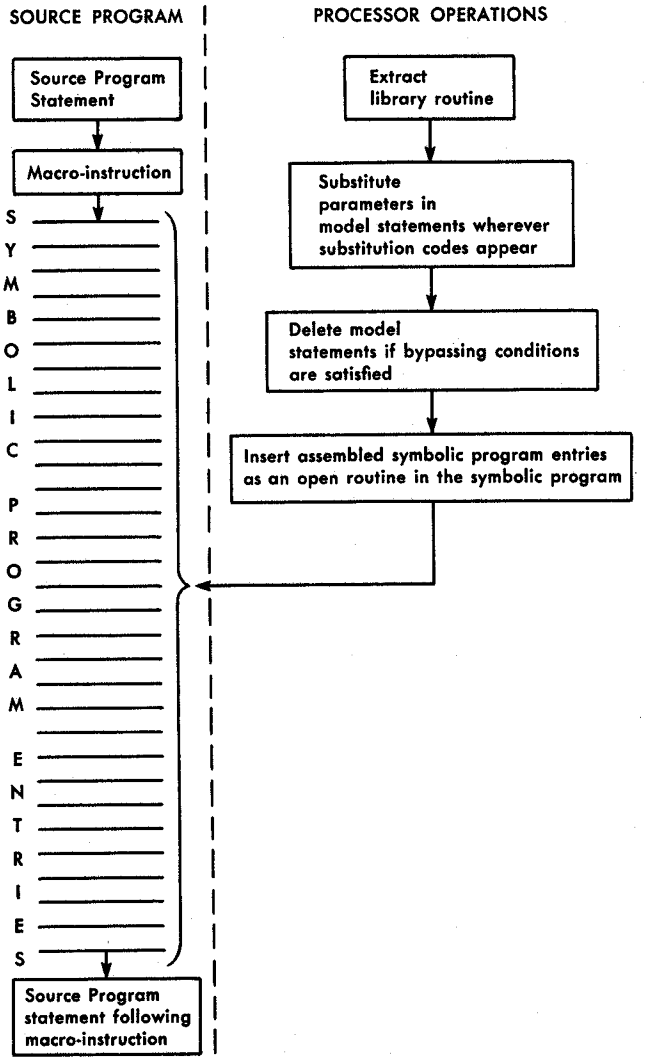
Figure 90. INCLD Processing

Macro Processing

Figures 89, 90, and 91 show the effect of the three different uses of library routines:

1. As extracted by a regular macro-instruction.
2. As extracted by the CALL macro.
3. As extracted by the INCLD macro.

The symbolic programs that result from the processor actions described in Figures 89, 90, and 91 are later processed as though the user had, himself, inserted all the entries in the source program. Symbolic entries are translated to machine-language instructions, constants cards are produced, etc.


SOURCE PROGRAM
PROCESSOR OPERATIONS
Source Program Statement
Macro-instruction
Extract library routine
Substitute parameters in model statements wherever substitution codes appear
Delete model statements if bypassing conditions are satisfied
Insert assembled symbolic program entries as an open routine in the symbolic program
SYMBOLIC PROGRAM ENTRIES
Source Program statement following macro-instruction

When a regular macro-instruction is encountered in the source program, the processor extracts the specified library routine, tailors it, and inserts it in-line in the users source program.

Figure 91. Macro Processing

DELET — Delete

This entry deletes a library routine or parts of a library routine from the library tape.

The programmer:

1. Writes the mnemonic operation code (DELET) in the operation field.

2. Writes the name of the library routine in the label field.

3. Writes in the operand field the line number(s) of the model statement(s) to be deleted. If a whole routine is to be deleted, the operand field contains only an M or S. If more than one model statement of a continuous sequence are to be deleted, the first and last numbers must be written separated by commas.

The processor deletes the model statement or statements specified in the operand field.

Result: The new library tape contains the modified library routine.

Examples: Figure 92 is a DELET statement that will cause the whole CHECK library routine to be removed from the library.

Figure 93 is a DELET statement that will cause the first model statement to be deleted from the CHECK library routine.

Figure 94 is a DELET statement that will cause model statements 2, 3, 4, and 5 to be deleted.

Label	Operation	OPERAND
CHECK	DELET	M

Figure 92. Deleting an Entire Library Routine

Label	Operation	OPERAND
CHECK	DELET	M,1

Figure 93. Deleting a Single Model Statement

Label	Operation	OPERAND
CHECK	DELET	M,2,5

Figure 94. Deleting Multiple Model Statements

INSER — Insert

This entry can be used to insert whole library routines or part of a library routine in the library tape.

The programmer:

1. Writes the mnemonic operation code (INSER) in the operation field.

2. Writes the name of the library routine in the LABEL field.

3. Writes the line number of the model statement after which the insertion is to be made. If two operands separated by a comma are written, the implied deletion will take place.

The processor deletes model statements, if necessary, and inserts the model statement(s) in the library routine.

Result: The new library tape contains the modified library routine.

Examples: Figure 95 is an INSER statement that will cause a library routine named CHECK to be inserted in the library tape.

Figure 96 is an INSER statement that will cause new model statement 1 to be inserted in the CHECK library routine.

Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
CHECK	INSER	M

Figure 95. Inserting an Entire Library Routine

Autocoder Statement

Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
CHECK	INSER	M,0

Model Statement

L	Label	Operation	Operand and Comments
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
		C	⌑001,⌑002

Figure 96. Inserting a Single Model Statement

Figure 97 is an INSER statement that will cause the first model statement that is presently in the library routine to be deleted and the model statement shown to be inserted in its place.

Figure 98 is an INSER statement that causes model statements 1 and 2 to be deleted and the model statements shown to be inserted in their places.

Autocoder Statement

Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
CHECK	INSER	M,1,1

Model Statement

L	Label	Operation	Operand and Comments
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
	⌑00A	B	PAR4

Figure 97. Substituting One Model Statement for Another

Autocoder Statement

Label	Operation	OPERAND
6 15	16 20	21 25 30 35 40 45 50
CHECK	INSER	M,1,2

Model Statement

L	Label	Operation	Operand and Comments
6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
	⌑00A	B	PAR4
		C	PAR1,PAR2
		BE	PAR3

Figure 98. Substituting Multiple Model Statements

Mnemonic Operation Codes

This section consists of a list of the mnemonic operation codes for Declarative, Processor Control, and Imperative operations.

The operation code, operand(s), operation code definition, and an example of an assembled machine-language instruction are shown. A, B, and I (instruction) addresses have been equated to 12345, 34567, and 56789, respectively, for illustrative purposes. Where d-modifier characters must be provided by the programmer, a D appears in the operand column and in the d-modifier position of the assembled machine-language instruction.

Declarative Operation Codes

Note that no assembled machine-language instruction is produced. These codes are directions to the processor program only.

Processor Control Operation Codes

Note that for explanation of the four operation codes, JOB, EJECT, RESEQ, and RUN, the reader is referred to the "Processor Control Operations" section of this publication.

Imperative Operations

The imperative operation codes are listed in the following order: Arithmetic, Data-Move, Compare and Look-up, Logical, Miscellaneous, and Input/Output commands.

PROCESSOR CONTROL OPERATIONS

OPCODE	OPERAND		INSTRUCTION
LOAD		PRECEDE OBJECT PROGRAM WITH LOADER	
CTL		OBJECT MACHINE SIZE AND PRINT OR PCH SUPRES	
PST		PUNCH SYMBOL TABLE	
EX	LABEL	EXECUTE	J .....
LTORG	*	LITERAL ORIGIN	
XFR	LABEL	TRANSFER	J .....
SFX	B	SUFFIX CHARACTER	
ORG	10000	ORIGIN	
END	LABEL	TERMINATES ASSEMBLY - GENERATES BRANCH TO ADDRESS OF LABEL	J .....
JOB		(SEE PROCESSOR CONTROL OPERATIONS SECTION)	
EJECT		(SEE PROCESSOR CONTROL OPERATIONS SECTION)	
RESEQ		(SEE PROCESSOR CONTROL OPERATIONS SECTION)	
RUN		(SEE PROCESSOR CONTROL OPERATIONS SECTION)	

DECLARATIVE OPERATIONS

	OPCODE	OPERAND		INSTRUCTION
A	EQU	12345	THE EQUATE INSTRUCTION	
B	EQU	34567		
I	EQU	56789		
	DA	1X2,G	DEFINE AREA	
	DCW	@ @	DEFINE CONSTANT WITH WORD MARK	
	DC	&2	DEFINE CONSTANT	
	DS	1	DEFINE SYMBOL	
	EQU	Z	EQUATE	

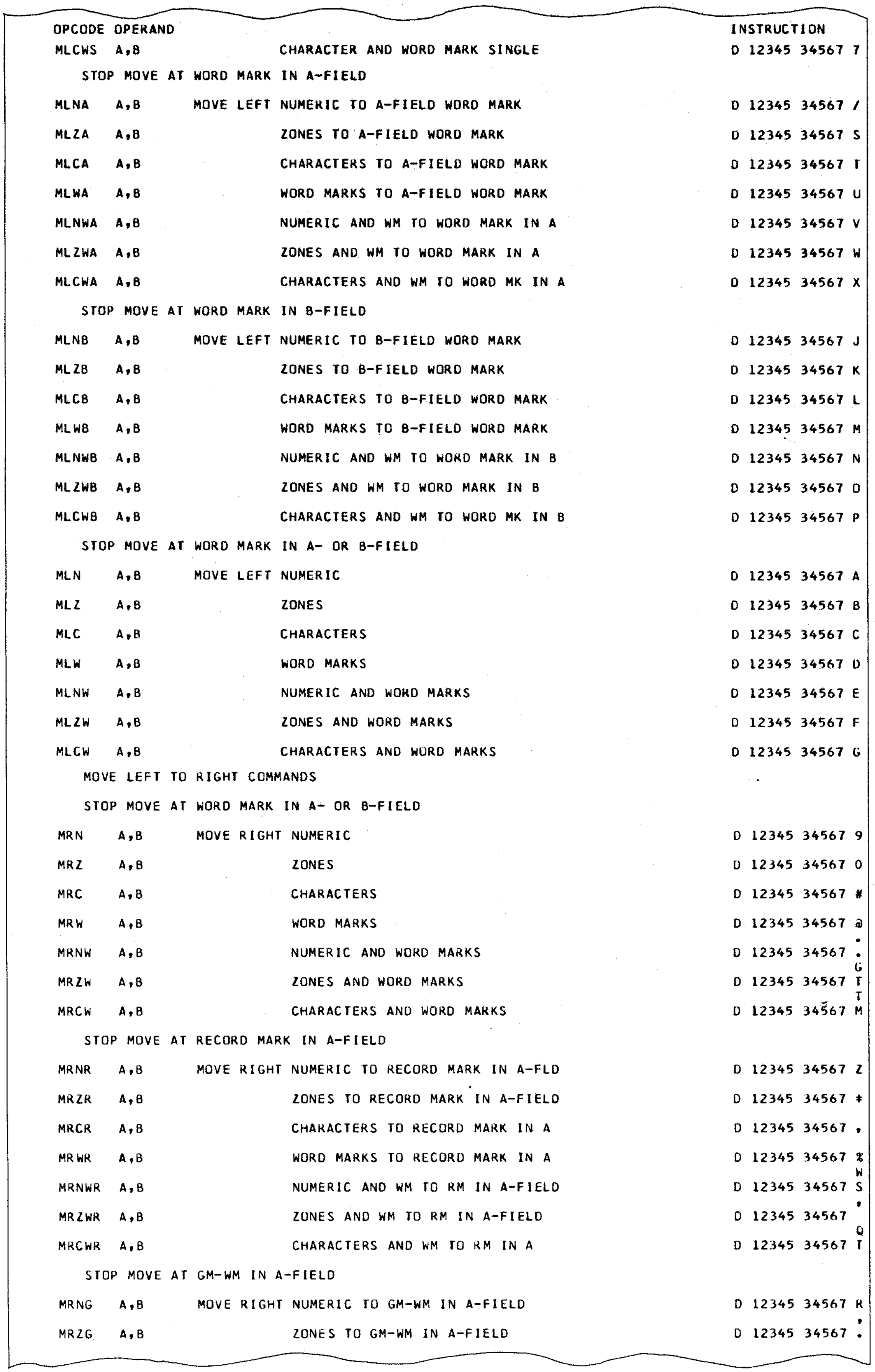
IMPERATIVE OPERATIONS

ARITHMETIC OPERATIONS

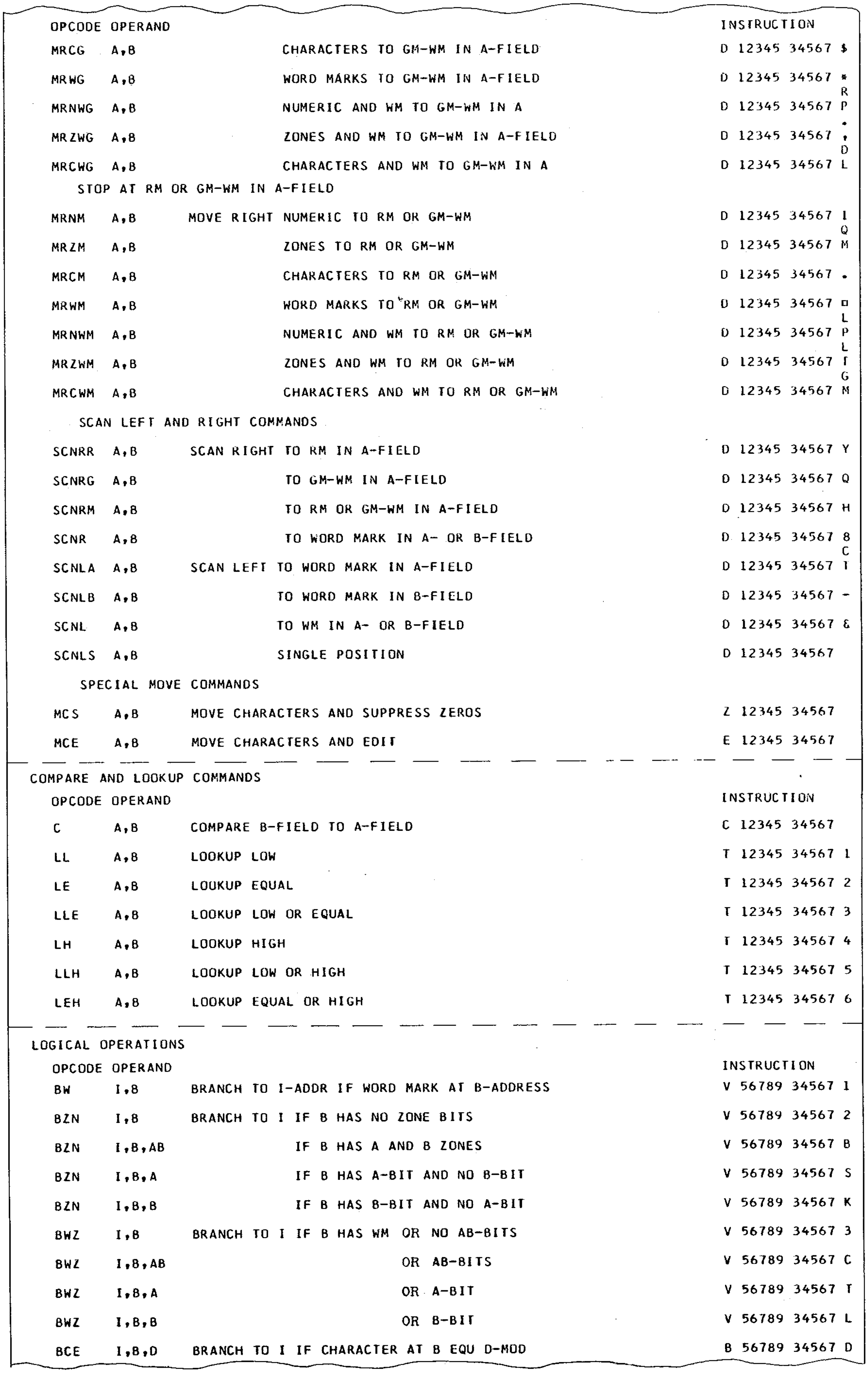
OPCODE	OPERAND		INSTRUCTION
A	A,B	ADD A-FIELD TO B-FIELD	A 12345 34567
S	A,B	SUBTRACT A FROM B	S 12345 34567
ZA	A,B	ZERO AND ADD A TO B	Q M 12345 34567
ZS	A,B	ZERO AND SUBTRACT A FROM B	' . 12345 34567
M	A,B	MULTIPLY	@ 12345 34567
D	A,B	DIVIDE	% 12345 34567

