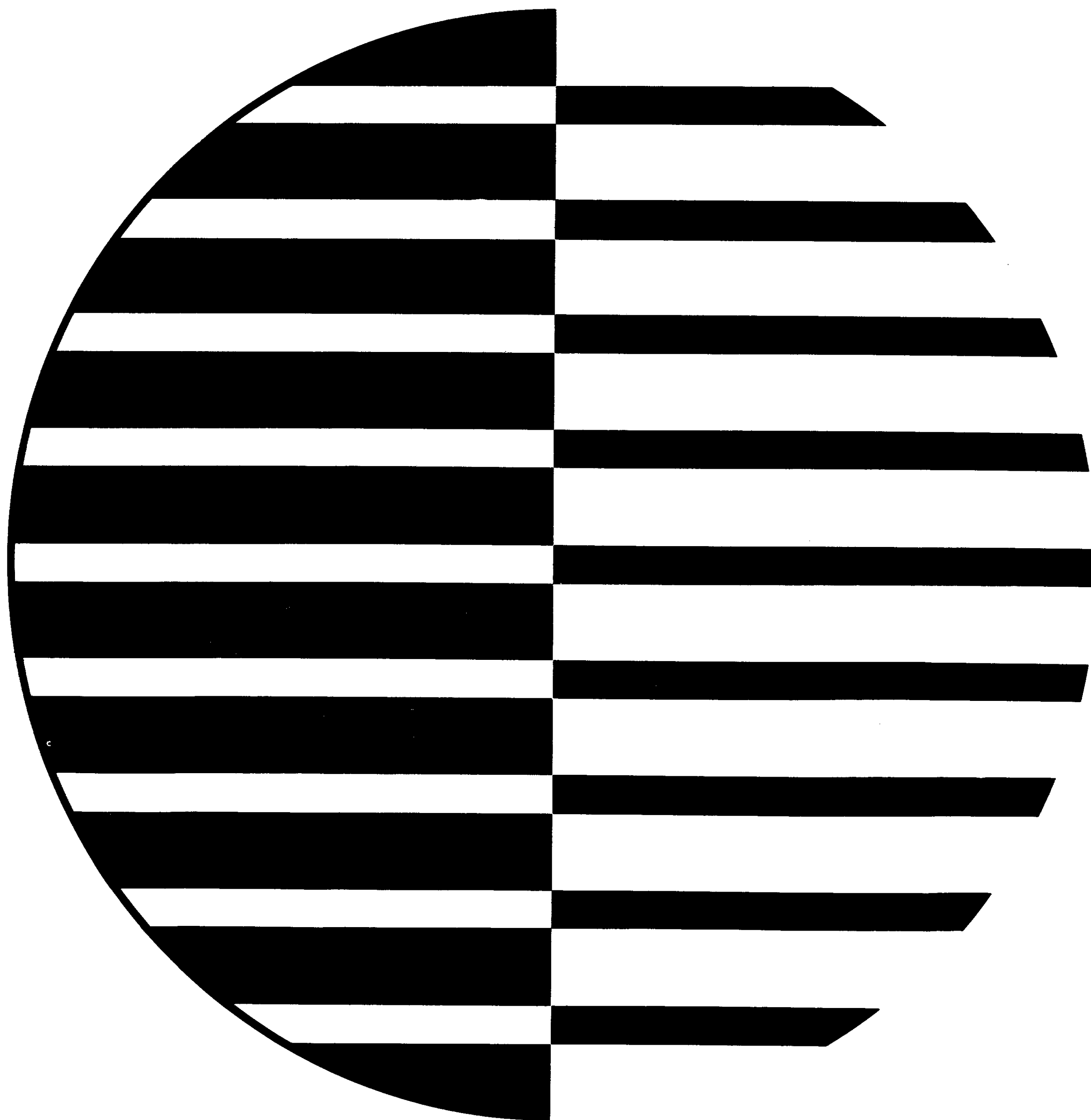
CONTROL DATA® 6400/6600 COMPUTER SYSTEMS

ASCENT/ASPER Reference Manual



Additional copies of this manual may be obtained from the nearest Control Data Corporation Sales office listed on the back cover.

July, 1966
Pub. No. 60172700

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

©1966, Control Data Corporation
Printed in the United States of America

CONTENTS

	INTRODUCTION	v
CHAPTER 1	THE ASCENT LANGUAGE	
	1.1 ASCENT Terms	1-1
	1.2 Language Specifications	1-6
	Instruction Fields	1-6
	Special Usages	1-8
	1.3 Central Processor Instructions	1-8
	Instruction Format	1-8
	Definitions	1-9
	Operating Registers	1-10
	Operation Codes	1-11
CHAPTER 2	THE ASPER LANGUAGE	
	2.1 ASPER Terms	2-1
	2.2 Instruction Fields	2-2
	2.3 Peripheral Processor Instructions	2-4
	Instruction Format	2-4
	Address Modes	2-5
	Operation Codes	2-6
CHAPTER 3	ASCENT/ASPER PSEUDO OPERATIONS AND MACROS	
	3.1 ASCENT/ASPER Pseudo Operations	3-1
	3.2 Programmer-Defined Macros	3-5
	Definition	3-5
	Rules	3-6
	Examples	3-7
CHAPTER 4	ASSEMBLER ERRORS	
	4.1 Error Flags	4-1
	4.2 Fatal Messages	4-1

CHAPTER 5 COSY

5.1	Input	5-1
5.2	Output	5-1
5.3	Corrections	5-1
5.4	COSY Identifier	5-2
5.5	Example	5-3
5.6	Diagnostics	5-4
5.7	Deck Format	5-5

APPENDIX A ASCENT CONTROL CARD A-1

TABLE 1.	CENTRAL PROCESSOR OPERATION CODES	A-2
TABLE 2.	PERIPHERAL PROCESSOR OPERATION CODES	A-4
TABLE 3.	ASCENT PSEUDO OPERATION CODES	A-6
TABLE 4.	ASPER PSEUDO OPERATION CODES	A-7
TABLE 5.	6400/6600 CHARACTER CODES	A-8

INTRODUCTION

The Control Data® 6400/6600 Assembly System consists of two language processing programs: ASCENT (for Central Processor instructions) and ASPER (for Peripheral Processor instructions). This manual defines the ASCENT and ASPER languages, including programmer macros, assembler diagnostics, and a description of COSY for both assembly languages.

The central processor handles the computational load; central memory stores operational and system programs together with the data they require. The peripheral processors transfer from peripheral equipment into central memory the programs to be executed by the central processor, as well as all input data required at execute time. Similarly, peripheral processors transfer (from central memory) output data generated by the central programs to peripheral equipment.

ASCENT and ASPER programs may be intermixed within an assembly, even though the individual instructions may not be mixed within a subprogram. To preserve processing efficiency, each of the two subsystems has a direct (though not exclusive) path for its own type of instruction.

This document represents version 1.1 of ASCENT/ASPER.

THE ASCENT LANGUAGE 1

1.1 ASCENT TERMS

The following terminology is used in ASCENT.

CHARACTERS

The character set comprises the letters A-Z, the digits 0-9, and the special characters + - / * = ( ) . , $ blank

SYMBOLS

A symbol is any arrangement of letters and digits which starts with a letter and contains no more than 7 characters.

Examples: T, PROG, ABSC123

Exceptions: A0,A1,...A7 are used for address registers.

B0,B2,...B7 are used for increment registers.

X0,X2,...X7 are used for arithmetic and operand registers.

CONSTANTS

Constants may be any of the following:

Integer

Up to 18 decimal digits, from $-(2^{59}-1)$ to $(2^{59}-1)$.

Examples: 3, 8125, 123456789012345678

Octal

Up to 20 octal digits (0-7) terminating with the letter B. It functions exactly like an Integer.

Examples: 47B, 770077B, 25252525252525252525B

Single-precision floating point

Up to 15 decimal digits, in the following forms:

With a single decimal point.

Examples: 1., .1, 0.1, 1.0, 5248.6153

With or without a decimal point, followed by EN or E±n, where n specifies the power of 10 to be applied to the constant. The sign may be omitted if it is positive; n may have up to 3 decimal digits.

Examples: 1E+5, 1.0E+250, .1E-30, 5248.6153E7, 14E51

Double-precision floating point

Up to 29 decimal digits, with or without a decimal point, followed by D or Dn or D±n, where n specifies the power of 10 to be applied to the constant. The sign of n may be omitted when positive; n may have up to 3 decimal digits.

Complex constant

Any pair of single-precision floating point constants separated by a comma and enclosed in parentheses. For example: (1.0,-2.2), (1E+5,.001E-15).

The computer treats a complex constant simply as a pair of constants to be stored in consecutive locations. When 1.0,-2.2 is coded on a constant card, 1.0 and -2.2 will be stored in consecutive locations. Parentheses have the effect of telling the assembler the constants should be single-precision floating point.

SPECIAL CHARACTERS

In the following instances only, special characters form part of a constant:

Decimal point in floating point numbers

Comma which unites the two parts of a complex number

Parentheses enclosing a complex constant or indicating a literal (the only functions of parentheses in ASCENT)

OPERATORS

Operators + - * / are used in three ways:

1. Between quantities to be combined at assembly time to form an 18-bit address field.

   Examples:
   ```
   SA1  10+1
   SA1  12-1
   SA1  2*5+1
   SA1  20+13/3
   ```

2. Between register names, indicating arithmetic functions to be carried out at run time; in such an instruction, an operator qualifies the operation code.

   Examples:
   ```
   FX1  X1+X2
   FX1  X1-X2
   FX1  X1*X2
   FX1  X1/X2
   ```

3. Between a register name and a quantity, when a plus or minus sign indicates that the quantity is to be added to or subtracted from the content of the register at run time.

   Examples:
   ```
   SA3  X4+365
   JP   B2+HOME
   ```

SEPARATORS

Normal separators are comma, blank, and equal sign; they are treated identically by the assembler, except in the location field. Conventionally, blanks are used between the operation code and address fields, and a comma is used as a separator within the address field.

```
Example:      EQ B1,B2,ALASKA
```

Any number of separators may divide the operation code from the address but once the address field has begun, two consecutive separators will terminate the assembler scan; anything further to the right on the card will be interpreted as a remark.

```
Examples:     EQ B1,B2,ALASKA   SHREWD
              EQ B1,B2,ALASKA,,SHREWD
```

A period or dollar sign anywhere in the location, operation code, or address field terminates the scan.

```
Examples:     .THIS IS A COMMENT CARD
                  NO     .THIS IS A NO-OP
                  EQ     B1,B2,ALASKA.SHREWD
```

LITERALS

Literals are used for addressing a core location the contents of which are specified by the value within parentheses. Literals may be any of the following forms:

(Constant)

(Symbol)

(Symbol±I) — I is an integer, octal or symbolic constant (a symbol that does not represent a relocatable address)

(Symbol - Symbol)

When a two-word form is defined, the first word is addressed:

(Floating Double-Precision Constant)

(Complex Constant)

Examples: (3.2), (SAM), (SAM+5), (770.,5.1E31), (3.4D70)

A second pair of parentheses is not used when a complex constant becomes a complex literal.

OPERANDS

Operands combine operators, symbols, literals, and constants in the following format:

OP TERM OP TERM OP TERM . . .

OP is + - * or /

TERM is * (current address), symbol, literal, or constant

Evaluating an Operand in the Address Field

A register is set to 0 and the leftmost OP and TERM are applied to the register. (If no leading OP appears, + is assumed; a leading * indicates the address of this location.) Each successive pair of OP and TERM is then applied to the register. Thus, 21 + 12/3 is evaluated as 11, not as 25.

The resultant address is relocatable if:

- the operand field contained only one relocatable symbol (A literal is a relocatable symbol.) and
- the operators * and / were not used, and
- the instruction is not LXi jk, AXi jk, or MXi jk.

Otherwise, the operand is absolute.

Symbolic Constant

A symbol constant is a symbol in the LOC field defined by the pseudo operation EQU to take the value of an absolute operand.

Examples:

TAG	EQU	20	(If A is relocatable, X4X is also relocatable and cannot be called a symbolic constant; otherwise TAG, X4X, SUE and JOE are all symbolic constants.)
X4X	EQU	A+100B	
SUE	EQU	X4X/2	
JOE	EQU	B*512+C	

(A,B, and C must be previously defined.)

