Burroughs TC 500

OPERATION AND PROGRAMING MANUAL

Part II

GENERAL PURPOSE LANGUAGE 300

Printed in U. S. America   February, 1969   For 1034006

TABLE OF CONTENTS

PART II GENERAL PURPOSE LANGUAGE 300

TABLE OF CONTENTS – PART II (continued)

TABLE OF CONTENTS – PART II (continued)

TABLE OF CONTENTS – PART II (continued)

TABLE OF CONTENTS – PART II (continued)

PART II - GENERAL PURPOSE LANGUAGE 300

1. INTRODUCTION

General Purpose Language 300 (G.P. 300) is a programing language, consisting of machine instructions to control system operation, and is used for writing applicational programs for Series 500 Terminal computers.

The G.P. 300 instruction list is implemented in the system by various "Firmware Sets"; the number of different instructions implemented is dependent on the particular Firmware Set used in the system. Firmware is defined as a control program, and is stored in a designated area of the system memory, not accessable to the programmer and/or user. It performs a great deal of the logic and control functions, programmatically, that are usually performed by hardware electronic circuits in larger computer systems. Thus, with much less hardware it provides much more sophisticated capabilities than would ordinarily be possible from the electronic circuitry alone in this size of system.

Firmware consists of "micro-programs" which implement each instruction of G.P. 300. A micro-program consists of a series or "string" of MICRO instructions, each of which performs a "small" step or function, to accomplish the function of the G.P. 300 instruction (sometimes referred to as a MACRO instruction since it is a "large" composite of a series of tiny steps - micro steps). Thus, in the execution of an application program, the firmware identifies each "macro instruction" used by the programmer, and selects the proper microprogram to perform the functions of the instruction.

This manual provides a detailed description of the functional results of each instruction in the G.P. 300 language. A series of "Firmware Sets" are available to implement the G.P. 300 language; each of which is micro programed to implement various instructions of the total G.P. 300 instruction list, and thus are referred to as firmware "subsets" of G.P. 300. Since the number of instructions varies with each subset, so does the amount of memory required by that firmware subset. Each subset is described in the appendix, and the amount of memory available to the programmer is indicated. Each firmware subset of G.P. 300 is independent of the others, enabling the system to function within the specified capabilities; that is, rather than selecting two or more subsets to be used in the system concurrently, a subset is selected that provides the degree of capabilities desired.

A system may operate with any of the G.P. 300 firmware subsets and at different times, provided that the system contains the hardware features necessary to permit the firmware to function. For example, a subset that implements the use of a paper tape reader requires a system that has hardware capabilities for Input/Output.

Changing from one firmware subset to another is accomplished by loading the new firmware program into the system memory. This requires a field engineer.

The following G.P. 300 firmware subsets are representative:

a. Basic Set: Provides for keyboard data entry, printing, 4 function arithmetic, logical comparisons, indexing and forms handling.

b. Basic Set plus Punched Paper Tape Input/Output: Provides for reading and/or punching paper tape, as well as basic set functions.

c. Basic Set plus Punched Card Input/Output: Provides for reading and/or punching tab cards, as well as basic set functions.

d. Basic Set plus Data Communication: Provides for transmitting and receiving data in a telecommunication network, as well as basic set functions.

e. Basic Set plus Data Communication and Punched Paper Tape Input/Output: See above.

f. Basic Set plus Data Communication and Punched Card Input/Output: See above.

1.1 MEMORY ORGANIZATION

Of the 1280 words of memory, 256 words (8 tracks) are set aside for the data communication processor (refer to section 11); the remaining 1024 words are considered MAIN MEMORY. The system MAIN MEMORY is considered in two sections: the CONTROL area and the NORMAL area. The CONTROL area is reserved to contain the Firmware subset and is not accessable to the programmer. The NORMAL area is used by the programmer to contain an application program, and to provide working memory for the accumulation of data. Refer to appendix A for the memory requirements of each firmware subset, and the remaining NORMAL memory. Illustrations and discussions following use "N" to represent the high order word available in Normal memory.

<u>NUMERIC WORD</u> - (15 Digits)

	MSD														LSD
Flags	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

4 bits each digit

<u>PRINT FORMAT WORD (MASK)</u> - (15 Digits)

	MSD														LSD
Flags	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

4 bits each digit

<u>ALPHA WORD</u> - (8 Characters: Space and "end alpha" code are considered as characters).

MSD							LSD
0	1	2	3	4	5	6	7

8 bits

<u>PROGRAM WORD</u> - (4 Instructions) or syllables

MSD			LSD
3	2	1	0

16 bits

Fig. 1 - 1 Word Organization As It Assumes Various Functions

1.2 MEMORY WORD

Each word of memory contains 16 digits (or 64 bits) and can be used in several ways: When storing numeric data, it contains 15 digits and sign. It may store 8 characters of alphanumeric information, or a print mask of 15 control codes plus 3 flags for controlling the printing of numeric words. These words are addressed by a word number (0 to N) which refers to their location in memory. "N" is defined as the high order word available to the user in Normal Memory. The word number is sometimes referred to as "Memory Address" (MA), or "memory location".

A word of memory may also contain program instructions. Four instructions may be stored in each word. Since instructions are provided to "Branch" (jump) from one area of a program to a specific instruction in another part of the program, each instruction is identified as a syllable (0, 1, 2, or 3) within a word. It is addressed as word (0 to N) syllable (0 to 3).

1.3 PROGRAM EXECUTION

Program instructions are executed in sequential order beginning with word 0, syllable 0 and progressing to syllable 1, 2, 3; word 1 syllable 0, etc. After power is turned on to the system and it is in the Ready mode, the Program mode is entered by depressing the START key. At this point, a program counter is loaded with the initial setting of word 0, syllable 0, and the program proceeds to execute the instructions in sequence, incrementing the program counter after each instruction. The sequence of program execution can be altered by using "branching" instructions, which permit going from any syllable of any word to any syllable of any other word. Instructions which cause "branching" do so by changing the contents of the program counter to the "jump to" address; from the new address, the sequential progression of program execution resumes.

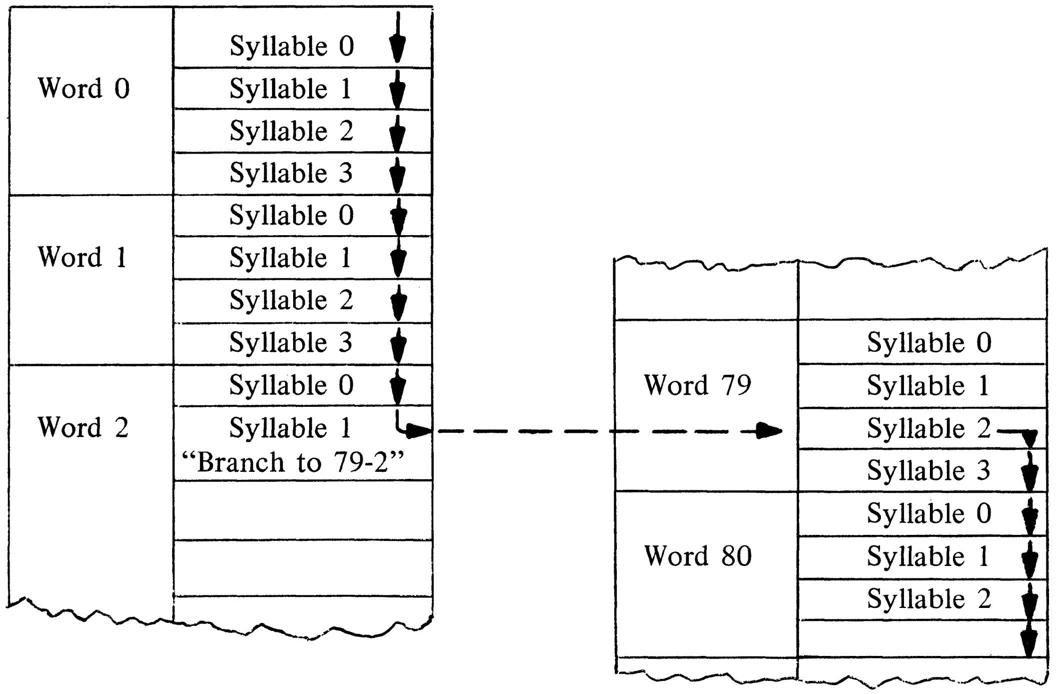
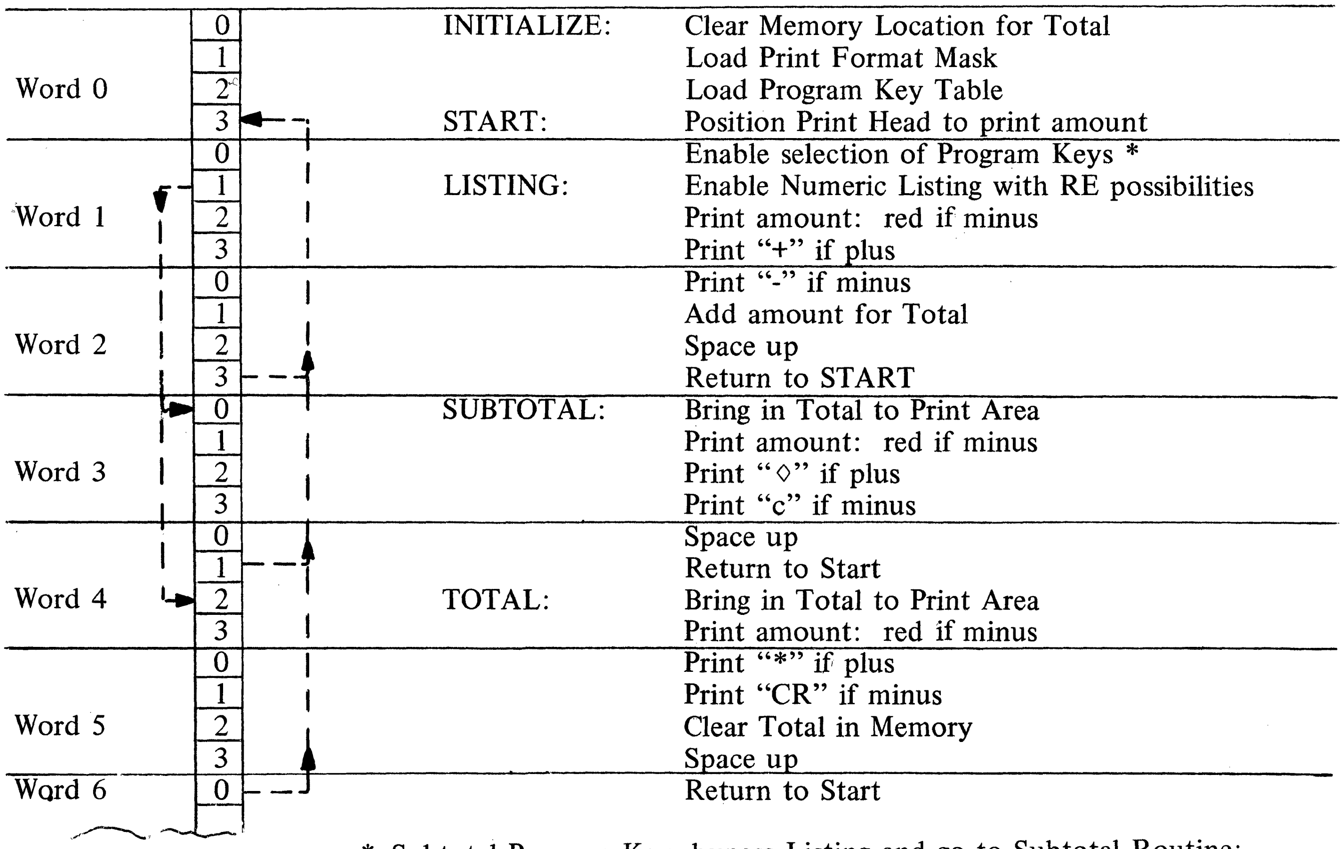


Fig. 1-2 Showing Sequential Program Execution and the effect of using the "Branch to 79-2" instruction.

To illustrate how the Computer functions in executing a program, the following example shows the individual program steps necessary to make the Computer perform as an adding machine. Although actual instructions are not used, each line of the "program" below represents a function that can be performed by one instruction and is related to one syllable of a memory word:

PROGRAMING AN ADDING MACHINE ROUTINE



Word 0	0	INITIALIZE:	Clear Memory Location for Total
	1		Load Print Format Mask
	2		Load Program Key Table
	3	START:	Position Print Head to print amount
Word 1	0		Enable selection of Program Keys *
	1	LISTING:	Enable Numeric Listing with RE possibilities
	2		Print amount: red if minus
	3		Print "+" if plus
Word 2	0		Print "-" if minus
	1		Add amount for Total
	2		Space up
	3		Return to START
Word 3	0	SUBTOTAL:	Bring in Total to Print Area
	1		Print amount: red if minus
	2		Print "◇" if plus
	3		Print "c" if minus
Word 4	0		Space up
	1		Return to Start
	2	TOTAL:	Bring in Total to Print Area
	3		Print amount: red if minus
Word 5	0		Print "*" if plus
	1		Print "CR" if minus
	2		Clear Total in Memory
	3		Space up
Word 6	0		Return to Start

* Subtotal Program Key, bypass Listing and go to Subtotal Routine; Total Program Key, bypass Listing and go to Total Routine; otherwise, go to Listing Routine

Fig. 1-3 Showing Programing Steps Necessary to Make the Computer Act as an Adding Machine.

1.4 INSTRUCTION FORMAT

Each instruction occupies four digits of a word, and is represented in memory by data consisting of Hexadecimal Codes (Hexa=6, Decimal=10.) Since each digit contains 4 bits, there are 16 possible combinations or codes in one Hexadecimal digit or 65,536 possible combinations in 4 digits 16^4.

	8	o	o	o	o	o	o	o	o	●	●
	4	o	o	o	o	●	●	●	●	o	o
Bits	2	o	o	●	●	o	o	●	●	o	o
	1	o	●	o	●	o	●	o	●	o	●
Decimal Number		"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"

Binary Coded Decimals
(Note: Anything beyond a "9" is an error.)

	8	o	o	o	o	o	o	o	o	•	•	•	•	•	•	•	•
	4	o	o	o	o	•	•	•	•	o	o	o	o	•	•	•	•
Bits	2	o	o	•	•	o	o	•	•	o	o	•	•	o	o	•	•
	1	o	•	o	•	o	•	o	•	o	•	o	•	o	•	o	•
Decimal Number		"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"	"10"	"11"	"12"	"13"	"14"	"15"
Hexadecimal Code		"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"	"A"	"B"	"C"	"D"	"E"	"F"

Binary Coded Hexadecimal
(All possible Bit Configurations are used)

Fig. 1-4 Binary Bit Configurations

This is the "machine" language or "object" program coding, and provides maximum utilization of memory for program storage. The coding is used infrequently during programing, but familiarity with it is required during program debugging.

The letters "A" through "F" are used to provide single character representation for hexadecimal digits with values from 10 through 15.

The instruction set for the Computer is expressed in Mnemonic (symbolic) codes, which permit a logical association of meaning, and in word addresses or other numeric parameters expressed decimally. After writing a program in this symbolic code, the programmer "Assembles" his program with a special software program which performs the function of converting his mnemonic coding to "machine language" hexadecimal coding.

The instruction, in mnemonic form, consists of an Operation Code and parameter fields. The operation code signifies the type of function, such as add or subtract. The parameter fields designate the word address, syllable, or other information needed to qualify the instruction. Three parameter fields are provided (A, B, and C); however, some instructions require no parameters, others require from one to three.

The following examples illustrate the instruction format:

	Op Code	A	B	C
Add contents of accumulator into memory word 243:	ADM	243		
Skip 3 instructions if accumulator sign flag is minus:	SK	A	-(minus)	3
Branch unconditionally to word 65, syllable 2:	BRU	65	2	

The discussion of each instruction in the following sections will indicate the range or limits of each parameter field by showing the smallest and largest number permitted; such as from 0 to 15, as "0:15".

For example:

ADM can select any word from 0 to N*; thus is shown:	ADM	0:N	
BRU can select any syllable of any word; thus is shown:	BRU	0:N	0:3

*N = high order word available in Normal Memory.

1.5 ACCUMULATOR

The system contains one word of memory which is called the Accumulator. It also contains 16 digits (or 15 digits and sign), but it is separate from the Normal area of memory set aside for the programmer ("N" words). It is not addressed by a word number, but rather, access to it is a function of certain instructions. It is a working memory location for the movement of data from one area to another. It receives all numeric data entered through the keyboard including the keys that set accumulator flags; it must contain any numeric data to be printed; it can sum up several amounts (crossfoot) and store the result in another word; it receives the product or quotient of computations; it must be used to accumulate one word of data into another; and it can be used to move alphanumeric information from one word to another. Certain of the instructions automatically destroy any prior contents of the Accumulator.

When numeric data is being stored in memory, whether in the Accumulator or in any other word of memory, the word contains the following characteristics:

NUMERIC WORD - (15 digits)

	MSD														LSD
Flags	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
															4 bits

1.6 FLAGS

The system contains 28 Flags which serve as a signal that certain conditions exist (such as minus data in the Accumulator) or that certain functions have occurred (such as depression of the Per Hundred "C" key). When such is the case, the flag is "Set"; when the condition does not exist or function has not occurred, the flag is "Reset". The "Setting" or "Resetting" of some flags is performed automatically by the system. However, most all flags may be "Set" or "Reset" by program instructions to permit "Recall" of certain conditions at a later point in the program. Instructions are provided so the program can interrogate or "test" each flag to enable selecting an alternate path of instructions when appropriate.

Each flag consists of one "bit". When the bit is "on", the flag is "Set"; when the bit is "off", the flag is "Reset".

Let's look again at the Numeric Word and see the Accumulator Flag positions specified:

NUMERIC WORD - (15 digits)

				MSD														LSD
M	C	S	-	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8	4	2	1															
BITS																		4 bits

Accumulator Flags — (M) Per Thousand — (S) Special
(C) Per Hundred — (-) Minus or Negative

If we were to examine the bit configuration for the flags, they would be represented as follows:

Bits	(-)	(S)	(C)	(M)
8	o	o	o	●
4	o	o	●	o
2	o	●	o	o
1	●	o	o	o

1.7 CATEGORIES OF INSTRUCTIONS

The instructions in G.P. 300 Programing Language are divided into the following groups for explanation and discussion.

Keyboard Instructions. Instructions permitting the use of the numeric and alphanumeric keyboards.

Printer Instructions. Those instructions for the printing of numeric and alphanumeric data, characters, symbols, print formatting, and printer positioning.

Forms Handling Instructions. Those instructions for opening and closing the forms transport, advancing one or more forms, and counting and limiting the lines on forms.

Arithmetic Instructions. Those instructions for adding, subtracting, multiplying, dividing, transferring (clearing and adding), clearing, accumulator shifting (right and left), insertion or loading of constants.

Flag Instructions. Those instructions pertaining to the setting, changing or resetting of accumulator flags, test flags, OCK flags, tape/card reader flags, tape/card punch flag, general purpose (X and Y) flags.

Index Register Instructions. Those instructions for loading, incrementing, decrementing, adding to Index Register and modifying other instructions with Index Registers.

Branch Instructions. Those instructions used for unconditional branching and jumping to or returning from subroutines.

Skip and Execute Instructions. Those instructions for Skipping or Executing instructions based on flag settings or accumulator conditions, such as zero or less than a constant.

Paper Tape (and Edge Punch Card) Reader. Those instructions that control the paper tape reader when it is used as Data Input into the computer.

Paper Tape (and Edge Punch Card) Punch. Those instructions that control the punching of data into paper tape for later rehandling by the Computer or for relaying to a Service Center for processing.

Tab Card Reader. Those instructions that control the Card Reader when it is used as Data Input into the Computer.

Tab Card Punch. Those instructions for punching data into Hollerith cards for later rehandling by the Computer or for processing by another computer.

Data Communication. Those instructions for sending, receiving, and storing messages that are being transmitted over telephone lines or other communications networks.

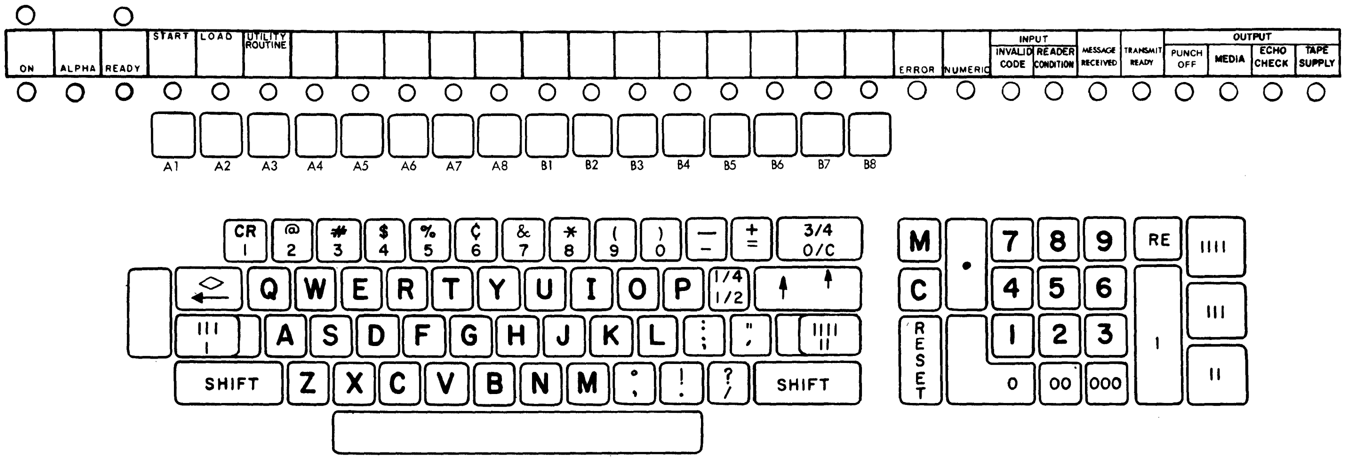


Fig. 2-1 Console Keyboard

2. KEYBOARD INSTRUCTIONS

The keyboard of the Computer Console is comprised of three sections:

Numeric Keyboard with Operation Control Keys
Typewriter Keyboard with Operation Control Keys
Program Keys and Indicator Lights

Separate instructions are utilized to control each section. These instructions and the functions of the three sections are described in the following paragraphs.

2.1 KEYBOARD BUFFER

To facilitate throughput of this operator-attended Computer, a Keyboard Buffer is available to allow the operator to continue to use the keyboard while the computer executes a program. This allows simultaneous use of the keyboard while program execution is underway, and permits several keyboard entries into the buffer at one time.

As keyboard codes are indexed (from depression of numeric, alphanumeric, control keys or program keys), these codes are entered into a Buffer:

$$\begin{tikzcd}\boxed{\text{Keyboard}} \arrow[d] \\ \boxed{\text{Buffer}}\end{tikzcd}$$

This buffer (referred to as main buffer) is capable of containing up to seven characters. Just prior to execution of an instruction is a "fetch" phase and during this phase, the main buffer is interrogated. If it contains three or more characters, they are moved from the main buffer to a buffer extension area composed of four more seven character buffers. As each instruction is executed, the main buffer is interrogated and characters are moved to the extension buffers.

When it is necessary to move data from the main buffer to the first extension buffer, it first moves any data in the first extension buffer to the second extension buffer etc.; then it moves the data from the main buffer to the first extension buffer. As succeeding data is to be moved, it continues to move data from one extension buffer to the next. In this manner, the first data entered from the keyboard is always in the highest numbered extension buffer and is kept track of by a "buffer pointer". When the program reaches a keyboard instruction and can act on the data entered, the buffer pointer indicates the location of data first entered and to be used first. Then it sequentially unloads the extension buffers and finally the main buffer until all information has been utilized.

If each extension Buffer and the main buffer contained the maximum 7 characters, the combined buffers would contain 35 characters of data. However, since the buffers are continually interrogated, moving 3 or more characters, conceivably, only 19 characters could be in the buffers before they would be considered "filled" (3 characters in each of the 4 extension buffers and 7 in the main buffer = 19.)

If the buffers are "filled", the system sound the alarm when the buffer is unloaded to the entry causing the filled condition. A depression of the RESET key will re-initiate the instruction where the error condition occurred, and the keyboard information must be re-entered.

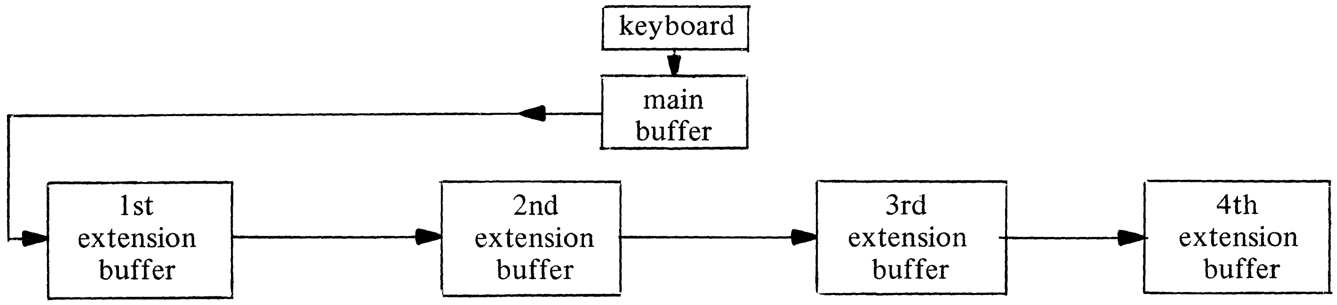


Fig. 2-2 Flow of Data from keyboard through buffer areas

It has been stated that as part of each instruction, a "fetch" phase takes place and during this phase the buffer is interrogated. Note that certain instructions might delay this interrogation because the operation eliminates frequent fetch phases:

Examples:

When a very long Alpha message is being called for from a "Type from Memory" instruction: While the message is printing, no other instruction can be executed, thus no interrogation of the buffer takes place.

When "Position Print Head" instruction for a long distance is called for: While the head is positioning, no other instruction can be executed, thus no interrogation of the buffer takes place.

In each of these examples, if the operator was indexing data on the Keyboard, only the main buffer is available since no interrogation or data moving takes place within the extension buffers, resulting in only 7 characters maximum buffering. Any characters exceeding 7 would cause the Alarm when that keyboard instruction was reached and the data would have to be re-indexed.

If the buffer extensions and the main buffer are filled and additional data is indexed, an error condition will not occur at the instant of filling the buffer. The program will continue to execute instructions, transferring data from the buffer until the character which caused the buffer to become filled is reached. At this point, an error condition will occur. A depression of the Reset key (see 2.4.01) will reinitiate the instruction during which the error condition occured, the keyboard information must be re-entered for that instruction and the program continues:

Example:

Between instructions 1 and 7 (see below), the operator entered these separate items, (54212; 2456; 15726; 345632) which were loaded in the buffer as indicated; the last item of data filled the buffer before all the digits were entered:

# 1 ------------------*		
	("542" moved to buffer area)	(Buffer Ext. 4)**
# 2 ------------------*		
	("12$\overset{O}{\underset{K}{C}}$" moved to buffer area)	(Buffer Ext. 3)**
# 3 ------------------*		
	("24" not moved)	
# 4 ------------------*		
	("2456$\overset{O}{\underset{K}{C}}$" moved to buffer area)	(Buffer Ext. 2)**
# 5 ------------------*		
	("157" moved to buffer area)	(Buffer Ext. 1)**
# 6 ------------------*		
	("26$\overset{O}{\underset{K}{C}}$3456 BUFFER FILLED 32")	(Main Buffer) **
# 7 NK 5 0	(unloads 54212)	
# 8 PN 4 1	(prints 542.12)	
# 9 POS 6 0		
#10 NK 5 0	(unloads 2456)	
#11 PN 4 1	(prints 24.56)	
#12 POS 7 0		
#13 NK 5 0	(unloads 15726)	
#14 PN 4 1	(prints 157.26)	
#15 POS 8 0		
#16 NK 8 0	(ERROR ALARM SOUNDS after unloading 3456)	

* "------------------" indicate various Instructions other than Keyboard Instructions.
** This is the way the Buffers appear after Instruction #6 was executed.

Note that all keyboard indexed data was handled correctly except for the last data when the Buffer was filled. When the Alarm sounded, program execution stopped on Instruction # 16 and only that instruction data needed to be re-indexed (345631).

If any of the keyboard commands were violated (too many digits or characters, or unenabled "RE", "C" or "M", or program keys where the instruction does not call for them) the Alarm would sound when the character that caused the error is reached. Program execution would stop on that instruction. A depression of the RESET key will reinitiate the instruction during which the error condition occurred, the keyboard information must be re-entered for that instruction and the program continues.

The buffer is also checked during any line advance instruction (section 4), after each line spacing takes place. If the instruction calls for 6 spacings, the buffer will be examined 6 times. This operation will permit 7 characters to be loaded into each buffer extension word.