DATA MOVE

OPCODE	OPERAND		INSTRUCTION
MOVE RIGHT TO LEFT COMMANDS			
MOVE SINGLE POSITION			
MLNS	A,B	MOVE LEFT NUMERIC SINGLE	D 12345 34567 1
MLZS	A,B	ZONES SINGLE	D 12345 34567 2
MLCS	A,B	CHARACTERS SINGLE	D 12345 34567 3
MLWS	A,B	WORD MARKS SINGLE	D 12345 34567 4
MLNWS	A,B	NUMERIC AND WORD MARK SINGLE	D 12345 34567 5
MLZWS	A,B	ZONE AND WORD MARK SINGLE	D 12345 34567 6

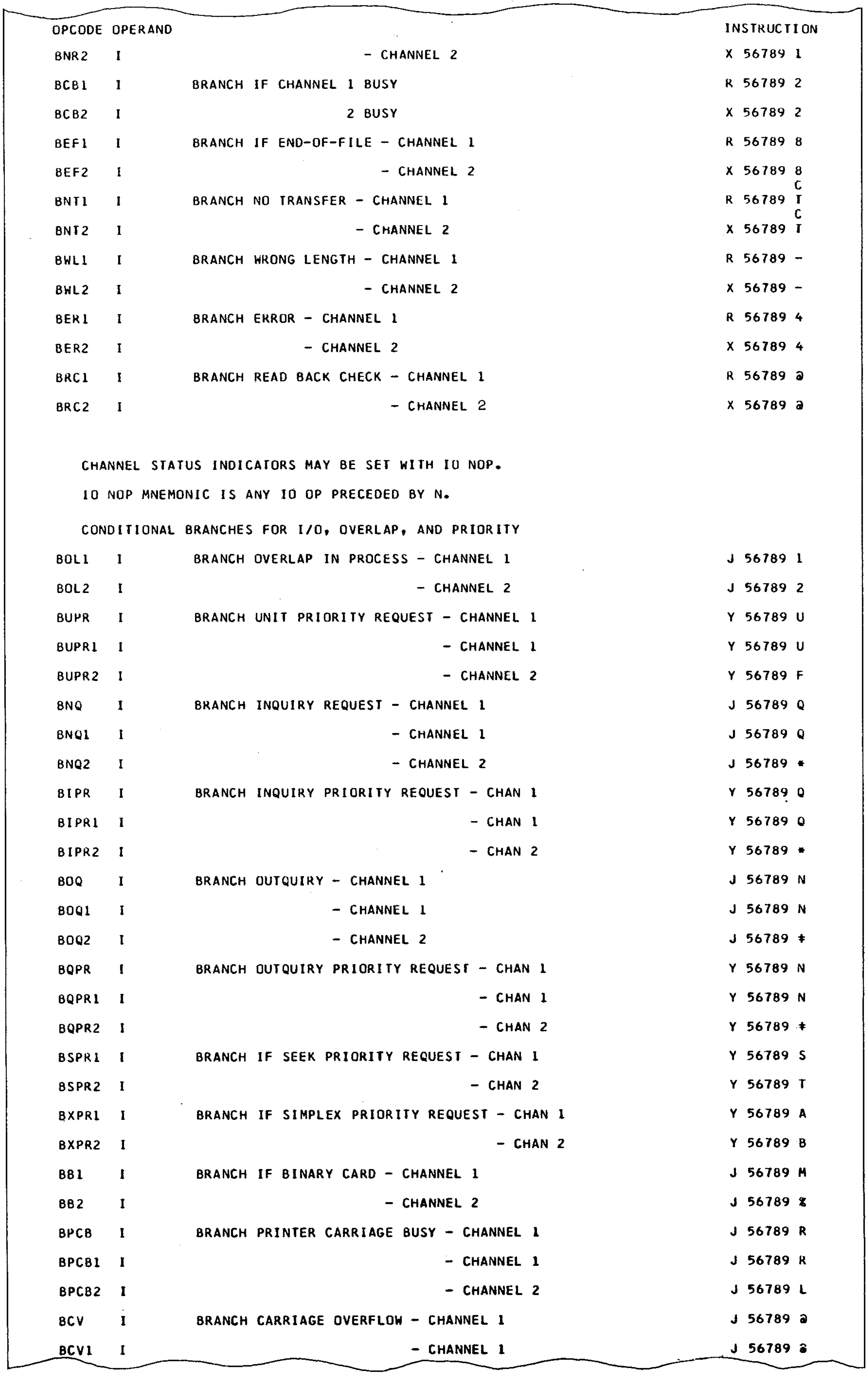


```
OPCODE OPERAND                                                                INSTRUCTION
MLCWS  A,B                CHARACTER AND WORD MARK SINGLE                      D 12345 34567 7
   STOP MOVE AT WORD MARK IN A-FIELD
MLNA   A,B      MOVE LEFT NUMERIC TO A-FIELD WORD MARK                        D 12345 34567 /
MLZA   A,B                ZONES TO A-FIELD WORD MARK                          D 12345 34567 S
MLCA   A,B                CHARACTERS TO A-FIELD WORD MARK                     D 12345 34567 T
MLWA   A,B                WORD MARKS TO A-FIELD WORD MARK                     D 12345 34567 U
MLNWA  A,B                NUMERIC AND WM TO WORD MARK IN A                    D 12345 34567 V
MLZWA  A,B                ZONES AND WM TO WORD MARK IN A                      D 12345 34567 W
MLCWA  A,B                CHARACTERS AND WM TO WORD MK IN A                   D 12345 34567 X
   STOP MOVE AT WORD MARK IN B-FIELD
MLNB   A,B      MOVE LEFT NUMERIC TO B-FIELD WORD MARK                        D 12345 34567 J
MLZB   A,B                ZONES TO B-FIELD WORD MARK                          D 12345 34567 K
MLCB   A,B                CHARACTERS TO B-FIELD WORD MARK                     D 12345 34567 L
MLWB   A,B                WORD MARKS TO B-FIELD WORD MARK                     D 12345 34567 M
MLNWB  A,B                NUMERIC AND WM TO WORD MARK IN B                    D 12345 34567 N
MLZWB  A,B                ZONES AND WM TO WORD MARK IN B                      D 12345 34567 O
MLCWB  A,B                CHARACTERS AND WM TO WORD MK IN B                   D 12345 34567 P
   STOP MOVE AT WORD MARK IN A- OR B-FIELD
MLN    A,B      MOVE LEFT NUMERIC                                             D 12345 34567 A
MLZ    A,B                ZONES                                               D 12345 34567 B
MLC    A,B                CHARACTERS                                          D 12345 34567 C
MLW    A,B                WORD MARKS                                          D 12345 34567 D
MLNW   A,B                NUMERIC AND WORD MARKS                              D 12345 34567 E
MLZW   A,B                ZONES AND WORD MARKS                                D 12345 34567 F
MLCW   A,B                CHARACTERS AND WORD MARKS                           D 12345 34567 G
   MOVE LEFT TO RIGHT COMMANDS
   STOP MOVE AT WORD MARK IN A- OR B-FIELD
MRN    A,B      MOVE RIGHT NUMERIC                                            D 12345 34567 9
MRZ    A,B                 ZONES                                              D 12345 34567 0
MRC    A,B                 CHARACTERS                                         D 12345 34567 #
MRW    A,B                 WORD MARKS                                         D 12345 34567 @
                                                                                            .
MRNW   A,B                 NUMERIC AND WORD MARKS                             D 12345 34567 .
                                                                                            G
MRZW   A,B                 ZONES AND WORD MARKS                               D 12345 34567 T
                                                                                            T
MRCW   A,B                 CHARACTERS AND WORD MARKS                          D 12345 34567 M
   STOP MOVE AT RECORD MARK IN A-FIELD
MRNR   A,B      MOVE RIGHT NUMERIC TO RECORD MARK IN A-FLD                    D 12345 34567 Z
MRZR   A,B                 ZONES TO RECORD MARK IN A-FIELD                    D 12345 34567 ‡
MRCR   A,B                 CHARACTERS TO RECORD MARK IN A                     D 12345 34567 ,
MRWR   A,B                 WORD MARKS TO RECORD MARK IN A                     D 12345 34567 %
                                                                                            W
MRNWR  A,B                 NUMERIC AND WM TO RM IN A-FIELD                    D 12345 34567 S
                                                                                            ,
MRZWR  A,B                 ZONES AND WM TO RM IN A-FIELD                      D 12345 34567
                                                                                            Q
MRCWR  A,B                 CHARACTERS AND WM TO RM IN A                       D 12345 34567 T
   STOP MOVE AT GM-WM IN A-FIELD
MRNG   A,B      MOVE RIGHT NUMERIC TO GM-WM IN A-FIELD                        D 12345 34567 R
                                                                                            ,
MRZG   A,B                 ZONES TO GM-WM IN A-FIELD                          D 12345 34567 .
```

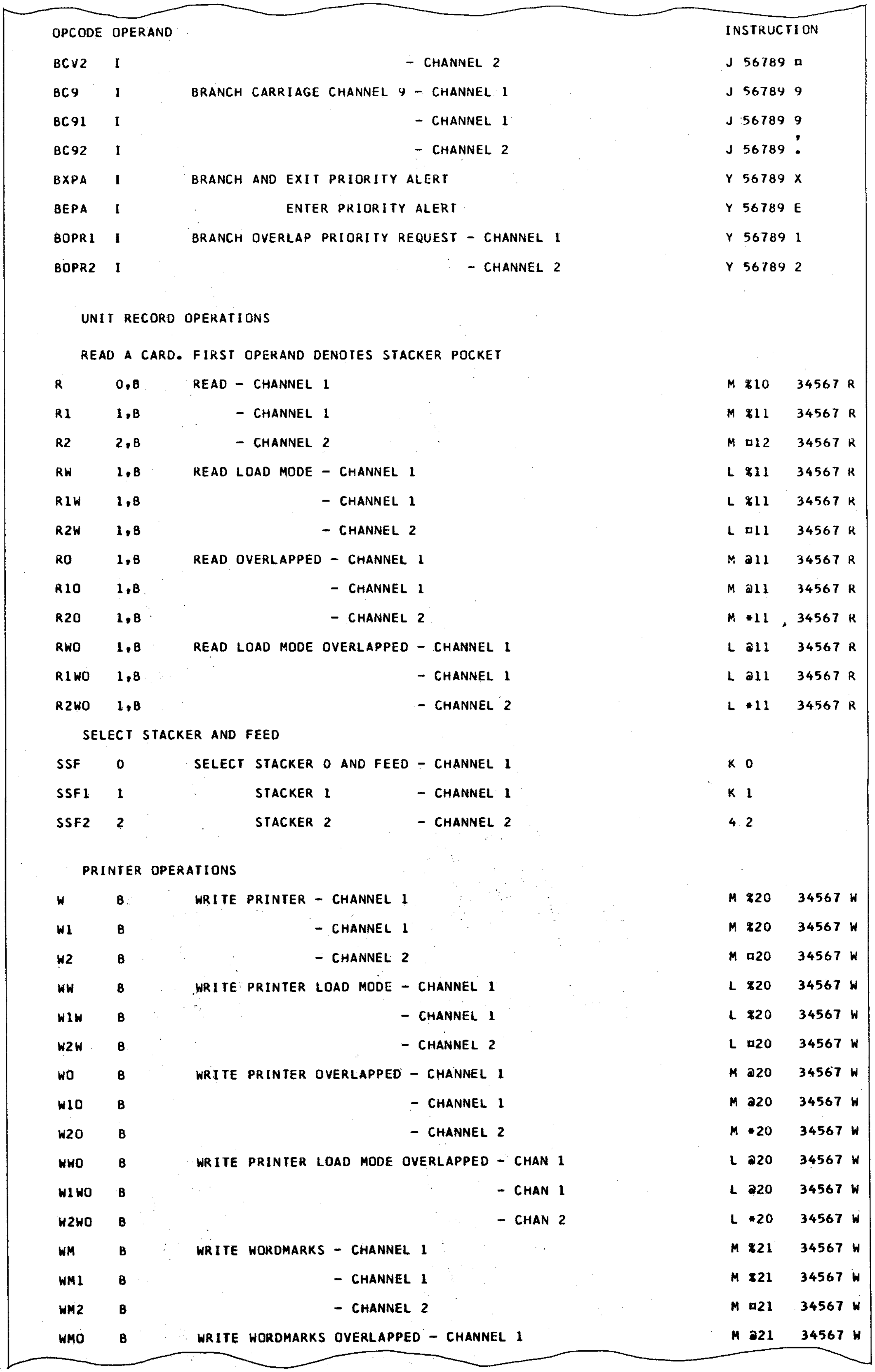


```
    OPCODE OPERAND                                                  INSTRUCTION
    MRCG    A,B              CHARACTERS TO GM-WM IN A-FIELD         D 12345 34567 $
    MRWG    A,B              WORD MARKS TO GM-WM IN A-FIELD         D 12345 34567 *
                                                                                  R
    MRNWG   A,B              NUMERIC AND WM TO GM-WM IN A           D 12345 34567 P
                                                                                  .
    MRZWG   A,B              ZONES AND WM TO GM-WM IN A-FIELD       D 12345 34567 ,
                                                                                  D
    MRCWG   A,B              CHARACTERS AND WM TO GM-WM IN A        D 12345 34567 L
       STOP AT RM OR GM-WM IN A-FIELD
    MRNM    A,B       MOVE RIGHT NUMERIC TO RM OR GM-WM             D 12345 34567 I
                                                                                  Q
    MRZM    A,B              ZONES TO RM OR GM-WM                   D 12345 34567 M
    MRCM    A,B              CHARACTERS TO RM OR GM-WM              D 12345 34567 .
    MRWM    A,B              WORD MARKS TO RM OR GM-WM              D 12345 34567 □
                                                                                  L
    MRNWM   A,B              NUMERIC AND WM TO RM OR GM-WM          D 12345 34567 P
                                                                                  L
    MRZWM   A,B              ZONES AND WM TO RM OR GM-WM            D 12345 34567 T
                                                                                  G
    MRCWM   A,B              CHARACTERS AND WM TO RM OR GM-WM       D 12345 34567 M
       SCAN LEFT AND RIGHT COMMANDS
    SCNRR   A,B       SCAN RIGHT TO RM IN A-FIELD                   D 12345 34567 Y
    SCNRG   A,B              TO GM-WM IN A-FIELD                    D 12345 34567 Q
    SCNRM   A,B              TO RM OR GM-WM IN A-FIELD              D 12345 34567 H
    SCNR    A,B              TO WORD MARK IN A- OR B-FIELD          D 12345 34567 8
                                                                                  C
    SCNLA   A,B       SCAN LEFT TO WORD MARK IN A-FIELD             D 12345 34567 T
    SCNLB   A,B             TO WORD MARK IN B-FIELD                 D 12345 34567 -
    SCNL    A,B             TO WM IN A- OR B-FIELD                  D 12345 34567 &
    SCNLS   A,B             SINGLE POSITION                         D 12345 34567
       SPECIAL MOVE COMMANDS
    MCS     A,B       MOVE CHARACTERS AND SUPPRESS ZEROS            Z 12345 34567
    MCE     A,B       MOVE CHARACTERS AND EDIT                      E 12345 34567
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  COMPARE AND LOOKUP COMMANDS
    OPCODE OPERAND                                                  INSTRUCTION
    C       A,B       COMPARE B-FIELD TO A-FIELD                    C 12345 34567
    LL      A,B       LOOKUP LOW                                    T 12345 34567 1
    LE      A,B       LOOKUP EQUAL                                  T 12345 34567 2
    LLE     A,B       LOOKUP LOW OR EQUAL                           T 12345 34567 3
    LH      A,B       LOOKUP HIGH                                   T 12345 34567 4
    LLH     A,B       LOOKUP LOW OR HIGH                            T 12345 34567 5
    LEH     A,B       LOOKUP EQUAL OR HIGH                          T 12345 34567 6
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  LOGICAL OPERATIONS
    OPCODE OPERAND                                                  INSTRUCTION
    BW      I,B       BRANCH TO I-ADDR IF WORD MARK AT B-ADDRESS    V 56789 34567 1
    BZN     I,B       BRANCH TO I IF B HAS NO ZONE BITS             V 56789 34567 2
    BZN     I,B,AB           IF B HAS A AND B ZONES                 V 56789 34567 B
    BZN     I,B,A            IF B HAS A-BIT AND NO B-BIT            V 56789 34567 S
    BZN     I,B,B            IF B HAS B-BIT AND NO A-BIT            V 56789 34567 K
    BWZ     I,B       BRANCH TO I IF B HAS WM OR NO AB-BITS         V 56789 34567 3
    BWZ     I,B,AB                           OR AB-BITS             V 56789 34567 C
    BWZ     I,B,A                            OR A-BIT               V 56789 34567 T
    BWZ     I,B,B                            OR B-BIT               V 56789 34567 L
    BCE     I,B,D     BRANCH TO I IF CHARACTER AT B EQU D-MOD       B 56789 34567 D
```

