

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

Section	Page
PREFACE	
I GENERAL PB 250 CHARACTERISTICS	
1.1 Introduction	1-1
1.2 Memory Organization	1-1
1.3 Command Word Configuration	1-5
1.4 Command Sequencing and Timing	1-6
1.5 Parity Check	1-9
II PB 250 COMMANDS	
2.1 General	2-1
III STANDARDS AND PROGRAMMING TECHNIQUES	
3.1 Programming Techniques	3-1
3.2 Use of Line 00	3-3
3.3 Sample Programs	3-5
3.4 Programming Conventions	3-5
3.5 Flow Diagramming Conventions	3-8
3.6 Annotation Conventions	3-11
3.7 Available PB 250 Programs	3-12
IV INPUT-OUTPUT TECHNIQUES	
4.1 Flexowriter	4-1
V COMPUTER OPERATION AND PROGRAM CHECKOUT	
5.1 Computer Operation	5-1
5.2 Program Checkout	5-1
5.3 Bootstrap Loading	5-3

APPENDICES

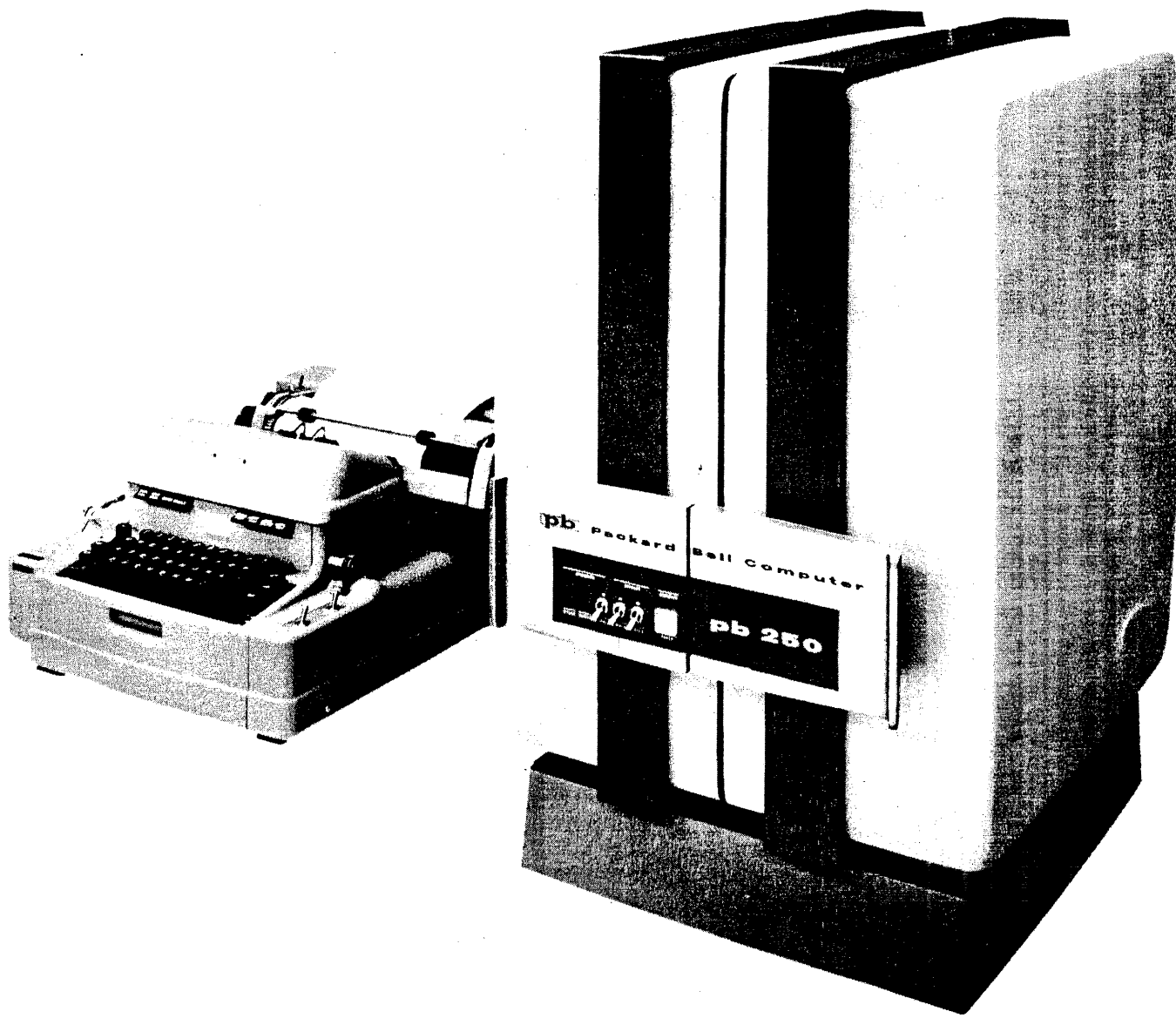
	Page
APPENDIX A: Binary-Octal Numbers.....	A-1
APPENDIX B: Numerical Conversion Tables.....	B-1
APPENDIX C: Octal Utility Program	C-1
APPENDIX D: Recirculation Chart.....	D-1

ILLUSTRATIONS

Figure		Page
1-1	Data Word Configuration	1-2
1-2	Index Register.....	1-4
1-3	Input Buffer.....	1-5
1-4	Command Word Configuration.....	1-5
1-5	Typical Command Word.....	1-6
4-1	Flexowriter Keyboard.....	4-2
4-2	Flexowriter Code.....	4-2
4-3	Flexowriter Characters.....	4-2

TABLES

Table		Page
1-1	Command Classification.....	1-10
2-1	Division Correction.....	2-35
2-2	Flexowriter Configurations for WOC Commands.....	2-59
3-1	Standard Flow Diagram Symbols.....	3-10
3-2	Summary of Available PB 250 Programs.....	3-13



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 re-writing the stored information as it is read, information continuously 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 (SSLL), 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 07; 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.

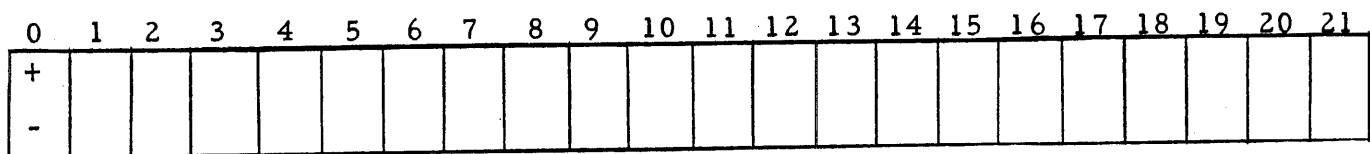


Figure 1-1. Data Word Configuration

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 it 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.

0	7 8	15 16	21
Operand Sector Counter	Sector Counter For Next Command	Index Register	

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.

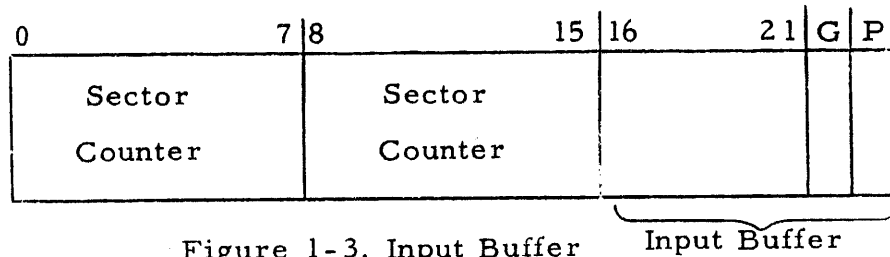


Figure 1-3. Input Buffer

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.

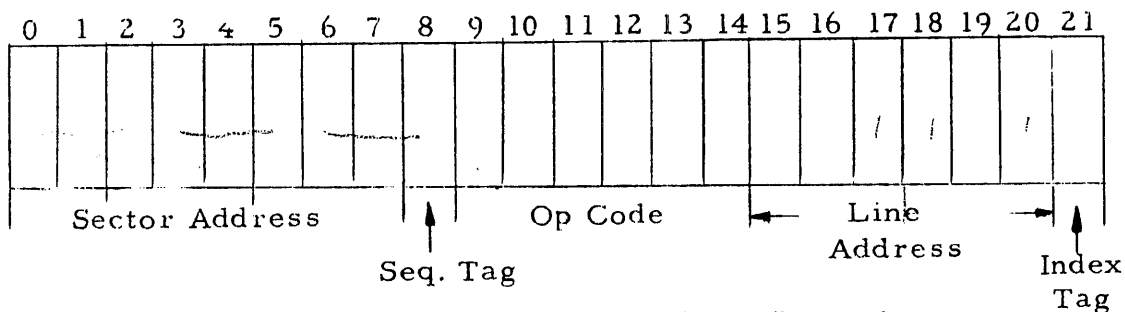


Figure 1-4. Command Word Configuration

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.

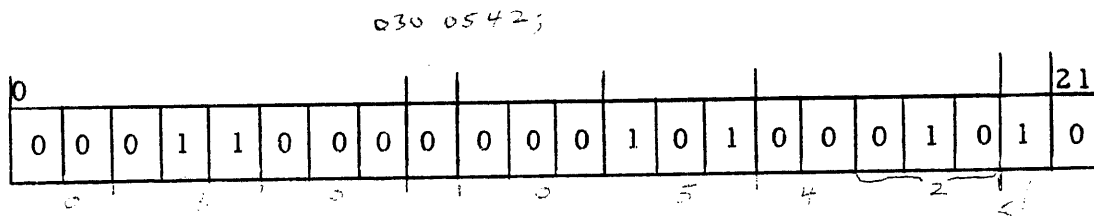


Figure 1-5. Typical Command Word

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	(80)*

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 SUBTRACT	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

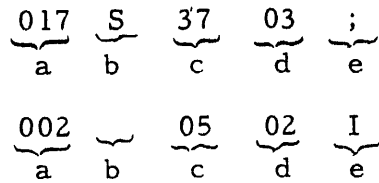
2.1.1 Command Structure

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 ($000 \leq SSS \leq 377$).
- 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 ($0 \leq LL \leq 77$).
- 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:



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.

00	HLT	halt	31	DVR	divide remainder
00	MAC	merge A into C	32	MUP	multiply
01	IAC	interchange A+C	33	SBR	shift B right
02	IBC	" B+C	33	LRS	logical right shift
03	ROT	rotate A, B, and C	34	TCN	transfer if C negative
04	LDC	load C	35	TAN	" " A "
05	LDA	load A	36	TBN	" " B "
06	LDB	load B	S37	TRU	transfer unconditionally
07	LDP	load double precision	40	EBP	extend bit pattern
10	STC	store C	41	GTB	gray to binary
11	STA	store A	42	AMC	and M and C
12	STB	store B	43	CLB	clear B
13	STD	store double precision	44	CLC	clear C
14	ADD	add	45	CLA	clear A
15	SUB	subtract	46	AOC	and or combined
16	DPA	double precision add	47	EXF	extract field
17	DPS	" " subtract	50	DIU	disconnect input unit
20	NAD	normalize and decrement	51	RTK	read typewriter keyboard
20	NOR	normalize	52	RPT	read paper tape
21	LSD	left shift + decrement	53	RFU	read fast unit
21	SLT	shift left	55	LAI	load A from input buffer
22	RSI	right shift and increment	56	CAM	compare A + M
22	SRT	shift right	57	CIB	clear input buffer
23	SAI	scale right + increment	6X	WOC	write output character
24	NOP	no operation	70	PTU	pulse to specified unit
25	IAM	interchange A + M	71	MCL	move command line block
26	MLX	move line X to line 7	72	BSO	block serial output
30	SQR	square root	73	BSI	block serial input
31	DIV	divide	75	TDF	transfer on overflow
			77	TES	transfer on external signal

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.

