Copy 432

pb 250

Programming Manual

PBC 1004 Revision 1

pb Packard Bell Computer

A SUBSIDIARY OF PACKARD BELL ELECTRONICS
1905 ARMACOST AVENUE • LOS ANGELES 25, CALIFORNIA • GRANITE 8-4247

March 15, 1961

NOTICE

This document involves confidential PROPRIETARY information of Packard Bell Computer Corporation and all design, manufacturing, reproductions, use, and sale rights regarding the same are expressly reserved. It is submitted under a confidential relationship for a specified purpose, and the recipient, by accepting this document assumes custody and control and agrees that (a) this document will not be copied or reproduced in whole or in part, nor its contents revealed in any manner or to any person except to meet the purpose for which it was delivered, and (b) any special features peculiar to this design will not be incorporated in other projects.

When this document is not further required for the specific purposes for which it was submitted, the recipient agrees to return it.

PREFACE

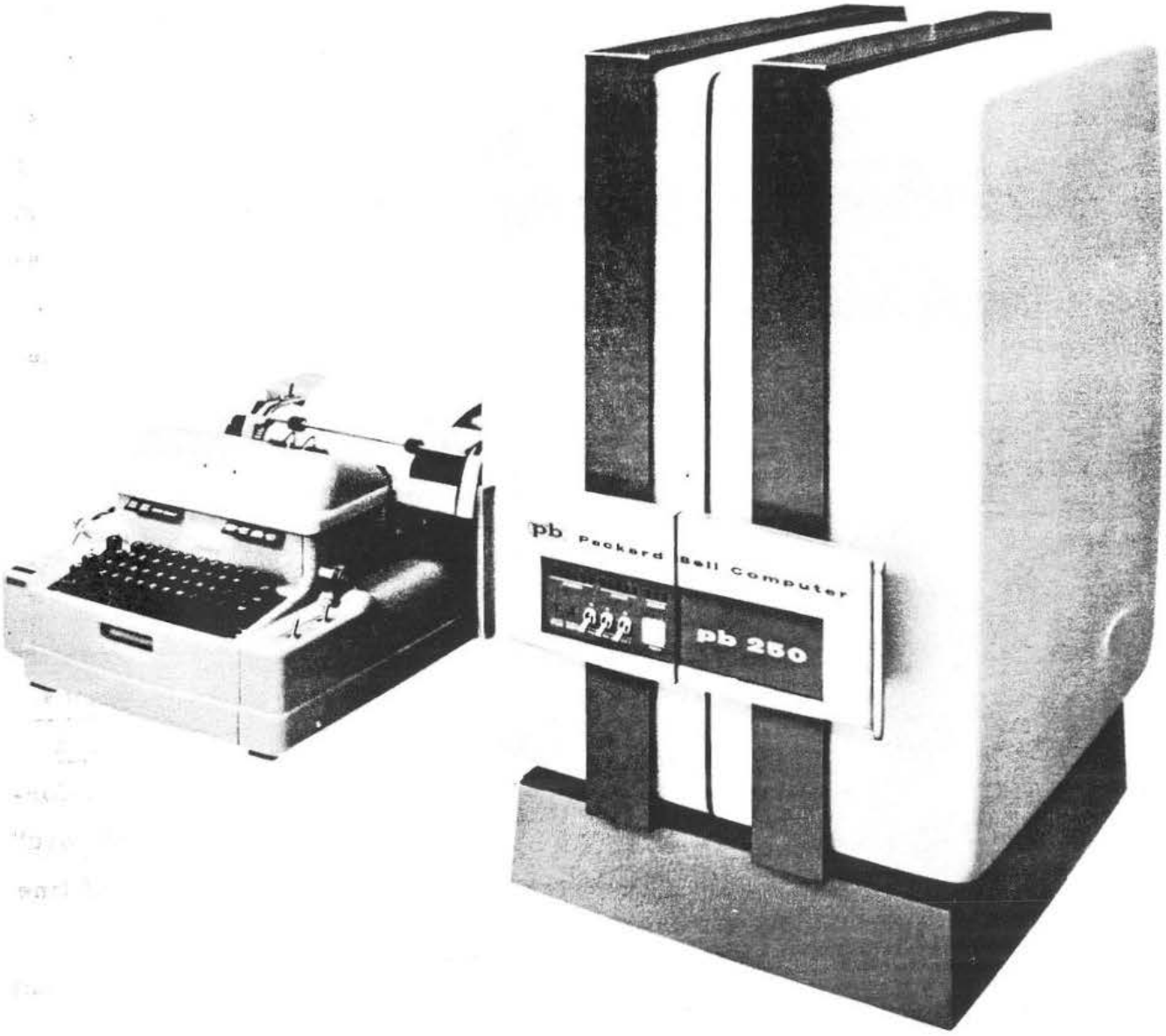
This manual is a guide to programming the PB250. Although a great deal of this material is similar to that which is included in the PB250 Reference Manual, it is presented here in more detail. The information provided in this manual will be useful in actual programming operations. Supplements and modifications to this manual will be published as a series of Programming Notes to be distributed to personnel possessing Programming Manuals.

CONTENTS

APPENDICES

ILLUSTRATIONS

TABLES



PB 250 General Purpose Digital Computer.

I. GENERAL PB 250 CHARACTERISTICS

1.1 INTRODUCTION

The Packard Bell PB 250 is a high-speed, completely solid-state general purpose digital computer in which both the data and the commands required for computation are stored in a homogenous memory. The storage medium is a group of nickel steel magnetostrictive lines along which acoustical pulses are propagated. At one end of each of these lines is a writing device for translating electrical energy into acoustical energy. At the other end of each line is a reading device for translating acoustical energy back into electrical signals. By rewriting the stored information as it is read, information continously circulates without alteration, except for alterations which result from the execution of the computer program. Use of the optional battery power supply will preserve memory information even during power interruptions.

1.2 MEMORY ORGANIZATION

The memory of the basic PB 250 contains ten lines, numbered octally (base eight) from 00 through 11, which may hold both data and instructions. Each long line, 01 through 11, contains 256 (decimal), or 400 (octal), locations, also called sectors, that are numbered 000 through 377. Note: All sector and line numbers are given in octal notation throughout this manual. Since the information in any location can be either data or a command, the generic term "word" is used to cover both. The location of any word is specified by a sector and line number (SSSLL), and these together are called an address. Line 00 is a 16-word Fast Access Line. Since line 00 is 1/16 the length of a long word line, any unit of information contained in it is available 16 times during each complete circulation of the 256-word lines. Any word in the Fast Access Line is identified by one of 16 channel addresses (see Recirculation Chart, Appendix D). Line 00

channels are designated F00 through F17. For example, channel F00 of the line 00 can be identified by the following addresses: 00000, 02000, 04000, 06000, ... 36000.

Fifty-three additional lines, each of which may have from one to 256 words, can be added. These lines are numbered 12 through 36, and 40 through 77. Line number 37 is used for the Index Register. If all of the additional lines are used, and if all hold 256 words, the memory capacity of the PB 250 is extended to 15,888 words. The PB 250 cabinet can hold a total of 16 lines.

Commands can be executed only from lines 00 through 17; these lines are therefore designated "Command Lines."

1.2.1 Data Word Configuration

Every number stored in the PB 250 is represented by a series of pulses which correspond to a series of zeros and ones that are the digits of the binary number system. The term "binary digit" is usually contracted to the word "bit." (A discussion of binary numbers may be found in Appendix A.)

A number stored in a location in the PB 250 consists of twenty-one bits that represent magnitude and a twenty-second bit to indicate sign. A negative number has a one in position zero, whereas a positive number has a zero in position zero. Negative numbers are expressed in their 2's complement form. (A discussion of complementary arithmetic may be found in Appendix A.) Figure 1-1 shows a PB 250 data word configuration.

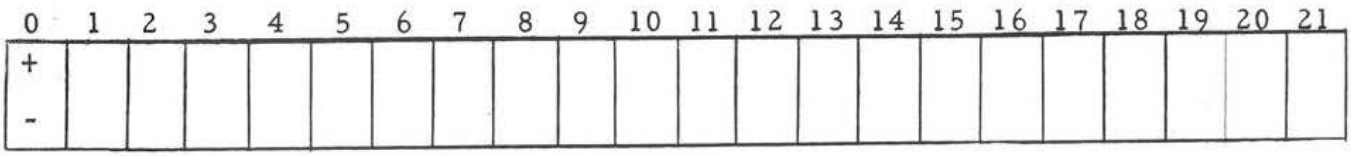


Figure 1-1. Data Word Configuration

These 22 positions are sufficient to represent a 6-digit decimal number.

Larger numbers may easily be represented by using the double precision features of the computer.

1.2.2 Arithmetic Registers

Three arithmetic registers, A, B, and C, are provided for arithmetic operations and information manipulation. Each register has exactly the same format as a memory location, including the sign, and all are available to the programmer. Double precision commands treat A and B as a double-length register. The contents of a register may be tested for non-positive values or compared against the contents of any memory location. In addition, information may be interchanged between A, B, and C. A record may be kept in one register of operations performed on the others.

1.2.3 Index And Buffer Registers

Both the Index and Buffer registers are part of special one-word registers. When loading the A, B, or C registers from either the Index or Buffer registers, suitable masking should be employed to avoid reading extraneous information.

1.2.3.1 Index Register

The Index register, which is part of the machine Instruction register (see Figure 1-2), stores a line number for use with commands which have an Index Tag of one. When used, the contents of the Index register replace the line number of the address in the command. This replacement is made during the reading of the command, but does not change the command as its stands in memory. For example, if the contents of the Index register are 01, then in the execution of the following program step:

OP Code	Address	Index Tag
ADD	03204	1

The contents of 03201, instead of the contents of 03204, will be added to the contents of the A register.

Line number 37 is reserved to designate the Index register. Addresses 00037 through 37737 all apply to this register, and bit position 16 through 21 are the useful positions for the line address. Thus, a STA into line 37, any sector, places bits 16 through 21 of A into the Index register, bits 16 through 21.

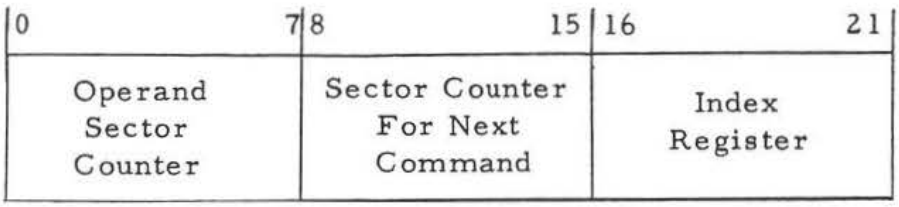


Figure 1-2. Index Register

The term "effective address," as used in this manual, means the actual location referred to by the computer when executing a command. In the event that the Index register is used, the effective address consists of the sector address specified by the command, plus the line address stored in the Index register, which replaces the line address of the command.

1.2.3.2 Input Buffer

The Input Buffer is part of the machine Sector Counter (see Figure 1-3). It receives the input from the Flexowriter and can accept up to an eight-bit character. This entry is logically accumulative for each bit of the character, requiring that the buffer be cleared before each input. The Input Buffer is enabled to accept information by either a READ TYPEWRITER KEYBOARD or a READ PAPER TAPE command. The single character sent by the reader, or provided by the depressed typewriter key, is loaded into

the buffer and, upon completion of buffer loading, the computer is signaled by the Flexowriter. This action requires a period of time during which it is possible to execute a large number of commands.

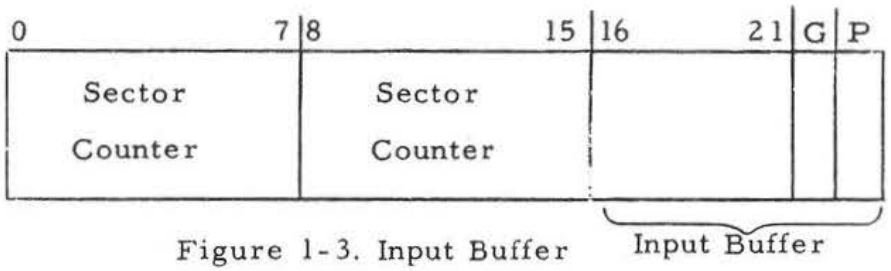


Figure 1-3. Input Buffer

1.3 COMMAND WORD CONFIGURATION

As previously described, information in any memory location may be either data or a command. When the information is a command, it has a definite configuration, or format, as illustrated in Figure 1-4.

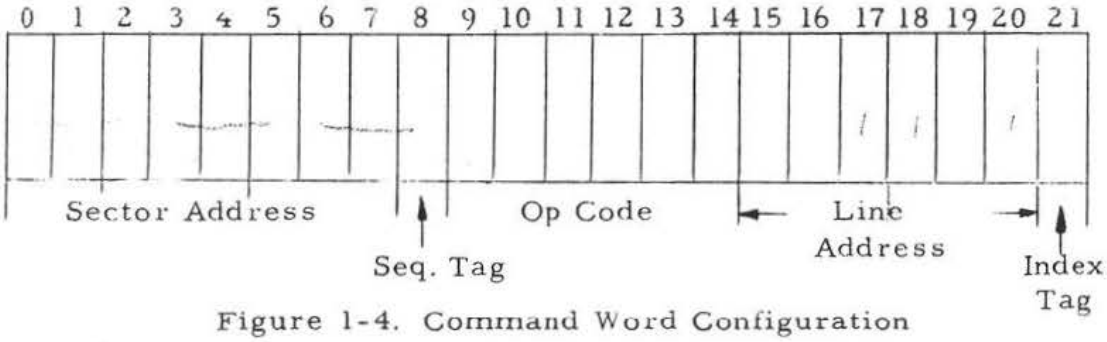


Figure 1-4. Command Word Configuration

Each subdivision, or field, of the command word is uniquely identified. The subdivisions are the sector address, sequence tag, op code, line address, and index tag fields. There will be frequent references in subsequent descriptions, to the address field of a command. Although the address is made up of a sector number and a line number, these numbers are not contiguous in the command format. The address field, however, is considered as a single entity. The address 03204 refers to sector 032 line 04. The contents of the address field in a command do not always designate a memory location.

For example, the shifting commands use the address field to indicate the number of positions to shift.

The sequence tag field may contain either a one or a zero, and its use is detailed in paragraph 1.4 "Command Sequencing and Timing."

The op code field contains a numeric code which specifies one of the PB250 commands.

The index tag field may contain either a one or a zero. When a one is placed in this field, the contents of the Index register are used (see paragraph 1.2.3.1); a zero in the field indicates no use of that register.

Bit position 20 contains a one only when referring to a line address of 40 or greater. For example, an LDA command referring to sector 30, line 42, has an address of 03042 and appears as shown in Figure 1-5.

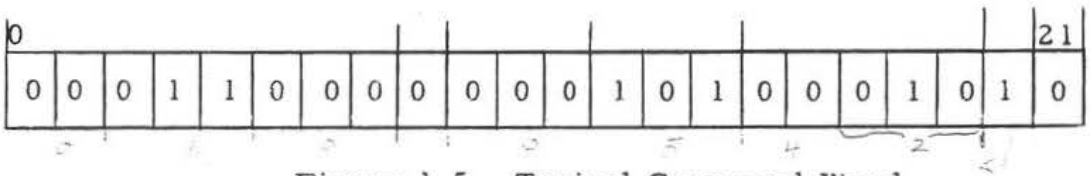


Figure 1-5. Typical Command Word

1.4 COMMAND SEQUENCING AND TIMING

The PB250 reads and executes commands from the circulating command lines. The words of the long lines are read serially in sector address sequence (000, 001, 002, --- 376, 377, 000, 001, ---). The time for each word to pass through a reading device is 12 microseconds; therefore, the time for all 256 words of a long line is 3072 microseconds. The performance of each command involves four phases:

Phase I	Wait to read next command.
Phase II	Read next command.
Phase III	Wait to execute command.
Phase IV	Execute command.

For example, a command 00001 to store A in 03004 will be read (Phase II) in sector 000, held for execution (Phase III) in sectors 001 through 027, executed (Phase IV) in sector 030, and held while waiting to read the next command (Phase I) in sectors 031 through 000. Phase II will follow in sector 001, causing the next command to be read from location 00101.