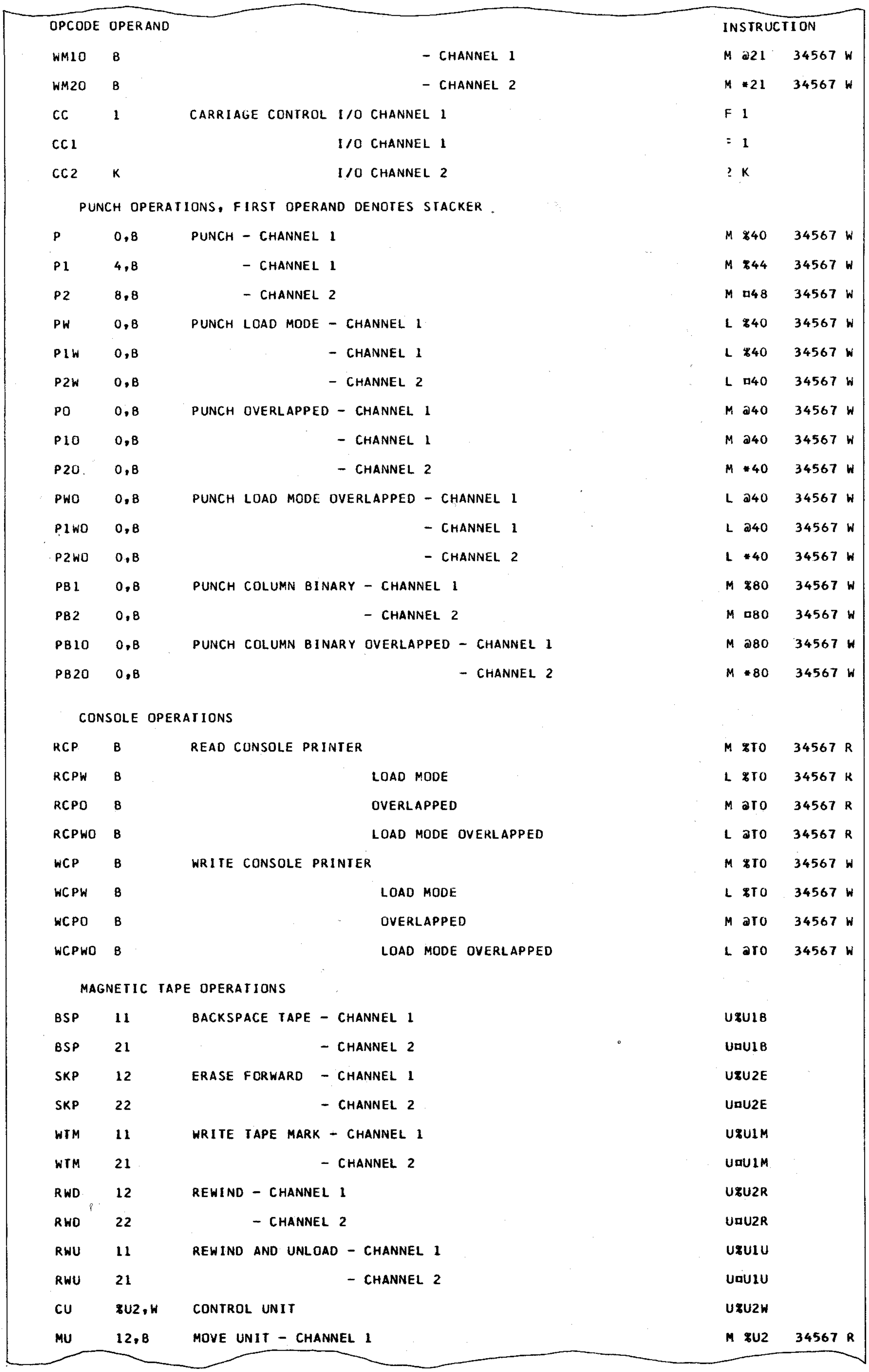
```
   OPCODE OPERAND                                                              INSTRUCTION

   BBE    I,B,D    BRNCH IF ANY BIT AT B MATCHES BIT IN D-MOD                  W 56789 34567 D

   B      I        UNCONDITIONAL BRANCH                                        J 56789

   BU     I        BRANCH IF COMPARE UNEQUAL                                   J 56789 /

   BE     I                          EQUAL                                     J 56789 S

   BL     I                          LOW                                       J 56789 T

   BH     I                          HIGH                                      J 56789 U

   BZ     I        BRANCH IF ZERO BALANCE                                      J 56789 V

   BAV    I        BRANCH IF ARITHMETIC OVERFLOW                               J 56789 Z

   BDV    I        BRANCH IF DIVIDE OVERFLOW                                   J 56789 W

MISCELANEOUS OPERATIONS
   OPCODE OPERAND                                                              INSTRUCTION
   SAR    A        STORE A-REGISTER                                            G 12345 A

   SBR    A        STORE B-REGISTER                                            G 12345 B

   SER    A        STORE E-REGISTER                                            G 12345 E

   SFR    A        STORE F-REGISTER                                            G 12345 F

   SW     A,B      SET WORD MARK AT A AND B                                    , 12345 34567

   CW     A,B      CLEAR WORD MARK AT A AND B                                  ¤ 12345 34567

   CS     B        CLEAR STORAGE                                               / 34567

   CS     I,B      CLEAR STORAGE AND BRANCH                                    / 56789 34567

   H      I        HALT AND BRANCH                                             . 56789

   NOP             NO OPERATION                                                N

   NOPWM           NO OPERATION WORD MARK                                      N

   STC    A        STORE TIME CLOCK                                            G 12345 T
      MNEMONICS FOR SUB-PROGRAMS, RELOCATABLE, AND FORTRAN

   DCWS   NAME     BRANCH TO NAMED SUBROUTINE

   DCWF   NAME     ADCON FOR ENTRY POINT OF NAMED SUBROUTINE

   DAV    1X2,G    DEFINE AREA IN COMMON AREA

   RSV    LABEL    APPLY DOWNWARD RELOCATION TO LABEL

   TITLE  SQRT      100000000000000


      FLOATING POINT ARITHMETIC INSTRUCTIONS

      INTERPRETED BY THE FORTRAN ARITHMETIC ROUTINES

   FRA    A        FLOATING RESET ADD                                          # 12345 R

   FST    A        FLOATING STORE                                              # 12345 L

   FA     A        FLOATING ADD                                                # 12345 A

   FS     A        FLOATING SUBTRACT                                           # 12345 S

   FM     A        FLOATING MULTIPLY                                           # 12345 M

   FD     A        FLOATING DIVIDE                                             # 12345 D

INPUT/OUTPUT COMMANDS
   OPCODE OPERAND                                                              INSTRUCTION
      BRANCH IF I/O CHANNEL STATUS INDICATOR ON

   BEX1   I,D      BRANCH EXTERNAL INDICATOR - CHANNEL 1                       R 56789 D

   BEX2   I,D                                - CHANNEL 2                       X 56789 D
                                                                                       G
   BA1    I        BRANCH ANY EXTERNAL INDICATOR - CHANNEL 1                   R 56789 M
                                                                                       G
   BA2    I                                      - CHANNEL 2                   X 56789 M

   BNR1   I        BRANCH IF NOT READY - CHANNEL 1                             R 56789 I
```



