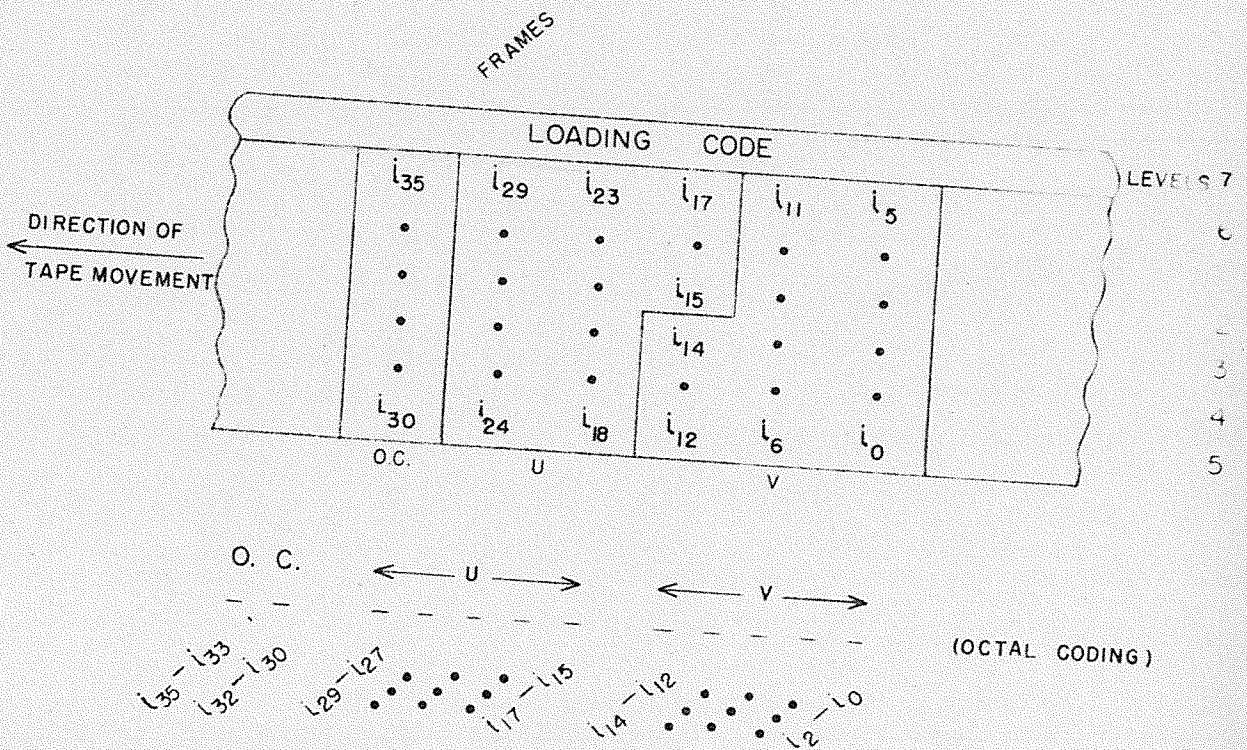
4. INPUT, OUTPUT SYSTEMS.

a. GENERAL. - The Input and Output Systems as used in conjunction with the 1103 computer provide the means of communication between the computer and the operator. To achieve this communication three things are essential to each input or output system: a medium for external representation of information; external equipment which is capable of, for output, receiving information internally represented and presenting it in its particular external medium of representation or, for input, sensing external information as it is represented by its particular medium of representation and presenting it to the computer; and a means of transmitting and controlling this transfer of information between the external equipment and the computer. The following paragraphs list different media of external information representation, external equipment, and the means of information transfer between and 1103 and listed external equipment.

(1) EXTERNAL INFORMATION REPRESENTATION.

(a) PUNCHED PAPER TAPE. - Paper tape is described as being divided into rows and columns: a single column of positions across the width of a tape is called a frame; frames are divided parallel to the length of the tape into seven levels. Six of these levels are used primarily to represent information to be placed in computer storage, the seventh is used to present loading directions. A hole punched in any of the six lower levels represents a one; the absence of a hole represents a zero. The tape is thus coded according to binary number notation although the binary digits may be grouped to represent decimal numbers or octal numbers. If the tape is punched to represent a computer instruction, the bits representing the instruction coded in bi-octal form are positioned on the tape as follows:



PX 71871

A bi-octal coded operand is represented on the tape with the same positioning of its 36 bits. Thus, six frames of tape are necessary to represent a bi-octal-coded instruction or operand.

(b) TYPED MANUSCRIPT. - Typed copy is presented on standard 8 1/2 by 11 inch typewriter paper and may consist of letters, numbers, signs, symbols, and punctuation marks spaced in any chosen typewritten format.

(c) TABULATING CARDS. - Information is represented, using 80 column tabulating cards, according to specified patterns of punched rectangular holes in the card.

(d) PRINTED COPY. - Letters and numbers are printed on paper 19 inches wide fed from an accordian-folded pack.

(2) EXTERNAL EQUIPMENT. - For input only to the 1103, a tape reader, which processes information presented to it as punched paper tape, is used.

The types of external equipment which are used for output only are the High Speed Punch which prepares punched paper tape in accordance with information received by it from the computer, the Typewriter (or Flexowriter) which prepares a typed copy of specified internal information, and the Line Printer which prints on roll paper, a line at a time, numbers and letters received from the computer. The Line Printer is optional equipment, i.e., not received as part of a standard 1103 installation, in this category.

An optional piece of equipment which can be utilized for both input and output operations occurring separately or simultaneously, is the Controlled Reproducer. The Controlled Reproducer prepares punched cards during its output mode of operation and processes the contents of punched cards for internal representation during its input mode of operation. Because of individual card channels for input and output operations, both modes may be selected for simultaneous "reading" and "writing".

(3) INFORMATION TRANSFER. - Information being transferred in or out of the computer is routed via a buffer (temporary storage)register and the X-Register in its transmission between external equipment and an addressed computer location. The buffer registers to be used are determined by the selection made of external equipment for receiving or loading information. There are several special buffer registers whose primary reason for existence is to provide temporary storage for such information between the X-Register of the computer and the external equipment. Information transmission in this manner (as explained in general in the following paragraphs) is performed as a function of coded computer instructions.

Information transfer to external equipment, via the special buffer registers, is initiated by computer instructions which direct the flow of information from an addressed computer location, via the X-Register, to the buffer register. The presence of the information in the buffer register allows under the proper conditions, the external equipment to sense the contents of the buffer register and to translate this information and present it according to the media of information representation of the equipment being used. Information transfer via

PX 71871

the special buffer registers from external equipment to the computer is similarly directed by a computer instruction. The information is routed under the proper conditions from the external equipment via a buffer register to an addressed computer location with the appropriate translation of information as represented externally to its representation machine-wise. The computer instructions which direct this flow of information are Punch, Print, External Write and External Read. During the execution of each of these instructions (and an External Function instruction), a "lockout" test is made to see if a previous use, if such is the case, of the particular register involved is completed. This is necessary before the current instruction can be completed and before the next instruction can be placed in computer control in readiness for its execution. If the necessary requirements are not met, a lockout condition exists which effectively stops computer operating time. For output operations a lockout condition exists if the control circuitry of the particular buffer register being referenced has not yet received an indication that the register has been cleared of previous information being routed to external equipment. The condition continues to exist, temporarily stopping computer operations, until the register is cleared, after which the current information may be received by it (machine operations being resumed), and the execution of the instruction completed. For input operations a lockout condition exists if the control circuitry of the particular register being referenced has not yet received an indication that the register has received the input information from external equipment. This condition continues to exist, temporarily stopping computer operations, until the information is received after which computer operations are resumed and the instruction completed.

A listing of the buffer registers mentioned above is as follows: the High-Speed Punch Register, HPR; the Typewriter Register, TWR; the Input-Output Register A, IOA; the Input-Output Register B, IOB. Since HPR and TWR each provide communication with a specific piece of external equipment, a discussion of these registers will be included in the discussion of the proper external equipment.

The Input-Output Registers A and B are, respectively, eight stage and 36 stage registers. They are used to provide communication between the computer and the input and/or output equipment for which no specific buffer register has been provided. Since no reference to a specific piece of external equipment is made by the External instructions which use the IOA and IOB buffer registers for information transfer, a means of selecting the appropriate piece of equipment must also be provided. The IOB register is used for this purpose also, according to the operations of the External Function instruction, EF-v. This instruction places representations of ones, as coded and located at address v, in selected stages of IOB and directs IOB Control to then interpret the contents of IOB and to accordingly select and initiate a function of external equipment.

In using the IOA and IOB registers it is possible to generate B Faults* because of timing requirements which have not been taken into consideration or because of faulty external equipment. An IOB (or IOA) Read Fault, Class II,* occurs if IOB (or IOA) receives information from external equipment before the control circuitry of the register has received an indication that previous output operations involving it have been completed. An IOB (or IOA) Read Fault Class I*, occurs if information is received by IOB or IOA from external equipment

* Note: See subparagraph d. (4)(b), Paragraph 5.

before the control circuitry of the register has received a signal to dispose of information placed there during previous input operations.

In summary, information being transmitted to or from an addressed computer location via special buffer registers, and according to coded instructions, proceeds as follows:

(a) For input operations:

External equipment → IOA and/or IOB → X-Register → addressed computer location according to External Function EF-v and External Read ERjv

(b) For output operations:

Addressed computer location → X-Register → HPR → Punch according to Punch, PUjv

Addressed computer location → X-Register → TWR → Typewriter according to Print, PR-v

Addressed computer location → X-Register → IOA and/or IOB → external equipment according to External Function EF-v and External Write EWjv

b. EXTERNAL EQUIPMENT FOR INPUT AND/OR OUTPUT

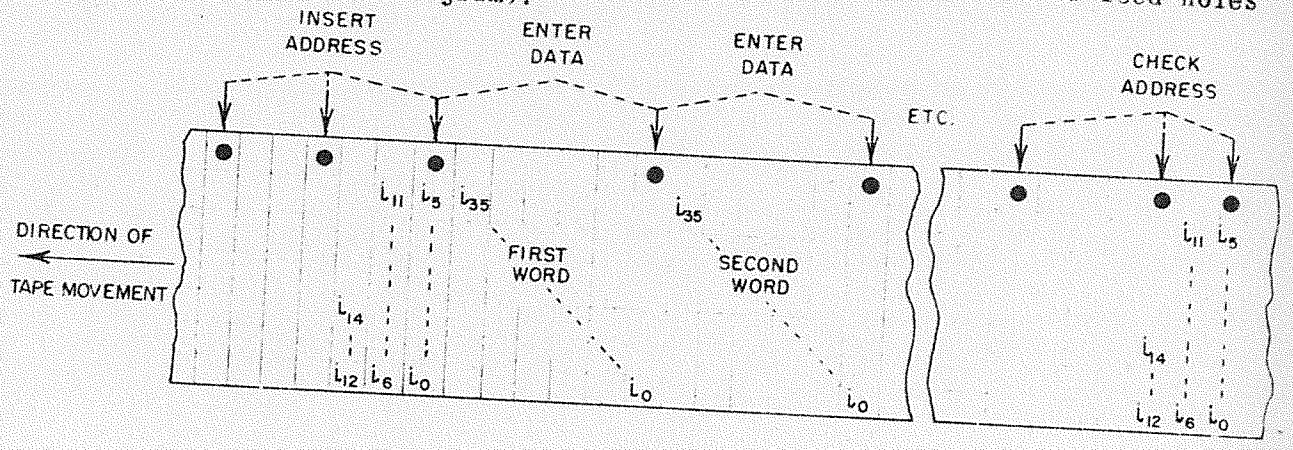
(1) INPUT ONLY - FERRANTI TAPE READER. - The Ferranti Tape Reader Input System utilizes the Reader, the IOA and IOB registers, and IOA and IOB control circuitry in presenting to the computer information from seven-level punched paper tape. The Ferranti Tape Reader has a single reading station composed of a column of seven photocells associated with the seven levels of a tape frame as the tape moves through the Reader. Also punched in the tape are feed holes parallel to the length of the tape between levels two and three. The passage of tape, with these feed holes in it, through the reader produces feed pulses which enable the transfer to IOA of information in the frames associated with the feed holes. (When operating in the NORMAL mode a missing feed pulse causes a B Fault and lights the Missing FP fault indicator on the Supervisory Control Panel.) Thus the six (or seven) bits represented by each frame are sensed simultaneously and transmitted to the six (or seven) lower order stages of IOA. The representation of ones in the tape frame are actually transmitted by a signal received by the corresponding stage of IOA; the representations of zeros are "transmitted" by the absence of such a signal. Thus IOA must be initially clear for an accurate representation of the contents of a frame before such transmissions occur. The transfer of information from IOA to an addressed computer location, via the X Register, is accomplished by programming an External Read instruction, ERjv with $j = 0$, to be executed each time IOA receives the contents of a frame from the tape reader.

The six lower levels of the punched tape (6, 1, ..., 5) contain information to be placed at an addressed computer location and may contain loading

PX 71871

instructions directing the insertion of information into storage; or the seventh level of the tape may be punched to effect the proper disposal in the computer of the information in the lower six levels. In either case the loading directions in the six or seventh levels of consecutive frames must be interpreted within the computer to achieve the designated storage of information.

Typical coding, as punched in the seventh level of a tape, for loading directions is shown on the following tape diagram: (Sprocket or feed holes are not shown on this diagram).



These distributions of holes, as represented by "one's" in the computer, are interpreted as follows:

1. Insert Address. - The address designated by $i_{14}...i_0$ is to be placed at an addressed location which will serve as a "dummy" PAK.

2. Enter Data. - The first word, $i_{35}...i_0$, is to be placed at the address held in the above "dummy" PAK; successive words are to be placed at consecutive addresses.

3. Check Address. - The address designated by $i_{14}...i_0$ should be identical to the address currently held in the "dummy" PAK.

Information loading from the Ferranti Tape Reader is controlled by using the External Function instruction to place bits in selected stages of IOB. The three stages in which a combination of ones must be placed are, with their corresponding interpretations, as follows:

IOB_{17}	Select Tape Reader
IOB_{16}	Start Tape Reader
IOB_{15}	Stop Tape Reader

The "Select Tape Reader" stage must contain a one for all operations affecting the tape reader. The combinations of one's in IOB effecting control of the Tape Reader are as follows:

PX 71871

Start (Free Run) - IOB_{17} and IOB_{16}, coded as EF-v, (v) being 00 00006 00000 (octal). This combination causes the reader to start and continue running with the six (or seven) bits of each consecutive tape frame being transmitted to IOA.

Stop - IOB_{17} and IOB_{15}, coded as EF-v, (v) being 00 00005 00000 (octal). This combination causes the reader to stop upon the transmission to IOA of the six (or seven) bits of the tape frame sensed by the reader immediately following the interpretation, by the control circuitry of the reader, of (IOB) as a stop.

Step Tape Reader - IOB_{17}, IOB_{16}, and IOB_{15}, coded as EF-v, (v) being 00 00007 00000, (octal). This combination causes the reader to start, transmit to IOA the six (or seven) bits of the first frame of tape it senses, and stop.

The normal rate of tape speed through the Reader is approximately 200 frames per second, but, as a function of line voltage, it may be as high as 230 frames per second. Thus the time interval between the sensing of successive frames of tape may vary from 4.3 to 5 milliseconds. Timing factors to be taken into consideration for the three controlled operations of the tape reader are as follows:

1. Start (Free Run) - Information is being transmitted from the tape to IOA at the rate of one frame every 4.3 milliseconds (assuming for synchronization the shortest time). The contents of IOA should be transmitted to the X-Register, using the External Read instruction, at corresponding time intervals for the most efficient use of computer time. The execution of the External Read instruction clears IOA after its contents have been transmitted to X. If another External Read is programmed before the contents of the next frame have been received from the reader, machine operations are temporarily halted by a lockout condition until IOA receives information. If information is transmitted to IOA from the tape reader before IOA has been cleared of the contents of the preceding tape frame, a B Fault occurs and is indicated by the IO Fault light on the Supervisory Control Panel.

2. Stop - To insure that the reader is stopped so that the next tape frame whose contents are transmitted to IOA is the last frame desired to be sensed by the reader, the External Function instruction coded for a reader stop should be programmed to be executed at least one millisecond before the sensing of the last desired frame. This can be done by programming it timewise to follow within three milliseconds of the previous External Read. After the reader is stopped an External Read instruction must be programmed to transmit the contents of IOA to the X-Register. If the stop is made during tape loading operations and IOA will not be used before the reader is again started, the External Read may be programmed to precede the start instruction. If the stop instruction is not executed before the transmission to IOA of the contents of the last desired frame and a read instruction is not programmed to transmit this content of IOA to X, a B Fault will be indicated and the computer will be stopped when IOA is to receive the contents of the next frame which must be sensed before a stop will actually occur.

PX 71871

When a stop occurs, a three millisecond time delay, allowing the reader to stop physically, is generated in the reader control circuitry. This time delay prevents the reader from being started for three milliseconds. Thus a start instruction, if programmed to be executed during this period, is not effective until the three millisecond period has elapsed.

3. Step - During step operations the first frame sensed by the reader is the frame which allows the reader stop to be performed. The contents of the first tape frame are transmitted to the IOA register. Thus consecutively programmed step instructions must each be followed by an External Read instruction with the first preceded by an External Read if IOA is not initially clear. There is a five-millisecond time lapse between successive frames at the operating speed of the reader. A three-millisecond delay is generated by the stop reader operation, and approximately a one-millisecond delay is generated by the reader start operation. Therefore, approximately nine milliseconds are consumed in a step instruction.

Manual operations necessary to the functioning of the Ferranti Tape Reader consist of inserting the tape to be read in the tape passage of the Reader and depressing the Start button on the Reader (which turns on the power). After these operations are completed, the computer is started with the first instruction to be executed being the initial instruction of an input routine for the Tape Reader. This input routine will include instructions to start the reader, read the (IOA) to addressed computer locations, and process the information being loaded according to the coded loading directions.

(2) OUTPUT ONLY.

(a) HIGH-SPEED PUNCH. - The High-Speed Punch Output System consists of the High-Speed Punch Register, HPR, the High-Speed Punch Control, and the Punch itself (as shown on Plate 6-4) which presents information on seven-level punched paper tape at the rate of 60 frames per second. The High-Speed Punch Register is a seven-stage buffer register which temporarily stores digits in their transmission between the computer and the punch. The High-Speed Punch Control Circuitry senses the contents of HPR and energizes the Punch to perforate the tape frame in the levels whose corresponding stages in HPR contain representations of ones. After information is received from HPR, the tape is advanced through the Punch by one frame, placing the next tape frame in position to receive information.

The transmission of information to the Punch is initiated by the coded computer Punch instruction, PUjv, operation code 63. This instruction directs the transmission, via the X-Register, of the contents of the right-hand six stages of v to the first six stages of HPR. Actually, only the representations of ones in stages $X_5 \ldots X_0$ are transmitted to the corresponding stages of HPR. A j of one is transmitted directly to the seventh stage of HPR. According to the contents of $HPR_6 \ldots HPR_0$, holes may be punched in the corresponding tape levels 7, 6, 1, ..., 5. To transmit from address v a word of 36 bits so that each group of six bits is punched in its correct relative tape position, the Punch

instruction must be executed six times, each execution being preceded by a shift instruction. The word must be shifted at the v-adress of PUjv six times, the first shift operation positioning the six most significant bits of the word in the six right-most stage of v with each succeeding shift operation positioning the next most significant six bits in the right-most stages of v.