There are four classes of commands in which the nature of Phase IV differs. A tabulation showing the class into which each command falls is provided in Table 1-1. This tabulation is referred to extensively in Section II of this manual.

1.4.1 Class 1

In this class of commands, execution always follows the reading of the command by skipping Phase III. The sector address of the command is used to designate the first sector number in which Phase IV is discontinued. This class of commands consists of all those which require an extended interval of execution, such as block transfer, shifting, and multiplication. The execution time for this class of command varies with the required duration. For example, block transfer requires 12 microseconds per word, shifting requires 12 microseconds per bit, and multiplication requires 12 microseconds per multiplier bit.

1.4.2 Class 2

In this class of commands, execution is always completed in the sector specified by the sector address of the command. This class consists of all one-sector operations such as load, store, add, and clear. All com-

mands of this class require 12 microseconds to execute. See RTK (p. 2-52) for explanation of modified class 2.

1.4.3 Class 3

Class 3 is an extension of Class 2 to handle double precision operations. As in Class 2, execution always starts in the sector specified by the sector address of the command, but the execution phase is always extended into the following sector. All commands of this class require 24 microseconds to execute.

1.4.4 Class 4

Class 4 consists of commands for conditional and unconditional transfer of control. The condition for a conditional transfer is tested in Phase II and, if the condition is met, the next command is read from the line and sector number specified by the command. If the condition is not met, the command directly following the transfer of control command is read. A conditional transfer where the condition is not met, thus requires no execution time. The unconditional transfer selects the next command with no restrictions. The execution time, when control is transferred, is 12 microseconds per sector for the interval between the transfer of control command and the next command.

1.4.5 Sequence Tag

With commands stored in sequential sectors, the indicated command sequence will proceed at the rate of one instruction per (3072 + 12) microseconds. To provide for a higher computation rate, a Sequence Tag of one may be used in bit position 8 of commands in Classes 1, 2, and 3. The use of this option will cause the next command to be read in the sector directly following the end of the execution phase. For example, a command in 00001 to store A in 03004 will be followed by the command 03101 if the Sequence Tag is a one.

1.5 PARITY CHECK

Each memory word carries an additional position for an even parity check. This position is not under program control and need not concern the programmer in the design and coding of his problem. The parity check is generated during the execution of the STORE and MOVE commands and is tested when loading the arithmetic registers, during adding and subtracting operations, and when reading commands.

Computation will stop on a parity error, and may be restarted by clearing the parity flip-flop with the BREAKPOINT switch and the ENABLE switch of the Flexowriter.

The actual PB250 word consists of 24 bits, of which 22 are accessible to the programmer. A parity bit precedes bit position 0 (see Figure 1-1), and a guard bit follows bit position 21.

Table 1-1 (Sheet 1 of 3)

COMMAND CLASSIFICATIONS

Class 1: Executed Between Command Location and Address Sector Number.		
NORMALIZE AND DECREMENT	NAD	(20)*
NORMALIZE	NOR	(20)*
LEFT SHIFT AND DECREMENT	LSD	(21)*
AB LEFT	SLT	(21)*
RIGHT SHIFT AND INCREMENT	RSI	(22)*
AB RIGHT	SRT	(22)*
SCALE RIGHT AND INCREMENT	SAI	(23)
NO OPERATION	NOP	(24)
INTERCHANGE A AND M	IAM	(25)
MOVE LINE X TO LINE 7	MLX	(26)
SQUARE ROOT	SQR	(30)
DIVIDE	DIV	(31)*
DIVIDE REMAINDER	DVR	(31)*
MULTIPLY	MUP	(32)
SHIFT B RIGHT	SBR	(33)*
LOGICAL RIGHT SHIFT	LRS	(33)*
WRITE OUTPUT CHARACTER	WOC	(6X)
PULSE TO SPECIFIED UNIT	PTU	(70)
MOVE COMMAND LINE BLOCK	MCL	(71)
BLOCK SERIAL OUTPUT	BSO	(72)
BLOCK SERIAL INPUT	BSI	(73)
HALT	HLT	(00)*

Table 1-1 (Sheet 2 of 3)

Class 2: Executed in Address Sector Number

INTERCHANGE A AND C	IAC	(01)
INTERCHANGE B AND C	IBC	(02)
LOAD A	LDA	(05)
LOAD B	LDB	(06)
LOAD C	LDC	(04)
STORE A	STA	(11)
STORE B	STB	(12)
STORE C	STC	(10)
ADD	ADD	(14)
SUBTRACT	SUB	(15)
EXTEND BIT PATTERN	EBP	(40)
GRAY TO BINARY	GTB	(41)
AND M&C	AMC	(42)
CLEAR A	CLA	(45)
CLEAR B	CLB	(43)
CLEAR C	CLC	(44)
AND OR COMBINED	AOC	(46)
EXTRACT FIELD	EXF	(47)
DISCONNECT INPUT UNIT	DIU	(50)
READ TYPEWRITER KEYBOARD	RTK	(51)
READ PAPER TAPE	RPT	(52)
READ FAST UNIT	RFU	(53)
LOAD A FROM INPUT BUFFER	LAI	(55)
COMPARE A AND M	CAM	(56)
CLEAR INPUT BUFFER	CIB	(57)
~~HALT~~	~~HLT~~	(00)*
MERGE A INTO C	MAC	(00)*

Table 1-1 (Sheet 3 of 3)

Class 3: Executed In Address Sector Number And Following Sector.

ROTATE	ROT	(03)
LOAD DOUBLE PRECISION	LDP	(07)
STORE DOUBLE PRECISION	STD	(13)
DOUBLE PRECISION ADD	DPA	(16)
DOUBLE PRECISION SUBRACT	DPS	(17)

Class 4: Executed Between Command Location And Address Sector Number.

TRANSFER UNCONDITIONALLY	TRU	(37)
TRANSFER IF A NEGATIVE	TAN	(35)
TRANSFER IF B NEGATIVE	TBN	(36)
TRANSFER IF C NEGATIVE	TCN	(34)
TRANSFER ON OVERFLOW	TOF	(75)
TRANSFER ON EXTERNAL SIGNAL	TES	(77)

* Asterisk indicates that the OP code has at least two meanings, depending on the address used with the command. See Section II for a detailed description of the commands.

II. PB 250 COMMANDS

2.1 GENERAL

2.1.1 Command Structure

For each PB 250 command, a 3-letter mnemonic code has been devised. These mnemonics are derived from an abbreviation of the command names and are a convenient device for remembering the function of the command.

When writing a command word, the language of the Octal Utility Program (Appendix C) will be used. This language is the standard language for the communication of programs. Thus, referring to the illustration of a typical command word (Figure 1-2), the fields are written as follows:

a) Sector Address: Three octal digits specifying the particular sector to be used ($000 \leqslant SSS \leqslant 377$).

b) Sequence Tag: If sequence tag is present, a capital S will be written; if no sequence tag is used, a blank space will separate the sector address and OP Code.

c) Operation Code (OP Code): Two digits which indicate what command will be executed.

d) Line Address: Two octal digits specifying the particular line to be used ($0 \leqslant LL \leqslant 77$).

e) Index Register: If the contents of the Index register are to replace the line address, there will be a capital I at the end of the command; if the Index register is not being used, there will be a semi-colon (;) at the end of the command.

The following two commands illustrate this procedure:

017	S	37	03	;
a	b	c	d	e

002	⌣	05	02	I
a	b	c	d	e

Note: The letters a, b, c, d, and e refer, respectively, to the sector address, sequence tag, op code, line address, and Index register.

2.1.2 Command Descriptions

In this manual, the notations A, B, and C will be used to refer, respectively to the A register, B register, and C register, while M will be used to refer to a particular memory location. Parentheses around the letter indicate the contents of the register or memory location; e.g., (A) refers to the contents of the A register.

The "contents of" always refers to all 22 bits of the appropriate register or memory word, unless indicated otherwise by numerical subscripts. These numerical subscripts tell to which particular bits reference is being made. For example: $(A)_{0-10}$ refers to bit positions 0 through 10, inclusive, of A; $(B)_5$ refers to bit position 5 of the B register; $(01502)_{3-6}$ sector 15, line 2, bits 3 through 6.

"Effective address" will be used to mean the actual address employed by the computer in execution of a command; if the Index register is used, then the effective address will be the contents of the Index register and the sector address specified by the command word.

It should again be noted that throughout this manual all op codes, line numbers, and sector numbers will be in octal notation.

Command descriptions in this section will consist of four parts, or less, as required, These parts will be:

a) Description: Details of what the command does - - its effect on registers, memory locations, etc.

b) Example: Specific numerical example showing the appearance of the registers and relevant memory locations before and after execution of the command. (In the case of such basic commands as CLEAR A, STORE A, etc., no example is given.)

c) Timing: The timing classification of the command plus, as required, optimization information such as addressing for optimum timing.

d) Usage: Exceptions to the use of the command or examples of how the command may be used. (Especially useful in such commands as GRAY TO BINARY and EXTEND BIT PATTERN, whose use might not be readily apparent to the programmer.)

2.1.3 Special Considerations

Codes 27, 54, 74, and 76 are unassigned and should not be used by the programmer. In the event that these op codes are used, the computer will not halt but will try to execute a command unintended by the programmer.

Certain computer commands operate in a modified manner as determined by the address of the command. These modifications are either described under the commands to which they apply or, if more appropriate, listed as separate commands.

It should be noted that sequence tagging (as described in Section I) never permits the command execution sequence to transfer to a different line, except in the case of a TRU. That is, if the computer executes a sequence-tagged command from line γ, the next command will always be executed from line γ, regardless of sequence tagging - - except in the case of a TRU command with a line address $\neq \gamma$.

Note: The term "execution time," as used in this section, includes the 12 microseconds needed to read the command in addition to the time necessary to perform the required operation.

HLT Halt (00)*

This command stops computation under the conditions noted below and turns on the parity error indicator light on the console. The OPERAND lights on the console will indicate the line address associated with this command. To continue execution of the program, the ENABLE switch and the BREAKPOINT switch on the Flexowriter must be depressed. This will turn off the parity error indicator and, upon release of the ENABLE switch, the program will continue. This command will not stop computation if the sector address equals $\alpha + 1$, when the HLT command itself is located in α. (See MAC description.)

Timing: HLT is a class 1 command. If parity is cleared, and the HLT command is sequence tagged, the next command is executed from β, where β is the sector address. If the HLT command is not sequence tagged, the next command is executed from $\alpha + 1$. where HLT is located in α.

Usage: Error halts in a program are easily identified if difference line numbers are used, thus providing a ready means of determining the location within the program at which the computer has halted, the line number being read from the console lights. The Octal Utility Program uses HLT $37)_8$ to indicate a checksum error.

MAC Merge A into C (00)*

This command is a special case of HALT (00). If a HALT command is given which has as its sector address $\alpha + 1$, where α is the sector of the HLT, the program will not halt. Instead, there will be a logical A OR C executed, with the result appearing in C. The contents of A are merged into the contents of C; a one is placed in those bit positions of C in which there are ones in the corresponding positions of A or C or in both. All 22 positions of A and C take part in this operation. The (A) and (B) are not affected by this command.

Example:

	(A)	(C)
Before execution of MAC	01100101	11010101
After execution of MAC	01100101	11110101

Timing: MAC operates as a class 2 command, being executed in sector $\alpha + 1$. If the sequence tag is 1, the next command executed will be in $\alpha + 2$; whereas, if the MAC is not sequenced, the next command follows from sector $\alpha + 1$. Note that this is different from a sequenced halt command, when the next command comes from the sector specified.

Usage: When the C register is cleared before execution of MAC, the command effectively functions as a "copy A into C", that is, the contents of A are duplicated in C. When using this command, it should be remembered that the sectors are addressed circularly, with sector 000 following sector 377.

IAC Interchange A and C (01)

The contents of the A register are loaded into the C register, and the contents of the C register are loaded into the A register. These operations occur simultaneously; thus, no information is lost.

Example:

	(A)	(C)
Before execution of IAC	+0123456	+6543210
After execution of IAC	+6543210	+0123456

Timing: IAC is a class 2 command. The sector address has meaning only in terms of sequence tagging (providing a transfer). The line address may be any number. The sector address, however, for minimum execution time (24 microseconds) must be $a + 1$, where a is the location of the INTERCHANGE A AND C command. The next command to be executed, under sequence tagging, will be taken from $a + 2$.

IBC Interchange B and C (02)

The contents of the B register are loaded into the C register, and the contents of the C register are loaded into the B register. These operations occur simultaneously; therefore no information is lost.

Example:

	(B)	(C)
Before execution of IBC	+2043177	+0021701
After execution of IBC	+0021701	+2043177

Timing: IBC is a class 2 command. (For further description, see IAC, 01, which is similar to IBC.)

ROT Rotate A, B, and C (03)

The contents of the A, B, and C registers are simultaneously rotated in the following fashion: the contents of C are placed in B; the contents of B are placed in A; and the contents of A are placed in C. No information is lost.

Example:

	(A)	(B)	(C)
Before execution of ROT	+ 1205721	+ 6201530	- 3170024
After execution of ROT	+ 6201530	- 3170025	+ 1205721

Timing: ROT is a class 3 command; 36 microseconds is the minimum execution time. Although the sector address has no meaning in terms of execution of the command, for optimum programming, the address $\alpha + 1$ is required, where α is the location of the ROT command. This addressing, in conjunction with the sequence tag, obtains a minimum execution time (36 microseconds). The next command will be executed from $\alpha + 3$. The line address may be any number. As in all other commands in which sector address has no meaning in terms of command execution, ROT may be used to provide a transfer by use of sequence tagging.

LDA Load A (05)

The A register is cleared and the contents of M, the effective address, are read into the A register. The previous contents of A are destroyed: the contents of M are not affected.

LDB Load B (06)

The B register is cleared and the contents of M, the effective address, are read into the B register. The previous contents of B are destroyed; the contents of M are not affected.

LDC Load C (04)

The C register is cleared and the contents of M, the effective address, are read into the C register. The previous contents of C are destroyed; the contents of M are not affected.

Timing: LDA, LDB, and LDC are class 2 commands. To obtain minimum execution time (24 microseconds), the operand which is to be loaded into the register must be located in the next sector after the command ($\alpha + 1$), but not necessarily in the same line, and the command must have a sequence tag of one. The next command to be executed will be taken from $\alpha + 2$, where α is the location of the load command.

LDP Load Double Precision A ← M+1 B ← M (07)

Both the A and B registers are cleared. The contents of M, the effective address, are read into the B register; the contents of M + 1 are read into the A register. The contents of M and M + 1 are not affected.

Timing: LDP is a class 3 command. To obtain minimum execution time (36 microseconds), the operand must be stored in $\alpha + 1$ and $\alpha + 2$, where LDP is located in α, in any line. Sequence tagging under these circumstances results in the next command being executed from $\alpha + 3$.

Usage: This command, along with the other double precision commands, provides double precision arithmetic capacity within the command structure of the PB 250. Furthermore, in terms of data handling, it is often convenient to pick up or store two consecutive words which are not a single number but are two separate units of information. The LDP command reduces the number of memory accesses necessary in a program.

Some discussion of double precision is in order. A double precision number consists of two words, or 44 bits. Commands functioning in the double precision mode will operate on two words and treat A and B as one register, where A is the Most Significant Word (MSW) and B is the Least Significant Word(LSW).

Double precision numbers must be stored in consecutive words; the effective address is the lower-ordered address. For example, if the specified memory location is 03404, the double precision number is stored in memory locations 03404 and 03504. Location 03404 contains the Least Significant Word (LSW), while 03504 contains the Most Significant Word (MSW).

STA Store A (11)

The contents of the A register are stored in M, the effective address. The previous contents of M are destroyed; the contents of the A register are not affected.

STB Store B (12)

The contents of the B register are stored in M, the effective address. The previous contents of M are destroyed; the contents of the B register are not affected.

STC Store C (10)

The contents of the C register are stored in M, the effective address. The previous contents of M are destroyed; the contents of the C register are not affected.

Timing: STA, STB, and STC are class 2 commands. To obtain minimum execution time (24 microseconds), the contents of the register must be stored in the next sector after the command (α + 1), but not necessarily in the same line, and the command must have a sequence tag of one. The next command to be executed will be taken from α + 2, where α is the location of the store command.