INSTRUCTION FORMAT

The following fields constitute an instruction:

Location	Operation Code	Address	Remarks
Provides a symbol for referencing by other instructions	Defines the operation, wholly or partially	Supplies the operands and/or completes the operation code	Contains programmer notes only. This field has no effect on the assembly process

1.2 LANGUAGE SPECIFICATIONS

The fields of the basic ASCENT instruction are described in the following sections.

1.2.1 INSTRUCTION FIELDS

LOCATION FIELD

Columns 2-9 constitute the location field. If the field is not blank, it must contain one of the following:

+ in any column, with blanks in the other columns. The assembler puts the instruction at the beginning of a 60-bit machine word, automatically inserting no-operation instructions to fill out the previous word if necessary.

- in any column, with blanks in the others, is significant only if the preceding instruction is JP, RJ, or PS. Normally the assembler fills any unused part of JP, RJ, or PS machine words with no-operation instructions so that the next instruction begins a new machine word; however, if the next instruction is coded with a minus sign in the field, it is assembled into the same word with the JP, RJ, or PS if there is room.

Symbol anywhere in the field (with blanks in the unused columns). It is equivalent to the address of the instruction or constant occupying the rest of the card. The assembler inserts one or more no-operation instructions before a labeled instruction when necessary to position it at the beginning of a machine word.

. or $ in any column, with preceding blanks, indicates a comment card.

No symbol may appear in the location field of more than one card in a single program. No symbol that occurs in the address field of a COMMON or EXT card may occur in a location field in the same program. However, a symbol that occurred in the location field of a COMMON card could appear in the location field of another card; these two pseudo operations are exceptional in several respects (see 3.1).

OPERATION CODE FIELD

The operation code field length is variable; it starts in or after column 11 and is terminated by one or more separators (normally blanks). The field may contain any of the following:

6400/6600 central processor mnemonic codes (Table 1, Appendix)

ASCENT pseudo codes (Table 3, Appendix)

Name of any programmer-defined macro

Numeric operation codes are not legal.

ADDRESS FIELD

The address field length is variable; its content varies with the instruction. Two consecutive separators terminate the field.

REMARKS FIELD

The remarks field starts after two consecutive separators, or after a period or dollar sign (except a period on a CON card or between parentheses, which is assumed to be a decimal point). If the first non-blank character on a card is a period, the entire card contains remarks. In a BCD or DPC card (3.1), the remarks field begins after the last character to be assembled. If the address field is blank, the remarks field must begin with a period.

OTHER COLUMNS

Only columns 1-72 are considered by ASCENT. Column 1 is either a blank or a C for comment card. Column 10 is a blank and serves as a separator between location and operation code fields. ASCENT ignores column 10 except to flag an error if non-blank.

1.2.2 SPECIAL USAGES

Ascent Forcing Convention

Following a PS, JP, or RJ instruction, ASCENT forces the next instruction to the upper portion of the next 60-bit word, unless there is a minus sign in the location field of that instruction. A plus sign in the location field of any instruction will cause it to be forced upper.

Asterisk

The asterisk, when used as an operand or part of an operand, assumes the value of the current object code address. The forms are:

*

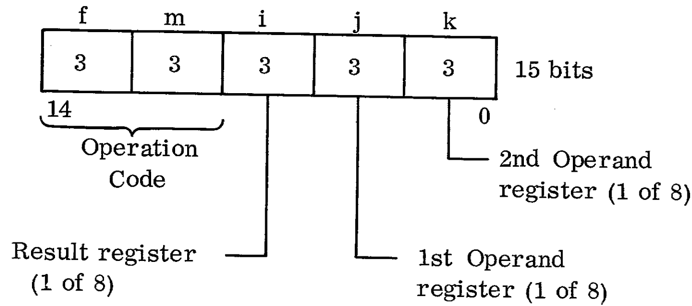
* operator constant

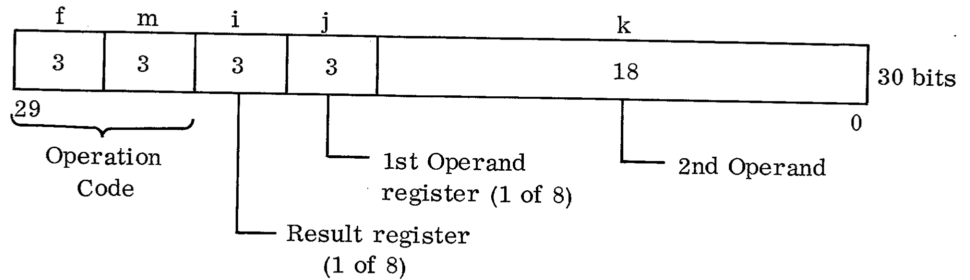
Examples:	SAi	*	Load current object code word
	JP	*+3	Jump forward three 60-bit words

1.3 CENTRAL PROCESSOR INSTRUCTIONS

1.3.1 INSTRUCTION FORMAT

Instructions may be 15 bits or 30 bits. Either format uses a 6-bit operation code. The result register requires 3 bits; the number of bits used for the operand varies with the instruction.





1.3.2 DEFINITIONS

The parameters used in the instructions are defined as follows:

fm	Operation code (6 bits)
i	Specifies the result register or the X register condition for a branch (3 bits)
j	Specifies the first operand register (3 bits)
k	Specifies the second operand register (3 bits)
jk	Constant, indicating number of shifts (6 bits)
K	Constant, indicating branch destination or second operand (18 bits)
A	One of eight 18-bit address registers
B	One of eight 18-bit increment registers
X	One of eight 60-bit operand registers

1.3.3 OPERATING REGISTERS

The 24 operating registers are identified by letters and digits:

A0,A1,...A7	Address registers
B0,B1,...B7	Increment registers
X0,X1,...X7	Operand registers

A Register

The execution of a SAi (i = 1-5) instruction produces an immediate memory reference to the address contained in Ai and reads the contents of that location into the corresponding operand register Xi (i = 1-5). When a SAi (i = 6 or 7) instruction is executed the contents of the corresponding X register is stored at the address specified by Ai. The address register A0 is used for temporary storage; the execution of a SA0 instruction does not affect X0.

Examples: SA3 A4+10

Adds 10 to the address in A4 and sets the A3 register to this sum. The X3 register is set to the contents of the location specified by the new A3.

SA6 A2-15

Stores the contents of X6 into the location obtained by subtracting 15 from the address in A2.

B Register

The increment register B0 is set permanently to an 18-bit positive zero which may be used in testing for zero or as an unconditional jump modifier. B1-B7 are used as modifiers and for program indexing. For example, B4 may be used to control the number of passes of a program loop, terminating when a given condition is reached.

Example: SB3 B5+B4

Adds the values contained in the two increment registers, B5 and B4, and places the result in B3.

X Register

Any of the registers X0-X7 may be used as a result or operand register. X1-X5 hold read operands from central memory; X6 and X7 hold results sent to central memory. The operand registers may be used and changed without causing a change in the corresponding address registers.

Examples: BX2 X2+X4

Performs the logical addition of X2 and X4 and places the resultant sum in X2.

SX6 A2-B5

Subtracts the contents of B5 from the contents of A2 and stores the result in X6.

1.3.4 OPERATION CODES

The instructions for the central processor are listed below. They are ordered by octal code which divides the instructions according to functional units. For each instruction, examples are given to show all the normal forms of address fields. In the examples, "K" represents what might be coded as any one of the following:

- One or more decimal or octal integers, symbolic constants, or ordinary symbols, connected by operators.
- External symbol.
- Common block segment name alone or followed by a plus sign and an integer or symbolic constant.
- Literal.

00 PS Program Stop

Stops the central processor at the current instruction. An exchange jump instruction is necessary to restart the central processor. PS is a 30-bit instruction.

e.g. PS

A comment after PS should begin with a period, otherwise it will look like an address field and may cause error flags.

01 RJ K — Return Jump to K

Stores an unconditional jump (04) and the current program address plus one in the upper 30 bits of K, and then branches to K + 1 for the next instruction. The contents of K after the instruction is executed appear as follows:

```
K      EQ      B0,B0,L+1
       PS      .
```

where L is the address of the executed RJ instruction

A jump to K at the end of the branch routine returns to the original programming sequence.

```
e.g.      RJ   K
```

02 JP Bi+K — Jump to Bi + K

Adds the contents of Bi to K and branches to the address specified by the resultant sum. When Bi = B0, the branch address is K. Addition is performed modulo $2^{18}-1$.

```
e.g.      JP  B2+K
          JP  K+B2
```

030 ZR Xj,K — Jump to K if Xj = 0

Branches to K if Xj is equal to zero. If the condition is not met, the next consecutive instruction step is executed. The test is made in the long add unit. Minus zero and plus zero both satisfy the test.

```
e.g.      ZR  X2,K
          ZR  K,X2
```

031 NZ Xj,K — Jump to K if Xj $\neq$ 0

Branches to K if Xj is not equal to zero. If the condition is not met, the next consecutive instruction step is executed. The test is made in the long add unit. Either plus zero or minus zero will fail the test.

```
e.g.      NZ  X2,K
          NZ  K,X2
```

032 PL Xj,K — Jump to K if Xj is Plus

Branches to K if Xj is positive. If the condition is not met, the next consecutive instruction step is executed.

```
e.g.     PL  X2,K
         PL  K,X2
```

033 NG Xj,K — Jump to K if Xj is Negative

Branches to K if Xj is negative. If the condition is not met, the next consecutive instruction step is executed.

```
e.g.     NG  X2,K
         NG  K,X2
```

034 IR Xj,K — Jump to K if Xj is In Range

Branches to K if Xj is in range, less than infinity ($377700\ldots0_8$).

```
e.g.     IR  X2,K
         IR  K,X2
```

035 OR Xj,K — Jump to K if Xj is Out of Range

Branches to K if Xj is out of range, greater than or equal to $377700\ldots0_8$.

```
e.g.     OR  X2,K
         OR  K,X2
```

036 DF Xj,K — Jump to K if Xj is Definite

Branches to K if Xj is definite. The test is a comparison against an indefinite quantity ($177700\ldots0_8$).

```
e.g.     DF  X2,K
         DF  K,X2
```

037 ID Xj, K Jump to K if Xj is Indefinite

Branches to K if Xj is indefinite. The test is a comparison against an indefinite quantity ($1777000\ldots0_8$).

e.g. ID X2, K
ID K, X2

04 EQ Bi, Bj, K Jump to K if Bi = Bj

Compares Bi with Bj and branches to K if Bi is equal to Bj. Minus zero is not equal to plus zero.

e.g. EQ B1, B2, K EQ B1, K, B2
EQ B2, B1, K EQ K, B1, B2

EQ K assembles as EQ B0, B0, K an unconditional jump.