The High-Speed Punch is ready for operation when the tape is properly threaded and the toggle switch at the lower right of the front panel is set to its "ON" position. This starts the punch in continuous cycles of operation. The time duration of a punch cycle is 16.7 milliseconds. The first punch instruction initiated during a punch cycle n does not make an effective punch reference, (HPR) → Punch, until the beginning of the next cycle and requires the first 12.5 ms of this succeeding cycle, n + 1, for the completion of its punch reference. Thus HPR is not ready to receive additional output information until the last 4.2 ms of the cycle n + 1. If a second punch instruction is initiated before such an indication of readiness is given to HPR, a computer lockout will occur stopping the computer until 4.2 ms before the beginning of cycle n + 2. If the second punch instruction immediately followed, during cycle n, the first punch instruction, the lockout time is 12.5 ms plus the unexpired time of cycle n after the lockout period effected by the second punch begins; thus, if the first punch instruction was initiated immediately after the beginning of cycle n, a lockout time of approximately 29 ms would occur (12.5 plus almost 16.7). This is the maximum lockout time possible. The punch reference made by the second punch instruction will not be completed until the first 12.5 ms of cycle n + 2 have elapsed. If punch instructions are programmed to be executed at 16.7 ms intervals, a computer lockout may be effected by the second Punch, but the execution of successive punch instructions will be in synchronization with the punch cycles because of the actual stopping of computer operating time.

(b) TYPEWRITER. - The Typewriter Output System consists of the Typewriter Register, TWR, the Typewriter Control, and the Typewriter itself, as shown on Plate 6-5, which produces typewritten manuscript with letters in upper and/or lower case, numbers of the decimal system typed on the same level as letters or elevated above this level as exponents, and a variety of characters such as a plus sign (+), a minus sign (-), and a comma (,), etc. These letters, numbers, and characters are typed in a format determined by the response of the Typewriter Control circuitry to six-bit codes which are interpreted as, and actuate the typewriter to, such physical operations as space, carriage return and tabulator. Incidental to a carriage return is a line spacing operation, the number of spaces between consecutive lines being determined by the manually set line spacer on the Typewriter. The tabulator operation also requires a manual setting; the interpretation of the tab code as such releases the carriage to move to a predetermined manual tab setting. A shift-up operation positions the type bars for upper case typing; to resume typing in lower case a shift-down operation must be executed.

Letters and numbers are also typed, as directed by the Typewriter Control, in response to six-bit codes with each code representing a single letter or number. To type a letter in upper case or a number as an exponent, the type bars must be shifted to their upper case position before the typewriter responds to the letter or number code. Two characters are represented by a single code, the character which is typed being determined by the location of the type bars in their upper or lower case position.

A list of the typewriter codes, given as two-digit octal numbers, are presented with their associated typewriter functions in Table 6-1. If an illegal code is sent to the Typewriter, an A Fault will occur, which is indicated by the Print fault light on the Supervisory Control Panel, and the computer will be stopped in its operations at the point when its control received a signal from the typewriter control that an illegal code had been detected. The computer may be restarted from where it ceased operations by first pushing the CLEAR A FAULT button and then by pushing the START button on the computer. This insures that the illegal code is cleared from TWR.

Typewriter operations for output are initiated by the computer instruction Print, PR-v, operation code 61. This instruction directs the transmission, via the X-Register, of the contents of the right-hand six stages of v to the six stages of the Typewriter Register which serves as a buffer register for information being transmitted from the computer to the typewriter. Actually only the representations of ones in stages $X_5 \ldots X_0$ are transmitted to the corresponding stages of TWR. The Typewriter control circuitry senses the representation of a six-bit code in TWR and in so doing causes the typewriter to type a single letter, number, or character or perform a single typewriter operation. For example, to type the single upper case letter M, the following two instructions would be programmed:

61-v, $(v_5 \ldots v_0)$ is 100 111

61-v, $(v_5 \ldots v_0)$ is 000 111

A movement left of the carriage by one space to reposition it for the next typing operation occurs automatically after each typing of a letter, number, or character.

The speed of the typewriter allows it to type approximately nine characters per second. Thus a timing period of approximately 105 ms will elapse between consecutive prints by the typewriter. (Coded typewriter functions require a longer timing period for their completion.) If successive Print instructions are programmed to be executed in less time, a lockout condition will exist until the control circuitry of TWR receives an indication that the typewriter is ready to receive another six bit code. The maximum lockout time for the typewriter is dependent upon which phase of the typewriter cycle is being executed currently when a test lockout reference of a successive Print instruction is made.

Manual preparation of the typewriter for operation consists of inserting paper and setting the OFF-ON switch to the right of the typewriter keyboard to the ON position.

(c) LINE PRINTER. - The Line Printer Output System utilizes the Line Printer with its Format Switchboard, located to the right of the platen assembly, its control circuitry, the IOB Register, and the IOB control circuitry. The medium of information representation is printed paper on which the printed characters may be decimal numbers, letters, a period, and a minus sign. The paper is positioned in the printer on a carriage similar to a typewriter carriage. The characters chosen to be printed on one line across the width of the paper are printed almost simultaneously, with the design of the Line Printer allowing the printing of 150 such lines per minute. The spacing between lines is automatic, if so desired, during continuous printing of lines and may be

TABLE 6-1 . TYPEWRITER CODES

The upper case, UC, or lower case, LC, character is typed according to the position of the type bars.

Type Letter UC	Type Letter LC	Octal
A	a	30
B	b	23
C	c	16
D	d	22
E	e	20
F	f	26
G	g	13
H	h	05
I	i	14
J	j	32
K	k	36
L	l	11
M	m	07
N	n	06
O	o	03
P	p	15
Q	q	35
R	r	12
S	s	24
T	t	01
U	u	34
V	v	17
W	w	31
X	x	27
Y	y	25
Z	z	21

Type Number UC	Type Number LC	Octal
1	1	52
2	2	74
3	3	70
4	4	64
5	5	62
6	6	66
7	7	72
8	8	60
9	9	33
0	0	37

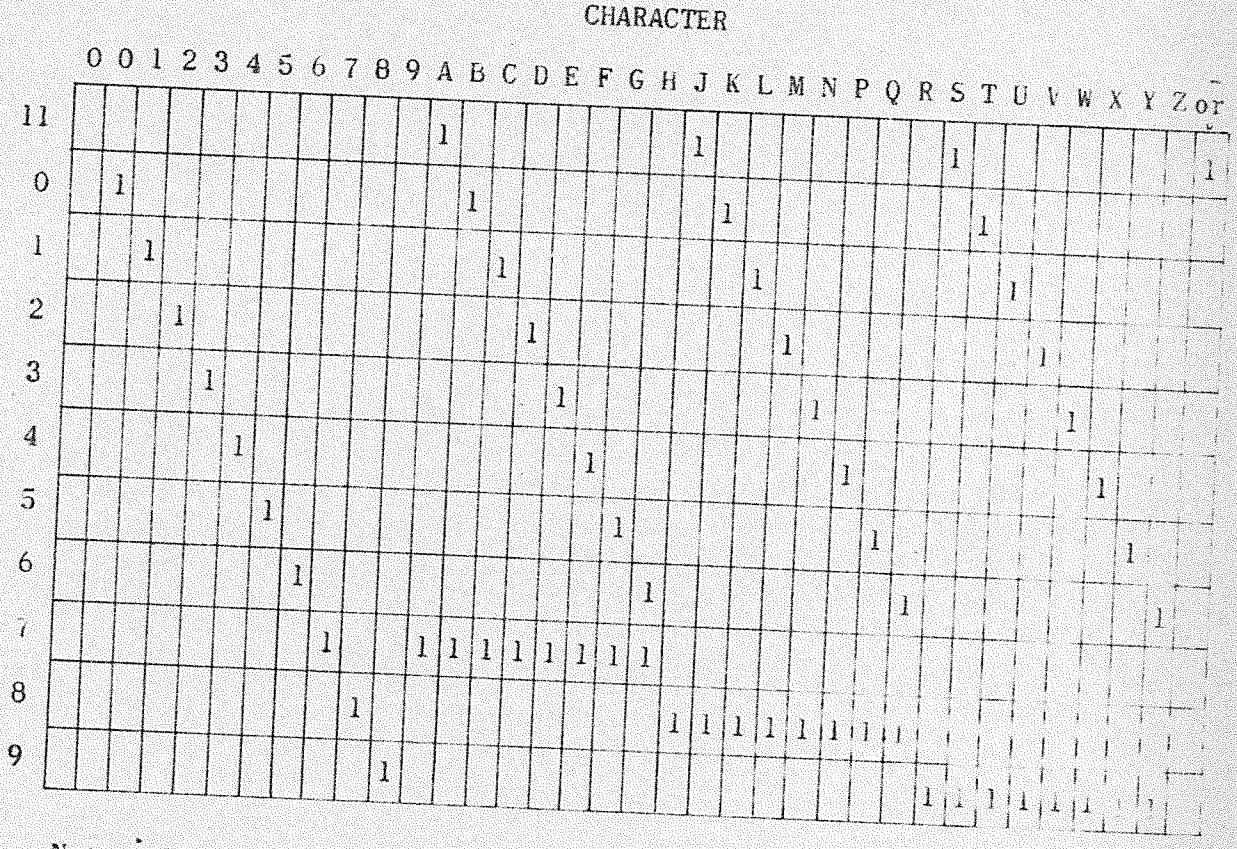
Perform Typewriter Operation	Octal
Space	04
Shift up	47
Shift down	57
Back space	61
Car. return	45
Tabulator	51
Color shift	02

Type Symbol UC	Type Symbol LC	Octal
⁻ (Superscript Minus)	- (Hyphen or Minus)	36
· (Multiply)	= (Equals)	14
/ (Virgule)	+ (Plus)	54
((Open Parens)	, (Comma)	46
) (Close Parens)	. (Period)	42
_ (Underline)	| (Absolute)	50

PX 71871

manually set for one, two, or three spaces. Each line may have any of the allowable characters printed in any of 92 columnar positions. Each column has an associated character wheel on which 35 characters are available for printing. Each of the 34 numbers and letters below are present on each character wheel. The period is present on the even-numbered character wheels, and the minus sign is present on the odd-numbered character wheels, thus allowing a period and a minus sign to be printed in every other columnar position on a line. The characters to be printed in given columnar positions on a given line are chosen according to an associated 11-row, 92-column coded image as assumed in the computer. Also, up to a maximum of nine zeros may be chosen to be printed in any columnar positions, starting immediately to the right of a coded printed character, by making mechanical settings on the printer.

The image has the following coding for available characters:



Note: Blank spaces denote zeros in code as stored in the computer.

Each row of the image is formed by positioning at three addressed computer locations the bits for that row which are part of the code of the characters which are to be printed. Thus, thirty-three addressed locations are required to represent an image. The image is divided into three fields: Field I consisting of the first or left-most 36 columns; Field II, the second 36 columns; and Field III, the third or right-most 20 columns. Three External Write instructions, which must be programmed consecutively, transmit the coded information of each row of the image to the IOB Register; or, as follows:

PX 71871

EWjv (v)→X, (X)→IOB; (v_{35} ... v_0) being Field I of Row r

EWjv (v)→X, (X)→IOB; (v_{35} ... v_0) being Field II of Row r

EWjv (v)→X, (X)→IOB; (v_{35} ... v_{16}) being Field III of Row r

The three instructions are repeated for each of the 11 rows, the rows being transmitted in the order r = 9, 8, ... 1, 0, 11. All three instructions must be executed even though there may be no information in all three fields to be printed.

Between successive External Writes the current contents of IOB must be transmitted to the Line Printer, allowing IOB to be cleared. Programming a properly coded External Functions instruction to precede the External Write instructions initiates the operation of the Line Printer, allowing it to receive information from IOB. Thus, for controlled operations of the Line Printer, the External Function instruction places "ones" digits in selected stages of IOB. The stages selected, with the operations of the Line Printer which are initiated by the EF instruction and the presence of ones in these stages, are as follows:

IOB_{34} - Start (master bit)
Causes one cycle of operation of the Line Printer and the automatic advance of the paper unless the Stop Paper or the Jump selections were also made.

IOB_{10} - Print
Permits one line of characters (an 11 row image) to be sensed and printed during a cycle of operation.

IOB_{9} - Stop Paper
Stops the advance of the paper during a cycle of operation.

IOB_{13} - Jump
Causes the paper to advance nine spaces during a cycle of operation (No printing permissible during this cycle.)

IOB_{12} - Skip
Causes the paper to advance according to the Format Switchboard settings. (A Print selection is not permissible during the same EF instruction.)

Each cycle of operation of the Line Printer is divided into 15 points, each point being 26.67 milliseconds in duration. During each of the first 11 points, (9, 8, ..., 1, 0, 11) of the cycle, a row of the image is sensed by the Printer providing that three External Write instructions are executed during this time and the IOB Print (and IOB Start) selections were made by a previous External Functions instruction. If these selections are in effect, the series of 11 row transmissions positions each of the 92 character wheels chosen with these characters being printed between the sixth and fifteenth points of a cycle. The paper is advanced automatically in the Printer during the first two to four points of a cycle, provided that the IOB Stop Paper stage was not selected by the previous EF instruction.

Time factors that must be taken into consideration in programming for Line Printer output are pointed out in the following program outline. A time lag due to inertia when the Printer is initially started may occur which will delay the execution of the first point of the cycle of operation. Note that an External Write of Field I for Row 9 should immediately follow the External Function instruction which starts the Printer and, for Rows 8, ... 0, 11 should occur before or during the early part of the corresponding ten points. Other computer instructions may be programmed and executed between cycles and during cycle points in the remaining available computer time.

To advance the paper and print one line:

```
EF-v                (v) = Start Line Printer
                          Print
```

Within 1.5 ms of the beginning of the point the first EW must be executed

```
EWjv                Write Field I
EWjv                Write Field II
EWjv                Write Field III
Other instructions
```

} point 9, 26.67 ms

Within 1.5 ms of the beginning of each point, the first EW must be executed

```
EWjv                Write Field I
EWjv                Write Field II
EWjv                Write Field III
Other instructions
  .                   .
  .                   .
  .                   .
```

} Repeat for points 8, 7 ... 0; each 26.67 ms

Within 1.5 ms of the beginning of the point, the first EW must be executed

```
EWjv                Write Field I
EWjv                Write Field II
EWjv                Write Field III
Other instructions

Other instructions
```

} point 11, 26.67 ms

} points 12, 13, 14, 15, each 26.67 ms

Continuous printing of lines may be programmed by repeating the preceding program the desired number of times with the instruction initiating each repeating cycle being programmed during point 12 of the preceding cycle; i.e., for continuous operation of the Line Printer, the External Function instruction to initiate a new cycle must be executed within 50 milliseconds of the Write Field III instruction for row 11.

PX 71871

By making the appropriate switch settings on the Format Switchboard, printing lines in accordance with a chosen format may be accomplished without executing External Functions instructions to initiate individual spacing operation of one, two, three, or nine lines. Skips of one, two, three, six or nine spaces during one printer cycle, or consecutive skips of any combination of these spacings during successive printer cycles, may be initiated by a single computer instruction, after which the computer is free to execute any other instructions.

The ten columns of switches on the Format Switchboard provide a "counting" facility for the changes of format available. The functions of the three portions of the switchboard, Control, Space, and Multiply, and their switch settings are explained below:

1 CONTROL. - The switch in the Hold position holds computer control of the printer in effect until the end of the current printer cycle initiated by an External Functions instruction containing Start and Skip IOB Select bits. The switch in the Advance position, in conjunction with Skip and Start bits in the preceding EF, releases computer control of the printer and advances the paper in the printer according to switch settings on the Space and Multiply portions of the switchboard. Adjacent switch settings of Advance in the Control portion will automatically continue the format control count across the switchboard with the function being executed according to the corresponding switch settings in Space and Multiply. The switch in the Clear position releases the counting facility and returns the format control count to the "one" or "home" position.

2 SPACING. - Spacing between lines is controlled during both computer controlled or format controlled operations. Under format control the paper is advanced the number of spaces according to the switch settings in both the Space and Multiply portions of the switchboard.

3 MULTIPLY. - Under format control the paper is advanced, one, two, or three spaces (according to the Space setting) if the Multiply switch is in the Off position; or three, six, or nine spaces (each space setting multiplied by three) if the Multiply switch is in the On position.

The following listing shows the optional Line Printer functions available under computer control and format control. If no format is being followed for printing operations, the format control "count" should be in the "one" position (this is insured by engaging the Clear control on the Format Switchboard for at least two seconds). The Format Switchboard settings below are for this "home" position.

PX 71871

Under Computer Control:

	FUNCTION	IOB SELECT BITS	FORMAT SWITCHBOARD SETTINGS CONTROL	SPACE	MULTIPLY
1.	Start, advance paper, and print (if desired)	Start Print (if desired)	Hold	1,2, or 3	OFF*
2.	Start, stop advance of paper, and print(if desired)	Start, Stop Paper Print (if desired)	Hold	1,2, or 3*	OFF*
3.	Start and jump 9 spaces without printing	Start, Jump	Hold	1,2, or 3*	OFF*

Under Format Control:

	FUNCTION	IOB SELECT BITS**	FORMAT SWITCHBOARD SETTINGS CONTROL	SPACE	MULTIPLY
4.	Single, Double, or Triple Space (plus automatic advance)	Start, Skip	Advance	1,2, or 3	OFF
5.	Space six or nine (plus automatic advance)	Start, Skip	Advance	2 or 3	ON
6.	Advance format control "count" to next position	Start, Skip	Clear	1,2,3*	OFF*

* These switches, although not necessary in setting up the function, should be in the "down" position.

** Included in a previous EF used for computer controlled operation.

Functions 1,2,3,4, and 5 require timewise one print cycle.

Function 6 requires approximately one second to clear to the "home" position.

Note that before printing the first copy in a chosen format, the paper must be adjusted one, two, or three spaces ahead of the position in which the first printing is desired unless the first EF instruction contains a Stop Paper bit. Note that under format control if it is desired to restrict the paper advance to what is indicated by the format control settings, the Stop Paper selection must be included in the EF instruction containing the Start and Skip selections. Also, the following conditions require that the Clear control on the Format Switchboard be engaged for at least two seconds to insure that the format control count is back in the home position. These conditions are

1. Paper is being changed.
2. A fault occurs.
3. The format is changed.
4. An improper program is scheduled.

PX 71871

An optional format for Line Printer output is illustrated in Figure 6-2. The form length used is 66 spaces, generally the number of spaces on the folded pack paper commonly used. The following listing shows the selections necessary to achieve the printing of a page in this format. (Each External Function instruction must be followed by the appropriately coded External Write instructions.)