STD Store Double Precision A → M+1 B → M (13)

This command operates on both the A and B registers. The contents of the B register are stored in M, the effective address; the contents of the A register are stored in M + 1. For example, if the specified address is 00004, the contents of B are stored in 00004 and the contents of A are stored in 00104. The previous contents of A and B are not affected; the previous contents of 00004 and 00104 are lost.

Timing: STD is a class 3 command.

The contents of M, the effective address, are algebraically added to the contents of the A register. This sum replaces the contents of A; the contents of M are unaffected. Overflow occurs when (A) and (M) initially have like signs and the result in A has a different sign.

Example: The command 011 1403; is executed. The contents of line 3, sector 011, are + 0210416.

	(A)	(01103)
Before execution of ADD	+0143115	+0210416
After execution of ADD	+0353533	+0210416

Timing: ADD is a class 2 command. To obtain the minimum execution time (24 microseconds), the operand which is to be added to (A) must be located in the next sector after the command, but not necessarily in the same line, and the command must have a sequence tag of one. The next command to be executed will be taken from $\alpha + 2$, where α is the location of the ADD command.

Usage: Reference should be made to the discussion of 2's complement arithmetic in Appendix A prior to coding arithmetic problems for the PB 250.

The contents of M, the effective address, are algebraically subtracted from the contents of the A register. The result replaces the contents of A; the contents of M are unaffected. Overflow occurs when (A) and - (M) initially have like signs and the result in A has a different sign.

Example: The command 125 1507; is executed. The contents of line 7, sector 125, are +0231234.

	(A)	(12507)
Before execution of SUB	+6120134	+0231234
After execution of SUB	+5666700	+0231234

Timing: SUB is a class 2 command.

DPA Double Precision Add A+(M+1) B+M (16)

The contents of the word pair starting at M, the effective address, are algebraically added to the contents of the combined A and B registers. This sum replaces the contents of A and B; the word pair beginning at M is not affected. Position 0 of the B register does not act as a sign; but is part of the magnitude of the number, and any carry from position 0 of B propagates into position 21 of A. Overflow occurs when (A) and (M+1) initially have like signs and the result in A has a different sign. The double precision word in memory starts with (M + 1), where (M) represents the least significant part of the double precision number.

Example: The command 002 1602; is executed. The contents of line 02, sector 003, are + 1210456. The contents of line 02, sector 002, are 73120604 (111011001010000110000l).

	(A)	(B)	(003)	(002)
Before execution of DPA	+0124471	31425000	+1210456	73120604
After execution of DPA	+1335150	24545604	+1210456	73120604

Timing: DPA is a class 3 command. To obtain the minimum execution time of 36 microseconds, the operand which is to be added to (AB) must be located in the next two sectors after the command, but not necessarily in the same line and the command must have a sequence tag of one. The next command to be executed will be taken from α + 3, where α is the location of the DPA command.

Usage: The DPA command may be used to accumulate a double precision sum, where six decimal digits are not sufficient in an arithmetic computation. Another use occurs when it is certain that the sum in B will not overflow to A; two separate sums may then be accumulated, one in A and one in B. ADD may be used to add to (A), while DPA may be used to add to (B), where the most significant word to be added to (AB) consists of all zeros. A further use of DPA is to

round a positive double precision number in (AB) to a single precision number in A. The number to be added to (AB) should appear as follows:

$$\alpha = -0000000$$

$$\alpha + 1 = +0000000$$

The contents of the word pair starting at M, the effective address, are algebraically subtracted from the contents of the combined A and B registers. The result replaces the contents of A and B; the word pair at M is not affected. Position 0 of the B register does not act as a sign, but is a part of the number, and any carry from position 0 of B propagates into position 21 of A. Overflow occurs when (A) and - (M + 1) initially have like signs, while the result in A has a different sign. The double precision word in memory starts with (M+ 1); (M) represent the least significant part of the double length number.

Example: The command 113 1705; is executed. The contents of line 5, sectors 114 and 113 are, respectively, +0124471 and 31425000.

	(A)	(B)	(114)	(113)
Before execution of DPS	+1210456	73120604	+0124471	31425000
After execution of DPS	+1063765	41473604	+0124471	31425000

Timing: DPS is a class 3 command.

NAD Normalize and Decrement (20)*

The address field of the NORMALIZE AND DECREMENT command is not used to specify the location of an operand, but contains an address number, N, which specifies the first sector following the completion of execution. In executing this command, the (AB) are shifted left until one of two conditions is met:

1) $(A)_0 \neq (A)_1$; i.e., the contents of A, position 0, do not equal the contents of A, position 1.

2) (AB) has been shifted S positions (where S is selected by the programmer).

The line address should not have a one in position 16 (see description of NOR command). The (C) are decremented by one for each position shifted. Position 0 of A does not move, but position 0 of B takes part in the shifting. The vacated positions of B are filled with zeros. The programmer should select S large enough so as not to inhibit proper normalization. S is used in determining N in the following manner:

$$N)_8 = \text{Sector location of the command})_8 + S)_8 + 1)_8 .$$

Example: The command 071 2000; is located in sector 015 of line 02.

Before execution of NAD	+0012461	34105614	+0000010
After execution of NAD	+5230560	42706000	+0000000

Timing: NAD is a class 1 command. If a sequence tag of one is used, the next command is read from N. With a sequence tag of zero, the next command is read from $\alpha + 1$, where α is the sector location of the NAD command.

Usage: This command may be used in "floating" a fixed-point number to obtain a normalized floating point representation. Choosing S equal to $53)_8$

allows for normalizing every possible number in AB, but still terminates the operation if (AB) equal zero. If normalization is accomplished before N time, the command is executed as a NOP (24) for the remaining sectors. Note that a shift of zero positions cannot be accomplished by any of the shifting commands.

NOR Normalize (20)*

The address field of the NORMALIZE command is not used to specify the location of an operand, but contains an address, N, which specifies the first sector following completion of execution. In executing this command, the (AB) are shifted left until one of two conditions is met:

1) $(A)_0 \neq (A)_1$

2) (AB) has been shifted S positions, where S is selected by the programmer.

The line address must have a one in position 16. (See description of NAD command.) The (C) are not affected by execution of NOR. Position 0 of A does not move, but position 0 of B takes part in the shifting and moves from 0 of B into 21 of A, etc. The vacated positions of B are filled with zeros. The programmer should select S large enough so as not to inhibit proper normalization. S is used in determining N in the following manner:

$$N)_8 = \text{Sector location of the command})_8 + S)_8 + 1)_8$$

Example: The command 071 2010; is located in 01502.

	(A)	(B)
Before execution of NOR	- 7731245	32001420
After execution of NOR	- 3124532	00142000

Timing: NOR is a class 1 command. If a sequence tag of one is used, the next command is read from N. With a sequence tag of zero, the next command is read from $\alpha + 1$, where α is the sector location of NOR.

Usage: Choosing $S = 53)_8$ allows for normalization of every possible number in AB, but still terminates the operation if (AB) equal zero. If normalization is accomplished before N time, the command is executed as a NOP (24) for the remaining sectors.

The (AB) are shifted left for S positions, S being determined by the programmer. The (C) are decremented by one for each position shifted. Bits shifted past position 1 of A are lost and zeros fill the vacated positions of B. Position 0 (the sign) of A is not moved, but position 0 of B takes part in the shifting. The line address of this command should not have a one in position 16. (See description of SLT command). The sector address field of this command is not used to specify the location of an operand, but contains an address, N, which is determined by:

$$N)_8 = \text{Sector location of the command})_8 + S)_8 + 1)_8 .$$

Example: The command 021 2100; is located in line 3, sector 015.

	(A)	(B)	(C)
Before execution of LSD	- 1532104	36124104	+0000007
After execution of LSD	- 5321043	61241040	+0000004

Timing: LSD is a class 1 command. The next command to be executed, when this command has a sequence tag of one, is the command located in N.

Usage: This command should be used only when it is desired to decrease (C) by 1 for each position shifted left. It is important to remember that the sign position of A does not participate in the shifting. Note: $S \geqslant 53)_8$ results in setting $(A)_{1-21}$ and $(B)_{0-21}$ equal to zero.

SLT Shift Left (21)*

The (AB) are shifted left for S positions, S being determined by the programmer. The (C) are not affected by this command. The line address of this command must have a one in position 16 (see description of LSD command). Bits shifted past position 1 of A are lost, and zeros fill the vacated positions of B. Position 0 of A is not moved (does not participate in the shifting), but position 0 of B does participate in the shifting. The sector address of this command is not used to locate an operand, but contains an address, N, which determines the length of the shift.

$$N)_8 = \text{Sector location of the command})_8 + S)_8 + 1)_8 .$$

Example: The command 021 2110; is located in line 03, sector 015.

	(A)	(B)
Before execution of SLT	-1532104	36124104
After execution of SLT	-5321043	61241040

Timing: SLT is a class 1 command. The next command to be executed, when this command has a sequence tag of one, is the command located in N.

Usage: This command may be used when it is desired to shift left without disturbing (C). The sign position of A does not participate in the shifting, and $S > 53)_8$ results in setting $(A)_{1-21}$ and $(B)_{0-21}$ equal to zero.

The (AB) are shifted right for S positions, S being determined by the programmer. The (C) are incremented by one for each position shifted. The bit in the sign position of A is copied into the vacated positions of A. Bits shifted past position 21 of B are lost. Position 0 (the sign) of A is not moved, but position 0 of B takes part in the shifting. The line address should not have a one in position 16. (See description of SRT command.) The address field of this command is not used to specify the location of an operand, but contains an address number, N, which is determined by:

$$N)_8 = \text{Sector location of the command})_8 + S)_8 + 1)_8 .$$

Example: Command 021 2200; is located in sector 015 of line 03.

	(A)	(B)	(C)
Before execution of RSI	-3120456	47217030	+0000000
After execution of RSI	-7312045	64721700	+0000003

Timing: RSI is a class 1 command.

Usage: Use RSI only when it is desired to shift (AB) right and to increment the C register. (when C register incrementing is undesirable, see description of SRT command.)

SRT Shift Right (22)*

The (AB) are shifted right S positions, S being determined by the programmer. The (C) are not affected. The bit in position 0 of A (sign position) is copied into the vacated positions of the A and B registers. Bits shifted past position 21 of B are lost. Position 0 (sign position) of A is not moved but position 0 of B takes part in the shifting. Note: The line address of this command must be such that bit position 16 contains a one. (See description of RSI command.) The sector address field of this command is not used to specify the location of an operand, but contains an address number, N, which is determined by:

$$N)_8 = \text{Sector location of the command})_8 + S)_8 + 1)_8 .$$

Example: The command 200 2210; is located in line 2, sector 171.

	(A)	(B)
Before execution of SRT	- 3177204	21643104
After execution of SRT	- 7731772	04216430

Timing: SRT is a class 1 command.

Usage: This command should be used when it is desired to shift (AB) right without affecting the (C). (If incrementing the C register is desirable, see description of RSI command.)

SAI Scale Right And Increment (23)

The (AB) are shifted right and the (C) are incremented by one for each position shifted. The operation continues until one of the two conditions is met:

1) $(C) \geqslant 0$
2) (AB) are shifted S positions, where S is selected by the programmer.

The bit in the sign position of A is copied into the vacated positions of A. Position 0 (the sign) of A is not moved, but position 0 of B takes part in the shifting. S should be so selected as not to inhibit the scaling. The line address of this command should be zero. The sector address field of this command is not used to specify the location of an operand, but contains an address number, N, which is determined by:

$$N)_8 = \text{Sector location of the command})_8 + S)_8 + 1)_8 .$$

Example: The command 004 2300; is located in 00002.

	(A)	(B)	(C)
Before execution of SAI	+1231046	21320040	-7777500
After execution of SAI	+0123104	62132004	-7777503

Timing: SAI is a class 1 command. If sequence tagging is used with the command, the next command to be executed will be taken from N, even if condition (1), above, is obtained before N sector time.

Usage: This command can be used in "fixing" floating point numbers at a particular scale factor. If (C) become $\geqslant 0$ before N time, the command is executed as a NOP (which, in this case, will have an op code number of 27) for the remaining sectors.

NOP No Operation (24)

This command causes the computer to continue in the regular command sequence. Memory and registers are not affected.

Timing: NOP is a class 1 command. Sector address has meaning only in the event that a maximum operation speed is to be obtained. Optimum programming requires a sequence tag of one and a sector address of $\alpha + 2$, where NOP is located in α. The next command to be executed will come from $\alpha + 2$. Line address may be any number. NOP may also function as a transfer to β, when the sector address of the NOP command is β (β must be in the same line as NOP).

IAM Interchange A and M (25)

This command interchanges information in the line designated by the line address, with the information in the A register. The interchange starts in the sector following the IAM command and continues up to, but not including, the address sector number. This command results in a one-word precession of the information in the designated line. The information originally in the A register is entered into the first sector and is replaced by the information in the last sector.

Example: The command 015 2503; is located in sector 012 of line 2.

	(A)	(01303)	(01403)
Before execution of IAM	+3214071	-5377210	+3246002
After execution of IAM	+3246002	+3214071	-5377210

Timing: IAM is a class 1 command.

Usage: This is a very convenient way of manipulating sector sequential data in memory without modifying addresses. In effect, the designated sectors and the A register function, temporarily, as a special line. Each time IAM is executed, a stepping of data takes place as shown below. Note: α is the sector location of the IAM command, but is not necessarily in the same line as $\alpha + 1$, $\alpha + 2$, etc., and $\alpha + N + 1$ is the IAM sector address.

Location	Initial Contents	After 1st IAM	After 2nd IAM
A register	X_a	X_n	X_{n-1}
$\alpha + 1$	X_1	X_a	X_n
$\alpha + 2$	X_2	X_1	X_a
.	.	.	.
.	.	.	.
.	.	.	.
$\alpha + N-1$	X_{n-1}	X_{n-2}	X_{n-3}
$\alpha + N$	X_n	X_{n-1}	X_{n-2}

MLX Move Line X to Line 7 (26)

This command transfers information from the effective line address to line 07. The transfer begins in the sector following the MLX command and continues up to, but not including, the sector address.

Timing: MLX is a class 1 command; timing is similar to that for MCL (71).

Usage: This command should be studied in conjunction with MCL (71). It is to be noted, that both of these commands, though similar, have certain significant differences. MCL moves an entire command line, or any part of a command line in which the MCL is actually located, into another line. MLX moves some specified line, not necessarily the one in which it is located, or part thereof, into line 7; thus, in the case of a machine in which subroutines are stored in lines 10, 11, etc., it may be desirable to move these subroutines into line 7 for execution. This can be accomplished by using the MLX command. An entire line may be moved by giving the address $\alpha + 1$, where the MLX command is located in α. It can be seen that both of these commands have a separate and important use in the PB 250. Judicious use of these commands provides an easy method for moving data from line to line, while preserving the same relative sector locations.

The argument must be in the combined AB registers. The (C) must be positive. The square root appears in B with the remainder in A. The C register takes part in this operation and its contents are replaced by the square root. The (C) will be the full root but will differ from the (B) in the least significant bit computed. If only A is loaded with the argument, (B) should be cleared or they may influence the least significant bit of the computed root.

The line address of this command should be zero. The sector address contains a number, N, which specifies the first sector location following the completion of the operation. The SQR command is a variable length operation, which permits the programmer to specify a quantity, S, which is the number of bits of the root that are to be developed. N is determined from S as follows:

$$N)_8 = \text{Sector location of the command})_8 + S)_8 + 1)_8$$

The argument, (AB), must be positive for this operation to be executed correctly. If S = 21, the full root is formed in B.

Example: The command 006 3000; is located in 36005.

	(A)	(B)	(C)
Before execution of SQR	+0100000	+0000000	+0000000
After exectuion of SQR	-5777776	+1000000	+1000001

Timing: The number whose square root is to be found should be at an even scale factor, 2Q. The result in the B register will be scaled at Q + 21 - S. For example, where $S = 21)_{10}$ and the (AB) are at 2Q = 20, the result in B is scaled at Q = 10. If S = 10, and the (AB) are at 2Q = 20, the result is in B at

Q = 21. Bit 11 of B will be a zero, and the result will be in bits 12 through 21; bit 0 of C will be a zero, and nine bits of the result will be in bits 1 through 9. SQR is a class 1 command.