04 ZR Bi, K Jump to K if Bi = B0

Compares Bi with B0 and branches to K if Bi is zero. Minus zero in Bi fails this test.

e.g. ZR B2, K
ZR K, B2 means EQ B0, B2, K

05 NE Bi, Bj, K Jump to K if Bi ≠ Bj

Compares Bi with Bj and branches to K if Bi is not equal to Bj. Minus zero is not equal to plus zero.

e.g. NE B1, B2, K
NE B2, B1, K
NE B2, K, B1
NE K, B1, B2

05 NZ Bi, K Jump to K if Bi ≠ B0

Compares Bi with B0 and branches to K if Bi is not zero. Minus zero in Bi passes this test.

e.g. NZ B2, K
NZ K, B2 means NE B0, B2, K

06 GE Bi, Bj, K — Jump to K if Bi ≥ Bj

Compares Bi with Bj and branches to K if Bi is greater than or equal to Bj. Plus zero is greater than minus zero.

```
e.g.    GE  B1, B2, K
        GE  B1, K, B2
        GE  K, B1, B2
```

06 PL Bi, K — Jump to K if Bi ≥ B0

Compares Bi with B0 and branches to K if the result is positive.

```
e.g.    PL  B1, K
        PL  K, B1
        means
        GE  B1, B0, K
```

07 LT Bi, Bj, K — Jump to K if Bi < Bj

Compares Bi with Bj and branches to K if Bi is less than Bj. Minus zero is less than plus zero.

```
e.g.    LT  B1, B2, K
        LT  B1, K, B2
        LT  K, B1, B2
```

07 NG Bi, K — Jump to K if Bi < B0

Compares Bi with B0 and branches to K if Bi is negative.

```
e.g.    NG  B1, K
        NG  K, B1
        means
        LT  B1, B0, K
```

10 BXi Xj — Transmit Xj to Xi

Transfers the 60-bit word in operand register Xj to Xi.

```
e.g.    BX2  X3
```

11 BXi Xj*Xk — Logical Product of Xj and Xk to Xi

Forms the logical product (AND function) of the 60-bit words in operand registers Xj and Xk and places the result in Xi.

```
Xj  0101
Xk  1100
    ----
Xi  0100
```

e.g. BX2 X3*X4

12 BXi Xj+Xk — Logical Sum of Xj and Xk to Xi

Forms the logical sum (inclusive OR) of the 60-bit words in operand registers Xj and Xk and places the result in Xi.

```
Xj  0101
Xk  1100
    ----
Xi  1101
```

e.g. BX2 X3+X4

13 BXi Xj-Xk — Logical Difference of Xj and Xk to Xi

Forms the logical difference (exclusive OR) of the 60-bit words in operand registers Xj and Xk and places the result in Xi.

```
Xj  0101
Xk  1100
    ----
Xi  1001
```

e.g. BX2 X3-X4

14 BXi -Xk — Transmit Xk Complement to Xi

Transmits the complement of the 60-bit word in operand register Xk to Xi. The contents of Xk are not changed.

e.g. BX2 -X3

15 BXi -Xk*Xj Logical Product of Xj and Xk Complement to Xi

Forms in Xi the logical product (AND function) of Xj and the complement of Xk. The contents of Xk and Xj are not changed.

Step 1	Xj	0101	Step 2	Xj	0101
	Xk	1100		-Xk	0011
				Xi	0001

e.g. BX2 -X3*X4

16 BXi -Xk+Xj Logical Sum of Xj and Xk Complement to Xi

Complements the 60-bit word in Xk, then forms the logical sum (inclusive OR) of this quantity and Xk and places the result in Xi. The contents of Xk and Xj are not changed.

Step 1	Xj	0101	Step 2	Xj	0101
	Xk	1100		-Xk	0011
				Xi	0111

e.g. BX2 -X3+X4

17 BXi -Xk-Xj Logical Difference of Xj and Xk Complement to Xi

Complements the 60-bit word in Xk, then forms the difference (exclusive OR) of this quantity and Xj, and places the result in Xi. The contents of Xk and Xj are not changed.

Step 1	Xj	0101	Step 2	Xj	0101
	Xk	1100		-Xk	0011
				Xi	0110

e.g. BX2 -X3-X4

20 LXi jk Shift Xi Left jk Places

Shifts the 60-bit word in Xi left circular jk places. Each step moves the leftmost bit of Xi into the rightmost position of Xi.

The 6-bit shift count jk is normally coded as an octal or decimal number. A complete circular shift of Xi is possible, when jk = 60.

e.g. LX2 36

21 AXi jk — Shift Xi Right jk Places

Shifts the 60-bit word in Xi right jk places. The rightmost bits of Xi are discarded and the sign bit is extended. The 6-bit shift count jk is normally coded as an octal or decimal number.

```
e.g.      AX2  36
```

22 LXi Bj,Xk — Left Shift Xk Nominally Bj Places to Xi

Shifts the 60-bit word in Xk the number of places specified by the low-order 6 bits of Bj and places the result in Xi.

If Bj is positive, Xk is shifted left circular.

If Bj is negative, Xk is shifted right (end off with sign extension).

When Bj is negative, the complement of the low-order 6 bits of Bj gives the number of places to be shifted.

```
e.g.      LX2  B1,X3
          LX2  X3,B1
```

23 AXi Bj,Xk — Arithmetic Right Shift Xk Nominally Bj Places to Xi

Shifts the 60-bit word in Xk the number of places specified by the low-order 6 bits of Bj and places the result in Xi.

If Bj is positive, Xk is shifted right (end off with sign extension).

If Bj is negative, Xk is shifted left circular.

When Bj is negative, the complement of the low-order 6 bits of Bj gives the number of places to be shifted.

```
e.g.      AX2  B3,X4
          AX2  X4,B3
```

24 NXi Bj,Xk — Normalize Xk in Xi and Bj

Normalizes the floating point quantity in Xk and places it in Xi. The number of left shifts required to normalize the quantity is placed in Bj during the operation. If the coefficient of Xk is zero, Xi is cleared to all zeros and Bj is set to 48. If the size of the exponent is less than the number of leading zeros in the coefficient of Xk, underflow occurs during normalizing and the exponent and coefficient of Xi are both cleared.

```
e.g.      NX2  B3,X4
          NX2  X4,B3
```

25 ZXi Bj, Xk — Round and Normalize Xk in Xi and Bj

Performs the same operation as NXi (24) except that the quantity in Xk is rounded before it is normalized. Normalizing a zero coefficient places the round bit in bit 47 and reduces the exponent by 48.

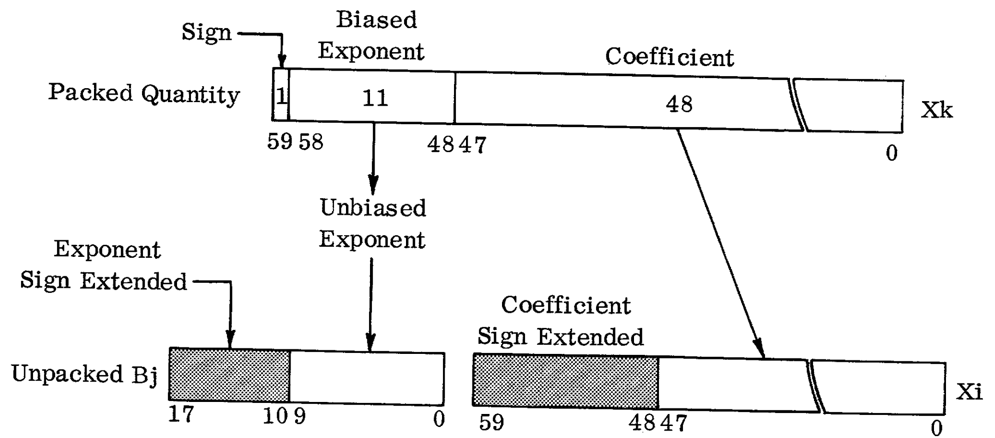
e.g. ZX2 B3, X4
      ZX2 X4, B3

26 UXi Bj, Xk — Unpack Xk to Xi and Bj

Unpacks the floating point quantity in Xk and sends the sign and 48-bit coefficient to Xi and the 11-bit exponent minus 2000_8 to Bj; then Bj contains the true 1's complement representation of the exponent. Xk may be an unnormalized number.

e.g. UX2 B3, X4
      UX2 X4, B3

The exponent and coefficient are sent to the low-order bits of the respective registers as shown in the following diagram.



27 PXi Bj, Xk — Pack Xi from Xk and Bj

Packs a floating point number in Xi. The coefficient of the number is obtained from the sign and low-order 48 bits of Xk and the exponent is obtained by adding 2000_8 to the low-order 11 bits of Bj. The coefficient is not normalized.

Exponent and coefficient are obtained from the low-order bits of the register and packed as shown in the diagram for the unpack (26) instruction. Overflow is produced during pack when Bj is a positive number of more than 10 bits; the overflow exit is optional. Underflow is produced (no exit) when Bj is a negative number of more than 10 bits.

```
e.g.      PX2  B3,X4
          PX2  X4,B3
```

30 FXi Xj+Xk — Floating Sum of Xj and Xk to Xi

Forms the sum of the floating point quantities in Xj and Xk and packs the result in Xi. The packed result is the upper half of a double precision sum.

Both arguments are unpacked, and the coefficient of the argument with the smaller exponent is entered into the upper half of a 96-bit accumulator. The coefficient is shifted right by the difference of the exponents. The other coefficient is then added into the upper half of the accumulator. If overflow occurs, the sum is shifted right one place, and the exponent of the result is increased by one. The upper half of the accumulator holds the coefficient of the sum, which is not necessarily in normalized form. The exponent and upper coefficient are then repacked in Xi.

If both exponents are zero and no overflow occurs, the instruction effects an ordinary integer addition.

```
e.g.      FX2  X3+X4
```

31 FXi Xj-Xk — Floating Difference of Xj and Xk to Xi

Forms the difference of the floating point quantities in Xj and Xk and packs the result in Xi. Alignment and overflow operations are similar to the floating sum (30) instruction, and the difference is not necessarily normalized. The packed result is the upper half of a double precision difference.

An ordinary integer subtraction is performed when the exponents are zero.

```
e.g.      FX2  X3-X4
```

32 DXi Xj+Xk — Floating DP Sum of Xj and Xk to Xi

Forms the sum of two floating point numbers as in the floating sum (30) instruction, but packs the lower half of the double precision sum with an exponent 48 less than the exponent of the upper sum.

e.g. DX2 X3+X4

33 DXi Xj-Xk — Floating DP Difference of Xj and Xk to Xi

Forms the difference of two floating point numbers as in the floating difference (31) instruction, but packs the lower half of the double precision difference with an exponent of 48 less than the exponent of the upper difference.

e.g. DX2 X3-X4

34 RXi Xj+Xk — Round Floating Sum of Xj and Xk to Xi

Forms the round sum of the floating point quantities in Xj and Xk and packs the upper sum of the double precision result in Xi. The sum is formed in the same manner as the floating sum (30) instruction except that the operands are rounded before the addition, as explained below, to produce a round sum.

A round bit is attached at the right end of both operands if both operands are normalized, or the operands have unlike signs.

For all other cases, a round bit is attached at the right end of the operand with the larger exponent.

e.g. RX2 X3+X4

35 RXi Xj-Xk — Round Floating Difference of Xj and Xk to Xi

Forms the round difference of the floating point quantities in Xj and Xk and packs the upper difference of the double precision result in Xi. The difference is formed in the same manner as the floating difference (31) instruction except operands are rounded before the subtraction, as explained below, to produce a round difference.

A round bit is attached at the right end of both operands if both operands are normalized, or the operands have like signs.

For all other cases, a round bit is attached at the right end of the operand with the larger exponent.

e.g. RX2 X3-X4

36 IXi Xj+Xk — Integer Sum of Xj and Xk to Xi

Forms a 60-bit one's complement sum of the quantities in Xj and Xk and stores the result in Xi. An overflow condition is ignored.

e.g. IX2 X3+X4

37 IXi Xj-Xk — Integer Difference of Xj and Xk to Xi

Forms the 60-bit one's complement difference of the quantities in Xj (minuend) and Xk (subtrahend) and stores the result in Xi.

e.g. IX2 X3-X4

40 FXi Xj*Xk — Floating Product of Xj and Xk to Xi

Multiplies the floating point quantities in Xj (multiplier) and Xk (multiplicand) and packs the upper product result in Xi.

The result is a normalized quantity only when both operands are normalized; the exponent is then the sum of the exponents plus 47 (or 48).

The result is unnormalized when either or both operands are unnormalized; the exponent is then the sum of the exponents plus 48.

e.g. FX2 X3*X4

41 RXi Xj*Xk — Round Floating Product of Xj and Xk to Xi

Attaches a round bit to the floating point number in Xk (multiplicand), multiplies this number by the floating point number in Xj, and packs the upper product result in Xi. (No lower product is available.)

The result is a normalized quantity only when both operands are normalized; the exponent is then the sum of the exponents plus 47 (or 48).

The result is unnormalized when either or both operands are unnormalized; the exponent is then the sum of the exponents plus 48.

e.g. RX2 X3*X4

42 DXi Xj*Xk Floating DP Product of Xj and Xk to Xi

Multiplies the floating point quantities in Xj and Xk and packs the lower product in Xi with an exponent 48 less than the exponent of the upper product. The result is not necessarily a normalized quantity.

e.g. DX2 X3*X4

43 MXi jk Form Mask in Xi, jk bits

Forms a mask in Xi. The 6-bit quantity jk defines the number of ones in the mask as counted from the highest order bit in Xi.

e.g. MX2 36

44 FXi Xj/Xk Floating Divide Xj by Xk to Xi

Divides the floating point quantities in Xj (dividend) by Xk (divisor) and packs the quotient in Xi. The exponent of the result in a no-overflow case is the difference of Xj and Xk exponents minus 48.