FORMAT CONTROL "COUNT"	FORMAT SWITCHBOARD SETTINGS CONTROL	SPACE	MULTIPLY	SPACES ADVANCED	IOB SELECT BITS (in a total of 27 coded EF-v instructions)
1	Hold	1	OFF	18	Start, Print (repeat this coded EF 18 times)
2	Advance	3	ON	9	Start, Stop Paper, Skip
3	Advance	3	ON	9	
4	Hold	3	OFF	21	Start, Print (repeat this coded EF 7 times)
5	Advance	3	ON	9	Start, Stop Paper, Skip
6	Clear	1	OFF	0	
7	Clear	1	OFF	0	
8	Clear	1	OFF	0	
9	Clear	1	OFF	0	
10	Clear	1	OFF	0	

Total of 66 spaces

During the operation of the Line Printer four types of faults may occur which are indicated visually by a light on the printer or a fault indication on the Supervisory Control Panel of the computer. These faults are as follows:

1. NO PAPER FAULT. - If the paper in the Line Printer is almost depleted, a fault circuit stops the computer and gives a No Paper fault indication on the printer control panel. Pressing the Clear button on the printer clears the fault; starting computations anew will initiate a second fault condition. Since a No Paper fault occurs slightly before the end of the paper has passed the printing position, several more lines, depending on the spacing, may be printed. This fault becomes effective after the current cycle is completed, thus no information will be lost.

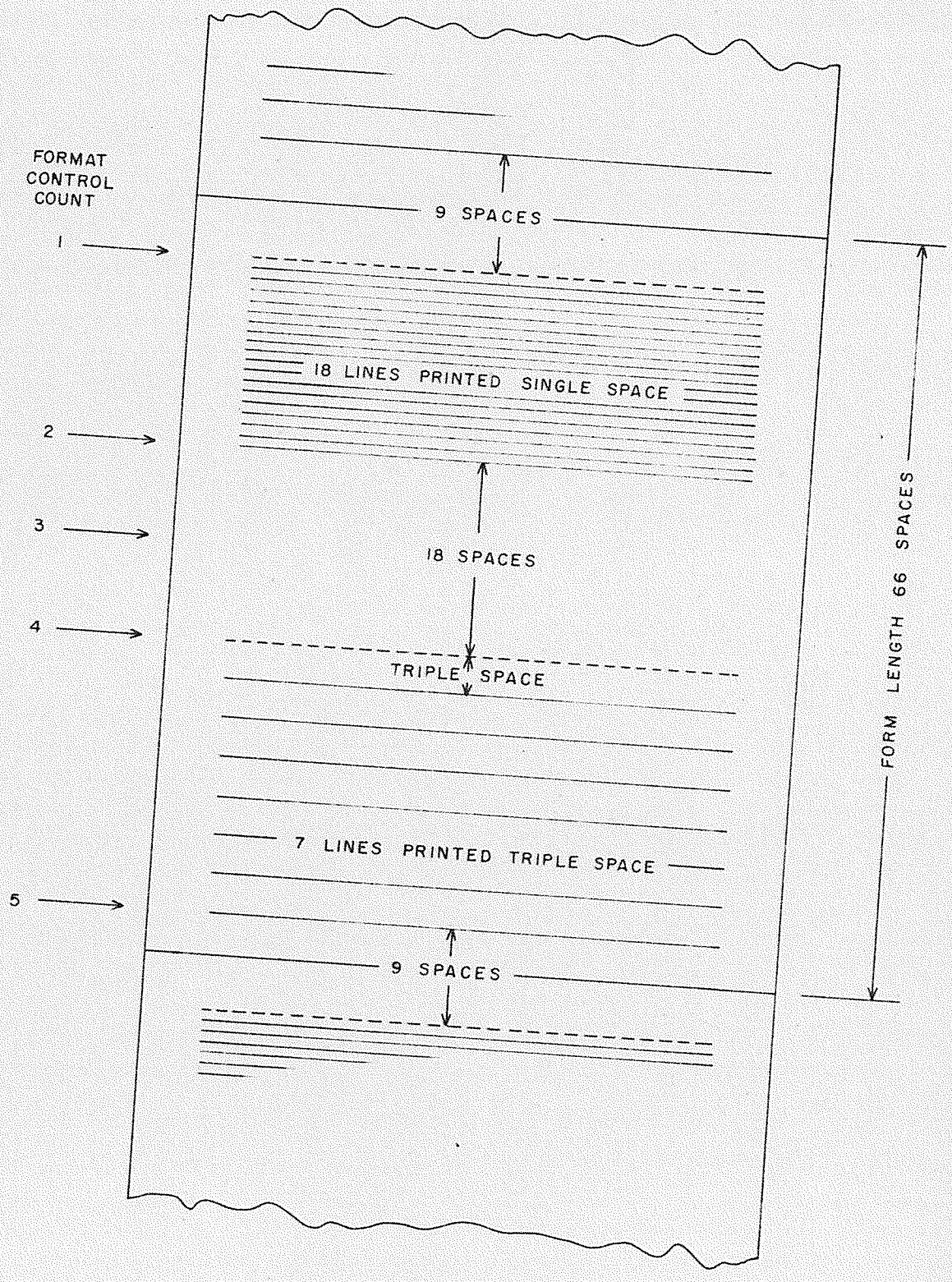


Figure 6-2. Optional Format for Line Printer Output

2 MECHANICAL JAM FAULT. - If the printer stops before completing a cycle, a B Fault stops the computer and printer and is indicated by the IO light on the Supervisory Control Panel. There is a possibility that this fault may be cleared by pressing the Master Clear button on the computer.

3 NO INFORMATION FAULT. - If IOB does not contain the Field I portion of a row (i.e., the first EW instruction has not been executed) within 1.5 milliseconds after the start of a cycle point, a B Fault occurs. The Line Printer completes the cycle before stopping but will print erroneously since no further EW instructions occur within the cycle.

4 OVERHEAT FAULT. - This fault will cause an A Fault computer stop. Operations may be resumed temporarily without loss of information by setting the Bypass Interlock switch on the Supervisory Control Panel, pressing the Clear A Fault button, and pressing the Start button on the computer.

(3) INPUT AND/OR OUTPUT - CONTROLLED REPRODUCER. - The Controlled Reproducer Input and/or Output System utilizes the Reproducer, as shown on Plate 6-6, the Reproducer Access Control, the IOA and IOB registers, and the IOA and IOB control circuitry. The media of representation is 80-column punched cards which may be punched or read at a rate of 120 cards per minute. The Reproducer, as used in operation with the 1103, utilizes a left-hand channel for punching cards and a right-hand channel for reading cards. Punched cards whose information content is to be transmitted to the computer are routed through the "read" channel of the Reproducer; blank cards which are to receive information and to be punched accordingly are routed through the "write" channel. Cards to be routed through the read channel are placed in the card read feed hopper and received, after their advancement through the channel, by the read receiving stacker. Cards to be routed through the write channel are placed in the card punch feed hopper and received, after their advancement through the channel, by the punch receiving stacker.

The tabulating card used by the Reproducer is divided into 12 horizontal rows and 80 vertical columns with groups of these columns being designated as Fields I, II, and III as shown in the diagram of an unpunched card in Figure 6-3. A rectangular hole, or index, may be punched at the intersection of any row and column. If the input or output information is coded to represent binary words, each row may be punched to represent two 36-bit words plus six additional bits of information. (A hole represents a one; the absence of a hole denotes a zero.) Tabulating cards may also be punched in a pattern to represent alphabetical and digital characters. An optional punched pattern of letters and decimal digits is shown in Figure 6-4.

Information, binary words or otherwise, is transmitted to or from the cards in response to programmed External Function and External Write or External Read instructions. A series of three of these EW or ER instructions, programmed consecutively, directs the transmission of each row of information, via X, between addressed computer locations and the Input-Output Registers, IOA and IOB. The sequence of information flow is as follows:

PX 71871

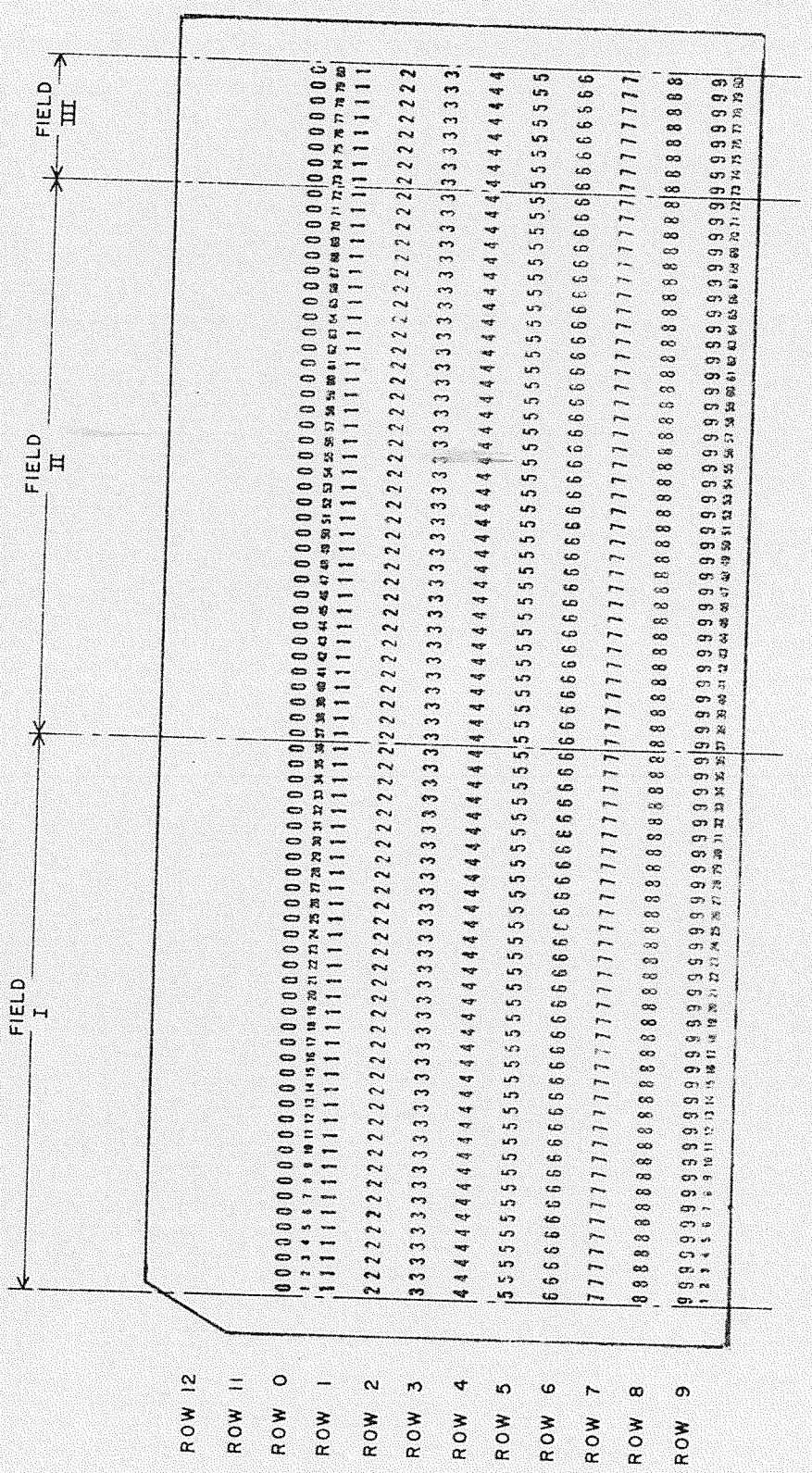
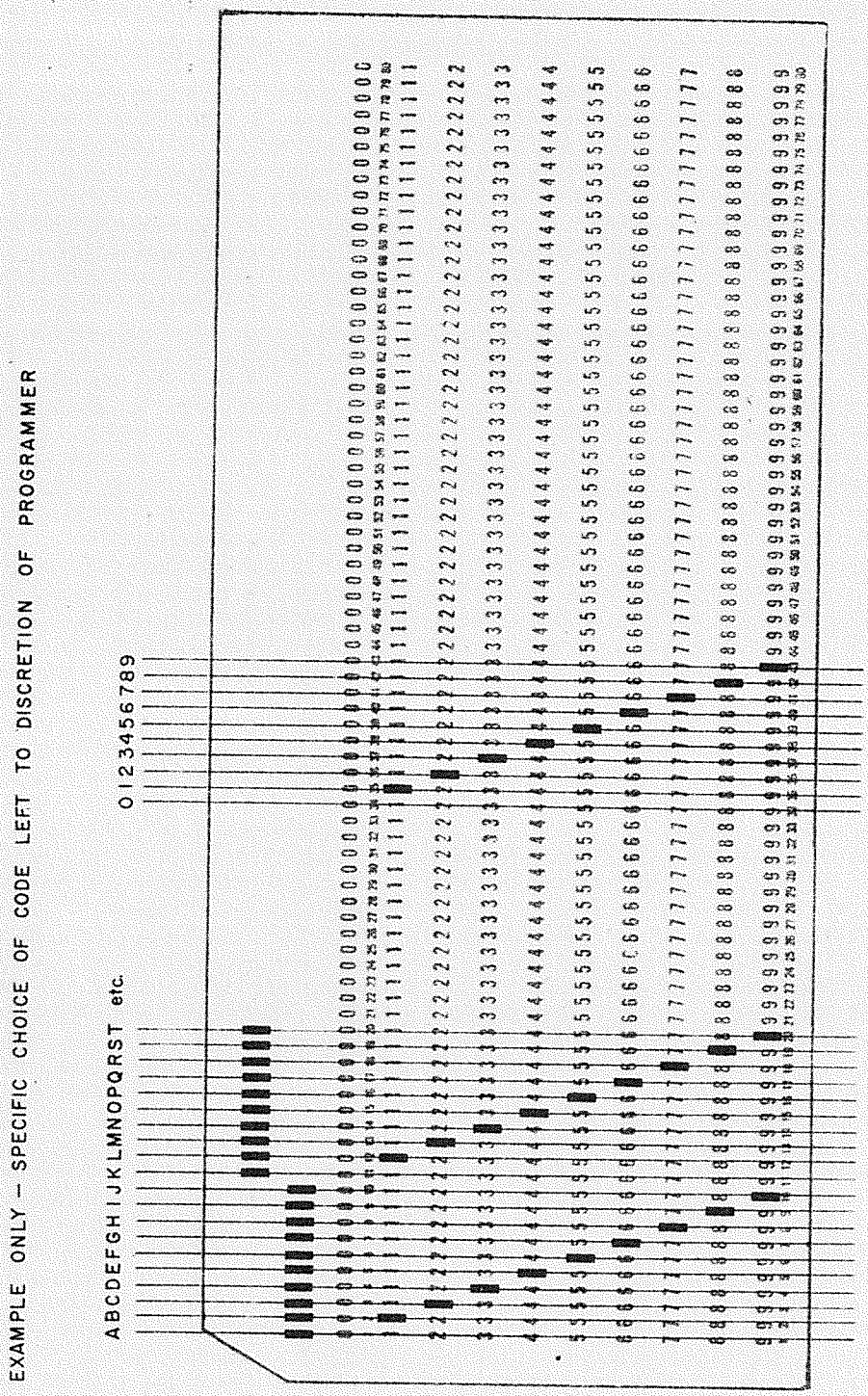


Figure 6-3. Tabulating Card Fields, Columns, and Rows

PX 71871



PX 71871

Figure 6-4. An Example of Punched Card Alphabetical and Digital Representation

To transmit information from the computer to a tabulating card, the group of following External Write instructions, preceded by appropriately coded External Function instructions, is executed twelve times, once for each card row:

EWjv, j = 0 (X) → IOA → Field III
EWjv, j = 1 (X) → IOB → Field I
EWjv, j = 1 (X) → IOB → Field II
} Row 9, 8, ... 0, 11, 12

To transmit information from a punched tabulating card to the computer, the group of following External Read Instructions, preceded by appropriately coded External Function instructions, is executed twelve times, once for each card row:

ERjv, j = 0
ERjv, j = 1
ERjv, j = 1
Row 9, 8, ... 0, 11, 12 {
Field III → (IOA) → X
Field I → (IOB) → X
Field II → (IOB) → X

Information transmission between IOA and Field III is optional and may be omitted by not executing EW or ER with j = 0 and by manually setting a switch on the Reproducer Access Control Cabinet to its up position. For reading and writing of all three fields the normal position of this switch, called the Enable Field III switch, is down.

Preceding and necessary for any reading and writing operations by the Reproducer, at least one External Function instruction must be executed to start the Reproducer, position the cards for reading and/or punching (these two conditions may be established also by manual operation), and establish the condition necessary for a read and/or write. The External Function EF-v transmits representations of ones to certain stages of IOB which means that, according to the selected stages, the following signals may be sensed by the Reproducer.

IOB_0	Read
IOB_1	Punch
IOB_2	Pick Reader Card
IOB_3	Pick Punch Card
IOB_4	Stop (and Drop Selections)
IOB_5	Run Free
IOB_6	Start Cycle

Start Cycle - This selection must be present in each EF-v to make any of the other selections effective. A signal from IOB_6 engages the drive mechanism of the Reproducer and causes one "cycle" of operation. The time consumption of each cycle is 500 milliseconds with each cycle consisting of a sequence of 18 points of approximately 27.8 milliseconds each. The points are numbered in the order (14) 15, 16, 17, 18, 9, 8,...1, 0, 11, 12, 13, 14.

PX 71871

Run Free - A signal from IOB_5 causes continuous cycle operation by retaining in effect the Start selection, along with other selections made simultaneously with the Run Free, until a Stop signal is received by the execution of another programmed EF-v.

Stop - A signal from IOB_4 permits only one more cycle of operation with the selections effective that were made by the previous EF-v.

Pick Punch Card - A signal from IOB_3 causes the bottom-most card in the punch card feed hopper to be withdrawn from the hopper and placed in the punch channel.

Pick Reader Card - A signal from IOB_2 causes the bottom-most card in the read card feed hopper to be withdrawn from the hopper and placed in the read channel.

Punch - A signal from IOB_1 enables the punch mechanism to receive information from IOB (and IOA) and prepares it to punch this information in a card during the next cycle.

Read - A signal from IOB_0 enables the brushes used for reading to sense the information in each row of a card as it passes beneath them.

After these IOB select signals are received by the Reproducer they are held in effect for one cycle of operation, or a number of cycles if a Run Free Select was chosen, by the Reproducer control circuitry so that IOB may be cleared for information transmission.

After a card is picked and withdrawn from either the punch card feed hopper or the read card feed hopper, it advances through a series of five positions or "stations" in the read channel or punch channel. A Pick Reader Card signal or a Pick Punch Card signal places a card in either Read Station 1 or Punch Station 1. These operations may be initiated by programming External Function instructions with bits in the appropriate stages of IOB or by manually operating switches on the Reproducer itself. After a card is in Station I, in either the punch or read channels, five cycles of operation are necessary to advance the card through the channel and into its final position in the receiving stacker. Each cycle of operation advances cards in both channels, regardless of their position, to the next station.

Punch Station 3 in the punch channel contains the punch mechanism. After a card is in Punch Station 1, two cycles of operations are needed to advance the card to the punch station. During the cycle which advances the card from Punch Station 2 to Punch Station 3 the information to be punched must be received from IOB (and IOA). More explicitly, during each point 9, 8, ..., 1, 0, 11, 12, of 27.8 milliseconds, the series of three (or two) External Write instructions should be executed consecutively, and the Punch select signal should be in effect so that information to be punched (in the correspondingly numbered card rows 9, 8, ..., 0, 11, 12) can be transmitted to the punch mechanism. At the beginning of the next cycle automatic punching of a complete card occurs before the card is advanced to Station 4. Outlines of programming for punching of consecutive cards are given in Tables 6-2 and 6-3.