DIV Divide (31)*

The dividend is in the combined AB registers and the divisor is in the C register. The quotient appears in the B register, with a remainder in A. The line address of this command should not have a one in either positions 15 or 19. The sector address field contains an address, N, which specifies the first sector location following the completion of the operation. The DIV command is a variable length operation, which permits the programmer to specify a quantity, S, which is the number of bits of the quotient (including sign) to be developed. If S is 22, the full quotient is formed in B, with a sign in $(A)_{21}$, and the unit bit in $(B)_0$. In case the divisor was greater than the dividend, the units bit will equal the sign bit, and the quotient will appear as a signed number in B only. N is determined as follows:

$N)_8$ Sector location of the command$)_8 + S)_8 + 1)_8$

Example: The command 027 3100; is located in 00003.

Before execution of DIV	+0700000	+0000000	-7100000
After execution of DIV	-6200001	+7777777	-7100000

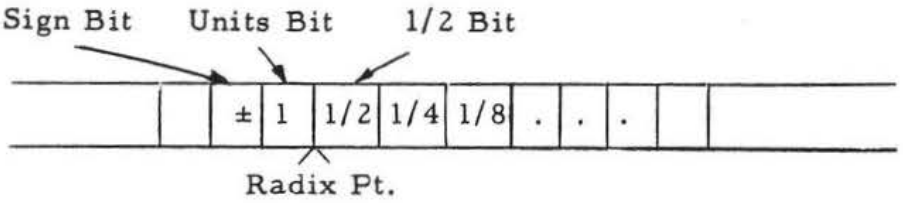
This is a divide with S = 22. The last bit of A is the sign of the quotient, which is negative. In canonical form, the quotient is -0000000, and the remainder is +0000000.

Timing: DIV is a class 1 command. If a sequence tag of one is used, the next command is executed from N.

Usage: 1) If the dividend is scaled at Q (a), and the divisor at Q (b), then the quotient is scaled at Q $\left[a - b + 22 - S\right]$.

2) The machine remainder is scaled at Q b-1 . The corrected remainder will be scaled at Q (b).

3) The binary point of the quotient is preceded by the unit bit and sign, and is succeeded by the 1/2 bit, 1/4 bit, 1/8 bit, etc. Bits to the left of the sign bit are not cleared.



QUOTIENT

In case the divisor is, in absolute value, greater than the dividend, then the sign and unit bits are equal. Whenever the quotient is less than 2 in absolute value, the unit bit reflects the true integral value. In case S = 22, the unit bit is in $(B)_S$, and the sign of the quotient is in $(A)_{21}$. This will affect the least significant bit of the remainder. For example, a full division of -1, scaled at Q (0), by itself, gives a quotient of + 1 scaled at Q (0), i.e., a one in $(B)_S$ and zeros in $(A)_{21}$ and $(B)_{1-21}$.

4) To obtain the undivided remainder at Q (b) from the machine remainder, shift (A) right one position, using an LRS with bit 15 equal to ~~zero~~ one; if $(A)_S$ and $(C)_S$ are now unequal, add (C) to (A). The undivided remainder is in the A register.

5) The canonical quotient is, in absolute value, less than, or equal to, the theoretical answer. This implies that the sign of the canonical divided remainder has the same sign as the quotient. In the PB 250, the quotient is always less than, or equal to, the theoretical answer. Therefore, the divided

remainder will always be positive. For example, using integers scaled at the right of the registers, -5 divided by + 3 is -1 with a divided remainder of -2/3 in canonical form. In the PB 250, a quotient of -2 and a divided remainder of + 1/3, is obtained which is mathematically correct. In the case of a negative quotient, the quotient and undivided remainder must be altered if canonical form is desired. Note that the quotient need only be corrected in the least significant bit position. Therefore, for most purposes, the machine quotient is sufficiently accurate.

6) The correction to canonical form, which is described in (7), can be avoided if the original dividend and divisor are both positive, i.e., if one attaches the sign to the quotient and remainder after the division takes place. The correction described in (8) must be applied in either case.

7) To obtain an answer in canonical form, the quotient is altered by adding a (+1) in bit position 21 if the quotient is negative. Table 2-1 shows how to go directly from the uncorrected machine remainder to the canonical undivided remainder. First shift (A) right one place using an LRS command with bit 15 equal to ~~zero~~. Then add or subtract (C), or leave (A) unchanged according to Table 2-1. This depends on the signs $(A)_S$, $(B)_S$, and $(C)_S$ after the shift and before the quotient is corrected. The remainder will have a scale of Q (b).

Table 2-1

DIVISION CORRECTION

$(C)_S$	$(A)_S$	$(B)_S$	Correction
+	+	+	none
+	+	-	-(C)
+	-	+	+(C)
+	-	-	none
-	+	+	none
-	+	-	+(C)
-	-	+	none
-	-	-	-(C)

8) After the correction to canonical form, the quotient may be exactly one unit less than the answer, in absolute value. This will be reflected by:

a) (remainder) = (divisor) if the quotient is positive.

b) (remainder) = - (divisor) if the quotient is negative.

In these cases, the quotient should be increased or decreased by a (+1) in bit position 21, and the remainder set equal to zero.

The remainder is in the combined AB registers, and the divisor is in the C register. The quotient appears in the B register; the remainder appears in A. The line number of this command should have a one in position 19 and a zero in position 15. The sector address field contains an address, N, which specifies the first sector location following the completion of the operation. The DVR command is a variable length operation, which permits the programmer to specify a quantity, S, which is the number of bits of the quotient to be developed. The quotient has no sign. If S=22, the most significant bit will be in $(B)_0$. N is derived as follows:

$$N)_8 = \text{Sector location of the command})_8 + S)_8 + 1)_8.$$

Example: A 4, scaled at 24, is divided by 3, scaled at 21. The result, with S=21, should be 1 1/3, scaled at 4. The result after the DIV is shown, and then the result after saving the quotient, clearing the B register, DVR with S=22, and replacing the original quotient into the A register, giving a double precision result.

	(A)	(B)	(C)
Before execution of DIV	+ 0000000	- 0000000	+ 0000003
After execution of DIV	- 7777776	+ 0525252	+ 0000003
After execution of DVR and splicing	+ 0525252	- 2525252	+ 0000003

Timing: DVR is a class 1 command. If sequence tag of one is used, the next command is executed from N.

Usage: The DVR operates on an uncorrected remainder. Before performing the DVR, if maximum accuracy is desired, the quotient should be saved and the B register should be cleared. For maximum accuracy, the original DIV should

have used an S of 21, maximum. This is because of the sign bit in $(A)_{21}$ when S = 22 (see DIV description). The quotient of the DVR, with S = 22, can be spliced to the quotient of the DIV. In general, the quotient of the DVR should be shifted left (22 - S) places before splicing it to the quotient of the DIV. The correction to the remainder, and the correction for canonical form, follow the procedure described in DIV, except that correcting the quotient requires a DOUBLE PRECISION ADD (DPA) command of + 1 in the 43rd bit of the quotient.

The multiplier must be loaded into the B register and the multiplicand must be loaded into the C register. The computer clears the A register before multiplying, provided that the line address of the command does not have a one bit in position 15. The product appears in the combined AB registers; (C) are unaffected. The sign of the product and the 21 most significant bits of magnitude appear in the A register; the ~~22~~ 21 least significant bits of magnitude appear in the B register.

The address field of the MULTIPLY command is not used to specify the location of an operand, but contains an address number, N, which specifies the first sector number following the completion of multiplication. The MULTIPLY command is a variable length operation and, as such, the programmer may specify a quantity, S, which is the number of bits, starting from the least significant end of the multiplier, B, to operate on the multiplicand, C. If the binary point is always considered to be to the right of the sign, and S is $22)_{10}$, or $26)_{8}$, then the full product is formed in A and B with the binary point to the right of the sign bit in A. Note that the sign of B is counted as a multiplier bit. If S is $23)_{10}$, or $27)_{8}$, one-half of the product is formed in A and B with the binary point to the right of the sign bit in A. N is determined from S in the following manner:

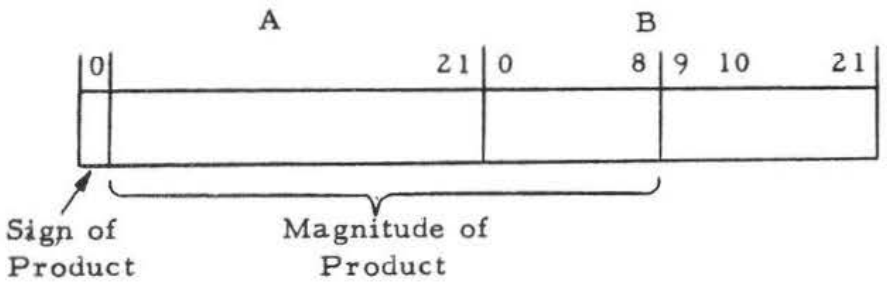
$$N)_8 = \text{Sector number of the command})_8 + S)_8 + 1)_8 .$$

Timing: MUP is a class 1 command. 12 microseconds are required to read the command; 12 S microseconds are required to carry out the command. In the event a sequence tag of 1 is used, the next command is executed from N.

Example: The command 037 3200; is located in 01003.

Before execution of MUP	irrelevant	+0000003	+0000004
After execution of MUP	+0000000	+0000030	+0000004

Usage: When $S = 26)_8$ is used, all the bits of the multiplier operate on all the bits of the multiplicand. Binary scaling follows the rule that the scale factor of the product equals the sum of the scale factors of the multiplier and the multiplicand. If the (B) are at Q = 10 and the (C) are at Q = 17, then the Q of the product is 27. (The binary point is between bit positions 5 and 6 of the B register.) When a product which is less than full length is formed (which reduces the time required to execute a MUP), S bits of the B register are combined with the 22 bits of the C register to form a product which occupies S + 21 significant bits of the combined AB registers, starting with the sign position of A. For example, if the multiplier (B) is known to be always no more than 9 bits plus sign, S would equal $12)_8$, and the product would appear as shown:



The bits which are originally in $(B)_{0-11}$ are moved to $(B)_{10-21}$, with the bit in $(B)_{10}$ repeated in $(B)_9$.

SBR Shift B Right (33)*

The (AB) are shifted right, S positions, S being determined by the programmer. The (C) are unaffected by the execution of this command. After (AB) are shifted right one bit position, the A register is cleared; thus, if $S \geqslant 2$, zeros are shifted into B after sector time $\alpha + 1$, where α is the location of the SRB command. Bits enter $(B)_0$ from $(A)_{21}$; bits shifted past position 21 of the B register are lost. The line address of this command must have a zero in position 15 (see description of LRS command). The sector address field of this command is not used to specify the location of an operand, but contains an address number, N, which is determined by:

$$N)_8 = \text{The Sector location of the command})_8 + S)_8 + 1)_8 .$$

Example: The command 004 3300; is located in sector 000 of line 3.

	(A)	(B)
Before execution of SBR	10101111	01011001
After execution of SBR	00000000	00101011

Timing: SBR is a class 1 command.

LRS Logical Right Shift (33)*

The (AB) are shifted right S positions, S being determined by the programmer. The (C) are unaffected by the execution of this command. LRS differs from RSI in that the sign position of A, $(A)_0$, participates in the right shift. The parity bit is copied into the sign position of A, and, if shifting continues, it is then copied into the vacated positions of AB. Bits shifted past position 21 of B are lost. The line address of this command must have a one in position 15. The sector address field of this command is not used to specify the location of an operand but contains an address number, N, which is determined by:

$$N)_8 = \text{The sector location of the command})_8 + S)_8 + 1)_8 .$$

Example: The command 012 3320; is located in sector 005 of line 07.

	(A)	(B)
Before execution of LRS	-2310724	76124500
After execution of LRS	XX514435	23705224

Note: XX are bit positions 0 through 3 of the A register, which are filled with the parity bit.

Timing: LRS is a class 1 command.

TAN Transfer if A Negative (35)

If the contents of the A register are negative, the computer will take its next command from the effective address, which may be in any command line. If the contents of A are not negative, the next sequential command is executed. A sequence tag of zero is required.

TBN Transfer if B Negative (36)

If the contents of the B register are negative, the computer will take its next command from the effective address, which may be in any command line. If the contents of B are not negative, the next sequential command is executed. A sequence tag of zero is required.

TCN Transfer if C Negative (34)

If the contents of the C register are negative, the computer will take its next command from the effective address, which may be in any command line. If the contents of C are not negative, the next sequential command is executed. A sequence tag of zero is required.

Timing: TAN, TBN, and TCN are class 4 commands, therefore all operate under the same timing considerations. If the register referred to is negative, the next command is read from the line and sector number specified by the command. If the register is not negative, the command directly following the transfer of control command is read. A conditional transfer, where the condition is not met, thus requires no execution time. The execution time, when control is transferred, is 12 microseconds per sector, for the interval between the transfer of control and the next command to be executed.

Usage: A sequence tag of one with either TAN, TBN, or TCN results in an unconditional transfer.

TRU Transfer Unconditionally (37)

The computer will take its next command from the specified address, which may be in any command line. For an unconditional transfer to be executed, a sequence tag of one must be present.

Timing: TRU is a class 4 command. The execution time is 12 microseconds to read the transfer command itself, plus 12 microseconds per sector for the interval between the transfer of control command and the next command to be executed. Optimum transfer location is $\alpha + 2$, where α is the location of the TRU command.

Usage: The TRU command functions as a TBN when the sequence tag of one is not present.

Starting from the right, each position of M, the effective address, is checked. If the position contains a zero, the corresponding position in A is unaffected; if the position contains a one, the corresponding position of A is changed so that it is the same as the bit written to its immediate right. The (M) are unaffected. All 22 positions of A and M take part in this operation.

Example:

	(M)	(A)
Before execution of EBP	111000111000	010101010001
After execution of EBP	111000111000	111101000001

Timing: EBP is a class 2 command.

Usage: (M) should not have a one in position 21, for this would "extend" the guard bit. This command can be used to determine the presence or absence of a one in any bit position of the A register, by extending that bit to the sign position of the A register and then performing a TAN to provide a transfer of control if there was a one in the position tested. EBP may also be used to extend a sign located in any other bit position into position 0.

The GRAY TO BINARY command sends the binary representation of a Gray-coded number in A to A. The result in A is correct only if the sign of the A register is positive. If the sign is negative, the one's complement of the result in A should be used. This command will also aid in parity tests on input data. If, after this command is given, the sign of A is negative, then A originally had an odd number of ones in bit positions 1 through 21.

Where the original bits in A are A_{21}, A_{20}, A_{19}, etc., in bit positions 21, 20, 19, etc., the GRAY TO BINARY command produces bits B_{21}, B_{20}, B_{19}, etc., in A, where

$$B_{21} = 0$$

$$\text{and} \quad B_i = 1 \text{ if } \left[\sum_{k=i+1}^{21} A_k \right] \text{is odd.} \quad 0 \leqslant i \leqslant 20$$

The theoretically correct values for the GRAY TO BINARY conversion are

$$B_0 = 0$$

$$\text{and} \quad B_i = 1 \text{ if } \left[\sum_{k=1}^{i} A_k \right] \text{is odd.} \quad 0 \leqslant i \leqslant 20$$

This command either gives the correct result for all bits or the one's complement of the correct result.

Example:

	(A)	
Before execution of GTB	00101110	(52 in Gray code)
After execution of GTB	00110100	(52 binary)

Timing: GTB is a class 2 command.

Usage: When used to check parity, an even number of ones in the A register will produce a zero in position 0 of the A register (A sign positive). An odd number of ones in the A register will produce a one in position 0 of A (A sign negative).

When used to convert Gray code to binary (a common requirement when analog information has been digitized), the GTB should always be followed by a TAN command. The address of the TAN should lead to a sequence whereby the one's complement of (A) may be found. If the (A) are positive, this need not be completed as the correct result will have been obtained.

A one is placed in each of those bit positions of B where there are ones in the corresponding positions of both C and M, the effective address. Zeros are placed in all other positions of B. (C) and (M) are not affected. All 22 positions of M, B, and C take part in this operation.