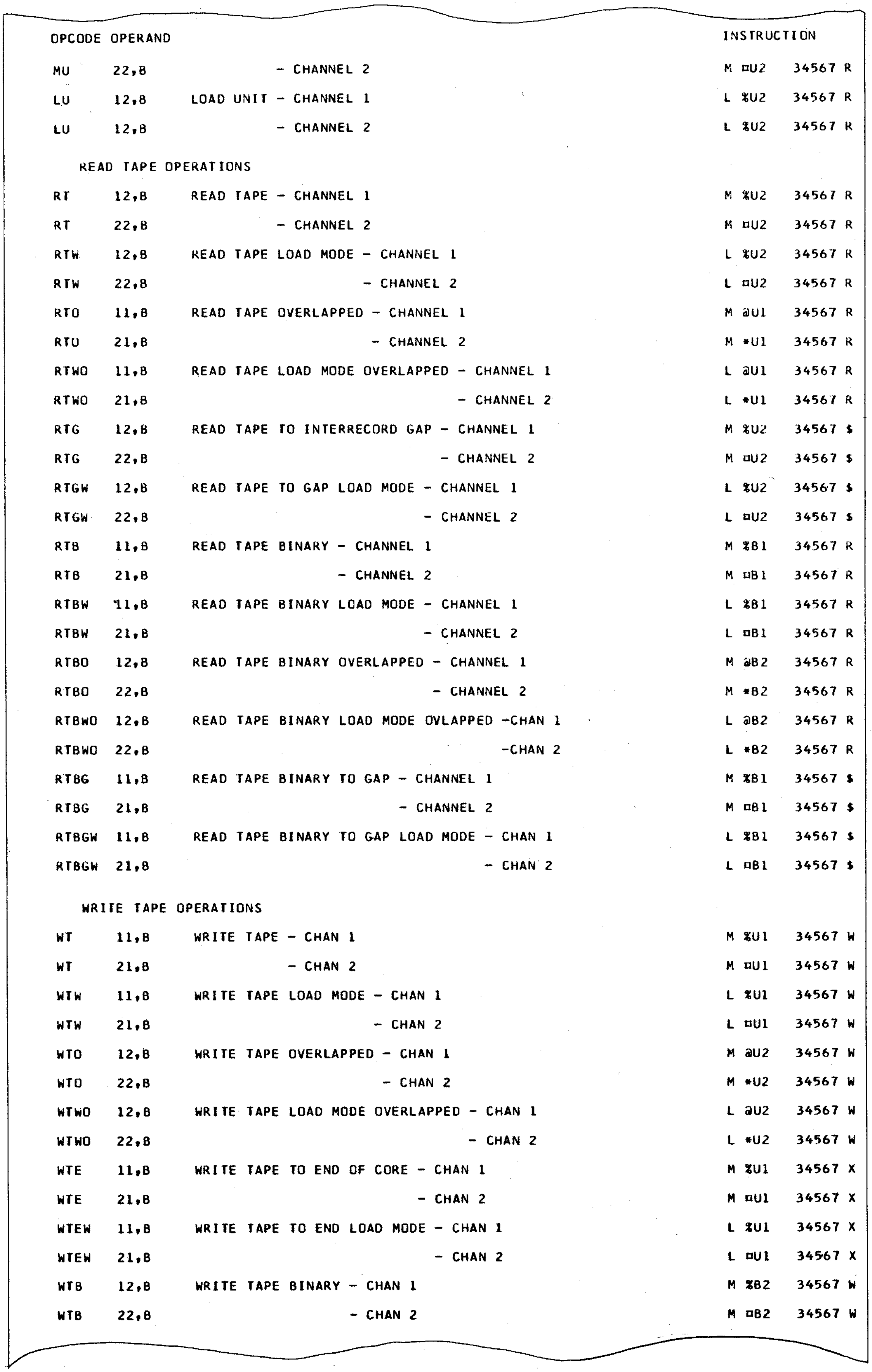
```
OPCODE OPERAND                                                          INSTRUCTION
BNR2    I                          - CHANNEL 2                          X 56789 1

BCB1    I       BRANCH IF CHANNEL 1 BUSY                                R 56789 2

BCB2    I                        2 BUSY                                 X 56789 2

BEF1    I       BRANCH IF END-OF-FILE - CHANNEL 1                       R 56789 8

BEF2    I                             - CHANNEL 2                       X 56789 8
                                                                                C
BNT1    I       BRANCH NO TRANSFER - CHANNEL 1                          R 56789 T
                                                                                C
BNT2    I                          - CHANNEL 2                          X 56789 T

BWL1    I       BRANCH WRONG LENGTH - CHANNEL 1                         R 56789 -

BWL2    I                           - CHANNEL 2                         X 56789 -

BER1    I       BRANCH ERROR - CHANNEL 1                                R 56789 4

BER2    I                    - CHANNEL 2                                X 56789 4

BRC1    I       BRANCH READ BACK CHECK - CHANNEL 1                      R 56789 @

BRC2    I                              - CHANNEL 2                      X 56789 @


   CHANNEL STATUS INDICATORS MAY BE SET WITH IO NOP.

   IO NOP MNEMONIC IS ANY IO OP PRECEDED BY N.

   CONDITIONAL BRANCHES FOR I/O, OVERLAP, AND PRIORITY

BOL1    I       BRANCH OVERLAP IN PROCESS - CHANNEL 1                   J 56789 1

BOL2    I                                 - CHANNEL 2                   J 56789 2

BUPR    I       BRANCH UNIT PRIORITY REQUEST - CHANNEL 1                Y 56789 U

BUPR1   I                                    - CHANNEL 1                Y 56789 U

BUPR2   I                                    - CHANNEL 2                Y 56789 F

BNQ     I       BRANCH INQUIRY REQUEST - CHANNEL 1                      J 56789 Q

BNQ1    I                              - CHANNEL 1                      J 56789 Q

BNQ2    I                              - CHANNEL 2                      J 56789 *

BIPR    I       BRANCH INQUIRY PRIORITY REQUEST - CHAN 1                Y 56789 Q

BIPR1   I                                       - CHAN 1                Y 56789 Q

BIPR2   I                                       - CHAN 2                Y 56789 *

BOQ     I       BRANCH OUTQUIRY - CHANNEL 1                             J 56789 N

BOQ1    I                       - CHANNEL 1                             J 56789 N

BOQ2    I                       - CHANNEL 2                             J 56789 ‡

BQPR    I       BRANCH OUTQUIRY PRIORITY REQUEST - CHAN 1               Y 56789 N

BQPR1   I                                        - CHAN 1               Y 56789 N

BQPR2   I                                        - CHAN 2               Y 56789 ‡

BSPR1   I       BRANCH IF SEEK PRIORITY REQUEST - CHAN 1                Y 56789 S

BSPR2   I                                       - CHAN 2                Y 56789 T

BXPR1   I       BRANCH IF SIMPLEX PRIORITY REQUEST - CHAN 1             Y 56789 A

BXPR2   I                                          - CHAN 2             Y 56789 B

BB1     I       BRANCH IF BINARY CARD - CHANNEL 1                       J 56789 M

BB2     I                             - CHANNEL 2                       J 56789 %

BPCB    I       BRANCH PRINTER CARRIAGE BUSY - CHANNEL 1                J 56789 R

BPCB1   I                                    - CHANNEL 1                J 56789 R

BPCB2   I                                    - CHANNEL 2                J 56789 L

BCV     I       BRANCH CARRIAGE OVERFLOW - CHANNEL 1                    J 56789 @

BCV1    I                                - CHANNEL 1                    J 56789 @
```



```
OPCODE OPERAND                                                       INSTRUCTION

BCV2    I                                - CHANNEL 2                 J 56789 ¤

BC9     I        BRANCH CARRIAGE CHANNEL 9 - CHANNEL 1               J 56789 9

BC91    I                                  - CHANNEL 1               J 56789 9

BC92    I                                  - CHANNEL 2               J 56789 .

BXPA    I        BRANCH AND EXIT PRIORITY ALERT                      Y 56789 X

BEPA    I                   ENTER PRIORITY ALERT                     Y 56789 E

BOPR1   I        BRANCH OVERLAP PRIORITY REQUEST - CHANNEL 1         Y 56789 1

BOPR2   I                                        - CHANNEL 2         Y 56789 2


   UNIT RECORD OPERATIONS

   READ A CARD. FIRST OPERAND DENOTES STACKER POCKET

R       0,B      READ - CHANNEL 1                                    M %10    34567 R

R1      1,B           - CHANNEL 1                                    M %11    34567 R

R2      2,B           - CHANNEL 2                                    M ¤12    34567 R

RW      1,B      READ LOAD MODE - CHANNEL 1                          L %11    34567 R

R1W     1,B                     - CHANNEL 1                          L %11    34567 R

R2W     1,B                     - CHANNEL 2                          L ¤11    34567 R

RO      1,B      READ OVERLAPPED - CHANNEL 1                         M @11    34567 R

R1O     1,B                      - CHANNEL 1                         M @11    34567 R

R2O     1,B                      - CHANNEL 2                         M *11    34567 R

RWO     1,B      READ LOAD MODE OVERLAPPED - CHANNEL 1               L @11    34567 R

R1WO    1,B                                - CHANNEL 1               L @11    34567 R

R2WO    1,B                                - CHANNEL 2               L *11    34567 R

   SELECT STACKER AND FEED

SSF     0        SELECT STACKER 0 AND FEED - CHANNEL 1               K 0

SSF1    1               STACKER 1          - CHANNEL 1               K 1

SSF2    2               STACKER 2          - CHANNEL 2               4 2


   PRINTER OPERATIONS

W       B        WRITE PRINTER - CHANNEL 1                           M %20    34567 W

W1      B                      - CHANNEL 1                           M %20    34567 W

W2      B                      - CHANNEL 2                           M ¤20    34567 W

WW      B        WRITE PRINTER LOAD MODE - CHANNEL 1                 L %20    34567 W

W1W     B                                - CHANNEL 1                 L %20    34567 W

W2W     B                                - CHANNEL 2                 L ¤20    34567 W

WO      B        WRITE PRINTER OVERLAPPED - CHANNEL 1                M @20    34567 W

W1O     B                                 - CHANNEL 1                M @20    34567 W

W2O     B                                 - CHANNEL 2                M *20    34567 W

WWO     B        WRITE PRINTER LOAD MODE OVERLAPPED - CHAN 1         L @20    34567 W

W1WO    B                                           - CHAN 1         L @20    34567 W

W2WO    B                                           - CHAN 2         L *20    34567 W

WM      B        WRITE WORDMARKS - CHANNEL 1                         M %21    34567 W

WM1     B                        - CHANNEL 1                         M %21    34567 W

WM2     B                        - CHANNEL 2                         M ¤21    34567 W

WMO     B        WRITE WORDMARKS OVERLAPPED - CHANNEL 1              M @21    34567 W
```

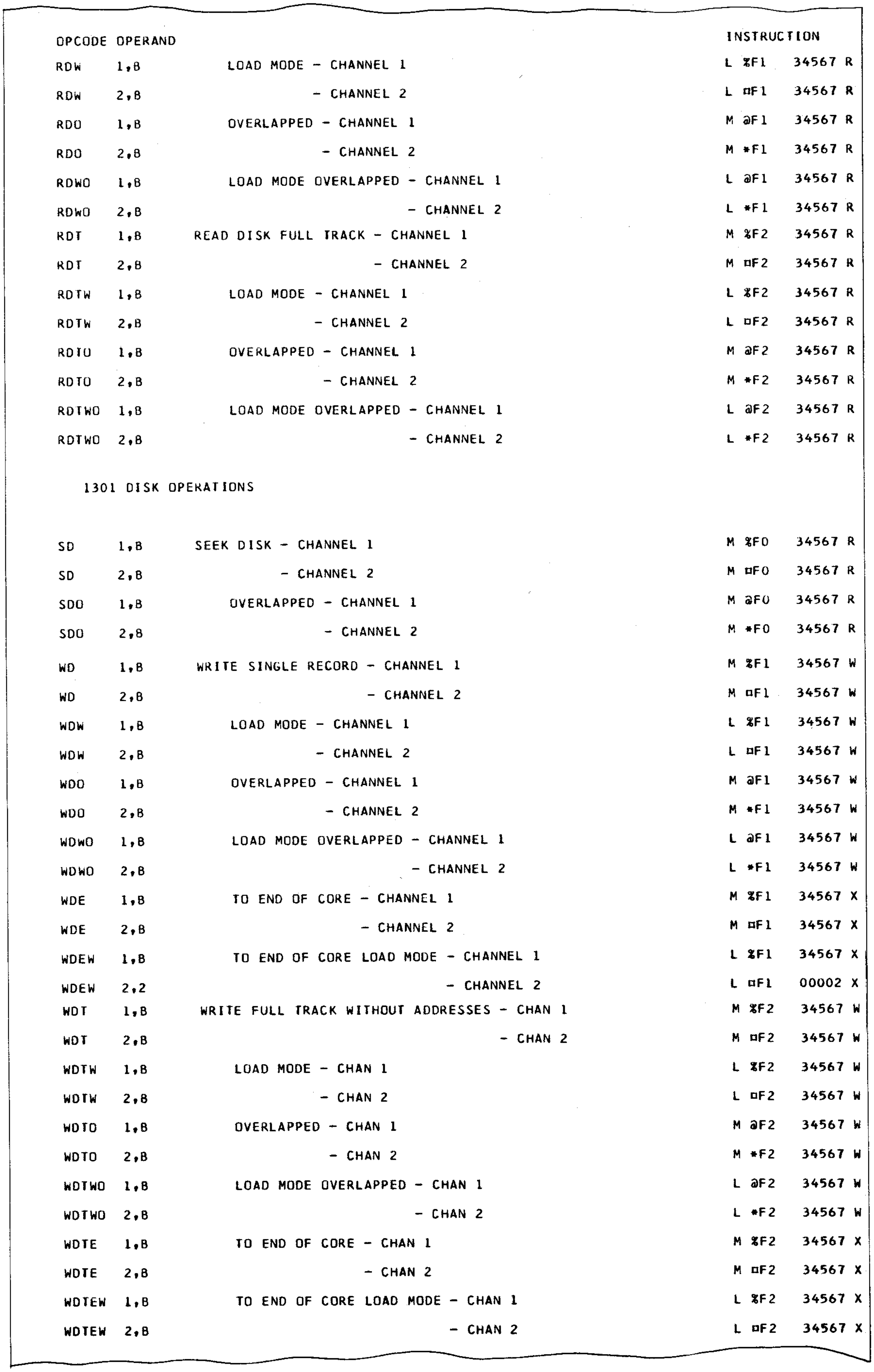


```
OPCODE OPERAND                                                                INSTRUCTION
WM10   B                                   - CHANNEL 1                        M @21   34567 W
WM20   B                                   - CHANNEL 2                        M *21   34567 W
CC     1        CARRIAGE CONTROL I/O CHANNEL 1                                F 1
CC1                              I/O CHANNEL 1                                = 1
CC2    K                         I/O CHANNEL 2                                ? K
   PUNCH OPERATIONS, FIRST OPERAND DENOTES STACKER
P      0,B      PUNCH - CHANNEL 1                                             M %40   34567 W
P1     4,B            - CHANNEL 1                                             M %44   34567 W
P2     8,B            - CHANNEL 2                                             M ⌑48   34567 W
PW     0,B      PUNCH LOAD MODE - CHANNEL 1                                   L %40   34567 W
P1W    0,B                      - CHANNEL 1                                   L %40   34567 W
P2W    0,B                      - CHANNEL 2                                   L ⌑40   34567 W
PO     0,B      PUNCH OVERLAPPED - CHANNEL 1                                  M @40   34567 W
P1O    0,B                       - CHANNEL 1                                  M @40   34567 W
P2O    0,B                       - CHANNEL 2                                  M *40   34567 W
PWO    0,B      PUNCH LOAD MODE OVERLAPPED - CHANNEL 1                        L @40   34567 W
P1WO   0,B                                 - CHANNEL 1                        L @40   34567 W
P2WO   0,B                                 - CHANNEL 2                        L *40   34567 W
PB1    0,B      PUNCH COLUMN BINARY - CHANNEL 1                               M %80   34567 W
PB2    0,B                          - CHANNEL 2                               M ⌑80   34567 W
PB1O   0,B      PUNCH COLUMN BINARY OVERLAPPED - CHANNEL 1                    M @80   34567 W
PB2O   0,B                                     - CHANNEL 2                    M *80   34567 W

   CONSOLE OPERATIONS
RCP    B        READ CONSOLE PRINTER                                          M %TO   34567 R
RCPW   B                             LOAD MODE                                L %TO   34567 R
RCPO   B                             OVERLAPPED                               M @TO   34567 R
RCPWO  B                             LOAD MODE OVERLAPPED                     L @TO   34567 R
WCP    B        WRITE CONSOLE PRINTER                                         M %TO   34567 W
WCPW   B                              LOAD MODE                               L %TO   34567 W
WCPO   B                              OVERLAPPED                              M @TO   34567 W
WCPWO  B                              LOAD MODE OVERLAPPED                    L @TO   34567 W

   MAGNETIC TAPE OPERATIONS
BSP    11       BACKSPACE TAPE - CHANNEL 1                                    U%U1B
BSP    21                      - CHANNEL 2                                    U⌑U1B
SKP    12       ERASE FORWARD  - CHANNEL 1                                    U%U2E
SKP    22                      - CHANNEL 2                                    U⌑U2E
WTM    11       WRITE TAPE MARK - CHANNEL 1                                   U%U1M
WTM    21                      - CHANNEL 2                                    U⌑U1M
RWD    12       REWIND - CHANNEL 1                                            U%U2R
RWD    22              - CHANNEL 2                                            U⌑U2R
RWU    11       REWIND AND UNLOAD - CHANNEL 1                                 U%U1U
RWU    21                         - CHANNEL 2                                 U⌑U1U
CU     %U2,W    CONTROL UNIT                                                  U%U2W
MU     12,B     MOVE UNIT - CHANNEL 1                                         M %U2   34567 R
```