This command is a special case of HALT (00). If a HALT command is given which has as its sector address $a + 1$, where a 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 $a + 1$. If the sequence tag is 1, the next command executed will be in $a + 2$; whereas, if the MAC is not sequenced, the next command follows from sector $a + 1$. 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.

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 $a + 2$.

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.)

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 $a + 1$ is required, where a 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 $a + 3$. 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 ($a + 1$), 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 $a + 2$, where a 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 $a + 1$ and $a + 2$, where LDP is located in a , in any line. Sequence tagging under these circumstances results in the next command being executed from $a + 3$.

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 ($a + 1$), 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 $a + 2$, where a 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.

ADD

Add

(14)

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 $a + 2$, where a 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.

SUB

Subtract

(15)

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 (1110110010100001100001).	(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 $\alpha + 3$, 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

DPA

Double Precision Add (cont.)

(16)

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.

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 $53)_8$

NAD

Normalize and Decrement (cont.)

(20)*

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.

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 20 10; 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 $a + 1$, where a 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: $S \geq 53)_8$ 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.

RSI

Right Shift And Increment

(22)*

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.)

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) $(C) \geq 0$
- 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 ≥ 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).

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: a is the sector location of the IAM command, but is not necessarily in the same line as $a + 1$, $a + 2$, etc., and $a + N + 1$ is the IAM sector address.

<u>Location</u>	<u>Initial Contents</u>	<u>After 1st IAM</u>	<u>After 2nd IAM</u>
A register	X_a	X_n	X_{n-1}
$a + 1$	X_1	X_a	X_n
$a + 2$	X_2	X_1	X_a
.	.	.	.
.	.	.	.
.	.	.	.
$a + N - 1$	X_{n-1}	X_{n-2}	X_{n-3}
$a + N$	X_n	X_{n-1}	X_{n-2}

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.

SQR

Square Root

(30)

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 execution 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

SQR

Square Root (cont.)

(30)

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

May 2 1968
4B
1000
(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:

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

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

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

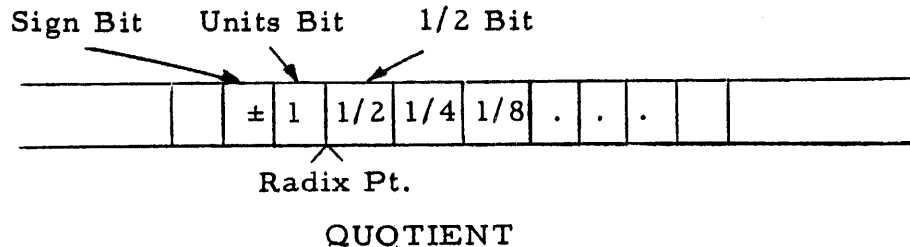
This is a divide with S = 22. The last bit of A is the sign of the quotient, which is negative. In canonical form, the quotient is -0000000, and the remainder is +0000000.

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 [a - b + 22 - S].

2) The machine remainder is scaled at $Q \cdot b^{-1}$. 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.



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~~^{one}. 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 2-1DIVISION 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 ²¹~~22~~ 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:

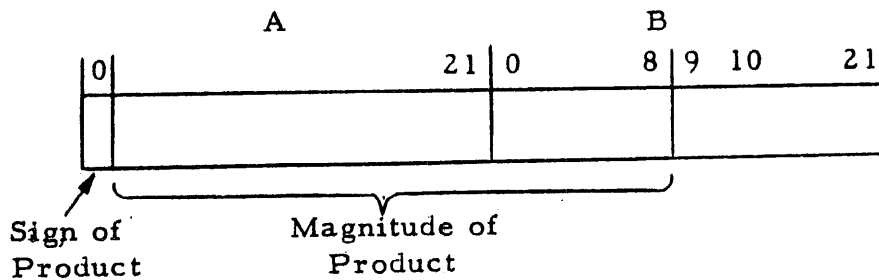
$$N)_8 = \text{Sector number of the command})_8 + S)_8 + 1)_8 .$$

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 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 $S \geq 2$, zeros are shifted into B after sector time $a + 1$, where a is the location of the SBR 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.

EBP

Extend Bit Pattern

(40)

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

$$B_{21} = 0$$

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

The theoretically correct values for the GRAY TO BINARY conversion are

$$B_0 = 0$$

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

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 $a + 1$; sequence tagging under these circumstances results in the next command coming from $a + 2$.

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 ($a + 1$), and that the command have a sequence tag of one. The next command to be executed will then be taken from $a + 2$, where a 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 $MC \text{ OR } \overline{MB}$, 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	11 001 010	01011100
After execution of AOC	1111 0000	11 001 010	11001100

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.

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.

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. Since 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.)

RFU

Read Fast Unit

(53)

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 UNIT command 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) (p. 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.

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:

<u>Location</u>	<u>Contents</u>	<u>Remarks</u>
a	CAM_{a+1}, S	Must be sequence tagged.
$a + 1$	00000 . . .	Location contains all zeros.
$a + 2$	TOF β	Transfer if (A) = 0, where $\beta \neq a + 3$.
$a + 3$	- - -	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.

WOC

Write Output Character

(6X)

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 $a + 1$, where a 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 from $a + 1$ 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.

WOC

Write Output Character (cont.)

(6X)

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	\$		

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 $\sim 3N$ milliseconds:

<u>Location</u>	<u>Contents</u>	<u>Seq. Tag</u>	<u>Remarks</u>
a	LDC $a + 1$	S	Initialize counter
$a + 1$	Count	}	
$a + 2$	LSD $a + 4$	S	PTU is "down" 36 μ sec each cycle
$a + 3$	not used	}	
$a + 4$	TCN $a + 6$	}	
$a + 5$	PTU $a + 2$	S	Execute
$a + 6$	Continue	}	

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

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 01257204; 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.

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.

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:

<u>Location</u>	<u>Op Code</u>	<u>Address</u>
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 consists 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:

<u>Sector</u>	<u>Line</u>	<u>Command</u>	<u>Remarks</u>
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 ($1 \leq i \leq 10$, $X \geq 0$) are stored in line 03, sectors 003 through 014. It is required to write a program which obtains the sum of these elements.

$\left(\sum_{i=1}^{10} X_i \right)$ 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 $\sum_{i=1}^{10} X_i$ stored in F03.

The optimized and unoptimized programs to perform the desired function are presented on the following two pages. These two examples 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 250 PROGRAM LISTING

PROBLEM Optimized Sample Program

PAGE 1 OF 1

PROGRAMMER R. L. H.

DATE 1/30/61

LOCATION	INSTRUCTION	SYMBOLIC OF CODE	REMARKS
00002	001S0502;	LDA	START
00102	-0000000		negative
00202	015S2503;	IAM	$X_i \rightarrow A; (A) \rightarrow M$
00302			not used
00402			
00502			
00602			
00702			
01002			
01102			
01202			
01302			
01402			
01502	017 3502;	TAN	
01602	017S4400;	CLC	$0 \rightarrow (C)$
01702	021 0033;	HLT	STOP
02002	021S0000;	MAC	$X_i \rightarrow (C)$
02102	000 0000;		not used
02202	023S1400;	ADD	$X_i + \sum_{1}^{i-1} X_i$
02302	000 0000;		not used
02402	025S0100;	IAC	$(A) \rightleftarrows (C)$
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	$\sum_{1}^i X_i \rightarrow M$
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.

SSLL

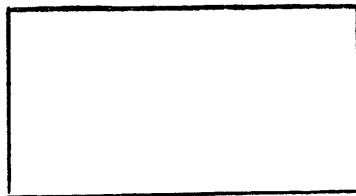

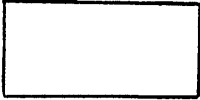
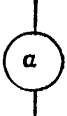
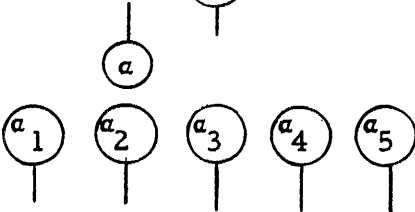



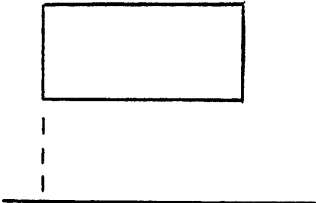


Table 3-1

STANDARD FLOW DIAGRAM SYMBOLS




<u>SYMBOL</u>	<u>MEANING</u>
	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.

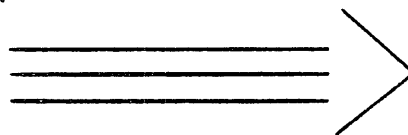
The wording appearing in the flow diagram box should be as descriptive as possible. Language contained in the micro flow diagram is more general than that contained in the listing annotations.

3.6 ANNOTATION CONVENTIONS

The following annotation symbols and conventions will be used:

- a)  Replaces: e.g., $(A) + (X)_i \text{---} (A)$, 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_i ; (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.