Example:

	(M)	(C)	(B)
Before execution of AMC	1100	1010	irrelevant
After execution of AMC	1100	1010	1000

Timing: This is a class 2 command. The optimum address is $\alpha + 1$; sequence tagging under these circumstances results in the next command coming from $\alpha + 2$.

Usage: This command produces the logical sum of the contents of the C register and the contents of memory, and places this logical sum in B. The most common use would be in applications requiring AND logic. An instance would be where corresponding bit positions in a group of words, each word representing elements of an ensemble, represent the presence (1) or absence (0) of a quantity. It is desired to know which quantities are present in all elements of the ensemble. This can be obtained by a series of AMC commands on the various elements (words) of the ensemble.

CLA Clear A (45)

Each bit in the A register is set to zero, including the sign position.

CLB Clear B (43)

Each bit in the B register is set to zero, including the sign position.

CLC Clear C (44)

Each bit in the C register is set to zero, including the sign position.

Timing: CLA, CLB, and CLC are class 2 commands. Although the sector address has no meaning, timing considerations for optimization require that the sector address be the next sector after the command ($\alpha + 1$), and that the command have a sequence tag of one. The next command to be executed will then be taken from $\alpha + 2$, where α is the location of the clear command. These commands effectively provide "transfer and clear" when sequence tagging is employed and the sector address of the command is $\beta - 1$, when it is desired to transfer to β.

Symbolically, this command is MC OR $\overline{M}$B, with the result appearing in B. For each one in M, the effective address, the bit in the corresponding position of C is copied into B. For each zero in M, the bit in the corresponding position of B is preserved. All 22 positions of M, B, and C take part in this operation; (M) and (C) are not affected.

Example:

	(M)	(C)	(B)
Before execution of AOC	1111 0000	11 001 0 1 0	0 1 0 1 1 1 0 0
After execution of AOC	1111 0000	11 001 0 1 0	1 1 0 0 1 1 0 0

Timing: AOC is a class 2 command.

Usage: This command effectively provides a means of inserting selected information from one word into another word. It is a convenient method of "packing" a word.

For each one in M, the effective address, a zero is put in the corresponding position in B. For each zero in M, the bit in the corresponding position of B is preserved. All 22 positions of M and B take part in this operation.

Example:

	(M)	(B)
Before execution of EXF	111000	110101
After execution of EXF	111000	000101

Timing: EXF is a class 2 command.

Usage: Selected positions of the B register may be zeroed out while all other positions are left unchanged. Sometimes a word is divided into two or more fields (groups of consecutive bit positions), where each field has a distinct meaning. This is called "packing" a word. Thus, it is possible to edit the (B) and remove (zero out) unwanted fields from a packed word.

DIU Disconnect Input Unit (50)

The Input Buffer is deactivated and all input devices are disabled from filling it. The Indicating light of the Flexowriter, if on, is turned off.

Timing: DIU is a class 2 command.

Usage: This command is used to disconnect an input device, especially a fast device, after the input is complete and before another device is activated. DIU can also be used after the computer has "waited" for a period of time and not received an input; for example, if the typewriter is activated and, after a certain period of time, no character is entered, the program can deactivate the keyboard and continue.

The Indicating light on the Flexowriter is turned on and the Input Buffer is activated to accept a character from the keyboard. After a key has been depressed, the Flexowriter sends a signal to the computer, which may be tested by a TES command having a line address of $36)_8$ to determine if the Input Buffer has been filled. Depressing a key also causes the light on the Flexowriter to go out. It is necessary to execute an RTK for each character to be read.

Timing: RTK is a modified class 2 command. Execution begins in sector $\alpha + 1$, where α is the sector location of the command, and continues through the sector specified by the command. If β is the sector address, and a sequence tag of 1 is used, the next command will come from $\beta + 1$. If a sequence tag of 0 is used, the next command will come from $\alpha + 1$.

Usage: RTK is always used when reading information from the typewriter keyboard. This information will be loaded into the buffer in 6-bit codes which may be loaded into the A register with an LAI command.

RPT Read Paper Tape (52)

This command functions exactly as RTK except that instead of turning on the keyboard light and waiting for a key to be depressed, it causes the tape reader to read one frame of tape. Sine the paper tape reader has 8 columns, as many as 8 bits per frame may be punched on it and loaded to the Input Buffer by means of the RPT command. It is necessary to execute this command for each frame of tape read.

Timing: Like RTK, RPT is a modified class 2 command which starts its execution in $\alpha + 1$ and continues through β, where α is the actual sector location of RTK and β is the sector address.

Usage: If an RPT command is given at the proper intervals, it is possible to keep the tape moving at 10 frames/second, which is the maximum input rate of the Flexowriter. (See Section IV for details on this operation.)

This command will cause the Input Buffer to be filled by a fast, special purpose unit. The PULSE TO SPECIFIED UNIT command is used to select, start, and stop these fast units. This command differs from the other read commands in that it is not self-disabling. The DISCONNECT INPUT UNITcommand must be used to terminate this operation deactivate the buffer.

Timing; RFU is modified class 2 command.*

Usage: This command may be used for fast input devices that require the Input Buffer.

* See RTK (53) (0.2-52) for explanation of modified class 2.

The capacity of the Input Buffer is any character up to eight bits. This command will load the contents of the Input Buffer into positions 14 through 21 of the A register under control of a Format Word, or "mask." Load A from Input Buffer always takes the Format Word from the specified sector and from the same line in which the LAI command is located. The sector location of the "mask" is specified by the sector address of the LAI command. Positions 0 through 13 of A may be affected if the mask contains ones in positions 0 through 13. The Format Word functions as follows: in those positions of the word where there are ones, the corresponding bit positions of the Input Buffer register are transferred to the corresponding positions of A. No other positions of A are altered. After the transfer of information to A, the Input Buffer is cleared.

Example:

	(A)	(IB)	(Mask)
Before execution of LAI	+0124000	104	+0000377
After execution of LAI	+0124104	000	+0000377

Timing: LAI is a class 2 command.

Usage: This command is always used when information is input to the PB 250 by way of the Input Buffer. Another use occurs if the mask contains all ones and is located in sector 376 of the appropriate line; if the Input Buffer has been previously cleared, zeros will be inserted in all positions of A. Selective insertion of zeros in A is possible by varying the mask, but the mask must be in sector 376 of the appropriate line.

CAM Compare A and M (56)

The contents of A (the effective address) are compared with the contents of M and, if the two are identical, the Overflow switch is turned on. If not, the Overflow switch will be turned off. In either case, the (A) and (M) are unaltered and command execution continues in the regular manner. All 22 positions of A and M are compared. The description of the TOF command should be studied in conjunction with the CAM command.

Timing: CAM is a class 2 command.

Usage: The following sequence effectively provides a transfer on zero in A:

Location	Contents	Remarks
α	CAMα + 1, S	Must be sequence tagged.
α + 1	00000 . . .	Location contains all zeros.
α + 2	TOF β	Transfer if (A) = 0, where $\beta \neq \alpha + 3$.
α + 3	- - -	Program continues here if (A) $\neq$ 0.

The eight bits of the Input Buffer are set to zero. Execution will occur during the sector address time.

Timing: CIB is a class 2 command.

Usage: This command is used when it is necessary to clear out old or unwanted information from the Input Buffer before accepting new data. The use of CIB as an in-line transfer is the same as for other clear commands. Although LAI clears the Input Buffer each time it is executed, extraneous information will get into the buffer when the sequence counter is reset to sector 0 of line 1 by the I key (I goes into the input buffer), or when single-stepping through a program by means of the C key (C goes into the input buffer). The input buffer, therefore, should be cleared prior to each use.

This command causes a single character up to eight bits to be sent to a specified output unit. The character is incorporated into the command and occupies bit positions 12 through 19 of the word; these bits are bits 12 through 14 of the op code field and bits 15 through 19 of the line number. The X in the numbered code (6X) is thus determined by the output character.

The unit to which the character is sent is specified by the command line in which the WOC command is located. Line 05 specifies the typewriter; line 06 specifies the punch; and line 00 specifies certain devices such as magnetic tape or a high-speed punch.

In order to provide the output device with a signal of sufficient duration to initiate operation, a delay number must be loaded into the C register before the execution of WOC. This number is decremented by one for each sector time after the command until the number goes negative. When the (C) go negative, the WOC command behaves as all other class 1 commands and terminates when the sector specified, β , is reached.

The signal to the output device is therefore sustained from $\alpha + 1$, where α is the location of WOC, until β , the specified sector, appears for the first time after the C register becomes negative. The (C) continue to be decremented, after they become negative, until the command terminates.

If the C register is initially negative, the output signal will be sustained only form $\alpha + 1$ to β; however, (C) will still be decremented.

Timing: WOC is a modified class 1 command and, as such, will cause the next command to be taken from the sector specified if the sequence tag is 1.

Usage: All output, except that controlled by the BSO or PTU commands, must be in the form of WOC commands. When forming WOC commands in a program, the output character is offset from the right end of the word by two bits, and the index tag is generally zero. The WOC configurations for the Flexowriter codes are as follows:

Table 2-2

FLEXOWRITER CONFIGURATIONS FOR WOC COMMANDS

Alphabetical Characters (available in both upper and lower case)		Numerical and Special Characters			Control Characters	
		Upper		Lower		
A 6101	N 6005	)	6100	0	UC	6132
B 6102	O 6006	‖	6001	1	LC	6134
C 6123	P 6027	√	6002	2	Tab	6136
D 6104	Q 6030	=	6023	3	C/R	6116
E 6125	R 6011	[	6004	4	Stop	6013
F 6126	S 6122	]	6025	5	Delete	6137
G 6107	T 6103	Ω	6026	6	Space	6020
H 6110	U 6124	&	6007	7		
I 6131	V 6105	*	6010	8		
J 6021	W 6106	(	6031	9		
K 6022	X 6127	?	6036	+		
L 6003	Y 6130	—	6037	−		
M 6024	Z 6111	:	6120	;		
		"	6033	'		
		,	6133	,		
		.	6113	.		
		/	6121	$		

PTU Pulse to Specified Unit (70)

This command produces a specified combination of signals on five output lines and an "activate" signal on a sixth line. These signals are used to start and stop equipment external to the computer. The line address of the PTU command specifies the combination of signals, while the sector address defines the first sector following execution. The activate signal is presented in the sectors between the command location and the sector address.

Timing: PTU is a class 1 command. The PTU signal will be held "on" until β comes up, where β is the sector address of the PTU command.

Usage: The following sequence of commands may be useful when desiring to hold a PTU "on" for ~ 3N milliseconds:

Location	Contents	Seq. Tag	Remarks
α	LDC α + 1	S	Initialize counter
α + 1	Count		
α + 2	LSD α + 4	S	PTU is "down" 36 μsec
α + 3	not used		each cycle
α + 4	TCN α + 6		
α + 5	PTU α + 2	S	Execute
α + 6	Continue		

Such a sequence can be used to condition the setting of relays external to the computer.

MCL Move Command Line Block (71)

The contents of the first word following the MCL command, and all subsequent words on that line up to, but not including, the address sector number, are copied into the corresponding sector positions of the effective line address.

Example: The command 010 7104; is located in 37006. When this command is executed, the information in line 6, beginning with sector 371, and continuing through sector 007, is moved to the corresponding sectors of line 4. The information which was originally in line 6, sectors 371 through 007, remains as before, but now this information has been duplicated in line 4, sectors 371 through 007.

Timing: MCL is a class 1 command. In this class of commands, the sector number of the command is used to designate the first sector number in which execution of the command is discontinued. Thus, 12 microseconds are required for reading this command, and 12 microseconds per sector transferred are required for executing this command.

Usage: This command is a convenient way of moving entire lines of information, one line at a time. By giving as the sector address $a + 1$, a complete line is moved from its original location to a new location. This method provides a convenient means of initializing subroutines in which addresses are to be modified. (Also see the MLX command, 26, in this connection.)

The BLOCK SERIAL OUTPUT command operates in a manner which is effectively the reverse of the BLOCK SERIAL INPUT (73) command. That is, the information in the data line is shifted into the External Register (ER) whenever a one appears in the Format Block. Nothing is done with information in those positions of the data line which correspond to zero bits in the Format Word. For details of this command, reference is made to the description of the BLOCK SERIAL INPUT (73) command. Computer memory and registers are unaffected by this command.

Example: The command 0125 7204; is located in 01 002. All ones are stored in 01102.

	(01104)	(ER--22 bits)
Before execution of BSO	+1215702	+0000000
After execution of BSO	+1215702	+1215702

Timing: BSO is a class 1 command. (See BSI description for further information.)

Usage: BSO can be used to provide a fast output, with format control, to an External Register.

This command loads information directly into memory at the rate of 0.5 microseconds per bit. Input information is presented to the computer in the form of a series of bits, normally from some external shift register (ER). The shifting operation in the external register must be under computer clock control. A Format Block determines when a bit will be accepted from the input device. This Format Block is formed by the binary configuration of information contained in that portion of the command line which begins with the sector following the BLOCK SERIAL INPUT command and continues up to, but not including, the sector address of the command. The information entering the computer will be loaded into the line specified by the line address of the command; it will occupy those positions of this line that correspond with one bits in the Format Block. Positions of this data line that correspond with zero bits in the Format Block will be loaded with zeros.

Example: The command 377S7305; is located in 37502. Location 37602 contains all ones. ER is the external register source from which information enters the computer.

	(37605)	(ER - - 22 bits)
Before execution of BSI	+ 0000000	+ 1234567
After execution of BSI	+ 1234567	+ 0000000

Timing: BSI is a class 1 command. The next command to be executed, when this command has a sequence tag of 1 (which it always should), will come from β, where β is the sector address. β will be the sector after the last sector of the mask.

Usage: The BSI and BSO commands provide a very fast and convenient method for communicating with an external register. In addition, formatting control is also provided. The most frequent use of these commands will come in computer systems work, where a high-speed buffer is used by the computer to communicate with equipment the computer is controlling.

TOF Transfer on Overflow (75)

An overflow results from generating a number too large for the capacity of the arithmetic registers, specifically from the ADD, SUBTRACT, DOUBLE PRECISION ADD, and DOUBLE PRECISION SUBTRACT commands. When an overflow occurs, the Overflow switch is turned on. The command COMPARE A AND M will also turn the Overflow switch on if (A) are equal to (M), but turn off the Overflow switch if this is not true. After execution of the command SQUARE ROOT, the Overflow switch is turned off.

The TRANSFER ON OVERFLOW command will cause the computer to take its next command from the specified address (if the Overflow switch is on), and then turn off the switch. If the Overflow switch is off, the next sequential command is executed and the switch remains off. Transfer may be to any sector of any command line. A sequence tag of zero is required for conditional transfer. A sequence tag of one provides an unconditional transfer and turns the Overflow switch off.

Timing: TOF is a class 4 command. Therefore, in the event a transfer is not executed, control proceeds to the next command and the total time required is the 12 microseconds required to read this command. In the event control is transferred, execution time is 12 microseconds per sector for the interval between the TOF command and the command to which control is being transferred, plus 12 microseconds to read the TOF command.

Usage: The TOF command should be studied in conjunction with the CAM command. It is the programmer's responsibility to see that the Overflow switch is off before executing a set of commands which are tested by a TOF.

TES Transfer on External Signal (77)

This command will cause the computer to take its next command from the specified address upon sensing a signal from the source external to the computer. The nature of this signal is specified by the line address of the TES command. In the standard PB 250, line addresses 25 through 37 are used to specify the following input signals:

Lines 25-30:	Arbitrary input signals.
Line 31:	High-speed punch sync. signal
Line 32:	Magnetic tape gap signal
Line 33:	Magnetic tape reader clock input signal
Line 34:	Photo tape reader sprocket input signal
Line 35:	BREAKPOINT switch input signal.
Line 36:	Typewriter or paper tape reader "character input complete" signal.
Line 37:	"Typewriter not ready for an output character" signal.

Line numbers 00 through 24 will provide additional input selectors which may be obtained as options for additional arbitrary input signals. Since the line number of the address is reserved for signal specification, the effected transfer can be only to some sector in the same line as the TRANSFER ON EXTERNAL SIGNAL command.

Example:

Location	Op Code	Address
02206	TES	02736

If a transfer is effected, the computer will take the next command from location 02706. If no transfer is effected, the next command will be executed from 02306. The sequence tag should always be zero for this command.