PX 71871

TABLE 6-2. WRITE - SINGLE OR CONSECUTIVE CARDS

The computer instructions below withdraw a single card from the write card feed hopper, position it for punching and punch information in it, and continue advancing it through the Reproducer until it reaches its final position in the receiving stacker. Other instructions may be programmed and executed during and between each cycle of operation of the Reproducer providing that the timing requirements noted are met.

EF-v	(v) = Start Pick Punch Card	1 cycle
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start Punch Within 112.7 ms of the start of this cycle the execution of the following three instructions should be initiated:	1 cycle
EW,0,v EW,1,v EW,1,v	Repeat for each card row, each repetition being initiated not later than 1.5 ms after the beginning of the corresponding point.	
EF-v	(v) = Start (punching occurs)	1 cycle
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start (channel cleared of last punched card)	1 cycle

To transmit information to n consecutive cards, without selecting the Run Free bit, include Pick Punch Card bits in (v) of the second and third External Function instructions above. Repeat the third EF-v instruction and the group of External Writes occurring in that cycle n times. Then at the conclusion of the program above, the nth or last punched card will be found in the punch stacker with cards remaining in the fourth and fifth punch stations.

PX 71871

TABLE 6-3. WRITE - FREE RUN

The computer instructions below withdraw singly a sufficient number of cards from the write card feed hopper to hold the information to be punched, position them for punching and punch information in them, and continue advancing them through the Reproducer until the last card picked reaches its final position in the receiving stacker. Other instructions may be programmed and executed during and between each cycle of operation of the Reproducer providing that the timing requirements noted are met.

Number n is the number of cards required to hold the information to be written.

EF-v	(v) = Start Pick Punch Card	1 cycle
EF-v	(v) = Start Pick Punch Card	1 cycle
EF-v	(v) = Start Free Run Pick Punch Card Punch Within 112.7 ms of the start of this cycle the execution of the following three instructions should be initiated:	
EW,0,v EW,1,v EW,1,v	Repeat for each card row, each repetition being initiated not later than 1.5 ms after the beginning of the corresponding point. The repetitive group of External Write instructions must be programmed n-1 times, each group being executed timewise within the cycle which enables the punching.	repeated n-1 cycles
EF-v	(v) = Start Stop (punching of n-1 card occurs) Within 112.7 ms of the start of this cycle the execution of the following three instructions should be initiated:	1 cycle
EW,0,v EW,1,v EW,1,v	Repeat for each card row, each repetition being initiated not later than 1.5 ms after the beginning of the corresponding point.	
EF-v	(v) = Start (n^{th} card punched)	1 cycle
EF-v	(v) =Start	1 cycle
EF-v	(v) = Start (n^{th} card placed in receiving stacker)	1 cycle
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start (channel cleared)	1 cycle

PX 71871

Read Station 2 in the read channel contains the brushes used for sensing the cards. After a card is in Read Station 1, one cycle of operation is necessary to advance the card into the read station. During this cycle the information is read, a row at a time, as the card is advanced past the brushes. This information must be transmitted to the computer from IOB (and IOA) as it is received. More explicitly, during each point 9, 8,... 1, 0, 11, 12 of 27.8 milliseconds, the series of three (or two) External Read instructions should be executed consecutively and the Read select signal in effect to enable the transmission to addressed storage locations of the information of the correspondingly numbered rows 9, 8,... 1, 0, 11, 12. Outlines of programming for reading of consecutive cards are given in Tables 6-4 and 6-5.

An outline of programming for simultaneous reading and writing of consecutive cards is given in Table 6-6 (and p.6-127). The transmission of Field III in any of the programs may be eliminated by not executing the External Read or Write instructions with j = 0 and by setting the Enable Field III switch properly.

The equipment is prepared for controlled operation, after power is applied at the Access Control Cabinet, by following the steps below:

Step 1 - If only 72 card columns are to be read or punched, set the FIELD III switch, S01, to the up position. If 80 columns are to be read or punched, set this switch down.

Step 2 - If card reading is to be performed, place the deck of cards to be read into the right-hand feed hopper of the reproducer. If card punching is to be performed, place a deck of cards in the left-hand feed hopper. Place the metal weights on top of the decks.

Step 3 - Set the reproducer toggle switches in the following manner:

DUPL.	away from operator
PUNCH	away from operator
MOTOR	left
DC	left
READ	away from operator
PICK READ	towards operator
PICK PUNCH	towards operator
STANDBY	towards operator

After these steps have been performed, the punched card program may be started in the computer.

The reproducer may be operated manually by switches and pushbuttons on the panel located below the feed hoppers. The procedures are as follows:

PX 71871

TABLE 6-4. READ - SINGLE OR CONSECUTIVE CARDS

The computer instructions below withdraw a single card from the read card feed hopper, position it for reading and transmit its contents to the computer, and continue advancing it through the Reproducer until it reaches its final position in the receiving stacker. Other instructions may be programmed and executed during and between each cycle of operation of the Reproducer providing that the timing requirements noted are met.

EF-v	(v) = Start Pick Read Card	1 cycle
EF-v	(v) = Start Read Within 112.7 ms of the start of this cycle the execution of the following three instructions should be initiated:	1 cycle
ER,0,v ER,1,v ER,1,v	Repeat for each card row, each repetition being initiated not later than 1.5 ms after the beginning of the corresponding point.	
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start (channel cleared of last read card)	1 cycle

To transmit information from n consecutive cards, without selecting the Run Free bit, include a Pick Read Card bit in (v) of the second External Function instruction above and repeat that EF-v instruction and the group of External Reads occurring in that cycle n times. Then at the conclusion of the program above, the nth or last read card will be found in the read stacker with an n + 1 card remaining in the fifth read station.

PX 71871

TABLE 6-5. READ - FREE RUN

The computer instructions below withdraw singly from the read card feed hopper the cards to be read, position them for reading and transmit their information content to the computer, and continue advancing them through the Reproducer until the last card picked reaches its final position in the receiving stacker. Other instructions may be programmed and executed between and during each cycle of operation of the Reproducer providing that the timing requirements noted are met.

Number n is the number of cards to be read from.

EF-v	(v) = Start Pick Read Card	1 cycle
EF-v	(v) = Start Free Run Pick Read Card Read Within 112.7 ms of the start of this cycle the execution of the following three instructions should be initiated:	
ER,0,v ER,1,v ER,1,v	Repeat for each card row, each repetition being initiated not later than 1.5 ms after the beginning of the corresponding point. The repetitive group of External Read instructions must be programmed n-1 times, each group being executed timewise within the cycle which enables the reading.	repeated n-1 cycles
EF-v	(v) = Start Stop (reading of n^{th} card) Within 112.7 ms of the start of this cycle the execution of the following three instructions should be initiated:	1 cycle
ER,0,v ER,1,v ER,1,v	Repeat for each card row, each repetition being initiated not later than 1.5 ms after the beginning of the corresponding point.	
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start (n^{th} card placed in receiving stacker)	1 cycle
EF-v	(v) = Start (channel cleared)	1 cycle

PX 71871

SIMULTANEOUS READ AND WRITE - SINGLE OR CONSECUTIVE CARDS

The outlined programs for reading and writing single or consecutive cards simultaneously may be deduced by interposing the two outlined programs for read and write, single or consecutive cards. A composite of the selections made in both programs in each individual cycle would be selected for one cycle of the simultaneous operations. If it is desired that the reading occur during the same cycle that the information to be punched is being received by the Reproducer, the second cycle of operation of the program for writing should be interposed with the first cycle of operation of the reading program. External Read instructions immediately precede External Write instructions when both occur during the same cycle.

The same timing considerations must be given the simultaneous performance of reading and writing as when they occur individually: other instructions may be programmed and executed between and during each cycle of operation providing that the timing requirements noted are met.

TABLE 6-6. SIMULTANEOUS READ AND PUNCH - FREE RUN (Cont'd)

The computer instructions below withdraw cards from the read card feed hopper and punch card feed hopper, position them for reading and writing and perform the information transmittal, and continue advancing them through their individual channels until the last cards picked reach their final positions in the receiving stackers. Other instructions may be programmed and executed between and during each cycle of operation of the Reproducer providing that the timing requirements noted are met.

Number n is the number of cards to be read from and the number of cards required to hold the information to be written.

EF-v	(v) = Start Pick Punch Card	1 cycle
EF-v	(v) = Start Pick Punch Card Pick Read Card	1 cycle
EF-v	(v) = Start Free Run Pick Read Card Read Pick Punch Card Punch Within 112.7 ms of the start of this cycle the execution of the following six instructions should be initiated:	
ER,0,v ER,1,v ER,1,v EW,0,v EW,1,v EW,1,v	Repeat for each card row, each repetition being initiated not later than 1.5 ms after the beginning of the corresponding point. The repetitive group of External Read and External Write instructions must be programmed n-1 times, each group being executed timewise within the cycle which enables the reading and writing.	repeated n-1 cycles

PX 71871

TABLE 6-6. SIMULTANEOUS READ AND PUNCH - FREE RUN (Concl.)

EF-v	(v) = Start Stop Within 112.7 ms of the start of this cycle the execution of the following six instructions should be initiated: ER,0,v ER,1,v ER,1,v EW,0,v EW,1,v EW,1,v Repeat for each card row, each repetition being initiated not later than 1.5 ms after the beginning of the corresponding point.	1 cycle
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start	1 cycle
EF-v	(v) = Start (n^{th} card placed in write receiving stacker)	1 cycle
EF-v	(v) = Start (n^{th} card placed in read receiving stacker)	1 cycle
EF-v	(v) = Start (channels cleared)	1 cycle

PX 71871

(a) TO START. - If cards are in the feed hoppers and it is desired to drive the reproducer through one or more card cycles, press the START pushbutton.

If no cards are in the hoppers and it is desired to run the reproducer, to empty cards from the channels, etc., press and hold both the CLEAR and START pushbuttons.

If the START button is pressed and released immediately, the reproducer will drive through one complete card cycle. If the button is held down, the reproducer will run continuously.

(b) TO PICK PUNCH CARDS. - If it is desired to feed cards into the punching channel, set toggle switch PICK PUNCH CARDS to the On position (away from the operator), then press and hold down the START pushbutton until the desired number of cards have fed.

(c) TO PICK READ CARDS. - If it is desired to feed cards into the reading channel, set toggle switch PICK READ CARDS to the On position (away from the operator), then press and hold down the START pushbutton.

(d) TO STOP. - To stop the reproducer during a controlled run so that cards may be added to the feed hoppers, etc., either set toggle switch STANDBY to the On position (away from the operator) or press and hold down the red STOP pushbutton. This causes the reproducer to stop at the end of the current card cycle so that the computer program is temporarily halted. If the toggle switch is reset or the red STOP pushbutton is released the system resumes normal operation.

When certain faults occur in the system, the reproducer stops. Certain of these faults result in B Fault computer stops which allow the reproducer to complete its current cycle before it stops. A few of these faults are described below.

(a) OVERHEAT. - If the temperature in any portion of the Access Control Cabinet rises above 100°F, a computer A Fault stop occurs and the amber OVERHEAT indicator on the end of the Access Control Cabinet glows. The Overheat fault in itself will not cause a reproducer stop but it may lead to a No Information A Fault which causes a B Fault computer stop and a reproducer stop.

(b) NO INFORMATION. - If the computer program specifies that punching and/or reading is to be executed but insufficient or tardy External instructions are being executed, a B Fault computer stop occurs and the amber NO INFORMATION indicator on the end of the Access Control Cabinet glows. The computer program must be restarted from the beginning if this occurs.

PX 71871

(c) VOLTAGE FAULT. - If power to the reproducer drive motor fails, the VOLTAGE FAULT indicator on the end of the Access Control Cabinet glows. The fault may be produced by faulty wiring or by blown fuses.

(d) NO CARD IN READER. - If card reading is supposed to occur and no card is present in the reading station, a B Fault computer stop is produced and the NO CARD IN READER indicator on the Access Control Cabinet glows. This fault may be caused by failure to feed cards or by errors in the computer program. When the source of error is removed, the computer program must be restarted from the beginning.

(e) NO CARD IN PUNCH. - If card punching is supposed to occur in the next cycle and no card will be present beneath the punch die, a B Fault computer stop is produced and the NO CARD IN PUNCH indicator on the Access Control Cabinet glows. This fault may be caused by a feed failure or by errors in the computer program. When the source of error is removed, the computer program must be restarted from the beginning.

(f) PUNCH JAM. - If a card jams under the edge of the punching die, the reproducer stops, all voltages to the reproducer are dropped, and the small red JAM indicator on the reproducer glows. If this occurs, shut off the equipment power.

(g) STOP. - A stop light on the end panel glows and the reproducer stops if any of the following occur:

1 Output read stacker full.

2 Output write stacker full.

3 Input read feed hopper empty.

4 Input punch feed hopper empty.

5 The Stop button is pressed.

6 Standby Switch is thrown to forward position (away from operator).

7 A Punch Jam occurs.

Note that all of these stops except 7 permit resumption of operation without loss of data. For stops caused by 3 or 4, move Standby to forward position during refill of input hoppers. Note that during card reading or card punching operations, the hopper not being used for such operations must contain at least one card to prevent the occurrence of the stop caused by 3 or 4 above.

PX 71871

5. OPERATING THE COMPUTER.

a. GENERAL. - The 1103 computer is set into operation by certain combinations of selections made on the Supervisory Control Panel, illustrated on Plates 6-7 through 6-11. The computer may be set into High Speed operation, or, if it is desired to manually superintend the internal actions of the computer and/or have a visual presentation of these internal actions, Step operation may be chosen. Those internal operations which are represented on the Supervisory Control Panel, or which occur in components of the computer represented on the Supervisory Control Panel, have as their visual counterpart the occurrence of lights in the corresponding designated positions. Thus the contents of any of the registers represented may be noted by interpreting the double rows of lights into a bi-octal code as follows: a light in the upper row represents a binary one in the stage as numbered, and a light in the lower row represents a binary zero in that stage; thus each group of three columns of lights represents an octal number.

In addition to the ability to oversee the operations of the computer as it executes a program already internally stored, operations may be performed upon information placed in the computer by manually setting it in the counterparts of the proper registers on the Supervisory Control Panel. The small white button at the lower right end of each register is depressed to clear the register (lighting the lower row indicators); the small black button below each column of two lights is depressed to place a "one" in the chosen stage of the register (lighting the upper row indicator). By following the proper procedure, the contents of these registers may then be used as desired in computer operations.

Also, by following the proper procedure, instructions may be manually placed in the computer by inserting them in the Supervisory Control Panel counterpart of the PROGRAM CONTROL REGISTER (MCR, UAK, and VAK), and setting the MAIN PULSE DISTRIBUTOR to zero. If MPD is set at six, the first instruction to be executed is taken from the address shown in PAK (as automatically or manually set).

In discussing in the following text the operating selections which are made on the Supervisory Control Panel, the "group" designations listed below will be used, each group being represented on the control panel by a set, enclosed in white lines, of pushbuttons, switches, and/or indicator lights: As located on the lower center section of the Supervisory Control Panel, Plate 6-8, from left to right -

Operating Rate group
Clock Source group
Operation Mode group
Start Selection group
Selective Jumps group
Selective Stops group
A/B Fault group (upper)
"Operating" group (lower)

As located on the right section of the Supervisory Control Panel, Plate 6-10, from left to right -

Fault Indicators group
Test Switch group

The computer may be operated in one of two modes: NORMAL or TEST. For each mode, selections must be made by manually depressing the desired button (or automatically by a foregoing selection) in the groups, Operating Rate, Clock Source, and Operation Mode, Plate 6-8. Selections made are indicated by a light above the proper lettering. Operating in the NORMAL mode requires an additional selection in the Start Selection group: a selection of MD START is necessary to start operations with the first instruction to be executed taken from PAK; and a selection of MT START is necessary to start operations with the Magnetic Tape being used as the source of the instructions to be executed. Operating in TEST mode requires an additional manual selection in the Operating Rate group: HIGH SPEED, one of the three buttons labeled MANUAL STEP, or either of the buttons labeled AUTOMATIC STEP. The AUTOMATIC STEP RATE switch controls the time rate at which Automatic Step Operation or Automatic Step Clock operations are performed. This timing control is applied to the rate at which instructions are executed if AUTOMATIC STEP OPERATION is selected; if AUTOMATIC STEP CLOCK is selected, the clock pulse rate is regulated accordingly. Timing during Manual Step operations is controlled by the selection of CLOCK, DISTRIBUTOR, or OPERATION, and the manual depression of the STEP button in the "Operating" group. Each depression of the Step button releases, respectively, one clock pulse, one distributor pulse, or the sequence of pulses necessary to the execution of one instruction.

The system is usually operated in the NORMAL mode; however, programmers occasionally use the TEST mode. The NORMAL mode of operation is presented in full, but the TEST mode, being far more complicated, is given in general; a more complete coverage is given in Volume 3, MAINTENANCE.

b. NORMAL MODE OF OPERATION

(1) BASIC SELECTIONS. - A group of basic selections are made on the Supervisory Control Panel (Plate 6-8) prior to actual operation; these are;

(a) NORMAL. - (This selection automatically selects the DRUM as a clock source and HIGH SPEED as a clock rate.)

(b) Either one of the "start" selections, i.e., M.D. START or M.T. START. These are discussed in subparagraph b. (3) below.

(c) One or more, or none, of the manually selective jump selections according to the instructions accompanying the program.

(d) One or more, or none, of the manually selective stop selections according to the instructions accompanying the program.

(e) The contents of the Program Address Counter, PAK, may or may not be altered according to conditions existing in the program.

PX 71671

(f) START. (This selection is made after all basic selections have been made and any other selections or procedures involved have been carried out.)

(2) ABNORMAL CONDITIONS AND FAULTS. - The computer cannot be put into operation in the NORMAL mode if an ABNORMAL CONDITION exists (Test Switch group, Plate 6-10). All disconnect switches must be in the "down" position. Furthermore, the selection of an ABNORMAL CONDITION after operation has been initiated will result in a fault and cause the computer to stop. In general, if a fault condition arises, the computer is prevented from running if not yet started or stopped if already in operation.

(3) COMPUTATION. - Computation can be initiated in two different ways: an M.D. START or an M.T. START. Both starts have the following preliminary steps in common. The controls are located on the Supervisory Control Panel, Plates 6-8 and 6-10.

Step 1. Set the F_1 switch (Test Switch group) to the 00000 position unless the instructions accompanying the program call for a 40001 setting.

Step 2. Check that all ABNORMAL CONDITION switches are in the "down" position and that the BY-PASS TEMPERATURE INTERLOCK key switch is in the "off" position.

Step 3. Prepare those external devices to be used by the program.

Step 4. Prepare those Magnetic Tape units which will be used.

Step 5. Press NORMAL button. NORMAL indicators in the "Operating" and Operation Mode groups, the DRUM indicator in the Clock Source group, and the HIGH SPEED indicator in the Operating Rate group are illuminated.

Step 6. Press those SELECT JUMP buttons called for in the program's instructions. The associated SELECTIVE JUMP indicators are illuminated.

Step 7. Press those SELECT STOP buttons called for in the program's instructions. The associated indicators (immediately above the SELECT STOP buttons) are illuminated.