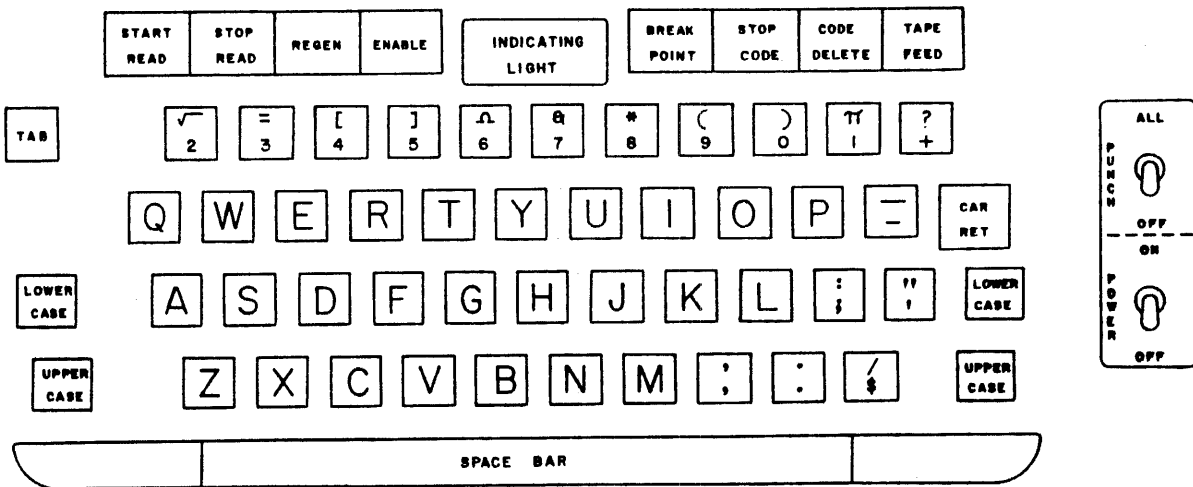


Figure 4-1. Flexowriter Keyboard

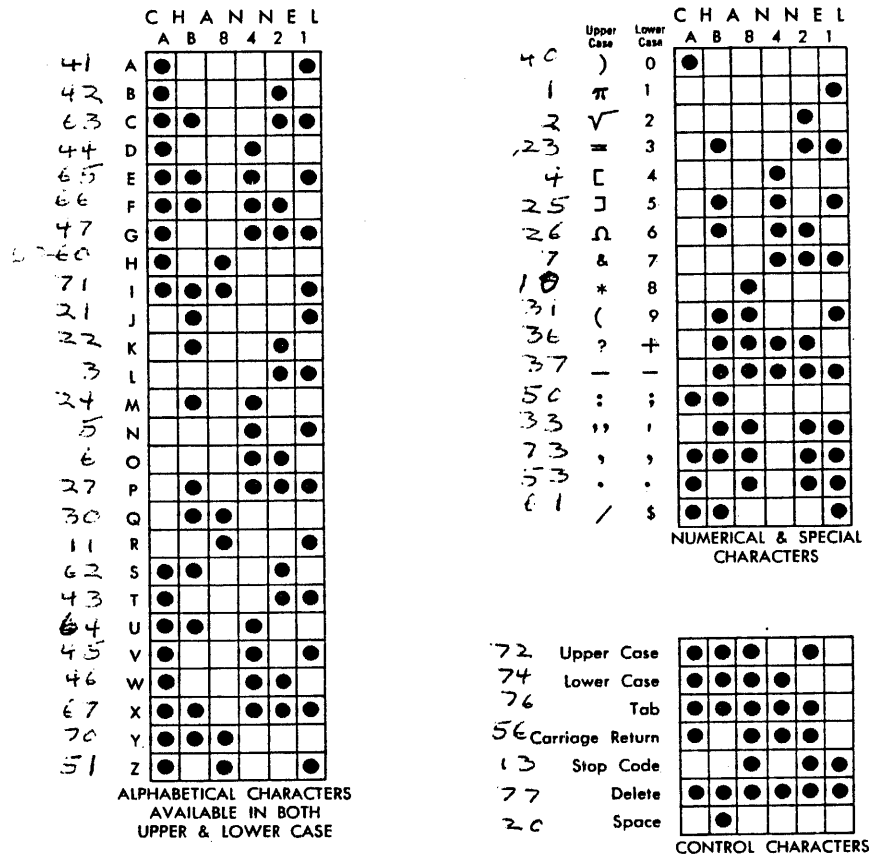


Figure 4-2. Flexowriter Code

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.

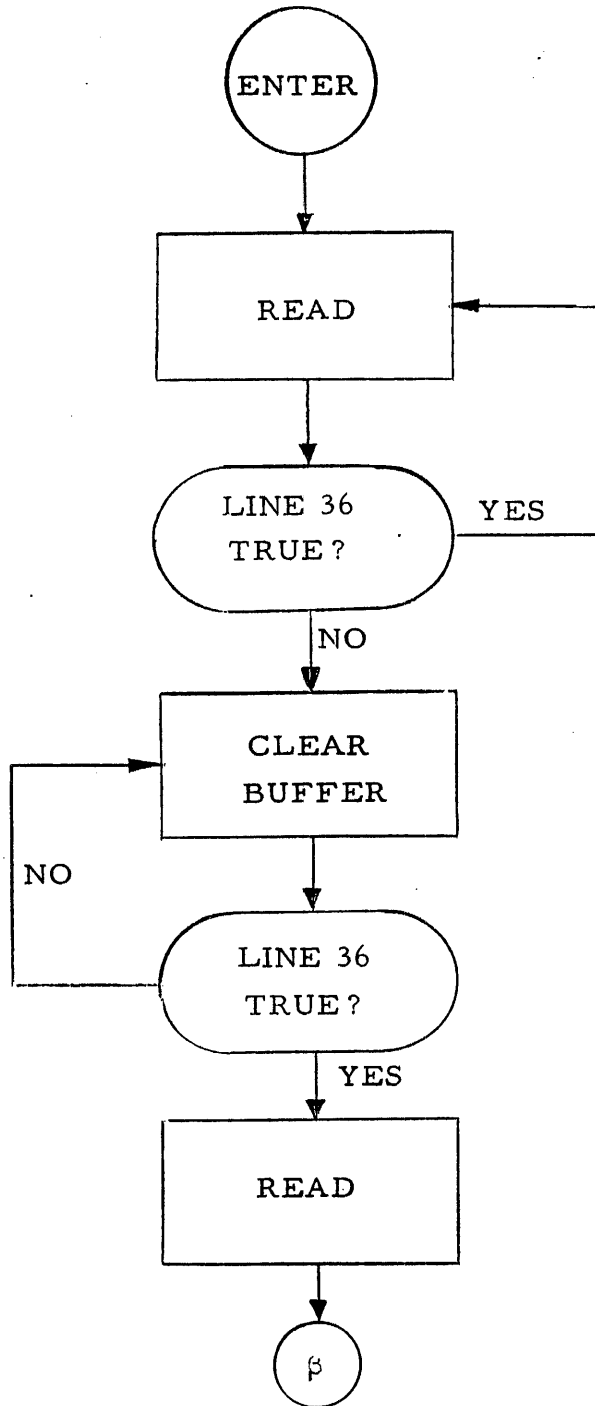
When either the tape reader or the keyboard has loaded the buffer, a signal is sent to the computer, which may be sensed by a TES command having a line address of $36)_8$. This signal deactivates the Input Buffer so that it cannot be loaded with further information. Any time after either an RTK or RPT 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	op	Line	Sector	Seq
<u>Sector</u>	<u>Code</u>	<u>Address</u>	<u>Address</u>	<u>Tag</u>
β	LAI (etc)	.	.	
.	.	.	.	
a	TRU	LL	$a + 2$	S
$a + 1$	READ	00	$\beta - 1$	S
$a + 2$	READ	00	$a + 3$	
$a + 3$	TES	36	$a + 2$	
$a + 4$	TES	36	$a + 1$	
$a + 5$	CIB	00	$a + 3$	S

The function of the sequence is as follows:

The sequence is entered at $a + 2$, where a READ command with sector $a + 3$ 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 $a + 1$. 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 $a + 1$ as possible, that is a or $a - 1$.

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.

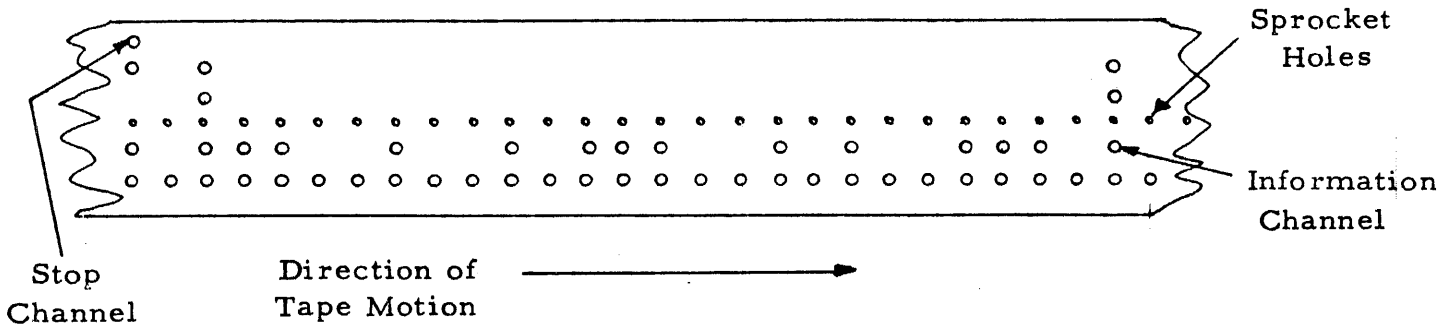
5.2.3 Use of The FILL Switch

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.

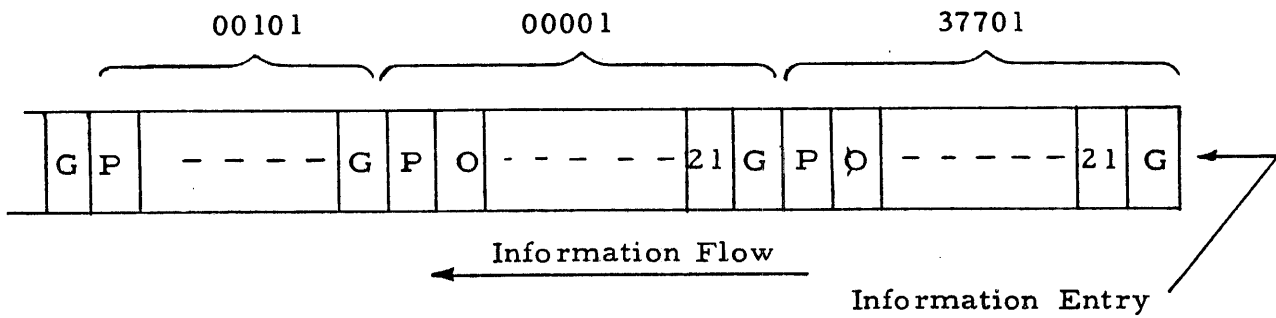
5.3 BOOTSTRAP LOADING

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 $a_n b^n + a_{n-1} b^{n-1} + a_2 b^2 + a_1 b^1 + a_0 b^0$, 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 $a_n (10^n) + a_{n-1} (10^{n-1}) + a_2 (10^2) + a_1 (10^1) + a_0 (10^0)$. 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 $a_n (2^n) + a_{n-1} (2^{n-1}) + a_2 (2^2) + a_1 (2^1) + a_0 (2^0)$. 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 $a_n(8^n) \dots + A_3(8^3) + a_2(8^2) + a_1(8^1) + a_0(8^0)$. 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.