A one-bit overflow is compensated by shifting the coefficient right one place and increasing the exponent by one. The exponent is then the difference of Xj and Xk exponents minus 47.

The result is a normalized quantity when both Xj and Xk are normalized.

e.g. FX2 X3/X4

45 RXi Xj/Xk Round Floating Divide Xj by Xk to Xi

Divides the floating point quantity in Xj (dividend) by Xk (divisor) and packs the round quotient in Xi. A 1/3 round bit is added to the least significant bit of the dividend (Xj) before division starts. The result exponent in a no-overflow case is the difference of Xj and Xk exponents minus 48.

A one-bit overflow is compensated by shifting the coefficient right one place and increasing the exponent by one. The exponent is then the difference of Xj and Xk exponents minus 47.

The result is a normalized quantity when both Xj and Xk are normalized.

e.g. RX2 X3/X4

46 NO Pass

No operation. This is a 15-bit instruction. A comment on the same card should begin with a period; otherwise it will look like an address field and may cause an error flag.

47 CXi Xk Count the Number of 1's in Xk to Xi

Counts the number of ones in Xk and stores the count in Xi.

e.g. CX2 X3

50 SAi Aj+K

e.g. SA2 A2+K
SA2 K+A2
SA2 A3-K

51 SAi Bj+K

e.g. SA2 B3+K
SA2 K+B3
SA2 B2-K
SA2 K means SA2 B0+K

52 SAi Xj+K

e.g. SA2 X3+K
SA2 K+X3
SA2 X3-K

53 SAi Xj+Bk

e.g. SA2 X3+B4
SA2 B4+X3
SA2 X3 means SA2 X3+B0

54 SAi Aj+Bk

e.g. SA2 A3+B4
SA2 B4+A3
SA2 A3 means SA2 A3+B0

55 SAi Aj-Bk

e.g. SA2 A3-B4

56 SAi Bj+Bk

e.g. SA2 B3+B4
SA2 B3 means SA2 B3+B0

57 SAi Bj-Bk

e.g. SA2 B3-B4

These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in Ai.

Operands are obtained from address (A), increment (B), and operand (X) registers as well as the K portion of the instruction. K is an 18-bit signed constant. As used in instructions 50,51, and 52, if the sign of K is minus, ASCENT places the 18-bit one's complement of K in the K portion of the instruction word. Operands obtained from an X register are the truncated lower 18 bits of the 60-bit register.

An immediate memory reference to the address specified by the final contents of address register Ai is effected by the execution of a SAi (i = 1-7) instruction. The operand read from memory address specified by A1-A5 is sent to the corresponding operand register X1-X5. The operand from X6 or X7 is stored at the address specified by the corresponding A6 or A7.

60 SBi Aj+K

e.g. SB2 A3+K
SB2 K+A2
SB2 A3-K

61 SBi Bj+K

e.g. SB2 B3+K
SB2 K+B3
SB2 B3-K
SB2 K means SB2 B0+K

62 SBi Xj+K

```
e.g.    SB2  X3+K
        SB2  K+X3
        SB2  X3-K
```

63 SBi Xj+Bk

```
e.g.    SB2  X3+B4
        SB2  B4+X3
        SB2  X3 means SB2  X3+B0
```

64 SBi Aj+Bk

```
e.g.    SB2  A3+B4
        SB2  B4+A3
        SB2  A3 means SB2  A3+B0
```

65 SBi Aj-Bk

```
e.g.    SB2  A3-B4
```

66 SBi Bj+Bk

```
e.g.    SB2  B3+B4
        SB2  B3 means SB2  B3+B0
```

67 SBi Bj-Bk

```
e.g.    SB2  B3-B4
```

These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in Bi.

Operands are obtained from address (A), increment (B), and operand (X) registers as well as the K portion of the instruction. K is an 18-bit signed constant. As used in instructions 60,61 and 62, if the sign of K is minus, ASCENT places the 18-bit one's complement of K in the K portion of the instruction word. Operands obtained from an X register are the truncated lower 18 bits of the 60-bit register.

70 SXi Aj+K

e.g. SX2 A3+K
SX2 K+A2
SX2 A3-K

71 SXi Bj+K

e.g. SX2 B3+K
SX2 K+B3
SX2 B3-K
SX2 K means SX2 B0+K

72 SXi Xj+K

e.g. SX2 X3+K
SX2 K+X3
SX2 X3-K

73 SXi Xj+Bk

e.g. SX2 X3+B4
SX2 B4+X3
SX2 X3 means SX2 X3+B0

74 SXi Aj+Bk

e.g. SX2 A3+B4
SX2 B4+A3
SX2 A3 means SX2 A3+B0

75 SXi Aj-Bk

e.g. SX2 A3-B4

76 SXi Bj+Bk

e.g. SX2 B3+B4
SX2 B3 means SX2 B3+B0

77 SXi Bj-Bk

e.g. SX2 B3-B4

These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in Xi.

Operands are obtained from address (A), increment (B), and operand (X) registers as well as the K portion of the instruction. K is an 18-bit signed constant. As used in instructions 70,71, and 72, if the sign of K is minus, ASCENT places the 18-bit one's complement of K in the K portion of the instruction word.

Operands obtained from an Xj register are the truncated lower 18 bits of the 60-bit register. Conversely, an 18-bit result placed in Xi carries the sign bit extended to the remaining bits of the 60-bit register.

THE ASPER LANGUAGE 2

2.1 ASPER TERMS

The following terminology is used in ASPER.

CHARACTERS

The character set is identical to that of ASCENT: A-Z, 0-9, and special characters + - / * = ( ) . , $ blank.

SYMBOLS

As in ASCENT, a symbol is any arrangement of letters and digits up to 7, starting with a letter. But unlike the rule for ASCENT, A3 or X7 would be a valid symbol in ASPER.

Examples: T, PROG, ABCD123

CONSTANTS

Constants may be any of the following:

A decimal integer less than 2^{18}.

Up to 6 octal digits (0-7) terminating with the letter B.

Examples: 47B, 7770B, 140B

OPERATORS

Operators are used in address manipulations only:

+ addition
- subtraction
* multiplication
/ division

SEPARATORS

The $ blank . = and , are used as in ASCENT, with the single exception that in ASPER a . is used only to terminate the scan; there are no floating point constants in ASPER. $ or . terminates the scan; anything following to the right is a remark.

OPERANDS

An operand is a combination of operators, symbols, and constants terminated by a separator. A single operand must not contain embedded separators. The format is:

OP TERM OP TERM OP TERM

OP	is + - * /
TERM	is constant, symbol, or *

When the operand begins with a TERM, it is treated as if preceded by +.

* as a TERM indicates the address of a 1-word instruction; in a 2-word instruction, it indicates the address of the first word.

Literals are not allowed in ASPER.

INSTRUCTION FORMAT

The fields constituting an instruction are the same as in ASCENT:

Location   Operation Code   Address   Remarks

2.2 INSTRUCTION FIELDS

The fields of the basic ASPER instruction are described in this section.

LOCATION FIELD

The location field occupies columns 2-9. The field may be blank or contain a symbol of up to 7 characters ending on or before column 9. Symbols may not be duplicated in the location field within a program. The special character * may not appear in the location field.

OPERATION CODE FIELD

The operation code field length is variable; it starts in or after column 11 and is terminated by a separator (two if there is no address field). The field may contain:

- 6400/6600 peripheral processor mnemonic codes (Table 2, Appendix)
- ASPER pseudo codes (Table 4, Appendix)
- Programmer-defined macros

The 6-bit machine code represented by mnemonic becomes the left half of the first instruction word. Pseudo operations are interpreted and used in assembler sequence control. Numeric operation codes are illegal.

ADDRESS FIELD

The address field length is variable; its content varies with the type of instruction.

Operand (6-bit address)

No address (mnemonic ends in N)	Example: LPN 77B
Direct (mnemonic ends in D)	Example: STD ALOC
Indirect (mnemonic ends in I)	Example: STI ALOC

Instructions of this class require only one peripheral memory location. The address field is restricted to one operand. The operand evaluation must produce an octal equivalent $\leq 77_8$. This value becomes the right half of the instruction word.

Operand,Operand (12-bit address and 6-bit index designator)

Memory (mnemonic ends in M)	Example: LDM RECORD, ALOC

Instructions of this class require two peripheral memory locations; the first contains the 6-bit operation code and the index designator ($I \leq 77_8$). The second word contains the address portion evaluated to $\leq 2^{12}-1$. The index designator may be blank.

Example: LDM RECORD

Operand (18-bit address)

Constant (mnemonic ends in C) Example: LDC 776304B

Instructions of this class require two peripheral memory locations. The first word contains the 6-bit operation code and the high-order 6 bits of the operand. The low-order 12 bits are placed in the second word. The address field is restricted to one operand, which must produce an octal equivalent $\leq 2^{18}-1$ when evaluated.

REMARKS FIELD

The remarks field follows the first . or $ or the first occurrence of two consecutive separators (comma, blank, or equal) after the beginning of the address field.

OTHER COLUMNS

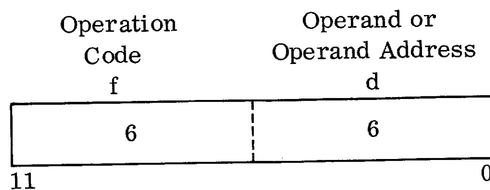
The blank column 10 separates the location and the operation code fields. Column 1 must contain blank or C; C indicates a comment card.

2.3 PERIPHERAL PROCESSOR INSTRUCTIONS

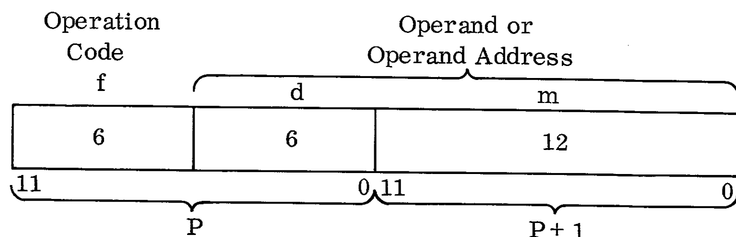
2.3.1 INSTRUCTION FORMAT

A PP instruction may be 12 bits or 24 bits to provide 6-bit or 18-bit operands and 6-bit, 12-bit or 18-bit addresses.

The 12-bit format has a 6-bit operation code, f, and a 6-bit operand or operand address, d.



The 24-bit format requires two memory words. The 6-bit quantity, d, of the first word is used with the 12-bit quantity, m, of the next consecutive word to form an 18-bit operand or operand address.



2.3.2 ADDRESS MODES

The usage of the quantities d and m varies with the addressing mode:

No address mode: d or dm is taken directly as an operand, eliminating the need for storing many constants. d is a 6-bit quantity $00\text{-}77_8$, but it may be considered as an 18-bit number with zero in the upper 12 bits. dm is an 18-bit quantity; d is the upper 6 bits and m the lower 12 bits.

Direct address mode: d or m + (d) is used as the operand address. d specifies one of the first 64 memory locations ($0000\text{-}0077_8$). m + (d) generates a 12-bit address for referencing all possible PP memory locations ($0000\text{-}7777_8$). If d = 0, m is taken as the operand address. If $d \neq 0$, the content of location d is added to m to produce an operand address (indexed direct addressing).

Indirect address mode: d specifies a location containing the address of the desired operand.

Indirect addressing and index direct addressing each require an additional memory reference.

Examples: $d = 25_8$ contents of loc. $25_8 = 0150_8$

$m = 100_8$ contents of loc. $150_8 = 7776_8$

contents of loc. $250_8 = 1234_8$

Mode	Instruction	Interpretation	A Register
No Address	LDN 25B	LDN d	000025
	LDC 250100B	LDC dm	250100
Direct Address	LDD 25B	LDD (d)	000150
	LDM 100B, 25B	LDM (m+(d))	001234
Indirect Address	LDI 25B	LDI ((d))	007776

Actual operation code formats are shown in Table 2, Appendix.