After these preliminary steps, the selection of M.D. START or M.T. START can be made. The following effects will be noted for each selection:

M.D. START

M.D. START indicator is illuminated.

READY indicator is illuminated

PROGRAM ADDRESS COUNTER is set to octal 40000.

MAIN PULSE DISTRIBUTOR is set to 6.

M.T. START

M.T. START indicator is illuminated.

READY indicator is illuminated.

PROGRAM ADDRESS COUNTER is set to octal 00000.

MAIN PULSE DISTRIBUTOR is set to 0.

"U" ADDRESS COUNTER is set to octal 00001.

"V" ADDRESS COUNTER is set to octal 0000.

MCR (Main Control Register) is set to octal 64.

If the instructions accompanying the program call for a setting of the PROGRAM ADDRESS COUNTER other than that obtained automatically, press the "clear (white) button and enter the binary value by pressing the "set" (black) buttons.

At this point, the system is ready for operation. Pressing the START button in the "Operating" group illuminates the OPERATING indicator and initiates the computation.

c. TEST MODE OF OPERATION. - In the TEST mode the number of optional selections that can be made are numerous. (While operating in the NORMAL mode, the computer is interlocked to prevent the changing of registers.) A general procedure will be given in this section plus a few detailed procedures for operations typical to a programmer's needs. The Magnetic Tape units and the external equipment are prepared as in the NORMAL mode.

After making the desired optional test selections in the Test Switch group, the following step-by-step procedure is followed. (Specific test procedures may introduce minor deviations from the procedure below:)

Step 1. Press the TEST button in the Operation Mode group. The associated TEST indicator is illuminated.

Step 2. Press the DRUM button or the OSC. (oscillator) button in the Clock Source group. The associated indicator is illuminated. (Note that DRUM is not automatically selected as in the NORMAL mode.)

Step 3. Select the desired clock rate in the Operating Rate group. (Note that HIGH SPEED selection is not automatically made as in the NORMAL mode.)

Step 4. Select the desired manual jumps and manual stops.

Step 5. Select either of the following starts: M.D. START, or M.T. START. The MASTER CLEAR signal is produced at this point after which any desired flip-flop may be changed. (The MASTER CLEAR signal does not clear the Magnetic Tape counters and the X-Register.)

Step 6. Press the START button. If HIGH SPEED, AUTOMATIC STEP CLOCK, or AUTOMATIC STEP OPERATION was selected in Step 3, the operation starts immediately. If MANUAL STEP CLOCK, MANUAL STEP DISTRIBUTOR, or MANUAL STEP OPERATION was selected in Step 3, the START selection merely lights the OPERATING indicator; each step of the operation must be initiated by pressing the STEP button in the "Operating" group.

While in the TEST mode, the accidental or intentional pressing of any "set" button (except those associated with the PROGRAM ADDRESS COUNTER) will alter the state of the associated flip-flop. To stop operations during High Speed or Automatic Step operations, press the FORCE STOP button. A Force Stop selection is not necessary during Manual Step operations but may be made for the purpose of establishing a desirable habit. If the FORCE STOP is pressed during manual (and automatic operations) the START button must be pressed to resume operations. If during manual operations, it is desired to change the clock rate, the FORCE STOP button must be pressed to allow the RELEASE button to be pressed and a new selection to be made.

The following procedures in TEST mode are frequently used to manually check portions of a stored program or enter a program into storage. Also included is a procedure for manually transferring a program from Magnetic Drum storage to Magnetic Core storage.

(1) MANUAL WRITING FROM THE Q-REGISTER. - The procedure below can be used to alter an existing program in storage or to insert a new program into storage without using a tape reader. The new words are entered into storage via the Q-Register by the following steps:

Step 1. Select MANUAL STEP OPERATION

Step 2. Select DRUM CLOCK SOURCE

Step 3. Select TEST

Step 4. Select MD START

Step 5. Set MPD to 0

Step 6. Set MCR to 75 (Repeat instruction)

Step 7. Set UAK to 50000 (set j to 5)

Step 8. Set VAK to 00000

Step 9. Press START button

Step 10. Press STEP button

Steps 6 through 10 set up an unterminated Repeat Sequence. Since j is 5, only the v-address will be advanced at the end of each storage reference.

Step 11. Press FORCE STOP button

Step 12. Clear PCR (Clear MCR, UAK, and VAK)

Step 13. Set MCR to 11 (Transmit Positive instruction)

Step 14. Set UAK to 10000 (Q address)

Step 15. Set VAK to first address to be written into

Step 16. Press START button

Step 17. Set up in Q-Register word to be written

Step 18. Press STEP button

Step 19. Clear Q-Register

Steps 13 through 18 manually enter the word set up in Q at the selected v-address. To continue writing in consecutive addresses, repeat steps 17, 18, and 19 for each word to be written. If only a single word is to be written, steps 6 through 12 and step 19 can be omitted since a Repeat operation is not needed.

(2) MANUAL READING TO THE Q-REGISTER. - The procedure below transmits the contents of chosen storage registers to the Q-Register with the words at consecutive addresses being displayed on the Supervisory Control Panel.

Step 1. Select MANUAL OPERATION STEP

Step 2. Select DRUM CLOCK SOURCE

Step 3. Select TEST

Step 4. Select MD START

Step 5. Set MPD to 0

Step 6. Set MCR to 75 (Repeat instruction)

Step 7. Set UAK to 60000 (set j to 6)

Step 8. Set VAK to 00000

Step 9. Press START button

Step 10. Press STEP button

Steps 6 through 10 set up an unterminated Repeat sequence, Since j is 6, only the u-address will be advanced at the end of each storage reference.

PX 71871

Step 11. Press FORCE STOP button

Step 12. Clear PCR (Clear MCR, UAK, and VAK)

Step 13. Set MCR to 11 (Transmit Positive instruction)

Step 14. Set UAK to address of first word to be read

Step 15. Set VAK to 10000 (Q address)

Step 16. Press START button

Step 17. Press STEP button

Steps 13 through 17 manually read the word at the selected u-address to the Q-Register where it is displayed for observation. Each time the STEP button is pressed, a word from a consecutive u-address will be displayed in Q. If only a single word is to be read, steps 6 through 12 can be omitted since a Repeat operation is not needed.

(3) PROGRAM CORRECTION. - If in reading to the Q-Register an incorrect word is noted, the following procedure is used to insert the correct word at the proper address. (This procedure can be used whether or not a Repeat sequence is being used in the manual reading):

Step 1. Press FORCE STOP button

Step 2. Clear Q

Step 3. Clear UAK and VAK

Step 4. Set UAK to 10000 (Q address)

Step 5. Set VAK to address to be written into

Step 6. Set up in Q-Register word to be written

Step 7. Press START button

Step 8. Press Step button

Steps 1 through 8 enter the correct word into storage. To return to the reading process:

Step 9. Press FORCE STOP button

Step 10. Clear UAK and VAK

Step 11. Set UAK to next address to be read from

Step 12. Set VAK to 10000 (Q address)

PX 71871

Step 13. Press START button

Step 14. Press STEP button

Steps 10 through 14 return control to the manual reading procedure.

(4) MANUAL BLOCK TRANSFER. - To effect a manual block transfer from Magnetic Drum Storage to Magnetic Core Storage, the following steps should be performed:

Step 1. Select MANUAL STEP OPERATION

Step 2. Select DRUM CLOCK SOURCE

Step 3. Select TEST

Step 4. Select MD START

Step 5. Set MPD to 0

Step 6. Set MCR to 75 (Repeat instruction)

Step 7. Set UAK to 3n (set j to 3 and n to the number of words to be transferred)

Step 8. Set VAK to a w-address containing a 56jv instruction

Step 9. Press START button

Step 10. Press STEP button

Steps 6 through 10 set up a terminated Repeat sequence. Since j is 3, both the u-address and the v-address will be advanced at the end of each storage reference.

Step 11. Press FORCE STOP button

Step 12. Release MANUAL STEP OPERATION

Step 13. Select HIGH SPEED

Step 14. Clear PCR (Clear MCR, UAK, and VAK)

Step 15. Set MCR to 11 (Transmit Positive instruction)

Step 16. Set UAK to initial MD address

Step 17. Set VAK to initial MC address

Step 18. Press START button

PX 71871

Steps 15 through 18 in conjunction with the Repeat cause the block transfer of n words. Since address w contains a 56 instruction (Manually Selective Stop) and F_1 contains a 45 instruction (Manually Selective Jump), at the termination of the block transfer, control will be transferred to F_1; from F_1 a jump to the 56 instruction is made with the result that computer operations are stopped.

d. RESTORATION OF OPERATION AFTER STOPS. - The computer ceases operation at the occurrence of a programmed Manually Selective Stop, a Force Stop, an emergency stop, or a fault condition. The stops by classes are discussed in subparagraphs (1) through (4) below, together with the steps necessary to resume operation.

(1) PROGRAMMED STOPS.

(a) MANUALLY SELECTIVE STOP. - The Manually Selective Stop instruction, 56jv, stops the computer operation if the programmed j (0, 1, 2, or 3) agrees with the selection made on the Supervisory Control Panel. (No button selection is provided for j = 0; the computer will always stop in this case.) Whether or not a stop occurs at the execution of this instruction, the next instruction will be taken from the v-address. When a stop occurs the OPERATING indicator is extinguished and the appropriate SELECTIVE STOP indicator (red) is illuminated. To resume operation, press the START button. During the stop, any stop or jump selections may be changed.

(b) FINAL STOP. - The Final Stop instruction, 57--, indicates the end of the program and all selections are dropped except NORMAL or TEST and those selections automatically made by the NORMAL selection. Also the SELECT STOP indicators and SELECTIVE JUMP indicators, if illuminated, will remain so. To resume operation, new selections must be made, initiating a new program or re-running the same one.

(2) FORCE STOP. - An unscheduled stop of the computer can be effected by pressing the FORCE STOP button. The OPERATING indicator is extinguished and the FORCE STOP indicator is illuminated. After the condition which prompted the stop has been corrected, operation is resumed by pressing the START button. During operation if it is desired to change any of the selections in the Operating Rate, Clock Source, Operation Mode, Start Selection, and Selective Jumps groups, pressing the FORCE STOP button allows the RELEASE buttons in any of the above groups to be depressed and a new selection to be made.

(3) EMERGENCY STOPS.

(a) MANUAL EMERGENCY STOP. - In cases of extreme emergency, such as a fire, pressing either EMERGENCY STOP button, (one is located on the Main Power Supply Panel and one on the Magnetic Core Power Supply Panel), removes almost all voltages from the equipment. Since the power is removed, a program in process is halted and cannot be immediately resumed.

(b) AUTOMATIC EMERGENCY STOPS. - Automatic emergency stops cause the same condition as pressing an EMERGENCY STOP button.

(4) FAULT CONDITIONS.

(a) A FAULT. An A Fault results in a stop which manifests itself much like a Manually Selective Stop or a Force Stop in that the OPERATING indicator is extinguished and the A FAULT indicator is illuminated rather than a SELECTIVE STOP or FORCE STOP indicator. In addition, the READY indicator is extinguished. The specific fault is indicated by the illumination of one of the following indicators (Fault Indicators group, Plate 6-10):

DIVIDE

SCC (Storage Class Control)

PRINT

TEMPERATURE

WATER

OVERFLOW

ABNORMAL CONDITION - Group X (only if operating in NORMAL mode)

The A Fault does not drop other operating selections; after correcting the fault, operation can be resumed.

Generally, a DIVIDE, SCC, PRINT, or OVERFLOW indication is derived from a program error. First check the appropriate instructions or operands, and, if these are correct, an actual machine malfunction is the cause and must be isolated and corrected. An SCC fault indication results from a reference to some address not permissible under the particular circumstances. A temperature indication is the result of a high air temperature at some point which is indicated by the illumination of one of the amber indicators mounted above the computer cabinet doors or on a piece of external equipment. A WATER indication necessitates correcting a water pressure fault (over- or under-pressure condition) before the program can be resumed. An ABNORMAL CONDITION indication usually occurs at the outset of a NORMAL mode of operation, unless a Test Disconnect switch has been accidentally or intentionally set to the "up" position after operation has been initiated. Movement of the switch to its "down" position will correct the condition. In the correction of some A Faults it is necessary to remake basic selections; in this event, it is mandatory to record the contents of all the principal registers, counters, and the MAIN PULSE DISTRIBUTOR if the operation is to be resumed from the stopping point caused by the fault. Then, after correction of the fault, the contents of the registers, counters, and MPD must be inserted manually before resuming operation.

After correction of the A Fault, press the CLEAR A FAULT button; this illuminates the READY indicator. Finally, press the START button; this illuminates the OPERATING indicator and operation is resumed.

(b) B FAULT. - A B Fault results in a stop which manifests itself much like a Final Stop in that most selections are dropped. Only the NORMAL or TEST selections and those selections automatically made by the NORMAL selection remain in effect. In addition, the B FAULT indicator and one of the indicators listed below (Fault Indicators group, Plate 6-10) are illuminated.

MCT (Main Control Translator)

VOLTAGE

IO (Input-Output)

MISSING FP (Missing Feed Pulse)

MISSING LP (Missing Line Pulse)

An IO fault is indicative of an EXTERNAL fault, IOA1 READ, IOB1 READ, IOA2 READ, or IOB2 READ fault. These fault indicators are shown on Plate 6-9. An EXTERNAL fault originates in external equipment; the IOA and IOB faults listed above occur when there has been no indication that the particular register involved is ready for the information being received by it. If the fault is due to an error in program timing considerations, an IOB1 Read fault indicates that an External Read to transmit input information in IOB to X has not been executed before additional input information is received by IOB from external equipment, and an IOB2 Read fault indicates that the external equipment has not received the information placed in IOB by an External Write or External Function instruction before IOB receives input information from external equipment. The IOA1 Read and IOA2 Read faults indicate similar conditions involving the IOA register.

A MISSING FP, MISSING LP, or a VOLTAGE indication necessitates corrective maintenance. The MISSING LP may be due to a magnetic tape breakage when operating in the NORMAL mode; such a condition is accompanied by an excessive hum caused by the high rotational speed of the tape reels. In the case of magnetic tape breakage, the STANDBY and POWER switches of the MT unit involved should be immediately set to the "down" (off) position to avoid overheating the tape drive motors. If a tape break occurs under conditions that do not stop the equipment, the FORCE STOP button should be pressed and then the STANDBY and POWER switches set to the "down" position. In either case a new tape must be inserted and the problem re-run from the beginning.

Since many operating selections are dropped at the occurrence of a B Fault, the computation must be started anew. If, in the execution of that part of a program which was completed before the fault, the contents of any addresses have been altered by the instructions, it would be well to re-load the computer before renewing computation.

PX 71871

6. CODING FOR THE COMPUTER.

a. SUMMARY OF MACHINE CHARACTERISTICS.

GENERAL	Parallel operation Internal binary number system Two address logic
WORD LENGTH	36 bits (binary digits)
NUMBER NOTATION	"1's complement" binary system
PARALLEL ACCESS REGISTERS (ARITHMETIC SECTION)	A 72-bit Accumulator ($A_{71}, A_{70}, \ldots, A_0$) A_R rightmost 36 bits of A (least significant) A_L leftmost 36 bits of A (most significant) Q 36-bit shifting register ($Q_{35}, Q_{34}, \ldots, Q_0$) X 36-bit exchange register ($X_{35}, X_{34}, \ldots, X_0$)
PARALLEL ACCESS STORAGE (INDIVIDUALLY ADDRESSED)	RAS 1,024 words of Rapid Access Storage MD 16,384 words of Magnetic Drum Storage
ARRANGEMENT OF MAGNETIC DRUM STORAGE, MD	4,096 bits on each track 4,096 words on a group of 36 tracks 4 groups of tracks, giving 16,384 words Variable interlace between the Storage Address Register and angular location counter permits choice of the angular interval between memory locations having consecutive addresses.
MAGNETIC TAPE STORAGE, MT	32 words per block 2,048 blocks per tape unit 4 Magnetic Tape units Total of 262,144 words
ALLOCATION OF ADDRESSES	RAS 00000--01777 (octal) 1,024 words Q 10000--17777 (octal) 1 word A 20000--27777 (octal) 1 double length word MD 40000--77777 (octal) 16,384 words
FIXED ADDRESSES ALLOCATION	F1 00000 (octal) in RAS, Rapid Access Storage or 40001 (octal) in MD Storage F2 00001 (octal) in RAS

PX 71871

COMPOSITION OF AN INSTRUCTION WORD

Instruction word	36 bits ($i_{35}, i_{34}, \ldots, i_0$)
Operation code	6 bits ($i_{35}, i_{34}, \ldots, i_{30}$)
First execution address, u	15 bits ($i_{29}, i_{28}, \ldots, i_{15}$)
Second execution address, v	15 bits ($i_{14}, i_{13}, \ldots, i_0$)

SECTIONS OF ADDRESSES

j one-digit octal number represented by u_{14}, u_{13}, u_{12}.
n four-digit octal number represented by $u_{11}, u_{10}, \ldots, u_0$.
k number of shifts, represented by $v_6, v_5, \ldots, v_0$.

NOTATION FOR CONTENTS OF REGISTERS

Brackets are used to denote "contents of". Thus:
(u)=36-bit word at address u.
(Q)=36-bit word in Q.
(A)=72-bit word in A.
(A_R)=36-bit word in A_R.
(A_L)=36-bit word in A_L.

DOUBLE-LENGTH EXTENTIONS

D(u)=72-bit word whose right-hand 36 bits are (u) and whose left-hand 36 bits are all alike and equal to the left-most bit of (u).
S(u)=72-bit word whose right-hand 36 bits are (u) and whose left-hand 36-bits are all zero.
D(Q), D(X), S(Q), and S(X) are similarly defined.
L(Q)(u)=72-bit word whose left-hand 36 bits are zeros and each of whose right-hand 36 bits is given by the bit-by-bit product of the corresponding bits of (u) and (Q).
L(Q')(v)=72 bit word whose left-hand 36 bits are zeros and each of whose right-hand 36 bits is given by the bit-by-bit product of the corresponding bits of (v) and the complement of (Q).

CONTROL REGISTERS

PAK Program Address Counter
SAR Storage Address Register
PCR Program Control Register
MCR Main Control Register
UAK U-Address Counter
VAK V-Address Counter

PROGRAM SEQUENCE CONTROL

The complete operation for the execution of a computer instruction consists of two parts:
Part One - the execution of the current instruction, CI.
Part Two - the acquisition of the next instruction, NI.

At the start of Part One, the Program Control Registers already contain CI as the result of the second part of the previous operation, and (PAK) is y plus 1, where y is the address from which CI was acquired.

During Part Two, NI is acquired from the address held in PAK at the end of Part one, and (PAK) is then increased by one.

PX 71871

Thus, provided that CI does not call for a change in (PAK), NI will be acquired from address y plus 1. In a normal program sequence, successive instructions are obtained from consecutive addresses.

A departure from the normal sequence is called a "jump", and is achieved by altering (PAK) during Part One of an operation. Instructions that call for a change in (PAK) are called "jump" instructions.

INPUT-OUTPUT REGISTERS

IOA An in-out register of 8 stages.
IOB An in-out register of 36 stages.
TWR A typewriter register of 6 stages.
HPR A high-speed punch register of 7 stages.