Timing: TES is a class 4 command. When a signal is not present, the command directly following TES command is read and the total execution time is 12 microseconds. If control is transferred, execution time is 12 microseconds,

plus 12 microseconds per sector for the interval between the TES command and the command to which control is being transferred.

Usage: Use of this command is further described in Section IV, " Input/Output Techniques." In general, the TES command acts as a " stoplight," indicating whether input/output commands should be executed or delayed. If a TES is executed which refers to an input line not physically present on the computer, the transfer will take place.

III. STANDARDS AND PROGRAMMING TECHNIQUES

3.1 PROGRAMMING TECHNIQUES

3.1.1 Introduction

There are two basic methods of programming the PB 250; relatively non-optimized, and relatively optimized. The detailed techniques and optimization rules are given for most of the commands described in Section II.

Considered as a computer without any capabilities for optimizing programs, the PB 250 still has the same command structure, and presents only the problems of any serial, binary, single-address computer. In this frame of reference, commands are generally executed from sequential sectors, at a rate of approximately three milliseconds per operation.

Partial optimization, i.e., locating the operand for class 2 commands in the next sector after the command, wherever possible, is relatively simple. For example, if a constant is needed, it is prestored in the sector after the sector for which it is required. This basic optimization greatly increases the operation speed of the machine, but does not make the most efficient use of memory. More complex optimization techniques will provide high operation speed while at the same time using memory efficiently. The programming time will be expected to increase as the complexity of techniques is increased. Although the more complex programming methods result in more efficient machine operation, a point of "diminishing returns" will be reached. After this point, more programming time will not appreciably increase either computer operation speed or efficiency of memory usage.

3.1.2 Optimization Considerations

The traditional 1 + 1 address serial computer offers a variety of possibilities for optimizing a command. If the next command cannot be placed in the optimum location (often the next section after the last operand required), then the sector one further down may be chosen, etc. On the PB 250, however, no such gradation exists. The next command is either in the optimum location (generally immediately following the operand) or it is completely unoptimized and simply follows the current command (which is in α) by appearing in $\alpha + 1$.

Paragraphs 3.2 and 3.3 describe the use of the fast line and show an example of the difference between an optimized and unoptimized PB 250 program. It is sufficient to state that the most effective way of using the fast line is as a fast access location for data frequently required during a computation, rather than as a means of storing a program to be executed. It is stressed that addresses which refer to the fast line are interpreted in exactly the same way as the addresses which refer to any of the long lines.

An important rule to remember for optimization is that memory accesses are always expensive in terms of program execution time. That is, the programmer should always think in terms of manipulating information in the A, B, or C registers, rather than storing and loading it back into these registers. Among the operations for manipulating information within the registers are the shifts (with or without affecting the C register), the register interchanges, the Rotate command, and the Merge A into C command (which can be used as a copy A into C if the C register is first cleared).

3.1.3 Special Techniques

One useful technique is the method of placing the two's complement (negative) of the (C) into A. This occurs under a one-sector multiplication, where

the B register has previously been loaded with a word whose last two bits (positions (20 and 21) are 01. All the variable length commands should be closely scrutinized by the programmer for possible special uses.

Another special technique consits of setting an internal switch by the use of RFU to turn the switch off, DIU to turn it on, and a TES $36)_8$ to determine whether the switch is on or off. Transfer will occur when the switch is on.

If additional externally operated controls are desired beyond the single BREAKPOINT switch on the Flexowriter, these may be furnished by using the surplus (unassigned) signal lines, together with external toggle switches. (See description of TES command.)

Any optimized program uses much more space in the computer than its unoptimized equivalent. However, these empty spaces do not have to be wasted. It is possible that at least one other optimized program can be interlaced with the original program in the available vacant sectors.

3.2 USE OF LINE 00

Line 00, the "fast access" line, provides fast access storage for 16 words. Any word placed in any sector of line 00 is read 16 times during each long line circulation time of 3072 microseconds. Thus, each word in line 00 is 16 times more accessible than a word stored in the long lines.

A number used repeatedly in a calculation can be stored in the fast line for ready availability. (See the Recirculation Chart in Appendix D.)

The following example illustrates the use of the fast line:

Sector	Line	Command	Remarks
023	06	024S0500;	(F04)→(A)
024		Not Used	
025		042S1406;	(A)=(F04)+(04206)
026			
027			
030			
042		Constant	
043		044S1100;	(A)→(F04)

A word is picked up from channel F04, a constant is added to it, and the sum is stored back into F04.

The programmer should be aware that optimization is possible only when reference is made to the proper sector of a channel. That is, an LDA command in 023, which is to pick up data from F04, must be sequence tagged and have a sector address of 024, not 004, 044, etc. If the sequence of commands in the previous example were written in the non-optimized modes, the execution time would be 3.072 milliseconds per command, or a total of 9.216 milliseconds. By optimization, the same computation is accomplished in 0.216 milliseconds.

Addresses referring to line 00 are not interpreted modulo $16)_{10}$, which is why the appropriate sector of a particular channel must be referenced for optimization purposes.

The fast line is extensively used in connection with such high-speed input/output devices as magnetic tape and photoelectric tape readers.

3.3 SAMPLE PROGRAMS

The sample problem may be stated as follows: Channel F03 is initially clear. X_i ($1 \leqslant i \leqslant 10$, $X \geqslant 0$) are stored in line 03, sectors 003 through 014. It is required to write a program which obtains the sum of these elements. $\left(\sum_{1}^{10} X_i\right)$ and, in addition, replaces each X_i by $\frac{X_i + 100}{4}$. Overflow will not occur. The program should halt with line address $33)_8$ and with $\sum_{1}^{10} X_i$ stored in F03.

The optimized and unoptimized programs to perform the desired function are presented on the following two pages. These two example should be studied as a contrast in techniques. The unoptimized program requires over 300 milliseconds to execute; the optimized program requires only 30 milliseconds to execute.

3.4 PROGRAMMING CONVENTIONS

Certain conventions and techniques should be followed as a program is being developed. These conventions ensure that:

a) Communications between programs is simplified.

b) Routines can be adapted to a wide variety of problems.

c) Necessary modifications can be implemented with the minimum amount of program rewrite.

pb Packard Bell Computer

PB 250 PROGRAM LISTING

PROBLEM UNOPTIMIZED SAMPLE PROGRAM PAGE 1 OF 1

PROGRAMMER R. L. HOOPER DATE 3-2-61

LOCATION	INSTRUCTION	SYMBOLIC OP CODE	REMARKS
00102	015 0502;	LDA	LOAD A
002	015 2503;	IAM	EXCHANGE A & MQ
003	014 3502;	TAN	TRANSFER IF A NEGATIVE
004	000 4400;	CLC	CLEAR C
005	006 0000;	MAC	MERGE A INTO C
006	003 1400;	ADD	ADD
007	000 0100;	IAC	1
010	016 1402;	ADD	ADD
011	014 2210;	SRT	
012	003 1000;	STC	STORE C
013	002S3702;	TRU	TRANSFER UNCONDITIONALLY
014	000 0033;	HLT	HALT.
015	-0000000		
016	+0000144		

PB 250 PROGRAM LISTING

PROBLEM Optimized Sample Program

PAGE 1 OF 1

PROGRAMMER R. L. H.

DATE 1/30/61

LOCATION	INSTRUCTION	SYMBOLIC OP CODE	REMARKS
00002	001S0502;	LDA	START
00102	-0000000		negative
00202	015S2503;	IAM	$X_i \longrightarrow A$; $(A) \longrightarrow M$
00302			not used
00402			
00502			
00602			
00702			
01002			
01102			
01202			
01302			
01402			not used
01502	017 3502;	TAN	THRU
01602	017S4400;	CLC	$0 \longrightarrow (C)$
01702	021 0033;	HLT	STOP
02002	021S0000;	MAC	$X_i \longrightarrow (C)$
02102	000 0000;		not used
02202	023S1400;	ADD	$X_i + \sum_{1}^{i-1} X_i$
02302	000 0000;		not used
02402	025S0100;	IAC	$(A) \rightleftharpoons (C)$
02502	000 0000;		not used
02602	027S1402;	ADD	$X_i + 100_{10}$
02702	+0000144		constant
03002	033S2210;	RST	$(X_i + 100)/4$
03102			not used
03202			not used
03302	043S1000;	STC	$\sum_{1}^{i} X_i \longrightarrow M$
04402	002S3702;	TRU	back to start of loop

d) Ease of understanding will be provided.

As previously described, the group of sectors in line 00 which simultaneously contain the same information, are called a channel. Line 00 channels are designated F00 through F17. For example, F00 refers to, collectively, locations 00000, 02000, 04000, 06000, 10000, 12000, 14000, 16000, 20000, 22000, 24000, 26000, 30000, 34000, and 36000.

Lines are referred to by their octal address, i.e., 00 through $77)_8$; sectors are also referred to in octal notation, i.e., 00 through $377)_8$.

Normally, the Index register, and F00 through F17, are available to any program or subroutine and must be preserved by the programmer before entering the subroutine, if these registers contain information which is to be used later in the main program.

Subroutines will generally be entered with the argument in the A register and the exit in the C register. If the argument requires two words, these words will be located in the A register and B register and the exit will be located in the C register. Subroutine exits will normally be complete instructions (unconditional transfers).

3.5 FLOW DIAGRAMMING CONVENTIONS

Flow diagrams are divided into two groups as follows:

a) Macro Flow Diagrams -- broad, descriptive flow diagrams, outlining a large, complex program. They are not oriented to the program logic but serve to provide a general picture of how the program operates, and also serve as a guide to a more detailed flow diagram.

b) Micro Flow Diagrams - - machine oriented diagrams whose functions is to define the program logic.

Table 3-1 lists the standard flow diagram symbols used in PB 250 programming. These symbols have been selected both for their convenience and universal acceptance. With the exception of the start symbol, they represent the flow chart symbols recommended for use by the Association for Computing Machinery.

Referring to the table, small English letters are used to identify fixed connectors while small Greek letters with numerical subscripts are used to identify variable connectors. To avoid possible confusion, it is recommended that the flow diagram page number be included with the connector to facilitate following the flow diagram.

To aid personnel unfamiliar with a particular program, important and significant micro flow diagram boxes are cross-referenced to the program listing by having the location (line and sector) of the first instruction executed within the respective box (in the upper right hand corner as shown below). It is emphasized that not all boxes of the flow diagram are keyed to the listing. Cross-referencing of all boxes on the flow diagram requires the performance of considerable updating by the programmer responsible for maintaining the program. In many cases, because of the auxiliary nature of this cross-referencing, the diagrams may not be kept up to date; therefore, the number of cross-reference boxes should be kept to a minimum.

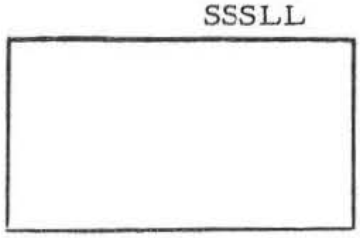
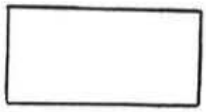
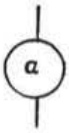
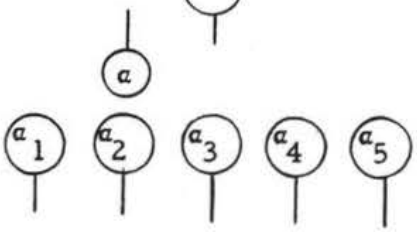
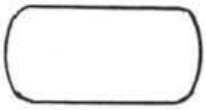
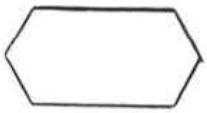
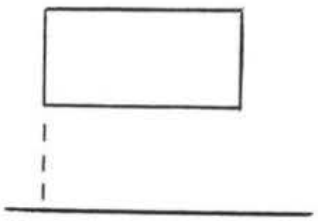


Table 3-1

STANDARD FLOW DIAGRAM SYMBOLS

SYMBOL	MEANING
	Tape (Magnetic)
	Operation, Function
	Fixed Connector
	Variable Connector
	Comparison, Test, Decision
	Closed Subroutine
	Start, Stop
	Assertion, Explanation

Care must be taken to make the flow diagram appear clear and uncluttered. This can be avoided by minimizing the number of boxes per page.

The wording appearing in the flow diagram box should be as descriptive as possible. Language contained in the micro flow diagram is more general than that contained in the listing annotations.

3.6 ANNOTATION CONVENTIONS

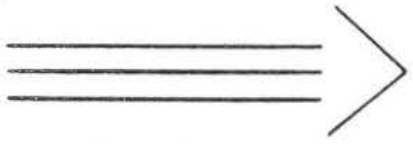
The following annotation symbols and conventions will be used:

a) ——————— Replaces: e.g., (A) + $(X)_i$ —— (A), contents of A plus contents of X_i replace contents of A.

b) ( ) Contents of: e.g., (A), contents of A register; (X_i contents of location X_1; (10002), contents of sector 100, line 02.

c) [ ] Modified Command: a command which is modified by another command within the same subroutine. Commands within a particular routine will never be modified by commands, outside that routine.

d) Brackets are used to identify all instructions included in a particular annotation, as follows:



e) The word "enter" is inserted to the left of the first instruction operated in a particular routine. The exit or exits from a routine should be clearly annotated.

f) Annotations should include the listing page number of all transfers whose locations are not included on the same page.

g) The binary point of a number is identified using Q notation, i.e., to represent an integer, N, on an annotated listing, the programmer would write: N @ 21.

3.7 AVAILABLE PB 250 PROGRAMS

Table 3-2 lists some of the standard routines which are available for the PB 250.

IV. INPUT-OUTPUT TECHNIQUES

4.1 FLEXOWRITER

A Model FL Flexowriter is used as the input-output control unit for the PB 250. The Flexowriter is also used to prepare, duplicate, and read tapes and can be used on-line (under control of computer), or off-line (under control of operation). This section is primarily concerned with the on-line mode of operation. General appearance and operations are similar to those of a standard electric typewriter. Such features as space lever, paper release lever, platen knobs, margin release lever, ribbon position lever, margin and tab stops, and type guide, are used in exactly the same manner as for a standard typewriter. See Figures 4-1, 4-2, and 4-3 for illustrations of the Flexowriter keyboard, code, and characters, respectively.

4.1.1 Input

The tape used with the Flexowriter has eight channels across its width. The keys of the typewriter, however, will only cause 6-bit codes to be punched on this tape. When punching tape under computer control, it is possible to output 8 bit of information at a time. It is desirable to utilize all eight channels on the tape wherever possible, since this reduces the number of frames of tape that must be input or output for a block of information.

When the READ TYPEWRITER KEYBOARD (RTK) command is given, the light on the front of the Flexowriter will come on and it will be possible to enter information, in the form of 6-bit codes, into the Input Buffer. Each time a key on the typewriter is depressed, the light will go off and it will be necessary to give another RTK command before another code can be entered.