2.2 NUMERIC KEYBOARD

A Numeric Keyboard instruction lights the Numeric Keyboard Indicator Lamp when the instruction is active, and it permits use of the following keys (see Fig. 2-4):

Numeric 0 through 9 (on either numeric or typewriter keyboard)

Double Zero (00) and Triple Zero (000)

Decimal Point

Reverse Entry (only with specific Numeric Keyboard instructions)

Per Hundred (C), Per Thousand (M) (only with specific Numeric Keyboard instruction)

Operation Control Keys (OCK's) 1, 2, 3, and 4

Program Keys (PK's) (when enabled by previous instruction)

Reset

Open/Close (in typewriter section)

Line Advance (in typewriter section)

Ready Push Button

If any other key is used, the Error Indicator is turned on and the program halts prohibiting the completion of the keyboard instruction

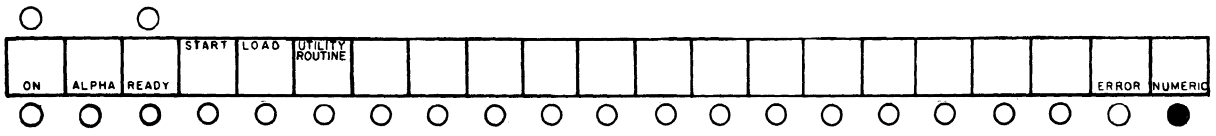


Fig. 2-3 Indicator Lamps

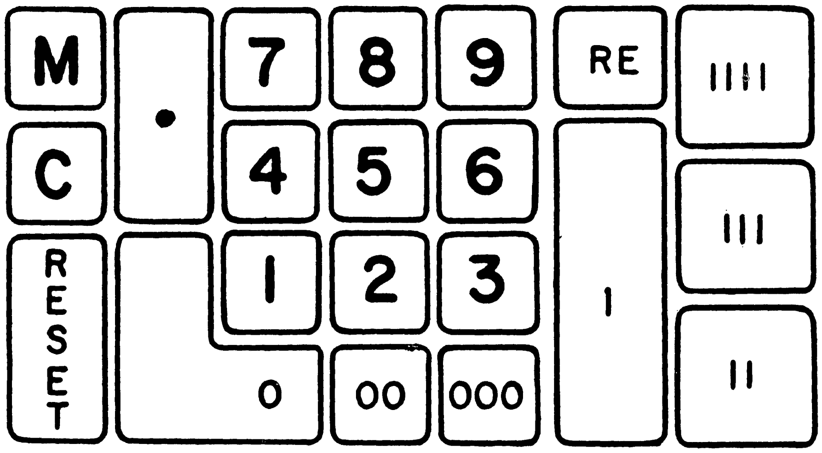


Fig. 2-4 Numeric Keyboard

2.2.01 Numeric Keyboard Instructions

The four Numeric Keyboard instructions below provide for entry of a maximum of 15 digits of numeric information into the accumulator. The "A" field specifies the maximum number of digits permitted to the left of the decimal point; the "B" field specifies the maximum number of digits permitted to the right of the decimal point. Depression of any OCK or any enabled PK (refer to 2.5) terminates the keyboard instruction and continues to the next instruction. Use of the Reverse Entry (RE) key, the Per Hundred (C) key, or the Per Thousand (M) key is permitted only with certain of the instructions as indicated.

NOTE: Printing the data is not part of the Numeric Keyboard instruction. With all Numeric instructions listed below, if printing is desired, they would be followed with a print numeric instruction (section 3). It is not necessary that the print instruction follow immediately, but may be programed at any subsequent point so long as the data is still in the accumulator (print occurs from the accumulator.) If a position instruction (POS - see section 3) is programed prior to the Numeric keyboard instruction, the numeric instruction causes the printer to move to the position specified although print will not occur until a print instruction is called for.

Following are the numeric keyboard instructions:

	Op Code	A	B
Numeric Keyboard	NK	0:15	0:15
Numeric Keyboard, permit Reverse Entry Key	NKR	0:15	0:15
Numeric Keyboard, permit C, and M Keys	NKCM	0:15	0:15
Numeric Keyboard, permit Reverse Entry, C and M Keys	NKRCM	0:15	0:15

2.2.02 Numeric Data

Data is entered using the numeral keys of the Numeric Keyboard, including the Double Zero (00) and Triple Zero (000) keys. Data may also be entered by using the numeral keys on the Typewriter keyboard. During the same instruction, these keys may be used from either keyboard alternately or in any sequence. The digits of a number are entered from left to right (most significant digit first). After the digits of the whole number have been entered, a decimal fraction is entered by depressing the Decimal Point key followed by the digits of the fraction.

Example: 12.875 is to be entered on the numeric keyboard

The "1" is entered first, followed by the "2". Then the decimal point key is depressed followed by the indexing of the "8", the "7", and finally the "5".

Note: Monetary amounts (dollars and cents) are normally treated as whole numbers for the purpose of listing the digits. Rather than depressing the decimal point key between dollars and cents with each entry, the decimal point is inserted by the print instruction (see section 3.4).

The maximum number of digits that may be stored in the accumulator is determined by the sum of the "A" and "B" fields of the numeric keyboard instruction. The "A" field determines the maximum number of digits permitted prior to the Decimal Point key recognition. The "B" field determines the number of least significant digit positions allotted for decimal digits, and in effect, it places a phantom decimal point in the accumulator. This corresponds to the maximum number of digits permitted following the use of the Decimal Point key.

The phantom decimal point is so named because a decimal key depression does not put a decimal point code in the accumulator. It simply denotes the end of a "whole number" entry and begins the "fraction" entry, thus the decimal point is assumed although not physically in the accumulator. The Print Mask inserts the decimal to be printed in the desired location (See 3.4.06).

If either the "A" or "B" limits are exceeded, the keyboard Error Indicator is turned on and the Alarm bell sounds, halting the program. If no digits are indexed, the accumulator is cleared.

Depression of the Double and Triple Zero keys produce the effect of double or triple depressions of the Zero key, that is, two or three zeros respectively are stored in the accumulator.

Under control of the "A" field, the programed number of digits enter the accumulator. Although the "B" field specifies how many digits can be entered to the right of the phantom decimal, it so happens that it specifies the digit position where the whole number enters the accumulator. The entry of each whole number digit causes the previously indexed digits to shift left one digit position permitting the digit to enter the vacated digit position. A Zero key depression counts as a digit even if used as the most significant digit entry; the Double and Triple Zero keys act in the same manner, counting two or three digits respectively.

Under control of the "B" field (following recognition of the Decimal Point key), the first digit is entered in the first position to the right of the phantom decimal point, the second digit in the second position, etc. A zero counts as a digit even if entered as the last digit after the decimal point key. It is not necessary to depress the Decimal Point key if there are no decimal digit entries, even though the "B" field permits decimals. When the "B" field is zero, the Error light will not light if the decimal point key is depressed without ensuing digit keys.

For Example:

Op Code	A	B
NK	5	2

and the operator indexes the numbers

13256

Here's what happens:

The MSD "1" is indexed, and enters the accumulator at digit position 2. The next digit "3" is indexed, and enters the accumulator at digit position 2 and shifts the "1" to digit position 3. The third digit "2" is indexed, shifting the "1" to digit position 4 and the "3" to digit position 3. The accumulator now contains:

Accumulator Digit Position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data												1	3	2		

The decimal key is now used and a "5" is indexed entering the accumulator at digit position 1 (B field minus 1.) The accumulator now contains:

Accumulator Digit Position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data												1	3	2	5	

The second decimal digit "6" is indexed and enters the accumulator at digit position 0 (B field minus 2.) The instruction is terminated and the accumulator now contains:

Accumulator Digit Position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data												1	3	2	5	6

If the same instruction is executed by indexing the same five digits and the decimal key is not used, all five digits will enter the accumulator at digit position 2 and will shift left, as previous explained, as each successive digit is indexed. When the instruction is terminated, the accumulator will contain:

Accumulator Digit Position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data										1	3	2	5	6	0	0

2.2.03 Accumulator Flags:

The Accumulator digit position 15 contains 4 flags designated "minus" (-), "special" (S), "per hundred" (C), and "per thousand" (M). These four flags are always reset ("off") at the start of any numeric keyboard or numeric entry instruction.

The -, C, or M flags may be set ("on") if the particular keyboard instruction enables the use of their related keys (RE, C, & M respectively), and if the operator depresses these keys during the instruction.

The Special flag ("S") cannot be set by depression of any keyboard key. Control of this flag is accomplished by the Flag set/reset instructions (See Section 6) which also can be used to set/reset the -, C & M flags.

The settings of the four flags transfer with the data from the accumulator to memory, and from memory back to the accumulator (See 5.2), and thus can be retained for future use in the program.

2.2.04 Reverse Entry Key

Use of the Reverse Entry Key is permitted only with an NKR or NKRCM instruction. Depression of the RE key during one of these instructions enables the entry minus data, as it causes the Accumulator "minus" (-) flag to be set (Accumulator Minus). This, in effect, allows a minus keyboard entry as one of its uses since normal keyboard amounts are plus. The RE key may be depressed in any sequence with the data digits, since it only sets the flag and does not interfere with the digits being entered.

M	C	S	-	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

("-" Sign flag is turned "ON" when RE key is used with Numeric Keyboard)

If the RE key is not depressed during Numeric Keyboard instructions, the Accumulator sign flag is reset (Accumulator is Plus).

Use of the Reverse Entry Key with an NK or NKCM instruction turns on the Error Indicator and sounds the alarm, halting the program (refer to 2.2.06).

2.2.05 Per Hundred (C) and Per Thousand (M) Keys

The use of C and M keys is permitted only with an NKCM or NKRCM instruction. Depression of the C key sets the C flag in the accumulator; and depression of the M key sets the M flag in the accumulator. Depression of both keys in the same instruction will set both flags (bits are "on"). If either or both keys are not used, the corresponding flags are reset (bits are "off") when an OCK or PK terminates the instruction.

Use of either the C or the M key with an NK or NKR instruction turns on the Error Indicator and sounds the Alarm, halting the program (refer to 2.2.06).

M	C	S	-	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

("M" and/or "C" bits are "ON")

Whenever a keyboard instruction enables the use of the C and M keys, and they are indexed, they will only set a flag in the accumulator flag position. At that point, they have no effect on the decimal positioning of the value. It is necessary to make provisions in the program to check whether these flags have been set so that either the value can be shifted after multiplication, or the shift register (section 5) can be set accordingly for multiplication. It is necessary to check both flags so that the program can determine which key (or keys) have been used.

2.2.06 Keyboard Error Light (Numeric Keyboard Instructions)

As mentioned above, the Keyboard Error light is turned on when any of the following conditions occur:

a. Depression of a numeral key which causes the amount indexed to exceed either the maximum number of digits permitted by the "A" field or the maximum number of decimal digits permitted by the "B" field of a numeric keyboard instruction.

b. The RE, C, or M key is depressed during a numeric keyboard instruction that does not permit their use.

c. A typewriter key is depressed (other than keys 0 through 9, open/close key, line advance key, or the typewriter OCK's) during a numeric keyboard instruction.

d. A Program Key is depressed which has not been enabled (refer to 2.5.02).

e. The numeric keyboard instruction is initiated when the capacity of the keyboard buffer has been exceeded and when the valid codes in the buffer do not terminate the instruction.

The Keyboard Error light, once turned on by any of the above conditions, remains on through continued keyboard entry and is turned off only upon depression of the Reset Key (see 2.4.01) or the Ready push button. When the Keyboard Error light is on, all keys are disabled from performing their function, except the Reset or the Ready push button. The entire entry must be reindexed following use of the Reset key.

In the case of an overloaded buffer, the Reset key must be depressed after the error indicator occurs.

2.2.07 Numeric Keyboard Programing Considerations

Numeric Keyboard instructions do not cause printing. They only allow numeric information to be entered into the accumulator and/or memory. To print, a print command must be specified (refer to section 3).

Although the decimal key is depressed to end a whole number and start the fractional entry of an amount, the decimal key does not enter a code into the accumulator. It simply determines where, in the accumulator, the digits are entered. The Print Mask will physically cause the decimal point to print properly.

Typical uses of Numeric Keyboard instructions:

Numeric Keyboard (NK):

Allows indexing of any number - Invoice number, Reference Number, Check Number, Product Number, etc. Entry of plus data is enforced with this instruction.

Numeric Keyboard, Permit Reverse Entry (NKR):

Allows indexing of an amount - accounting entries (dollars and cents), quantities, units, etc. - where both plus and minus accumulations or Arithmetic are received.

Numeric Keyboard, Permit Per C, Per M (NKCM):

Allows indexing of a Price or Cost figure that would be influenced by Per Hundred or Per Thousand factors. Entry of plus data is also enforced with this instruction.

Numeric Keyboard, Permit Reverse Entry, Permit Per C, Per M (NKRCM)

Allows indexing of Prices, Cost figures, Quantities, etc., where both Per Hundred and Per Thousand factors must be considered and Reverse Entry Arithmetic must be available.

2.3 TYPEWRITER KEYBOARD

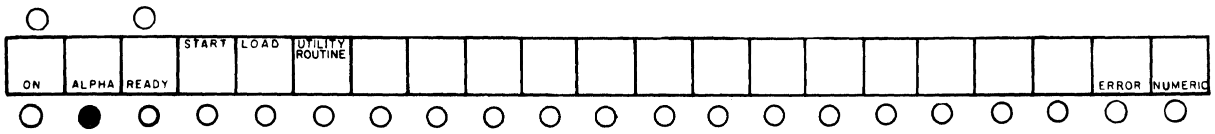


Fig. 2 - 5 Indicator Lamps

A Typewriter Keyboard instruction lights the Typewriter Keyboard Indicator Lamp (ALPHA) when the instruction is active, and it permits use of the following keys (see Fig. 2-6):

Typing (44 Keys:	44 lower case characters;
	20 upper case characters)

Shift
Space
Backspace
Open/Close
Line Advance
Operation
Control Keys (OCK's) (on both keyboards)
Enabled Program Key (A1 through A8, B1 through B8)
Reset
Ready Push Button

If any other key code is processed, (because of incorrect operator key depression) the error indicator is turned on.

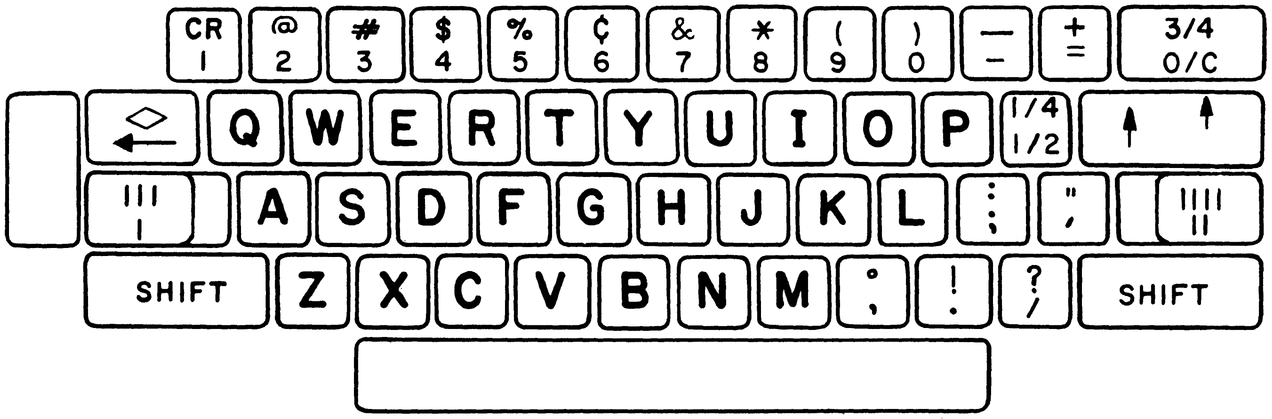


Fig. 2-6 Typewriter Keyboard (United States)

2.3.01 Typewriter Keyboard Instructions:

The following are the Typewriter instructions:

TK	Type
TKM	Type Into Memory
EAM	Enter Alpha Into Memory (Non-Print)

The type instruction provides for typing and printing (except EAM) a maximum of 150 alphanumeric characters. It is completed by the depression of an OCK or an enabled PK (refer to 2.5). If an OCK is used, it sets its appropriate "Flag" which can be tested further in the program (until reaching another Keyboard instruction.) Printing of the first character will begin at the position of the print head. If printing in a specified area is desired, the print head must be pre-positioned to the beginning left-hand positon of the area before the Typewriter instruction is reached in the program. Each character typed escapes the print head 1/10th inch. This position is recorded in the "Position Register." The length of the printed field must be less than or equal to the value contained in the "A" field. The maximum field is 150 characters ("A" = 150). If typing of more than the number of characters specified in the "A" field is attempted, the Error Indicator is lit, and further typing is prevented.

After this error condition, the depression of the Reset key effectively cancels the keys depressed in error, permitting the use of an OCK or enabled PK. If Reset key is depressed during a Type instruction without an error condition, the instruction will be re-initiated, and the print head will return to the start (see 2.4.01).

A Typewriter instruction that prints will print with the ribbon in the black color position, unless it was preceded by a Red Ribbon (RR Section 3) since the last printing operation, in which case the Typewriter instruction prints in red.

If the Ready push button is depressed before, or during, the execution of any Typewriter instruction, any characters still in the keyboard buffer will be ignored, and the system will return to the ready mode.

The three Typewriter instructions and their specific functions are reviewed below:

2.3.02 Type Instruction

	Op Code	A	B
Type	TK	0:150	

The Type instruction provides for typing and printing a maximum of the number of alphanumeric characters specified in the A field. It is completed by the depression of an OCK or an enabled PK (2.5).

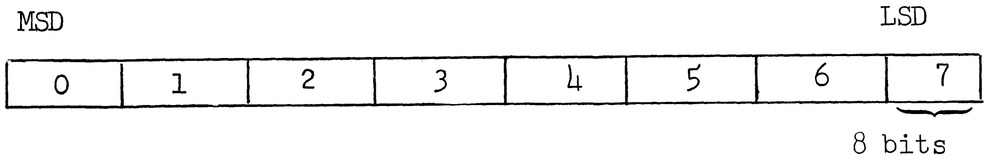
2.3.03 Type Into Memory Instruction

	OP Code	A	B
Type into Memory	TKM	0:150	

The Type into Memory instructions (TKM) differs from the Type instruction (TK) in that, in addition to printing alphanumeric information, the characters are also stored in memory. Note that the space character (escape) is considered a print character in that it too stores a code in memory. However, the codes for Backspace, Open/Close, Line Advance, OCK's, and Program Keys are not stored in memory.

The code, for each key depressed before instruction termination, is stored in memory with the first character stored in the most significant character location of the word specified by the Keyboard Base Register. (See 2.3.06 for loading the Keyboard Base Register.) Up to 8 characters can be stored in a single memory word:

ALPHA WORD - (8 Characters)



If typing continues beyond 8 characters, the information is entered in the next sequential word of memory. When the instruction is terminated (by depression of an OCK or an enabled program key), an "end alpha" code (value of 00) is entered in the character position following the last character stored and the remainder of the word is filled with zeros. Thus, if 8 characters are typed, the End Alpha code is entered in the first position of the next memory word in sequence. (Note that all 8 characters of this second word are set to zero.) A maximum of 150 alphanumeric characters can be stored in memory in a single typing sequence.

The depression of the Backspace key effectively removes the last typing key code from memory. Backspacing does not occur past the first typing position.

On a TKM instruction, the first word is not cleared immediately. The termination of the TKM instruction with an OCK will clear the unused portion of the word, but until such action is taken the word contains whatever is in it from the last time it was used. If no typing is done and the TKM instruction is terminated by an OCK, the word is all clear. If the number of characters typed to Memory is such that the next word is referenced, that word will not be cleared until an OCK is used. If exactly 8 characters were entered and then an OCK was used, the 2nd word would contain all zeros.

2.3.04 Enter Alpha Into Memory Instruction

	Op Code	A	B
Enter Alpha into Memory	EAM	0:150	

This instruction is identical to the TKM instruction except that no printing occurs, the print head does not escape, nor does the Position Register advance.

2.3.05 Typing, Space, Backspace, Shift Keys

Recognition of any of the typing key codes causes the corresponding character to be printed.

The hyphen/underline key has a second actuation point which causes the key to repeat at the keyboard cycle rate (15.5 codes per second) until the key is released.

The space bar causes the printer-carrier to escape 1/10" to the right. When the key is depressed to a second actuation point, it is repeated at 15.5 spaces per second until the key is released.

The backspace key causes the printer-carrier to space 1/10" to the left. When depressed and held to a second actuation point, it repeats at 15.5 spaces per second until the carrier reaches the location at which it began for this Type instruction or until the key is released, whichever is first.

On TKM and EAM (see above), each backspace code erases one character of the alphanumeric entry into memory, unless there is no character to erase.

Recognition of either shift key with a key having an upper case character causes the corresponding upper case character to print. Keys having only lower case characters when used with a shift key cause the lower case character to print.

Control keys 1 and 2 used with a shift key cause the corresponding function for control keys 3 and 4 to be performed.

On consoles with split platen, depression of a shift key with the Line Advance Key causes the right platen to advance; without shift key, the left platen advances.

Repeating keys repeat the code of the upper case character at 15.5 codes per second when used with a shift key.

2.3.06 Load Keyboard Base Register

	Op Code	A	B
Load Keyboard Base Register	LKBR	0:N	

The LKBR instruction specifies the starting memory location into which information will be transferred for all succeeding TKM and EAM instructions (until another LKBR instruction is executed). The "A" field specifies the word location.

The keyboard base register contains the data that is loaded into it until a subsequent LKBR instruction loads new data into it.

2.3.07 Keyboard Error Light (Typewriter Keyboard Instructions)

The keyboard error light is turned on when any of the following conditions occur:

a. Type instruction is initiated and valid codes in the buffer do not include a code to terminate (OCK, PK) the type instruction prior to the signal that the buffer capacity has been exceeded.

b. A key on the numeric keyboard is depressed when executing the Type instruction.

c. More than the number of keys specified by the "A" field have been depressed.

d. A non-enabled program key is depressed.

When the keyboard error light is on, no key except the Reset key (see 2.4.01) and the Ready push button will perform its function.

If the Ready push button is depressed prior to, or during the execution of, a Typewriter instruction, any characters still in the buffer will be ignored and the system will return to the Ready mode.

2.3.08 Typewriter Keyboard Programing Considerations

Typical uses of Typewriter Keyboard instructions are reviewed:

Type (TK)

This instruction is used wherever an Alpha description is desired, but no storage of the data is necessary. Unlike the numeric instructions, TK prints as typing is accomplished.

The following two instructions must be preceded by an LKBR instruction to identify the word in memory where Alpha is to be stored:

Type into Memory (TKM)

This instruction is used when an Alpha message is desired several times during a program. As an example, when continuation forms are necessary to complete a customer's invoice during billing it is often desired to repeat the customer's name on each form. TKM is programed when the name is typed on the first page; the name then prints from memory automatically (by program) on each succeeding page.

Enter Alpha into Memory (EAM)

This instruction would be desireable when typing to memory must be suppressed from printing at the time typing is done, although it is to be called for later in the program and printed at that time.

2.4 MISCELLANEOUS CONSOLE KEYS

2.4.01 Reset Key

Depression of the Reset key at any time during a Numeric Keyboard instruction re-initiates that instruction and clears all previous entries that pertain to that instruction. If the Error light is on, it is turned off.

The depression of the Reset key will re-initiate a Typewriter Keyboard instruction and position the print head to the start position, if the system is not in an error condition. If the system is in an error condition, depression of the Reset key will remove the error condition; the head will not move. If the system is in an error condition and it is desired to re-initiate the TK instruction, 2 depressions of the reset key are required.

On the TKM and EAM instructions, the Reset key will remove an error condition (if one was present) without moving the print head. If Reset key is depressed when no error is present, such as a second depression following an error key, the instruction is re-initiated, causing the print head to return to the starting position for that instruction.

2.4.02 Open/Close Key

Depression of this key causes the forms transport to open if closed, or close if open. The Open/Close key is operative in Ready mode and during Keyboard instructions, both alpha and numeric. If this key is depressed following the opening of the transport by an Open Instruction, the transport closes and advances the number of lines specified by the Open Instruction (see 4.1.01).

2.4.03 Line Advance Key

A single depression of this key causes the forms contained within the forms transport to be vertically spaced 1/6". Depression to the second actuation point causes spacing repeatedly at 15.5 lines per second until the key is released. Each 1/6" line advance resulting from depression of this key increments the associated Forms Count Register. The Forms Count Register is used to determine programmatically certain positions on a form, and is discussed in section 4.

On machines equipped with split platens, holding down the Shift Key while depressing the Line Advance Key advances the right platen; without the Shift Key, the left platen advances.

Advance Key is operative in the Ready mode and during Keyboard instructions, both alpha and numeric.

2.5 OPERATION AND PROGRAM CONTROL KEYS

The system contains Operation Control Keys (OCK's) and Program Select Keys (PK's) both of which terminate keyboard instructions and permit operator direction of the program.

2.5.01 Operation Control Keys (OCK's) 1, 2, 3, 4 (See Keyboard Diagram Fig. 2-3)

Depression of any of the Operation Control Keys (on either the Numeric or Typewriter Keyboard) terminates the Numeric or Typewriter Keyboard instruction, sets the corresponding OCK flag, resets the other OCK flags, and causes the next instruction in the program to be executed. All Program Key Indicators are turned off (see 2.5.02).

Each Control Key has a flag associated with it in the system. When the flag is set, it is "turned on"; when reset, it is "off". By means of these flags, the system can "remember" (until the next keyboard instruction) which OCK was depressed, and therefore permit directing the program to alternate routines when required. Instructions are provided to check these flags, and will be discussed in a following section.

There are only two OCK's in the typewriter keyboard. OCK1 operated with the shift key becomes an OCK3 operation. OCK2 operated with the shift key becomes an OCK4 operation.

2.5.02 Program Keys and Related Indicators

Program keys (PK's) can be used to terminate Numeric or Typewriter Keyboard instructions. They may be used only when called for (enabled) by program instruction, which must be prior to the keyboard instruction:

<u>Enable Program Keys Instructions:</u>

	Op Code	A	B
Enable Program Key Group A	PKA	1,2,3,4, 5,6,7,8	
Enable Program Key Group B	PKB	1,2,3,4, 5,6,7,8	

Program Key Group A refers to programs key A1 through A8. Program Key Group B refers to program keys B1 through B8. Any one or any combination of the eight keys of one group can be enabled by one instruction.

Associated with each program key is a keyboard indicator light:

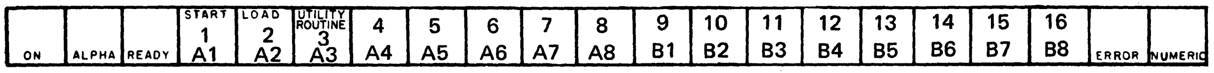


Fig. 2-7 Program Key Indicator Lights

The "A" field of the PKA or PKB instruction turns on the designated indicators and enables the appropriate PK. All indicators are extinguished at the completion of the next Numeric or Typewriter keyboard instruction, whether an enabled PK is used or an OCK is used. The use of a PK to terminate a keyboard instruction resets all OCK flags.

The function of a Program Key is to select and execute one instruction, programed and stored in an area of memory called a Program Key Table. This instruction is executed before resuming the normal instruction sequence in the program. Often the one instruction is a "Branch" instruction allowing the program to be directed to one of several routines or an alternate routine according to which PK was used.

The instruction may not necessarily be a "Branch", however, if the program variation can be accomplished with one instruction, some other single instructions that may be used include "Load Index Register", "Print Alphanumeric", "Print Character", and "Set or Reset Flag" commands. Each of these commands will be discussed in later sections.

The PKA or PKB instructions may enable any or all of the program keys. However, the keys that are enabled and the indicators that are lit are determined by the last PKA and/or PKB instructions executed since the last keyboard instruction. All PK's that are desired must be specified by the PK command for the group (PKA or PKB), as a later command calling for the same group will void the effect of the earlier command for that group.

Example: If the program calls for PKA 2 and then subsequently calls for PKA 5 (Note both call for Group A), both instructions occurring between keyboard instructions, when the next keyboard instruction is reached, only the PKA 5 indicator will be lit and only the 5 key will be enabled. However, if PKA 2 is executed followed by a separate instruction PKB 5 (different Groups are called for), then both keys will have their indicators lit and both keys will be enabled.

Depression of a non-enabled program key lights the error indicator and sounds the alarm without further executing the program. To reestablish the correct condition, the Reset key must be depressed, which re-initiates the instruction (see 2.4.01).

When in the Ready Mode, PK's A1, A2, and A3 have specially assigned functions and are always enabled (refer to Part I, 5.3). These functions take precedence over any functions programed for these keys in the PK table (only in the Ready Mode).

2.5.03 Program Key Table Selection

An instruction "Load Program Key Register" (LPKR) is used to establish the first word of a four word Program Key Table (16 instructions.)

	Op Code	A	B
Load Program Key Base Register	LPKR	0:N	

The LPKR instruction loads the program key register with the contents of the "A" field. This register determines the location of the program key table in memory for all subsequent program key operations until another LPKR instruction is used. The "A" field is the starting word location, or base address for the four words in succession.

Each PK has one instruction in the table. PKA 1 utilizes the "0" syllable of the first word (base address); PKA 2 utilizes the "1" syllable; PKA 3, the "2" syllable; PKA 4, the "3" syllable; PKA 5 utilizes the "0" syllable of the second word of the table (base address + 1); PKA 6, the "1" syllable, etc.; PKB 1 utilizes the "0" syllable of the 3rd word (base address + 2): PKB 2, the "1" syllable, etc.

1st word (Base Address)	0	PKA 1	# 1
	1	PKA 2	# 2
	2	PKA 3	# 3
	3	PKA 4	# 4
2nd word (Base Address + 1)	0	PKA 5	# 5
	1	PKA 6	# 6
	2	PKA 7	# 7
	3	PKA 8	# 8
3rd word (Base Address + 2)	0	PKB 1	# 9
	1	PKB 2	# 10
	2	PKB 3	# 11
	3	PKB 4	# 12
4th word (Base Address + 3)	0	PKB 5	# 13
	1	PKB 6	# 14
	2	PKB 7	# 15
	3	PKB 8	# 16

There may be more than one Table in memory at a time for the PK's. The LPKR instruction must be used prior to changing the functions of the PK's, to establish which table will be active. To re-establish the first table, another LPKR would be used to designate the appropriate words of memory.

The LPKR instruction is normally programed at the start of a program in the "Initialize" routine, although as just discussed, it may be a part of the actual program. Normally, 16 PK's functions are adequate for a program.