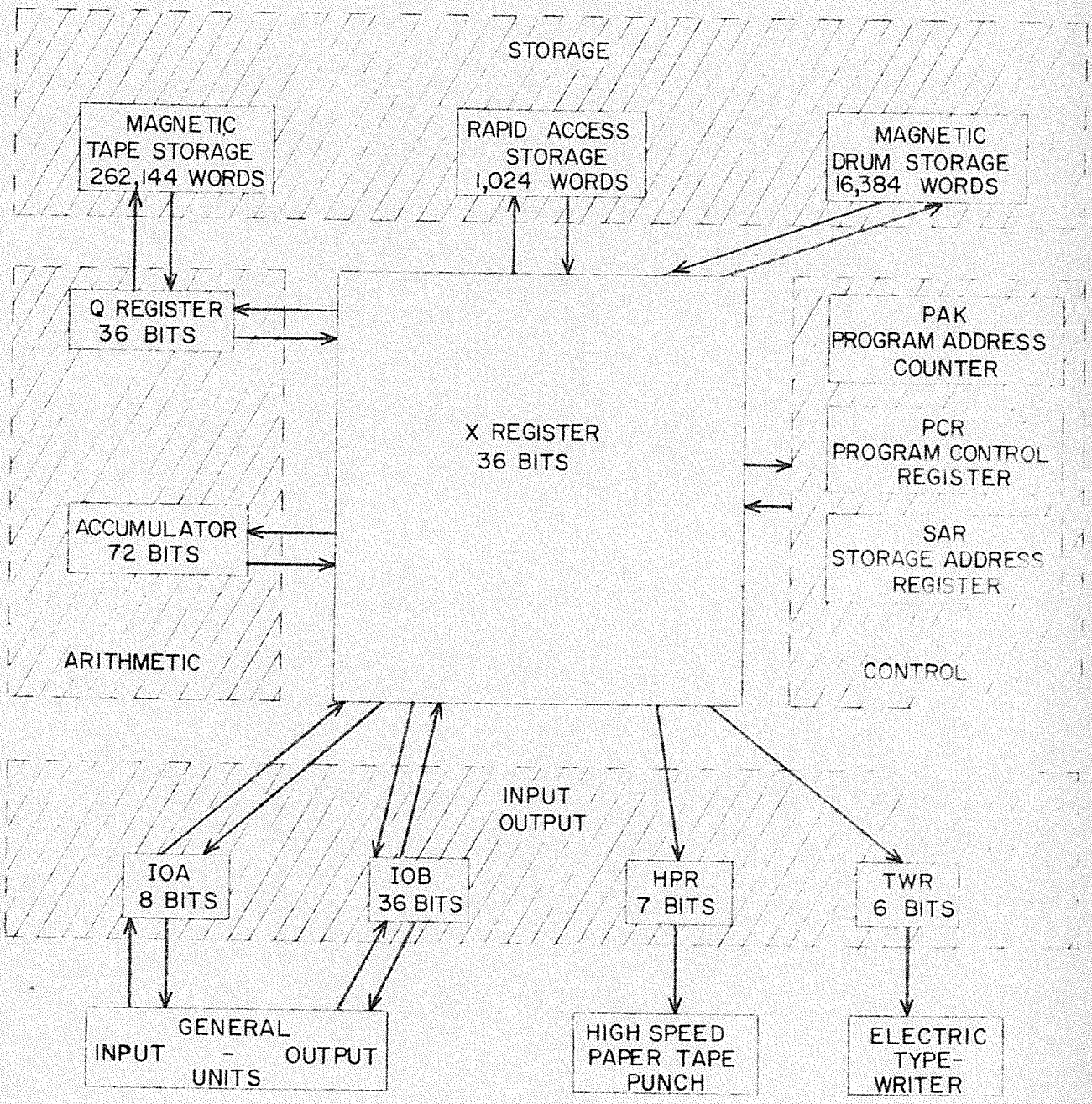
INPUT DEVICES

Photoelectric Paper Tape Reader
Controlled Reproducer (card reader)
Other optional peripheral equipment

OUTPUT DEVICES

Typewriter
High-Speed Paper Tape Punch
Controlled Reproducer (card punch)
Line Printer
Other optional peripheral equipment

PX 71871



PX 71871

Figure 6-5. Programmers Simplified Block Diagram of the 1103 Computer

TABLE 6-7. REPERTOIRE OF INSTRUCTIONS

TPuv	11	TRANSMIT POSITIVE	$(u) \rightarrow v$
TMuv	12	TRANSMIT MAGNITUDE	$\lvert(u)\rvert \rightarrow v$
TNuv	13	TRANSMIT NEGATIVE	$(u)' \rightarrow v$
IPxx	14	INTERPRET	$Y + 1 \rightarrow E_1$, take (F_2) as NI
TUuv	15	TRANSMIT U-ADDRESS.	$(u_{29-15}) \rightarrow v_{29-15}$
TVuv	16	TRANSMIT V-ADDRESS	$(u_{14-0}) \rightarrow v_{14-0}$
EF-v	17	EXTERNAL FUNCTION	Select Ext. Equipment and perform (v)
RAuv	21	REPLACE ADD	$[(u) + (v)] \rightarrow u$
RSuv	23	REPLACE SUBTRACT.	$[(u) - (v)] \rightarrow u$
CCuv	27	CONTROLLED COMPLEMENT . . .	$[(u) \oplus (v)] \rightarrow u$
SPuk	31	SPLIT POSITIVE ENTRY . . .	$S(u) \rightarrow A$, Shift (A) by k
SAuk	32	SPLIT ADD	(A) + S(u), Shift (A) by k
SNuk	33	SPLIT NEGATIVE ENTRY . . .	$[S(u)]' \rightarrow A$, Shift (A) by k
SSuk	34	SPLIT SUBTRACT.	(A) - S(u), Shift (A) by k
ATuv	35	ADD AND TRANSMIT	$[(A) + D(u)] \rightarrow v$
STuv	36	SUBTRACT AND TRANSMIT . . .	$[(A) - D(v)] \rightarrow v$
RJuv	37	RETURN JUMP	$y + 1 \rightarrow u$, take (v) as NI
IJuv	41	INDEX JUMP	$[D(u)-1] \rightarrow A$; $(A)_f +$, $(A)_f \rightarrow u$, take (v)
TJuv	42	THRESHOLD JUMP.	(u) > (A), take (v) as NI
EJuv	43	EQUALITY JUMP	(u)=(A), take (v) as NI
QJuv	44	Q-JUMP	(Q)+, take (v); (Q)-, take (u); (Q) left 1
MJjv	45	MANUALLY SELECTIVE JUMP . .	j=0, or j=1,2,3, & MJS=, take (v)
SJuv	46	SIGN JUMP	(A)-, take (u); (A)+, take (v)
ZJuv	47	ZERO JUMP	$(A) \neq 0$, take (u); (A)=0, take (v)

-1

PX 71871

TABLE 6-7. REPERTOIRE OF INSTRUCTIONS (CONCL'D)

QTuv	51	Q-CONTROLLED TRANSMIT . . .	$L(Q)(u) \longrightarrow v$
QAuv	52	Q-CONTROLLED ADD	$\left[(A) + L(Q)(u)\right] \longrightarrow v$
QSuv	53	Q-CONTROLLED SUBSTITUTE . .	$\left[(L(Q)(u) + L(Q)'(v)\right] \longrightarrow v$
LAuk	54	LEFT SHIFT IN A	$D(u) \longrightarrow A$, Shift (A) by k, $(A)_f \longrightarrow u$
LQuk	55	LEFT SHIFT IN Q	$(u) \longrightarrow Q$, Shift (Q) by k, $(Q)_f \longrightarrow u$
MSjv	56	MANUALLY SELECTIVE STOP . .	j=0, stop; j=1,2,3, & MSS=, stop
FS--	57	FINAL STOP	Stop and indicate
PR-v	61	PRINT	Typewriter performs code in v_{5-0}
PUjv	63	PUNCH	Punch (v_{5-0}); j=1, 7th level also
RMjnv	64	READ MAGNETIC TAPE	Read n blks jMT to RAS, starting at v
WMjnv	65	WRITE MAGNETIC TAPE	Write n blks jMT from RAS, starting at v
AMjn-	66	ADVANCE MAGNETIC TAPE . . .	Advance jMT n blocks
BMjn-	67	BACK MAGNETIC TAPE	Back jMT n blocks
MPuv	71	MULTIPLY.	$(u)(v) = (A)$
MAuv	72	MULTIPLY ADD.	$(A)_i + (u)(v) = (A)_f$
DVuv	73	DIVIDE	$(A)_i = (u) \cdot (Q) + (A)_f$, $(Q) \longrightarrow v$; $(A)_f = +R$
SFuv	74	SCALE FACTOR	Shift D(u) in A until $A_{34} \neq A_{35}$, (SK) $\longrightarrow$ v
RPjnw	75	REPEAT	Execute Nl "n" times, jump to I_1
ERjv	76	EXTERNAL READ	j=0, (IOA) $\longrightarrow$ v; j=1, (IOB) $\longrightarrow$ v
EWjv	77	EXTERNAL WRITE	j=0, (v) $\longrightarrow$ IOA; j=1, (v) $\longrightarrow$ IOB

PX 71871

b. WRITING A PROGRAM.

(1) INTRODUCTION. - The word "coding" is usually used to indicate the preparation of the explicit list of instructions that a computer must execute in order to solve a particular problem. The person who does the coding may be referred to as a "coder". The resultant list of instructions is referred to as either a "routine" or, if the coded product is part of a larger one, a "subroutine". Sometimes the term "programming" is used to refer to the process described above as coding. Ordinarily, however, "programming" refers to a more inclusive process, which includes not only the coding but the final stages of formulation of the problem for the computer; i.e., the numerical analysis, the selection of computational procedures, the specification of input-output formats; etc.

(2) INSTRUCTION NOTATION. - In the following text the mnemonic (alphabetic) notation is used to indicate the operation code of an instruction. For example, "TP" is used instead of "11" to indicate the Transmit Positive instruction; "MP" is used instead of "71", etc. The address portions of the instruction are indicated by octal numbers. For example, the composite symbol

TP 01012 01013

denotes the Transmit Positive instruction calling for the transmission of the contents of address 01012 to address 01013. If this were written in octal, it would appear as

11 01012 01013

In binary, it would appear as

001 001 000 001 000 001 010 000 001 000 001 011.

As illustrated above, it would be impractical to code using the binary system. For this reason, and because the conversion of numbers from base 2 to base 8, and vice versa, is immediate, the octal number system is used in coding.

When preparing a routine, not only the instructions which are to be executed must be specified, but also the placement of these instructions in the computer must be indicated. The storage location in the computer of an instruction is referred to as the storage address of the instruction. It is customary to indicate this address to the left of the notation for the instruction, as follows:

01010 TP 01012 01013

It is common practice when displaying routines for exposition to give some explanatory comment to the right of the instruction. In the following routine the left column contains the location in storage; the central column contains the contents of that location, for instance an instruction or datum; and the right column indicates the author's reason for including this word in the routine.

Routine for replacing the data at 00101, 00102, - - -, 00110 by the first backward difference of this data:

00010	RS	00110	00107	form backward difference
00011	RS	00010	00015	decrease (00010)
00012	EJ	00014	00000	has (00010) reached (00014)?
00013	MJ	00000	00010	
00014	RS	00100	00077	terminal dummy instruction
00015	00	00001	00001	constant (address decrement)

(3) LOOPS. - The above is an example of a routine which has a "loop". The loop in this case consists of the three instructions at 00010, 00011 and 00012. The first of these forms one of the backward differences and stores it where the minuend of the subtraction was. The second instruction, the Replace Subtract at address 00011, then modifies the instruction which forms the backward difference by decreasing both of its addresses by 1. Thus, the next time the loop is traversed and a backward difference is formed, it will be the difference of the contents of the decreased addresses. The next instruction, the Equality Jump, tests to see whether or not the loop has been traversed a sufficient number of times. If equality exists between the most recently modified content of address 00010 and the dummy instruction at address 00014, the task has been completed and an exit occurs to address 00000. If equality does not exist, the next instruction at address 00013 is executed. This instruction provides a jump back to the beginning of the loop at 00010, and the whole process is repeated once more.

The above example illustrates one very important facet of coding: a number of the instructions in any routine are always involved only indirectly with the desired results. In this example only one instruction, the first, produces directly the first backward difference. The others are concerned with keeping the process going, advancing addresses, and testing to see if the process has been completed. Such instructions are frequently referred to as "housekeeping" instructions, and the functions of address modification, terminal testing, and jump provisions are referred to as "housekeeping" functions.

It is common practice to indicate the fact that certain instructions are altered during the course of the computation by putting brackets around the instruction or that part of it which is altered; thus, the first line of the above example could have properly been written as

00010 [RS 00110 00107]

or as

00010 RS [00110] [00107].

The latter is less preferable than the former because it suggests that the addresses are independently modified, which is not true in this case. Bracketing the so-called modified instructions or modified addresses serves to bring attention to the fact that at some later time during the computation, the addresses may not be as they appear on the page. In particular, in the example above, at the end of the subroutine the addresses at 00010 will have been modified to agree with those appearing at storage address 00014.

It is generally good practice to have instructions prefacing each loop which set the addresses being modified within the loop to their desired initial values. In this way the routine can be used any number of times while it is in the memory without reloading it from a permanent storage device. Note that the above example does not meet these specifications. Below is the same example coded with such "self-restoring" properties. In this routine the loop consists of the instructions at 00010, 00011, and 00012. Note that a different means is provided to terminate the loop. Note also the use of boxing to indicate a loop and the use of a straight line, following an unconditional jump, to indicate a separation.

Routine for replacing the data at 00101, 00102, . . ., 00110 by the first backward difference of this data:

00006	TP	00017	00016	Prestore counter, (00016)
00007	TP	00014	00010	Prestore (00010)
00010	OO	00000	00000	Form backward difference
00011	RS	00010	00015	Decrease (00010)
00012	IJ	00016	00010	All differences formed?
00013	MJ	00000	00000	Exit
00014	RS	00110	00107	Initial contents of 00010
00015	00	00001	00001	Constant (address decrement)
00016	00	00000	00000	Counter, initially n-1.
00017	00	00000	00006	Constant, n-1, initial value of counter.

In this case the termination test is performed by an Index Jump instruction which is executed after each traversal of the loop. Execution of the Index Jump at address 00012 causes a "1" to be subtracted from the contents of 00016 and a test to be made to see if the result is negative. If the result is not negative, a jump to address 00010 is made for another traversal of the loop. Memory location 00016 is called a "counter" since it contains at each step a count-down tally of the number of traversals of the loop. If the Index Jump instruction is at the end of a loop (wiich is to be traversed n times), this counter should have an initial value of n-1. The quantity is a constant

stored at location 00017. Entry of the routine at address 00006 causes this quantity n-1 to be transferred to the counter at location 00016. The termination test could also have been coded at the beginning of the loop. In this case, however, the initial value of the counter should be "n" to effect n traversals of the loop.

When the termination test indicates that the loop has been traversed n times, the execution of the Index Jump instruction at address 00012 does not cause a jump but allows the next instruction to be taken from 00013. Here is stored the exit instruction, a Manual Jump.

Any of the conditional jumps, Index Jump, Threshold Jump, Equality Jump, Q-Jump, Sign Jump, or Zero Jump, can be used as the decision instruction to determine the number of times a loop has been traversed.

In summary, a loop consists of four elements:

(a) A series of self-restoring operations which set up the loop for the first traversal. These initial "housekeeping" instructions place in a "counter" a number which determines the number of times, n, that the computer will traverse the loop. The instructions also set in the routine the initial addresses of any data referenced by the instructions within the loop.

(b) The computational instructions which lead to the solution of the particular problem.

(c) The "housekeeping" instructions which advance/decrease the addresses which reference the data so that successive loop computations are performed on successive data.

(d) The "housekeeping" instructions which advance/decrease the counter, and the decision instruction(s) which test the counter to determine if the loop has been traversed n times.

(4) SUBROUTINES.

(a) INTRODUCTION. - A portion of a routine which is complete in itself and can be isolated from the context of the larger routine is known as a "subroutine". A subroutine is a self-contained list of instructions for executing some particular operation. If it contains the calculations necessary to compute a function such as $\sqrt{x}$, sine x, tan x, etc., it should be coded in such a way that it may be used in a number of routines wherever such a function is desired.

(b) SUBROUTINES WITH PARAMETERS. - In the case of subroutines which are to be used in many different programs, absolute addresses may not be assigned to certain quantities. These addresses are regarded as parameters of the subroutine and chosen in accordance with the main program. In such a case it is conventional to write in lieu of an absolute address some designation for the quantity itself enclosed in braces. Thus, "$\{x\}$" means "the address at which the quantity x is stored". For example, the instruction for

PX 71871

transmitting x to address 01033 would be written

TP $\{x\}$ 01033.

The following subroutine for adding two vectors together exemplifies the use of parameters. The two vectors are denoted by x and y with coordinates x_i and y_i, respectively. The sum vector is denoted by z with coordinate z_i. Thus, it is necessary to form the sums

$$z_i = x_i + y_i \qquad i = 1, 2, 3, - - -, n.$$

00100	TP	00114	00117	set counter to n-1
00101	TU	00112	00104	set in $\{x_1\}$
00102	TU	00113	00105	set in $\{y_1\}$
00103	TV	00112	00105	set in $\{z_1\}$
00104	TP	$[\{x_i\}]$	20000	form
00105	AT	$[\{y_i\}]$	$[\{z_i\}]$	$z_i = x_i + y_i$
00106	RA	00104	00116	Advance i by 1
00107	RA	00105	00115	
00110	IJ	00117	00104	termination test
00111	MJ	00000	[00000]	exit
00112	00	$\{x_1\}$	$\{z_1\}$	constant
00113	00	$\{y_1\}$	00000	constant
00114	00	00000	n-1	constant
00115	00	00001	00001	constant
00116	00	00001	00000	constant
00117	00	[00000	00000]	counter

This subroutine assumes that tabular values of x_i and y_i are stored in consecutive order somewhere in the memory and that a table of values of z_i is to be stored in consecutive order in the memory. The constants located at 00112 and 00113 are, in effect, parameters of this subroutine because these constants give the addresses of x_1 and y_1, and the address where z_1 is to be stored.

Another parameter of this subroutine is the terminal value of i (i=1, 2, 3, - - -, n). The subroutine has a loop from 00104 to 00110 which needs to be traversed n times.

(c) SUBROUTINE ENTRANCE AND EXIT. - To make use of a subroutine, the main routine must provide a jump to the correct entry point of the subroutine, and the subroutine must provide an exit back to the main routine. In the subroutine for adding two vectors the entry address is 00100. The v-address portion of the Manual Jump instruction at 00111 must be set to a jump-out address in the main routine.

The simplest method of entering a subroutine is to use the Return Jump instruction. To enter the above subroutine the instruction below would be written at address y in the main routine:

RJ 00111 00100.

Execution of this instruction causes the quantity y + 1 to be placed in the v-address portion of 00111 and the contents of 00100 to be taken as the next instruction. Thus, a return to the main routine at address y + 1 is provided upon completion of the subroutine. In this way the subroutine may be entered from many different points in the main routine.