2.3.3 OPERATION CODES

00 PSN d — Pass

A no operation instruction. A comment after the operation should begin with a period; otherwise it looks like an address field and may cause an error.

01 LJM m+(d) — Long Jump

Jumps to sequence beginning at address m + (d). If d = 0, m is not modified.

02 RJM m+(d) — Return Jump

Stores the current program address plus two (P + 2) at location m + (d), and jumps to location m + (d) + 1.

03 UJN d — Unconditional Jump

Unconditional jump of up to 31 steps forward or backward from current program address, depending on value of d. Forward if d is positive ($01\text{-}37_8$). Backward if d is negative ($40\text{-}76_8$). Program stops when d equals 00 or 77.

04 ZJN d — Zero Jump

Conditional jump of up to 31 steps forward or backward from current program address if A register is zero. If A is nonzero, the next instruction is executed. Negative zero (777777) is treated as nonzero. See instruction 03 for an interpretation of d.

05 NJN d — Nonzero Jump

Conditional jump of up to 31 steps forward or backward from current program address if A register is nonzero. If A is zero, the next instruction is executed. Negative zero (777777) is treated as nonzero. See instruction 03 for an interpretation of d.

06 PJN d — Plus Jump

Conditional jump of up to 31 steps forward or backward from current program address if A register is positive. If A is negative, the next instruction is executed. See instruction 03 for an interpretation of d.

07 MJN d — Minus Jump

Conditional jump of up to 31 steps forward or backward from the current program address if A register is negative. If A is positive, the next instruction is executed. See instruction 03 for an interpretation of d.

10 SHN d — Shift

Shifts contents of A register right or left d places. If d is positive ($00\text{-}37_8$), shift is left circular; if d is negative (40-77), A is shifted right (end off with no sign extension). A left shift of 6 places results when d = 6 and a right shift of 6 places results when d = 71_8.

11 LMN d — Logical Difference

Forms in the A register the bit-by-bit logical difference of d and the lower 6 bits of A. This is equivalent to complementing the individual bits in A which correspond to one bits in d. The upper 12 bits of A are not altered.

```
A   001110101011001001
d               001010
    ------------------
    001110101011000011
```

12 LPN d — Logical Product

Forms in the A register the bit-by-bit logical product of d and the lower 6 bits of A. The upper 12 bits of A are zero.

```
A   001110101011001001
d               001010
    ------------------
    000000000000001000
```

13 SCN d — Selective Clear

Clears the lower 6 bits of the A register where corresponding bits of d are ones. The upper 12 bits of A are not altered.

```
A   001110101011001001
d               001010
    ------------------
    001110101011000001
```

14 LDN d Load

Clears the A register and loads d into the lower 6 bits of A. The upper 12 bits of A are zero.

15 LCN d Load Complement

Clears the A register and loads the complement of d into the lower 6 bits of A. The upper 12 bits of A are set to ones.

16 ADN d Add

Adds the 6-bit positive quantity d to the contents of the A register.

17 SBN d Subtract

Subtracts the 6-bit positive quantity d from the contents of the A register.

20 LDC dm Load

Clears the A register and loads the 18-bit quantity consisting of d as the upper 6 bits and m as the lower 12 bits.

21 ADC dm Add

Adds to the A register the 18-bit quantity consisting of d as the upper 6 bits and m as the lower 12 bits.

22 LPC dm Logical Product

Forms in the A register the bit-by-bit logical product of the contents of A and the 18-bit quantity dm.

```
 A   001110101011001001
dm   001110000011001010
     ------------------
     001110000011001000
```

23 LMC dm — Logical Difference

Forms in the A register the bit-by-bit logical difference of the contents of A and the 18-bit quantity dm. This is equivalent to complementing the individual bits in A which correspond to one bits in dm.

```
 A   001110101011001001
dm   000010000000001010
     ------------------
     001100101011000011
```

24 — Pass

A no operation instruction. There is no mnemonic for this.

25 — Pass

A no operation instruction. There is no mnemonic for this.

26 EXN d — Exchange Jump

Transmits an 18-bit address from the A register to the central processor and directs the central processor to perform an exchange jump, with the address in A as the starting location of a 16-word file containing information about the CP program to be executed. The 18-bit initial address must be entered in A before this instruction is executed. The central processor replaces the file with similar information from the interrupted CP program. The PP program is not interrupted.

27 RPN d — Read Program Address

Transfers contents of the central processor program address register to the peripheral processor A register to allow the PP to determine whether the central processor is running.

30 LDD (d) — Load

Clears the A register and loads the contents of location d into the lower 12 bits of A. The upper 6 bits of A are zero.

31 ADD (d) Add

Adds to the A register the 12-bit positive quantity in location d.

32 SBD (d) Subtract

Subtracts from the A register the 12-bit positive quantity in location d.

33 LMD (d) Logical Difference

Forms in the A register the bit-by-bit logical difference of the lower 12 bits of A and the contents of location d. This is equivalent to complementing individual bits of A which correspond to one bits in the contents of d. The upper 6 bits of A are not altered.

```
A   001110101011001001
d         010100001010
    ------------------
    001110111111000011
```

34 STD (d) Store

Stores the lower 12 bits of the A register into location d. The contents of A are not altered.

35 RAD (d) Replace Add

Adds the 12-bit quantity in location d to the contents of the A register and stores the lower 12 bits of the result back in location d. The result is also left in the A register at the end of the operation.

36 AOD (d) Replace Add One

Adds one to the original value in location d and stores the lower 12 bits of the result back in location d. The result is also left in the A register at the end of the operation.

37 SOD (d) Replace Subtract One

Subtracts one from the original value in location d and stores the lower 12 bits of the result back in location d. The result is also left in the A register at the end of the operation.

40 LDI ((d)) Load

Clears the A register and loads into A the 12-bit quantity obtained by indirect addressing. The upper 6 bits of A are zero.

41 ADI ((d)) Add

Adds to the contents of the A register a 12-bit positive operand obtained by indirect addressing.

42 SB I ((d)) Subtract

Subtracts from the A register a 12-bit positive operand obtained by indirect addressing.

43 LMI ((d)) Logical Difference

Forms in the A register the bit-by-bit logical difference of the lower 12 bits of A and the 12-bit operand obtained by indirect addressing. This is equivalent to complementing individual bits of A which correspond to one bits in the operand. The upper 6 bits of A are not altered.

```
     A   001110101011001001
 ((d))         010100001010
         ------------------
         001110111111000011
```

44 STI ((d)) Store

Stores the lower 12 bits of the A register into the location specified by the contents of d. The contents of A are not altered.

45 RAI ((d)) Replace Add

Adds A register contents to the operand obtained from the location specified by the contents of d. The resultant sum is left in the A register at the end of the operation and the lower 12 bits of A replace the original operand in memory.

46 AOI ((d)) Replace Add One

Adds one to the operand obtained from the location specified by the contents of d. The resultant sum is left in the A register at the end of the operation and the lower 12 bits of A replace the original operand in memory.

47 SOI ((d)) Replace Subtract One

Subtracts one from the operand obtained from the location specified by the contents of d. The resultant difference is left in the A register at the end of the operation and the lower 12 bits of A replaces the original operand in memory.

50 LDM (m+(d)) Load

Clears the A register and loads a 12-bit operand obtained by indexed direct addressing into the lower 12 bits of A. The upper 6 bits of A are zero. If d = 0, the operand address is m. If d ≠ 0, m plus the contents of location d is the operand address.

51 ADM (m+(d)) Add

Adds to the contents of the A register a 12-bit positive operand obtained by indexed direct addressing. (See instruction 50.)

52 SBM (m+(d)) Subtract

Subtracts from the A register a 12-bit positive operand obtained by indexed direct addressing. (See instruction 50.)

53 LMM (m+(d)) Logical Difference

Forms in the A register the bit-by-bit logical difference of the lower 12 bits of A and a 12-bit operand obtained by indexed direct addressing. This is equivalent to complementing individual bits of A which correspond to one bits in the operand. The upper 6 bits of A are not altered.

```
       A   001110101011001001
  (m+(d))        010100001010
           ------------------
           001110111111000011
```

54 STM (m+(d)) Store

Stores the lower 12 bits of the A register in the location determined by indexed direct addressing. The contents of A are not altered. (See instruction 50.)

55 RAM (m+(d)) Replace Add

Adds A register contents to the operand obtained from the location determined by indexed direct addressing. The resultant sum is left in the A register at the end of the operation and the lower 12 bits of A replace the original operand in memory. (See instruction 50.)

56 AOM (m+(d)) Replace Add One

Adds one to the operand obtained from the location determined by indexed direct addressing. The sum is left in the A register at the end of the operation and the lower 12 bits of A replace the original operand in memory. (See instruction 50.)

57 SOM (m+(d)) Replace Subtract One

Subtracts one from the operand obtained from the location determined by indexed direct addressing. The result is left in the A register at the end of the operation and the lower 12 bits of A replace the original operand in memory. (See instruction 50.)

60 CRD d Central Read from (A) to d

Transfers a 60-bit central memory word to 5 consecutive PP memory locations. The A register must contain the 18-bit absolute CM address before the instruction is executed. The 60-bit CM word is disassembled beginning at the left; d + 1, the next 12-bit word, etc. The A register contents are unchanged.

61 CRM m,d Central Read (d) words from (A) to m

Reads a block of 60-bit words from central memory into peripheral processor memory. The A register contains the 18-bit CM starting address and must be loaded prior to the execution of this instruction. The contents of A are increased by one as each 60-bit CM word is disassembled and stored. The block length or number of CM words to be read is contained in location d. The number also goes to the Q register where it is reduced by one as each CM word is processed. Transfer is complete when Q = 0.

The current contents of the P register are stored in PP location 0000, and the PP starting address m in the P register, which is increased by one as each 12-bit word is stored. Five words are required for each CM word read, since each CM word is disassembled into five successive PP words. The original contents of P are restored upon completion of the transfer.

62 CWD d — Central Write from d to (A)

Assembles five successive 12-bit words into a 60-bit word and stores it in central memory. The 18-bit CM address must be in the A register prior to the execution of the instruction, and it remains there unchanged afterwards.

The first word to be read out of PP memory is contained in location d; it appears as the leftmost 12 bits of the 60-bit word. The remaining 12-bit groups are taken from successive addresses in PP memory.

63 CWM m,d — Central Write (d) words from m to (A)

Assembles a block of 60-bit words and writes them in central memory. The A register contains the beginning central memory address and must be loaded prior to the execution of this instruction. The number in A is increased by one after each 60-bit word is assembled, to provide the next CM address.

The contents of location d specify the number of 60-bit words to write. The number also goes to the Q register where it is reduced by one as each CM word is assembled. Transfer is complete when Q = 0.

The original contents of the P register are stored in PP location 0000. The address of the first word to read from PP memory, m, goes to the P register which is increased by one as each 12-bit word is read to provide the next PP memory address. The original contents of the P register are restored at the completion of the transfer.

64 AJM m,d — Jump to m if channel d active

Conditional jump to a new program sequence beginning at address m if the channel specified by d is active. If the channel is inactive, the current program sequence continues.

65 IJM m,d — Jump to m if channel d inactive

Conditional jump to a new program sequence beginning at address m if the channel specified by d is inactive. If the channel is active, the current program sequence continues.

66 FJM m,d — Jump to m if channel d full

Conditional jump to a new program sequence beginning at address m if the channel specified by d is full. If the channel is empty, the current program sequence continues.

When the input equipment sends a word to the channel register and sets the full flag, the channel remains full until the PP accepts the word and clears the flag.

On output, the PP places a word in the channel register and sets the full flag. The channel is empty when the output equipment accepts the word and notifies the PP.

67 EJM m,d — Jump to m if channel d empty

Conditional jump to a new program sequence beginning at address m if the channel specified by d is empty. If the channel is full, the current program sequence continues.

70 IAN d — Input to A from channel d

Transfers a word from input channel d to the lower 12 bits of the A register. The upper 6 bits are cleared.

71 IAM m,d — Input (A) words from channel d to m

Transfers a block of words from input channel d to PP memory beginning at a location specified by m. The A register contains the block length; it is reduced by one as each word is read. The input operation is complete when A = 0.

The current contents of the P register are stored in PP location 0000 and the starting address, m, in P, which is increased by one as each word is stored to give the next address. The original contents of the P register are restored at the end of the operation.