2.6 KEYBOARD ERRORS SUMMARY

During the execution of a keyboard instruction, an error condition can arise as a result of an invalid key depression. When this occurs, the error indicator is illuminated and the alarm will sound. In all cases, depression of the Reset key removes the error condition. The causes and recovery from error conditions are described below.

2.6.01 Numeric Keyboard

Causes:

The integer entry (number of digits preceding the decimal point) exceeded the number of digits specified by the "A" field of the instruction.

The decimal entry (number of digits following decimal point) exceeded the number of digits specified by the "B" field of the instruction.

The C (per hundred) or M (per thousand) key was depressed during the execution of an NK or NKR instruction.

The reverse entry key was depressed during the execution of an NK or NKCM instruction.

A typewriter key was depressed (other than Open/Close, Line Advance or Typewriter Numerics) during execution of any Numeric Keyboard instruction.

A non-enabled program key was depressed.

Recovery:

The Reset key, when depressed, will remove the error condition, turn the error indicator off and re-initiate the original keyboard instruction.

2.6.02 Typewriter Keyboard

Causes:

The number of printable characters exceeded the number of characters specified by the "A" field of the instruction.

A key on the numeric keyboard was depressed.

A non-enabled program key was depressed.

Recovery:

One depression of the Reset key will remove the error condition, turn the error indicator off, and allow the execution of the instruction to be completed.

A second depression of the Reset key will re-initiate the Type instruction including positioning the print head to the position it occupied at the beginning of the instruction.

Ready Push Button:

If the Ready Push button is depressed while in an Error condition, the error condition will be removed and the system will return to the Ready mode. If Reset key is depressed while in the Ready mode, the instruction is re-initiated.

3 PRINTING INSTRUCTIONS

All printing is accomplished by a 64-character, serial ball printer riding horizontally in front of the platen on a mechanism, called a servo-carrier. Printing takes place one character at a time at the rate of 20 characters per second. The basic character set is shown below.

Printer Character Set
United States

```
1234567890
ABCDEFGHIJKLM
NOPQRSTUVWXYZ
-=CR@#$%¢&*()_
+¾°½¼;':",./°!?
```

Instructions are provided to print in three modes: 1) Alphanumeric printing of data either from keyboard entry (see Section 2) or from memory. When printing in this mode, the left most digit position of the field is always the beginning print position no matter how long the Alphanumeric message (referred to as "left justified"). 2) Printing data either from keyboard entry or from memory. In this mode, the printer moves directly to the most significant digit present and printing of the numeric value is automatically "right justified" in the field; that is, the right hand digit (least significant digit) position is always aligned no matter how large the numeric value. 3) Printing of a single character, left justified, with the actual character specified by the instruction. Following are examples of each type of printing mode:

Alphanumeric:	Numeric:	Single Character:
Any Customer Name	1,250.00	*
1234 Any Street	9.00	A
Any City, U.S.A.	.25	@
48152	6,345,786.41	
"Left justified"	"Right justified"	"Left justified"

3.1 PRINTER POSITIONING

The Serial Ball Printer (hereafter referred to as the "Printer" or "Print Head") can be positioned, by instruction, to any of the positions "1" through "150" (left to right) on the 15" platen prior to the print instructions at an average speed of 20" per second. A printing sequence or positioning instruction that attempts to position the printer beyond position "150" will cause the System to go to the "Ready Mode". The direction of positioning can be to either the right or left, depending on the prior position of the print head.

Before printing data in any of the three modes, the left most position of the print field must be designated by loading the position number in the "Position Register". The "Position Register" directs the printer to each position.

3.1.01 Position Register Instruction

	Op Code	A	B
Load Position Register	POS	1:150	

The Position Register is loaded with the value of the "A" field. This corresponds with the actual location that the printer is to be situated. The printer does not move until the program reaches an instruction which specifies that a character is to be printed, or until a keyboard instruction is reached.

The actual positioning occurs according to the print instruction as follows:

a. Print Alphanumeric:

   The printer moves unconditionally to the position specified by the Position Register.

b. Print Numeric:

   The printer is moved directly to the position of the first significant digit. This position is determined by adding the number of preceding zeros and punctuation suppressed (not significant zeros) from printing to the contents of the Position Register. If the print mask does not provide for suppressing preceding zeros from printing, then preceding zeros are significant and are not added to the Position Register. If the numeric value of the accumulator is zero and the print mask suppresses zeros from printing, then no printing occurs and the printer is not moved at all from its preceding location; however, the Position Register is incremented by the number of zeros and punctuation. This is necessary to correctly align the print of any character (called for by a "Print Character instruction) following a Print Numeric instruction.

c. Print Character:

   The printer moves unconditionally to the position specified by the Position Register.

d. When a program reaches a Numeric Keyboard instruction, the print head will normally move directly to the position specified by the Position Register. However, if the operator gets ahead of the program, and has completed indexing the data for the next entry before the program reaches that NK instruction, the buffer would contain the entry with its terminating OCK (or PK) and would cause the system to bypass positioning the print head for that NK instruction; instead, the positioning would wait until a print instruction is reached (usually following the NK) and would go to the most significant digit position, provided that a significant digit(s) was actually present. If no significant digit is sensed, the print head would not move at all from the original location. It is conceivable that a series of keyboard listings could be stepped through without ever moving the print head if a zero entry (and a terminating OCK) were rapidly indexed for each (a zero entry requires no key depression other than an OCK or PK).

e. During a Typewriter Keyboard instruction, the print head is moved before printing the first character (since printing is a function of the typewriter keyboard instructions, except EAM).

3.2 RIBBON SHIFT

Printing of data normally is with the ribbon color black, except for certain print instructions that cause minus amounts to print in red. However, a ribbon shift instruction is provided to change the normal color of printing (usually to red, but see below).

3.2.01 Ribbon Shift Instruction

	Op Code	A	B
Red Ribbon	RR		

The RR instruction is used to change the ribbon color of the next printing instruction. After executing the next print or type instruction (even though printing may not actually occur), the effect of the RR instruction is removed.

The ribbon color will be opposite (not necessarily red) to the color normally expected from the data and type of next print instruction.

3.3 ALPHANUMERIC PRINTING FROM MEMORY

Alphanumeric information can be stored in memory using the Type into Memory (TKM) or Enter into Memory (EAM) instruction (Section 2), to be retained for subsequent printing with a print from memory instruction. The 64 print characters and the space code are stored. The codes for BACKSPACE, OPEN/CLOSE, LINE ADVANCE, and the codes for OCK's and PROGRAM KEYS are not stored in memory.

3.3.01 Print Alphanumeric Instruction

	Op Code	A	B
Print Alphanumeric	PA	O:N	

The PA instruction prints alphanumeric information from memory starting with the first character (the most significant digit position) in the memory location specified by the "A" field. Printing (space included) continues until an "end of alpha" code is encountered, regardless of the number of words used. (See Section 2, Type into Memory and Enter into Memory).

For the PA instruction, the ribbon will be in the black position. If an intervening Red Ribbon instruction has been executed, the ribbon color will be red.

3.4 NUMERIC PRINTING

Numeric values to be printed must be contained in the accumulator and can be up to 15 digits. It is not possible to print numeric values directly from memory.

The Print Numeric instructions specify in the parameter fields the starting digit position for printing (Pointer digit position), and a printing format (mask) to control the type of printing.

Printing is justified right in the numeric print field. The field size is the maximum number of digits and punctuation characters permitted by the combination of the pointer location and the selected mask. The field size plus the value in the Position Register determines the right-most (or LSD) printing position. This permits consistent decimal alignment of all numeric values, printed in a given format.

3.4.01 Print Numeric Instruction

	Op Code	A	B
Print Numeric	PN	0:14	0:15
Print Numeric, Shift Ribbon if Minus	PNS-	0:14	0:15
Print Numeric, Shift Ribbon if Plus	PNS+	0:14	0:15

The PN instruction prints the contents of the accumulator with the ribbon in the black position, regardless of sign, unless a Red Ribbon instruction has been executed since the last Print or Type instruction, in which case the ribbon color will be red.

The PNS- instruction shifts the ribbon if the sign of the Accumulator is negative. It is normally used for a "red ribbon if minus" operation.

The PNS+ instruction shifts the ribbon if the accumulator is positive, and is normally used for a "red ribbon if plus" operation.

A previous Red Ribbon instruction complements the ribbon color; that is, if used just before PNS-, a minus amount would print black; if used just before PNS+, a plus amount would print black.

3.4.02 Accumulator Pointer ("A" Field)

The accumulator digit positions are numbered from zero through 14 (from LSD to MSD). The "A" field contains the Accumulator position number or Pointer for the most significant digit to be printed (independent of what is contained in the format mask). All positions higher than the position designated by the "A" field are ignored.

Since the digit positions are numbered zero through 14, if a maximum of 5 digits would ever be printed at a given position, the "A" field should contain a "4" and 5 digit positions would be allowed to print (0, 1, 2, 3, and 4). If 9 digits were to be provided for, an "8" should be entered in the "A" field.

Examples:

Op Code	A	B
PN	5	0

The "5" in the "A" field signifies that a maximum of 6 digits can be printed from this instruction. If the accumulator contained more than 6 digits, the high-order (most significant) digits would be lost from printing as digits to the left of the pointer are disregarded.

3.4.03 Print Format (Mask) ("B" Field)

The "B" field of the instruction specifies the mask to be used during printing. The value of the "B" field determines the location of the mask relative to the base address of the mask table. The first print mask word (the one stored in the Base Address word in the mask table) is called for by a "0".

3.4.04 Print Numeric Base Register

Mask words are grouped into a table in memory. A Print Numeric Base Register contains the base address or starting word of the table, and is loaded with a "Load Print Numeric Base Register" instruction. The location of a mask word is the specified mask number relative to the base address contained in the register. For example, if the Print Numeric Base Register is loaded with a value of 200, then mask #0 would be located in word 200, mask #1 in word 201, etc.

WORD 200	MASK "0"
WORD 201	MASK "1"
WORD 202	MASK "2"

Fig. 3-2 Print Mask Table

A maximum of 16 different masks can be referenced relative to the base address value in the Print Numeric Base Register. If more than 16 masks are required, the register must be reloaded with a new value before referencing the masks in the second table (by calling for an LPNR instruction), and then reloaded with the original value before re-using the first 16 masks by calling for another LPNR instruction. If fewer than 16 masks are required (normally the case), those words of memory never referenced as mask numbers may be used for any other purpose, such as to contain program instructions and constants.

3.4.05 Load Print-Numeric Base Register Instruction

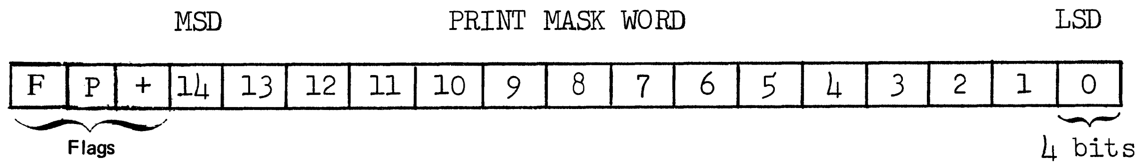
	Op Code	A	B
Load Print-Numeric Base Register	LPNR	O:N	

The Print-Numeric Base Register is loaded with the value of the "A" field to designate the word number of the base address of the mask table for all subsequent Print-Numeric instructions (until another LPNR instruction is executed).

3.4.06 Print Format (Mask) Word

When numeric data is to be printed, consideration must be given to whether or not the entire contents of the Accumulator should be printed, particularly leading zeros, and how the printed digits are to be punctuated. Any uncontrolled printing of the accumulator contents would print all leading zeros as well as any significant digits (all digits to the right of the first non-zero digit), and would provide no punctuation to indicate the decimal point, etc.

The mask enables printing in varied formats to provide a customary and acceptable appearance of printed numbers. The mask word consists of control codes and control flags. The control codes are entered into the mask word in digit positions 0-14. They control the printing (or non-printing) and punctuation of each corresponding accumulator digit. Mask flags are entered into digit position 15 of the mask word, and are used to modify the effects of the control codes.



Numeric Print Flags:	(F) Safeguard
	(P) Punch leading Zeros (into tape or card)
	(+) Suppress Punctuation

Fig. 3-3 Showing the Numeric Print Flags
Stored in a Mask Word

For example, assume that the accumulator contains the following data in positions 14 through 0:

0 0 0 0 0 0 5 1 5 6 0 4 3 0 0

This data may represent dollars and cents, a whole number, a whole number with a decimal fraction, etc. Thus, it may be desired to print this data in any one of several ways:

a. ¦0 0 0 0 0 0 5 1 5 6 0 4 3 0 0¦

This format indicates a whole number showing every one of the digits up to an assumed maximum size, regardless of whether the digits are significant. Numbers printed in this format can be considered as having one field.

```
b.        |0,000,005,156,043|.00|
```

This format uses monetary punctuation, and can be considered to have two fields: whole numbers (dollars) and fractional numbers (cents). It shows every digit of each field, up to an assumed maximum size, without regard to significant digits.

```
c.        |         5,156,043|.00|
```

This format also uses monetary punctuation with two fields, but all leading zeros in the left field (whole numbers or dollars) are suppressed from printing; All digits in the right field (fractions or cents) are shown, as is customary for dollar monetary amounts.

```
d.        |         5,156,043|   |
```

This format uses numeric punctuation which provides for two fields: whole numbers and fractions. However, non-significant fraction zeros, are suppressed from printing. Also, leading zeros in the whole number field are suppressed, which is customary in printing numbers. All remaining digits to the right of the first "5" (first significant digit) in the whole number field will print, even if a zero, because they are now considered significant (significance was established by the "5").

```
e.        |         515 604 300|
```

This indicates a numeric format in which spaces have been substituted for comma punctuation, and can be considered to have one field - whole numbers only. Leading zeros have been suppressed.

All of the above formats and many others are permitted, depending on how many mask words are utilized and what type of mask codes are in the words.

From the above examples, it can be seen that a mask word can be organized to recognize "field" when printing numbers, to cause the suppression (non-printing) of leading and/or terminal zeros. The suppression of leading zeros is very common in printing both numbers and monetary amounts; The suppression of terminal zeros is common in printing decimal fractions or numbers that may or may not have a suffix number:

For example:

```
Decimal fractions: |  30|.12  |     (with a fraction value)
                   |  30|.008 |      "    "    "    "
                   |  30|     |     (with no significant fraction value)

Product Number     |45678|.112|     (with a suffix)
                   |45678|    |     (without a suffix)
```

A "leading zero suppression field" would be set up in the mask word to correspond to each digit position of the accumulator that is desired to be so affected, left to the pointer. A "terminal zero suppression field", when desired, would also be set up in the mask word corresponding to certain digits.

3.4.07 Mask Control Codes (Mask)

Some Mask Control Codes cause a digit to print only if it is significant (value greater than zero); or, if a zero value, to print only if another Accumulator digit other than zero has has established significance in this printing instruction. This permits suppressing the printing of leading (preceding) or terminal zeros. Leading zeros to the left of a digit position may be suppressed from printing (leading zero suppression field) and /or terminal zeros to the right of a digit position may be suppressed from printing (terminal zero suppression field). However, some mask control codes cause the digit to print regardless of significance; these are referred to as "UNCONDITIONAL". Or, the code may cause the digit to be ignored regardless of significance; which is referred to as "IGNORE".

Leading zero suppression field: Significance is established if the digit to be printed is greater than zero, or if any preceding digit of the accumulator has printed (from any mask code) left to the pointer, during this print instruction. Note that the accumulator may contain significant digits left of the pointer which would not establish significance since anything to the left of the pointer is ignored; however, "unconditional" mask codes to the left of "leading zero suppression" mask codes, up to the pointer, will establish significance and therefore eliminate zero suppression. "Unconditional" mask codes for digit positions to the left of the pointer are ignored.

Terminal zero suppression field: Significance is established the digit to be printed or any digit to the right of it in the field is greater than zero. Significance is dropped when all remaining digits in the field are zero. Field size includes any "terminal zero suppression" mask codes used in succession, but stops when other codes are used.

The examples below illustrate the filtering and control that a mask word and its control codes exert over the printing of each accumulator digit:

Sample: Printing decimal fractions allowing for a 7 digit whole number and 3 decimal places:

Example #1:

```
                              pointer ──┐whole number │ fraction│
    Instruction:    PN  9  1            ├────── field→│←field──→│
                                        ↓
    Accumulator:              0 0 0 0 0 0 0 0 1 6 5 0 1 2 0

    Mask  # 1:                Z Z Z,Z Z Z,Z Z·Z,Z Z Z.X X X

    Printed Result:                           1,6 5 0.1 2
```

Mask#1 provides 1 field for whole numbers and 1 for decimal fractions: The "Z" and "Z," mask codes establish a "leading zero suppression field" from digit position 3 through the pointer in position 9, and the proper comma punctuation for whole numbers; Thus, digit positions 7, 8, & 9 are suppressed because they are not significant. The "X" and ".X" mask codes establish a "terminal zero suppression field" from digit position 0 through 2 and provide the decimal point, thus digit position zero is suppressed because it is non-significant.

Example #2:

```
                              pointer ──┐whole number │fraction│
    Instruction:    PN  9  1            ├────── field→│←field─→│
                                        ↓
    Accumulator:              0 0 0 0 0 0 0 0 1 6 5 0 0 0 0

    Mask  # 1:                Z Z Z,Z Z Z,Z Z Z,Z Z Z.X X X

    Printed Result:                           1,6 5 0
```

Using the same mask word as in example 1, this illustrates the printing effect when there is no significant fraction value, the printed result being only a whole number. Also, as in example 1, digit positions 7, 8, & 9 are suppressed for lack of significance. In both examples, digit positions 10 through 14 are ignored due to the pointer having been specified at position 9. (Review 3.1.01b, Position Register.)

In the following description of the mask codes, the word "escape" should be interpreted as meaning to space horizontally (that is, to leave one print position without actually printing). The print head does not always move immediately on such an escape. In some cases, successive escapes are added to the position calculation (on automatic function of printing) preceding the print cycle. This enables positioning to the exact print position of the most significant digit in a word to be printed.

CODE	NAME	PRINTING RESULTS
D	Digit	Accumulator Digit prints unconditionally.
.D	Decimal Point & Digit	Print Decimal Point (or escape if suppress punctuation flag is set); Print Accumulator Digit unconditionally.
D:	Digit & Decimal Point	Prints Accumulator digit unconditionally; Prints decimal point (or escapes if suppress punctuation flag is set).
D,	Digit & Comma	Print Accumulator Digit unconditionally; Print Comma (or escape if suppress punctuation flag is set).
Z	Leading Zero Suppress	Print Accumulator Digit if it is significant, or if a preceding accumulator digit has established significance; Escape if accumulator digit is not significant, and if preceding accumulator digits have not established significance.
Z:	Leading Zero Suppress and Decimal Point	Prints Accumulator digit followed by a decimal point (or escape for decimal point if suppress punctuation flag is set) if the digit is significant, or if a preceding Accumulator digit established significance. Escape two positions if Accumulator digit is not significant, and if preceding Accumulator digits have not establishes significance.
Z,	Leading Zero Suppress & Comma	Print accumulator digit followed by a comma (or escape for comma if suppress punctuation flag is set) if the digit is significant, or if a preceding accumulator digit established significance; Escape two positions if accumulator digit is not significant, and if preceding accumulator digits have not established significance.
C	Units of Cents	Print Accumulator digit if it is significant, if a preceding accumulator digit has established significance, or if there is a significant digit in this terminal zero suppression field; Ignore if accumulator digit is not significant, and if significance is not established by either a preceding digit or a digit in this terminal zero suppression field, printer does not escape.
.C	Tens of Cents	Print Decimal Point (or escape if suppress punctuation flag is set) followed by the accumulator digit if it is significant, or if significance is established by either a preceding digit or a digit in this terminal zero suppression field; Ignore if accumulator digit is not significant, and if significance is not established by either a preceding digit or a digit in this terminal zero suppression right field.
X	Terminal Zero Suppress	Print accumulator digit if it is significant, or if any succeeding digit in this terminal zero suppression field is significant; Ignore if accumulator digit is not significant nor any succeeding digit in this terminal zero suppression field is significant.

.X	Decimal Point and Terminal Zero Suppress	Print Decimal Point (or escape if suppress punctuation flag is set) followed by accumulator digit if it or any succeeding digits in this terminal zero suppression field are significant; Ignore if the accumulator digit and all succeeding digits in this terminal zero suppression field are not significant.
I	Ignore Digit	Accumulator digit is ignored; printer does not escape.
E	Ignore Digit End	Accumulator digit is ignored and the print instruction is terminated; printer does not escape.
S	Single Digit Zero Suppress	Print accumulator digit if it is significant; Escape if it is not significant; Preceding and succeeding significance has no effect.

NOTE: If non-significant digits are suppressed by a "C" ".C" "X" or ".X" mask code, the printer does not move and the print position register is not changed. This permits "print character" instructions to print justified left to the last number printed, if desired.

Mask codes need not fill up an entire mask word as they are ignored to the left of the pointer digit position. For example, if the maximum number of digits that will ever be printed with a particular mask is 9, then only 9 mask codes are required in the least significant positions of the mask word.

3.4.08 Mask Flag Codes

In addition to the "Mask Control Codes", a mask word contains three flags which are located in digit position 15, and which modify the mask function. These are the Safeguard Flag, the Suppress Punctuation Flag, and the Punch Leading Zeros Flag. These flags are "set" by entering their appropriate code in the mask word preceding the first (MSD) Mask Control Code. Any combination of the three flag codes may be entered.

CODE	NAME	
F	Safeguard	When the Safeguard flag is set, the safeguard symbol ($) is printed to the left of the most significant digit printed. See paragraph 3.4.09, examples c and d.
+	Suppress Punctuation	When the Suppress Punctuation flag is set, print positions, where commas or decimal points would normally be inserted, are replaced by spaces (the printer escapes). See paragraph 3.4.09, examples e and j.
P	Punch Leading Zeros	The Punch Leading Zeros flag has no effect on printing, but when set, it causes preceding zeros to punch even though they may not print, starting at the pointer.

TABLE OF MASK CODES

PRINT FUNCTION	MASK CODE	FIELD ON WHICH SIGNIFICANCE BASED	PRINTED RESULT: (examples) "9" = digit; "0" = zero "sp" = 1 space (escapement) IF SIGNIFICANT	IF NOT SIGNIFICANT
Digit	D	UNCONDITIONAL	9	0
Decimal Point & Digit	.D	UNCONDITIONAL	.9	.0
Digit & Comma	D,	UNCONDITIONAL	9,	0,
Digit & Decimal Point	D:	UNCONDITIONAL	9.	0.
Leading Zero Suppress	Z	LEADING	9	sp
Leading Zero Suppress & Comma	Z,	LEADING	9,	sp sp
Leading Zero Suppress & Decimal Point	Z:	LEADING	9.	sp sp
Units of Cents	C	LEADING & TERMINAL	9	(no sp)
Tens of Cents	.C	LEADING & TERMINAL	.9	(no sp)
Terminal Zero Suppress	X	TERMINAL	9	(no sp)
Dec. Pt. & Terminal Zero Sup.	.X	TERMINAL	.9	(no sp)
Ignore Digit	I	IGNORE	(no sp)	(no sp)
Ignore Digit & End	E	IGNORE	(no sp)	(no sp)
Single Digit Zero Suppression	S	DIGIT	9	sp

MASK FLAGS	FLAG CODE	FLAG AFFECT ON PRINTING
Safeguard	F	$ Prints to left of most significant digit
Suppress Punctuation	+	Spaces replace Commas & Decimal Points
Punch Leading Zeros	P	Punch leading zeros in punched field

Fig. 3-4 Mask Codes and Flags

3 .4.09 Examples of Numeric Print Masks and the resulting printing of data. Note that ribbon color and sign symbols are controlled by the Print instructions based on the Accumulator sign flag, rather than by the print mask selected:

a. Monetary Punctuation, With Clear Signal:

```
                                             Mask # 0:  Z,ZZZ,ZZZ,ZZZ,ZZZ.DD

Instructions:
  POS   60      )   Contents of ( 000001250046550 =          12,500,465.50
  PNS-  11   0  >     Accum:    <-000000000001500 =  (red)         15.00-
  PC-   --      )               ( 000000000000000 =                  .00
```

b. Monetary Punctuation, Without Clear Signal:

```
                                             Mask # 1:  Z,ZZZ,ZZZ,ZZZ,ZZZ .CC

Instructions:
  POS   60      )   Contents of ( 000000000000500 =                   5.00
  PNS-  11   1  >     Accum:    < 000000000000005 =                    .05
  PC-   -       )               ( 000000000000000 =  (no print)
```

c. Amount Protection, With Punctuation:

```
                                             Mask # 2: FZ,ZZZ,ZZZ,ZZZ,ZZZ.DD

Instructions:
  POS   60      )   Contents of ( 000000000125000 =              $1,250.00*
  PNS-  10   2  >     Accum:    < 000000000052500 =                $525.00*
  PC-   CR      )               ( 000000000000005 =                  $.05*
  PC+   *       )
```

Note that the Safeguard (F) flag is set in the mask word.

d. Amount Protection, Special Format:

```
                                             Mask # 3: FZZ,ZZZ,ZZDED
                                             Mask # 4:             DD
Contents of Accumulator:
     000000000152575

Instructions:
  POS  60
  PN    9    3-----$1,525            )
  RR    .     --------- AND    CTS   )                     (red)
  PA  100                            )                     /   \
  POS  70                            >          $1,525AND 75CTS
  PA  102     -------  ___    ___    )
  POS  73                            )
  PN    1    4------        75       )

 Contents of Accumulator:                   (red)
                                             |  |
      000001050023575 =     $10,500,235AND 75CTS

      000000000000005 =             $0AND 05CTS
```

With several printing instructions and stored alpha messages, nearly any type of amount protection can be programed. Above, the $ and whole dollars print first, ignoring the cents digits; then the alpha messages "AND CTS" and "__ __" are printed, followed by the cents digits. Changing the ribbon color for alpha printing provides visual separation of the print.

e. Monetary Without Punctuation: Clear Signal:

```
                                          Mask # 5: +ZZZ,ZZZ,ZZZ.DD

Instructions:
  POS   60          Contents of ( 0000012500465550 =  12 500 465 50
  PNS-  10             Accum:  <  0000000000001500 = (red)    15 00-
  PC-    -     5                ( 0000000000000000 =              00
```

Note that the Suppress Punctuation (+) flag is set in the mask word.

f. Numeric Punctuation, With Clear Signal:

```
                                          Mask # 6: Z,ZZZ,ZZZ,ZZZ,ZZD

Instructions:
  POS  60           Contents of ( 0000000002345678 =         2,345,678
  PN    9   6          Accum:   ( 0000000000000000 =                 0
```

g. Numeric Punctuation Without Clear Signal:

```
                                          Mask # 7:  Z,ZZZ,ZZZ,ZZZ,ZZZ

Instructions:
  POS   60          Contents of ( 0000500003345678 =           345,678
  PN     9    7        Accum:   ( 5000000000000000 =  (no print)
```

Note that significant digits to the left of the pointer do not affect the suppression of preceding zeros.

h. Decimal Fractions:

```
                                          Mask # 8:      Z,ZZZ,ZZD.XXX

Instructions:
  POS   60          Contents of ( 0000000000026125 =          26.125
  PN     9      8      Accum:   ) 0000000000030000 =          30
                                ) 0000000000005001 =           5.001
                                ( 0000000000150200 =         150.2
```

i. Percentages:

```
                                          Mask # 9:      Z,ZZZ,ZZD.X

Instructions:
  POS   60          Contents of ( 0000000000000125 =          12.5%
  PN     7    9        Accum:  <  0000000000000150 =          15%
  PC     %                      ( 0000000000000000 =           0%
```

j. Special Formatting: Social Security Number:

```
                                          Mask # 10: +DDD,DD,DDDD

Instructions:
  POS  60           Contents of ( 0000000123456789 =  123 45 6789
  PN    8    10        Accum:   ( 0000000045030612 =  045 03 0612
```

Note that the use of punctuation in the mask word can occur in any pattern to separate the digits into required groupings; the use of the "Suppress Punctuation Flag" (+) in the mask then serves to put in a space between these groups of digits.

k. Split Word; 2 Monetary Amounts:

```
                                        Mask # 11:  ZZ,ZZZ.DDEZZ,ZZZ.DD

Instructions:
  POS   60          Contents of ⎧ 000125000007500 =          12.50    75.00
  PNS-    14  11       Accum:   ⎨ 000000000000000 =            .00      .00
  PC-     -                     ⎩-000052500003000 = (red)    5.25-   30.00-
  POS   70
  PNS-     6  11
  PC-     -
```

Note that the printing of two separate amounts from one word requires two separate printing instructions to enable suppression of preceding zeros for each amount. One mask word can be used to print both amounts: When printing the left half, the pointer starts with digit position 14, and printing ends after position 8 by the "E" mask code in position 7; the second print instruction prints the right half with the pointer starting in position 6 and ending with position 0. Position 7 cannot be printed by this method. Minus amounts can be printed with sign so long as both amounts always have the same sign value, since a word has only one sign position.

3.5 SINGLE CHARACTER PRINT

Instructions are provided to print selectively any of the 64 print characters or to space the printer one position.

3.5.01 Unconditional Print Character Instructions:

	Op Code	A	B
Print Character	PC	(actual char.)	
Print Character Previous Ribbon	PCP	(actual char.)	

These instructions unconditionally print the character specified in the "A" field (refer to Figure 3-1). If the "A" field is blank, the instruction causes a single printer space (escapement) operation.

The PC instruction prints with the ribbon in the black position unless preceded by an RR instruction.

The PCP instruction prints with the same ribbon shift as that used on the last print operation, but unconditionally opposite to the last ribbon color if preceded by an RR instruction.