If a subroutine has parameters, these must be set up before it is executed. The parameters may be inserted by instructions in the main routine. Thus, in the subroutine for adding two vectors, the contents of addresses 00112 and 00113 could have been entered by instructions in the main routine. If the parameters occur in many different instructions throughout a subroutine, the subroutine itself will contain the instructions for inserting the parameters in all necessary locations. Again in the subroutine for adding two vectors, instructions 00100 through 00103 distribute the parameters throughout the routine. Since each parameter is used only once, it would be as easy for the main routine to set the parameters directly in addresses 00104 and 00105 as to place them in addresses 00112 and 00113. However, when a parameter occurs more than once in a subroutine, it is better that the main program insert the parameter in a single location and the subroutine distribute it from there. This avoids the possibility of a programmer omitting some of the parameter settings.

A convenient means of repeatedly executing a subroutine with a different parameter each time is illustrated next. The subroutine is referenced by a Return Jump instruction followed immediately by the list of parameters. Following the last parameter is the next instruction to be executed after the repeated use of the subroutine.

y	37	00010	00011
y+1	00	a_1	n_1
y+2	00	a_2	n_2
.	.	.	.
.	.	.	.
.	.	.	.
y+k	00	a_k	n_k
y+k+1	NI		

Here the parameters a_i and n_i, $i = 1, \ldots, k$, are for the following subroutine for punching $n + 1$ consecutive words at addresses a_j, $j=i, i+1, \ldots, i+n_i$.

PX 71871

Subroutine for Punching n + 1 Consecutive Words

00010	MJ	00000	[00000]	Exit
00011	SP	00010	00017	Entrance
00012	TU	20000	00013	y+1 → u-address of 00013, (y+1) = P_1
00013	TP	[P_i]	10000	P_i → Q
00014	QT	00036	20000	If (A)=0, this is a parameter; go to 00021
00015	ZJ	00016	00021	If (A)≠0, this is NI; go to 00016
00016	SP	00013	00071	y+k+1 → v-address of 00010
00017	TV	20000	00010	
00020	MJ	00000	00010	to exit instruction
00021	TU	10000	00023	a_i → u-address of 00023
00022	TV	10000	00037	n_i → 00037
00023	TP	[a_j]	10000	(a_j) → Q; j = i, i+1, - - -, i+n_i
00024	TP	00040	00041	set counter to punch 5 frames
00025	LQ	10000	00006	shift to punch next two octal digits
00026	PU	00000	10000	punch
00027	IJ	00041	00025	have 5 frames been punched?
00030	LQ	10000	00006	shift to punch last two octal digits
00031	PU	10000	10000	punch with 7th level to show end of word
00032	RA	00023	00042	increase (00023) to pick up next word
00033	IJ	00037	00023	have n_i+1 words been punched?
00034	RA	00013	00042	increase (00013) to pick up next parameter
00035	MJ	00000	00013	return to pick up next parameter
00036	77	00000	00000	mask to detect parameter
00037	[00	00000	n_i]	counter for number of words to be punched
00040	00	00000	00004	initial value of counter, 00041
00041	[00	00000	00000]	counter for number of frames per word
00042	00	00001	00000	constant to augment u-address

PX 71871

The Return Jump instruction at address y places y + 1 in the v-address portion of 00010. Therefore the subroutine can recover from 00010 the address of the first parameter, P_1. Instructions 00014 and 00015 test the content of y + 1 to see if it is a parameter or if it is the next instruction. This test is based on the fact that there is no operation code "00". When "NI" is detected, the exit in 00010 is to y+k+1.

Note that this device permits listing any number k of parameters. The mechanics may easily be varied to accomodate sets of parameters of any form, e.g., sets of parameters which occupy more than one word. Different methods of detecting the end of the parameter list are possible.

The 1103 instruction repertoire includes the Interpret instruction for referencing subroutines with parameters. An Interpret instruction at address y places y + 1 in the v-address of F_1 and takes (F_2) as the next instruction. In other words, the operation code of the Interpret instruction in itself indicates the same operation as the instruction RJ 00000 00001. The advantage of the Interpret instruction lies in the availability of 10 octal digits in which information may be stored. Address F_2 usually contains a jump to a subroutine which uses the information in the Interpret instruction as parameters for its operation. The subroutine can set up its own exit from the v-address of F_1.

Frequently, complex systems of pseudo-coding are built around the Interpret instruction. These "interpretive systems" simplify the job of programming for a computer because of the use of pseudo-instructions whose functions define operations which are actually carried out by a series of ordinary machine instructions. Such series of stored instructions are in the form of stored subroutines, the operations of which are initiated by the interpretation of the pseudo-instructions.

Chief among the Interpretive systems are those providing for arithmetic operations on floating binary point numbers. (See subparagraph c.) In a typical system, the Interpret instruction is composed as follows, each character denoting an octal digit:

IP RA u_3 u_2 u_1 u_0 v_3 v_2 v_1 v_0

Here the pseudo-code "RA" refers to the Replace Add subroutine, corresponding to the machine operation of Replace Add, for floating binary point numbers. Each of the operand addresses u and v has four octal digits rather than five. Since only four octal digits are needed to span the range of addresses in RAS, this is not a severe restriction. The interpretation routine deciphers the third and fourth octal digits, in this case RA, to determine which of the subroutines in the system is to be executed.

(d) LIBRARY OF SUBROUTINES. - In any computational laboratory having an 1103, there is a library of subroutines which can be used to calculate the frequently encountered mathematical or logical operations required by the particular laboratory. These are then available for any programmer to use in coding his own programs. In addition to saving coding time, library routines have the advantage of being "debugged" and thoroughly tested. And presumably, more effort has gone into their design than could be afforded by individual

PX 71871

programmers. On the other hand, library routines may be more general then necessary for a particular program (and hence wasteful of time and storage).

Subroutines may be grouped into three basic categories. The following listing is intended to give the beginning coder an idea of the type of subroutines that would be contained in a typical library of subroutines. This list is by no means intended to be complete.

1 COMPUTATIONAL SUBROUTINES.

Basic arithmetic:

Floating point add, multiply, divide, etc.
Complex number operations
Multiple precision routines

Function evaluation:

Trigonometric, sin x, cos x, tan x, etc.
Exponential, $f(x) = x^a$, a^x, etc.
Square root
Hyperbolic functions

Numerical Analysis:

Solution of n simultaneous equations
Matrix operations
Numerical integration
Numerical differentiation

Logical:

Sorting
Collating
Boolean algebra

2 SERVICE SUBROUTINES AND/OR ROUTINES.

Compilers
Assembly
Programming aids (debugging subroutines)
Machine testing routines

3 INPUT-OUTPUT SUBROUTINES.

Punch card (read and punch)
Paper type (read and type)
Editing (typewriter, paper tape or card output)

The need to become thoroughly familiar with the available library of subroutines cannot be over-emphasized. An understanding of what each subroutine will accomplish and the requirements for its use is important.

(e) ASSEMBLY OF SUBROUTINES. - If each of the library subroutines were coded for a specific location in the computer memory, a programmer wishing to use subroutines would be faced with two unpleasant alternatives: (1) code his main routine around the fixed locations of the subroutines, or (2) manually recode the subroutines to operate from locations which he chooses.

Instead, the practice is to adopt some standard form for subroutines and to use an "assembly program" to convert the standard form of the subroutine to coding suitable for execution at a specified location.

The simplest of these assembly routines assumes that the standard form of the subroutine is already at the desired location. It remains for the assembly routine to modify addresses (within the instructions of the subroutine) relative to that location.

A more sophisticated assembly routine provides not only for address modification but for transfer of the subroutine to a designated location from a library file on magnetic tape or magnetic drum.

Furthermore, there are in use "compiling" routines which relieve the programmer of assigning storage locations to subroutines. The programmer need only refer to a subroutine by means of an identifying index. The compiler routine then arranges the subroutines in storage and modifies all subroutine references to provide for a jump to the assigned location.

One standard form for subroutines in use at several 1103 installations is as follows:

1 The initial address of each subroutine is address 01000 in rapid access storage. The service routines (assembly, compiling, etc) treat address 01000 as a "relative" address.

2 The first instruction of the subroutine provides for the "alarm" exit". Most subroutines are coded to detect the unexpected or undesirable, such as entry with an out-of-range argument or incorrect parameters. When such is detected, a jump to 01000 occurs. At 01000 is the instruction

37 76000 76002

which provides entrance to an "Alarm Print Routine". This routine prints (A), (Q), and the address of the instruction at which the alarm condition occured.

3 The second instruction provides for the normal exit from the subroutine. The address of this instruction, 01001, is the u-address of the RJuv instruction used to enter the subroutine. This instruction is a Manually Selective jump, coded as follows:

01001 MJ 0(=j) [30000].

PX 71871

Actually the v-address of this instruction of 30000 is an unallowable address which will cause an SCC computer fault if the jump to 30000 occurs. Normally the RJ instruction used to enter the subroutine would replace the v-address of the MJ instruction with y+1. However, if the RJ instruction had been incorrectly coded, or the subroutine had been entered without an RJ instruction or without first changing the v-address of (01001), the computer would be stopped after executing the subroutine when (01001) is taken as the next instruction. Thus would an incorrect entrance into the subroutine be indicated.

4 The third instruction of the subroutine (01002), is the entry line into the subroutine. This address, 01002, is the v-address of the RJ instruction used to enter the subroutine. If more than one entry into a subroutine is necessary, instructions referencing the various desired entry points in the subroutine are usually stored at consecutive memory positions beginning with the third line of the subroutine.

5 A jump instruction must be placed at the end of the subroutine computations to cause a jump to 01001, the normal exit line of the subroutine.

6 All instructions and "modifiable" operands should be placed in consecutive memory positions. "Modifiable" operands are any words referencing storage locations of the subroutine which would be changed upon translation of the subroutine. The "unmodifiable" operands should be stored directly following the above, at consecutive locations.

7 The subroutine must contain all the housekeeping instructions that make it self-restoring.

The Square Root Routine given below is a routine coded in the standard form just described. This routine is to be entered with the argument in the Accumulator; the square root of that argument is in the Accumulator upon exiting from the routine. Note the jump at 01002 to the alarm exit should the argument be negative, and the jump at 01007 to the alarm exit should the argument be out-of-range.

Routine for Evaluating the Square Root of N

Entrance conditions: $(A_L) = 0$, $(A_R) = N$

Exit conditions: $(A_R) = \sqrt{N} \cdot 2^{17}$

The method used here is the Newton-Raphson iteration

$$X_{n+1} = X_n + 1/2 \left[\frac{N}{X_n} - X_n \right]$$

$$= X_n + \Delta X_n$$

The first approximation is $X_0 = 2^{35-1}$ and convergence is assumed when $\Delta X \geq 0$.

PX 71871

01000	37	76000	76002	Alarm Exit
01001	45	00000	[30000]	Normal Exit
01002	46	01000	01003	Entrance; N Negative?→ Alarm
01003	11	01016	00005	$2^{35}-1 \rightarrow (t_5)$
01004	11	20000	00003	$N \rightarrow (t_3)$
01005	43	01016	01001	$N = 2^{35}-1 \rightarrow$ Exit
01006	36	00003	10000	$(N) - (A_R) \rightarrow (A)$
01007	47	01000	01010	$(A) \neq 0 \rightarrow$ Alarm
01010	31	00003	00041	$1/2\ N \cdot 2^{34} \rightarrow (A)$
01011	73	00005	00004	$1/2\ N/x_i \cdot 2^{34} \rightarrow (Q), (t_4)$
01012	54	00005	00107	$1/2\ x_i \cdot 2^{34} \rightarrow (A), (t_5)$
01013	23	10000	00005	$(1/2\ N/x_i - 1/2\ x_i) \cdot 2^{34} \rightarrow (Q) = \Delta x$
01014	21	00005	00004	$1/2\ x_i \cdot 2^{34} + 1/2\ N/x_i \cdot 2^{34} = (x_{i+1})\ 2^{34} \rightarrow (t_5)$
01015	44	01010	01001	Δx Negative, Repeat Loop
01016	37	77777	77777	$2^{35}-1$

Note the use of addresses 00003, 00004, 00005 (t_3, t_4, t_5) as temporary storage for partial results obtained during computation.

Most routines require working storages such as these during their operation; upon completion of the routine the contents of these locations are no longer of use. A single location may be used several times in the same routine for temporary storage of different quantities. For instance in this routine 00005 contains successively x_i, $1/2\ x_i \cdot 2^{34}$ and $x_{i+1} \cdot 2^{34}$. Since it is possible for many different subroutines to use the same locations for temporary storage, it is conventional to reserve an area of the memory for temporary storage. Several installations have reserved RAS locations 00000-00037 for this use.

In this routine the words in which addresses are to be modified when the subroutine is assembled are in 01000 to 01015. The word in 01016 is a constant and is not to be modified.

An example follows of an assembly routine which is suitable for modifying subroutines in the previously given standard form. Recall that each subroutine to be assembled must be coded in absolute form as if its initial instruction were at address 01000. In actual fact the subroutine is placed in

RAS at any location selected by the coder. It is the function of the assembly routine to modify all addresses that are dependent on the location of the subroutine. The assembly routine must be informed of the actual RAS address s of the initial instruction of each subroutine and of the number n of successive instructions to be modified in each case. It will then scan (s+j-1) instructions, j=1, 2, ... n, altering the u and v addresses of the instructions by adding s-01000 to them when necessary.

A particular address will be modified if, and only if, the 10th bit from the right in its 15 bit array is a one; i.e., if it is of the form XXX XX1 XXX XXX. For RAS addresses this means that all addresses of the form 01xxx (x an arbitrary octal digit) and only such addresses will be modified. Thus, addresses of temporary storage such as those described previously (addresses 00000 through 00037) will not be altered. Since n successive instructions are modified starting with the initial one, constants should be stored at addresses 01000 + n and following.

The routine below will assemble k subroutines in succession. It is assumed that the parameters for each, n_i and s_i, are of the form

$$00 \quad n_4\ n_3\ n_2\ n_1\ n_0 \quad s_4\ s_3\ s_2\ s_1\ s_0$$

and are stored consecutively at addresses 00121 + i, i = 1, 2, ... k. Address 00121+k+1 must contain 0 to indicate the end of the parameter list.

Address Modification Routine

00100	11	[00122]	20000	(A) = (00121 + i)
00101	47	00102	00000	Exit -- All modified
00102	73	00073	00005	(00005) = n_i
00103	16	20000	00112	$(00112)_V = s_i$
00104	36	00121	00006	(00006) = s_i -01000
00105	41	00005	00110	Exit for next subroutine
00106	21	00100	00073	Advance i by one
00107	45	00000	00100	Jump to 00100
00110	16	00112	00116	$(00116)_V = (00112)_V$
00111	11	00040	00007	(00007) = zero
00112	21	00007	[00000]	$(00007) = (s_j)_i$
00113	55	20000	00033	Shift right nine binary digits
00114	51	00075	10000	(Q) = 00 00001 00001

PX 71871

00115	71	10000	00006	(A) = increment to $(s_j)_i$
00116	35	00007	[00000]	$(s_j)_i$ = modified initial $(s_j)_i$
00117	21	00112	00074	s_j advanced by one
00120	45	00000	00105	Jump to 00105
00121	00	00000	01000	Constant 1000 octal

Note the references to addresses 00040, 00073, 00074, 00075. The contents of these addresses are commonly used constants:

00040	00	00000	00000
00073	00	00001	00000
00074	00	00000	00001
00075	00	00001	00001

An 1103 installation will sometimes reserve an area of RAS for a "constant pool", i.e., a collection of frequently used constants. All subroutines can then refer to the constant pool rather than having each contain its own constants. This results in a considerable saving of storage space when many subroutines are used. In the particular case of this assembly routine, it is assumed that there is a constant pool at addresses 00040 through 00077.

In the case of routines, such as the assembly routine, which are to be used by many different persons, it is important that the routine be accompanied by complete and easily understood operating instructions. The operating instructions for the previous assembly routine are given below.

Operating Instructions for the Assembly Modification Routine

1 Only <u>suitably coded</u> subroutines may be assembled. A subroutine is suitably coded if it satisfies the following conditions.

a It is written in absolute form as if the initial instruction were located at address 01000.

b All addresses independent of the location of the routine are of the form XXX XX0 XXX XXX XXX.

c All addresses to be modified are of the form XXX XX1 XXX XXX XXX.

d All subroutine constants follow immediately the last instruction.

2 To use this routine:

a Load all subroutines to be modified in their proper locations in RAS.

PX 71871

b Load this assembly program in 00100-00121.

c Load the constant pool in 00040-00077.

d Load the parameters

00 n_i s_i $i = 1, 2, \ldots k$

for all k subroutines into addresses 00121+i, i = 1, ..., k.

e Load zero into address 00121 + k + 1.

f Provide for the proper exit in the u-address of 00101 and enter at address 00102.

The assembly routine which has been described is an example of a service routine designed to relieve a coder of part of the work in preparing a program for execution by the computer. The text following describes a coding technique, simplifying the job of coding, which is possible because of another type of service routine.

(5) RELATIVE ADDRESSING. - In coding routines it is often preferable not to assign absolute memory locations to quantities refered to in the program. For instance, it may be convenient to postpone the assignment of absolute addresses to data, constants, temporary storage and the like, until the coding of the entire problem has been completed. Similarly, it may not be desirable to assign locations to subroutines until coding is completed and it is known what the storage requirements are for each of the sections of the program.

It is possible to postpone the assignment of absolute addresses by coding with "relative" addresses. For instance the instruction

01010 11 01012 01013

which is coded with absolute addresses 01010, 01012, and 01013 might be written with relative addresses as

C10 11 C12 C13.

In this case, the addresses are relative to the address C. The alphabetic character C could denote any address allowable in the 1103; for example, it could be assigned the address 01000. The numerals following an alphabetic character in a relative addressing scheme usually are interpreted as being additive; i.e., if C denotes 01000, then C12 denotes 01012, etc. Almost any of the alphabetic characters are usually allowable in most relative addressing schemes. A few exceptions are the reservations of the letter A, denoting the Accumulator with the address 20000, and the letter Q denoting the Q-register with the address 10000. Other exceptions to the use of alphabetic characters need not concern the beginning 1103 coder. The allowable characters depend upon the conventions in use at a particular 1103 establishment.

Each new alphabetic character used in a program may indicate a new "region" in the storage of the computer. In fact, a greater number of regions may be accomodated by using combinations of alphabetic or alphabetic and numeric characters. By using relative addresses it becomes comparatively simple to assign segments of a problem to various regions with the routines assigned to each region performing a separate calculation or function.

As an example of a routine coded with relative addresses, consider the following:

Compute the function f(y), where $f(y) = \frac{5y-2}{y+7}$.

Assume that $y+7 \neq 0$ and $|y+7| < 2^{35}$.

Region	Instruction			Function
(Program of instructions)	Operation code	u-address	v-address	
C1	TP	f1	A	Place y in A
C2	AT	E1	f2	Place y + 7 in f2
C3	TN	E3	A	Place -2 in A
C4	MA	E2	f1	Form 5y-2 in A
C5	DV	f2	f3	Place the result of 5y-2/y+7 in f3
C6	FS	0	0	Stop
(Constant Storage)				
E1	0	0	7	
E2	0	0	5	Constants
E3	0	0	2	
(Temporary Storage)				
f1	-	-	-	Variable y
f2	0	0	0	Temporary storage holding y+7
f3	0	0	0	Temporary storage holding f(y)

PX 71871

After a program is coded with relative addresses, it must be converted to absolute addresses before execution by the computer. The 1103 may be used to perform this conversion. "Translation" routines have been written which not only convert relative addresses to absolute addresses but which also convert decimal information to 1103 binary representations. Frequently these translation routines include an assembly subroutine.