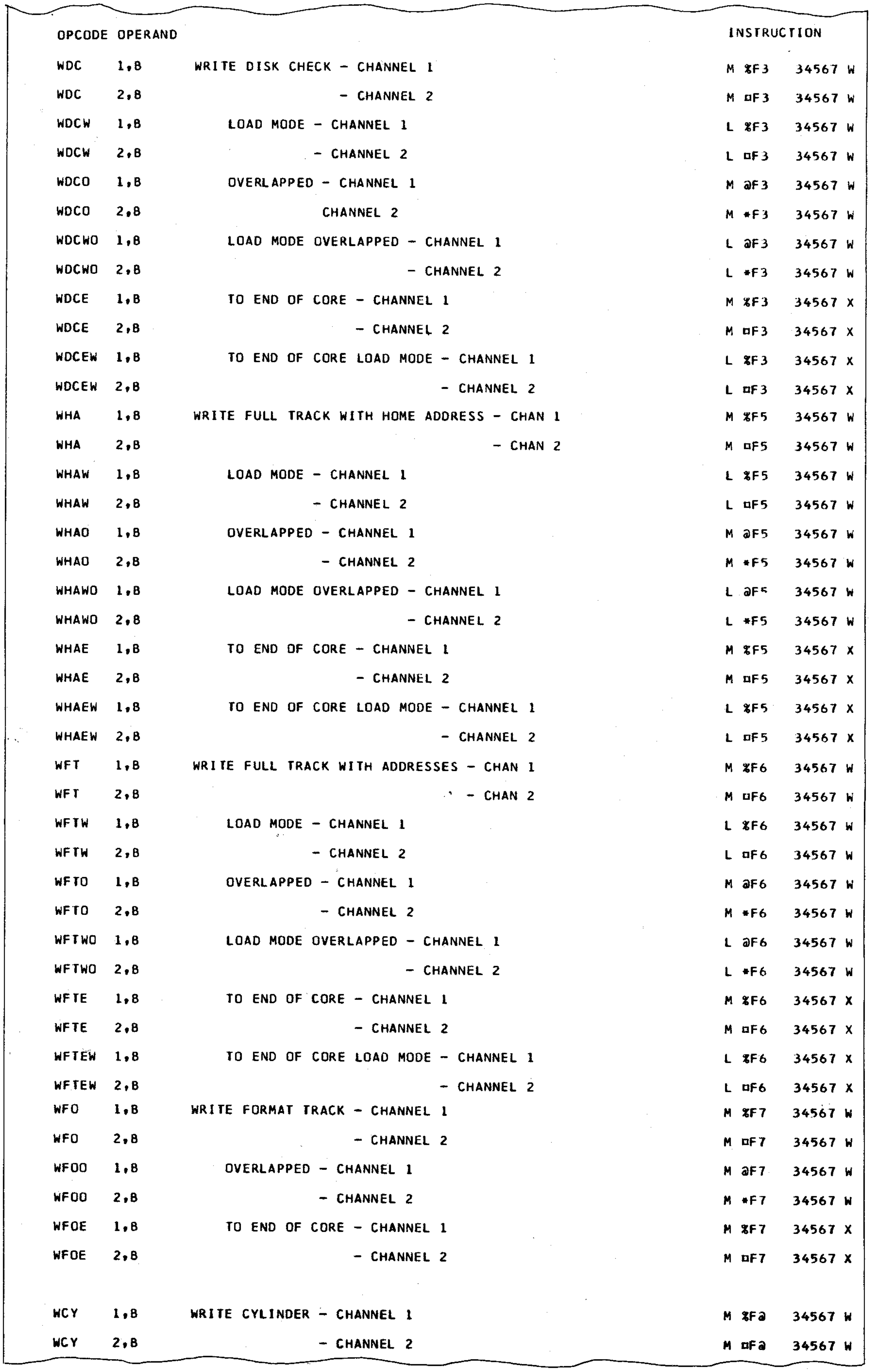


```
OPCODE OPERAND                                                        INSTRUCTION

MU      22,B              - CHANNEL 2                                 M ⌑U2    34567 R

LU      12,B     LOAD UNIT - CHANNEL 1                                L %U2    34567 R

LU      12,B              - CHANNEL 2                                 L %U2    34567 R

   READ TAPE OPERATIONS

RT      12,B     READ TAPE - CHANNEL 1                                M %U2    34567 R

RT      22,B              - CHANNEL 2                                 M ⌑U2    34567 R

RTW     12,B     READ TAPE LOAD MODE - CHANNEL 1                      L %U2    34567 R

RTW     22,B                        - CHANNEL 2                       L ⌑U2    34567 R

RTO     11,B     READ TAPE OVERLAPPED - CHANNEL 1                     M @U1    34567 R

RTO     21,B                         - CHANNEL 2                      M *U1    34567 R

RTWO    11,B     READ TAPE LOAD MODE OVERLAPPED - CHANNEL 1           L @U1    34567 R

RTWO    21,B                                    - CHANNEL 2           L *U1    34567 R

RTG     12,B     READ TAPE TO INTERRECORD GAP - CHANNEL 1             M %U2    34567 $

RTG     22,B                                  - CHANNEL 2             M ⌑U2    34567 $

RTGW    12,B     READ TAPE TO GAP LOAD MODE - CHANNEL 1               L %U2    34567 $

RTGW    22,B                                - CHANNEL 2               L ⌑U2    34567 $

RTB     11,B     READ TAPE BINARY - CHANNEL 1                         M %B1    34567 R

RTB     21,B                     - CHANNEL 2                          M ⌑B1    34567 R

RTBW    11,B     READ TAPE BINARY LOAD MODE - CHANNEL 1               L %B1    34567 R

RTBW    21,B                                - CHANNEL 2               L ⌑B1    34567 R

RTBO    12,B     READ TAPE BINARY OVERLAPPED - CHANNEL 1              M @B2    34567 R

RTBO    22,B                                 - CHANNEL 2              M *B2    34567 R

RTBWO   12,B     READ TAPE BINARY LOAD MODE OVLAPPED -CHAN 1          L @B2    34567 R

RTBWO   22,B                                         -CHAN 2          L *B2    34567 R

RTBG    11,B     READ TAPE BINARY TO GAP - CHANNEL 1                  M %B1    34567 $

RTBG    21,B                             - CHANNEL 2                  M ⌑B1    34567 $

RTBGW   11,B     READ TAPE BINARY TO GAP LOAD MODE - CHAN 1           L %B1    34567 $

RTBGW   21,B                                       - CHAN 2           L ⌑B1    34567 $

   WRITE TAPE OPERATIONS

WT      11,B     WRITE TAPE - CHAN 1                                  M %U1    34567 W

WT      21,B                - CHAN 2                                  M ⌑U1    34567 W

WTW     11,B     WRITE TAPE LOAD MODE - CHAN 1                        L %U1    34567 W

WTW     21,B                          - CHAN 2                        L ⌑U1    34567 W

WTO     12,B     WRITE TAPE OVERLAPPED - CHAN 1                       M @U2    34567 W

WTO     22,B                           - CHAN 2                       M *U2    34567 W

WTWO    12,B     WRITE TAPE LOAD MODE OVERLAPPED - CHAN 1             L @U2    34567 W

WTWO    22,B                                     - CHAN 2             L *U2    34567 W

WTE     11,B     WRITE TAPE TO END OF CORE - CHAN 1                   M %U1    34567 X

WTE     21,B                               - CHAN 2                   M ⌑U1    34567 X

WTEW    11,B     WRITE TAPE TO END LOAD MODE - CHAN 1                 L %U1    34567 X

WTEW    21,B                                 - CHAN 2                 L ⌑U1    34567 X

WTB     12,B     WRITE TAPE BINARY - CHAN 1                           M %B2    34567 W

WTB     22,B                       - CHAN 2                           M ⌑B2    34567 W
```

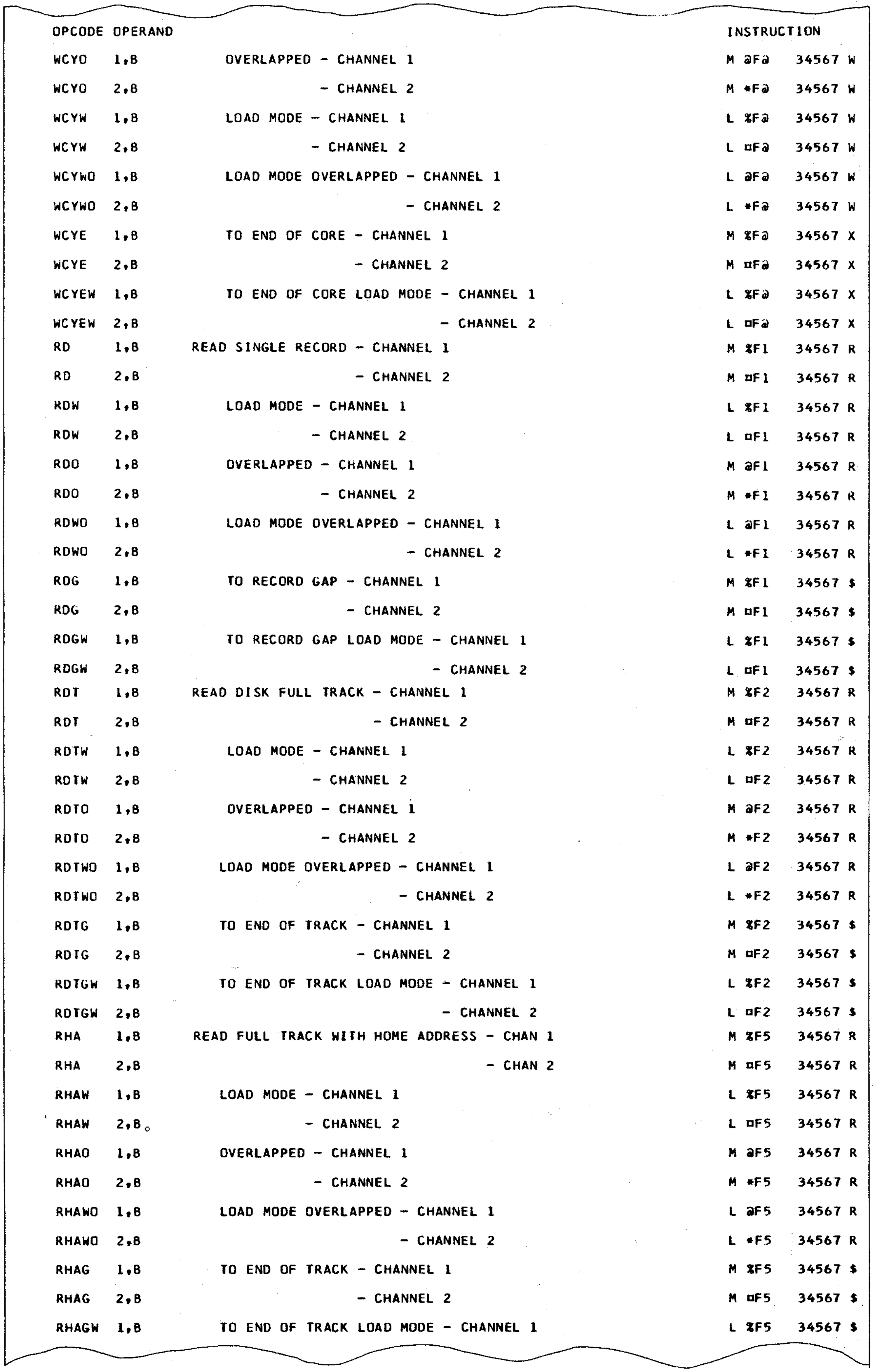
```
OPCODE OPERAND                                                        INSTRUCTION
WTBW   12,B     WRITE TAPE BINARY LOAD MODE - CHAN 1                  L %B2    34567 W
WTBW   22,B                                 - CHAN 2                  L ¤B2    34567 W
WTBO   11,B     WRITE TAPE BINARY OVERLAPPED - CHAN 1                 M @B1    34567 W
WTBO   21,B                                - CHAN 2                   M *B1    34567 W
WTBWO  11,B     WRITE TAPE BINARY LOAD MODE OLAPPED -CHAN 1           L @B1    34567 W
WTBWO  21,B                                         -CHAN 2           L *B1    34567 W
WTBE   12,B     WRITE TAPE BINARY TO END OF CORE - CHAN 1             M %B2    34567 X
WTBE   22,B                                      - CHAN 2             M ¤B2    34567 X
WTBEW  12,B     WRITE TAPE BINARY TO END LOAD MODE - CHAN 1           L %B2    34567 X
WTBEW  22,B                                        - CHAN 2           L ¤B2    34567 X

   1405 DISK OPERATIONS
SD     1,B      SEEK DISK - CHANNEL 1                                 M %F0    34567 R
SD     2,B                - CHANNEL 2                                 M ¤F0    34567 R
SDO    1,B      SEEK DISK OVERLAPPED - CHANNEL 1                      M @F0    34567 R
SDO    2,B                           - CHANNEL 2                      M *F0    34567 R


WD     1,B      WRITE DISK SINGLE RECORD - CHANNEL 1                  M %F1    34567 W
WD     2,B                               - CHANNEL 2                  M ¤F1    34567 W
WDW    1,B          LOAD MODE - CHANNEL 1                             L %F1    34567 W
WDW    2,B                    - CHANNEL 2                             L ¤F1    34567 W
WDO    1,B          OVERLAPPED - CHANNEL 1                            M @F1    34567 W
WDO    2,B                     - CHANNEL 2                            M *F1    34567 W
WDWO   1,B          LOAD MODE OVERLAPPED - CHANNEL 1                  L @F1    34567 W
WDWO   2,B                               - CHANNEL 2                  L *F1    34567 W


WDT    1,B      WRITE FULL TRACK - CHANNEL 1                          M %F2    34567 W
WDT    2,B                       - CHANNEL 2                          M ¤F2    34567 W
WDTW   1,B          LOAD MODE - CHANNEL 1                             L %F2    34567 W
WDTW   2,B                    - CHANNEL 2                             L ¤F2    34567 W
WDTO   1,B          OVERLAPPED - CHANNEL 1                            M @F2    34567 W
WDTO   2,B                     - CHANNEL 2                            M *F2    34567 W
WDTWO  1,B          LOAD MODE OVERLAPPED - CHANNEL 1                  L @F2    34567 W
WDTWO  2,B                               - CHANNEL 2                  L *F2    34567 W

WDC    1,B      WRITE DISK CHECK - CHANNEL 1                          M %F3    34567 W
WDC    2,B                       - CHANNEL 2                          M ¤F3    34567 W
WDCW   1,B          LOAD MODE - CHANNEL 1                             L %F3    34567 W
WDCW   2,B                    - CHANNEL 2                             L ¤F3    34567 W
WDCO   1,B          OVERLAPPED - CHANNEL 1                            M @F3    34567 W
WDCO   2,B                     - CHANNEL 2                            M *F3    34567 W
WDCWO  1,B          LOAD MODE OVERLAPPED - CHANNEL 1                  L @F3    34567 W
WDCWO  2,B                               - CHANNEL 2                  L *F3    34567 W

RD     1,B      READ DISK SINGLE RECORD - CHANNEL 1                   M %F1    34567 R
RD     2,B                              - CHANNEL 2                   M ¤F1    34567 R
```