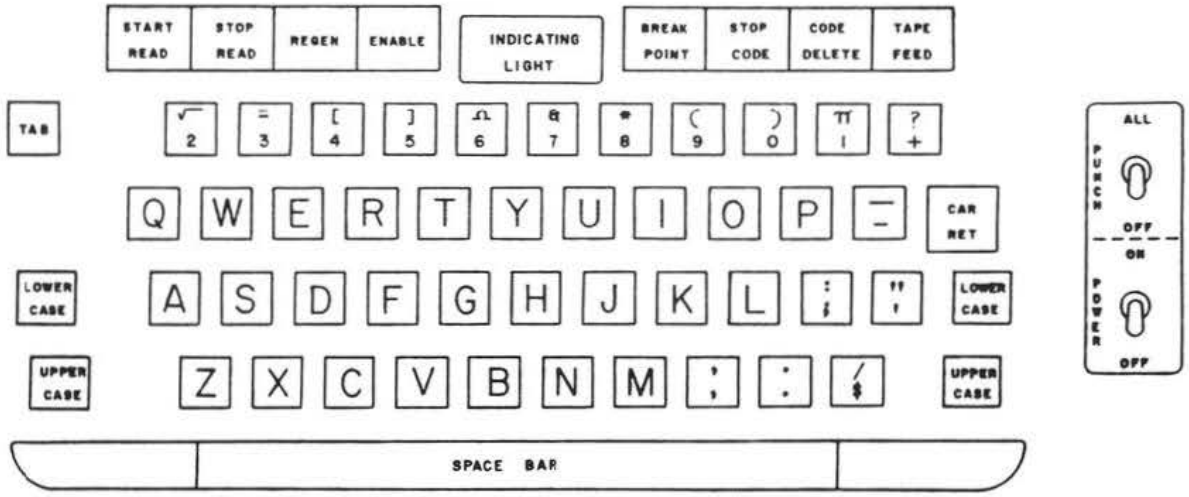


Figure 4-1. Flexowriter Keyboard

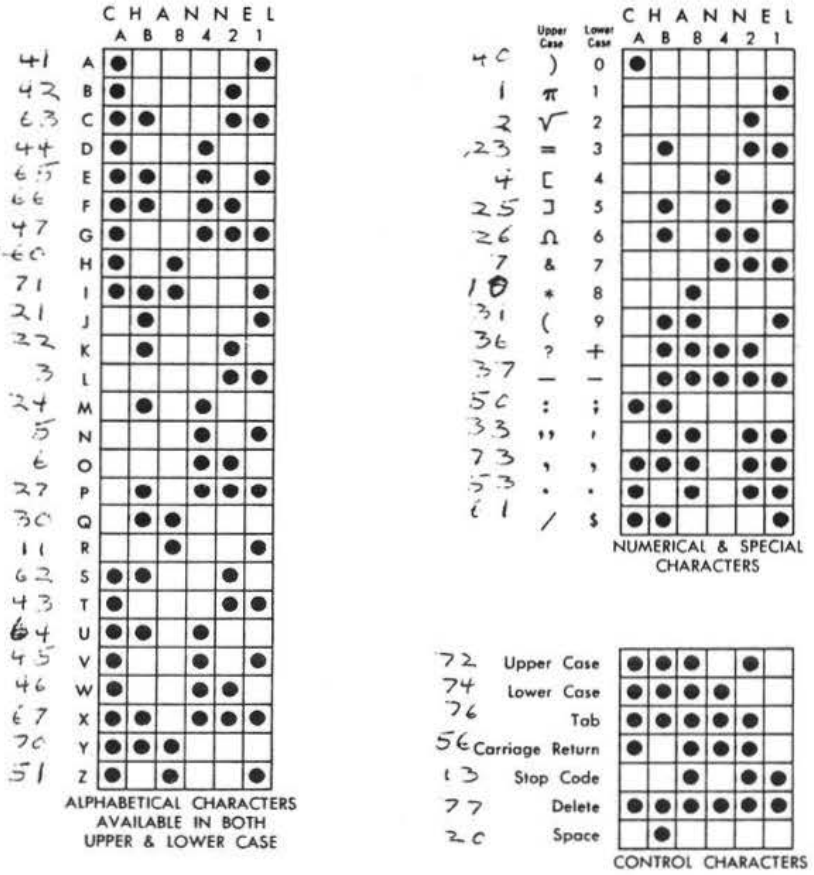


Figure 4-2. Flexowriter Code

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z π √ = [ ] Ω & * ( ) ? _ " : / . ,
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 1 2 3 4 5 6 7 8 9 0 + - ' ; $. ,

Figure 4-3. Flexowriter Characters

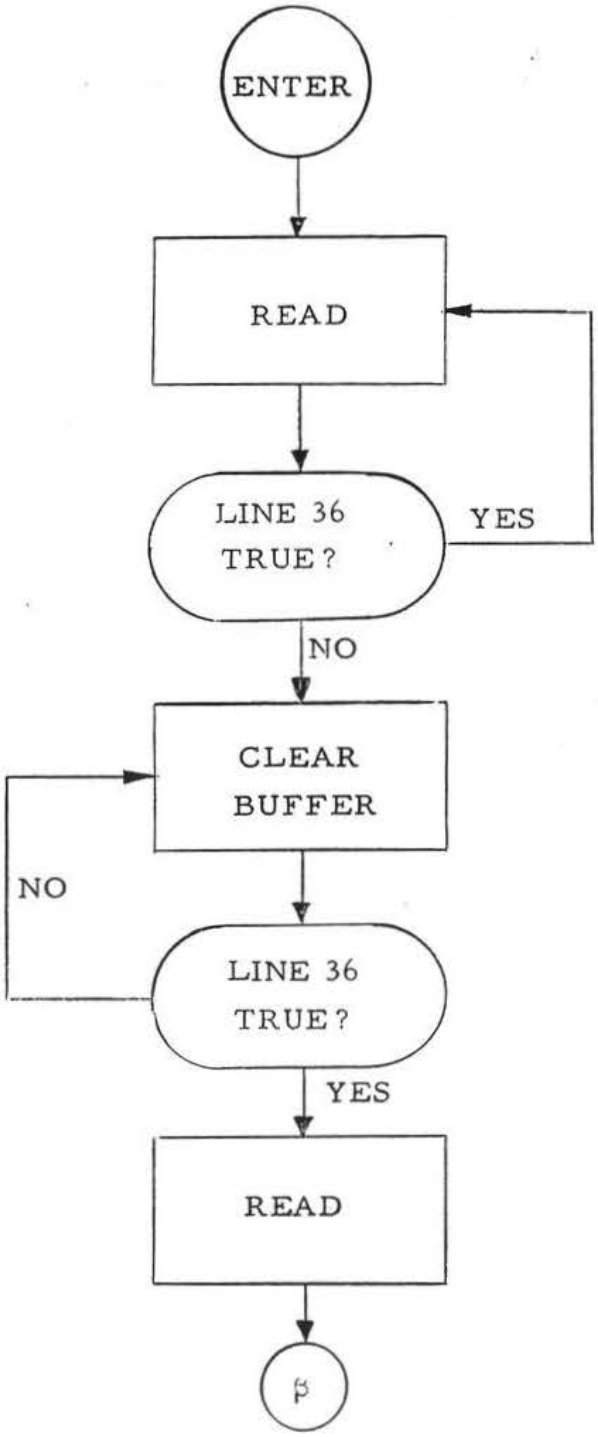
The READ PAPER TAPE command will cause the tape reader to read one frame of tape and then advance the tape one frame. Eight bits of information will be loaded into the buffer. If the tape was prepared in the PB 250 Flexowriter format, only six of these eight bits will be significant; however, if the tape was prepared by the computer, all eight bits may have significance.

When either the tape reader or the keyboard has loaded the buffer, a signal is sent to the computer, which may be sensed by a TES command having a line address of $36)_8$. This signal deactivates the Input Buffer so that it cannot be loaded with further information. Any time after either an RTK or RPT command is given, the presence of information in the buffer may be sensed by giving a TES command with a line number of $36)_8$. If the buffer has been filled, the transfer will occur.

Since the maximum speed of the Flexowriter for both the reader and the keyboard is 10 characters/second, and the PB 250 operates at microsecond speeds, it is possible for a program to be ready for another input before the Flexowriter has finished with the previous input; if a READ command were given during this time period, the same character would be read again.

To keep the Flexowriter tape reader operating at its maximum rate, and at the same time avoid reading the same character twice, a sequence of commands can be used with either the RPT or RTK commands to provide an automatic method of determining if character read-in is complete. This method proceeds by giving a READ command and then testing line $36)_8$ after only 3 ms. If line $36)_8$ is true, it can be assumed that a previous character is being read, since the Flexowriter cannot react in 3 ms. The sequence then cycles through these two commands, READ and TES $36)_8$, until the TES fails, which will occur only when the previous read-in is complete. Then, by clearing the buffer and waiting for line 36 to go true, the next READ will fill the Input Buffer with a new character.

The command sequence is illustrated in the following flow diagram (nth character to be read):



if line 36 is true, previous character is being read.

line 36 is typewriter or paper tape reader "character input complete" signal. Apparently goes false for a fixed time interval (.03 sec?) when input is completed, then true again.

The command sequence for the read operation is as follows:

Location Sector	op Code	Line Address	Sector Address	Seq Tag
β	LAI (etc)			
.	.	.	.	
.	.	.	.	
α	TRU	LL	α + 2	S
α + 1	READ	00	β - 1	S
α + 2	READ	00	α + 3	
α + 3	TES	36	α + 2	
α + 4	TES	36	α + 1	
α + 5	CIB	00	α + 3	S

The function of the sequence is as follows:

The sequence is entered at α + 2, where a READ command with sector α + 3 and no sequence tag is executed. After 3 ms, line 36 is tested and if the line is "true", control returns to the READ command. If line 36 is not "true", control will pass through to the CIB command which clears the buffer and returns control to the second TES 36. The program will wait in this TES-CIB loop until line 36 goes "true", at which time the TES 36 will transfer to the READ in α + 1. This READ will execute for the greater part of a memory circulation and then transfer control to β , the next operation. Although β is not a fixed location it should be as far from α + 1 as possible, that is α or α - 1.

4.1.2 Output

There are two ways to obtain output on the Flexowriter: the typewriter, which has a speed of 10 characters/second, and the punch, which operates up to 15 characters/second.

To type out on the typewriter, the WOC command must be located in line 05. In order to give the Flexowriter time to respond to the output signal, it is necessary to load the C register with a delay number before executing the WOC command. This number will be decremented by one for each sector of execution until it goes negative, at which time the WOC acts like a standard class 1 command. For the typewriter, a signal of 20 milliseconds duration is always sufficient; however, for some Flexowriters, less time may suffice. To obtain this delay, an octal number, + 0003232, should be loaded into the C register before execution.

In order to avoid sending an output signal before the typewriter has completed a previous character, a TES command with a line number of $37)_8$ should be used to test for "typewriter busy." Line 37 will become "true" 11-13 milliseconds after the WOC command has started, and will remain true for as long the typewriter is busy typing a character. The TES 37 command may be used to transfer back on itself, and in this way produce a one-word loop until the typewriter is ready to receive the output character.

Information output on punched paper tape is faster than output using the typewriter and is controlled in almost the same way as the typewriter, except that the WOC command is located in line 06 instead of 05. In the case of the punch, a 15-millisecond delay is always long enough to start the punching operation, instead of the 20-millisecond delay required for the typewriter. There is, however, no way to test for the punch being busy and the programmer must always allow sufficient time between characters. One method of testing is to calculate the amount of time used by the program in its operations between characters, and then to make up the remainder of the time by using a larger delay number for the WOC command. It is permissible to use a WOC for longer than 15 milliseconds, but no longer than approximately 60 milliseconds. In this way, it is possible to output a tape without the necessity of using an additional counter.

For the 15-millisecond delay, an octal number of + 0002424 should be loaded into the C register.

V. COMPUTER OPERATION AND PROGRAM CHECKOUT

5.1 COMPUTER OPERATION

The POWER button on the front panel of the computer is the only control necessary to turn the machine ON or OFF. When the computer is on, this button will be illuminated. The Flexowriter ON-OFF switch is located on the Flexowriter.

When loading a program, the Octal Utility Program, which is presented in Appendix C, should be used. This utility package simplifies control of the PB 250 during program operation and checkout.

The delay line memory of the PB 250 is erased when power is removed and, upon turning the machine on again, the contents of memory will not necessarily be all zeroes, but will be a random bit configuration. In consequence, parity halts may be generated by trying to load the A, B, or C registers with sectors in which information has not been previously stored.

5.2 PROGRAM CHECKOUT

5.2.1 Dumping and Tracing

Once a program has been coded, punched and loaded into the computer, the question still remains as to whether the program, as written, is correct. In the event that the program produces a print-out of results, these results can be compared with known results obtained by hand computation of test cases. In the event the program does not perform as predicted, several courses are open to the operator. A static dump (memory print-out) of the contents of appropriate memory locations may be made, or the program may be traced, which is a dynamic process showing the conditions of the various registers as computation proceeds.

5.2.2 Single-Step Operation

An easier approach than either dumping or tracing, is to single-step the computer through the program and, by comparing the results shown on the console lights with annotated coding sheets, find the flaw or flaws in the program. Single-stepping may be accomplished by depressing the ENABLE switch and depressing the C Key on the Flexowriter once for each program step to be executed. Note: Each time any Flexowriter key is depressed, the Input Buffer is loaded with this character, In addition, certain commands appear in the OPERATION lights as other than that which is actually being executed; these commands are as follows:

ROT (03), which shows as 01
LDP (07), which shows as 05
STD (13), which shows as 11
DPA (16), which shows as 14
DPS (17), which shows as 15

For class 1 commands, such as MUP, DIV, etc., the information displayed in the OPERAND lights will not reflect the actual line number of the command being executed.

Conditional transfer commands will not appear in the OPERATION lights unless the condition necessary for transfer is present. For example, TBN (36) will always be executed (i.e., either a transfer will take place if B is negative, or the regular instruction sequence will continue if B is not negative) but will not appear in the OPERATION lights unless the B register is negative when this command is being executed.

Within the limitations previously described, the console indicator lights may be interpreted as follows:

OPERATION lights (6) ------- Op code of command
OPERAND lights (5) ------- Line address of command
COMMAND lights (3) ------- Line location of command

Note that single-stepping through class 1 commands located in line 00 will in general give incorrect results.

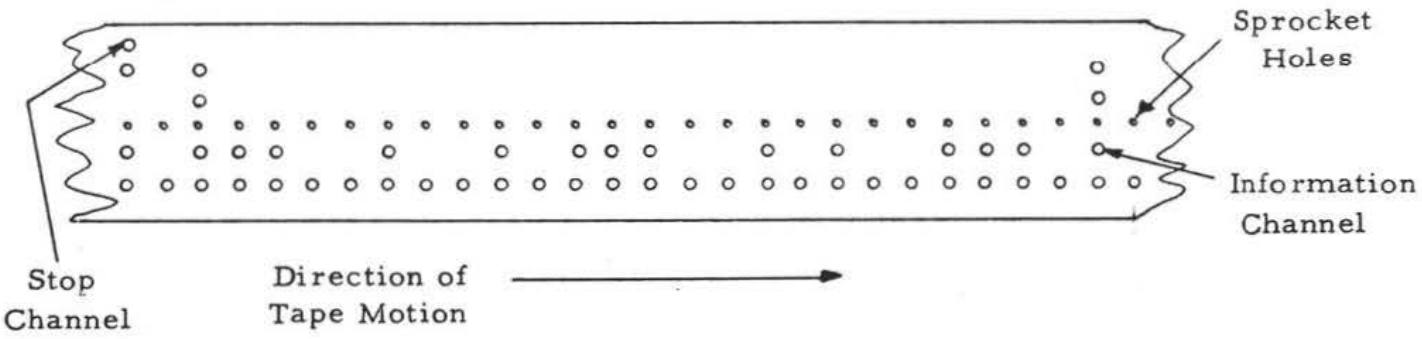
5.2.3 Use of The FILL Switch

During checkout, it may be necessary to reload the Octal Utility Package using the FILL switch. Programs other than the Octal Utility Package will be destroyed when the FILL switch is turned on if the extreme left-hand light of the OPERATION lights is illuminated. To turn this light off, single-step the computer from the Flexowriter until the light goes out. The bootstrap leader on the Octal Utility Package may then be loaded by the FILL switch without disarranging the rest of memory.

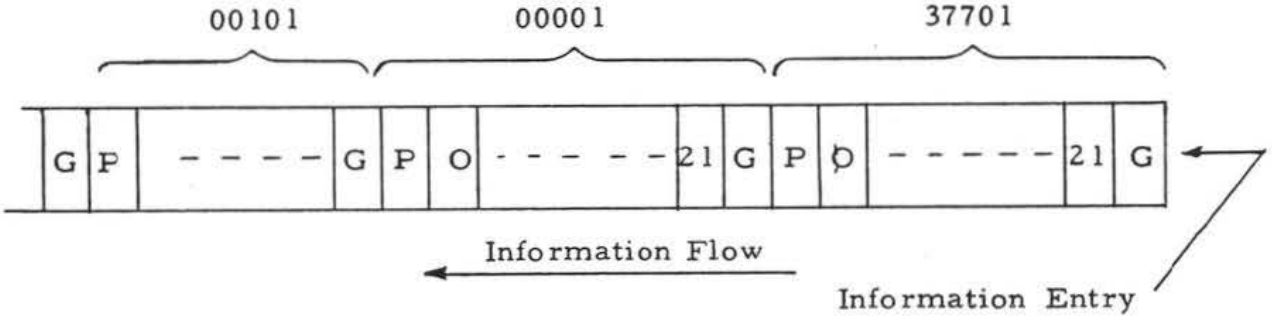
5.3 BOOTSTRAP LOADING

5.3.1 Method

When the computer is first turned on, it is necessary to load a small service routine, called a bootstrap, into the computer by turning on the FILL switch, which is located on the computer console. This bootstrap program, in turn, is used to load the Octal Utility Package which is capable of loading tapes in conventional 6-channel or 8-channel format. The bootstrap tape is a special binary information tape with the information arranged as shown in the following diagram.