	<u>Remainders</u>
2 25	1 ← least significant bit (a_0)
2 12	0
2 6	0
2 3	1
2 1	1 ← most significant bit (a_4)
0	

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

This method follows from the fact that when converting an integer, N , to the form $N = a_n 2^n + \dots + a_1 2^1 + a_0 2^0$, 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

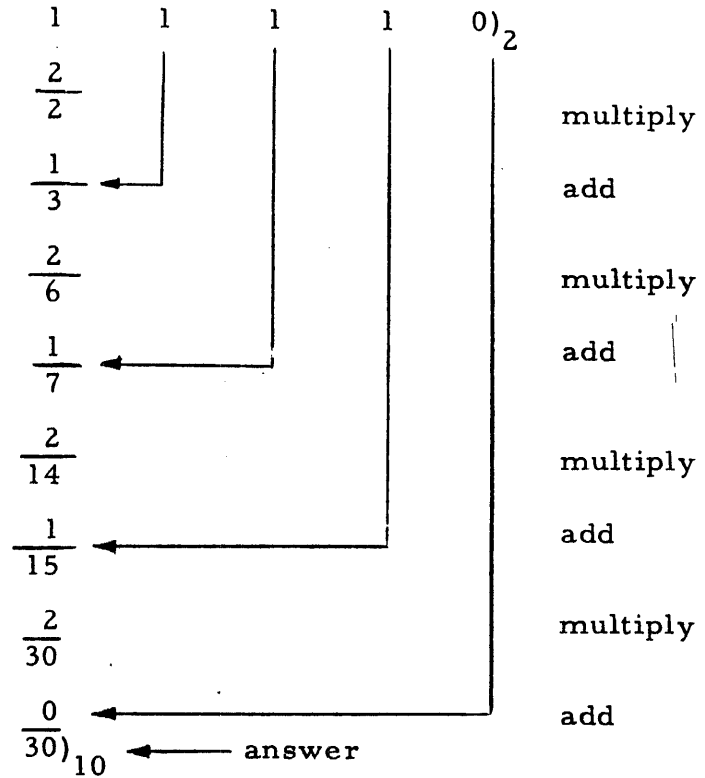
Assume it is desired to convert $11110)_2$ to decimal form:

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

$$\begin{aligned} 11110)_2 &= 1(2^4) + 1(2^3) + 1(2^2) + 1(2^1) + 0(2^0) \\ &= 16 + 8 + 4 + 2 + 0 = 30)_{10} \end{aligned}$$

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.

$$\begin{array}{r}
 8 \overline{) 75} \quad 3 \longleftarrow \text{least significant digit} \\
 \underline{8 } \\
 8 \quad 1 \\
 \underline{8 } \\
 8 \quad 1 \longleftarrow \text{most significant digit} \\
 \underline{8 } \\
 8 \quad 0
 \end{array}$$

Therefore, $75)_{10} = 113)_8$

This method follows from the fact that when an integer, N , is converted to the form $N = a_n 8^n + \dots + a_1 8^1 + a_0 8^0$, 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.

$$\begin{array}{r}
 1 \quad 5 \quad 5)_8 \\
 \frac{8}{8} \quad \text{multiply} \\
 \underline{5} \quad \text{add} \\
 13 \\
 \frac{8}{104} \quad \text{multiply} \\
 \underline{5} \quad \text{add} \\
 109)_{10} \quad \text{Answer}
 \end{array}$$

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 $\underbrace{101}_5 \quad \underbrace{011}_3 \quad \underbrace{010}_2$ or 532_8

and 0.110111_2

becomes $\underbrace{.110}_6 \quad \underbrace{111}_7$ 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 $\underbrace{6}_{110} \quad \underbrace{1}_{001} \quad \underbrace{2}_{010}$ or 110001010_2

6. Decimal Fractions to Octal or Binary

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

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

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 $D = a_{-1} 8^{-1} + a_{-2} 8^{-2} + a_{-3} 8^{-3} + \dots$, or to the binary form

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

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.

$$\begin{array}{r}
 .725 \\
 \times 2 \\
 \hline
 1.450 \\
 \times 2 \\
 \hline
 0.900 \\
 \times 2 \\
 \hline
 1.800 \\
 \times 2 \\
 \hline
 1.600
 \end{array}$$

a_{-1} term \longrightarrow

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

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.

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

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.

	<u>Decimal</u>	<u>Binary</u>	<i>2's Complemented</i>	<u>Binary</u>
a)	+12	+1100		0 1100
	<u>-04</u>	<u>-0100</u>		<u>1 1100</u>
	+08	-1000		0 1000
b)	+10	+1010	<i>1's Comp.</i> 0 1010	0 1010
	<u>-10</u>	<u>-1010</u>	<u>1 0101</u>	<u>1 0110</u>
	+00	+0000	1 1111	0 0000
c)	+12	+1100		0 1100
	<u>-14</u>	<u>-1110</u>		<u>1 0010</u>
	-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 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	.000350	.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	.000866
.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

APPENDIX C

OCTAL UTILITY PROGRAM

IDENTIFICATION: OCTAL UTILITY PACKAGE II

AUTHOR: A. W. England, PBCC

ACCEPTED: February 27, 1961

PURPOSE: To provide simplified control of the PB250 during program operation and checkout. The utility program operations are easily controlled by means of appropriate code letters, which allow the user to enter, inspect, and output information in a variety of formats.

RESTRICTIONS: Only those codes which are recognizable by the program should be entered; these include 0 to 9, +, -, semicolon (;), comma (,), period (.), \$, Tab, C/R, Delete, Space, B, C, D, F, G, I, S, T, W, Z. Entry may either be from paper tape or from the Flexowriter keyboard.

Of the remaining codes, any which have an octal configuration of 40 or greater will cause erratic and unpredictable operation. In this group are A, E, H, U, V, X, Y, apostrophe ('), U/C, L/C. Any codes which have an octal configuration less than 40 will be interpreted as octal digits, the value being determined by the least significant three bits of the code; included are J, K, L, M, N, O, P, Q, R, /, Stop Code, Tape Feed.

SPACE: The program uses all sectors of line 01 for instructions and storage, plus additional memory as follows: (1) When punching, sectors 376 and 377 of line 06; (2) When typing, sectors 376 and 377 of line 05.

TIMING: Since this is essentially an input-output program, its speed is determined by the speed of the Flexowriter. All operations will proceed at the maximum rate for the Flexowriter, which is about 10 characters/second for reading tape and typing, and 15 characters/second for punching.

USE:

1. Loading the Program

The utility program has a bootstrap and is self-loading. To load the program, insert the tape into the reader. The starting point is not critical, but it must be before the first set of holes. Raise the Fill switch on the console and clear any parity by depressing both the Enable and Breakpoint switches. Be sure to raise one of these after the tape starts moving.

The tape will stop when the bootstrap has been entered. To read the remainder of the utility tape, first lower the Fill switch, then clear parity and initialize by depressing both the Enable and Breakpoint switches and striking the I key while the Enable switch is down. When the Enable switch is raised, the tape will be read under program control (The Breakpoint switch may be up or down).

When the tape has been read in correctly (the check sum compares), the light on the Flexowriter will come on, indicating keyboard control. If the Flexowriter light does not come on, and 0037)₈ appears in the Operation and Operation lights of the console, the check sum did not compare and the program should be loaded again.

The functions of the utility package will normally be controlled from the Flexowriter keyboard. The keys for the input, output, control, and function operations are described in the following paragraphs.

2. Input Codes

a) \$ (Set Location)

Causes the specified location to be set for the input of information. The user first types five octal digits of the form SSSL, where SSS is the sector and LL the line number of the location to be set, and then \$. SSS may be any octal number from 000 to 377, and LL any octal number from 00 to 37. Regardless of the number of digits entered prior to the \$, only the last five will

USE (cont.):

be interpreted as an address. The sector number will be inserted into an indexed store command, and the line number will be stored in the Index register.

b) Carriage Return (Enter)

Used to enter a word of information into a location previously set with \$. After one word is entered, the location counter is advanced by one, with sector 000 following 377, so that the next C/R will enter a word into the next location. Each time the C/R is given, the contents of the program accumulator are entered into the location specified by the sector counter; the contents of the accumulated word are not affected. Regardless of the number of characters preceding the C/R, only the last 21 bits will be entered into the specified location.

c) F (Fill)

Causes the program to begin reading paper tape. This tape may be prepared ahead of time in the same format used when entering from the keyboard, in which case the control codes are interpreted as if they were typed. A location of the form SSSLL may be specified before the F code, and this will be set the same as with \$. However, any \$ on the tape will override the setting of the F.

The tape may also have been prepared by the utility program in a binary format in blocks of 256 words (one long line, see discussion of Output Codes) plus check sum. In this case, it is only necessary to set a line location either by typing LLF or by having placed LL\$ or LLF on the tape before the binary block was punched.

At the beginning of the binary block will be a G, placed there at the time of punching by the program, which marks the start of the block. After loading the line specified, the check sum on the tape is compared with the

USE (cont.):

sum computed during loading. If the check sum was correct, the program will continue to read in the normal F mode unless the Breakpoint switch is down, in which case control will return to the keyboard.

If the check sums do not compare, the program will halt with a line number of $37)_8$ appearing on the console. Control may be returned to the keyboard by depressing both the Enable and the Breakpoint switches together; when the Enable switch is raised, the Flexo-writer light will come on. The computed check sum will be stored in F17 of line 00. It may be typed out in octal by typing "01700D."

A "W" will also cause control to return to the keyboard, regardless of the position of the Breakpoint switch.

d) G (Guard)

This code guards the beginning of a binary block and is always punched by the program when preparing a binary tape. It should never be necessary for the user to depress this key.

3. Output Codes

a) B (Binary)

A line number ranging from 00 to $37)_8$, followed by a B, will cause the indicated line to be punched in a binary format starting from sector $177)_8$ and proceeding backward to sector $200)_8$. In this format, three frames on tape are required for each word in memory. The first has six bits of information, whereas the next two each contain eight bits. At the end of the tape, a check sum will be punched. This check sum is compared when the tape is reentered into the computer. Twice this check sum will be stored in line 00, channel F17.

A G code will be punched to mark the beginning of the tape.

USE (cont.):

b) C (Command Type)

To type out a word in command format, first type the location of the word (SSLL) followed by a C; the program will then type this word, carriage return, and if the Breakpoint switch is up, type the next word. Typing will continue until the Breakpoint switch is depressed. If the Breakpoint switch is down when C is depressed, only one word will be typed.

c) D (Data Type)

To type out a word in octal format, follow the same procedure as in C, except depress the D key instead of the C key.

d) Output Note

It is not possible to punch a listable tape directly, however, if the punch is turned on while the program is typing out in command or octal format, a tape will be punched which can be read into the computer.

4. Transfer-Control Codes

a) . (Period)

The period will cause control to be transferred to the location specified by the preceding five octal digits (SSLL). Control can be transferred to any sector of lines 00 thru 07.

b) W

When read from tape by the program, W will cause control to be returned to the keyboard. It is useful at the end of listable, octal format tapes, to return control to the keyboard. It is also useful at the end of binary tapes, if control is to be returned to the keyboard regardless of the position of the Breakpoint switch.

USE (cont.):

c) T

This code causes an unconditional transfer to sector 000 of line 02. The transfer command is located in sector 306 of line 01 and can be changed for use by a specific program. Any program which changes 306 01 should also make provision for restoring the original contents of this location upon completion of the program.

d) , (Comma)

Whenever a \$, F, B, C, D, or . code is read, the utility program stores the two octal digits preceding the code in the Index register. The comma (,) code makes use of this fact and transfers control to sector 000 of the line specified in the Index register; it can be used for a self-starting program tape that may go into any of several lines.

e) Enable - I

Control may always be returned to the keyboard from any place at any time by depressing the Enable switch and striking the I key. When the Enable switch is raised, the Flexowriter light will come on unless there is a parity which must also be cleared.

5. Function Codes

a) Z (Zero One Line)

This code will cause the contents of the indexed line to be set to zero. It is first necessary to set the desired line number into the Index register with an LL\$, or equivalent, and then to zero the accumulated word by typing +000000. Then, when the Z code is read, the desired line will be cleared and control will return to reading from whichever mode (tape or keyboard) the code was given. The accumulated word is not stored in the sectors of the line, but it must be cleared before the operation functions properly. The time required for this operation is less than 2 seconds.