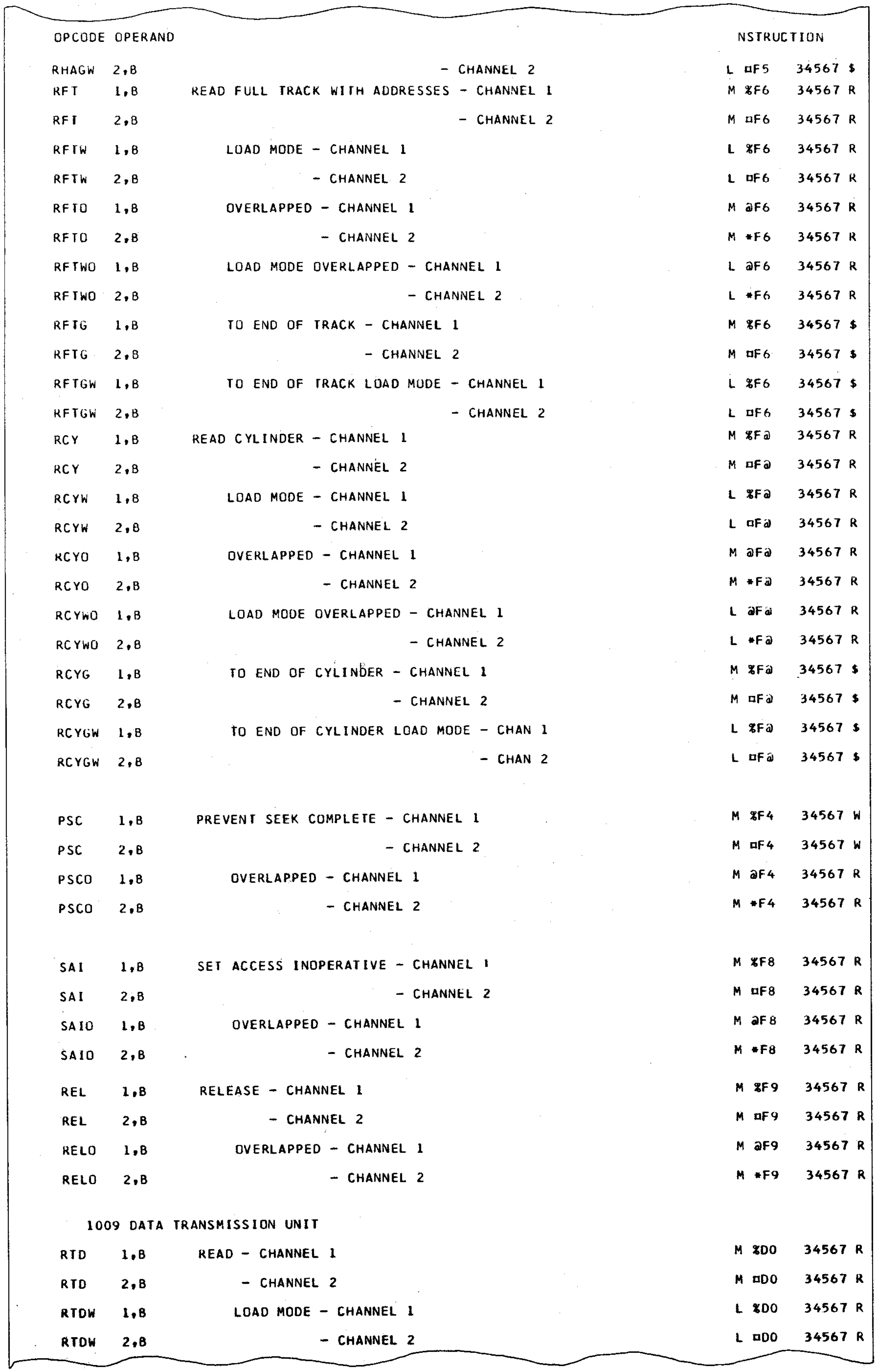
```
OPCODE OPERAND                                                           INSTRUCTION
RDW     1,B         LOAD MODE - CHANNEL 1                                L %F1   34567 R
RDW     2,B                   - CHANNEL 2                                L ¤F1   34567 R
RDO     1,B         OVERLAPPED - CHANNEL 1                               M @F1   34567 R
RDO     2,B                    - CHANNEL 2                               M *F1   34567 R
RDWO    1,B         LOAD MODE OVERLAPPED - CHANNEL 1                     L @F1   34567 R
RDWO    2,B                              - CHANNEL 2                     L *F1   34567 R
RDT     1,B     READ DISK FULL TRACK - CHANNEL 1                         M %F2   34567 R
RDT     2,B                          - CHANNEL 2                         M ¤F2   34567 R
RDTW    1,B         LOAD MODE - CHANNEL 1                                L %F2   34567 R
RDTW    2,B                   - CHANNEL 2                                L ¤F2   34567 R
RDTO    1,B         OVERLAPPED - CHANNEL 1                               M @F2   34567 R
RDTO    2,B                    - CHANNEL 2                               M *F2   34567 R
RDTWO   1,B         LOAD MODE OVERLAPPED - CHANNEL 1                     L @F2   34567 R
RDTWO   2,B                              - CHANNEL 2                     L *F2   34567 R

   1301 DISK OPERATIONS

SD      1,B     SEEK DISK - CHANNEL 1                                    M %F0   34567 R
SD      2,B               - CHANNEL 2                                    M ¤F0   34567 R
SDO     1,B         OVERLAPPED - CHANNEL 1                               M @F0   34567 R
SDO     2,B                    - CHANNEL 2                               M *F0   34567 R
WD      1,B     WRITE SINGLE RECORD - CHANNEL 1                          M %F1   34567 W
WD      2,B                         - CHANNEL 2                          M ¤F1   34567 W
WDW     1,B         LOAD MODE - CHANNEL 1                                L %F1   34567 W
WDW     2,B                   - CHANNEL 2                                L ¤F1   34567 W
WDO     1,B         OVERLAPPED - CHANNEL 1                               M @F1   34567 W
WDO     2,B                    - CHANNEL 2                               M *F1   34567 W
WDWO    1,B         LOAD MODE OVERLAPPED - CHANNEL 1                     L @F1   34567 W
WDWO    2,B                              - CHANNEL 2                     L *F1   34567 W
WDE     1,B         TO END OF CORE - CHANNEL 1                           M %F1   34567 X
WDE     2,B                        - CHANNEL 2                           M ¤F1   34567 X
WDEW    1,B         TO END OF CORE LOAD MODE - CHANNEL 1                 L %F1   34567 X
WDEW    2,2                                  - CHANNEL 2                 L ¤F1   00002 X
WDT     1,B     WRITE FULL TRACK WITHOUT ADDRESSES - CHAN 1              M %F2   34567 W
WDT     2,B                                        - CHAN 2              M ¤F2   34567 W
WDTW    1,B         LOAD MODE - CHAN 1                                   L %F2   34567 W
WDTW    2,B                   - CHAN 2                                   L ¤F2   34567 W
WDTO    1,B         OVERLAPPED - CHAN 1                                  M @F2   34567 W
WDTO    2,B                    - CHAN 2                                  M *F2   34567 W
WDTWO   1,B         LOAD MODE OVERLAPPED - CHAN 1                        L @F2   34567 W
WDTWO   2,B                              - CHAN 2                        L *F2   34567 W
WDTE    1,B         TO END OF CORE - CHAN 1                              M %F2   34567 X
WDTE    2,B                        - CHAN 2                              M ¤F2   34567 X
WDTEW   1,B         TO END OF CORE LOAD MODE - CHAN 1                    L %F2   34567 X
WDTEW   2,B                                  - CHAN 2                    L ¤F2   34567 X
```



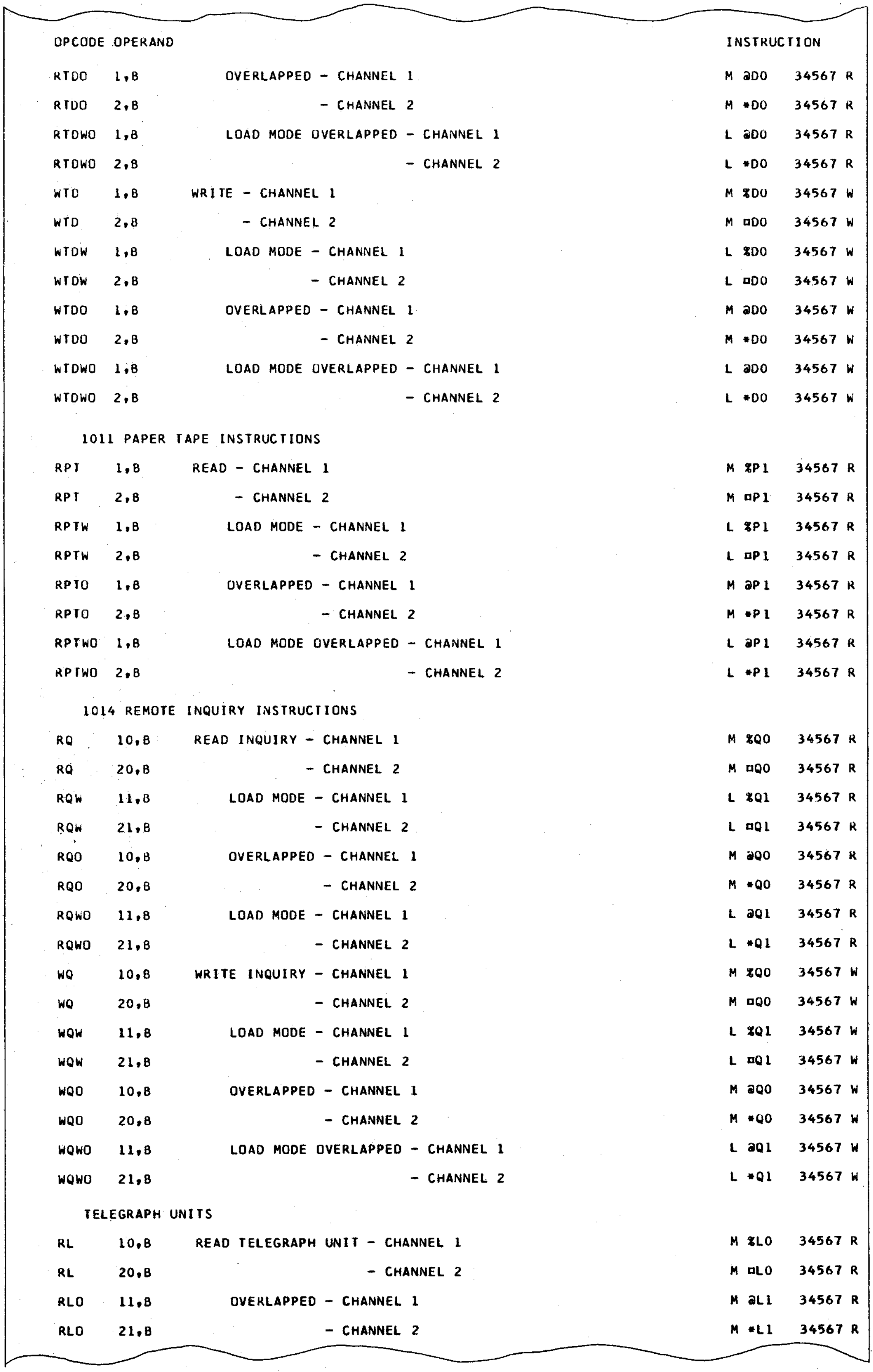
OPCODE	OPERAND		INSTRUCTION
WDC	1,B	WRITE DISK CHECK - CHANNEL 1	M %F3 34567 W
WDC	2,B	- CHANNEL 2	M □F3 34567 W
WDCW	1,B	LOAD MODE - CHANNEL 1	L %F3 34567 W
WDCW	2,B	- CHANNEL 2	L □F3 34567 W
WDCO	1,B	OVERLAPPED - CHANNEL 1	M @F3 34567 W
WDCO	2,B	CHANNEL 2	M *F3 34567 W
WDCWO	1,B	LOAD MODE OVERLAPPED - CHANNEL 1	L @F3 34567 W
WDCWO	2,B	- CHANNEL 2	L *F3 34567 W
WDCE	1,B	TO END OF CORE - CHANNEL 1	M %F3 34567 X
WDCE	2,B	- CHANNEL 2	M □F3 34567 X
WDCEW	1,B	TO END OF CORE LOAD MODE - CHANNEL 1	L %F3 34567 X
WDCEW	2,B	- CHANNEL 2	L □F3 34567 X
WHA	1,B	WRITE FULL TRACK WITH HOME ADDRESS - CHAN 1	M %F5 34567 W
WHA	2,B	- CHAN 2	M □F5 34567 W
WHAW	1,B	LOAD MODE - CHANNEL 1	L %F5 34567 W
WHAW	2,B	- CHANNEL 2	L □F5 34567 W
WHAO	1,B	OVERLAPPED - CHANNEL 1	M @F5 34567 W
WHAO	2,B	- CHANNEL 2	M *F5 34567 W
WHAWO	1,B	LOAD MODE OVERLAPPED - CHANNEL 1	L @F5 34567 W
WHAWO	2,B	- CHANNEL 2	L *F5 34567 W
WHAE	1,B	TO END OF CORE - CHANNEL 1	M %F5 34567 X
WHAE	2,B	- CHANNEL 2	M □F5 34567 X
WHAEW	1,B	TO END OF CORE LOAD MODE - CHANNEL 1	L %F5 34567 X
WHAEW	2,B	- CHANNEL 2	L □F5 34567 X
WFT	1,B	WRITE FULL TRACK WITH ADDRESSES - CHAN 1	M %F6 34567 W
WFT	2,B	- CHAN 2	M □F6 34567 W
WFTW	1,B	LOAD MODE - CHANNEL 1	L %F6 34567 W
WFTW	2,B	- CHANNEL 2	L □F6 34567 W
WFTO	1,B	OVERLAPPED - CHANNEL 1	M @F6 34567 W
WFTO	2,B	- CHANNEL 2	M *F6 34567 W
WFTWO	1,B	LOAD MODE OVERLAPPED - CHANNEL 1	L @F6 34567 W
WFTWO	2,B	- CHANNEL 2	L *F6 34567 W
WFTE	1,B	TO END OF CORE - CHANNEL 1	M %F6 34567 X
WFTE	2,B	- CHANNEL 2	M □F6 34567 X
WFTEW	1,B	TO END OF CORE LOAD MODE - CHANNEL 1	L %F6 34567 X
WFTEW	2,B	- CHANNEL 2	L □F6 34567 X
WFO	1,B	WRITE FORMAT TRACK - CHANNEL 1	M %F7 34567 W
WFO	2,B	- CHANNEL 2	M □F7 34567 W
WFOO	1,B	OVERLAPPED - CHANNEL 1	M @F7 34567 W
WFOO	2,B	- CHANNEL 2	M *F7 34567 W
WFOE	1,B	TO END OF CORE - CHANNEL 1	M %F7 34567 X
WFOE	2,B	- CHANNEL 2	M □F7 34567 X
WCY	1,B	WRITE CYLINDER - CHANNEL 1	M %F@ 34567 W
WCY	2,B	- CHANNEL 2	M □F@ 34567 W