3.5.02 Conditional Print Character Instructions

	Op Code	A	B
Print Character if Accumulator Minus, Previous Ribbon	PC-	(actual char.)	
Print Character if Accumulator Plus, Previous Ribbon	PC+	(actual char.)	

Printing by these instructions is dependent upon the status of the accumulator sign flag. This flag is set or reset by a Numeric Keyboard Instruction (Section 2), by the Accumulator Flag Instruction (Section 6), or by arithmetic operations (Section 5).

The character specified in the "A" field (refer to Figure 3-1) is printed for the following conditions:

PC- Print if accumulator negative, sign flag set; do not print if plus.

PC+ Print if accumulator positive, sign flag reset; do not print if negative.

When the accumulator sign does not agree with the condition specified, the character is not printed, the printer does not escape nor does the position register change.

The ribbon color is determined by the ribbon color of the previous print instruction.

To assure the printing of one character or another in cases where either is desired (such as the printing of an "*" character for a plus total or subtotal or a "CR" character for a minus total or subtotal), one "print character" instruction should follow the other:

Example:

	A	B
PNS-	5	2
PC+	*	
PC-	CR	

If the accumulator contained a plus 525.00 amount, it would print as:

525.00*

If the accumulator contained a minus 525.00 amount, it would print as:

525.00CR (red)

Note: "CR" is a single character and occupies 1/10" printing space.

It is important to note that the character not desired would not print, but that one or the other would <u>always</u> print.

If the "A" field is blank, the instruction will cause a single printer space (escapement), provided the accumulator sign flag agrees with the condition specified (plus or minus); otherwise, it will not space.

It has been stated that a Red Ribbon instruction can be used to change the ribbon color of the next printing instruction. After executing this print (or type) instruction (even though printing may not actually occur), the effect of the Red Ribbon instruction is removed. If the instruction PC- (or PC+) does not result in actually printing a character, the effect of the Red Ribbon is still removed (a PC+, PC- or PCP following this, which does print, prints with the previous ribbon).

The following example to print a symbol opposite the color of the amount is incorrect since the PC- instruction is not affected by the RR instruction:

	A	B
PNS-	5	2
RR		
PC+	*	
PC-	CR	

If the amount in the accumulator was a plus 525.00, it would print as:

525.00* (red)

But if the amount was a minus 525.00, it would print as:

525.00CR (red)

because the instruction for the non-printed character (*) removes the effect of the Red Ribbon instruction.

To assure either character would print and in the opposite color to the color of the amount, the following programing should be followed. Note that "RR" precedes each Print Character instruction:

	A	B
PNS-	5	2
RR		
PC+	*	
RR		
PC-	CR	

If the amount in the accumulator was a plus 525.00, it would print as 525.00* (red)

If the amount was a minus 525.00, it would print as: 525.00CR (red)

3.6 PRINT PROGRAMING CONSIDERATIONS

Print instructions should normally follow all Numeric Keyboard entries if printing is desired, since Numeric Keyboard instructions do not print and since the accumulator data may be changed by subsequent instructions.

Typing Keyboard entries, by nature, print (except EAM), thus do not require a print instruction.

Prior to printing, the Position Register must be loaded. The exact time when positioning takes place and the exact location where the printer positions is determined by the print or keyboard instruction following it (see 3.1.01).

RR affects only the Print instruction following it.

The Print Alpha (PA) instruction will not Backspace, Open or Close the Transport, nor Advance the platen, as these codes are not stored in memory. Print Alpha can be in red is preceded by an RR instruction.

Print Numeric Instructions will normally be used as follows:

PN allows printing the contents of the Accumulator. No conditional color change is possible. Reference Numbers or Numeric Codes should be programed with this instruction.

PNS - allows printing the contents of the Accumulator with automatic ribbon color determined by the sign of the Accumulator. Accounting entries (Dollars and cents), Units, Quantities, etc., should be programed with this instruction.

Print Character instructions are normally used as follows:

PC allows an unconditional print of any character. It will be in black unless preceded by an RR instruction. It would normally be used to print Reference Codes such as ^{o} (degrees), % (percentages) or any time a single character with no ribbon change conditions is needed.

PCP would be used to print any character in the same color as the last printed data.

PC+ and PC- will print (in the color corresponding to the previous printer data) or non-print according to the sign of the Accumulator. An illustration of their use is to describe a plus total or subtotal as "*" or a minus total or subtotal as "CR". It can be used any time conditional printing (based on sign) is desired.

4. FORMS CONTROL

The Forms Control instructions provide the ability to open and close the forms transport and to advance the forms a specified number of lines, or to a specified line including automatic alignment of a rear-fed document to the first line. Advancing a form to a specific line is controlled by a Forms Count Register and a Forms Limit Register within the computer, which operate according to the stored program. Program "Channel Tapes" are not required for the vertical spacing mechanism of either the platen or continuous form pin feed device.

To accommodate rear-fed forms automatically, the Open instruction may designate the number of lines to advance the form when the transport is closed for the first time after execution of the Open instruction.

Forms may be advanced while wrapped around the platen or by way of a single or dual pin feed device. The dual pin feed device contains an upper pin feed shaft for one set of continuous forms and is controlled by any forms instruction specifying the "right" advance control. The lower pin feed shaft for the second set of continuous forms is controlled by any forms instruction specifying the "left" advance control.

When an instruction is used to advance forms, it may advance the platen and/or a pin feed shaft of the pin feed device (if the device is attached to the console.) Whether the platen and/or the pin feed shaft are advance is dependent on the type of platen (split or solid), the style of pin feed device, and the particular instruction.

Refer to Part I, Sections 8 and 9 for a complete discussion of spacing control with each of the various combinations of split or solit platen and single or dual pin feed device considerations.

When using rear-fed documents around the platen, it is necessary to engage the platen lower pressure rollers (rearward position of the Alignment Protector Lever) to insure positive control over the forms.

When using the Pin Feed Device with continuous forms, the platen lower pressure rollers must be disengaged (forward position of the Alignment Protector Lever) to allow positive forms control.

When using the Pin Feed Device, rear-fed unit documents cannot be accommodated. If 2 independent forms are to be controlled, the Dual Pin Feed Device must be used. Roll journals are acceptable.

The platen may be split, in which case the left platen is controlled by "left" forms handling instructions, and the right platen by "right" instructions. It is important to note that the splits do not need to correspond to the placement of continuous forms if using the pin feed device.

4.1 FORMS TRANSPORT OPEN AND CLOSE

4.1.01 Open Transport Instruction

	Op Code	A	B
Open Forms Transport	OC	0:255	

The OC instruction is used to openthe forms transport mechanism in order to permit the removal of a completed unit document and to insert (rear feed) a new document. It performs one additional function. The "A" parameter field of the OC instruction specifies the number of lines the LEFT FORMS mechanism is to advance when the transport mechanism is next closed, (refer to Par. 4.4.03). This closing may be from any of the following sources:

1. The execution of a PN or PA instruction of any type

2. The entering of alpha information at a TK instruction.

   (NOTE: **If a TK instruction were terminated by an OCK without the entering of alpha data, the transport mechanism would not close).**

3. A CC (Close) instruction.

4. Manual depression of the Open/Close key on the keyboard.

The parameter field of the OC instruction is stored in a special register until the transport mechanism is closed. Upon executing the OC instruction, the program continues. Thus, the OC instruction is normally followed closely by a keyboard entry instruction to permit removal and insertion of forms.

4.1.02 Close Transport Instruction

	Op Code	A	B
Close Forms Transport	CC		

The CC instruction closes the forms transport. This instruction usually is not required since execution of any Print instruction or depression of a typing key during a Type instruction automatically closes the forms transport. (refer to par. 4.4.03)

Note: If the transport is open as a result of executing an OC, when the Close Instruction is executed, the left forms advance mechanism will advance the number of lines specified by the Open Instruction.

4.2 PLATEN CONTROL REGISTERS

Four registers are provided for control of vertical spacing of the left and right platens; these are the left and right Forms Count Registers, and the left and right Forms Limit Registers. In addition, there is a Forms Limit Flag (one of the Test Flags, par. 6.2).

A Forms Count Register is associated with each platen advance mechanism, and is automatically incremented by 1 each time that its platen is advanced one line. A Forms Limit Register is also associated with each platen advance mechanism, and provides a limit to which the Count Register can be compared.

The Forms Limit Registers are preset (by instruction) to a specified number of 1/6" spaces. On the line advance following when a forms count register equals its related forms limit register, the Forms Limit Flag is set; on every other line advance and resultant counting of the Register, the Forms Limit Flag is reset. Thus, the Forms Limit Flag provides a signal as to when a Count Register has exceeded its Limit Register, and from it the program can select an alternate course of action. This is effective on single line advances only, since on multiple line advances the flag would usually be reset before it could be tested. A Forms Count Register is reset to "1" on the next line advance occurring after it equals its Limit Register.

In controlling rear-fed unit documents, the Limit Register is normally loaded with the last line number on which printing is to occur; that is, the last usable line on the form, but not the last 1/6" increment line on the form. The Count Register is loaded with the first printing line number. After each single line advance, the program interrogates the status of the Forms Limit Flag to determine if the maximum allowable lines have been used. If the flag is set (Count Register reset to 1), it indicates having used the last line and permits the program to go to a subroutine (see section 9: Skip and Execute instructions). The subroutine might print out summary information on the bottom of the form, open the forms transport to allow the insertion of a continuation form, perform any desired function on the continuation form such as printing a partial heading, and then space down to the body of the form to continue with additional entries. If the flag is not set, the last line has not been used and the program may continue with entries.

In controlling pin-fed continuous forms there are normally two separate line counts needed: the number of lines on the overall form, and the number of posting lines in the body of the form:

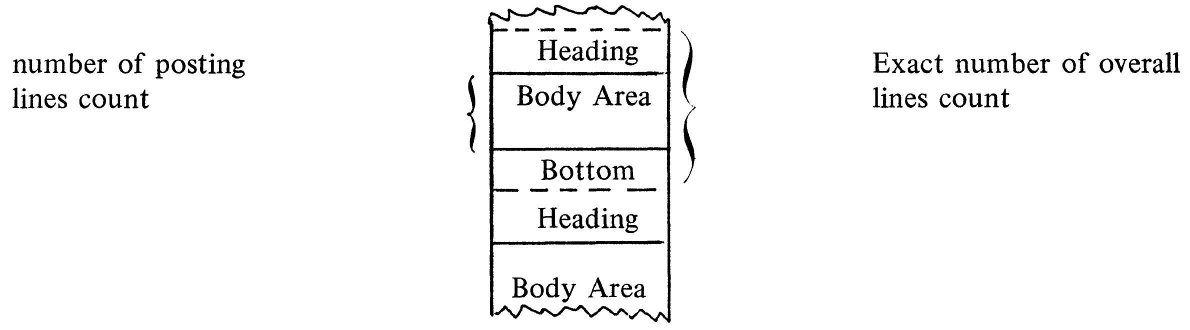

number of posting lines count
Heading
Body Area
Bottom
Heading
Body Area
Exact number of overall lines count

Fig. 4-1 Continuous Pin Fed forms showing line counts needed

A knowledge of the number of lines in the overall form is needed to control the advancement from one form to the next form automatically.

The number of lines within the body of the form (the posting lines) must also be counted because a form may not contain enough lines for the maximum number of potential entries, and because it may be desired to allow a bottom margin on the form.

When only one continuous form is used in the pin feed device at one time, both Count Registers may be used to satisfy the need for two line counts.

For example, assume that the continuous forms are mounted on the lower pin feed shaft, associated with the left spacing mechanism.

The Left Limit Register would be loaded with the exact number of 1/6" increment lines that are on the over-all form. In the case of an 11" form, this would be 66 lines. The Left Count Register is loaded with the line number where printing starts. When one form is completed, an instruction is used which causes the forms to advance to a specific line number (first print line) on the next form. The Forms Limit Flag is not tested in this situation. Since the Left Count Register resets to "1" on the line after it equals the Left Limit Register, it contains "1" on the first line of the next form and spacing continues until the specified line is reached.

Since the Left Count Register is being used to keep track of the overall form lines, the Right Count Register and Right Limit Register are used for the usable line count. In this case, the Forms Limit Flag is tested after each single line advance, to determine when the last allowable line has been used, just as in the previous discussion of rear fed forms. Each single line advance instruction must specify both left and right spacing mechanism to keep both count registers together. However, when advancing from one form to the beginning print line on the next, only the left spacing mechanism is used; This requires that the Right Count Register be reset back to the starting line number as each new form is started. The roles of the Left and Right registers, spacing mechanism, and tractor pin feed shafts may be reversed, providing the same results.

4.2.01 Platen Count and Limit Registers Instructions:

	Op Code	A	B
Load Left Platen Count Register	LLCR	0:255	
Load Right Platen Count Register	LRCR	0:255	
Load Left Platen Limit Register	LLLR	0:255	
Load Right Platen Limit Register	LRLR	0:255	

The LLLR and LRLR instructions load the left and right Platen Limit Registers respectively with the contents of the "A" field.

Once a Limit Register (Left or Right) is loaded with a number (0-255) it can be changed only by another Load Limit Register instruction.

The LLCR and LRCR instructions load the left and right Platen Count Registers respectively with the contents of the "A" field.

The Count Registers (left and right) are loaded with a number (0-255), and their contents change. Changes may be caused by the following:

1. Forms advance instructions (AL, AR, ALR, ALTO, ARTO) or the depression of the Line Advance key on the keyboard. The count register is incremented by 1 for every line advance from any of these sources. It is not incremented when the platen twirlers are manually rotated.
2. The Count Register will be set to 1 on the next line advance after the count register equals its limit register.
3. The execution of another LLCR or LRCR instruction.

On the line advance following when the Count Register contents are equal to its corresponding Limit Register, the Forms Limit Flag is set. This flag can be interrogated using a Skip or Execute type of instruction (see section 9).

4.3 LINE ADVANCE INSTRUCTIONS

NOTE: A thorough review of the description of the various forms transport devices in Part I will be required to use Line Advance instructions properly.

Instructions are provided to advance a form a specified number of lines (without regard to any particular line on the form) and to advance a form to a specified line. Forms advance up to 20 lines per second (80 ms first line; 50 ms each additional line).

4.3.01 Advance Platen Instructions

	Op Code	A	B
Advance Left Platen	AL	0:255	
Advance Right Platen	AR	0:255	
Advance Left & Right Platens	ALR	0:255	
Advance Left Platen To	ALTO	1:255	
Advance Right Platen To	ARTO	1:255	

The AL, AR and ALR instructions advance a form the number of lines specified by the "A" field. These provide a single line advance, and can also advance up to 255 lines.

The ALTO and ARTO instructions advance a form until the associated count register is equal to the value of the "A" field. If the count register equals the line number specified in the ALTO or ARTO instruction prior to its execution, no advance occurs.

Note: An attempt to specify "0" or a number larger than the contents of the limit register in the "A" field of ALTO/ARTO is a programing error which will result in a continuous search for a line number that does not exist.

4.4 USE OF FORMS CONTROL INSTRUCTIONS

4.4.01 Alto - Arto

When an ALTO or ARTO (left or right) instruction is being executed, the form will be advanced 1 vertical space at a time. With each space, the corresponding Count Register is incremented by 1 and the Count Register is compared to the parameter field of the ALTO or ARTO instruction. Form spacing continues until the two are equal. When they are equal, the spacing will stop. For the following example, assume the Limit Register does not exist.

Example # 1:

Op Code	A
LLCR	4
⌇	
ALTO	12

The form will be advanced 8 lines at which time the Count Register will be equal to the Alto parameter field and the forms advance will stop. The Count Register will contain 12.

The previous example has not taken the Left Limit Register (LLLR) into consideration. However, the execution of the ALTO instruction does involve the LLLR instruction. Each time the Count Register is incremented by 1, it is compared to the corresponding Limit Register. On the next line advance after the Count Register and Limit Registers are equal, the Count Register is set to 1.

NOTE: The Count Register is set to 1 and not to 0.

Example # 2:

```
Op Code    A
LLLR      255
LLCR       19
  ⌠
ALTO        3
```

The form will be advanced 239 lines. On each line advance, the Count Register is incremented by 1 and is compared to the ALTO parameter field. Since the Count Register is already greater than the ALTO field, incrementation and spacing will continue until the Count Register reaches 255. At this point the Count Register and Limit Register are equal so on the next line advance, the Count Register will be set to 1 (not 0) and will continue advancing until the Count Register contains 3. Forms advance would now stop because the Count Register is equal to the ALTO parameter field.

Example #3:

```
Op Code    A
LLLR       30
  ⌠
ALTO        3
```

This is an example of the type of programing employed when using pin fed continuous forms with the requirement that the program automatically advance from the last line used on one form to the first line (Line 3) of a new form. Assume contents of Left Count Register = 19. The form will advanced 14 lines. The Count Register will be incremented by 1 on each advance. When the Count Register reaches 30 (after 11 advances) it is equal to the Limit Register so the next line advance will set the Count Register to 1. However, advancing will continue until the Count Register equals the ALTO parameter field which is 3.

The use of the commands ALTO and ARTO must always take into consideration the value in both the Count Register and the Limit Register for either the left or right platen spacing mechanism. The Limit Register, by storing a value with which to compare the Count Register, enables recognizing when the last print line is reached, and will normally be loaded with the number of lines contained on a form from the top edge to the bottom edge. For example, for an 8½" by 11" form, this would be 66 lines.

When the program begins, the Count Register will usually contain some value from a previous operation on the system. Therefore, it must always be loaded with the desired value or reset to zero at least once in the program before attempting to use ALTO/ARTO). Indeterminable spacing would otherwise result.

4.4.02 AL - AR

The AL or AR instruction will advance the form the exact number of lines specified by the parameter field. The most common use of AL or AR is to advance a form one line; however, the maximum is 255. The Count Register is incremented by 1 on each line advance. The next line advance after the Count and Limit Registers are equal, sets the Forms Limit flag and sets the Count Register to 1. Advancing will continue until the number of lines called for has been accomplished. No comparison is made between the AL parameter field and the Count Register since it is not looking for a specific line number.

Example #4:

Op Code	A
LLLR	42
LLCR	26
∫	
AL	20

The form will be advanced 20 lines. The Count Register will be incremented from 26 to 42. The next line advance will set it to 1 (not 0). Spacing will continue until the full 20 lines have been accomplished. When spacing stops, the Count Register will contain 4.

4.4.03 OC - CC and Rear-Fed Unit Documents

When unit documents are used, the OC instruction is required for the rear feed forms handling mechanism to automatically align the form. The vertical spacing designated with an OC instruction is held up until the transport actually closes, even though subsequent instructions continue to be executed. Aside from using a CC instruction, the transport will not close until an instruction is reached where printing actually occurs (Keyboard or Print instruction). When printing is ready to begin, the transport closes and vertical spacing of the left platen spacing mechanism occurs, incrementing the left Count Register at that point. Thus, any reloading of the Count Register after the opening and before the closing of the transport may not be effective since it will be incremented when the transport closes.

The instruction CC will generally be programed AFTER a Keyboard or Print instruction that follows a transport opening. If it is used after opening and before a keyboard instruction, the operator would usually not have time to remove the last form and insert the new form before the closing occurred. Since a TK instruction, which is terminated with an OCK (or PK) without an entry, does not print anything nor close the carriage, the CC instruction may be programed after the keyboard instruction to provide for this event.

Example # 5:

Op Code	A
LLCR	9
OC	14
∫	
TK	24
CC	

When typing begins, the transport will be closed and the rear-fed unit document will advance 14 vertical spaces. The Count Register will be increased by 14 and contain 23. The above example ignores the Limit Register.

Example #6:

Op Code	A	B
LLLR	30	
LLCR	21	
OC	14	
⌇		
PN	6	0

The transport mechanism will close and the rear-fed unit document will advance 14 lines. The Count Register is incremented 9 times until it reaches 30. On the next line advance, the Count Register is set to 1. Advancing will continue until the form advances 14 lines. At that time, the Count Register contains 5.

The Forms Transport is equipped with a Form Guide Bail/Form Heading Holder. When the form is advanced the number of lines specified by the OC instruction, this bail will close to hold the form against the platen for printing. The OC parameter field must contain at least a value of 10 in order for the top of the form to be under this bail when it closes. The actual parameter field of the OC instruction may be 0 if it is to be modified by an Index Register instruction. In this case, the Index Register should contain at least a 10.

4.4.04 Alignment To The First Print Line

When programing for automatic alignment of rear-fed unit documents, the number that must be placed in the OC parameter field must be 3 greater than the line number of the first actual line of print.

Example #7:

Assume a unit document that must be aligned to line number 14 as measured in 1/6" increments from the top of the form. This then is actually line number 14 on the form.

The OC instruction must contain a 17 in the parameter field.

When the unit document is placed at the back of the forms transport mechanism, with the mechanism open, it rests against a limit bail. This limit is roughly the equivalent of 3 vertical line spaces below the actual print line of the machine. (i.e., the top of the form is underneath the platen and cannot be seen). Therefore, to align to the 14th actual 1/6 increment line from the top of the form, the OC parameter field must contain a 17 and not a 14. If OC is programed with a 14 instead of 17, when the forms transport is closed and the form aligned, it would actually be aligned to the 11th line from the top of the form.

Note that in the above example, even though the form is aligned to the 14th print line, the Left Count Register was incremented 17 times. Thus, it may be desirable to reload the Count Register with 14 before any further vertical spacing is performed, especially if absolute control must be maintained for "Align To" commands. This would permit the instructions to specify line numbers consistent with the actual form measurements to the desired printing areas.

Example #8:

Use of the Limit Register to enable the program to know when 40 lines have been filled on the invoice: The total length of the invoice is 8½ inches (51 lines). The first print line is line 14 as measured from the top of the form.

Op Code	A
LLLR	40
LLCR	37
OC	17
TK	10
CC	

When the Forms Transport is closed, the form will advance 17 lines. The first three lines increment the Count Register to 40, the next advance will set the Count Register to 1. After an advance of the remaining 13 lines the Count Register will be at 14. This is the actual first line number, and the number wanted in the Count Register.

4.5 FORMS CONTROL PROGRAMING CONSIDERATIONS

The following may be considered as General Rules governing the Forms Handling instructions. Violations are considered as programing errors.

1. The ALTO/ARTO parameter fields should never be 0.

   Example:

Op Code	A
ALTO	0

   The form will be advanced indefinitely, looking for a line that does not exist.

2. The ALTO/ARTO parameter field should never be a number greater than the LLLR/LRLR parameter field.

   Example:

Op Code	A
LLLR	40
ALTO	50

   The form will be advanced indefinitely - looking for a line that does not exist because the Count Register resets to "1" after reaching "40" and thus will never find "50".

3. The Count Registers (LLCR/LRCR) parameter fields should never be a number greater than the Limit Register parameter.

   Example:

Op Code	A
LLLR	40
LLCR	52
ALTO	8

   The form will be advanced an indeterminite number of lines.

4. The Limit Registers (LLLR/LRLR) should never be set to 0.

	Op Code	A
Example:	LLLR	0
	LLCR	40
	$\Bigg\}$	
	ALTO	7

The form will be advanced an indeterminite number of lines.

5. ARITHMETIC INSTRUCTIONS

The Arithmetic Instructions provide the ability to add, subtract, multiply and divide. A zero result developed by any of these functions will always be positive. Instructions are also furnished to modify a single digit of the Accumulator and to shift the Accumulator right or left up to 16 digits. Information is transferable between the Accumulator and memory.

5.1 ADDITION AND SUBTRACTION

The contents of the accumulator and the contents of memory location A are algebraically added or subtracted per the instruction used. The sign flag of the sum or difference is reset (plus) if the result is positive or set (minus) if negative.

The per thousand (M) and per hundred (C) flags of the result are unconditionally reset.

The overflow flag is set if an overflow occurs and reset if there is no overflow.

5.1.01 Addition Instruction

	Op Code	A	B
Add to Accumulator	ADA	0:N	
Add to Memory	ADM	0:N	

The ADA instruction provides for adding the contents of a memory location, specified by the "A" field, to the contents of the Accumulator. The resultant sum is placed in the Accumulator leaving memory location "A" undisturbed.

The ADM instruction provides for adding the contents of the Accumulator to the contents of the memory location specified in the "A" field. The resultant sum is placed in memory location "A" leaving the Accumulator undisturbed.

These two commands cannot be used to move alpha data, even if the receiving location is clear.

5.1.02 Subtraction Instructions

	Op Code	A	B
Subtract from Accumulator	SUA	0:N	
Subtract from Memory	SUM	0:N	

The SUA instruction provides for subtracting the contents of the memory location specified by the "A" field from the contents of the Accumulator. The difference is placed in the Accumulator leaving memory location "A" undisturbed.

The SUM instruction provides for subtracting the contents of the Accumulator from the contents of the memory location specified by the "A" field. The difference is placed in memory location "A" leaving the Accumulator undisturbed.

5.2 TRANSFER (MOVE) INSTRUCTIONS

Transfer instructions are provided to move data (alpha or numeric) from a memory location to the Accumulator, and from the Accumulator to a memory location. When numeric data is transferred, the sign (-), special (S), per hundred (C) and per thousand (M) flags are also transferred, in accordance with their set/reset condition.

	Op Code	A	B
Transfer to Accumulator	TRA	0:N	
Transfer to Memory	TRM	0:N	

The TRA instruction provides for transferring the contents of a memory location specified by the "A" field to the Accumulator; the contents of the memory location remain unchanged.

The TRM instruction provides for transferring the contents of the Accumulator to the memory location specified by the "A" field; the contents of the Accumulator remain unchanged.

5.3 CLEAR INSTRUCTIONS

	Op Code	A	B
Clear Accumulator and Insert Constant	CLA	0:15	0:15
Clear Memory Word	CLM	0:N	

The CLM instruction clears all 16 digits of the memory location specified in the "A" field. The contents of the Accumulator remain unchanged.

The CLA instruction sets all 16 digits of the Accumulator to zero, thus resetting the four accumulator flags (M, C, special, and sign); it places the digit specified by the "B" field in the digit position of the Accumulator specified by the "A" field.

Although the B parameter field of the CLA instruction permits designating a value of zero to 15 on a program coding sheet, any value above 9 is placed in memory as one hexadecimal digit expressed as a character A through F (A representing ten, B representing eleven, etc.) rather than as two decimal digits. It may be desirable to use values above nine for special situations such as perhaps in punching special output codes, but the use of these values would not be possible in any type of arithmetic. Addition, subtraction, multiplication, etc., could only use values of from zero to nine in any digit position. Any values over nine would be lost during arithmetic.

5.4 INSERT CONSTANT, ADD CONSTANT, SUBTRACT CONSTANT

These instructions provide the ability to manipulate the contents of the Accumulator, providing versatility that could otherwise be obtained only by extensive programing.

5.4.01 Insert Constant Instruction

	Op Code	A	B
Insert Constant in Accumulator	INK	0:15	0:15

The INK instruction places the digit specified by the "B" field in the digit position of the Accumulator specified by the "A" field.

Zero is the least significant digit position; 14 is the most significant position of the Accumulator; 15 is the flags position. The previous value in that digit position is replaced; the remaining digit positions are unaffected. The overflow flag is not changed.

The "B" parameter field in this instruction also permits entering a value of from zero to 15 which it must be understood, is a hexadecimal value of 15, rather than a decimal value of 15. The same comments apply as mentioned under Clear Accumulator and Insert Constant.

5.4.02 Add Constant Instruction

	Op Code	A	B
Add Constant to Accumulator	ADK	0:14	0:9

The ADK instruction provides algebraic addition of the digit contained in the B field to the digit in the Accumulator position specified by the "A" field, with carries propagated in succeeding digits.

The special (S), per thousand (M) and per hundred (C) flags are unconditionally reset.

The sign flag is reset (+) if the result is positive or set (-) if negative.

The overflow flag is set if an overflow occurs and reset if there is no overflow.

5 .4.03 Subtract Constant Instruction

	Op Code	A	B
Subtract Constant from Accumulator	SUK	0:14	0:9

The SUK instruction provides algebraic subtraction of the digit contained in the B field from the digit in the Accumulator position specified by the A field with carries propagated in succeeding digits.

The special (S), per thousand (M), per hundred (C), sign and overflow flags are treated the same as specified for the ADK instruction.

5.5 MULTIPLICATION AND DIVISION

The multiply and divide operations are accomplished by a series of additions and subtractions, respectively, which create a partial product and a partial quotient, respectively. The complete product and quotient are developed by a series of shifts, the number and type of which are designated by a shift register. The contents of the shift register are pre-loaded by the programmer with a value determined by a knowledge of the product or quotient that is to be developed.

5.5.01 Multiply-Divide Shift Register Instruction

	Op Code	A	B
Load Shift Register	LSR	0:15	

The LSR instruction provides for loading the multiply-divide shift register with the contents of the A field. The effect of this shift register is discussed with the Multiply and Divide instructions.

If shifting or scaling is to be accomplished during the multiplication or division instruction, the shift register must be loaded prior to the execution of the Multiply or Divide command. Once the shift register has been loaded, the value loaded will remain until a subsequent Load Shift Register instruction is executed. Therefore, it is important to load this at least once during the program as it may contain a value from some previous program that was operated on in the system. Also when used during multiply and divide, it should be re-loaded for each such calculation if preceding calculations used different shift requirements.

5.5.02 Multiply Instructions

	Op Code	A	B
Multiply	MUL	O:N	
Multiply and Round	MULR	O:N	

The MUL instruction multiplies the Accumulator contents by the contents of the memory location specified by the "A" field. The product is shifted right the number of places specified by the value in the Shift Register, and the digits shifted off are lost. After shifting, the next 15 low order digits are set into the Accumulator as the product.