USE (cont.):

6. Input-Output Formats

a) Command Format

A command format word has three octal digits for sector number, one bit for sequence tag, two octal digits for operation code, two octal digits for line number, and a one-bit index tag.

For example: In command 123S4507I, 123 is the sector number, S indicates that there is a sequence tag, 45 is the operation code, 07 the line number, and I indicates that there is an index tag. If there were no sequence tag, a space should be typed instead of the S; likewise, if there were no index tag, semicolon (;) should be typed instead of the I. Output will be in the same form as input.

The line number consists of six bits (two octal digits), but these are arranged with the most significant bit on the right of the six, next to the index tag. It is not necessary for the user to concern himself with this, however, as the program will automatically arrange this bit on input and rearrange it for output.

b) Data or Octal Format

An octal format word has a sign and seven octal digits. For example, +1234567 or -3214276. Negative numbers are not complemented either on input or output. The minus sign causes a one bit to be entered for the sign position; plus produces a zero in the sign position.

c) Tab and Code Delete

For convenience, the tab is ignored when entered from either tape or keyboard. The code delete is also ignored when read from tape.

METHOD:

1. When reading information in octal format from either tape or keyboard, the program inspects each character for the presence of a bit in the most significant position. If a one is present, it interprets this as a control code and jumps to the appropriate routine. If the high-order bit is a zero, the program assumes the character to be an octal digit and loads the three low-order bits of the character into the low-order positions of an accumulating word which is shifted left to allow insertion of the digit.

2. The four one-bit characters, S, Space, I, semicolon (;), cause insertion of only one bit into the accumulator. In addition to inserting one bit, I and ;, also cause the preceding six bits to be rearranged by moving the most significant bit of the six to the least significant position.

3. The control characters which require an address assume that this address is the last thing entered into the program accumulator. The Index register is then set with the line number of the address, and the sector number is placed into an appropriate load or store command. None of these control characters rearrange the line number, therefore, it is impossible to set a line number greater than 37)₈.

APPENDIX A

OPERATIONS SUMMARY

OCTAL UTILITY PACKAGE II

Set location counter	SSLL\$
Enter accumulated word and advance location counter	C/R
Set location counter and fill from tape	SSLLF
Punch line in binary format	LLB
Type word in command format	SSLLC
Type word in data (octal) format	SSLLD
Jump to specified location	SSLL.
Return control from tape to keyboard	W
Jump to sector 000 of line 02	T
Jump to sector 000 of last line indexed	,
Clear indexed line to +0000000	LL\$+0000000 Z
Sequence tag: one (1)	S
Sequence tag: zero (0)	Space
Index tag: one (1)	I
Index tag: zero (0)	;
Sign plus: + (0)	+
Sign minus, - (1)	-
Ignored codes:	{ Tab Code delete

pb Packard Bell Computer**PB 250 PROGRAM LISTING**

Catalog Number 0001A

PROBLEM OCTAL UTILITY PACKAGEPAGE 1 OF 9PROGRAMMER A. W. ENGLANDDATE 2-28-61

LOCATION	INSTRUCTION	SYMBOLIC OF CODE	REMARKS
00001\$	263S0701;	LDP	LOAD RTK'S
001	377 0000;	CONST	-7740000 [SECTOR DECREMENT]
002	013S2100;	LSD	8
003	027S5501;	LAI	[BINARY SWITCH]
004	017 3601;	TBN	WHEN WORD COMPLETE
005	001S4001;	EBP	TO FILL SIGN OF A
006	017 1100;	STA	IN CK. SUM
007	377 0401;	LDC	LINE COUNT
010	011S0701;	LDP	TO LOAD MARKER INTO B
011	000 00161	CONST	+0000071 [I CODE]
012	002S5100;	RTK	CHANGEABLE READ COMMANDS
013	014 5100;	RTK	
014	013 7736;	TES	TO REJECT OLD CHARACTER
015	012 7736;	TES	TO SENSE NEW CHARACTER
016	014S5700;	CIB	BACK TO TES
017	301 3401;	TCN	LINE COMPLETE
020	010 26411	TEMP	ALSO [STORE (2)]
021	017 1400;	ADD STA	CK. SUM CK. SUM
022	017 1100;	LDA	STORE (2)
023	020 0501;	LDA	STORE (2)
024	001 1401;	ADD	SECTOR DECREMENT
025	020 1101;	STA	STORE (2)
026	010S3701;	TRU	TO LOAD MARKER
027	000 01771	CONST	+0000377
030	033S4001;	EBP	TO TEST FOR CONTROL CHARACTER
031	033 4001;	EBP	
032	141S4400;	CLC	
033	377S7720;	CONST	-7777700
034	271 3501;	TAN	TO CONTROL SECTOR
035	315 5601;	CAM	WITH SPACE CODE

pb Packard Bell Computer**PB 250 PROGRAM LISTING**

Catalog Number 0001A

PROBLEM OCTAL UTILITY PACKAGEPAGE 2 OF 9PROGRAMMER A.W. ENGLANDDATE 2-28-61

LOCATION	INSTRUCTION	SYMBOLIC OP CODE	REMARKS
03601	344 7501;	TOF	TO SPACE ROUTINE
037	000 0200;	IBC	
040	044 2210;	RSO	3
041	000 0100;	IAC	
042	066 2210;	RSO	19
043	01234500;	CLA	BACK TO READ
044	226 0501;	LDA	KEY TEMP.
045	061 5601;	CAM	0
046	136 7501;	TOF	
047	07652200;	RSI	22
050	057 2100;	LSD	6
051	020 0601;	LDB	TEMP W
052	054 3300;	SBR	1
053	05752100;	LSD	3
054	055 7101;	TEMP	FOR TYPE OUT FLAG
055	00053701;	TEMP	FOR TYPE OUT KEY
056	06350200;	IBC	
057	11453701;	TRU	
060	000 3501;	TAN	TO START
061	000 0000;	CONST	+0000000 [D FLAG]
062	17757720;	CONST	+7777700 [D KEY WORD]
063	05431301;	STD	IN TYPE OUT TEMPS
064	066 2100;	LSD	1
065	066 1237;	STB	IN INDEX REGISTER
066	076 2100;	LSD	7
067	073 0401;	LDC	LOAD (1)
070	07154601;	AOC	TO SET UP SECTOR
071	00057777	CONST	+0037777
072	073 1201;	STB	LOAD (1)
073	253 05001	LDA	[LOAD (1)]

pb Packard Bell Computer**PB 250 PROGRAM LISTING**

Catalog Number 0001A

PROBLEM OCTAL UTILITY PACKAGEPAGE 3 OF 9PROGRAMMER A.W. ENGLANDDATE 2-28-61

LOCATION	INSTRUCTION	SYMBOLIC OF CODE	REMARKS
07401	020 1101;	STA	TEMP W
075	055 0601;	LDB	KEY
076	000 4500;	CLA	
077	102 2100;	LSD	2
100	226 1201;	STB	KEY TEMP
101	020 0601;	LDB	TEMP W
102	000 0100;	IAC	
103	105 2100;	LSD	1
104	106 3300;	SBR	1
105	050 3401;	TCN	KEY DIGIT 0
106	110 2100;	LSD	1
107	153 3401;	TCN	KEY DIGIT 1
110	112 2100;	LSD	1
111	114 3401;	TCN	KEY DIGIT 2
112	114S2100;	LSD	1 FOR KEY DIGIT 3
113	217 3501;	TAN	TO WORD OUT
114	020 1201;	STB	TEMP W
115	143S4300;	CLB	
116	117 0030;	MAC	COPY A TO C
117	374S4100;	GTB	TO CHECK PARITY
120	061 5601;	CAM	0
121	124 3401;	TCN	PARITY CORRECT
122	315 1401;	ADD	PARITY BIT
123	122 7501;	TOF	TO ADD PARITY BUT AGAIN FOR 0 CODE
124	127 2100;	LSD	2
125	127 1601;	DPA	RETURN AND WOC 0
126	143S0300;	ROT	
127	044S3701;	TRU	[RETURN]
130	000 6000;	WOC	[WOC 0]
131	146S3701;	TRU	[DUMMY CHAR. RETURN]

LOCATION	INSTRUCTION	SYMBOLIC OF CODE	REMARKS
13201	376 1305;	STD	IN TYPE OUT SECTORS
133	311 0401;	LDC	DELAY NO.
134	134 7737;	TES	TYPEWRITER BUSY
135	376S3705;	TRU	TO TYPE OUT
136	137S0701;	LDP	TYPE C/R AND RETURN
137	000 6116;	WOC	C/R
140	307S3701;	TRU	C/R RETURN
141	132S3701;	TRU	TO STORE AND LOAD DELAY
142	143 0030;	MAC	
143	362S0200;	IBC	
144	115S4400;	CLC	
145	131S0200;	IAC	
146	013 7735;	TES	B. P. FOR END OF TYPING
147	073 0501;	LDA	LOAD (1)
150	001 1501;	SUB	SECTOR DECREMENT
151	073 1101;	STA	LOAD (1)
152	073S3701;	TRU	LOAD (1)
153	020 1201;	STB	TEMP W
154	156 3300;	SBR	1 AND CLA
155	054 1501;	SUB	FLAG
156	061 5601;	CAM	0
157	177 7501;	TOF	TO SELECT + OR -
160	054 1101;	STA	FLAG
161	171 3501;	TAN	SELECT S OR SPACE
162	020 0601;	LDB	TEMP W [SELECT I OR ;]
163	167 3601;	TBN	LOAD I CODE
164	165S0501;	LDA	; CODE
165	000 0014;	CONST	+0000060 [; CODE]
166	123S4300;	CLB	TO ADD WOC 0
167	011 0501;	LDA	I CODE