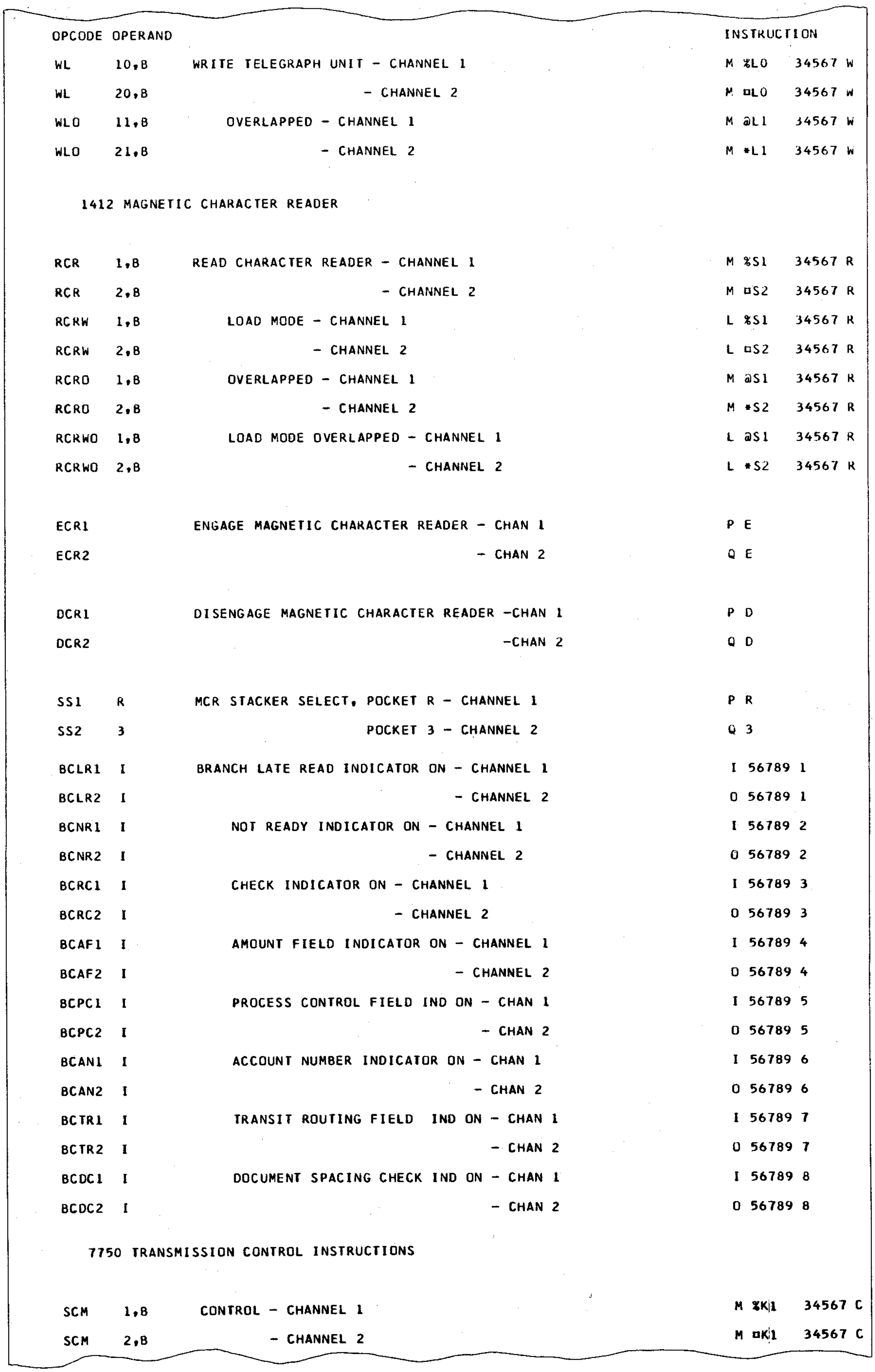
```
OPCODE OPERAND                                                                INSTRUCTION
WCYO    1,B       OVERLAPPED - CHANNEL 1                                      M @F@    34567 W
WCYO    2,B                  - CHANNEL 2                                      M *F@    34567 W
WCYW    1,B       LOAD MODE - CHANNEL 1                                       L %F@    34567 W
WCYW    2,B                 - CHANNEL 2                                       L ⌑F@    34567 W
WCYWO   1,B       LOAD MODE OVERLAPPED - CHANNEL 1                            L @F@    34567 W
WCYWO   2,B                            - CHANNEL 2                            L *F@    34567 W
WCYE    1,B       TO END OF CORE - CHANNEL 1                                  M %F@    34567 X
WCYE    2,B                      - CHANNEL 2                                  M ⌑F@    34567 X
WCYEW   1,B       TO END OF CORE LOAD MODE - CHANNEL 1                        L %F@    34567 X
WCYEW   2,B                                - CHANNEL 2                        L ⌑F@    34567 X
RD      1,B     READ SINGLE RECORD - CHANNEL 1                                M %F1    34567 R
RD      2,B                        - CHANNEL 2                                M ⌑F1    34567 R
RDW     1,B       LOAD MODE - CHANNEL 1                                       L %F1    34567 R
RDW     2,B                 - CHANNEL 2                                       L ⌑F1    34567 R
RDO     1,B       OVERLAPPED - CHANNEL 1                                      M @F1    34567 R
RDO     2,B                  - CHANNEL 2                                      M *F1    34567 R
RDWO    1,B       LOAD MODE OVERLAPPED - CHANNEL 1                            L @F1    34567 R
RDWO    2,B                            - CHANNEL 2                            L *F1    34567 R
RDG     1,B       TO RECORD GAP - CHANNEL 1                                   M %F1    34567 $
RDG     2,B                     - CHANNEL 2                                   M ⌑F1    34567 $
RDGW    1,B       TO RECORD GAP LOAD MODE - CHANNEL 1                         L %F1    34567 $
RDGW    2,B                               - CHANNEL 2                         L ⌑F1    34567 $
RDT     1,B     READ DISK FULL TRACK - CHANNEL 1                              M %F2    34567 R
RDT     2,B                          - CHANNEL 2                              M ⌑F2    34567 R
RDTW    1,B       LOAD MODE - CHANNEL 1                                       L %F2    34567 R
RDTW    2,B                 - CHANNEL 2                                       L ⌑F2    34567 R
RDTO    1,B       OVERLAPPED - CHANNEL 1                                      M @F2    34567 R
RDTO    2,B                  - CHANNEL 2                                      M *F2    34567 R
RDTWO   1,B      LOAD MODE OVERLAPPED - CHANNEL 1                             L @F2    34567 R
RDTWO   2,B                           - CHANNEL 2                             L *F2    34567 R
RDTG    1,B      TO END OF TRACK - CHANNEL 1                                  M %F2    34567 $
RDTG    2,B                      - CHANNEL 2                                  M ⌑F2    34567 $
RDTGW   1,B      TO END OF TRACK LOAD MODE - CHANNEL 1                        L %F2    34567 $
RDTGW   2,B                                - CHANNEL 2                        L ⌑F2    34567 $
RHA     1,B     READ FULL TRACK WITH HOME ADDRESS - CHAN 1                    M %F5    34567 R
RHA     2,B                                       - CHAN 2                    M ⌑F5    34567 R
RHAW    1,B      LOAD MODE - CHANNEL 1                                        L %F5    34567 R
RHAW    2,B                - CHANNEL 2                                        L ⌑F5    34567 R
RHAO    1,B      OVERLAPPED - CHANNEL 1                                       M @F5    34567 R
RHAO    2,B                 - CHANNEL 2                                       M *F5    34567 R
RHAWO   1,B      LOAD MODE OVERLAPPED - CHANNEL 1                             L @F5    34567 R
RHAWO   2,B                           - CHANNEL 2                             L *F5    34567 R
RHAG    1,B      TO END OF TRACK - CHANNEL 1                                  M %F5    34567 $
RHAG    2,B                      - CHANNEL 2                                  M ⌑F5    34567 $
RHAGW   1,B      TO END OF TRACK LOAD MODE - CHANNEL 1                        L %F5    34567 $
```



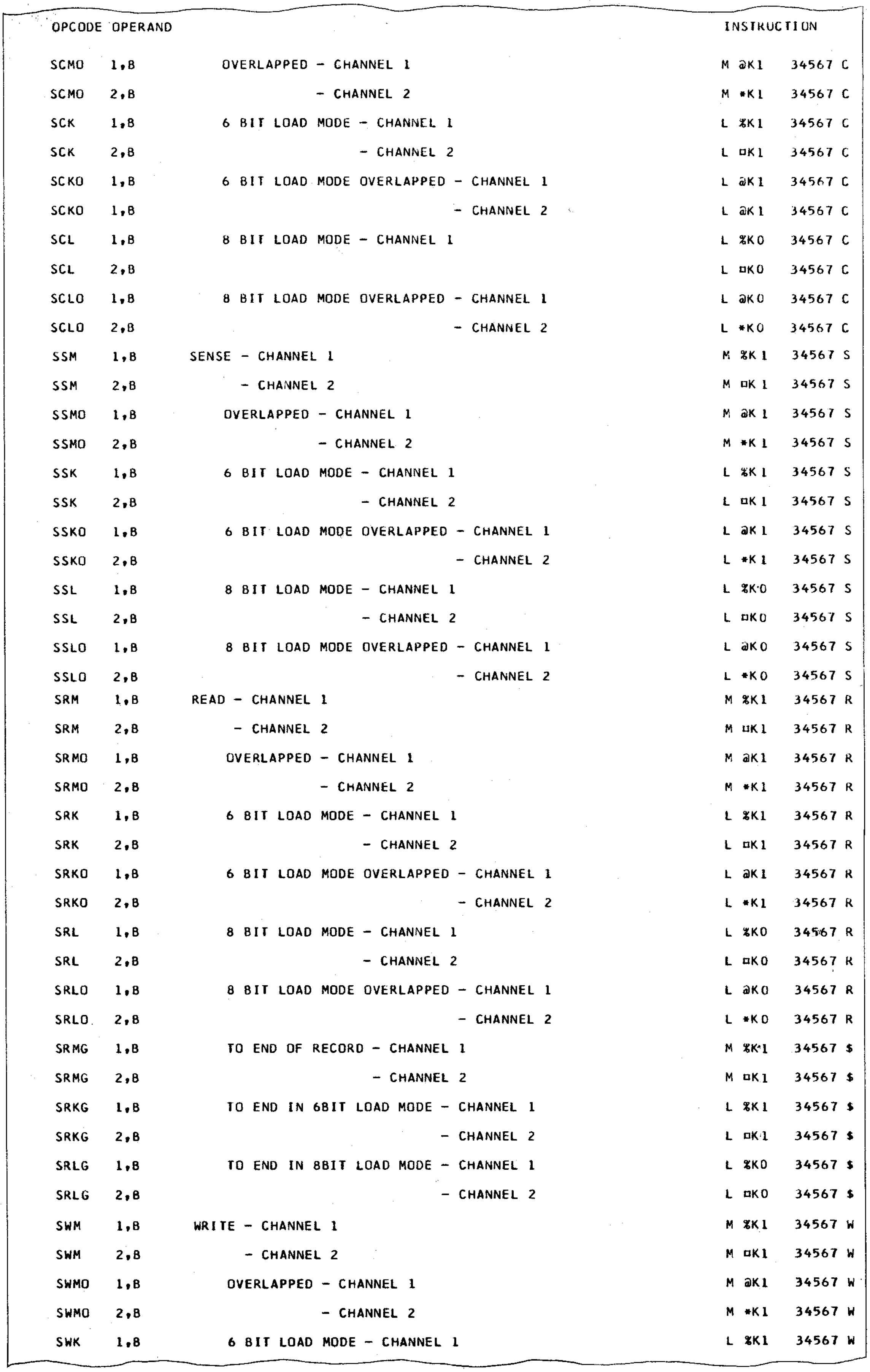
```
OPCODE OPERAND                                                    NSTRUCTION

RHAGW   2,B                                       - CHANNEL 2     L ¤F5   34567 $
RFT     1,B     READ FULL TRACK WITH ADDRESSES - CHANNEL 1        M %F6   34567 R
RFT     2,B                                         - CHANNEL 2   M ¤F6   34567 R
RFTW    1,B        LOAD MODE - CHANNEL 1                          L %F6   34567 R
RFTW    2,B                  - CHANNEL 2                          L ¤F6   34567 R
RFTO    1,B        OVERLAPPED - CHANNEL 1                         M @F6   34567 R
RFTO    2,B                   - CHANNEL 2                         M *F6   34567 R
RFTWO   1,B        LOAD MODE OVERLAPPED - CHANNEL 1               L @F6   34567 R
RFTWO   2,B                             - CHANNEL 2               L *F6   34567 R
RFTG    1,B        TO END OF TRACK - CHANNEL 1                    M %F6   34567 $
RFTG    2,B                        - CHANNEL 2                    M ¤F6   34567 $
RFTGW   1,B        TO END OF TRACK LOAD MODE - CHANNEL 1          L %F6   34567 $
RFTGW   2,B                                  - CHANNEL 2          L ¤F6   34567 $
RCY     1,B     READ CYLINDER - CHANNEL 1                         M %F@   34567 R
RCY     2,B                   - CHANNEL 2                         M ¤F@   34567 R
RCYW    1,B        LOAD MODE - CHANNEL 1                          L %F@   34567 R
RCYW    2,B                  - CHANNEL 2                          L ¤F@   34567 R
RCYO    1,B        OVERLAPPED - CHANNEL 1                         M @F@   34567 R
RCYO    2,B                   - CHANNEL 2                         M *F@   34567 R
RCYWO   1,B        LOAD MODE OVERLAPPED - CHANNEL 1               L @F@   34567 R
RCYWO   2,B                             - CHANNEL 2               L *F@   34567 R
RCYG    1,B        TO END OF CYLINDER - CHANNEL 1                 M %F@   34567 $
RCYG    2,B                           - CHANNEL 2                 M ¤F@   34567 $
RCYGW   1,B        TO END OF CYLINDER LOAD MODE - CHAN 1          L %F@   34567 $
RCYGW   2,B                                     - CHAN 2          L ¤F@   34567 $

PSC     1,B     PREVENT SEEK COMPLETE - CHANNEL 1                 M %F4   34567 W
PSC     2,B                           - CHANNEL 2                 M ¤F4   34567 W
PSCO    1,B        OVERLAPPED - CHANNEL 1                         M @F4   34567 R
PSCO    2,B                   - CHANNEL 2                         M *F4   34567 R

SAI     1,B     SET ACCESS INOPERATIVE - CHANNEL 1                M %F8   34567 R
SAI     2,B                            - CHANNEL 2                M ¤F8   34567 R
SAIO    1,B        OVERLAPPED - CHANNEL 1                         M @F8   34567 R
SAIO    2,B                   - CHANNEL 2                         M *F8   34567 R
REL     1,B     RELEASE - CHANNEL 1                               M %F9   34567 R
REL     2,B             - CHANNEL 2                               M ¤F9   34567 R
RELO    1,B        OVERLAPPED - CHANNEL 1                         M @F9   34567 R
RELO    2,B                   - CHANNEL 2                         M *F9   34567 R

   1009 DATA TRANSMISSION UNIT
RTD     1,B     READ - CHANNEL 1                                  M %D0   34567 R
RTD     2,B          - CHANNEL 2                                  M ¤D0   34567 R
RTDW    1,B        LOAD MODE - CHANNEL 1                          L %D0   34567 R
RTDW    2,B                  - CHANNEL 2                          L ¤D0   34567 R
```