Both the multiplicand (operand in Accumulator) and the multiplier (operand in specified memory location) may contain up to 15 digits. However, if the product exceeds 15 digits after shifting according to the value of the shift register, the amount exceeding 15 digits is lost and the overflow flag is set; otherwise, the flag is reset. In the event of overflow, an indicator lamp is not turned on. If there is a possibility of this happening, the program must provide for interrogating the flag to determine if a corrective routine must be activated.

If the signs of the operands are alike, the sign of the product is positive (accumulator sign flag is reset +); if the signs are unlike, the sign of the product is negative (accumulator sign flag is set -).

Multiplication takes into account the value of the shift register even though an LSR instruction has not preceded the MUL instruction. The only time LSR must precede a MUL instruction is if the shift requirement changes from one MUL (or Divide) instruction to another in the same program.

Since the number of significant digits in the multiplier determines the time for execution of multiply, the operand with the least number of digits should be used as the multiplier when this is predictable. The number of digits in the multiplicand has no effect on timing.

The MULR instruction is the same as the MUL instruction except that a 5 is added to the last digit shifted off in the product; thus, the product that is set into the Accumulator after multiplication is increased by 1 if the last digit shifted off was greater than or equal to 5. If the shift register value is zero there will be no rounding, thus no difference between the product of MUL and MULR.

5.5.03 Divide Instruction

	Op Code	A	B
Divide	DIV	0:N	

The DIV instruction divides the contents of the Accumulator by the contents of the memory location specified by the "A" field. The quotient is placed in the Accumulator. The contents of the shift register determines the number of decimal digits developed in the quotient. After division has been carried out for the number of decimal places specified in the shift register, any remainder is placed in working memory (in the Control area) (see 5.5.05).

Both the Dividend and the Divisor may contain up to 15 digits. If the signs of the operands are alike, the sign of the quotient is positive (accumulator sign flag is reset +); if the signs are unlike, the sign of the quotient is negative (accumulator sign flag is set -). The remainder is always positive.

	Op Code	A
Example:	LSR	5
	DIV	200

Accumulator (dividend)	=	100	
Memory location 200 (divisor)	=	3	
Multiply-Divide Shift Register	=	5	
Accumulator (quotient)	=	3333333 =	printed with decimal = 33.33333 printing of decimal provided by print mask.
Remainder	=	1	

The division process treats the contents of the Accumulator and the specified memory location as whole numbers, even though they may have "assumed" decimal points; for example: $6_\wedge25 \div 5_\wedge00$ produces a quotient of 1 and a remainder of 125 if the shift register has a zero value:

Accumulator (dividend)	=	625	
Memory location 200 (divisor)	=	500	
Shift register	=	0	
Accumulator (quotient)	=	1 =	could be printed as "1" or "1.". Since it is in first digit position, any other decimal places shown in printing would require shifting it left such as to permit "1.0000"
Remainder	=	125	

Thus, since division halts once the dividend can no longer be divided, the shift register must contain a value equal to the number of decimal places desired beyond what the "whole numbers" themselves would provide. In the above example, by giving the shift register a value of 4, the quotient reflects the "assumed" decimal values:

Accumulator (dividend)	=	625	
Memory location 200 (divisor)	=	500	
Shift register	=	4	
Accumulator (quotient)	=	12500	(printed with decimal = 1.2500)
Remainder	=	0	

The value to be loaded into the shift register can be determined in the following manner with a knowledge of the "assumed" decimal places needed in the quotient as well as the dividend and divisor:

	{assumed decimal places in DIVISOR}	PLUS	{assumed decimal places in QUOTIENT}	LESS	{assumed decimal places in DIVIDEND}	=	Value of SHIFT REGISTER
Ex:	$5_\wedge00$		$1_\wedge2500$		$6_\wedge25$		
	2	+	4		2	=	4

5.5.04 Quotient Overflow

If the quotient after final shift exceeds 15 digits, the overflow flag is set; otherwise the flag is reset. The size of the quotient can be estimated and a prediction of possible overflow made if the following rule is used:

"Add the MAXIMUM size DIVIDEND to the Value of the SHIFT REGISTER plus 1,subtract the MINIMUM size DIVISOR and that equals the MAXIMUM size Quotient possible."

The rule is in terms of the number of significant digits expected in each operand including intervening and terminal zeros, and without regard to "assumed" decimal places.

Example:

	Maximum size DIVIDEND	+ 1 +	Value of SHIFT REG.	–	Minimum size DIVISOR	=	Maximum size QUOTIENT
Ex:	(9999) 4	+ 1 +	(2) 2	–	(1) 1	=	(999900) 6
Ex:	(9999) 4	+ 1 +	(3) 3	–	(100) 3	=	(99990) 5

When an overflow occurs, the division is halted and the result in the Accumulator is meaningless (reflects some stage of partial quotient development).

An attempt to divide a number by zero sets the overflow flag and produces an undeterminable answer. Dividing zero by any number produces a quotient of zero.

5.5.05 Transfer Remainder to Accumulator Instruction

	Op Code	A	B
Transfer Remainder to Accumulator	REM		

The REM instruction transfers the remainder of a division operation to the accumulator. The remainder is stored in the control area of memory by the division process. It is always positive. Its transfer into the accumulator resets all accumulator flags.

5.6 SHIFT ACCUMULATOR

The Shift Accumulator instructions are designed to give digit and character accessibility to a system that is basically word oriented. They make it possible to extract a digit(s) or character(s) from the Accumulator, and consequently from memory. This allows single digit, or alphanumeric character, data manipulation.

These instructions can also be used for additional scaling if a product or quotient must be used in several decimal forms.

5.6.01 Shift Off Instruction

	Op Code	A	B
Shift Off	SLRO	0:14	0:14

The SLRO instruction first causes the 15 digits of the accumulator to be shifted left-off the number of positions specified by the "A" field. Any non-zero digit shifted off causes the overflow flag to be set. If all digits shifted off are zero, the flag is reset.

The 15 Accumulator digit positions are then shifted right-off the number of positions specified by the "B" field. Any non-zero digit shifted off does not set the overflow flag.

Digits shifted off are lost. Rounding is not performed.

If the "B" field is zero, the instruction is effectively a shift left-off; or if the "A" field is zero, the instruction is effectively a shift right-off.

5.6.02 Shift Off with Sign Instruction

	Op Code	A	B
Shift Off with Sign	SLROS	0:15	0:15

The SLROS instruction is the same as the SLRO instruction except that the sign position is also shifted.

This instruction may be used to shift alphabetic information.

6. FLAGS

Twenty-eight flags are provided in the system: four accumulator flags: four test flags; four operation control key (OCK) flags; four reader flags; four punch flags and eight general purpose flags. These flags may be set or reset by instruction and/or computer operations. Test flags are only set/reset by computer operations; General Purpose flags are set/reset only by instruction. (The print mask flags are not included in the above 28 since they are not affected by instructions or operations, and are generally considered as part of the identity of a particular mask.) The execution of program instructions can be dependent on these flag settings, by using "Skip" or "Execute" instructions.(See Section 9.)

6.1 ACCUMULATOR FLAGS

When the accumulator contains numeric information, it contains four flags (designated "A" flag group) in addition to the fifteen numeric digits:

1.	Sign	(-)
2.	Special	(S)
3.	Per Hundred	(C)
4.	Per Thousand	(M)

The M, C, and Sign flags can be set by the Numeric Keyboard instructions (see Section 1) or changed by the Arithmetic instructions (see Section 5). They can also be set, reset or changed by the Flag Set/Reset instructions (see 6.7). Data moved from the Accumulator to memory with "TRM" or vice versa with "TRA" retains the flag settings of that word.

Print Conditional

Certain Print Character instructions are effected by the Accumulator Sign flag (Section 3).

6.2 TEST FLAGS

The four Test Flags are designated the "T" flag group:

1. Accumulator Overflow Flag	(0)
2. Forms Limit Flag	(L)
3. Index Register Flag	(I)
4. Unassigned Flag	(U)

The Accumulator Overflow Flag (O) can be set or reset by Arithmetic and Shift instructions (section 5).

The Forms Limit Flag (L) is set by an Advance Platen (AL, AR, ALR, ALTO, and ARTO) or during a Keyboard instruction (Section 2). The Forms Limit Flag is reset initially by AL, AR, ALR, ALTO and ARTO.

The Index Register Flag (I) is set or reset by an Index Register instruction (Section 7).

6.3 OPERATION CONTROL KEY FLAGS

Operation Control Key Flags (the "K" flags) can be set or reset by the Flag Set/Reset instructions (6.7). In addition, when an Operation Control Key is used to terminate a Keyboard (Numeric or Typewriter) instruction, its corresponding flag is set.

OPERATION CONTROL KEY (OCK)	FLAG
1	1
2	2
3	3
4	4

All Operation Control Flags are reset by the next Keyboard instruction.

6.4 GENERAL PURPOSE FLAGS

Eight General Purpose Flags, divided into two groups ("X" and "Y") of four flags each, are included in the flag group.

These flags are set or reset only by the Flag Set and Reset instructions. They are numbered: 1, 2, 3, 4 for each group (X and Y).

6.5 READER FLAGS

Four flags (the "R" flags) are reserved for the tab card and punched tape reader input option and the Data Communication Processor. These flags may be set and reset as a function of the tape or card read instructions or the Data Communication Processor as well as by the flag instructions (section 6.7).

The Reader flags are numbered: 1, 2, 3, 4.

The state of these flags is displayed by four of the Input/Output Option Indicators. Refer to Figure 2-1 for the location of these indicators.

These flags can be used as general purpose flags only when a reader is not used as an input device.

6.6 PUNCH FLAGS

Four flags ("P" flags) are reserved for the Punched Tab Card or Paper Tape output option. These flags may be set and reset as a function of the tape or card punch instructions as well as the Flag Set and Reset instructions (Section 6.7). The Punch Flags are numbered: 1, 2, 3, 4.

The state of these flags is displayed by four of the Input/Output Option Indicators. Refer to Figure 2-1 for the location of these indicators. These flags may be used as general purpose flags when a punch is not used as an output device.

6.7 FLAG SET AND RESET INSTRUCTIONS

The Flag Set and Reset instructions allow complete manipulation of the flags in the following flag groups, under program control:

Designation	Flag Group
A	Accumulator Flags
K	Operation Control Key Flags
X	General Purpose Flags
Y	General Purpose Flags
R	Reader (Paper Tape or Card) Flags
P	Punch (Paper Tape or Card) Flags

6.7.01 Load Flags Instruction

	Op Code	A	B
Load Flags	LOD	A,K,X, Y,R,P	1,2,3,4 -,S,C,M,

The LOD instruction permits "setting" selected flags of any one flag group. The flag group is designated in the "A" field; the flags to be set are designated by numbers of symbols in the "B" field. Any or all of the four flags of a group may be set; All other flags in the group not set, are reset. Other groups are not affected.

6.7.02 Set Flags Instruction

	Op Code	A	B
Set Flags	SET	A,K,X, Y,R,P	1,2,3,4, -,S,C,M,

The SET instruction "sets" selected flags of any one flag group. The flag group is designated in the "A" field; the flags to be set are designated by numbers or symbols in the "B" field. Any or all of the four flags of a flag group may be set. All other flags in the group not set, are left unaltered.

6.7.03 Reset Flags Instructions

	Op Code	A	B
Reset Flags	RST	A,K,X, Y,R,P	1,2,3,4, -,S,C,M

The RST instruction "resets" selected flags of any one flag group. The flag group is designated in the "A" field; the flags to be reset are designated by numbers or symbols in the "B" field. Any or all of the four flags of a flag group may be reset; All other flags in the group not reset, are left unaltered.

6.7.04 Change Flags Instruction

	Op Code	A	B
Change Flags	CHG	A,K,X, Y,R,P	1,2,3,4, -,S,C,M

The CHG instruction complements the state (set/reset) of selected flags of any one flag group. That is, if a flag was set, it is reset; or if it was reset, it is set. The flag group is designated in the "A" field; the flags to be changed are designated by numbers or symbols in the "B" field. Any or all of the four flags of a flag group may be changed; All other flags in the group not changed, are left unaltered.

7. INDEX REGISTERS

Four index registers (1, 2, 3 and 4) are included in G.P. 300. They may be used to modify an instruction parameter field. Each index register is capable of storing the numbers zero through 255.

Index Registers serve several useful functions:

a. They can be loaded with numeric data (by instruction) that may signify a relative position within a range of numbers. The Index Register can then be used to modify the base address in an instruction to accumulate data. For example, in distributing a sales amount to one of 50 departments, assume that $5.00 goes to Department 3; Words 201 to 250 are set aside to receive the distribution: "3" is loaded in an index register; that index register is then used to modify an instruction that says "accumulate into word 200". When the modified instruction is executed, the $5.00 will actually accumulate in word 203. The instruction is not permanently modified, thus each entry will always be distributed relative to the base address of 200.

b. Index Registers can be used to count up to a limit, and when reaching the limit, set a flag to permit altering the path of instructions. For example, it may be necessary to know when an intermediate point on a form (other than the bottom) is reached. Assuming a fixed starting line on the form, an index register is incremented by one each time the form is advanced. When a specified limit is reached, a flag is set to signal the designated intermediate line.

7.1 INDEX REGISTER MODIFICATION

The following instructions are provided to load or change the value of any of the four index registers.

7.1.01 Loading Index Registers:

	Op Code	A	B
Load Index Register	LIR	1:4	0:255

The index register designated in the "A" field (1, 2, 3, or 4) is loaded with the number contained in the "B" field, which can be any plus value from 0 to 255. The prior contents are destroyed.

7.1.02 Incrementing Index Registers

	Op Code	A	B
Increment Index Register	IIR	1:4	0:255

The IIR instruction increments, by 1, the contents of the index register designated by the "A" field. The register is capable of storing the numbers 0 through 255. If the index register contains 255, incrementing causes the register to become 0. The "B" field designates a value which is compared to the contents of that index register.

If the contents of the index register, designated by the "A" field, is equal to the value of the B field before incrementing is effected, the Index Register Flag (one of the Test flags) is set. If an equal condition does not exist, the Index Register Flag is reset. Thus, if the flag is set during one incrementing, it will be reset during the very next incrementing. Therefore, it is necessary to test this flag after each incrementing, if a particular program routine is required after incrementing to a limit during a series of repetitive operations. The value in the "B" field does not halt incrementing or turn the register back to zero, once incrementing has reached that limit. It merely provides a signal that a procedure or function in the program has occurred a given number of times.

7.1.03 Decrementing Index Registers

	Op Code	A	B
Decrement Index Register	DIR	1:4	0:255

The DIR instruction decreases, by 1, the contents of the index register designated by the "A" field. If the index register contains 0, decrementing causes the value 255 to be entered into the register. The "B" field designates a value which is compared to the contents of that index register.

Except as noted above, the DIR instruction is the same as the IIR instruction. If the contents of the register are equal to the value specified in the B field before decrementing, the Index Register Flag is set; otherwise, the flag is reset.

7.1.04 Adding to Index Registers

	Op Code	A	B
Add to Index Register	ADIR	1:4	0:255

The number contained in the "B" field is added to the contents of the index register designated by the "A" field. Both the "B" field number and the register contents are always plus. If the sum of the prior register contents and the "B" field value equals 256 the register is reset to zero; If the sum is greater than 256, only the amount that exceeds 256 is retained in the register; and in both cases, the overflow causes the Index Register Flag to be set; if the sum is under 256, the flag is reset. For example, if the value 40 is added to index register contents of 230 (40 + 230 = 270), then 14 is the resulting contents of that register (270 - 256 = 14), and the Index Register test flag is set.

7.1.05 Transferring Accumulator Contents to Index Register

	Op Code	A	B
Transfer Accumulator to Index Register	TAIR	1:4	

The TAIR instruction transfers the contents of the Accumulator to the index register designated in the "A" field. The prior contents of that index register are destroyed. The value of the Accumulator is treated as an absolute number, regardless of any "assumed" decimal places during entry into the Accumulator, and regardless of the setting of the Sign flag.

Since an index register has a capacity of 255, an Accumulator value greater than 255 that is transferred to an index register is accepted as that amount that exceeds a multiple of 256 (up to 1024) For example:

If the accumulator contains 316, then 60 is transferred (316 - 256 = 60);

If the accumulator contains 525, then 13 is transferred (525 - 512 = 13);

If the accumulator contains 256, then 0 is transferred (256 - 256 = 0).

If the value in the accumulator exceeds 1023, an undeterminable value will result in the index register; this would be a programing error.

7.2 MODIFICATION OF PROGRAMS WITH INDEX REGISTERS

Programs can be altered or modified in two ways with index registers. The Index Register Flag (one of the test flags) can be interrogated by means of a "Skip" or "Execute" instruction, directing the program to alternate routines based on the flag setting (see section 9); the flag having been set/reset by IIR, DIR, or ADIR as discussed in 7.1 above.

In addition, the contents of an index register can be used to modify the parameter field(s) of an instruction, temporarily for one execution of that instruction, by means of a "Modify" instruction.

7.2.01 Modify By Index Register Instruction

	Op Code	A	B
Modify By Index Register	MOD	1:4	

The MOD instruction provides for adding the contents of the index register designated by the "A" field to the parameter(s) of the next instruction in program sequence following the MOD instruction. The modified instruction is then executed in accordance with the combined parameter values.

This modification does not change the instruction stored in memory. It occurs during the execution of the instruction, as the parameter field is extracted from the instruction and placed in a special register. The MOD instruction affects the execution of only the one instruction immediately following it.

Example:

Assume Index Register #1 contains 35.

```
MOD   1
ADM  200
```

The index register value of 35 combined with the instruction parameter of 200 causes the contents of the Accumulator to be added into word location 235.

Although the MOD instruction is most generally used to modify those instructions which address word locations in memory, it may also be used to modify the parameters of most other instructions. The contents of the index register are added to the parameter field in modulo 256. Modulo 256 means that if the index register (maximum capacity of 256), when added to the parameter field (also a maximum capacity of 256 in machine language), exceeds 256, a "carry" of 1 is generated and the excess value starts back at zero. For example, an index register with a value of 150, when added to an ADM 200 instruction, generates a "carry" of 1 and a remaining parameter of 94 (350 - 256 = 94). The carry is propagated to the machine language operation code. Therefore, caution must be exercised in modifying most instructions since any "carry" is added to the OP Code and changes it to the next OP Code in sequence. Some instructions actually have 2 machine language OP Codes to permit a potential parameter range of 0 up to 511; in this case, a "carry" beyond 511 would create a third Op Code which may be an entirely different instruction. Also, the maximum amount of user (Normal) memory available in a given system determines the real potential parameter range. The results of the addition of the index register to the parameter of the instruction that is being modified vary with the type of parameter. Different types of instructions will have the "A" parameter, or the "B" parameter, or both the "A" and "B" parameters modified. Some instructions cannot be modified.

The following paragraphs list the instructions in categories according to which parameters can be modified, and indicate the maximum combined value (instruction parameter + index register value) beyond which a "carry" will cause an improper result and may be considered a programing error. Protection against this type of error can be accomplished by "sizing" a number by program before it is used in an index register for modification. "Sizing" can determine if the number falls within a permissible value range.

7.2.02 Instructions in Which Only the "A" Parameter Can Be Modified

The contents of the index register specified by the MOD instruction are added to the "A" parameter. If the combined value exceeds the range shown for each instruction parameter, either a "carry" will create a new instruction, or the instruction will otherwise be improperly modified:

Op Code	A	B	Op Code	A	B	Op Code	A	B
ADA	0:N		LRLR	0:255		SUA	0:N	
ADM	0:N		LSR	0:15		SUM	0:N	
AL	0:255		LXC	0:255		TAIR	1:4	
ALR	0:255		MUL	0:N		TK	0:150	
ALTO	0:255		MULR	0:N		TKM	0:150	
AR	0:255		OC	0:255		TRA	0:N	
ARTO	0:255		PA	0:N		TRAB	0:15	
BRU	0:N	0:3	PAB	0:150		TRB	1:15	
CLM	0:N		POS	1:150		TRBA	0:16	
CPA	0:N		REAM	0:150		TRF	0:255	
DIV	0:N		RCP	1:255		TRM	0:N	
EAM	0:150		RTK	0:150		TSB	1:15	
IRCP	0:255		RTKM	0:150		XA	0:N	
LKBR	0:N		RXEAM	0:150		XB	0:255	
LLCR	0:255		RXTK	0:150		XEAM	0:150	
LLLR	0:255		RXTKM	0:150		XMOD	0:255	
LPKR	0:N		SCP	1:255		XPA	0:N	
LPNR	0:N		SRJ	0:N	0:3	XTK	0:150	
LRBR	0:N		SRR	1:4		XTKM	0:150	
LRCR	0:255							

7.2.03 Instructions in Which Only the "B" Parameter Can Be Modified

In the following instructions, only the "B" parameter field is modified; other parameter fields are unmodified. The contents of the index register is added to the "B" parameter of the instruction. If the combined value exceeds 255, either a "carry" will create a different instruction, or the instruction will otherwise be improperly modified:

Op Code	A	B
ADIR	1:4	0:255
DIR	1:4	0:255
IIR	1:4	0:255
LIR	1:4	0:255

7.2.04 Instructions in Which Both "A" and "B" Parameters Can Be Modified; Each Parameter Can Specify Only One Item:

In these instructions, either, or both, the "A" or the "B" parameters can be modified. The "C" parameter, if one exists, is not modified. Since the parameter field of an instruction can have a maximum hexadecimal value of 256, the "A" and "B" parameters combined cannot exceed 256; The sixteen possibilities in the "B" parameter requires a value from 0 to 15 in the index register for modification; the sixteen possibilities in the A parameter field require a value expressed in multiples of 16 (reflecting the digit position value of the A parameter in the instruction format).

The following table illustrates the proper values to be loaded in the index register to achieve the desired values for the "A" and "B" parameters.

Number desired in A field	"m" Value to be contained in Index Reg.	Number desired in B field	"n" Value to be contained in Index Reg.
0	0	0	0
1	16	1	1
2	32	2	2
3	48	3	3
4	64	4	4
5	80	5	5
6	96	6	6
7	112	7	7
8	128	8	8
9	144	9	9
10	160	10	10
11	176	11	11
12	192	12	12
13	208	13	13
14	224	14	14
15	240	15	15
"m" + "n" = total value to be contained in register.			

Example: Modify "NK" 0 0" to provide 8 whole numbers and 3 decimal fractions:

```
Parameters required:        Index Register value required:
   A  =  8           =           128

   B  =  3           =             3
                                 ---
                                 131  (total value)

Thus:  LOD     1      131
       MOD     1
       NK      0        0
```

The index register value of 131 modifies the NK instruction to permit 8 whole numbers and 3 fractions.

Any time that the modification of the B parameter results in a carry (exceeds 15), the carry will add to the A parameter changing its specification. A carry resulting from modification of the A parameter (exceeds 255) will add to the Op Code causing an improper modification.

Op Code	A	B	C
ADK	0:14	0:9	
CLA	0:15	0:9	
EXL	0:15	0:15	1:4
INK	0:14	0:9	
NK	0:15	0:15	
NKCM	0:15	0:15	
NKRCM	0:15	0:15	
NKR	0:15	0:15	
SKL	0:15	0:15	1:4
SLRO	0:14	0:14	
SLROS	0:15	0:15	
SUK	0:14	0:9	
RNK	0:15	0:15	

Op Code	A	B
PN	0:14	0:15
PNS+	0:14	0:15
PNS-	0:14	0:15
TRCB	0:15	0:15
XN	0:14	0:15
XPN	0:14	0:15
XPNS+	0:14	0:15
XPNS-	0:14	0:15
XC	0:15	0:15

7.2.05 Instructions in Which the "A" and "B" Parameters Can Be Modified; One Parameter Can Specify One or More Items

The A and B parameters of some instructions represent a binary pattern to the machine. The PKA, PKB instructions as well as the LOD, SET, RST and CHG flag instructions are programed by listing the digits 1-8 (in the case of the PK instructions) and 1-4 (in the case of the flag instructions) in the "A", "B" or "A" and B" parameters for the desired pattern.

The EX, EXE, SK and SKE instructions are programed by listing the digits 1-4 in the "B" parameter to designate the particular flag pattern desired.

To modify this binary pattern, it is necessary to find the decimal equivalent of the pattern desired and add it to the Index Register used in the MOD instruction. The value table below may be used to determine the number necessary to obtain the desired pattern.

Value Table			
	Decimal Equivalent		
No. in A, B or A & B Fields	PKA PKB A & B field	Flag Instructions B field only	Flag Instructions A field
1	1	2	Punch = 0
2	2	4	Read = 16
3	4	8	X = 64
4	8	1	Y = 80
5	16		T = 128
6	32		K = 144
7	64		A = 192
8	128		

For PK's, add together all of the equivalent values for the PK's specified in the "A" field, to determine the total value which must be loaded in the index register.

For Flag instructions (Set/Reset and Skip/Execute), add together the equivalent values for the flags specified in the B parameter; If the flag group is also to be modified, add its value to the total value for the individual flags, and the resulting sum is the value to be loaded in the index register.

To modify these instructions it is essential to originate them with 0 in the parameter fields and the desired pattern in the index register.

If these instructions are originated with some significant value in the parameter fields, an attempt to modify the parameters can propagate a carry which will be added to the Op Code, changing it to the next Op Code in sequence.

Op Code	A	B	C
PKA	12345678		
PKB	12345678		
LOD	A,K,X, Y,R,P	1234	
SET	A,K,X, Y,R,P	1234	
RST	A,K,X, Y,R,P	1234	
CHG	A,K,X, Y,R,P	1234	
EX	A,T,K,X, Y,R,P	1234	1:4
EXE	A,T,K,X, Y,R,P	1234	1:4
SK	A,T,K,X, Y,R,P	1234	1:4
SKE	A,T,K,X, Y,R,P	1234	1:4

7.2.06 Modification Of Print Character Instruction

The character in the "A" field of a PC instruction may be modified to obtain a different character. Appendix B contains a chart listing the 64 print characters with the corresponding internal codes. The MOD instruction adds the contents of the index register to the internal code of the character in the "A" parameter of the Print Character Instruction. The value to be loaded in the index register is the difference between the value of the resulting printed character and the value of the character specified in the "A" parameter.

For example:
```
MOD  1
PC   A
```

If "PC A" (A = value of 65) is to be modified to print "M" (M = value of 77), a value of 12 is loaded into index register 1 (77 - 65 = 12).

The character specified in the "A" parameter must have a lesser indexing value than the desired character to be printed.

This applies to PC, PC+, PC–, and PCP.

7.2.07 Modification of a Modify Instruction

A Modify instruction may be used to modify another Modify instruction with the same or a different index register. Each MOD instruction in sequence adds the contents of the specified register to the contents of the ensuing specified register. The last MOD instruction then adds the cumulative total to the parameter of the next instruction, according to the rules for that particular type of instruction.

7.2.08 Unmodifiable Instructions

The following instructions cannot be modified:

Op Code	A	Op Code	A	Op Code	A
REL		RSA		LTN	
NOP		RRA		LSN	
CC		RTN		LPR	
ALARM		RTH		LPF	
RR		RSN			
STOP		RPR			
REM		RPF			
SKZ	1:4	LSA			
EXZ	1:4	LRA			

8. BRANCH INSTRUCTIONS

A word of memory contains four instructions. The location of an instruction within a word is referred to as a syllable.

The normal sequence of instructions is determined by the program counter. The program counter contains the memory location of the instruction being executed. The program counter consists of three parts:

Syllable address	0:3		In programing, the block number is normally not used since the system software (assembler) compensates for the block number and recognizes sequential word locations 0:N.
Word address	0:255	word= (0:N)	
Block address	0:1		

Upon completion of an instruction (other than a branch instruction), the next instruction is determined by adding one to the program counter. If the syllable address is zero, one, or two, it is increased by one. If the syllable address is three, it is reset to zero, and the word address is increased by one. If the syllable address is three and the word address is 255, both sections are reset to zero and one is added to the block section. The branch instructions manipulate the contents of the program counter to alter the sequential execution of programed instructions.

8.1 BRANCH UNCONDITIONAL INSTRUCTION:

	Op Code	A	B
Branch Unconditional	BRU	0:N	0:3

Load the program counter with the contents of the "A" and "B" fields. The "A" field will be the new word address; the "B" field, the new syllable address within the word. The next instruction executed is determined by the new contents of the program counter.

8.2 SUBROUTINE JUMP AND RETURN

Many times in the execution of a program, a routine is called upon several times and from various locations in the program. The Unconditional Branch would be a way of getting to the routine from many different locations but once in the routine, there is no simple way to return to the same relative address as before using Branch, since each "Branch Back to where you came from" is not possible with the instruction "Unconditional Branch". (BRU). Modify might be used to modify the Branch instruction. Flags could be "set" and tests made to determine where to return to. However, Subroutine Jump and Subroutine Return do it automatically through the use of a Subroutine-return "Stack", which has the capacity to store four instruction addresses.

When a Subroutine Jump instruction is executed, the contents of the program counter (before the jump takes place) is increased by "one" and stored in the top of the stack. The other addresses in the stack are pushed down one. If the stack contains four return addresses when the subroutine jump is executed, the return address at the bottom of the stack is lost.

When a Subroutine-Return instruction is executed, the program counter is loaded with a memory instruction address from the stack followed by a push-up of the stack.

8.2.01 Subroutine Jump Instruction

	Op Code	A	B
Subroutine Jump	SRJ	0:N	0:3

The Subroutine Jump instruction causes the contents of the program counter to be increased by one and then stored in the top of the Subroutine-Return stack (for return purposes).

The program counter is then loaded with the contents of the "A" and "B" fields, where "A" is the word address and "B" the syllable address within the word. The next program instruction executed, following the SRJ instruction, is determined by the new contents of the program counter.

8.2.02 Subroutine Return Instruction

	Op Code	A	B
Subroutine Return	SRR	1:4	

SRR 1: The most common operation will call for a one in the "A" field. For this condition the return address stored at the top of the Subroutine-Return stack is transferred to the program counter. This is the same address (plus one) that was in the program counter before execution of the last Subroutine Jump instruction. The execution of the SRR 1 instruction will also cause all the addresses in the Subroutine-Return stack to be pushed up one.