PROBLEM OCTAL UTILITY PACKAGEPAGE 5 OF 9PROGRAMMER A. W. ENGLANDDATE 2-28-61

LOCATION	INSTRUCTION	SYMBOLIC OF CODE	REMARKS
17001	123S4300;	CLB	TO ADD WOC 0
171	174 3601;	TBN	TO LOAD S CODE [SELECT S OR SPACE]
172	315 0501;	LDA	SPACE DOCODE
173	123S4300;	CLB	TO ADD WOC 0
174	175S0501;	LDA	S CODE
175	000 0054;	CONST	+0000062 [S CODE]
176	123S4300;	CLB	TO ADD WOC 0
177	201 2100;	LSD	1 [SELECT + OR -]
200	235 1401;	ADD	+ CODE
201	123S4300;	CLB	TO ADD WOC
202	000 7735;	TES	B. P. FOR RETURN TO KEYBOARD
203	265S4300;	CLB	TO RESET BIN. SW. ROUTINE
204	226 1101;	STA	LOAD (2)
205	113 0501;	LDA	WITH TAN TO WORD OUT
206	245 1101;	STA	OUT SW.
207	000 4500;	CLA	
210	017 1100;	STA	CK. SUM
211	212S0701;	LDP	WOC G. AND RETURN
212	000 6107;	WOC	G
213	223S3701;	TRU	G RETURN
214	376 1306;	STB	IN PUNCH OUT SECTORS
215	311 0401;	LDC	DELAY NO.
216	376S3706;	TRU	PUNCH OUT
217	226 0501;	LDA	LOAD (2)
220	001 1401;	ADD	SECTOR DECREMENT
221	226 1101;	STA	LOAD (2)
222	252 7501;	TOF	LINE END
223	224S0501;	LDA	TRU BACK FROM 06
224	245S3701;	TRU	BACK FROM 06
225	377 1106;	STA	RETURN SECTOR

pb Packard Bell Computer

PB 250 PROGRAM LISTING

Catalog Number 0001A

PROBLEM OCTAL UTILITY PACKAGE

PAGE 6 OF 9

PROGRAMMER A. W. ENGLAND

DATE 2-28-61

LOCATION	INSTRUCTION	SYMBOLIC OP CODE	REMARKS
22601	177 05001	TEMP	[LOAD (2)]
227	020 1101;	STA	TEMP B
230	017 1400;	ADD	CK. SUM
231	017 1100;	STA	CK. SUM
232	020 0601;	LDB	TEMP B
233	315 0401;	LDC	COUNTER
234	235S4500;	CLA	
235	000 0047;	CONST	+0000036 [+ CODE]
236	245 2100;	LSD	6
237	020 1201;	STB	TEMP B
240	000 4300;	CLB	
241	244 2110;	LSO	2
242	130 1401;	ADD	WOC 0
243	376 1106;	STA	PUNCH OUT SECTOR
244	214S0100;	IAC	TO LOAD DELAY NO.
245	000 3501;	TAN	[OUT SW.]
246	020 0401;	LDC	TEMP B
247	000 0300;	ROT	
250	261 2100;	LSD	8
251	237S7501;	TOF	TO STORE IN TEMP
252	060 0501;	LDA	WITH TAN START [LINE END]
253	245 1101;	STA	OUT SW.
254	017 0600;	LDB	CK. SUM
255	232S4500;	CLA	TO PUNCH OUT WORD ROUTINE
256	027 5501;	LAI	FOR BINARY SW. ON
257	177 11001	STA	FOR STORE (2)
260	020 1101;	STA	IN STORE (2)
261	003 1201;	STB	IN BINARY SW.
262	005S4500;	CLA	TO BINARY START
263	002S5100;	RTK	FOR READ SEQUENCE

pb Packard Bell Computer

PB 250 PROGRAM LISTING

Catalog Number 0001 A

PROBLEM OCTAL UTILITY PACKAGE

PAGE 7 OF 9

PROGRAMMER A. W. ENGLAND

DATE 2-28-61

LOCATION	INSTRUCTION	SYMBOLIC OF CODE	REMARKS
26401	014 5100;	RTK	FOR READ SEQUENCE
265	012 1301;	STD	IN READ SEQUENCE
266	267S0401;	LDC	WITH BIN. SW. OFF
267	027S5501;	LAI	FOR BIN. SW. OFF
270	003S1001;	STC	BIN. SW.
271	020 1201;	STB	IN ACCUMULATED WORD [CONTROL SELECTOR]
272	000 4300;	CLB	
273	303 2200;	RSI	7
274	060 1601;	DPA	TAN 0
275	277 1201;	STB	JUMP
276	020 0601;	LDB	ACCUMULATED WORD
277	326 3501;	TAN	[JUMP] (A IS NOW ALWAYS NEG.)
300	036S4500;	CLA	[O ROUTINE]
301	017 5600;	CAM	CK. SUM [LINE END FOR BINARY INPUT]
302	202 7501;	TOF	IF CK. SUM COMPARES
303	000S0037;	HLT	37 [BAD CK. SUM ERROR HALT]
304	306 2100;	LSD	1 [B ROUTINE]
305	317S0501;	LDA	WITH INITIALIZED LOAD (2)
306	000S3702;	TRU	SECTOR 000 LINE 02 [T ROUTINE]
307	130S0701;	LDP	DUMMY CHAR. AND RETURN
310	344S0200;	IBC	[D ROUTINE]
311	000 1205;	CONST	+0002424 [C FLAG] [DELAY NO.]
312	275S7124;	CONST	-3676320 [C KEYWORD]
313	054S1301;	STD	IN TYPE OUT TEMP'S.
314	000S3701;	TRU	START [W ROUTINE]
315	000 0004;	CONST	+0000020 [SPACE CODE]
316	256S0701;	LDP	BIN. W. ON + INITIAL STORE (2)
317	177 05001	LDA	[INITIAL STORE (2)]
320	203S1237;	STB	INDEX REGISTER
321	323S2100;	LSD	1

PROBLEM OCTAL UTILITY PACKAGE

PAGE 8 OF 9

PROGRAMMER A. W. ENGLAND

DATE 2-28-61

LOCATION	INSTRUCTION	SYMBOLIC OF CODE	REMARKS
32201	037 0401;	LDC	COUNT FOR ZERO LINE [Z ROUTINE]
323	013 3401;	TCN	BACK TO READ
324	325S2500I	IAM	256
325	320S4500;	CLA	TO LSD 1 (317)
326	330 2100;	LSD	1 [. ROUTINE]
327	336S1237;	STB	INDEX REGISTER
330	340 2100;	LSD	7
331	334 0401;	LDC	STORE (1)
332	071 4601;	AOC	TO SET SECTOR
333	371S0300;	ROT	
334	055 1200I	STB	[STORE (1)] [C/R ROUTINE]
335	334 0501;	LDA	STORE (1)
336	371S4400;	CLC	
337	347S2100;	LSD	7
340	027S0300;	ROT	[; ROUTINE]
341	363S0300;	ROT	
342	344 2100;	LSD	1 [\$ ROUTINE]
343	327S1237;	STB	INDEX REGISTER
344	340S0100;	IAC	[S AND SPACE ROUTINE]
345	061S0701;	LDP	D FLAG AND KEYWORD
346	352S0200;	IBC	[C ROUTINE]
347	352 0401;	LDC	TRANSFER
350	071 4601;	AOC	TO SET SECTOR
351	352 1201;	STB	TRANSFER
352	054S3700I	TRU	[TRANSFER]
353	311S0701;	LDP	C FLAG AND KEYWORD
354	000 0200;	IBC	[F ROUTINE]
355	356S0701;	LDP	RPT'S FOR READ SEQUENCE
356	002S5200;	RPT	FOR READ SEQUENCE
357	014 5200;	RPT	