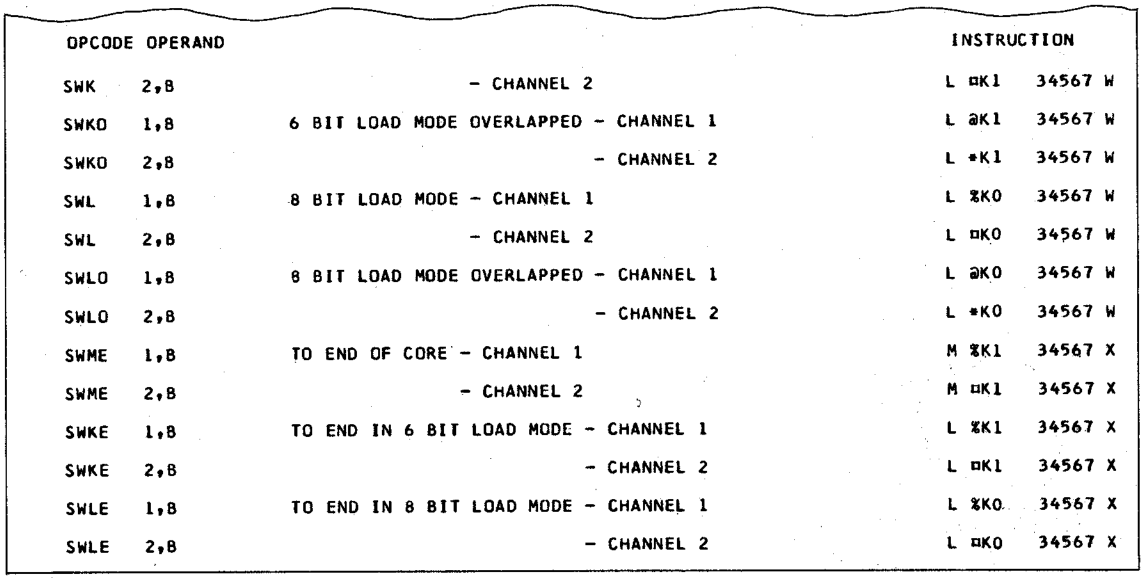
```
OPCODE OPERAND                                                                 INSTRUCTION

RTDO    1,B          OVERLAPPED - CHANNEL 1                                    M @DO    34567 R
RTDO    2,B                     - CHANNEL 2                                    M *DO    34567 R
RTDWO   1,B          LOAD MODE OVERLAPPED - CHANNEL 1                          L @DO    34567 R
RTDWO   2,B                               - CHANNEL 2                          L *DO    34567 R
WTD     1,B       WRITE - CHANNEL 1                                            M %DO    34567 W
WTD     2,B            - CHANNEL 2                                             M ¤DO    34567 W
WTDW    1,B          LOAD MODE - CHANNEL 1                                     L %DO    34567 W
WTDW    2,B                    - CHANNEL 2                                     L ¤DO    34567 W
WTDO    1,B          OVERLAPPED - CHANNEL 1                                    M @DO    34567 W
WTDO    2,B                     - CHANNEL 2                                    M *DO    34567 W
WTDWO   1,B          LOAD MODE OVERLAPPED - CHANNEL 1                          L @DO    34567 W
WTDWO   2,B                               - CHANNEL 2                          L *DO    34567 W

   1011 PAPER TAPE INSTRUCTIONS
RPT     1,B       READ - CHANNEL 1                                             M %P1    34567 R
RPT     2,B            - CHANNEL 2                                             M ¤P1    34567 R
RPTW    1,B          LOAD MODE - CHANNEL 1                                     L %P1    34567 R
RPTW    2,B                    - CHANNEL 2                                     L ¤P1    34567 R
RPTO    1,B          OVERLAPPED - CHANNEL 1                                    M @P1    34567 R
RPTO    2,B                     - CHANNEL 2                                    M *P1    34567 R
RPTWO   1,B          LOAD MODE OVERLAPPED - CHANNEL 1                          L @P1    34567 R
RPTWO   2,B                               - CHANNEL 2                          L *P1    34567 R

   1014 REMOTE INQUIRY INSTRUCTIONS
RQ      10,B      READ INQUIRY - CHANNEL 1                                     M %QO    34567 R
RQ      20,B                   - CHANNEL 2                                     M ¤QO    34567 R
RQW     11,B         LOAD MODE - CHANNEL 1                                     L %Q1    34567 R
RQW     21,B                   - CHANNEL 2                                     L ¤Q1    34567 R
RQO     10,B         OVERLAPPED - CHANNEL 1                                    M @QO    34567 R
RQO     20,B                    - CHANNEL 2                                    M *QO    34567 R
RQWO    11,B         LOAD MODE - CHANNEL 1                                     L @Q1    34567 R
RQWO    21,B                   - CHANNEL 2                                     L *Q1    34567 R
WQ      10,B      WRITE INQUIRY - CHANNEL 1                                    M %QO    34567 W
WQ      20,B                    - CHANNEL 2                                    M ¤QO    34567 W
WQW     11,B         LOAD MODE - CHANNEL 1                                     L %Q1    34567 W
WQW     21,B                   - CHANNEL 2                                     L ¤Q1    34567 W
WQO     10,B         OVERLAPPED - CHANNEL 1                                    M @QO    34567 W
WQO     20,B                    - CHANNEL 2                                    M *QO    34567 W
WQWO    11,B         LOAD MODE OVERLAPPED - CHANNEL 1                          L @Q1    34567 W
WQWO    21,B                              - CHANNEL 2                          L *Q1    34567 W

   TELEGRAPH UNITS
RL      10,B      READ TELEGRAPH UNIT - CHANNEL 1                              M %LO    34567 R
RL      20,B                          - CHANNEL 2                              M ¤LO    34567 R
RLO     11,B         OVERLAPPED - CHANNEL 1                                    M @L1    34567 R
RLO     21,B                    - CHANNEL 2                                    M *L1    34567 R
```



```
OPCODE OPERAND                                                        INSTRUCTION
WL      10,B     WRITE TELEGRAPH UNIT - CHANNEL 1                     M %L0    34567 W
WL      20,B                          - CHANNEL 2                     M □L0    34567 W
WLO     11,B         OVERLAPPED - CHANNEL 1                           M @L1    34567 W
WLO     21,B                    - CHANNEL 2                           M *L1    34567 W

   1412 MAGNETIC CHARACTER READER


RCR     1,B      READ CHARACTER READER - CHANNEL 1                    M %S1    34567 R
RCR     2,B                            - CHANNEL 2                    M □S2    34567 R
RCRW    1,B          LOAD MODE - CHANNEL 1                            L %S1    34567 R
RCRW    2,B                    - CHANNEL 2                            L □S2    34567 R
RCRO    1,B          OVERLAPPED - CHANNEL 1                           M @S1    34567 R
RCRO    2,B                     - CHANNEL 2                           M *S2    34567 R
RCRWO   1,B          LOAD MODE OVERLAPPED - CHANNEL 1                 L @S1    34567 R
RCRWO   2,B                               - CHANNEL 2                 L *S2    34567 R


ECR1             ENGAGE MAGNETIC CHARACTER READER - CHAN 1            P E
ECR2                                              - CHAN 2            Q E


DCR1             DISENGAGE MAGNETIC CHARACTER READER -CHAN 1          P D
DCR2                                                 -CHAN 2          Q D


SS1     R        MCR STACKER SELECT, POCKET R - CHANNEL 1             P R
SS2     3                           POCKET 3 - CHANNEL 2              Q 3

BCLR1   I        BRANCH LATE READ INDICATOR ON - CHANNEL 1            I 56789 1
BCLR2   I                                      - CHANNEL 2            0 56789 1
BCNR1   I            NOT READY INDICATOR ON - CHANNEL 1               I 56789 2
BCNR2   I                                   - CHANNEL 2               0 56789 2
BCRC1   I            CHECK INDICATOR ON - CHANNEL 1                   I 56789 3
BCRC2   I                               - CHANNEL 2                   0 56789 3
BCAF1   I            AMOUNT FIELD INDICATOR ON - CHANNEL 1            I 56789 4
BCAF2   I                                      - CHANNEL 2            0 56789 4
BCPC1   I            PROCESS CONTROL FIELD IND ON - CHAN 1            I 56789 5
BCPC2   I                                         - CHAN 2            0 56789 5
BCAN1   I            ACCOUNT NUMBER INDICATOR ON - CHAN 1             I 56789 6
BCAN2   I                                        - CHAN 2             0 56789 6
BCTR1   I            TRANSIT ROUTING FIELD  IND ON - CHAN 1           I 56789 7
BCTR2   I                                          - CHAN 2           0 56789 7
BCDC1   I            DOCUMENT SPACING CHECK IND ON - CHAN 1           I 56789 8
BCDC2   I                                          - CHAN 2           0 56789 8

   7750 TRANSMISSION CONTROL INSTRUCTIONS


SCM     1,B      CONTROL - CHANNEL 1                                  M %K1    34567 C
SCM     2,B              - CHANNEL 2                                  M □K1    34567 C
```



```
OPCODE OPERAND                                                              INSTRUCTION

SCMO    1,B          OVERLAPPED - CHANNEL 1                                 M @K1    34567 C
SCMO    2,B                     - CHANNEL 2                                 M *K1    34567 C
SCK     1,B          6 BIT LOAD MODE - CHANNEL 1                            L %K1    34567 C
SCK     2,B                          - CHANNEL 2                            L ¤K1    34567 C
SCKO    1,B          6 BIT LOAD MODE OVERLAPPED - CHANNEL 1                 L @K1    34567 C
SCKO    1,B                                     - CHANNEL 2                 L @K1    34567 C
SCL     1,B          8 BIT LOAD MODE - CHANNEL 1                            L %K0    34567 C
SCL     2,B                                                                 L ¤K0    34567 C
SCLO    1,B          8 BIT LOAD MODE OVERLAPPED - CHANNEL 1                 L @K0    34567 C
SCLO    2,B                                     - CHANNEL 2                 L *K0    34567 C
SSM     1,B      SENSE - CHANNEL 1                                          M %K1    34567 S
SSM     2,B            - CHANNEL 2                                          M ¤K1    34567 S
SSMO    1,B          OVERLAPPED - CHANNEL 1                                 M @K1    34567 S
SSMO    2,B                     - CHANNEL 2                                 M *K1    34567 S
SSK     1,B          6 BIT LOAD MODE - CHANNEL 1                            L %K1    34567 S
SSK     2,B                          - CHANNEL 2                            L ¤K1    34567 S
SSKO    1,B          6 BIT LOAD MODE OVERLAPPED - CHANNEL 1                 L @K1    34567 S
SSKO    2,B                                     - CHANNEL 2                 L *K1    34567 S
SSL     1,B          8 BIT LOAD MODE - CHANNEL 1                            L %K0    34567 S
SSL     2,B                          - CHANNEL 2                            L ¤K0    34567 S
SSLO    1,B          8 BIT LOAD MODE OVERLAPPED - CHANNEL 1                 L @K0    34567 S
SSLO    2,B                                     - CHANNEL 2                 L *K0    34567 S
SRM     1,B      READ - CHANNEL 1                                           M %K1    34567 R
SRM     2,B           - CHANNEL 2                                           M ¤K1    34567 R
SRMO    1,B          OVERLAPPED - CHANNEL 1                                 M @K1    34567 R
SRMO    2,B                     - CHANNEL 2                                 M *K1    34567 R
SRK     1,B          6 BIT LOAD MODE - CHANNEL 1                            L %K1    34567 R
SRK     2,B                          - CHANNEL 2                            L ¤K1    34567 R
SRKO    1,B          6 BIT LOAD MODE OVERLAPPED - CHANNEL 1                 L @K1    34567 R
SRKO    2,B                                     - CHANNEL 2                 L *K1    34567 R
SRL     1,B          8 BIT LOAD MODE - CHANNEL 1                            L %K0    34567 R
SRL     2,B                          - CHANNEL 2                            L ¤K0    34567 R
SRLO    1,B          8 BIT LOAD MODE OVERLAPPED - CHANNEL 1                 L @K0    34567 R
SRLO    2,B                                     - CHANNEL 2                 L *K0    34567 R
SRMG    1,B          TO END OF RECORD - CHANNEL 1                           M %K1    34567 $
SRMG    2,B                           - CHANNEL 2                           M ¤K1    34567 $
SRKG    1,B          TO END IN 6BIT LOAD MODE - CHANNEL 1                   L %K1    34567 $
SRKG    2,B                                   - CHANNEL 2                   L ¤K1    34567 $
SRLG    1,B          TO END IN 8BIT LOAD MODE - CHANNEL 1                   L %K0    34567 $
SRLG    2,B                                   - CHANNEL 2                   L ¤K0    34567 $
SWM     1,B      WRITE - CHANNEL 1                                          M %K1    34567 W
SWM     2,B            - CHANNEL 2                                          M ¤K1    34567 W
SWMO    1,B          OVERLAPPED - CHANNEL 1                                 M @K1    34567 W
SWMO    2,B                     - CHANNEL 2                                 M *K1    34567 W
SWK     1,B          6 BIT LOAD MODE - CHANNEL 1                            L %K1    34567 W
```



```
OPCODE OPERAND                                                        INSTRUCTION

SWK     2,B                    - CHANNEL 2                          L ¤K1   34567 W
SWKO    1,B   6 BIT LOAD MODE OVERLAPPED - CHANNEL 1                L @K1   34567 W
SWKO    2,B                              - CHANNEL 2                L *K1   34567 W
SWL     1,B   8 BIT LOAD MODE - CHANNEL 1                           L %K0   34567 W
SWL     2,B                   - CHANNEL 2                           L ¤K0   34567 W
SWLO    1,B   8 BIT LOAD MODE OVERLAPPED - CHANNEL 1                L @K0   34567 W
SWLO    2,B                              - CHANNEL 2                L *K0   34567 W
SWME    1,B   TO END OF CORE - CHANNEL 1                            M %K1   34567 X
SWME    2,B                  - CHANNEL 2                            M ¤K1   34567 X
SWKE    1,B   TO END IN 6 BIT LOAD MODE - CHANNEL 1                 L %K1   34567 X
SWKE    2,B                             - CHANNEL 2                 L ¤K1   34567 X
SWLE    1,B   TO END IN 8 BIT LOAD MODE - CHANNEL 1                 L %K0   34567 X
SWLE    2,B                             - CHANNEL 2                 L ¤K0   34567 X
```

Index

Printed in U.S.A. C28-0309

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York