72 OAN d — Output (A) on channel d

Transfers a word from the lower 12 bits of the A register to output channel d. The A register remains unaltered.

73 OAM m,d — Output (A) words from m on channel d

Transfers a block of words on output channel d from PP memory beginning at the location specified by m. The number of words is specified by the contents of the A register, which is reduced by one as each word is transferred. The output operation is completed when A = 0.

The current contents of the P register, m, are stored in PP location 0000. P is increased by one as each word is read to give the next address. The original contents of the P register are restored at the end of the operation.

74 ACN d — Activate channel d

Activates the channel specified by d. This instruction must precede instructions 70-73. Activating a channel alerts the I/O equipment for the exchange of data.

75 DCN d — Disconnect channel d

Deactivates the channel specified by d; stops the I/O equipment and terminates the buffer.

76 FAN d — Function (A) on channel d

Sends on channel d the external function code in the lower 12 bits of the A register.

77 FNC m,d — Function m on channel d

Sends on channel d the external function code specified by m.

ASCENT/ASPER PSEUDO OPERATIONS AND MACROS 3

3.1 ASCENT/ASPER PSEUDO OPERATIONS

The following ASCENT and ASPER pseudo operations apply to both languages except where noted.

ASCENT (ASCENT only)

Defines the beginning of a program and its name. Sets assembly mode to ASCENT. Must be the first instruction of an ASCENT routine.

END $v_1, \ldots, v_n$ (v_i separated by commas represent a variable number of symbols or integers in the location field.)

Indicates last card of assembly. If address field is non-blank, a main program is assumed and relocation bits are not punched in the Chippewa binary deck. Apart from this, v_i has no significance. In an ASPER program, it has none at all.

SUBRT (ASCENT only)

This should be used only as the next card after an ASCENT pseudo operation. The program containing it becomes a non-main program, so that relocation bits will be punched regardless of how the END card is coded.

EXT $v_1, v_2, \ldots, v_n$

Defines v_i as external symbols. Meaningful only if relocatable binary decks are punched.

Symbols used with EXT must be different from all normally defined symbols and from common block segment names.

ENTRY $v_1, v_2, \ldots, v_n$ (ASCENT only)

Defines v_i to be entry points. Meaningful only if relocatable binary decks are punched.

LIST v_1

Controls the side-by-side listing so that sections of coding may be omitted.

If $v_1 = 0$, print the listing

If $v_1 \neq 0$, suppress the listing

A LIST pseudo operation overrides permanently the choice of list or non-list mode on the ASCENT job control card. During assembly, a line that contains an error flag is always listed.

SPACE v_1

Spaces v_1 (1-63) lines on the listing.

EJECT

Ejects the listing to the top of the next page.

ASPER (ASPER only)

Defines the beginning of an ASPER program. It must be the first card of the program, except for comment cards.

MACRO name, $v_1, v_2, \ldots, v_n$

Indicates the start of a macro definition; name is the name associated with the macro. v_i are formal parameters. (See section 3.2.)

ENDM

Indicates the end of the macro definition.

IFF v_1, v_2, v_3

If $v_1 = 0$, assembles the next card if $v_2 \neq v_3$.

If $v_1 \neq 0$, assembles the next card if $v_2 = v_3$.

IFZ v_1, v_2

Assembles the following v_2 cards if the value of $v_1 = 0$.

IFN v_1, v_2

Assembles the following v_2 cards if the value of $v_1 \neq 0$.

REPLACE v_1, v_2

Replace cards from alter number v_1 to v_2. If v_2 is omitted, replace card v_1. All succeeding cards up to but not including the next REPLACE, DELETE, INSERT, or COSY, constitute the replacement.

DELETE v_1, v_2

Delete cards from alter number v_1 to v_2. If v_2 is omitted, delete card v_1.

INSERT v_1

Insert new cards after alter number v_1. All succeeding cards are inserted up to but not including the next INSERT, DELETE, REPLACE, or COSY.

COSY

Indicates the end of modifications and start of the COSY deck.

ORG v_1 (ASPER only)

Sets the location counter to the value of v_1.

ORGR v_1 (ASPER only)

Sets the location counter to the value of v_1.

LOC BSS v_1

Reserves the number of words specified by v_1 beginning at the next available location. The contents of the reserved locations are not set. The LOC symbol is equated to the address of the first word of the area. Any symbol appearing in the address field must be previously defined.

LOC BSSZ v_1

Same as BSS; the contents of the reserved locations are set to zero in the object code.

LOC EQU v_1

Assign the value v_1 to LOC. Any symbol appearing in the address field must be previously defined.

LOC DPC
or *Comment*
LOC BCD

Converts the characters enclosed by the asterisks (any number of characters) to display code or to BCD code, starting at the next available location. The last word, if incomplete, is padded with DPC or BCD blanks. The LOC symbol is equated to the address of the first word of the area. Characters between the asterisks have no effect on the assembler; for instance, a dollar sign does not terminate the scan. The comment field begins after the second asterisk.

LOC DPC
or nnComment
LOC BCD

Converts nn characters to display code or BCD code, beginning at the next available location (nn must be a two-digit decimal number). The last word, if incomplete, is padded with DPC or BCD blanks. The LOC symbol is equated to the address of the first word of the area. Characters following nn have no effect on the assembler; the comment field begins after the nn assembled characters.

LOC VFD (ASCENT only)

The VFD card generates a 60-bit word. Field specifications are:

Dnn/v_1 generates nn bits of display code. nn must be a multiple of 6, the first character must be alphabetic, and all other chacters must be letters or digits. If special characters are to be used, the use of Nnn/v_1 and the octal equivalent is necessary.

Nnn/v_2 generates nn bits as an integer; v_2 must be an integer, decimal or octal; it may be preceded by + or -.

Ann/v_1 generates nn bits as an address (if v_1 is relocatable, nn must be 18 and the 18-bit byte must be positioned in bits 0-17, 15-32, or 30-47). v_1 must be a single unmodified symbol.

The sum of all nn's must be ≤ 60. If less than 60, the result will be left justified with 0 fill.

Examples: VFD D30/INPUT, N12/0, A18/NAME

LOC COMMON $v_1, n_1, v_2, n_2, \ldots$ (ASCENT only)

Defines common block LOC and segments v_i of lengths n_i. v_i must be symbols; n_i must be integers. Meaningful only in relocatable binary decks. Blank common is defined by leaving the location field blank. A simple variable must be defined as a one-word segment and a multi-dimensioned array as a one-dimension segment.

LOC CON $v_1, v_2, \ldots$

In ASCENT, each of the v_i must be one of the following:

> Integer, octal constant, single or double precision floating point constant, complex constant, or operand. In this case, the operand is evaluated as an integer constant. A literal must not occur in such an operand.

In ASPER, each of the v_i must be acceptable as the first operand in an instruction with operation code LDM. It will be converted into the same 12-bit word as it would be in the LDM instruction.

The constants into which v_i are assembled will be assigned to consecutive memory locations.

3.2 PROGRAMMER-DEFINED MACROS

3.2.1 DEFINITION

Programmer-defined macros are defined by the programmer within an ASCENT or ASPER subprogram with a MACRO pseudo-instruction in the following form:

Location	OpCode	Address
blank	MACRO	symbol, list

MACRO is the pseudo operation code

symbol is the macro name

list means the formal parameters of the macro

Macros are called by writing the macro name in the operation code field, and the quantities, symbols, or register names to be substituted for the formal parameters in the address field.

3.2.2 RULES

The following rules apply to ASCENT and ASPER macros:

- The definition of a macro must precede the first executable instruction of the subprogram in which it is used.
- Programmer-defined macros are local to the subprogram in which the definition appears.
- A maximum of 100 macros is allowed per subprogram.
- Macros may be nested to any depth; they may be used in the definition of other macros, provided they too are defined prior to use. A macro may not be used in its own definition.
- Macro names may be any arrangement of letters and digits up to 7, starting with a letter.
- The macro name may not be identical to a machine mnemonic code, a pseudo code, or any other programmer-defined macro in the same subprogram.
- A maximum of 16 parameters is allowed in a macro parameter list.
- The order and count must be the same for formal and actual parameters.
- When the actual parameter is zero, a zero is inserted in the generated instruction if the formal parameter is in the address field; a blank is inserted if the formal parameter is in the location field.
- An ENDM pseudo operation must be the last instruction in the macro definition.
- Each location field between the MACRO and ENDM cards must contain blank, plus, minus, or a formal parameter; it must not contain a symbol which is to appear in the location field each time the macro is called; that could produce a multiple definition of the symbol.

3.2.3 EXAMPLES

ASCENT

```
Location    OpCode      Address

            MACRO       ABC,D,BA,AB,BN,RESULT,X
D           SA1         BA
            SAB         BN+X
            FX6         X1*X2
            SA6         RESULT
            ENDM

            MACRO       DEF,XA,AM,F,H,Z,L
            SAM         OP
K           ABC         E,B3,A3,Z,F,G
H           FX7         X6/XA
L           SA7         G
            ENDM
```

Using the definitions above, a macro call of

```
            DEF         X5,A5,LOC1,LOC2,U*V+Q-10,0
```

would generate the following set of instructions:

```
            SA5         OP
K           SA1         B3
            SA3         U*V+Q-12B+G
            FX6         X1*X2
            SA6         LOC1
LOC2        FX7         X6/X5
            SA7
```

ASPER

```
            MACRO       XYZ,OP,A,B,C
            LDM         A,B
            OP          C
            STM         A,B
            ENDM
```

Using the definition above, a macro call of

```
LOC         XYZ         SBC,D1,D2,D3
```

would generate the following set of instructions:

```
LOC         LDM         D1,D2
            SBD         D3
            STM         D1,D2
```

This definition is incorrect; NG is disregarded when the macro is called:

```
            MACRO       WICKED,AB,AC
            SA1         AB
NG          SA2         AC
            ENDM
```

ASSEMBLER ERRORS 4

4.1 ERROR FLAGS

The error codes that appear in the left margin of the listing are shown below:

O operation code error

U undefined symbol in the variable field

D doubly defined symbol in the variable field or location field

V address portion of VFD pseudo operation in error

R range error for ASPER jump instructions; in ASCENT, an operand with absolute value of not less than 2^{18}.

F field error in the variable field of a CON, BCD, or DPC card; or d greater than 63, or m greater than 4096 in ASPER instruction

L location field error

T literal table overflow

B symbol table overflow in ASPER assembly

Z program length exceeds 7777B in ASPER, or 177777B in ASCENT

S a symbol in the address field contains more than 7 characters

No flag but message in address field of BSS-BSSZ instruction, indicates undefined arguments in pass 1.

4.2 FATAL MESSAGES

The following messages indicate fatal assembler errors:

MACRO PARAMETER TABLE OVERFLOW

TOO MANY ASSEMBLER OPTIONS REQUESTED

ATTEMPT TO PUNCH DECK WITH NO ENTRY POINT

BREAKDOWN BUFFER AREA OVERFLOW

SYMBOL TABLE OVERFLOW

CHECKSUM ERROR ON COSY CARD

MACRO DICTIONARY OVERFLOW

MACRO SKELETON TABLE OVERFLOW

INSERT TABLE OVERFLOW

COMMON BLOCK TABLE OVERFLOW

ENTRY POINT TABLE OVERFLOW

EXTERNAL TABLE OVERFLOW

NOT ENOUGH FIELD LENGTH

5 COSY

The 6400/6600 assembler is capable of producing and assembling compressed symbolic (COSY) data. A COSY deck contains all information from a source deck reduced in size by a ratio of 5:1 to 10:1, depending on the number of comments.

5.1 INPUT

The COSY input file is specified on the ASCENT control card in the sixth parameter. Input is read from this file once a COSY identifier is encountered. The file name may be INPUT or any name other than SCR. If not specified it is assumed to be INPUT.

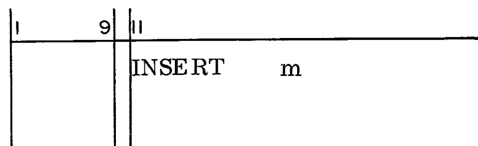
COSY correction decks containing alter cards and symbolic corrections to be included in the assembly are entered via the input file. Since ASCENT begins assembly for each subprogram by reading a BCD card image from the input file, the first card for each COSY subprogram must be one of the COSY instructions: INSERT, REPLACE, DELETE, or COSY.