PB 250 PROGRAM LISTING

Catalog Number 0001A

PROBLEM OCTAL UTILITY PACKAGE

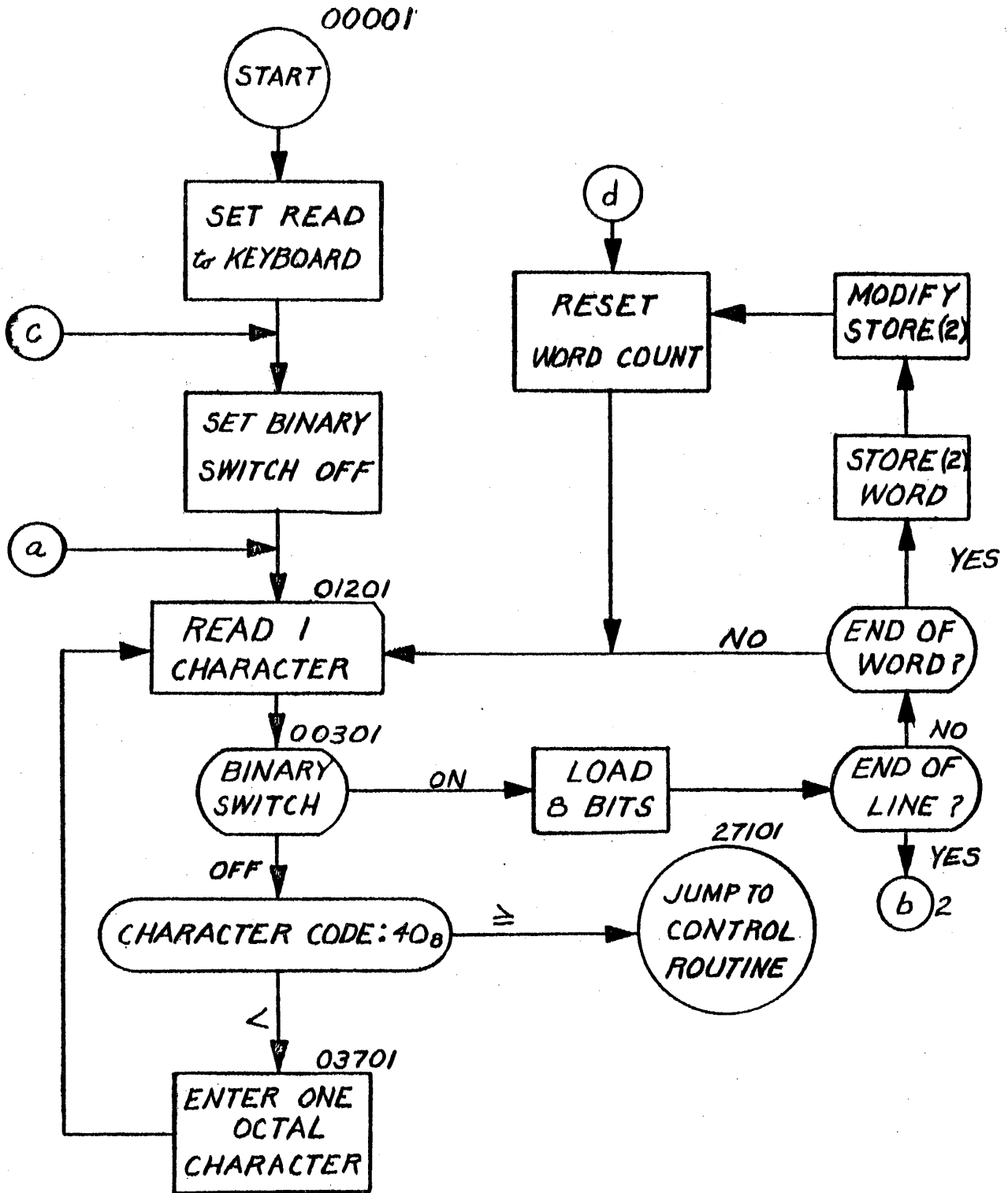
PAGE 9 OF 9

PROGRAMMER A. W. ENGLAND

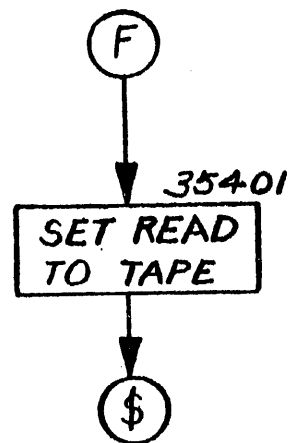
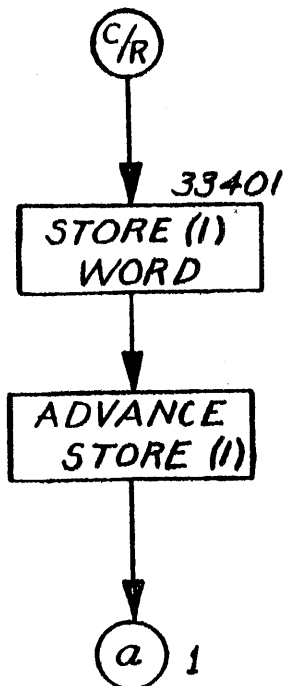
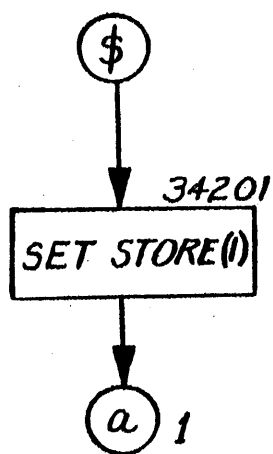
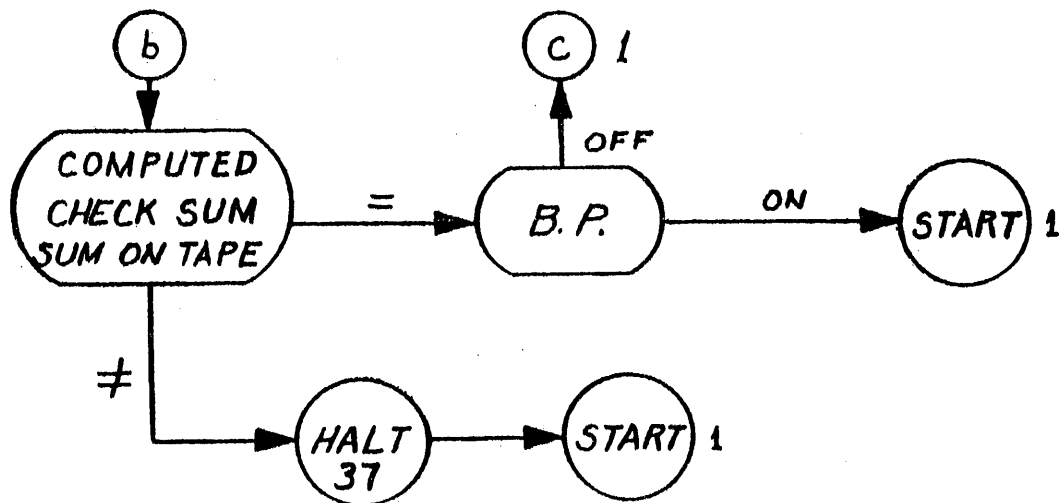
DATE 2-28-61

LOCATION	INSTRUCTION	SYMBOLIC OF CODE	REMARKS
36001	012 1301;	STD	IN READ SEQUENCE
361	341S0200;	IBC	TO \$ ROUTINE
362	337S0401;	LDC	[I ROUTINE]
363	011 2210;	RSO	21
364	366S0100;	IAC	
365	013S2200;	RSI	21
366	000S3700I	TRU	SECTOR 000 INDEXED LINE [, ROUTINE]
367	020 0401;	LDC	ACCUMULATED WORD
370	033 4601;	AOC	TO SET LINE NO.
371	340S0100;	IAC	
372	001 1501;	SUB	SECTOR DECREMENT
373	334 1101;	STA	STORE (1)
374	012S4500;	CLA	TO READ [TAB ROUTINE]
375	117S0100;	IAC	TO RECALL ORIGINAL OCTAL DIGIT TO A
376	012S4500;	CLA	TO READ [CODE DELETE ROUTINE]
377	000 4002;	CONST	+0010010 [LINE COUNT]

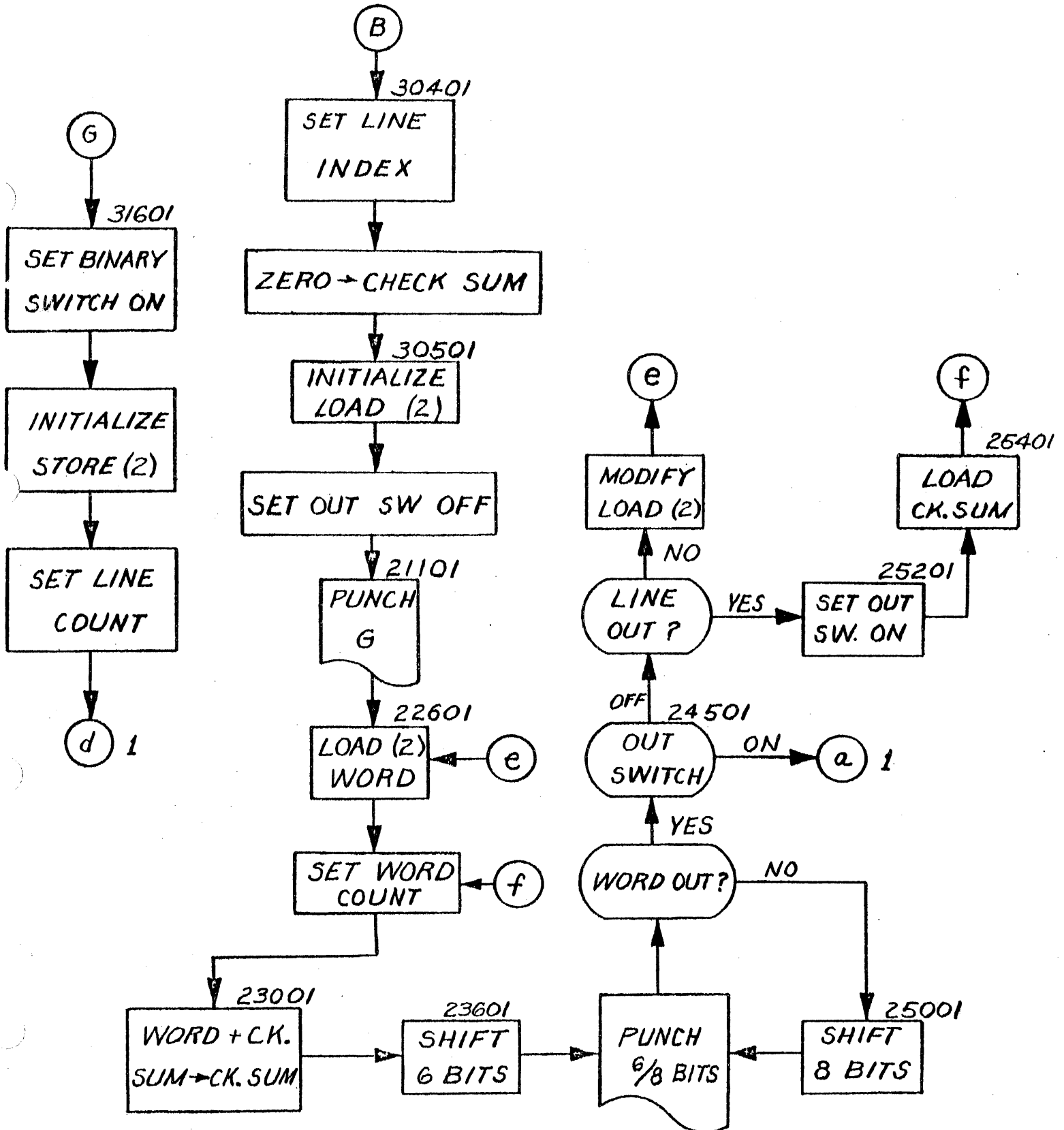
APPENDIX C
 OCTAL UTILITY PACKAGE II
 FLOW DIAGRAM