The next instruction executed is determined by the new contents of the program counter.

SSR 2: When the "A" field of the SRR instruction contains a "2", the addresses in the subroutine-return stack are pushed up 1 position, (this address is lost), the address at the top of the stack is transferred to the program counter, and then the Subroutine-Return stack is again pushed up 1 position.

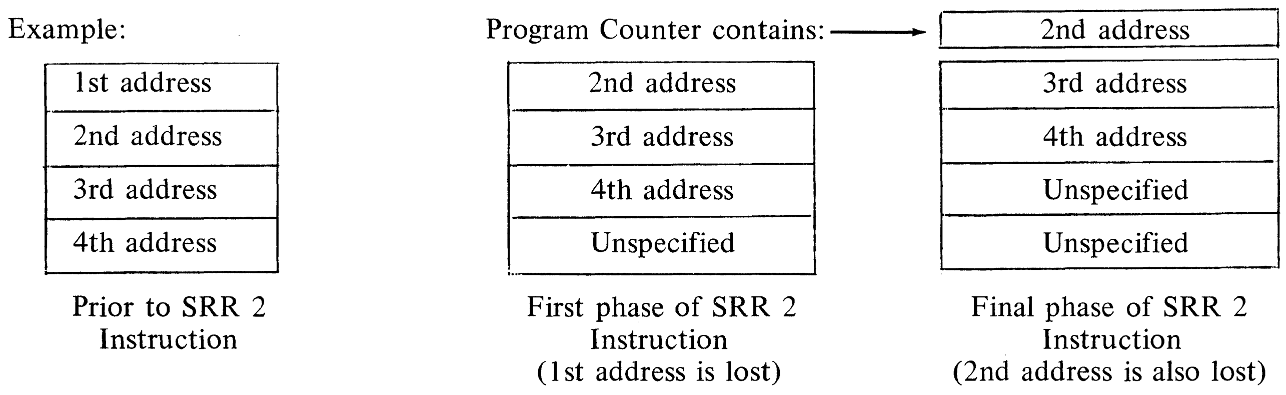

Example:
Program Counter contains:
2nd address
1st address
2nd address
3rd address
4th address
2nd address
3rd address
4th address
Unspecified
3rd address
4th address
Unspecified
Unspecified
Prior to SRR 2 Instruction
First phase of SRR 2 Instruction (1st address is lost)
Final phase of SRR 2 Instruction (2nd address is also lost)

Fig. 8-1 Subroutine Return Address Stack

SRR 3: When the "A" field of the SRR instruction contains a "3", the addresses in the subroutine-return stack are pushed up 2 positions, (both addresses are lost), the address then at the top of the stack is transferred to the program counter, and then the subroutine-return stack is again pushed up 1 position. See Example above and recognize that the Final phase would leave only the 4th address in the stack and the 3rd address would be in the Program Counter.

SRR 4: When the "A" field of the SRR instruction contains a "4", the addresses in the subroutine-return stack are pushed up 3 positions, (all three addresses are lost), the address at the top of the stack is transferred to the program counter, and then the Subroutine-Return stack is again pushed up 1 position (leaving the stack containing only unspecified information generated as a result of the stack function), and the 4th address would be in the Program Counter. (See example above.)

Any attempt to use Subroutine Return unless preceding it by the use of a Subroutine Jump will result in unspecified results as the Return is governed by the contents of the Stack.. (This may even be loaded as a result of an earlier program or operation unrelated to the present program.) A possibility of this happening is if the Subroutine program is entered into by a BRU instruction. When it comes to the SRR instruction, no valid Return has been specified and erratic results will ensue.

9. SKIP AND EXECUTE INSTRUCTIONS

Skip and Execute instructions are provided to direct the path of program execution to alternate routines or procedures based on the following conditions:

1- An Accumulator digit value compared to a constant (less than)

2- Zero or non-zero state of the Accumulator

3- Comparison of 2 Alpha words (equal to, less than, greater than)

4- Flag settings (set or reset) of specified flags

The Skip and Execute instructions interrogate the following flags:

Flags	Flag Symbol
Group A - Accumulator Flags*	
Sign	-
Special	S
Per Hundred (C)	C
Per Thousand (M)	M
Group T - Test Flags**	
Accumulator Overflow	O
Forms Limit	L
Index Register	I
Unassigned	U
Group K - Operation Control Keys Flags	1234
Group X - General Purpose X Flags	1234
Group Y - General Purpose Y Flags	1234
Group R - Reader Flags	1234
Group P - Punch Flags	1234

All Skip instructions cause instruction(s) following the Skip instruction to be skipped if the condition specified is true. The "A" field specifies which one of the seven groups of flags is to be tested. The "B" field of the instruction is used to specify the individual flags (up to four) to be tested if in the "set" state, while the "C" field specifies the number of instructions to be skipped.

All Execute instructions cause the instruction(s) following the Execute instruction to be executed if the condition specified is true. The "A" field specifies which one of the seven groups of flags is to be tested. The "B" field of the instruction is used to specify the individual flags (up to four) to be tested if in the "set" state, while the "C" field specifies the number of instructions to be executed.

9.1 SKIP AND EXECUTE FLAG INSTRUCTIONS

9.1.01 Skip Flag Instructions

	Op Code	A	B	C
Skip If Any Flag	SK	A,T,K, X,Y,R,P	OLIU 1234 -SCM	1:4
Skip If Every Flag	SKE	A,T,K, X,Y,R,P	OLIU 1234 -SCM	1:4

* The Accumulator Flags may be set or reset by numeric keyboard operations (Section 2), by arithmetic operations (Section 5), or by Flag instructions (Section 6).

**The Test flags may be set or reset by forms handling operations (Section 4), by arithmetic operations (Section 5), and by Index Register instructions (Section 7).

The SK instruction causes the next "C" instruction(s) in sequence to be skipped if ANY of the flags specified in the "B" field (of the flag group specified by the "A" field) are set. Otherwise, the next instruction(s) is executed.

The SKE instruction causes the next "C" instruction(s) in sequence to be skipped if EVERY ONE of the flags specified in the "B" field (of the flag group specified by the "A" field) are set. Otherwise, the next instruction(s) is executed.

Example: Skip 4 instructions if OCK 4 was used to bypass certain instructions:

```
Op Code       A        B        C
   S
SK ─┐         K        4        4
--- │         If OCK 4 was used, skip 4 in-
--- │         structions. If any other OCK's
--- │         were used, execute the next
--- │         instruction.
--- <┘
```

The SKE would be used in the same manner as above if, for example, every 'X' Flag 1,2,3,4 were "set":

```
Op Code       A        B          C
SET           X        1,4
 S
SET           X        2,3
 S
SKE ─┐        X        1,2,3,4    3
---  │        Three instructions would be skipped
---  │        since all four flags were set as illustrated.
---  │
--- <┘
```

9.1.02 Execute Flag Instructions

	Op Code	A	B	C
Execute If Any Flag	EX	A,T,K, X,Y,R,P	OLIU 1234 -SCM	1:4
Execute If Every Flag	EXE	A,T,K, X,Y,R,P	OLIU 1234 -SCM	1:4

The EX instruction causes the next "C" instruction(s) in sequence to be executed if ANY of the flags specified in the "B" field (of the flag group specified by the "A" field) are set. Otherwise, the next "C" instruction(s) are skipped.

The EXE instruction causes the next "C" instruction(s) in sequence to be executed if EVERY ONE of the flags specified in the "B" field (of the flag group specified by the "A" field) are set. Otherwise, the next "C" instruction(s) are skipped.

Example:

The Shift Register would be loaded with "2" if the "C" key was used and with "3" if the "M" key was used: Assume "M" key was used:

	Op Code	A	B	C
	EX	A	C	1
(not executed)	LSR	2		
	EX	A	M	1
(executed)	LSR	3		
	---	Since only the "M" key was used, only the instruction pertaining to the "M" key was executed, the other being skipped.		

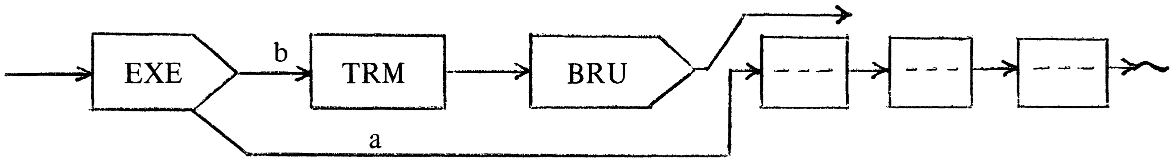
Example:

Assume the EXE instruction is reached from one of two program routines (a or b):

```
                                   Op Code       A          B          C

Routine a          Routine b
                   Set X 1
Set X 1

                   Set X 2

                 ┌-EXE             X          1,2          2
               a │ TRM             100
                 │ BRU ──b──>
                 └>---
                   ---
                   ---
                   ---
```

Coming in from Routine "a", the two instructions following EXE are not executed since EVERY flag was not "set". Routine "b" did "set" the flags and thus all instructions were executed.

Flowcharted, the program appears as follows:



9.2 ACCUMULATOR LESS THAN CONSTANT, SKIP AND EXECUTE

9.2.01 Accumulator Digit Less Than Constant Skip

	Op Code	A	B	C
Skip If Digit Less than Constant	SKL	0:15	0:15	1:4

The SKL instruction causes the next "C" instructions in sequence to be skipped if the absolute value of the Accumulator digit specified by the "A" field is less than the constant contained in the "B" field. Otherwise, the next instruction(s) is executed. All other Accumulator digits are ignored. If the specified Accumulator digit is equal to the Constant, the next instruction(s) is executed.

Example:

A 5-digit customer number signifies that Tax is to be calculated on the invoice total. A 4-digit number signifies no tax is to be calculated. Assume 1524 was the number tested:

```
                          Op Code         A          B          C
                             (
                             )
                            SKL──┐        4          1          1
Go to calc. tax  <───────── SRJ  │
                            — <──┘        Digit position 4 is a "0" which is less
                            —             than "1" so Tax calculation (SRJ) is
                                          bypassed.
```

9.2.02 Accumulator Digit Less Than Constant Execute

	Op Code	A	B	C
Execute If Digit Less Than Constant	EXL	0:15	0:15	1:4

The EXL instruction causes the next "C" instructions in sequence to be executed if the absolute value of the Accumulator digit specified by the "A" field is less than the constant contained in the "B" field. Otherwise, the next "C" instructions are skipped. All other Accumulator digits are ignored. If the Accumulator digit is equal to the Constant, the next instruction(s) is skipped.

NOTE: The contents of the Accumulator are not disturbed.

Example:

Based on a product number, an amount is to be distributed 50 ways if digit 2 is less than a value of "5". If greater than "5" distribute the amount to one total. Assume the number listed is 243:

```
                              Op Code      A       B       C
                                 (
                                LIR        1       0
                                 (
                                 )
                                NK         3       0
                            ┌── EXL        2       5       4
The "2" in digit position 2 │   —
is less than "5", thus the  │   —
4 instructions are executed.│   —
                            │   —
                            └─> —
                                —
```

If 643 had been indexed, skipping of the four commands following EXL would have taken place.

NOTE: The value expressed in the "B" parameter field represents a single digit and is normally a value "0" through "9". If the value exceeds "9", it would only be for expressing a hexadecimal value in that single digit position.

9.3 ACCUMULATOR ZERO SKIP AND EXECUTE

9.3.01 Accumulator Zero Skip

	Op Code	A	B
Skip If Accumulator Zero	SKZ	1:4	

The SKZ instruction causes the next "A" instructions in sequence to be skipped if the contents of the Accumulator is zero. Otherwise, the next instructions is executed.

Example:

Bypass a calculation routine if there is no amount upon which the calculation is based:

```
                          Op Code     A      B      C

                          TRA        200
                          SKZ ──┐    1
To calculation routine    BRU   │
                          --- ←─┘     If the Accumulator is zero, skip the next
                          ---         instruction which would go to a calculation
                          ---         routine based on the amount in the Accumulator.
                          ---
```

9.3.02 Accumulator Zero Execute

	Op Code	A	B
Execute If Accumulator Zero	EXZ	1:4	

The EXZ instruction causes the next "A" instructions in sequence to be executed if the value of the accumulator is zero. Otherwise, the next "C" instructions is skipped.

NOTE: The accumulator sign digit (A_{15}) is not tested or altered by this instruction.

Example:

Enforce a Keyboard Listing:

```
  Op Code      A          B

┌→NK           5          1
│ EXZ          1
└─BRU
  PNS-         If no keyboard listing was made and an
  ---          OCK depressed, the BRU would be executed
  ---          and the listing would again be called for. If
  ---          a listing was made, the BRU would be ignored.
  ⌇
```

9.4 COMPARE ALPHANUMERIC

Compare Alphanumeric is similar to the skip and execute instructions because when two words are compared, the result of the comparison is the skipping or executing of instructions based on an equal to (=), greater than (>) or less than (<) condition.

9.4.01 Compare Alphanumeric

	Op Code	A	B
Compare Alphanumeric	CPA	0:N	

The CPA instruction compares the contents of the memory word, specified by the "A" field, to the contents of the Accumulator. Execute the next instruction in sequence if the content of the specified word is equal (=) to the content of the accumulator. Skip the first instruction in sequence and execute the second instruction, if the content of the specified word is less than (<) the content of the Accumulator. Skip the first two instructions in sequence and execute the third instruction if the content of the specified word is greater than (>) the content of the accumulator.

Refer to Appendix B for the collating sequence of the character set. The low order character is the "end alpha" code (0,0) followed by the space code (2,0); the high order character is the special character for ~ (<>) (7,E).

10. MISCELLANEOUS INSTRUCTIONS

10.1 AUDIBLE ALARM FOR OPERATOR ERRORS

When an operator error can be determined programmatically, the operator can be alerted to this error through the instruction "Alarm". As an example, it may be desired to enforce the entry of a product number through the numeric keyboard. If the operator fails to enter the number and terminates the instruction with an OCK, this is a legitimate system operation; however, it is an operator error which can be brought to the operator's attention with the ALARM instruction. Normally, after testing for the entry and finding none, the program would execute ALARM and branch back to that entry instruction.

10.1.01 Alarm Instruction

	Op Code	A	B
Alarm	ALARM		

Execution of the ALARM instruction will sound the Error Alarm once. The system does not go into the error state.

10.2 No Operation Instruction

	Op Code	A
No Operation	NOP	

No operation is performed. However, 10 milliseconds are expended when this instruction is used. Program execution continues sequentially uninterrupted.

10.3 Stop Program Instruction

	Op Code	A	B
Stop	STOP		

The STOP instruction halts the execution of a program in progress and returns the computer to the READY MODE. From the Ready Mode, the operator can then initiate the loading of a new program (Load - PK 2), the selection of a utility routine (Utility - PK 3), or re-initiate the same program just halted (Start - PK 1). As an example, this instruction might be used after printing out grand totals at the completion of a particular job.

11. DATA COMMUNICATIONS INSTRUCTIONS

Recognizing and responding to Polls and Selections from the data center, and the control of transmission of messages from the TC 500 and the receipt of messages directed to the TC 500 are automatic functions of the Data Communication firmware. The user program in the TC 500 is responsible for the preparation of messages to be sent to the data center and for the use of any message data that has been received from the data center.

The Data Communication memory has two buffers of 256 characters capacity each; one is the Receive Buffer, the other is the Transmit (Send) Buffer. Associated with each buffer is a flag and indicator light:

Message Received Flag (R2) and Indicator
(Input Indicator light 3)

Transmit Ready Flag (R3) and Indicator
(Input Indicator light 4)

All messages are received from the data center into the Receive buffer, and all messages are transmitted to the data center from the Transmit buffer. When a message has been received, the user program must provide for transferring the message data from the Receive buffer into the Accumulator and/or Normal memory for printing, accumulation, processing, etc. When a message is to be transmitted, the user program must provide for transferring the message data from Normal memory and/or from the keyboard into the Transmit buffer. The Receive and Transmit instructions covered in this section permit the transfer of an entire Receive or Transmit buffer to or from the Data Communication memory, or permit the message to be broken apart or assembled in small sections directly in the Data Communication memory. The frequency of transmissions and the availablility of Normal memory will determine which method should be used.

The TC 500 Data Communication processor may be in both a Receive Ready state and a Transmit Ready state simultaneously, and responds to whichever (Select or Poll) occurs first; upon responding to the first occurence, it is then immediately ready to respond to the second, and in the meantime, the user program in Normal memory may proceed with other work.

11.1 RECEIVE READY STATE

The TC 500 is placed in a Receive Ready State by programing a RESET instruction to reset the Message Received Flag (Reader flag 2). This indicates to the Data Communication processor that the user program has finished with the last message received, and thus, 'the Receive buffer may accept another message. This permits the Data Communication processor to respond with an ACK (acknowledge) to the next Selection from the data center (after checking the parity of each character of the Selection message); it responds with a NAK when it is not receive ready. After receiving the ACK response, the data center transmits the message to the TC 500:

The Data Communication processor parity checks each character, places it in the Receive buffer, checks the Block Check character for longitudinal parity accuracy, checks the Transmission number, sets the Message Received flag (R2) and turns on the Message Received indicator (Input indicator 3). The user program can determine that a message has been received by interrogating the flag using SKIP and EXECUTE instructions, and can then perform any necessary operations with the message data.

Once a message has been received and the Message Received Flag is turned on, the Data Communication processor does not permit the receipt of other messages until the flag is turned off by the user program. Thus, the message may remain in the Receive buffer indefinitely, and the user program may continue to operate on other work (such as preparing a message for transmission) until it is convenient to use the received message data.

The receipt of a valid message while in the Receive Ready state causes the automatic transmission of an ACK to the data center. The receipt of a message that has character or longitudinal parity errors causes the automatic transmission of a NAK to the data center. If the TC 500 does not receive a message from the data center following acknowledgement to the data center of a Selection, no response is made by the TC 500. This is treated in the same manner as failure of the data center to receive a transmitted ACK or NAK. It causes a "time out" at the data center and the Selection message must be re-transmitted.

If the message transmitted to the TC 500 from the data center contains more than 255 characters of text the TC 500 will respond with a negative acknowledgement (NAK).

11.2 TRANSMIT READY STATE

Messages to be transmitted are prepared and stored in the Data Communication Transmit buffer by the user program. When the message is complete, or when it is known that all positions available in the buffer have been filled, the user program sets the Transmit Ready Flag (R3) by means of a SET instruction. The Data Communication processor then sets the remote in the transmit state so that when the next poll is addressed to this remote, automatic transmission will occur. In the meantime, the user program may commence with preparation of the next message to be transmitted or it may begin processing a message that has been received.

When the Poll is received from the data center and each character of the poll message has been parity checked, the Data Communication processor initiates the transmission of the message. It automatically inserts the Communication Control Codes and Transmission number (SOH, AD1, AD2, TR #, STX, ETX), generates a parity bit for each character, generates a longitudinal parity character (BCC), placing it after the ETX, and transmits the message.

Following transmission of the message, the Data Communication processor is in a Polling Message Response state and awaits receipt of an acknowledgement (ACK) from the data center; after receiving the ACK, it transmits an EOT character. If the data center responds with a NAK, the Data Communication processor automatically retransmits the message. Failure of the data center to transmit a recognizable character while the Data Communication processor is in the Polling Message Response state will not affect the remote. It is still output ready and the message can be re-transmitted when this address is re-polled. Failure of the data center to receive EOT may cause a "time out" at the data center indicating that the TC 500 must be repolled.

When transmission of the message is successful, the Data Communication processor automatically increments the Transmission number, turns off the Transmit Ready light, and resets the Transmit Ready Flag (R3). This permits the user program to determine that the Transmit buffer is available to receive data from the keyboard or Normal memory for the next message.

11.3 DATA COMMUNICATION RECEIVE BUFFER

The Receive Buffer has a capacity of 256 characters (32 words of 8 characters each, each character occupies 8 bits). When a message is received, only the text of the message and ETX are placed in the Receive Buffer; that is, all of the Heading, including the Start of Text character (STX), and the Block Check character (BCC) are stripped off of the message. The End of Text character (ETX) will be placed in the Receive Buffer immediately following the last character of the actual text. If the text is greater than 255 characters, the Data Communication Processor will return a Negative Acknowledge (NAK) to the data center.

When a message has been correctly received, the Message Received Flag (R2) is set and the Message Received Indicator light is turned on. The user program determines that a message has been received by interrogating flag R2 using Skip or Execute instructions and branching to an appropriate routine to break apart or "unpack" the message data for printing, processing, etc. This "unpacking" may be done directly from the Receive buffer, or the entire contents of the Receive buffer may first be transferred into a Normal memory working area of 32 words which is then referred to as a "Receive Record Area" and which may be in any available section of Normal memory (except words 0 to 31). Several such Receive Record areas may be used if desired. When a message has been transferred from the Receive buffer to a Receive Record area (prior to unpacking), it permits receipt of another message while the first message is being processed.

The message data is contained in the Receive buffer or Receive Record area without regard to word boundaries, but rather as one continuous string of characters. The instructions used to unpack the message automatically keep track of word boundaries and the character position location in the buffer or record area so that the program may access one character or any size group of characters, thus providing complete character addressability.

11.4 DATA COMMUNICATION TRANSMIT BUFFER

The Transmit Buffer also has a capacity of 256 characters (32 words of 8 characters each, each character occupies 8 bits). In preparing a message for transmittal to the data center, only the text of the message needs to be assembled and may be up to 255 characters in length. The heading, STX, ETX and BCC characters are automatically inserted in the message during transmission. The ETX is automatically placed after the last actual character of text and the remainder of the buffer capacity is ignored during transmission. If the text is more than 255 characters, only 255 characters are accepted and ETX is inserted as the 256th character. The Transfer instruction is terminated and the Overflow Test flag is set when an attempt is made to place more than 255 characters in the buffer or record area. The user program may take corrective action after having interrogated this test flag.

The Transmit Buffer is available to receive data for the next transmission once the Transmit Ready Flag (R3) has been Reset (by the Data Communication processor). The user program makes this determination by interrogating flag R3 with Skip or Execute instructions and branching to an appropriate routine to load data into the Transmit buffer.

The message may be constructed directly in the Transmit Buffer, or it may first be assembled in a Normal memory working area of 32 words which is then referred to as a "Send Record Area" and which may be in any available section of Normal memory (except words 0 to 31). Several such Send Record areas may be used if desired. When the message has been completed, one instruction permits the entire contents of the Send Record area to be transferred to the Transmit buffer.

The instructions used to construct a message include the capability to place the message data in the buffer or record area sequentially without regard to word boundaries, as one continuous string of characters. As the message is assembled, the instructions automatically keep track of word boundaries and the character position location of the last character entered, thus providing complete character addressability.

11.5 DESIGNATING BUFFERS, RECORD AREAS, AND CHARACTER POSITIONS

The unpacking of messages received and the constructing of messages to transmit involves moving or transferring data FROM certain areas of memory TO certain other areas of memory. Designating the actual "FROM" location and "TO" location is accomplished by loading special registers which specify the word and character location during the execution of certain instructions (refer to 11.6 and 11.7) which are designed to pack and unpack messages.

11.5.01 Load Receive Buffer Register

Op Code	A	B
LRBR	0 or 1:N	

The LRBR instruction specifies the starting word location FROM which data is to be transferred during the execution of certain character transfer instructions described in 11.6 and 11.7.

MESSAGES RECEIVED: The LRBR instruction designates the area of memory from which the message is to be unpacked. A parameter of "O" selects the first word of the Receive Buffer; a parameter of 32 to "N" would be used to select the beginning word of a Receive Record area in Normal memory (words 0 to 31 may not be used).

CONSTRUCTING MESSAGES: The LRBR instruction designates the location in memory from which data is to be transferred to the Transmit buffer or Send Record area. A parameter of 1 to "N" selects the beginning word.

The execution of LRBR sets the Receive Character Pointer (RCP) to one (1), which is the first character position in the starting word specified by LRBR. The LRBR value is retained until another LRBR instruction is executed, which would be the case when more than one Receive Record area were to be used.

11.5.02 Set Receive Character Pointer

Op Code	A	B
RCP	1:255	

The Receive Character Pointer specifies the starting character position relative to the starting word location (LRBR) FROM which data is to be transferred during execution of certain character transfer instructions. This pointer keeps track of the current character position as data is unpacked from the Receive Buffer or a Receive Record area, or as data is transferred from a location in memory to the Transmit buffer or Send Record area. The RCP instruction sets the pointer at the character position specified in the "A" parameter relative to the last LRBR word location. However, as character transfer instructions are executed that affect the LRBR location, the pointer is automatically incremented for each character so transferred. If the RCP is incremented past 255, the Overflow Test Flag is set.

Since the RCP is automatically set to one (1) by an LRBR instruction, and since it is incremented by each character transferred, the RCP instruction would normally be used only when the transfer of data is to begin from a known character position in the middle of a message.

11.5.03 Increment Receive Character Pointer

Op Code	A	B
IRCP	0:255	

The IRCP instruction increments the Receive Character Pointer by the number of character positions designated in the "A" field, or until the next field identifier code (refer to 11.10) is reached. The Pointer is incremented for the field identifier code also. This permits bypassing a data "field" in a message containing variable field lengths when that information is not needed in the processing of the message by the TC 500 user program.

11.5.04 Load Keyboard Base Register

Op Code	A	B
LKBR	0 or 1:N	

The LKBR instruction specifies the starting word location TO which data is to be transferred during the execution of certain character transfer instructions described in 11.6 and 11.7.

CONSTRUCTING MESSAGES: The LKBR instruction designates the starting word of the Transmit Buffer or Send Record area TO which data is to be transferred in the preparation of a message. A parameter of "0" designates the first word of the Transmit Buffer in Data Communication memory; a parameter of 32 to "N" would be used to select the beginning word of a Send Record area in Normal memory (words 0 to 31 may not be used).

MESSAGES RECEIVED: The LKBR instruction designates the starting word in Normal memory TO which alphanumeric data is to be transferred for temporary or permanent storage. A parameter of 1 to "N" selects the beginning word.

The execution of LKBR automatically sets the Send Character Pointer (SCP) to one (1), which is the first character position in the starting word specified by LKBR. The LKBR value is retained until another LKBR instruction is executed, which would be the case when more than one Send Record Area were to be used.

11.5.05 Set Send Character Pointer

Op Code	A	B
SCP	1:255	

The Send Character Pointer specifies the starting character position relative to the starting word location (LKBR) TO which data is to be transferred during execution of certain character transfer instructions. This pointer keeps track of the current character position as data is assembled in the Transmit Buffer or Send Record area, or as data is transferred TO a location in memory for storage from the Receive Buffer or Receive Record area. The SCP instruction sets the pointer at the character position specified by the "A" parameter relative to the last LKBR word location; however, as character transfer instructions are executed that affect the LKBR location, the pointer is automatically incremented for each character so transferred.

Since the SCP is automatically set to one (1) by an LKBR instruction, and since it is incremented by each character transferred, the SCP instruction would normally be used only when constructing messages with fixed field lengths (this would require that the buffer or record area be cleared at the start of each message to insure that any intervening unused positions did not contain unwanted data from a prior message).

11.6 INSTRUCTIONS TO PROCESS MESSAGES RECEIVED

The following instructions are designed to permit the unpacking and processing of messages received in groups of characters that may have no relation to word boundaries or word capacity (8 characters). Thus, they are character transfer instructions that refer to the LRBR, RCP, LKBR, and SCP register values as necessary during their execution. Also, an instruction is provided to transfer the Receive Buffer contents to Normal Memory.

11.6.01 Transfer Receive Buffer

Op Code	A	B
TRB	1:N	

The TRB instruction transfers the contents of the Data Communication Receive Buffer to the Normal memory Receive Record area (32 words or one track) specified by the "A" parameter. A parameter of "0" is not permissible. All 256 characters of the Receive Buffer are transferred to the Normal memory Receive Record area, destroying and replacing any prior contents of that working area. For example, a parameter value of 1 would mean that the transfer was into Normal memory words 32 through 63. This instruction permits the use of one or several Receive Record areas in Normal memory.

11.6.02 Transfer to Accumulator as Numeric:

Op Code	A	B
TRBA	0:16	

The TRBA instruction transfers the number of characters specified in the "A" field from the Receive Buffer, or working record area, to the Accumulator as Numeric digits. The buffer or Receive Record area is the one specified by the last LRBR instruction, and the beginning character is determined by the current position of the RCP. The TRBA instruction is terminated by the transfer of the number of characters specified or by a field identifier code, whichever comes first. The field identifier code sets a specified flag pattern (see 11.10). The RCP is incremented for each character transferred and for the field identifier code (which is not transferred into the Accumulator). The Overflow flag will be set if the RCP is incremented past 255 and the instruction will be terminated; otherwise, the Overflow flag is reset.

Although alpha numerals occupy 2 digit positions (8 bits) for the character in either the Receive Buffer or Receive Record area, the TRBA instruction places then in the Accumulator as numeric digits (4 bits). Thus, up to 16 buffer characters can be transferred to the Accumulator as 16 digits (any data required for computational purposes must be limited to 15 digits).

Valid codes accepted by TRBA are any codes from column 3 of the USASCII table. These include the numerals 0 to 9 and : ; > = < ? In addition, the minus (-) and plus (+) codes and any field identifier codes from columns 0 and 1 are valid. When used in a numeric field, the minus or plus code may be any character in the field. After first use in a given numeric field, subsequent plus or minus codes are invalid. The minus code will set the sign flag in the accumulator; the plus code will reset the sign flag. The minus or plus code will not be counted as one of the characters transferred as specified by the parameter field, however, the RCP will be incremented for this character. The field identifier codes are not transferred to the Accumulator but do terminate the TRBA instruction. The characters : ; > = < and ? are transferred to the accumulator as hexadecimal digits (undigits) with binary values of 10, 11, 12, 13, 14 and 15 respectively (values are designated by A, B, C, D, E, and F).