5.2 OUTPUT

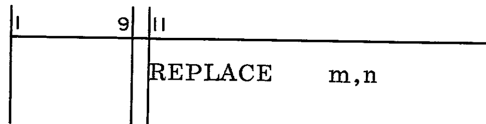
The assembler produces COSY output when the fourth parameter on the ASCENT control card is non-zero. This output will contain any COSY corrections included in the assembly. It is written on file P80C, which may be assigned to the punch, tape, or disk. The COSY edition number is increased by 1 on each COSY output.

5.3 CORRECTIONS

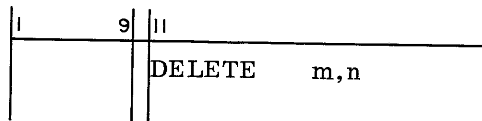
Corrections may be made to a COSY deck with the alter instructions INSERT, REPLACE, or DELETE. Each may be followed by a correction set consisting of symbolic instructions punched in the usual format. A correction set is terminated by another alter instruction or by a COSY identifier.



Succeeding symbolic instructions are inserted after card m, which is the sequence number of a symbolic card from the COSY deck.



Cards m through n are replaced by the symbolic instructions which follow. If n is omitted, only one card is replaced.

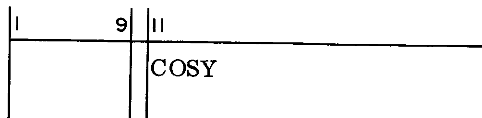


Cards m through n are deleted. A single card is deleted if only m is specified. If instructions follow DELETE, they will replace the deleted cards as in the REPLACE instruction.

References to symbolic sequence numbers by INSERT, REPLACE, or DELETE cards need not be in ascending order.

However, a reference on a COSY alter card must not conflict with other COSY alter cards, such as, DELETE 100 and INSERT 100 in the same deck, or DELETE 10,30 and DELETE 20,40 in the same deck.

5.4 COSY IDENTIFIER



This instruction signals the end of the COSY correction deck. Upon encountering this identifier, ASCENT begins reading COSY input. This card must appear if the assembly is from COSY input, even if there are no corrections.

5.5 EXAMPLE

The following deck is on tape in COSY format:

```
      ASPER   EXM             00001
      ORG     100B            00002

AX    LJM     0,0             00003
      LDN     0               00004
      STN     77B             00005

 B    LDI     77B             00006
      STM     20000B,77B      00007
      AOD     77B             00008
      SBN     77B             00009
      ZJN     AX              00010
      END                     00011
```

In the following job, corrections will be inserted, a new COSY deck and a binary deck punched, and a listing generated:

```
 1 2      9  11
 JOB,17,10,60000.
†ASSIGN CP,P80C.
 REQUEST COSY.
 REWIND(COSY)
 ASCENT(L,0,P,P,0,COSY)
                         (7,8,9 record separator)
             REPLACE        10
             NJN            B
             UJN            AX
             INSERT         3
  A          EQU            *-1
             REPLACE        5
             STD            77B
             COSY
                         (6,7,8,9 file separator)
```

† This card is no longer legal for version 1.1.

The listing from the assembly will appear as follows:

```
                                    REPLACE  10          CHANGE
                                    NJN      B           INSERTED
                                    UJN      AX          INSERTED
                                    INSERT   3           CHANGE
                               A    EQU      *-1         INSERTED
                                    REPLACE  5           CHANGE
                                    STD      77B         INSERTED

                                    ASPER    EXM         00001
                                    ORG      100B        00002
  0100     0100     0000       AX   LJM      0,0         00003
000101                         A    EQU      *-1         INSERTED
  0102     1400                     LDN      0           00004
  0103     3477                     STD      77B         INSERTED
  0104     4077                B    LDI      77B         00006
  0105     5477     2000            STM      2000B,77B   00007
  0107     3677                     AOD      77B         00008
  0110     1777                     SBN      77B         00009
  0111     0572                     NJN      B           INSERTED
  0112     0365                     UJN      AX          INSERTED
                                    END                  00011
```

5.6 DIAGNOSTICS

PRECEDING CHANGE OVERLAPPED-IGNORED

Erroneous correction set is not processed. Message appears on the listing.

Sequence numbers specified on a COSY correction card conflict with those of previous corrections.

CHECKSUM ERROR ON COSY CARD

Job is terminated. Message is placed in the dayfile.

This refers to the 30-bit checksum; a zero checksum is not checked.

INSERT TABLE OVERFLOW

Job is terminated. Message is placed in the dayfile.

The Insert Table is not large enough to contain all of the symbolic corrections.

5.7 DECK FORMAT

Columns	Rows	Code	Purpose
1	7,9	P	Deck identification
2	12,11,0-9	c	12-bit checksum inserted by peripheral punch program
3	12,11,0-3	E	6-bit edition number; appears only on first card of a COSY deck
	4-9	C	30-bit checksum; upper 6 bits
4-5	12,11,0-9	C	30-bit checksum; lower 24 bits
6-75	12,11,0-9	S	Compressed symbolic data: 14 words (60-bit)
76-80	12,11,0-9	H	5-digit COSY sequence number in Hollerith

COSY information consists of packed display codes representing BCD card images. In addition to the normal display characters (1-57), the following codes are used:

55 - 1 blank
64 - 2 blanks
65 - 3 blanks
66 - 4 blanks
67 - 5 blanks
70 - 6 blanks

71 - 7 blanks
72 - 8 blanks
73 - 9 blanks
74 - 10 blanks
75 - 20 blanks
76 - 30 blanks
77 - 40 blanks
00 - End of symbolic card image.

For example: 777455 represents 51 consecutive blanks.

Rows: 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Column 1: P (row 7), P (row 9)
Column 2: c
Column 3: E (row 0), C (row 6)
Columns 4–7: C C S S ...
Columns 75–80: ... S H H H H H

Columns: 1 2 3 4 5 6 7 ... 75 76 77 78 79 80

APPENDIX SECTION

ASCENT (l, x, pa, pc, pb, cosy)

l	if non-zero, a listing will be written on the output file.
x	inoperative
pa	if non-zero, binary deck will be punched.
pc	if non-zero, a COSY deck will be punched.
pb	inoperative
cosy	if zero or INPUT, COSY input is on INPUT file; otherwise this parameter is the name of the file from which COSY input is to be read.

Examples:

List only: ASCENT.
or ASCENT(L)

List and punch binary deck:
† ASSIGN CP, P80C.
ASCENT(L, 0, PA)

List and punch a COSY deck:
† ASSIGN CP, P80C.
ASCENT(L, 0, 0, PC)

List and write a COSY deck on tape 51:
ASSIGN 51, P80C.
ASCENT(L, 0, 0, PC)

Insert modifications into a COSY deck on tape 51, list, and punch a binary deck:
JOB.
† ASSIGN CP, P80C.
ASSIGN 51, COSY.
ASCENT(L, 0, PA, 0, 0, COSY)
(7, 8, 9 record separator)
Mod pack ending with COSY card
(6, 7, 8, 9 file separator)

† This card is no longer legal for version 1.1.

TABLE 1
CENTRAL PROCESSOR OPERATION CODES

Octal Op. Code	Mnemonic	Address	Comments
BRANCH UNIT			
00	PS		Program stop
01	RJ	K	Return jump to K
02	JP	Bi + K	Jump to Bi + K
030	ZR	Xj K	Jump to K if Xj = 0
031	NZ	Xj K	Jump to K if Xj ≠ 0
032	PL	Xj K	Jump to K if Xj = plus (positive)
033	NG	Xj K	Jump to K if Xj = negative
034	IR	Xj K	Jump to K if Xj is in range
035	OR	Xj K	Jump to K if Xj is out of range
036	DF	Xj K	Jump to K if Xj is definite
037	ID	Xj K	Jump to K if Xj is indefinite
04	EQ	Bi Bj K	Jump to K if Bi = Bj
04	ZR	Bi K	Jump to K if Bi = B0
05	NE	Bi Bj K	Jump to K if Bi ≠ Bj
05	NZ	Bi K	Jump to K if Bi ≠ B0
06	GE	Bi Bj K	Jump to K if Bi ≥ Bj
06	PL	Bi K	Jump to K if Bi ≥ B0
07	LT	Bi Bj K	Jump to K if Bi < Bj
07	NG	Bi K	Jump to K if Bi < B0
BOOLEAN UNIT			
10	BXi	Xj	Transmit Xj to Xi
11	BXi	Xj*Xk	Logical Product of Xj and Xk to Xi
12	BXi	Xj + Xk	Logical sum of Xj and Xk to Xi
13	BXi	Xj - Xk	Logical difference of Xj and Xk to Xi
14	BXi	- Xk	Transmit comp. of Xk to Xi
15	BXi	- Xk*Xj	Logical product of Xj and Xk comp. to Xi
16	BXi	- Xk + Xj	Logical sum of Xj and Xk comp. to Xi
17	BXi	- Xk - Xj	Logical difference of Xj and Xk comp. to Xi
SHIFT UNIT			
20	LXi	jk	Left shift Xi, jk places
21	AXi	jk	Arithmetic right shift Xi, jk places
22	LXi	Bj Xk	Left shift Xk nominally Bj places to Xi
23	AXi	Bj Xk	Arithmetic right shift Xk nominally Bj places to Xi
24	NXi	Bj Xk	Normalize Xk in Xi and Bj
25	ZXi	Bj Xk	Round and normalize Xk in Xi and Bj
26	UXi	Bj Xk	Unpack Xk to Xi and Bj
27	PXi	Bj Xk	Pack Xi from Xk and Bj
43	MXi	jk	Form mask in Xi, jk bits
ADD UNIT			
30	FXi	Xj + Xk	Floating sum of Xj and Xk to Xi
31	FXi	Xj - Xk	Floating difference Xj and Xk to Xi
32	DXi	Xj + Xk	Floating DP sum of Xj and Xk to Xi
33	DXi	Xj - Xk	Floating DP difference of Xj and Xk to Xi
34	RXi	Xj + Xk	Round floating sum of Xj and Xk to Xi
35	RXi	Xj - Xk	Round floating difference of Xj and Xk to Xi

Octal Op. Code	Mnemonic	Address	Comments
LONG ADD UNIT			
36	IXi	Xj + Xk	Integer sum of Xj and Xk to Xi
37	IXi	Xj - Xk	Integer difference of Xj and Xk to Xi
MULTIPLY UNIT			
40	FXi	Xj * Xk	Floating product of Xj and Xk to Xi
41	RXi	Xj * Xk	Round floating product of Xj and Xk to Xi
42	DXi	Xj * Xk	Floating DP product of Xj and Xk to Xi
DIVIDE UNIT			
44	FXi	Xj / Xk	Floating divide Xj by Xk to Xi
45	RXi	Xj / Xk	Round floating divide Xj by Xk to Xi
46	NO		No operation
47	CXi	Xk	Counts in Xi the number of 1's in Xk
INCREMENT UNIT			
50	SAi	Aj + K	Set Ai to Aj + K
50	SAi	Aj - K	Set Ai to Aj + comp. of K
51	SAi	Bj + K	Set Ai to Bj + K
51	SAi	Bj - K	Set Ai to Bj + comp. of K
52	SAi	Xj + K	Set Ai to Xj + K
52	SAi	Xj - K	Set Ai to Xj + comp. of K
53	SAi	Xj + Bk	Set Ai to Xj + Bk
54	SAi	Aj + Bk	Set Ai to Aj + Bk
55	SAi	Aj - Bk	Set Ai to Aj - Bk
56	SAi	Bj + Bk	Set Ai to Bj + Bk
57	SAi	Bj - Bk	Set Ai to Bj - Bk
60	SBi	Aj + K	Set Bi to Aj + K
60	SBi	Aj - K	Set B to Aj + comp. of K
61	SBi	Bj + K	Set Bi to Bj + K
61	SBi	Bj - K	Set Bi to Bj + comp. of K
62	SBi	Xj + K	Set Bi to Xj + K
62	SBi	Xj - K	Set Bi to Xj + comp. of K
63	SBi	Xj + Bk	Set Bi to Xj + Bk
64	SBi	Aj + Bk	Set Bi to Aj + Bk
65	SBi	Aj - Bk	Set Bi to Aj - Bk
66	SBi	Bj + Bk	Set Bi to Bj + Bk
67	SBi	Bj - Bk	Set Bi to Bj - Bk
70	SXi	Aj + K	Set Xi to Aj + K
70	SXi	Aj - K	Set Xi to Aj + comp. of K
71	SXi	Bj + K	Set Xi to Bj + K
71	SXi	Bj - K	Set Xi to Bj + comp. of K
72	SXi	Xj + K	Set Xi to Xj + K
72	SXi	Xj - K	Set Xi to Xj + comp. of K
73	SXi	Xj + Bk	Set Xi to Xj + Bk
74	SXi	Aj + Bk	Set Xi to Aj + Bk
75	SXi	Aj - Bk	Set Xi to Aj - Bk
76	SXi	Bj + Bk	Set Xi to Bj + Bk
77	SXi	Bj - Bk	Set Xi to Bj - Bk