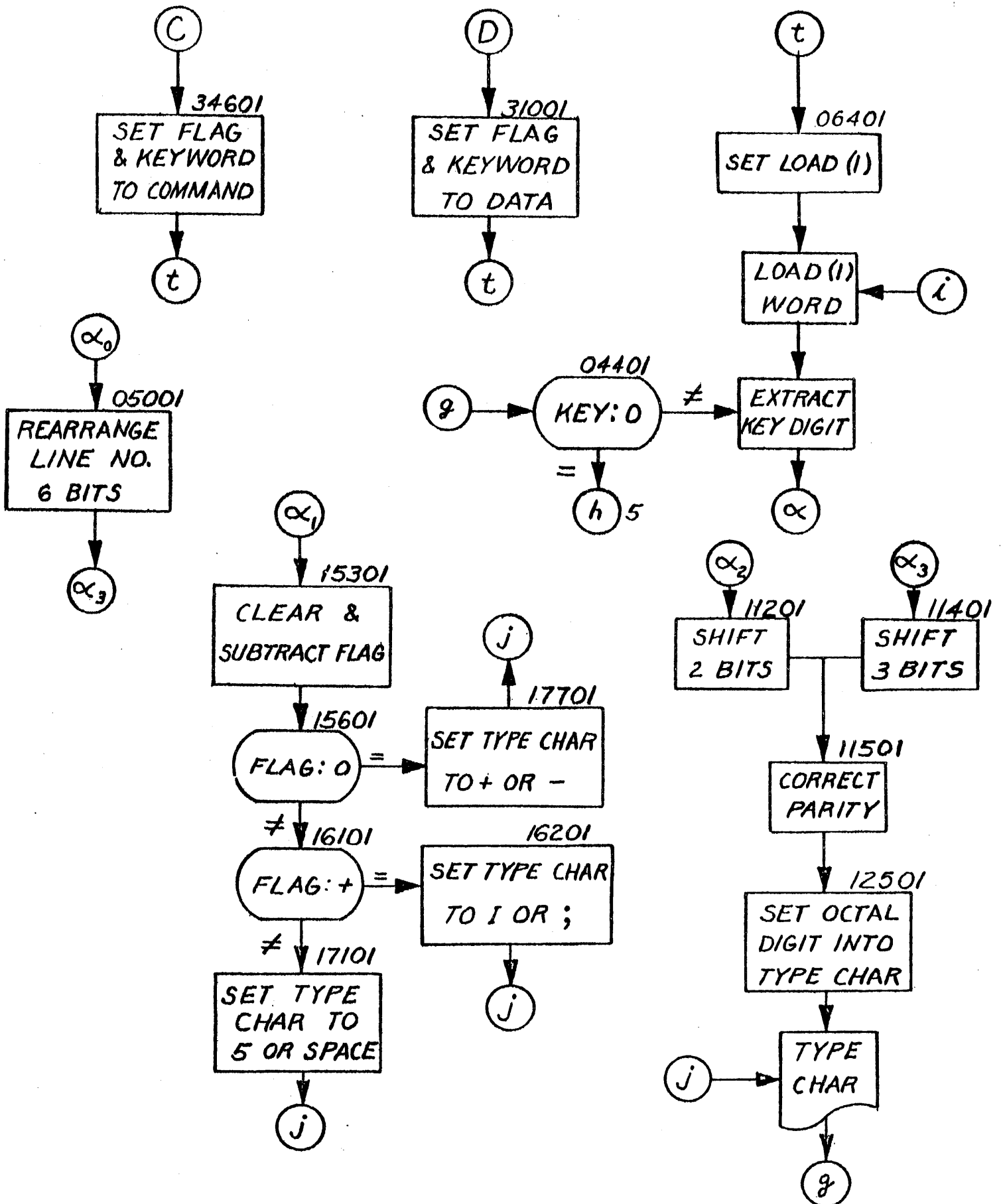


OCTAL UTILITY PACKAGE II
 FLOW DIAGRAM

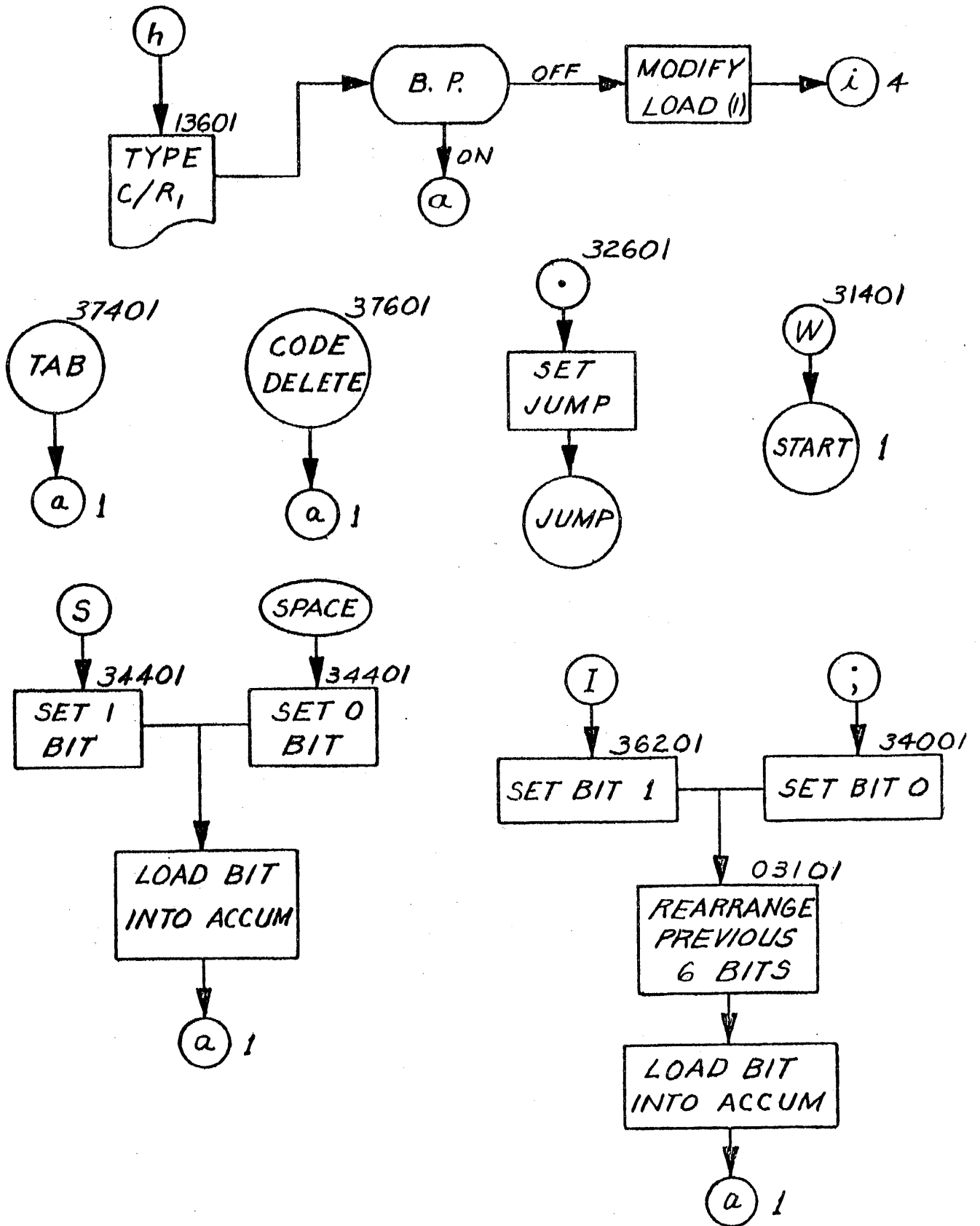


OCTAL UTILITY PACKAGE II
 FLOW DIAGRAM

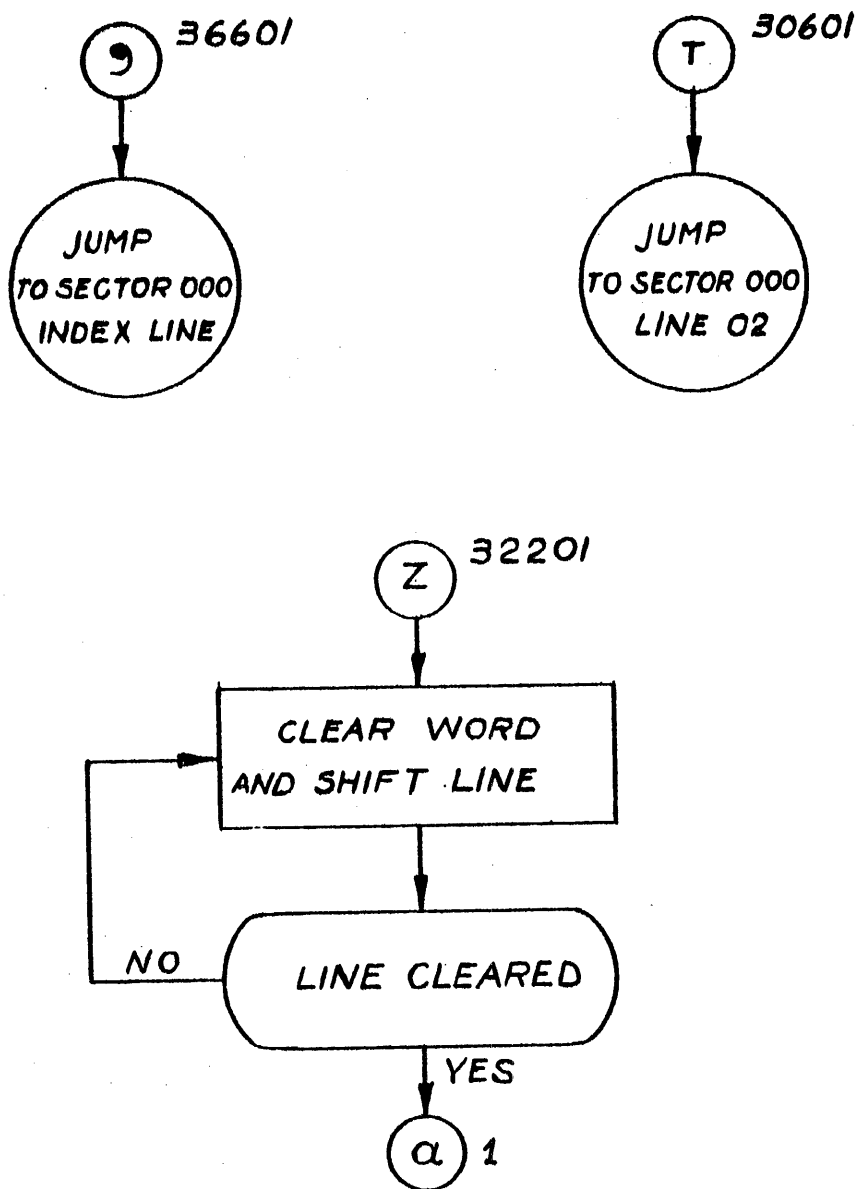




OCTAL UTILITY PACKAGE II
 FLOW DIAGRAM



FLOW DIAGRAM



APPENDIX D

OCTAL UTILITY PACKAGE II
BOOTSTRAP

The bootstrap section of this program is actually the binary loading portion of the complete program. After the bootstrap is loaded by means of the binary fill mode (controlled by the Fill switch on the console), the bootstrap pulls in the rest of the program around it, without using more than one additional sector in another line. The bootstrap routine itself occupies 25 sectors from $377)_8 - 027)_8$; of this, sectors $001 - 027)_8$ are actually part of the completely loaded program.

In the bootstrap only, the binary loading routine starts loading into sector 000 and loads the main program backwards through sector $030)_8$. This method produces the shortest possible tape and the fastest loading time. Ordinarily, loading starts with sector 177.

pb Packard Bell Computer**PB 250 PROGRAM LISTING**

Catalog Number 0001A

PROBLEM OCTAL UTILITY PACKAGE BOOTSTRAPPAGE 1 OF 1PROGRAMMER A.W. ENGLANDDATE 2-28-61

LOCATION	INSTRUCTION	SYMBOLIC OF CODE	REMARKS
37701\$	+0007230	CONST	[LINE COUNT]
000	005S4500;	CLA	FOR CK. SUM
001	-7740000	CONST	[EDP MASK] [SECTOR DECREMENT]
002	013S2100;	LSD	8
003	027 5501;	LAI	LAI MASK
004	017 3601;	TBN	TO WORD END
005	001S4001;	FBP	TO FILL SIGN OF A
006	017 1100;	STA	CK. SUM
007	377 0401;	LDC	LINE COUNT
010	011S0701;	LDP	TO LOAD MARKER INTO B
011	+0000071	CONST	[MARKER]
012	002S5200;	RPT	TO LOAD BUFFER
013	014 5200;	RPT	TO REJECT OLD CHAR.
014	013 7736;	TES	
015	012 7736;	TES	TO WAIT FOR NEW CHAR.
016	014S5700;	CIB	
017	301 3401;	TCN	LINE END
020	000 1101;	STA	[STORE (2)] FOR LINE 01
021	017 1400;	ADD	CK. SUM
022	017 1100;	STA	CK. SUM
023	020 0501;	LDA	STORE (2)
024	001 1401;	ADD	SECTOR DECREMENT
025	020 1101;	STA	STORE (2)
026	010S3701;	TRU	TO LOAD MARKER
027	+0000377	CONST	[LAI MASK]

APPENDIX D

RECIRCULATION CHART

Recirculation Chart For a 16-Word Line

F00	F01	F02	F03	F04	F05	F06	F07	F10	F11	F12	F13	F14	F15	F16	F17
0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27	30	31	32	33	34	35	36	37
40	41	42	43	44	45	46	47	50	51	52	53	54	55	56	57
60	61	62	63	64	65	66	67	70	71	72	73	74	75	76	77
100	101	102	103	104	105	106	107	110	111	112	113	114	115	116	117
120	121	122	123	124	125	126	127	130	131	132	133	134	135	136	137
140	141	142	143	144	145	146	147	150	151	152	153	154	155	156	157
160	161	162	163	164	165	166	167	170	171	172	173	174	175	176	177
200	201	202	203	204	205	206	207	210	211	212	213	214	215	216	217
220	221	222	223	224	225	226	227	230	231	232	233	234	235	236	237
240	241	242	243	244	245	246	247	250	251	252	253	254	255	256	257
260	261	262	263	264	265	266	267	270	271	272	273	274	275	276	277
300	301	302	303	304	305	306	307	310	311	312	313	314	315	316	317
320	321	322	323	324	325	326	327	330	331	332	333	334	335	336	337
340	341	342	343	344	345	346	347	350	351	352	353	354	355	356	357
360	361	362	363	364	365	366	367	370	371	372	373	374	375	376	377