Other characters will be considered as invalid, will cause the "S" flag of the Accumulator to be set, will count as a code transferred, but the instruction will not be terminated.

11.6.03 Transfer Alpha

Op Code	A	B
TRF	0:255	

The TRF instruction transfers alphanumeric characters from the memory location specified by the LRBR instruction beginning at the current RCP position, into another memory location as specified by the LKBR instruction beginning at the current SCP position. The number of characters to be transferred is specified by the parameter field of the TRF instruction; the instruction is terminated by the transfer of the exact number of characters specified or by a field identifier code. Upon termination of the instruction, an End of Alpha code is automatically inserted into the next character position of memory; however, the SCP is not incremented for that code.

The TRF instruction provides the ability to move data from one character and word location to another character and word location in memory; thus, it is used not only for unpacking data from messages received, but also to prepare messages for transmission. The LRBR and RCP identify the location from which the data is being moved; the LKBR and SCP identify the location to which the data is being moved.

The RCP and SCP are incremented for each character transferred; the RCP will also be incremented by a field identifier code if one is present. The Overflow flag will be set if either pointer is incremented past 255, and the instruction will be terminated; otherwise, the Overflow flag will be reset. The field identifier code sets a specific flag pattern (see 11.10). All USASCII codes are valid.

11.6.04 Print Alpha From Receive Buffer

Op Code	A	B
PAB	0:150	

The PAB instruction prints the number of characters specified in the "A" field from the Receive Buffer or Receive Record area specified by the last LRBR instruction, beginning at the character designated by the current position of the RCP. The instruction is terminated after printing the specified number of characters or by a field identifier code, whichever comes first. The field identifier code sets a specified flag pattern (see 11.10). The RCP is incremented for each character printed and also for the field identifier code.

The Overflow flag will be set if the RCP is incremented beyond 255, and the instruction will be terminated. The system will return to the Ready Mode if printing attempts to extend beyond position 150 in the forms transport.

11.6.05 Programing Steps to Receive a Message

A. Unpacking a message direct from Data Communication Receive buffer:

1. Test to determine if the Message Received flag is Set (Reader Flag 2): SK, SKE, EX, EXE R 2.

2. Select Data Communication Receive buffer:

   LRBR 0

3. Unpack message: IRCP, TRBA, TRF (LKBR, SCP), PAB
4. Reset Message Received flag (R2) to allow next message: RST R 2

B. Unpack message from a Normal memory Receive Record area:

1. Test to determine if the Message Received flag is Set (Reader Flag 2): SK, SKE, EX, EXE R 2.
2. Transfer message from Data Communication Receive buffer to Normal memory Record area: TRB 10
3. Reset Message Received flag (R2) to allow next message: RST R 2
4. Select Normal memory Receive Record area:

   LRBR 320
5. Unpack message: IRCP, TRBA, TRF (LKBR, SCP), PAB

11.7 INSTRUCTIONS TO PREPARE MESSAGES FOR TRANSMISSION

The following instructions are designed to enable constructing a message that contains only essential data as required at the data center. Thus, they are character transfer instructions that are not restricted to word boundaries or word capacity; they refer to the LRBR, RCP, LKBR and SCP register values as necessary during their execution. Also, an instruction is provided to transfer the contents of a Send Record area to the Transmit buffer. Once the message is assembled and transferred to the Transmit buffer, the user program must set the Transmit Ready flag (R3) to permit transmission when the next POLL is received from the data center.

If any of these instructions are used to transfer data into the Transmit Buffer and the Transmit Ready Flag has not yet been reset by the Data Communication Processor, the instruction is held up from being executed. It is expected that the user program will interrogate the Transmit Ready flag before moving any data directly into the Transmit Buffer.

11.7.01 Transfer Send Record Area

Op Code	A	B
TSB	1:N	

A message may be prepared for transmission in a Normal memory Send Record area and then be moved in completed form to the Data Communication Transmit Buffer for actual transmission. The TSB instruction transfers the entire contents of the Normal memory Send Record area (32 words or one track) specified by the "A" field to the Data Communication Transmit buffer. A parameter of "0" is not permissable since track zero cannot be used for a Send Record area. All 255 characters of the selected Send Record area are moved to the Transmit Buffer. The ETX character is automatically placed after the last actual character of text during transmission, and all subsequent positions are ignored.

11.7.02 Transfer Accumulator to Send Record Area

Op Code	A
TRAB	0:15

The TRAB instruction transfers up to 15 numeric digits (of 4 bits each) from the Accumulator into the Transmit Buffer or Send Record area specified by the LKBR instruction as alpha numerals (7 bits each) any Accumulator flags that are set are ignored. The "A" field designates the number of digits to be transferred, which are always the low order digit positions of the Accumulator; the digits enter the Transmit Buffer or Send Record area beginning at the current SCP position. A parameter of "0" causes the transfer of only significant digits (no leading zeros); a parameter of "1" to "15" causes the transfer of the exact number of low order digits specified (including any leading zeros). Any other data in the Accumulator is ignored. The SCP is incremented by each digit transferred into a character position of the Send Record area, or Transmit buffer.

When it is necessary to include any Accumulator flag settings in a message, the presence of a set Accumulator flag would be tested using a Skip or Execute instruction, and then an appropraite code inserted into the message using a TRCB instruction (see 11.7.04).

11.7.03 Transfer Alpha

Op Code	A	B
TRF	0:255	

Refer to 11.6.03 for a complete discussion of this instruction and how it applies to the preparation of messages.

11.7.04 Transfer Character

Op Code	A	B
TRCB	0:15	0:15

The TRCB instruction transfers the USACII code designated by the decimal value in the "A" and "B" fields into the Send Record area or Transmit Buffer specified by the last LKBR instruction at the character position designated by the SCP. The SCP is incremented one character position. The "B" field represents the decimal value of the code's lower 4 bits; the "A" field represents the decimal value of the upper 4 bits, however it is not necessary to include the parity bit (b_8) in this value as it is generated automatically (if it is included, it is ignored). The proper parameter value for each USASCII code can be obtained by using the code's column and row number in the USASCII table. The column number is placed in the "A" field; the row number is placed in the B field. For example, the code "asterisk" (*) is in column 2, row 10; this would be programed with "TRCB 2 10".

11.7.05 Type to Memory

Op Code	A	B
TKM	0:150	

The TKM instruction permits entering data directly into the Transmit Buffer or a Send Record area from the typewriter keyboard. The "A" field specifies the maximum number of characters that may be typed. Entry of data begins at the word in memory specified by the last LKBR instruction, starting at the current SCP position. The SCP is incremented for each character entered. The termination of the TKM by an OCK or PK causes an End of Alpha code to be placed in memory. However, the SCP is not incremented by this End of Alpha character. The TRCB instruction may then be used to transfer an appropriate field identifier, if required, into the character position, overlaying the End of Alpha character.

The use of the backspace key with the TKM instruction will cause the SCP to be decremented for each backspace. However, the SCP will not decrement beyond the point at which the TKM began.

11.7.06 Programing Steps to Transmit a Message

A. Message is constructed directly in the Data Communication Transmit buffer:

1. Test to determine if Transmit Ready Flag (R3) is reset: SK, SKE, EX, EXE R 3
2. Select Data Communication Transmit buffer: LKBR 0
3. Construct message: TRAB, TKM, TRCB, TRF (LRBR, RCP)
4. Set Transmit Ready Flag: SET R 3

B. Message is constructed in a Normal memory Send Record Area:

1. Select Normal memory Record area: LKBR 320
2. Construct message: TRAB, TKM, TRCB, TRF (LKBR, RCP)
3. Test to determine if Reader Flag 3 is reset: SK, SKE, EX, EXE R 3
4. Transfer message to Data Communication Transmit buffer: TSB 10
5. Set Transmit Ready Flag: SET R 3

11.8 OTHER DATA COMMUNICATION INSTRUCTIONS

The ability to change the TC 500 address and to alter, if necessary, the Send and Receive transmission numbers, is provided. In addition, the TC 500 may shut itself off after completion of transmission or upon direction from the data center.

Each character in an address or transmission number is a USASCII code of 7 bits. When transferred into the Accumulator, as described in the following instructions, the character occupies 2 digit positions of the Accumulator.

11.8.01 Retrieve Send Address

Op Code	A	B
RSA		

This instruction transfers the two-character send machine address from the Send Address Register in the Data Communications Processor into the four (4) most significant digit positions of the Accumulator (each character will occupy 2 digit positions). The balance of the Accumulator will contain zeros. These two characters may be any characters from columns 2 through 6 of the USASCII set (except "circumflex" and "underline"). With a range of 78 different characters in each of two positions, the total machine address range potential would be 6,084 different "numbers".

11.8.02 Retrieve Receive Address

Op Code	A	B
RRA		

This instruction functions in exactly the same fashion as RSA except that it will transfer the machine address from the Receive Address Register in the Data Communications Processor into the four (4) most significant digit positions of the Accumulator. The balance of the accumulator will contain zeros.

Generally, the Receive and Send Machine Addresses are alike. However, a condition can exist where they would have to be different, though only for a short time.

By changing its address, a TC 500 can cause Selection of it to be ignored. This can permit interrupting a series of Selects from the data center; the data center would then interpret the ignore as a request for a Poll to the TC 500.

11.8.03 Load Send Address Register

Op Code	A	B
LSA		

This instruction transfers the contents of the Accumulator into the Send Machine Address Register in the Data Communications Processor. Only the four (4) most significant digit positions of the accumulator may contain significant characters (2 digits per character). The balance of the Accumulator must contain zeros.

11.8.04 Load Receive Address Register

Op Code	A	B
LRA		

This instruction transfers the contents of the Accumulator into the Receive Machine Address Register in the Data Communications Processor. Only the four (4) most significant digit positions of the Accumulator may contain significant characters. The balance of the Accumulator must contain zeros.

11.8.05 Retrieve Expected Transmission Number

Op Code	A	B
RTN		

This instruction transfers the 1, 2 or 3 USASCII Numeric Character "Expected Transmission Number" from its appropriate Register into the 2, 4 or 6 most significant digit positions of the Accumulator. The balance of the Accumulator will contain zeros.

11.8.06 Retrieve Header Transmission Number

Op Code	A	B
RTH		

This instruction transfers the entire word containing the header portion of the transmission from Data Communication memory into the 14 low order digit positions of the Accumulator. The balance of the Accumulator will contain zeros. This word will contain from 5 to 7 USASCII characters as the header portion, and will have the following format in the accumulator:

digit positions:	15	14	13 12	11 10	9 8	7 6 5 4 3 2	1 0
	0	0	SOH	AD1	AD2	Trans. No.	STX

The transmission number can be 1, 2 or 3 characters. The above example shows a 3-character transmission number. A fewer number of characters would result in the other characters being closer together and hence occupying less digit positions of the Accumulator.

11.8.07 Load Expected Transmission Number Register

Op Code	A	B
LTN		

This instruction transfers the contents of the Accumulator into the Expected Transmission Number Register for messages received. Only the 2, 4 or 6 most significant digit positions of the Accumulator may have significant characters. The balance of the accumulator must contain zeros.

11.8.08 Retrieve Send Transmission Number

Op Code	A	B
RSN		

This instruction transfers the 1, 2 or 3 digit USASCII Numeric Character Send Transmission Number from its appropriate Register into the 2, 4 or 6 most significant digit positions of the Accumulator. The balance of the Accumulator will contain zeros.

11.8.09 Load Send Transmission Number Register

Op Code	A	B
LSN		

This instruction transfers the contents of the Accumulator into the Send Transmission Number Register in Data Communication memory. Only the 2, 4 or 6 most significant digit positions of the Accumulator may contain significant characters. The balance of the Accumulator must contain zeros.

11.8.10 Retrieve Character Pointer Register

Op Code	A	B
RPR		

This instruction transfers the contents of the Character Pointer Register into the Accumulator. The values are represented by hexedecimal digits. The format of the Accumulator will be as follows:

	(word)		(BLK)	(BLK)(word)	(word)		(BLK)	(BLK)(word)
Digit Positions:	15 14	13 12	11	10 9 8	7 6	5 4	3	2 1 0
		RCP		BASE LRBR		SCP		BASE LKBR
	WORKING LRBR		WORKING LRBR		WORKING LKBR		WORKING LKBR	

11.8.11 Load Character Pointer Register

Op Code	A	B
LPR		

This instruction transfers the contents of the Accumulator into the Character Pointer Register.

11.8.12 Power Off

Op Code	A	B
OFF		

The OFF instruction provides the ability for the TC 500 to turn itself off by causing the power to the entire system to be turned off. This instruction permits the data center to notify a TC 500 to shut down, by sending a reserved character or other unique data (selected by user) to it. Upon testing and recognizing this character, the TC 500 would branch to the instruction OFF as a part of the user program.

11.8.13 Retrieve Polled Flags

Op Code	A	B
RPF		

The RPF instruction transfers the four (4) bits of this Data Communication Processor Polled Flag register into the flags position of the Accumulator (refer to 11.9.04). The balance of the Accumulator will contain zeros. This permits testing the status of the flags using the Skip and Execute instructions.

11.8.14 Load Polled Flags Register

Op Code	A	B
LPF		

The LPF instruction transfers the contents of the Accumulator flags position into the Polled Flags register of the Data Communication Processor. Only the flag positions of the Accumulator may contain significant bits. The balance of the Accumulator must contain zeros. This permits setting these flags initially in a program to a known value, in order to properly evaluate any subsequent testing of the flags (refer to 11.9.04).

11.9 DATA COMMUNICATION FLAGS AND FLAG INSTRUCTIONS

Two flags are provided which have indicator lights associated with them for communication with the operator. In addition, several other flags are available to interrogate the status of the Data Communication Processor through program control.

11.9.01 Message Received and Transmit Ready Flags

The Message Received flag and the Transmit Ready flag are part of the Reader Flags group (R2 and R3 respectively). They may be set or reset by using the SET/RESET/LOAD/CHANGE instructions discussed in section 6; or they may be interrogated by using the SKIP/EXECUTE instructions discussed in section 9. Each of these flags has an indicator light associated with it. When either flag is set, its indicator light is turned on; when either flag is reset, its indicator light is turned off.

The Message Received flag (R2) is automatically set and its indicator light turned on by the successful receipt of a message in the Data Communication Receive buffer. No other messages will be received while this flag is set. The user program must RESET the flag after final use has been made of the message data.

The Transmit Ready flag (R3) is SET and its indicator light turned on by the user program, after having placed a message in the Data Communication Transmit buffer. The successful transmission of the message automatically causes the flag to be reset and the light to be turned off. The user program is held up if it attempts to place another message in the Transmit buffer while the Transmit Ready flag is still set.

11.9.02 Keyboard Buffer Empty Flag

A Keyboard Buffer Empty flag is provided as a means for the operator to interrupt an automatic operation of rather long duration, such as the receipt of report data from the data center. The Keyboard Buffer Empty flag remains Set as long as there are less than 3 codes in the buffer; by indexing 3 keyboard keys, the operator causes the Keyboard Buffer Empty flag to be Reset. The user program may periodically test this flag to see if the operator is trying to interrupt the operation, and if so, branch to an appropriate routine for preparing a message, etc.

The Keyboard Buffer Empty Flag is in the Buffer Flags group "B", and is number 3 of that group. It may be interrogated by the SKIP/EXECUTE instructions, in the following form:

Op Code	A	B	C
SK	B	3	1:4
EX	B	3	1:4

Keyboard Buffer flags 1, 2, and 4 are not used by the programmer.

11.9.03 Data Communications Processor Flag Register

The "D" Flag Group consists of 4 flags provided in the Data Communications Processor for purposes of Maintenance Test Routines. However, these flags may be tested by using SKIP/EXECUTE instructions:

1. Transmission Failure Flag: This flag is reset as long as the transmission number of the received message agrees with the expected transmission number. It is set if the actual transmission number does not agree with the expected transmission number; however, the TC 500 will Acknowledge (ACK) the data center if the message is correct in other respects.
2. Message Received Flag: This flag is the counterpart of and causes the setting of the Main Memory Message Received Flag.
3. Transmit Ready Flag: This flag is the counterpart of and causes the resetting of the Transmit Ready Flag in Main Memory.
4. A Micro Flag (of no use to the programmer).

Using the Skip and Execute instructions, it is possible to interrogate the Message Received flag and the Transmit Ready flag directly in the Data Communications Processor.

11.9.04 Data Communications Processor Polled Flags Register

Another register is provided in the Data Communications Processor that may be considered as a flag register; however, it is not possible to interrogate it directly with the regular Skip or Execute Flag instructions. Instead, the contents of this register would be transferred into the flag position of the Accumulator with the RPF instruction described in 11.5.13. When in the Accumulator flag position, the flags may be interrogated using the Skip and Execute on Accumulator Flag instructions.

Flags C (No. 2) and M (No. 3) may be of use in programing, and provide the following information:

C (2). This TC 500 has been Polled or Selected: This flag is set if polling or selecting has been directed to this remote.

M (3). Some TC 500 on the line has been polled or selected: This flag is set if there has been any Polling or Selecting activity on this line.

Flags 1 and 4 are not of use in programing.

In order to utilize these flags properly, the user program must initially reset these flags (load the Accumulator flags as reset using the LPF instruction described in 11.8.14). Then, at some later point in the program, provide for transferring this register into the Accumulator for testing this activity with the regular Skip and Execute instructions.

11.10 FIELD IDENTIFIER CODES

The following USASCII codes, if they appear as part of a text message will cause certain of the Data Communications Macro instructions to be terminated and will set the flag pattern indicated in the following table.

No Flags Set	Y Flags Set*		K Flags Set*		Test Flags Set	
		3 2 1 4		3 2 1 4		U I L Ø
NUL	SOH	0 0 0 1	DC1	0 0 0 1	ETX	0 0 0 1
	STX	0 0 1 0	DC2	0 0 1 0		
			DC3	0 0 1 1		
			DC4	0 1 0 0		
	ENQ	0 1 0 1	NAK	0 1 0 1		
	ACK	0 1 1 0	SYN	0 1 1 0		
	BEL	0 1 1 1	ETB	0 1 1 1		
	BS	1 0 0 0	CAN	1 0 0 0		
	HT	1 0 0 1	EM	1 0 0 1		
	LF	1 0 1 0	SUB	1 0 1 0		
	VT	1 0 1 1	ESC	1 0 1 1		
	FF	1 1 0 0	FS	1 1 0 0		
	CR	1 1 0 1	GS	1 1 0 1		
	SO	1 1 1 0	RS	1 1 1 0		
	SI	1 1 1 1	US	1 1 1 1		

* Y and K flags designated are set if "1" and reset if "0"

It is generally agreed that many of the above USASCII codes should never appear in a text. EOT is specifically filtered out by the Data Communications Processor. NUL does serve as a field identifier but, as indicated in the chart above, it terminates the instruction but does not set any flags; neither does it reset any previous flags. It merely terminates the instruction. ETX has special significance in that when ETX is detected during a transfer instruction, the Overflow flag will be set and the instruction terminated.

Many of the above codes should not appear in a text but if they do, the TC 500 will accept them and set the flag pattern indicated above. The chart below shows the codes that normally can be a part of a text message. Some codes from the above chart may be excluded by Central Processors or by Line

Adapters. Applicational programing should consider these as termination and flag setting codes rather than the entire range listed in the above table.

No flags	Y flags	K flags	Test Flags
	3 2 1 4	3 2 1 4	U I L Ø
NUL			ETX 0 0 0 1
		DC1 0 0 0 1	
		DC2 0 0 1 0	
		DC3 0 0 1 1	
		DC4 0 1 0 0	
	BS 1 0 0 0		
	HT 1 0 0 1		
	LF 1 0 1 0		
	VT 1 0 1 1		
	FF 1 1 0 0	FS 1 1 0 0	
		GS 1 1 0 1	

12. INPUT WITH PUNCHED PAPER TAPE/EDGE PUNCHED CARD READER

Instructions are provided to read punched paper tape or edge punched cards into a TC 500, using a Burroughs style A 581 Paper Tape/Edge Card Reader as the input adjunct. All subsequent reference to "paper tape" applies both to punched paper tape and to edge punched cards, unless indicated otherwise.

Tape reading is serial, one character at a time, at a speed up to 40 characters per second when no printing accompanies it. When reading paper tape and printing on the TC 500 printer, the reading speed is up to 20 characters per second; when reading and punching only (no printing), reading speed is up to 40 cps.

The internal character code of the TC 500 is USASCII, however, any 5, 6, 7, or 8 channel paper tape code can be read and interpreted by the TC 500 by utilizing a Table of Input Code Assignments for conversion of the paper tape code into the internal USASCII code. The functional codes in a code set may be used as field identifier codes to terminate tape reading and set flag patterns, or may be ignored (refer to the Table of Input Code Assignments section 12.4). The scheme of character parity checking for a particular code set is also a function of the table of code assignments. Firmware for 5 channel code is different than that for 6, 7, or 8 channel "table look-up" firmware or for USASCII No Table firmware.

12.1 PAPER TAPE READER INSTRUCTIONS

The Paper Tape Reader instructions are designed to function in two ways: When proper tape reading conditions exist, reading of tape will occur according to the specifications of the instruction. If all of the necessary tape reading conditions do not exist, the reader is "not operable" and the function of the instruction can revert to its counterpart keyboard input instruction, permitting manual input by the operator. Thus, a program may depend primarily on reader input but be able to substitute keyboard input when needed. When a reader condition occurs, the keyboard buffer is cleared in anticipation of reverting the instruction to its keyboard counterpart. This prevents misoperation from a subsequent normal keyboard input indexed before execution of the reader instruction was attempted.

Therefore, it is important that the reader instruction be reached before a manual entry is made in place of it as any keyboard input prior to the reader instruction is lost.

The mnemonic representations of the read instructions are the same as selected keyboard instructions with the addition of a prefix letter "R".

Instructions that involve punching paper tape along with reading of paper tape will inhibit the punch part of the instruction if the tape perforator is turned off. In addition, the Punch Off Indicator light is turned on and Punch Off Flag is set (refer to section 13).

12.1.01 Reading Alphanumeric Data and Printing

	Op Code	A	B
Read Alpha and Print	RTK	0:150	

The RTK instruction reads from tape and prints the number of alphanumeric character specified by the A field. The instruction is terminated after reading the specified number of characters, or upon reading a field identifier code. The flag patterns to be set by the field identifier codes are determined by the table of input code assignments.

The RTK instruction can revert to a TK instruction if the tape reader is not operable.

12.1.02 Reading Alphanumeric Data, Printing and Punching

	Op Code	A	B
Read Alpha, Print and Punch	RXTK	0:150	

The RXTK instruction reads from tape and prints the number of alphanumeric characters specified by the A field. Tape punching occurs simultaneously with printing at 20 codes per second. The instruction is terminated after reading the specified number of characters, or upon reading a field identifier code (refer to table of input code assignments).

The RXTK instruction can revert to an XTK instruction if the tape reader is not operable, to an RTK instruction if the tape perforator is turned off, or to a TK instruction if neither the reader nor perforator is operable.

12.1.03 Reading Alphanumeric Data, Printing, and Entering into Memory

	Op Code	A	B
Read Alpha into Memory and Print	RTKM	0:150	

The RTKM instruction reads from tape into memory and prints the number of alphanumeric characters specified by the A field. This instruction requires the prior execution of an LKBR instruction to specify the starting word location in memory. The word of entry is incremented to the next higher word after each eight characters have been read. The instruction is terminated after reading the specified number of characters, or upon reading a field identifier code (refer to table of input code assignments).

The RTKM instruction can revert to a TKM instruction if the tape reader is not operable.

12.1.04 Reading Alphanumeric Data, Printing, Punching, and Entering into Memory

	Op Code	A	B
Read Alpha into Memory, Print and Punch	RXTKM	0:150	

The RXTKM instruction is the same as the RTKM instruction, except that tape punching occurs simultaneously with printing at 20 codes per second.

The RXTKM instruction can revert to an XTKM instruction if the tape reader is not operable, to an RTKM instruction if the tape perforator is turned off, or to a TKM instruction if neither the reader nor perforator is operable.

12.1.05 Reading Alphanumeric Data into Memory, No Printing

	Op Code	A	B
Read Alpha into Memory, Non-print	REAM	0:150	

The REAM instruction reads from tape into memory, without printing, the number of alphanumeric characters specified by the A field. This instruction requires the prior execution of an LKBR instruction to specify the starting word location in memory. The word of entry is incremented to the next higher word after each eight characters have been read. The instruction is terminated after reading the specified number of characters, or upon reading a field identifier code (refer to table of input code assignments). Reading is up to 40 characters per second.

The REAM instruction can revert to an EAM instruction if the tape reader is not operable.

12.1.06 Reading Alphanumeric Data into Memory, Punching, but no Printing

	Op Code	A	B
Read Alpha into Memory and Punch, Non-print	RXEAM	0:150	

The RXEAM instruction is the same as the REAM instruction, except that punching occurs simultaneously with reading.

The RXEAM instruction can revert to an XEAM instruction if the tape reader is not operable, to an REAM instruction if the tape perforator is turned off, or to an EAM instruction if neither the reader nor perforator is operable.

12.1.07 Valid Codes for Read Alpha Instructions

The Read Alpha instructions described in the above paragraphs recognize all codes as valid. Any input code, whose parity is not in agreement with the table of input code assignments, will be considered as a parity error and will turn on the Invalid Code Indicator and Set Reader Flag R1 (see 12.3.01) on Invalid Code Indication.

All codes, except those which are coded in the table of input code assignments to be ignored, will be counted as a code read. Thus, in programing, where a Read Alpha (RTK, etc.) is to be terminated by a field identifier code, the number of graphic character codes read and printed or stored in memory would be one less than the parameter field. Where a field identifier code is not used, the number of graphic character codes read will be equal to the parameter field.

12.1.08 Reading Numeric Data into the Accumulator

	Op Code	A	B
Read Numeric into Accumulator	RNK	0:15	0:15

The RNK instruction reads from tape the total number of characters specified by the A and B fields into the Accumulator. The characters are treated as numeric digits and enter the Accumulator in the low order digit positions. The instruction is terminated after the total number of characters specified have been read (the A field plus the B field), or upon reading a field identifier code (refer to table of input code assignments). Reading is up to 40 codes per second; printing does not occur with this instruction.

The RNK instruction does not consider the A field as whole numbers and the B field as decimal digits, but rather acts on the sum of the two fields; the separation of the fields into whole and fractional digits is provided only to permit keyboard entry when the reader is not operable. Thus, to read a number into the Accumulator, the tape must contain as many codes as the total number of digits specified by the instruction. This requires that preceding zeros up to the maximum field size be included in the tape; or if preceding zeros are not included in the tape, a field identifier code must follow the number in the tape to halt reading. Also, to read a decimal fraction into the Accumulator, the tape must contain as many digits or zeros to the right of the decimal as the maximum number of decimal places allowed. This is because the codes enter the low order positions.

For Example: Read 12.25 into the Accumulator, allow for three decimal places, with instruction "RNK 6 3"

Tape must contain: 000012250
(with no field identifier code)

or: 12250FS
("FS" denotes a field identifier code)

Note that field identifier codes permit the tape to contain only significant digits in the whole number even though the instruction accommodates larger numbers.

The RNK instruction reverts to an NKRCM instruction if the tape reader is not operable. This requires the parameters to specify whole and fractional digits.

VALID CODES FOR READING INTO THE ACCUMULATOR: The RNK instruction accepts all codes as valid, except in the case of parity errors (refer to 12.3.01). The codes for the numerals 0 to 9 are read into the Accumulator as digits. The other 54 graphic characters may appear in a numeric field. They will not count as a digit read in satisfying the parameter. However, they will set a flag pattern in the Accumulator Flag position. The pattern for each such character is shown in the table below. Field identifier codes do count as a code read in the parameter. Any input code that is coded in the table of input code assignments to be ignored will not be counted as a code read.

In programing, a Read Numeric (RNK) instruction that is to be terminated by a field identifier code may read up to a maximum of 14 digits as the field identifier code would count as the 15th code read. If no field identifier code is used, an RNK instruction may read up to 15 numeric digits.

Codes that set Accumulator Flags during Read Numeric (when code is contained in table of code assignments) "1" indicates flags that are set; "0" indicates flags that are reset

Character					Accumulator Flags			
					M	C	S	–
Space		@	P		0	0	0	0
!		A	Q		0	0	0	1
”		B	R		0	0	1	0
#		C	S		0	0	1	1
$		D	T		0	1	0	0
%		E	U		0	1	0	1
&		F	V		0	1	1	0
’		G	W		0	1	1	1
(		H	X		1	0	0	0
)		I	Y		1	0	0	1
*	:	J	Z		1	0	1	0
+	;	K	[(3/4)		1	0	1	1
,	< (½)	L	\ (¢)		1	1	0	0
-	=	M	](CR)		1	1	0	1
.	> (¼)	N	^ (°)	~ (<>)	1	1	1	0
/	?	O	–		1	1	1	1

12.1.09 Opening the Media Clamp

	Op Code	A	B
Release Media Clamp	REL		

This instruction will cause the media clamp for paper tape or edge punched cards to open, thus halting any further reading until the operator places new media in the reader. This instruction is most useful when using edge punched cards, to release the card after necessary information has been read from it, and to prevent any additional information on the card from enabling the read instruction for the next entry.

12.2 LOADING PROGRAMS WITH THE PAPER TAPE READER

The paper tape reader may be used to load programs into the TC 500 in place of the Memory Loader device in the front of the console. This is accomplished with a "Reader Load" Utility routine. This Utility routine would be stored in the Utility area of Control Memory and would be activated by PK A3 (Utility) while in the Ready Mode. The use of this program in the Utility area of memory does not preclude the use of any other Utility routine, but only one such routine would reside in the Utility area of memory at one time. The "Reader Load" routine would be reloaded in the Utility area after use of the other routine(s).