TABLE 2
PERIPHERAL PROCESSOR
OPERATION CODES

Octal Op. Code	Mnemonic	Address	Comments
00	PSN		Pass
01	LJM	m,d	Long jump to m + (d)
02	RJM	m,d	Return jump to m + (d)
03	UJN	d	Unconditional jump d
04	ZJN	d	Zero jump d
05	NJN	d	Nonzero jump d
06	PJN	d	Plus jump d
07	MJN	d	Minus jump d
10	SHN	d	Shift d
11	LMN	d	Logical difference d
12	LPN	d	Logical product d
13	SCN	d	Selective clear d
14	LDN	d	Load d
15	LCN	d	Load complement d
16	ADN	d	Add d
17	SBN	d	Subtract d
20	LDC	dm	Load dm
21	ADC	dm	Add dm
22	LPC	dm	Logical product dm
23	LMC	dm	Logical difference dm
24			Pass
25			Pass
26	EXN		Exchange jump
27	RPN		Read program address
30	LDD	d	Load (d)
31	ADD	d	Add (d)
32	SBD	d	Subtract (d)
33	LMD	d	Logical difference (d)
34	STD	d	Store (d)
35	RAD	d	Replace add (d)
36	AOD	d	Replace add one (d)
37	SOD	d	Replace subtract one (d)
40	LDI	d	Load ((d))
41	ADI	d	Add ((d))
42	SBI	d	Subtract ((d))
43	LMI	d	Logical difference ((d))
44	STI	d	Store ((d))
45	RAI	d	Replace add ((d))
46	AOI	d	Replace add one ((d))
47	SOI	d	Replace subtract one ((d))
50	LDM	m,d	Load (m + (d))
51	ADM	m,d	Add (m + (d))

Octal Op. Code	Mnemonic	Address	Comments
52	SBM	m,d	Subtract (m + (d))
53	LMM	m,d	Logical difference (m + (d))
54	STM	m,d	Store (m + (d))
55	RAM	m,d	Replace add (m + (d))
56	AOM	m,d	Replace add one (m + (d))
57	SOM	m,d	Replace subtract one (m + (d))
60	CRD	d	Central read from (A) to d
61	CRM	m,d	Central read (d) words from (A) to m
62	CWD	d	Central write to (A) from d
63	CWM	m,d	Central write (d) words to (A) from m
64	AJM	m,d	Jump to m if channel d active
65	IJM	m,d	Jump to m if channel d inactive
66	FJM	m,d	Jump to m if channel d full
67	EJM	m,d	Jump to m if channel d empty
70	IAN	d	Input to A from channel d
71	IAM	m,d	Input (A) words to m from channel d
72	OAN	d	Output from A on channel d
73	OAM	m,d	Output (A) words from m on channel d
74	ACN	d	Activate channel d
75	DCN	d	Disconnect channel d
76	FAN	d	Function (A) on channel d
77	FNC	m,d	Function m on channel d

Notation

d	Implies d itself
(d)	Implies the contents of d
((d))	Implies the contents of the location specified by d
m	Implies m itself used as an address
m + (d)	The contents of d are added to m to form an operand (jump address)
(m + (d))	The contents of d are added to m to form the address of the operand
dm	Implies an 18-bit quantity with d as the upper 6 bits and m as the lower 12 bits

TABLE 3
ASCENT PSEUDO OPERATION CODES

Op. Code	Meaning
ASCENT	Defines CP program
END	Defines end of CP program
EXT	Defines external symbols
SUBRT	Results in relocation bits in Chippewa binary deck
ENTRY	Defines entry points
BSS	Reserves central memory region
BSSZ	Reserves central memory region and presets it to zero
EQU	Equates a symbol to a value
DPC	Inserts display-coded characters into program
BCD	Inserts BCD characters into program
CON	Defines constants in program
LIST	Controls listing
SPACE	Spaces listing
EJECT	Ejects page on listing
MACRO	Beginning of MACRO definition
ENDM	End of MACRO definition
IFF	Determine if next card is assembled
IFZ	Determine if following cards are assembled
IFN	Determine if following cards are assembled
REPLACE	Replace specified cards
DELETE	Deletes specified cards
INSERT	Inserts source cards
COSY	Indicates end of modifications; start of COSY deck
VFD	Generates a 60-bit word as specified
COMMON	Defines common block and arrays

TABLE 4
ASPER PSEUDO OPERATION CODES

Op. Code	Meaning
ASPER	Defines PP program
END	Defines end of PP program
ORG	Specifies starting address of PP program
ORGR	Specifies starting address of PP program
BSS	Reserves peripheral memory region
BSSZ	Reserves peripheral memory region and presets it to zero
EQU	Equates a symbol to a value
DPC	Inserts display-coded characters into program
BCD	Inserts BCD characters into program
CON	Constructs 12-bit constants
LIST	Controls side-by-side listing
SPACE	Spaces side-by-side listing
EJECT	Ejects page on side-by-side listing
MACRO	Beginning of a MACRO definition
ENDM	End of a MACRO definition
IFF	Determine if next card is assembled
IFN	Determine if following cards are assembled
IFZ	Determine if following cards are assembled
REPLACE	Replaces specified cards
DELETE	Deletes specified cards
INSERT	Inserts source cards
COSY	Indicates end of modifications; start of COSY deck

TABLE 5
6400/6600 CHARACTER CODES

Character	Display Code	Printer Code	Hollerith Punch Positions	Character	Display Code	Printer Code	Hollerith Punch Positions
A	01	61	12-1	8	43	10	8
B	02	62	12-2	9	44	11	9
C	03	63	12-3	+	45	60	12
D	04	64	12-4	-	46	40	11
E	05	65	12-5	*	47	54	11-8-4
F	06	66	12-6	/	50	21	0-1
G	07	67	12-7	(	51	34	0-8-4
H	10	70	12-8	)	52	74	12-8-4
I	11	71	12-9	$	53	53	11-8-3
J	12	41	12-9	=	54	13	8-3
K	13	42	11-2	blank	55	20	space
L	14	43	11-3	,	56	33	0-8-3
M	15	44	11-4	.	57	73	12-8-3
N	16	45	11-5	≠		14	8-4
O	17	46	11-6	:		00	
P	20	47	11-7	≤		15	
Q	21	50	11-8	%		16	
R	22	51	11-9	[		17	
S	23	22	0-2	]		32	
T	24	23	0-3	→		35	
U	25	24	0-4	≡		36	
V	26	25	0-5	∧		37	
W	27	26	0-6	∨		52	
X	30	27	0-7	↑		55	
Y	31	30	0-8	↓		56	
Z	32	31	0-9	>		57	
0	33	12	0	<		72	
1	34	01	1	≥		75	
2	35	02	2	¬		76	
3	36	03	3	;		77	
4	37	04	4				
5	40	05	5				
6	41	06	6				
7	42	07	7				

CONTROL DATA

CORPORATION

COMMENT AND EVALUATION SHEET

6400/6600 Computer Systems
ASCENT/ASPER Reference Manual

Pub. No. 60172700 July, 1966

THIS FORM IS NOT INTENDED TO BE USED AS AN ORDER BLANK. YOUR EVALUATION OF THIS MANUAL WILL BE WELCOMED BY CONTROL DATA CORPORATION. ANY ERRORS, SUGGESTED ADDITIONS OR DELETIONS, OR GENERAL COMMENTS MAY BE MADE BELOW. PLEASE INCLUDE PAGE NUMBER REFERENCE.

FROM NAME : ____________________

BUSINESS
ADDRESS : ____________________

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

FOLD FOLD

STAPLE STAPLE

CONTROL DATA SALES OFFICES

ALAMOGORDO, NEW MEXICO
ALBUQUERQUE, NEW MEXICO
ATLANTA, GEORGIA
AUSTIN, TEXAS
BILLINGS, MONTANA
BIRMINGHAM, ALABAMA
BOSTON, MASSACHUSETTS
BOULDER, COLORADO
CAPE CANAVERAL, FLORIDA
CEDAR RAPIDS, IOWA
CHICAGO, ILLINOIS
CINCINNATI, OHIO
CLEVELAND, OHIO
COLORADO SPRINGS, COLORADO
DALLAS, TEXAS
DAYTON, OHIO
DENVER, COLORADO
DETROIT, MICHIGAN
DOWNEY, CALIFORNIA
GREENSBORO, NORTH CAROLINA
HARTFORD, CONNECTICUT
HONOLULU, HAWAII
HOUSTON, TEXAS
HUNTSVILLE, ALABAMA
IDAHO FALLS, IDAHO
INDIANAPOLIS, INDIANA
KANSAS CITY, KANSAS
LAS VEGAS, NEVADA
LIVERMORE, CALIFORNIA
LOS ANGELES, CALIFORNIA
MADISON, WISCONSIN
MIAMI, FLORIDA
MILWAUKEE, WISCONSIN
MINNEAPOLIS, MINNESOTA
MONTEREY, CALIFORNIA
NEWARK, NEW JERSEY
NEW ORLEANS, LOUISIANA
NEW YORK, NEW YORK
OAKLAND, CALIFORNIA
OMAHA, NEBRASKA
PALO ALTO, CALIFORNIA
PHILADELPHIA, PENNSYLVANIA
PHOENIX, ARIZONA
PITTSBURGH, PENNSYLVANIA
PORTLAND, OREGON
ROCHESTER, NEW YORK
SACRAMENTO, CALIFORNIA
ST. LOUIS, MISSOURI
SALT LAKE CITY, UTAH
SAN BERNARDINO, CALIFORNIA
SAN DIEGO, CALIFORNIA
SAN FRANCISCO, CALIFORNIA
SAN JUAN, PUERTO RICO
SANTA BARBARA, CALIFORNIA
SEATTLE, WASHINGTON
TULSA, OKLAHOMA
VIRGINIA BEACH, VIRGINIA
WASHINGTON, D. C.

ADELAIDE, AUSTRALIA
AMERSFOORT, THE NETHERLANDS
AMSTERDAM, THE NETHERLANDS
ATHENS, GREECE
BOMBAY, INDIA
CALGARY, ALBERTA, CANADA
CANBERRA, AUSTRALIA
DUSSELDORF, GERMANY
FRANKFURT, GERMANY
GENEVA, SWITZERLAND
HAMBURG, GERMANY
JOHANNESBURG, SOUTH AFRICA
KASTRUP, DENMARK
LONDON, ENGLAND
LUCERNE, SWITZERLAND
MELBOURNE, AUSTRALIA
MEXICO CITY, MEXICO
MONTREAL, QUEBEC, CANADA
MUNICH, GERMANY
OSLO, NORWAY
OTTAWA, ONTARIO, CANADA
PARIS, FRANCE
ROME, ITALY
STOCKHOLM, SWEDEN
STUTTGART, GERMANY
SYDNEY, AUSTRALIA
TEHERAN, IRAN
TEL AVIV, ISRAEL
TOKYO, JAPAN (C. ITOH ELECTRONIC COMPUTING SERVICE CO. LTD.)
TORONTO, ONTARIO, CANADA
VANCOUVER, BRITISH COLUMBIA, CANADA
ZURICH, SWITZERLAND

8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440

CONTROL DATA
CORPORATION

Pub. No. 60172700

Litho in U.S.A.