Bootstrap tapes load one information bit at a time, starting with the guard bit of sector 377 of line 01. The next bit enters the guard bit of 377 and pushes the bit previously loaded, down to position 21 of 377. This continues through the parity bit of 377 and into the guard bit of 000 of line 01, as follows:



Codes on the bootstrap tape are as follows:

(Zero)	0	0
	H	1
	C/R	Guard Bit
	Stop Code	Stop Loading (After last C/R) Always preceded by a zero

For each word that is loaded, a parity bit must have been computed and punched. A stop code on the tape will cause the tape read in to cease, at which time the operator may transfer to 00001 by first turning off the FILL switch then depressing both the ENABLE and BREAKPOINT switches, striking the I key and raising the ENABLE switch.

BINARY-OCTAL NUMBERS

A. NUMERICAL SYSTEMS

Any number can be represented as the sum of a group of terms, having the form: $a_n b^n \ldots + a_3 b^3 + a_2 b^2 + a_1 b^1 + a_0 b^0$, where $b > 1$ and $0 \leq a \leq (b-1)$. The integer "b" is called the base or radix, of the particular numerical system while "a" represents the range of numerical values in that system.

1. Decimal System

The numerical system of radix 10 is called the decimal system. In this case, numerical values are specified by combining powers of ten in the form $a_n (10^n) \ldots + a_3 (10^3) + a_2 (10^2) + a_1 (10^1) + a_0 (10^0)$. The usual practice, when writing decimal numbers, is to omit the powers of ten and write out only the values of "a". For example, consider the decimal number 1875. This number actually represents $1 (10^3) + 8 (10^2) + 7 (10^1) + 5 (10^0)$ but for the sake of convenience is merely written as 1875, with the position of the particular decimal digit indicating with which power of ten the digit is associated.

2. Binary System

The PB 250 operates in the binary, or radix 2, mode; therefore, to understand the operation of the computer, an understanding of binary arithmetic is essential.

Here, numerical values are specified by combining powers of 2 in the form $a_n (2^n) \ldots + a_3 (2^3) + a_2 (2^2) + a_1 (2^1) + a_0 (2^0)$. As before, the usual practice when writing binary numbers is to omit the powers of 2 and write out only the values of the "a" terms. For example, consider the binary number 1011. This number actually represents $1 (2^3) + 0 (2^2) + 1 (2^1) + 1 (2^0)$ but for the sake of convenience is merely written as 1011, with the position of the particular binary

digit (or bit) indicating with which power of 2 the digit is associated. The only digits available in binary notation are 0 and 1.

3. Octal System

In the octal system, numbers are specified by combining powers of 8 in form $a_n(8^n) \ldots + A_3(8^3) + a_2(8^2) + a_1(8^1) + a_0(8^0)$. For the decimal and binary systems, the powers of the base (8 in this case) are omitted, and only the values of the "a" terms are written. For example, the octal number 7142 actually represents $7(8^3) + 1(8^2) + 4(8^1) + 2(8^0)$. The digits available in octal notation are 0, 1, 2, 3, 4, 5, 6, and 7.

B. RADIX CONVERSION

It is frequently necessary to convert numbers from one base, or radix, to another during programming operations. The more common conversions are described in this section.

1. Decimal-to-Binary Integer Conversion

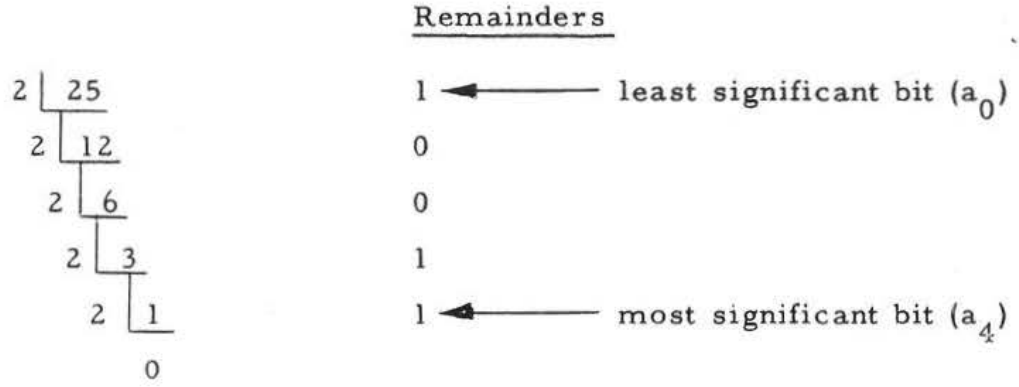
Assume it is desired to convert $25)_{10}$ to binary form. Note: The notation $)_{10}$ indicates radix 10, or decimal system; $)_8$ indicates radix 8, or octal system; $)_2$ indicates radix 2, or binary system.

a) From the definition of the general binary form, it can be seen that the decimal integer can be broken down into a summation of successive powers of 2.

$25)_{10} = 1(2^4) + 1(2^3) + 0(2^2) + 0(2^1) + 1(2^0)$

For larger decimal integers, make use of the Table of Powers of 2, in Appendix B. Note: Adding the above terms would yield 16 + 8 + 0 + 0 + 1 = 25.

b) The decimal integer can be divided repeatedly by 2; the successive remainders, when read from the end, will be the desired value.



As before, $25)_{10} = 11001)_2$

This method follows from the fact that when converting an integer, N, to the form $N = a_n 2^n + \ldots a_1 2^1 + a_0 2^0$, the remainder, when N is divided by 2, is a_0; dividing this first quotient by 2 yields a_1 as a remainder, etc.

2. Binary-to-Decimal Integer Conversion

Assume it is desired to convert $11110)_2$ to decimal form:

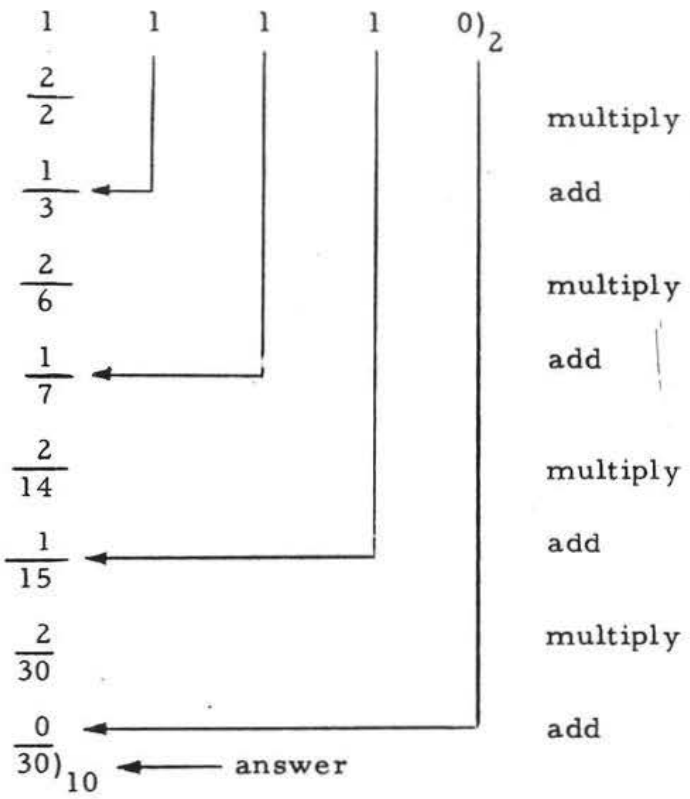
a) The values of the powers of 2 can be summed up to give the decimal equivalent.

$$11110)_2 = 1(2^4) + 1\ (2^3) + 1\ (2^2) + 1\ (2^1) + 0\ (2^0)$$

$$= 16 + 8 + 4 + 2 + 0 = 30)_{10}$$

Therefore, $11110)_2 = 30)_{10}$

b) A second method is to multiply the most significant bit by 2, add the next most significant bit, multiply the resulting sum by 2, add the next most significant bit, etc.



As before, $11110)_2 = 30)_{10}$

This method follows from factoring the general binary term for a 5-bit number to obtain the form

$$N = a_0 + 2(a_1 + 2\,(a_2 + 2(a_3 + 2(a_4\,))))$$

Evaluating N, starting at the inner parentheses, gives the required decimal integer.

3. Decimal-to-Octal Integer Conversion

To convert a decimal integer to octal form, divide the number repeatedly by 8; the successive remainders, when read from the end, will be the desired octal value.

For example, convert $75)_{10}$ to octal.

```
8 | 75        3 <---- least significant digit
  8 | 9       1
    8 | 1     1 <---- most significant digit
      8 | 0
```

Therefore, $75)_{10} = 11\ 3)_8$

This method follows from the fact that when an integer, N, is converted to the form $N = a_n\ 8^n + \ldots\ a_1\ 8^1 + a_0\ 8^0$, the remainder, when N is divided by 8, is a_0; dividing this first quotient by 8 yields a_1 as a remainder, etc.

Note: It is usually convenient for the programmer to refer to the Octal-Decimal Integer Conversion Table, Appendix B, when converting integers from decimal to octal and vice-versa. The use of this table is self-evident.

4. Octal-to-Decimal Integer Conversion

To convert an octal integer to decimal form, multiply the most significant digit of the number by 8, add the next most significant digit, multiply the resulting sum by 8, add the next most significant digit, etc. For example, convert $155)_8$ to decimal.

```
   1      5      5)8
   8      |      |          multiply
  ---     |      |
   8      |      |
   5 <----+      |          add
  ---            |
  13             |
   8             |          multiply
  ---            |
 104             |
   5 <-----------+          add
  ---
 109)10 <-- Answer
```

Therefore, $155)_8 = 109)_{10}$

This method follows from factoring the general octal term (for a 3-digit number) to obtain

$$N = a_0 + + 8\,(a_1 + 8\,(a_2\,))$$

Evaluating N, starting at the inner parentheses gives the required decimal integer.

5. Binary and Octal Number Relationships

Since $2^3 = 8$, it can be seen that three binary digits are represented by one octal digit. This applies for fractional quantities as well as for integers.

The binary-to-octal conversion is performed by grouping the binary number into 3-bit units, starting from the binary point, and interpreting each unit individually. For instance, $101011010)_2$

becomes $\underbrace{1\,0\,1}_{5}\ \ \underbrace{0\,1\,1}_{3}\ \ \underbrace{0\,1\,0}_{2}$ or $532)_8$

and $0.110111)_2$

becomes $\underbrace{.110}_{6}\ \ \underbrace{111}_{7}$ or $.67)_8$

Conversely, it can be seen that any octal number can be converted to binary by writing the binary equivalent of each octal digit. For example, $612)_8$

becomes $\underbrace{6}_{1\,1\,0}\ \ \underbrace{1}_{0\,0\,1}\ \ \underbrace{2}_{0\,1\,0}$ or $110001010)_2$

6. Decimal Fractions to Octal or Binary

Keeping in mind that the general term for a fraction, base b, is

$$a_{-1}\, b^{-1} + a_{-2}\, b^{-2} + a_{-3}\, b^{-3} + \text{----}$$

it is evident that multiplying by the base, b, will produce the a_{-1} term in the units position (immediately to the left of the radix point). Successive multiplication by the base will successively isolate the a_{-2} term, a_{-3} term, etc.

By this process, a decimal fraction, D, can be converted to the octal form $D = a_{-1}\, 8^{-1} + a_{-2}\, 8^{-2} + a_{-3}\, 8^{-3} + \text{----}$, or to the binary form

$$D = a_{-1}\, 2^{-1} + a_{-2}\, 2^{-2} + a_{-3}\, 2^{-3} + \text{----}$$

Note: A fraction in one base will not usually transform to a finite fraction in another base.

For example, to transform $0.725)_{10}$ into a binary fraction, multiply the fraction successively by 2, isolating the units position after each multiplication, until the desired number of bits are generated.

a_{-1} term ⟶

.725

2 | 1.450

2 | 0.900

2 | 1.800

2 | 1.600

Therefore $.725)_{10} = .1011\text{ ---})_2$

To convert $.082)_{10}$ to octal, multiply the fraction successively by 8, isolating the units position after each multiplication, until the desired number of octal digits are generated.

$$
\begin{array}{lr}
 & .082 \\
 & \underline{8} \\
a_{-1}\ \text{term} \longrightarrow & \boxed{0}.656 \\
 & \underline{8} \\
 & \boxed{5}.248 \\
 & \underline{8} \\
 & \boxed{1}.984
\end{array}
$$

Therefore $.082)_{10} = .0517\text{---})_8$

The Octal-Decimal Fraction Conversion Table, Appendix B, is useful for decimal-to-octal or octal-to-decimal fractional conversions.

7. Binary or Octal Fractions to Decimal

Remembering the general notation for a fraction, it is evident that a binary fraction can be converted to decimal by adding up the negative powers of 2, referring to the Table of Powers of 2, Appendix B.

For example, convert $.101)_2$ to decimal

This fraction equals $1\ (2^{-1}) + 0\ (2^{-2}) + 1\ (2^{-3})$

Therefore, $.101)_2 = .625)_{10}$

It is also possible to convert the binary fraction to octal and look up the corresponding decimal value in the Octal-Decimal Fraction Conversion Table.

In the above example, $.101)_2 = .5)_8$

From the table, $.05)_8 = .078125)_{10}$

Multiplying both sides by 8: $.5)_8 = .078125 \times 8)_{10} = .625)_{10}$

C. BINARY COMPLEMENTARY ARITHMETIC

Certain computer operations, such as subtraction or the manipulation

of negative numbers, are performed in the computer by using the complement of the particular number. An understanding of complementary arithmetic is therefore important as an aid in understanding computer operation.

The 1's complement of a binary number is defined as the number that must be added to the original number to give a result consisting of all 1's. The 1's complement is obtained by simply inverting, i.e., by changing all 1's to 0's and changing all 0's to 1's in the given binary number. For example, the 1's complement of 1010110 would 0101001.

The 2's (or "true") complement of a binary number is formed by first finding the 1's complement of the number and then adding 1 to the least significant bit position.

For example, the 2's complement of 1010110 would be the 1's complement (0101001) plus 1, or 0101010.

Some examples are given on the following page in decimal, binary and complemented binary forms. The complemented binary form has a leading 0 to indicate positive numbers, which becomes a leading 1 when complemented for negative numbers. A negative answer appears in complemented form with a leading 1.

Note that in 2's complement a number plus its negative gives zero. This is not true in 1's complement.

	Decimal	Binary		2's Complemented Binary
a)	+12	+1100		0 1100
	-04	-0100		1 1100
	+08	-1000		0 1000
			1's comp.	
b)	+10	+1010	0 1010	0 1010
	-10	-1010	1 0101	1 0110
	+00	+0000	1 1 1 1 1	0 0000
c)	+12	+1100		0 1100
	-14	-1110		1 0010
	-02	-0010		1 1110

Table of Powers of 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125

Octal-Decimal Integer Conversion Table

0000 to 0777 (Octal)	0000 to 0511 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

1000 to 1777 (Octal)	0512 to 1023 (Decimal)

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

Octal-Decimal Integer Conversion Table

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

2000	1024
to	to
2777	1535
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

3000	1536
to	to
3777	2047
(Octal)	(Decimal)

Octal-Decimal Integer Conversion Table

4000 (Octal)	2048 (Decimal)
to	to
4777 (Octal)	2559 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

5000 (Octal)	2560 (Decimal)
to	to
5777 (Octal)	3071 (Decimal)

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

Octal-Decimal Integer Conversion Table

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

Octal	Decimal
6000 to 6777	3072 to 3583

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

Octal	Decimal
7000 to 7777	3584 to 4095

Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000850	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949