The "Load Memory" Utility routine and any other Utility routines would be entered into the Utility area of memory by using the regular Memory Loader device in the front of the TC 500.

12.3 INPUT INDICATOR LIGHTS AND FLAGS

Two indicator lights and two Reader flags are provided to enable program and operator control over the paper tape reader:

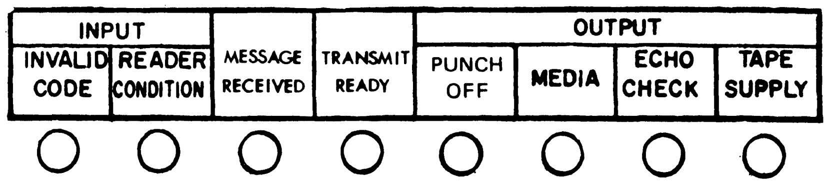


Fig. 12-1 Input Indicator Lights

12.3.01 Invalid Code Indicator Light (Input Indicator 1)

This indicator light will be turned on and Reader Flag R4 set when an invalid tape code is read. Code validity is determined by the stored Table of Input Code Assignments. Reading is not halted on the invalid tape code; the code will count as a code read and will be processed. The next read instruction will reset the flag and turn off the indicator.

A code is invalid if it has a parity error. Tape code parity is a function of the table of Input code assignments; therefore, any code can be forced into a parity error situation by means of the value placed in its position in the input table. Thus, aside from natural parity errors, any codes so desired can be rendered invalid to this system causing the flag to be set and the indicator light to be turned on. Note however, that rather than be treated as invalid, a tape code may be ignored for use in the system once it has been parity checked; again, this is a function of the value placed in the table for that code.

12.3.02 Reader Condition Indicator Light (Input Indicator 2)

The Reader Condition Indicator advises the operator whether the paper tape reader is operable and is active when the program reaches a tape read instruction. The following conditions must be satisfied for the paper tape reader to be operable:

a. The Paper Tape Reader must be connected to the system, and must be turned on.
b. Media (paper tape or an edge punched card) must be positioned in the reader.
c. The media clamp must be closed.
d. The Reader light (photoelectric device) must be illuminated.

When the program reaches a tape read instruction and all of these conditions are not satisfied, the Reader Condition Indicator Light (Input Indicator 2) is turned on, Reader flag 1 is set, the keyboard buffer is cleared, and the instruction is held up from execution pending operator action. This action is based on one of two conditions:

a. The reader is intended to be used: In this case, the operator may correct the condition(s) (place media in the reader, close the media clamp, etc.) and depress the Read Key. This re-initiates the read instruction, causing reading of the tape in accordance with the instruction; the indicator is turned off and the flag is reset.
b. The reader is not intended to be used: In this case, the operator may make a manual entry through the keyboard. As this action is started, the instruction reverts to its keyboard instruction counterpart, and is executed accordingly (refer to section 2 for details of keyboard instructions). Once the keyboard entry has begun and prior to termination with an OCK or PK, the use of the Reset Key will re-initiate the Tape read instruction.

Once the operator has taken either course of action above, the indicator light is turned off, and reader flag 1 is reset. Note that since the buffer was cleared, any following manual entries that may have been indexed ahead of the reader instruction would require re-indexing.

12.3.03 Flag Instructions (Load, Set, Reset, Change)

The execution of a Load, Set, Reset or Change Flag instruction involving the Reader "R" flags will also cause the indicators to either be turned on or off depending on the instruction used.

12.3.04 Program Keys

Program Keys (PK's) that have been enabled prior to a paper tape read instruction will be ignored during the tape reading even if the PK is depressed during reading. If the operator causes the tape read instruction to revert to its keyboard counterpart, use of such a PK will be recognized and will function as during any normal keyboard entry. When the instruction is terminated, either as a tape read or as a keyboard instruction, any PK's so enabled will be turned off.

12.4 TABLE OF INPUT CODE ASSIGNMENTS

A Table of Input Code Assignments provides the means by which any type of paper tape code (BCL, etc.) may be read and interpreted into the TC 500 internal code (USASCII). The table not only permits any type of code (from any 5, 6, 7, or 8 channel tape), but also enables assigning any desired character or certain functions to be interpreted from a particular code. Tables are available for such common code sets as BCL, IBM 046, Friden and 5 Channel Teletype (Baudot); however, any other code set (up to 8 channels or bits) may be incorporated.

Input tape that contains USASCII code does not require a table for conversion, but may use a table if special functions are desired from certain codes.

The conversion table, where required, is stored in the Normal (or user) area of memory and occupies up to 16 words. Each code (character) in the tape is represented by a pattern of punches in one position (or frame) which constitutes a unique configuration of "bits". As codes are read from tape, each code references its own character position in the conversion table based on its "bit" configuration. In other words, the bit configuration of the code serves as an "address" to a specific position in the table. The way in which that code is interpreted is determined by the internal code value that the programmer has placed in that position of the table. The tables available represent "standard" interpretations of characters and functional codes. The internal code representing an input code may be changed in the table to suit a user's particular need and give any desired interpretation as outlined in the following paragraphs.

12.4.01 Input Functions for 6, 7, or 8 Channel Tape Based on the Table of Code Assignments

The TC 500 interprets an input code in any one of the following ways depending on the internal code placed in its position in the table (does not apply to 5 channel code):

a. Interprets the incoming code as one of the TC 500 printable (graphic) characters when the TC 500 internal code for that character is contained in that position of the table.

b. Ignores the incoming code when the TC 500 internal code for Ignore is contained in that position of the table.

c. Interprets the incoming code as an invalid character when a forced parity error is contained in that position of the table. This turns on the Invalid Code Indicator Light and sets Reader Invalid Code flag (refer to 12.3.01).

d. Causes the incoming code to set any or all of the flags of one flag group (the Y or K flag groups). The flags can then be tested as part of the user program, to provide alternate results. Codes that set the Y or K flags also terminate the read instruction (refer to 12.4.03). Incoming codes interpreted in this manner serve as Field Identifier codes and do not provide a printable character in the TC 500.

e. Causes the incoming code to set any or all of the Accumulator (A) Flags during a Read Numeric instruction (see 12.1.08). This permits numeric data to be read as minus and/or identified uniquely (as per hundred, etc.). The flags can then be tested to cause alternate results as part of the user program. Codes that set the Accumulator Flags do not terminate the Read instruction; therefore, they can be located in any character position in the data field on the tape. They do not provide a printable character during read-in, but as in the case of the Sign flag, subsequent Print Numeric instructions can be affected.

The codes described in paragraphs d and e above may or may not correspond to those codes that are normally considered "control" or "functional" in a given code set, depending on the interpretation value given to them in the table by the programmer.

12.4.02 Firmware Subsets for the Table of Code Assignments

Specific G.P. 300 Firmware subsets are provided with paper tape input/output capability. However, the Table of Code Assignments is usually loaded into memory as part of the user program load procedure. This permits using various code sets at different times with the same user program, or permits use of a different code set with each separate user program without changing the firmware, with certain exceptions:

a. Input with any code set requiring conversion to the internal code (USASCII), with a table of code assignments, requires a firmware subset that provides "table look-up". Input with USASCII does not require "table look-up" firmware since no conversion is necessary. However, various code sets can be used as input to the same system, along with USASCII, so long as "table look-up" firmware is used and a table of code assignments is provided for each code set including USASCII.

b. Firmware for 5 channel code includes "table look-up" capability; however, it is different than firmware for 8, 7, or 6 channel code, or for USASCII (no table look-up).

12.4.03 USASCII Paper Tape Code Without Table Look-up Firmware

When USASCII is the paper tape input code, a table of code assignments is not required, and a separate Firmware subset is provided.

The following chart shows the code that represents each of the USASCII characters on tape (even parity). Each character is represented by two hexadecimal digits: the left for the upper four bits, the right for the lower four bits. As the tape is read and after parity checking, the parity bit (b8) is set to zero before the character is stored in memory; therefore, if a tape code's upper four bits are A, B, C, or D, they would become 2, 3, 4, or 5 in memory. (See Appendix B.)

Sp	A, 0	0	3, 0	@	C, 0	P	5, 0
!	2, 1	1	B, 1	A	4, 1	Q	D, 1
"	2, 2	2	B, 2	B	4, 2	R	D, 2
#	A, 3	3	3, 3	C	C, 3	S	5, 3
$	2, 4	4	B, 4	D	4, 4	T	D, 4
%	A, 5	5	3, 5	E	C, 5	U	5, 5
&	A, 6	6	3, 6	F	C, 6	V	5, 6
'	2, 7	7	B, 7	G	4, 7	W	D, 7
(	2, 8	8	B, 8	H	4, 8	X	D, 8
)	A, 9	9	3, 9	I	C, 9	Y	5, 9
*	A, A	:	3, A	J	C, A	Z	5, A
+	2, B	;	B, B	K	4, B	3/4 ([)	D, B
,	A, C	½ (<)	3, C	L	C, C	¢ (\ )	5, C
-	2, D	=	B, D	M	4, D	CR (])	D, D
.	2, E	¼ (>)	B, E	N	4, E	° (^)	D, E
/	A, F	?	3, F	O	C, F	_	5, F
				DEL	F, F	<> (~)	7, E

FIELD IDENTIFIER (TERMINATION) CODES: The following chart shows the paper tape USASCII control codes which cause tape read instructions to be terminated, and some of which set a specified flag pattern. Each code is represented by two hexadecimal digits. Codes in Column 0 of the USASCII table do not affect any flags; codes in column 1 of the table set "K" flags. These codes do not enter into memory.

USASCII COLUMN 0 FIELD IDENTIFIER CODES (DO NOT AFFECT FLAGS)	
CODE	PAPER TAPE VALUE
NUL	0, 0
SOH	8, 1
STX	8, 2
ETX	0, 3
EOT	8, 4
ENQ	0, 5
ACK	0, 6
BEL	8, 7
BS	8, 8
HT	0, 9
IF	0, A
VT	8, B
FF	0, D
CR	8, D
SO	8, E
SI	0, F

USASCII COLUMN 1 FIELD IDENTIFIER CODES					
		FLAG PATTERN SET BY CODE *			
		K FLAG NUMBER			
CODE	PAPER TAPE VALUE	3	2	1	4
DLE	9, 0	0	0	0	0
DC1	1, 1	0	0	0	1
DC2	1, 2	0	0	1	0
DC3	9, 3	0	0	1	1
DC4	1, 4	0	1	0	0
NAK	9, 5	0	1	0	1
SYN	9, 6	0	1	1	0
ETB	1, 7	0	1	1	1
CAN	1, 8	1	0	0	0
EM	9, 9	1	0	0	1
SUB	9, A	1	0	1	0
ESC	1, B	1	0	1	1
FS	9, C	1	1	0	0
GS	1, D	1	1	0	1
RS	1, E	1	1	1	0
US	9, F	1	1	1	1

* 0 = flag is reset
1 = flag is set

The NUL code is the same as a sprocket feed code in that no channels are punched in a frame, and thus, it functions differently than the other field identifier codes. During a read tape instruction, it is ignored (treated like a delete code – DEL) until the first significant character of data is read. If encountered after the first significant character, it will then be treated as a field identifier code and will terminate the read instruction. It should not be used for a field identifier code if a variable field of data would ever contain no significant data but only a field identifier code. This would cause the NUL code to be ignored, since a significant character was not read, and it would not serve its intended function to terminate the instruction. This would result in the paper tape getting out of step with the program.

The END OF ALPHA code (0 0) is the same as the NUL code.

The DEL (Delete) code is completely ignored by all paper tape read instructions, and does not count as a character read. It consists of a punch in all 8 channels in a frame of tape.

13. OUTPUT WITH PAPER TAPE/EDGE PUNCHED CARD PERFORATOR

The instructions described in this section provide the means to output data from the TC 500 into punched paper tape and/or edge punched cards by using a style A 562 Paper Tape/Edge Punched Card Perforator as the output adjunct. All subsequent reference to "paper tape" applies both to punched paper tape and to edge punched cards, unless indicated otherwise.

Tape punching is serial at a speed up to 40 characters per second when no printing accompanies it. When printing with the TC 500 printer accompanies punching paper tape, the punching speed is up to 20 characters per second.

The internal character code of the TC 500 is USASCII and output to paper tape will normally be in this code. However, any 5, 6, 7, or 8 channel paper tape code can be punched by the TC 500 by utilizing a Table of Output Code Assignments for conversion of the internal code into the desired paper tape code (refer to 13.5). The firmware for 5 channel code is different than that for 6, 7, or 8 channel "table look-up" firmware or for USASCII No table firmware.

The instructions provide the ability to print and punch data from the Accumulator, print and punch alphanumeric data from memory, and type or type into memory while punching. In addition, a register is provided which counts the number of codes punched. This enables the use of continuous edge punched cards by making it possible to determine when one continuous card has been filled or to fill any unused portion of a continuous card with feed codes before aligning the next continuous card to the first sprocket hole.

The Paper Tape Punch instructions are designed to function in three ways: (1) When proper tape punching conditions exist, punching will occur according to the specifications of the instruction. (2) If the perforator is not connected or is turned off, the punch portion of the instruction is inhibited and the instruction is executed in accordance with its counterpart keyboard or print instruction. (3) If the perforator is turned on but does not have media loaded, execution of the punch instruction is held up until the condition is corrected. Thus, although the program may provide for punching, the perforator may be turned off or disconnected without affecting the operation of the rest of the system.

The mnemonic representations of the punch instructions are the same as selected keyboard and print instructions with the addition of a prefix letter "X".

13.1 PUNCHING ALPHANUMERIC DATA

The following instructions provide for punching alphanumeric data during keyboard entry or directly from storage in memory.

13.1.01 Typing and Punching

	Op Code	A	B
Type, Punch and Print	XTK	0:150	

The XTK instruction allows typing, printing and punching up to the maximum number of characters specified in the A field. This instruction functions like a TK instruction in every respect (refer to section 2) except that punching occurs with it. The termination of this instruction with an OCK or PK does not cause a code to punch. If the perforator is turned off, XTK will operate only as a TK instruction.

13.1.02 Typing Into Memory and Punching

	Op Code	A	B
Type Into Memory, Punch and Print	XTKM	0:150	

The XTKM instruction allows typing into memory, printing and punching up to the maximum number of characters specified in the A field. The prior use of LKBR designates the starting word for entry into memory. This instruction functions like a TKM instruction in every respect (refer to section 2) except that punching occurs with it. The termination of this instruction with an OCK or PK places an End of Alpha code in memory but does not cause a code to punch. If the perforator is turned off, XTKM will operate only as a TKM instruction.

13.1.03 Entering Alpha Into Memory and Punching

	Op Code	A	B
Enter Alpha Into Memory and Punch, Non-Print	XEAM	0:150	

The XEAM instruction functions exactly like the XTKM instruction except that printing does not occur. If the perforator is turned off, XEAM will operate only as an EAM instruction.

13.1.04 Printing Alpha From Memory and Punching

	Op Code	A	B
Print Alpha and Punch	XPA	0:N	

The XPA instruction simultaneously prints and punches the alphanumeric data stored in the memory location designated by the A field. The instruction is terminated upon reaching an End of Alpha code in the data; the End of Alpha code does not punch. This instruction functions like a PA instruction in every respect (refer to section 3) except that punching occurs with it. If the perforator is turned off, XPA will operate only as a PA instruction.

13.1.05 Punching Alpha From Memory, Non-Print

	Op Code	A	B
Punch Alpha from Memory, Non-Print	XA	0:N	

The XA instruction functions exactly like the XPA instruction except that printing does not occur. If the perforator is turned off, XA will operate as a "No Operation" (NOP) instruction.

13.1.06 Punching Special Codes

	Op Code	A	B
Punch Code	XC	0:15	0:15

The XC instruction punches into tape the bit pattern specified by the parameter fields. This instruction permits creating any one of 256 possible codes that can be derived from 8 channels (or bits). The A field represents the decimal value of the high order 4 bits (b_8, b_7, b_6, b_5 having decimal values of 8, 4, 2, and 1 respectively); the B field represents the decimal value of the low order 4 bits (b_4, b_3. b_2, b_1 having decimal value of 8, 4, 2, and 1 respectively) in the bit configuration of the desired code. The parity bit must be included in the appropriate bit position when applicable. In the case of USASCII code, the column number of the desired code in the table represents the A field (parity bit must be added to this when applicable); the row number of the desired code represents the B field. No printing occurs with this instruction. If the perforator is turned off, XC will operate as a "No Operation" (NOP) instruction.

EXAMPLE: Punch the USASCII code "RS" (Record Separator)

	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1
bit pattern ("1" = hole in tape)	0	0	0	1	1	1	1	0
decimal value	8	4	2	1	8	4	2	1
Parameter values	A =	1			B =	14		

This corresponds to the USASCII table location of RS in column 1, row 14

13.2 PUNCHING NUMERIC DATA FROM THE ACCUMULATOR

The following instructions provide for punching numeric data from the Accumulator. The pointer designates the high order digit position of the Accumulator at which printing and punching begin; the printing format and punching are controlled by the mask word selected. The instruction is terminated after punching and printing through digit position zero or when an "E" (End) mask code is encountered in the mask word.

The Punch Flag (P) in the mask word, when set, causes leading zeros to punch even though leading zero suppression mask codes (Z Z,) prevent their printing.

When the Punch Flag is reset and leading zero suppression mask codes are used, leading zeros will neither print nor punch.

The Punch Flag has no effect on the other mask codes.

Punctuation that is inserted by the mask codes during printing does not punch, only the numeric data itself. If an Ignore (I) mask code is used, the corresponding digit in the Accumulator will not punch. If the End (E) mask code is used, the corresponding digit neither prints nor punches and the instruction is terminated. All other mask codes cause the corresponding digit to punch. A mask word is used for all punch numeric instructions, even though printing may not be a function of a given instruction (see 13.2.04).

13.2.01 Print and Punch Numeric Data

	Op Code	A	B
Print and Punch Numeric	XPN	0:14	0:15

The XPN instruction prints and punches the contents of the Accumulator, starting at the high order digit position designated by the A field, in accordance with the print mask designated by the B field. The print mask value is relative to the mask table established by the last LPNR instruction. The Accumulator Flags position is ignored, as is any other data in the Accumulator to the left of the pointer. This instruction functions like a PN instruction in every respect (refer to section 3) except that punching occurs with it. If the perforator is turned off, XPN will operate only as a PN instruction.

When it is desired to punch a code reflecting the sign value of numeric data, this is accomplished by interrogating the Sign Flag using a Skip or Execute instruction and then punching the appropriate code using the XC instruction (refer to 13.1.06 above). This procedure also applies to punching a code to reflect the status of any other Accumulator Flags.

Mask Code	Printing	Punching
F	Print $	No Effect
+	Suppress Punctuation	No Effect
P	No Effect	Leading zeros punch if P flag set
D D, D	Print Character regardless of significance	Punch Character regardless of significance
X X	Trailing zero suppression	
C C	Leading zero & trailing zero suppression	
Z Z,	Print if: (1) Accum. digit not zero (2) A non-zero digit has been printed	Punch if: (1) P is Set (2) Accum. digit not zero (3) A non-zero digit has been punched
S	Print only if Accum. digit not zero	
I	Ignore	Ignore
E	Terminate, Non-Print	Terminate, Non-Punch

Fig. 13-1 Print & Punch Characteristics of Mask Codes

13.2.02 Print and Punch Numeric Data, Shift Ribbon if Minus

	Op Code	A	B
Print and Punch Numeric, Shift Ribbon if Minus	XPNS-	0:14	0:15

The XPNS- instruction is the same as the XPN instruction except that the ribbon color is changed (normally to red, refer to section 3) if the Accumulator Sign Flag is set (minus). If the perforator is turned off, XPNS- will operate only as a PNS- instruction.

13.2.03 Print and Punch Numeric Data, Shift Ribbon if Plus

	Op Code	A	B
Print and Punch Numeric, Shift Ribbon if Plus	XPNS+	0:14	0:15

The XPNS+ instruction is the same as the XPN instruction except that the ribbon color is changed (normally to red) if the Accumulator Sign Flag is reset (plus). If the perforator is turned off, XPNS+ will operate only as a PNS+ instruction.

13.2.04 Punch Numeric Data, Non-Print

	Op Code	A	B
Punch Numeric, Non-Print	XN	0:14	0:15

The XN instruction is the same as the XPN instruction except that no printing occurs. A mask word is used with this instruction since it controls the punching. The mask word selected may be the same as is used with other Print Numeric instructions since it would not affect the non-print function of this instruction. If the perforator is turned off, XN will operate as a "No Operation" (NOP) instruction.

13.3 OTHER PUNCHING INSTRUCTIONS

A Punch Count Register is utilized to count the number of codes that have been punched. This is necessary to permit the proper handling of continuous edge punched cards. After punching in one card, it is necessary to punch feed codes (sprocket holes) in the unused portion of the card so that the first sprocket hole of the next card can be aligned in the punch station. These instructions control the Punch Count Register in this use.

13.3.01 Loading the Punch Count Register

	Op Code	A	B
Load Punch Count Register	LXC	0:255	

The LXC instruction will load the number contained in the A field into the punch count register, and is normally used at the start of each new continuous edge punched card to reset the count. The punch count register is incremented by one for each code punched from any punching instruction. If the register contains 255, incrementing causes the register to become zero.

13.3.02 Modifying Instructions by the Punch Count Register

	Op Code	A	B
Modify by Punch Count Register	XMOD		

The XMOD instruction will modify the parameter field of the next instruction by the contents of the Punch Count Register. This instruction is similar to the MOD instruction in that it will modify most instructions (see section 7). However, it uses the value in the Punch Count Register instead of one of the Index Registers. It cannot be changed by the Index Register instructions (ADIR, IIR, LIR, DIR, TAIR).

13.3.03 Punching Feed Codes

	Op Code	A	B
Punch Feed Codes	XB	0:255	

The XB instruction causes feed (sprocket) holes to be punched. The number of codes punched will be the difference between the number in the A field and 255. If the perforator is turned off, XB will operate as a "No Operation" (NOP) instruction. When Edge Punched Cards are the media present, punching of sprocket holes is inhibited. Therefore, the card is just advanced with no punching.

13.3.04 Use of the Punch Count Register

Advancing the unused portion of a single edge punched card to the end of card: When the number of codes to be punched for one record will never exceed the length of one card, single edge punched cards may be used; the unused portion could be advanced in this manner:

Assume that a card 10 inches in length (100 codes) is used, and that the following data is punched (LXC is loaded with 0 prior to punching data):

Product Number	6 codes
Description	30 codes
Price	6 codes
Field Identifier codes	3 codes

The Punch Count Register now contains 45. The following instructions will advance the card to the end:

```
XMOD              (adds 45 to parameter value of XB instruction)
XB     140        (parameter is the difference between 255 and the length of
                   card in terms of number of frames less 15 frames to eject
                   card from punch station).
```

The XB parameter of 140 plus the register value of 45 = 185; thus, the XB causes the feeding of 70 frames (255 - 185 = 70). Since the card length is 100, and since 45 codes of data were entered, 55 frames are needed plus 15 frames to eject.

Advancing the unused portion of a continuous edge punched card to the end: When the maximum number of codes to be punched for one record can exceed the length of one card, continuous edge punched cards may be used; the unused portion can be advanced in this manner so that the next new card is aligned to the first frame. This procedure is valid if the total record length does not exceed 255 codes.

Note the use of an Index Register in conjunction with the Punch count register.

Assume that a card 10 inches in length (100 codes) is used, and that 180 codes of data were punched (LXC is loaded with zero before punching data). Punch Count Register now contains 180. The following instructions will advance the card to the start of the next continuous card.

```
     1  XMOD              (Punch Count Reg. value added to LIR
                           parameter)
     2  LIR   1     0     (Index Reg. 1 loaded with 180)
STEP 3  ADIR  1   155     (Difference between 255 & card length (100)
                           added)
     4  EX    T     I  2  (Test I.R. Flag for overflow)
     5  ADIR  1     1     (Add 1 to Index register)
     6  BRU STEP 3
     7  MOD        1
     8  XB         0      (Punch feed codes)
```

The first time step 3 is executed, Ind. Reg. 1 contains 79 (335 - 256 = 79) and the I Test flag is set at step 4 causing step 5 & 6 to be executed. 1 is added to I.R. 1 at step 5, resulting in a value of 80, and a branch back to step 3. The second time step 3 is executed, I.R. 1 then contains 235 (155 + 80) and the I Test flag is not set at step 4, causing steps 5 and 6 to be skipped. Step 7 adds 235 to the XB parameter in step 8 causing the XB to feed 20 frames (255 – 235 = 20). Since 180 codes of data were punched, 20 frames were unused in the last card requiring advancing 20 frames.

Regardless of how many codes had been punched (not exceeding 255), the above routine would cause the correct number of sprocket holes to be advanced so that the first hole of the next card would be aligned at the punch station. The length of the card (reflected in step 3 parameter) may be changed to meet individual requirements, and a substitution of another index register may be made without affecting the routine.

13.4 OUTPUT INDICATOR LIGHTS AND FLAGS

Four punch indicator lights are provided on the keyboard of TC 500's with Input/Output capability to alert the operator as to the status of the perforator. In addition, there are four Punch Flags, each one associated with an indicator light, to permit the program to interrogate the status of the perforator.

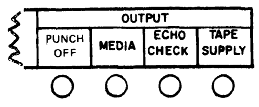


Fig. 13-2 Output Indicator Lights

13.4.01 Punch Off Indicator

The Punch Off Indicator light is turned on and Punch Flag P4 is set if the paper tape perforator ON/OFF switch is in the "OFF" position during the execution of a punch instruction. The instruction will be executed; however, the punching portion of it will be inhibited. The correction of the condition by turning on the perforator will cause the indicator to be turned off and Punch Flag P4 to be reset on the next punch instruction.

13.4.02 Media Not Present Indicator

If the program is attempting to execute a punch instruction and media is not present in the punch station, the instruction is held up; the Media indicator light is turned on and Punch Flag P1 is set. The subsequent placing of an edge punched card in the punch and depression of the Card Lock button, or the placing of tape in the punch will cause the system to resume execution of the punch instruction. During the execution of the instruction, the indicator light is turned off and Punch Flag P1 is reset.

13.4.03 Echo Checking

If the Echo Checking feature indicates that incorrect punching has occurred during a punch instruction, the Echo Check Indicator light is turned on and Punch Flag P2 is set. Punching is not inhibited; the flag stays set and the indicator remains on during subsequent punch instructions.

To use this feature properly, the program must provide for checking flag P2 at least after each line (or Transaction) of punching. When the flag is set, a Skip or Execute instruction would enable performing the necessary instructions to sound the Alarm, punch a tape error code, or to take other corrective action, and to reset flag P2.

13.4.04 Tape Supply Indicator

When reel tape is being used and the supply is nearly exhausted (approximately 20 feet remaining), the Tape Supply Indicator light is turned on and Punch Flag P3 is set. Correction of this condition by placing a new roll of tape in the supply reel will turn off the indicator and reset the flag on the next punch instruction. This condition does not halt execution of the program nor inhibit the punching.

Normally, when this light turns on, there is adequate tape remaining to finish that entry, or even the next several. However, to insure that the operator does not ignore the condition, the flag should be checked at the beginning of each line entry, with provision made to sound the Alarm and halt the processing if the flag is set. Thus, the Alarm would ring at the start of each entry until the condition was corrected.

13.4.05 Flag Instructions

The execution of a LOAD, SET, RESET or CHANGE instruction involving the Punch Flags will also cause the associated indicator light(s) to either be turned on or off depending on the instruction used.

13.4.06 Initializing the Program

Since the status of the perforator is apparent only as a punch instruction is being executed, it is recommended that a punch instruction be used during the program initialization routine with subsequent testing of the Punch Flags. This can be accomplished with a "Punch Code" instruction using parameter values of zero (XC 0 0), or a "Punch Feed Codes" instruction with a parameter of 254 (XB 254). In either case, the result would be to punch one sprocket hole in the tape and to set any Punch Flags affected if improper punching conditions existed. A test of the flags could then cause the Alarm to ring if any flags were set.

13.5 TABLE OF OUTPUT CODE ASSIGNMENTS

When a code set other than USASCII is desired in the output tape, or when certain variations may be desired in the USASCII set, a Table of Output Code Assignments may be used. This permits output into any 5, 6, 7, or 8 channel code without modification to the Perforator. Output in USASCII code does not require a table.

The table is loaded into a Normal memory area and occupies up to 16 words. The loading may accompany regular loading of user programs. This table is a separate table from the Table of Input Code Assignments described in section 12.4. Each TC 500 internal character selects a particular character position in the output table. The 8-bit code that is put in each character position of the table is the code that will be punched into the output tape.

Normally, the Punch Code (XC) instruction will be used to punch field identifier (functional) codes. However, since any of the TC 500 internal characters, through the table, can cause any 8-bit code to be punched, field identifier codes may be punched in this manner also.

The programmer may construct an output table to achieve any desired output code. However, tables are available that contain "standard" values for the following code sets:

BCL/IBM	8 channel
Friden	8 channel
USASCII	8 channel
Teletype	5 channel (Baudot)

The bit configuration of most Friden tape codes is the same as BCL. However, many of the functional code names given to the various codes are different, and for that reason a table is provided for ease in interpretation.

13.5.01 Firmware Subsets for the Table of Code Assignments

The firmware which includes "table look-up" for conversion of the internal code to the output code is different than firmware which does not use "table look-up" (output in USASCII). Thus, a USASCII table is available for use in systems that require "table look-up" firmware due to varying output code requirements.

NOTE: Output in 5-channel tape code requires firmware that is different from either 8-channel "table look-up" firmware or for output in USASCII without "table look-up".