(6) MECHANICS OF CODING.

(a) FLOW DIAGRAMS. - After it has been established that a particular problem can be solved by the computer, it is necessary to formulate the problem in terms of the language of the 1103 computer; i.e., a program of instructions and the necessary data needed for the solution of the problem must be derived.

A "flow diagram" is helpful in facilitating the coding or programming of the problem. A flow diagram or flow chart indicates the "flow" or steps in the computation which lead to the solutions of the particular problem.

A basic flow diagram usually lists the series of simple arithmetic steps which are to be performed by the computer. It is imperative that the coder be thoroughly familiar with the overall operations and peculiarities of each computer instruction so that he can construct the outline with regard to the capabilities of the computer. A description of each instruction is presented in Paragraph 3 of this volume, and tabular information on the instructions and the contents of the arithmetic registers before and after the execution of each instruction is presented in Content of Registers, Appendix A, Volume 1.

Usually more than one flow diagram is formed for the more complicated problems. The first flow outline may be nothing more than equations in mathematical language, written in the sequence in which they will be computed together with brief explanations of the steps involved. The second flow chart usually formulates the flow of computation as the problem will be computed on the computer. This chart will usually contain the instructions necessary for the data input, the instructions that operate on the input data to obtain the solutions, and the necessary instructions for the output of the results.

Many times the problems are of such a nature that the second type of flow diagram will consist of many charts and/or diagrams, each a more detailed presentation of the preceding charts.

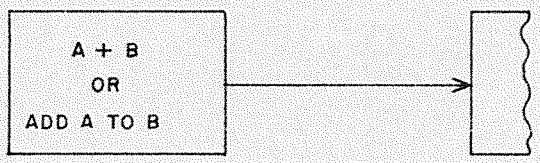
To facilitate the task of forming flow diagrams and to allow other programmers to understand a particular flow diagram, certain symbols have been more or less standardized. The following list of symbols is by no means complete, but is intended to give the coder an example of the basic symbols used in drawing the second type of flow chart:

PX 71871

1 LINES OF FLOW.

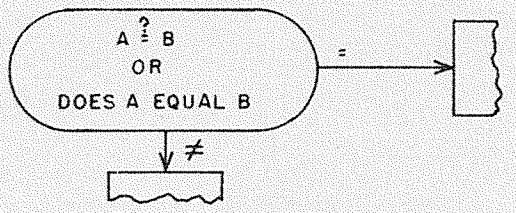
A solid line with an arrow touching the next element of the flow diagram is usually used to indicate the path to be followed by the computer; or more precisely, the path to be followed by the coder who is formulating the computer instructions from the flow diagram.

2 OPERATION SYMBOL.

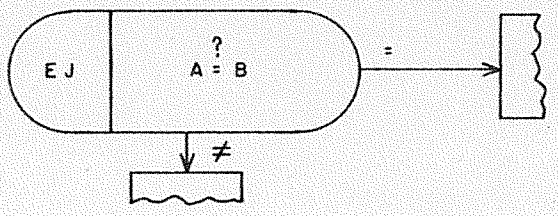


The rectangular box usually contains a statement about a computer or mathematical operation. The contents of the box may be a simple statement or a mathematical expression.

3 DECISION SYMBOL.

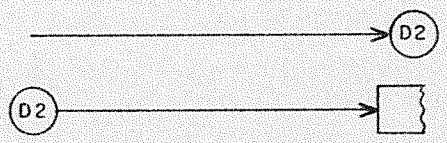


The symbol above is used to indicate a two-way decision. This symbol is sometimes written as



where the letters EJ designate the use of the Equality Jump instruction to make the decision in the computer.

4 CONNECTORS AND REMOTE CONNECTORS.



To eliminate as much as possible the crossing of lines of flow on a diagram, the above symbols are used to indicate a destination not easily reached in the diagram. Thus, the flow can be broken at a convenient point by terminating it in an arbitrary symbol which can be used to initiate the flow in another region of the paper.

5 EXAMPLE. - To illustrate the use of the above symbols, consider the following problem:

Compute the function $f(x_i)$ where

$$f(x_i) = \sum_{i}^{n} \frac{bx_i - c}{x_i + d}, \quad i = 1, 2, 3, \ldots, n$$

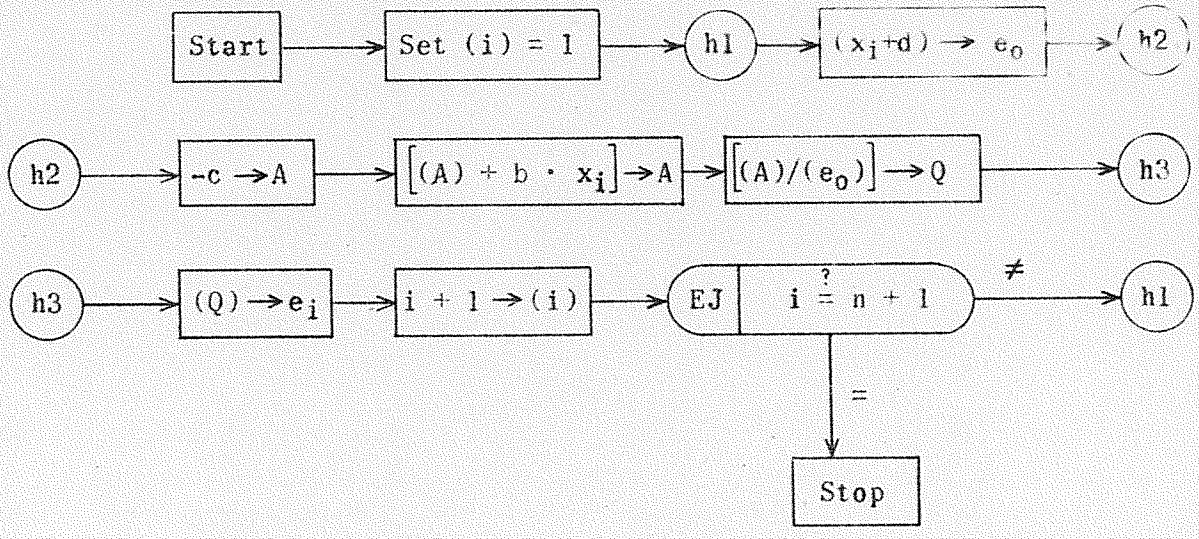
For this problem, it is assumed that b, c, d and x_i are integers and are contained in the computer in the Rapid Access Storage. Also, it is assumed that

$$x_i + d \neq 0$$

$$|x_i + d| < 2^{35}$$

The function $f(x_i)$ is stored at the location whose address is ei. The symbol (i) indicates the storage of the "i" term.

FLOW DIAGRAM



PX 71871

(b) CODING STEPS. - In summary, there are usually four phases in the preparation of a coded computer program:

1 The construction of a "flow diagram" or outline which shows the general computational steps to be accomplished.

2 The creation of a program, written in terms of 1103 instructions, using numeric or alphabetic symbols to indicate operation codes and the u and v address portions of the instructions.

3 If the program created in Step 2 is coded in octal, the program is ready for execution after being put onto some input medium and loaded into the computer. If the program is coded in symbolic notation, it must be translated into a binary representation before it can be automatically executed by the computer. Service programs for this translation may be used to facilitate this process by allowing the computer to perform the conversion.

4 The "debugging" (computer check-out) of the final program by means of trial runs on the computer.

The last phase, debugging, is described in the following paragraphs.

(7) DEBUGGING A PROGRAM.

(a) INTRODUCTION. - After the coded program has been written out completely and the manuscript carefully reviewed, the program must be put on some input medium for loading into the computer. The input mediums and the devices which transfer information from these mediums to the 1103 memory are described in Paragraph 4 of this volume.

Once the program has been stored in the computer in binary form, either directly from the input medium or by means of some translation routine, it is ready for execution by the computer. However, if the program is lengthy, it is likely that the coded program as stored has errors. Three common types of errors are:

1 Tape or card preparation errors made in the preparation of the program for input.

2 Coding errors such as listing incorrect or incomplete addresses, transposition of digits in the addresses or operation codes, transcription errors, etc.

3 Logical errors; incomplete or erroneous methods used to obtain the solution(s).

"Debugging" is the term applied to the process of locating errors in a program and correcting them. Debugging a program usually involves a series of trial runs of the program on the computer. Each time the program fails to run properly, the failure must be analyzed and the error corrected. Frequently one can immediately discover the error, correct it manually from the control console,

PX 71871

and proceed with the next trial run. At other times the detection of errors is more difficult and requires a thoughtful reconsideration of the program, keeping in mind any clues as to the origin of the error which may have been provided during the run. Error detection may be facilitated by the use of service routines coded expressly for this purpose.

The methods that can be employed to debug a program will vary widely from installation to installation depending on the nature of the programs, the service routines available, the availability of computer time, and the personal preference of the individual for one procedure over another. The following remarks suggest possible patterns to follow in debugging.

(b) TAPE OR CARD PREPARATION ERRORS. - This type of error can be minimized by systematic checking of all input tapes and cards before their information is read into the computer. The usual method that is used to prepare input paper tapes is as follows: (A similar routine is used in punched card preparation.)

1 A paper tape is punched from the coder's manuscript using the Flexowriter.

2 The resulting typed manuscript from the typewriter is discarded.

3 A new typed manuscript is prepared from the punched paper tape.

4 This typed manuscript is compared with the coder's manuscript to detect errors in punching.

5 All detected punching errors are corrected and a new corrected tape is prepared.

6 From the corrected punched tape a new manuscript is obtained which should be retained by the coder to use while debugging.

7 If seven-level codings are used they should be sight-checked before an attempt is made to read the tape into the computer.

(c) MANUAL DEBUGGING. - Manual debugging is the process which is used to locate errors in a program by controlling the operations of the program from the Supervisory Control Panel and visually checking these operations as they occur and are indicated on the control panel. Although manual debugging does not make efficient use of computer time, it can, if done correctly, reduce the overall time required to debug a program because of the versatility afforded by a "thinking" programmer at the control panel.

After loading the program in the computer, a run at high speed is usually tried. (For a discussion of the various speeds at which the computer may run, see Paragraph 5. Many of the suggestions for program debugging require a knowledge of operating the computer from the control console. This is also described in Paragraph 5.) This run may show the program to be free of errors, in which case there is no debugging to be done, or this run at high speed may end immediately with a computer fault. Frequently an MCT (Main Control Translator) fault or SCC (Storage Class Control) fault will arise. The former

indicates an illegal operation code, the latter an illegal address. (Computer faults are described in subparagraph d, Paragraph 5.)

When a fault occurs, the contents of PAK (Program Address Counter) should first be noted. This will almost always indicate the exact location of the erroneous instruction.

The SCC fault usually arises from either a transcription error, a punching error, or incorrect modification of an instruction by the program. These faults are ordinarily easy to correct. An MCT fault may be caused by a punching or transcription error. Frequently it will result from an erroneous program jump. In this case the address in PAK may indicate that the jump was to an area which is not occupied by the program. This fault must be traced to the instruction which caused the jump.

The computer may also stop on a DIVIDE fault or an OVERFLOW fault. Erroneous coding may cause these faults; or they may be the result of an error in judgement concerning the range of the numbers involved in the computation.

Obviously, the first phase of debugging is to correct the program so that it will run without computer faults.

Although the program is running without incurring computer faults, errors in its execution may still be detected. For instance, the program may be coded to produce output but no printing or punching occurs. Correction of this type of error may be facilitated by observation of the monitoring oscilloscope located on the upper center section of the Supervisory Control Panel (see Plate 6-9). This oscilloscope displays a point for each of the 1024 registers in RAS every time one of the registers is referenced. If, while debugging a program, the scope displays a distinct non-terminating repeated pattern (one dot or many) for a questionable period of time, it can be assumed that the computer is executing an erroneous "loop". This usually means that one or more of the "decision" instructions has an incorrect jump reference causing a return to itself or to a series of instructions which lead back to the incorrectly-addressed jump instruction.

It may be possible to detect from the oscilloscope the area in which such a loop is operating; or it may be desirable to FORCE STOP the computer, select MANUAL STEP OPERATION, and, by executing one instruction at a time, trace the operation through the loop. By noting each jump to a new address and traversing the loop at least once, the incorrect jump will probably be located.

Thus, the second phase of debugging is to eliminate any non-terminating loops.

After these two phases of debugging a program have been completed, output from the corrected program will reveal if the program is being executed properly. If the output format is in error, an investigation of that portion of the program devoted to output is necessary. This may be accomplished by entering the output routine with a typical result and, if the output routine is short, executing the routine one instruction at a time, observing the results at each step. If the output routine is complicated, it will probably be necessary to inspect the manuscript of the coded output routine, using the

PX 71871

erroneous results as a guide in looking for the trouble. (Output routines from the subroutine library would not be expected to cause any difficulty.)

In summary, the third phase of debugging is to correct any output routines used in the program.

Since frequent output during the running of a program is a positive indication of proper or improper running, it is often convenient to provide for output of partial results of computation. Since these results are of no interest when the program is known to be running properly, and since any output from the computer is time consuming, routines providing such output should be entered via an optional jump. During the debugging process the optional jump switches may be "on" to provide the output for monitoring purposes. After the program is completely checked, the optional jump switches may be "off", thus eliminating the intermediate output. An intermediate printout may even be of value when the program is in actual use: if normal output is very infrequent or onto a medium which is not easily read, an operator may wish to occasionally sample the intermediate output as a check on the program's operation.

Since the problem in debugging a program is to isolate each of the errors, it is desirable to have the coding arranged in small segments. The coder should be sufficiently familiar with his program to anticipate the results of each of the segments of the computation. In particular, he should be able to recognize an incorrect result.

A convenient way to examine the results of these program segments is to terminate each with an OPTIONAL STOP. Like the optional jumps described above, these stops may be "on" while the program is being debugged and then "off" when the program has been checked out. If optional stops have been coded in the program, the program may be executed from one stop to another until trouble is encountered. Then, because it is known in which segment the error lies, that section may be re-run on MANUAL STEP OPERATION.

The points in a program at which these optional stops and jumps are placed are called "breakpoints".

Thus, the fourth phase of debugging makes use of programmed breakpoints to isolate errors.

The instructions for optional stop and optional jump each provide three selections which incur the jump possibility. Although the number of options may be increased by combinations of the selections, an option at all desirable breakpoints can not be provided in a lengthy program. Regardless of how judiciously the breakpoints have been selected, in an actual debugging process, a breakpoint is often desired where an option for one has not been coded in the program. In this case it is possible to superimpose a breakpoint on the program. If a stop, for instance, is desired at address a and addresses b and b+1 are unused, the instruction at a may be replaced with the instruction

56 00000 b.

PX 71871

The program instruction which was in a is then stored in b, and the instruction

45 00000 a+1

is stored in b+1.

Thus, the fifth phase of debugging is to insert breakpoints if programmed breakpoints are inadequate.

Thus far the process of debugging has been described in terms of the programmer himself manually debugging his routine.

It cannot be over emphasized that in order to do manual debugging, the coder must be thoroughly familiar with the problem, its coding, and in addition, with the peculiarities of the computer.

(d) DUBUGGING WITH SERVICE ROUTINES. - A number of service routines exist which facilitate the process of program debugging. Consider phase 1 of the process of debugging, the elimination of computer faults. Two types of errors may be hard to find: (1) an SCC fault arising from improper modification of an instruction and (2) an MCT fault arising from an incorrect jump. The search for either of these errors may be facilitated by an "address sort" routine. Such a routine scans all instructions of the program for a given address. Any instruction containing that address is printed with its location. In case (1) the trouble may be located by a search for references to the modified instruction; the faulty jump in case (2) may be located by a search for references to (PAK)-1 at the time of the MCT fault.

A service routine of use in eliminating faulty loops is a "jump trace" routine. A "trace" or "automonitor" is a routine which executes interpretively all the instructions of a program which is being debugged. As each instruction is executed, the instruction itself and its address are printed out together with (A), (Q), (u), and (v). In the case of a jump trace routine, print-out occurs only if the instruction is a jump. With this print-out, the sequence of jumps in a program can be followed and those which are incorrect can be noted.

When a program has been run, a "memory dump" will often reveal the operations which occurred during the run. A memory dump routine lists octal print-outs of the contents of those memory locations containing the program. The examination of indices, temporary storage, modified instructions, etc., usually reveals the portions of the program which were improperly executed. This aids in locating the improper modifications of the program and recovering partial results. A "changed word post-mortem" routine is a selective memory dump in which only those words which have been changed in the course of running a program are printed. A changed word post-mortem routine requires that the original program as loaded be stored in the memory. After the program is run, a word-by-word comparison is made of the original program and the executed program. The advantages of a changed word post-mortem routine over a memory dump routine are that it operates faster, and there is less print-out to be examined. The memory dump, by virtue of printing all the program, gives evidence of misloading or punching errors which may not be detected by a

changed word post-mortem.

The use of breakpoints in debugging a program can be facilitated by any one of several "breakpoint routines". The simplest of these merely replaces selected instructions with manual stops and stores separately the instructions and the jumps back into the program. Upon reaching one of these stops in the program, the programmer may (1) examine the contents of certain locations which contain partial results, or (2) manually step through the next part of the program. A more elaborate breakpoint routine may provide for automatic printout of the contents of certain registers at a breakpoint. Usually these printouts are either in octal or decimal notation. It is also possible to invoke at a breakpoint a trace routine which prints the results of each of the next few instructions. A great saving in time is effected by using these sampling and tracing routines at a breakpoint rather than manually examining the results at that point.

The debugging service routines which have been described are of two types, "static" and "dynamic". The static type is employed after a program has been run and has stopped. The memory jump and address sort routines are examples of static routines. The dynamic routines operate in conjunction with the program as it is run. These may be "executive" routines which execute interpretively the program instructions and provide printouts of the results, i.e., trace routines; or they may be routines to which control is transferred at specific breakpoints, i.e., sampling routines. The static debugging routines usually require less computer time for their operation then the dynamic type. Therefore, in practice they are used more often although more information can be obtained from the dynamic type.

(e) ERROR CORRECTION. - It may be possible during the debugging process to correct a routine by making simple alterations in the contents of certain memory locations. After these corrections have been made a new input paper tape or punched card is usually needed for reloading the corrected version of the routine. To obtain the corrected version which is in the memory of the computer, a "punch storage for reload" or "bioctal dump" service routine may be used. This produces a paper tape or deck of punched cards containing the program coded in octal with appropriate insert and check addresses.

In order to conserve computer time while making a number of manual corrections, the following method is suggested:

Select Manual Step Operation

Select Drum Clock Source

Select Test

Select MD Start

Set MPD to 3

Set MCT to 75 (Repeat instruction)

PX 71871

Set UAK to 70000 (set j to 7)

Set VAK to 00000

Set X to ll 10000 v, where v is the first address to be written into.

Press Start button

Press Step button

Set in Q-Register the word to be written
Press Step button
Clear Q-Register
} Repeat

It should be noted that a j of 7 is being used which sets up an unterminated repeat sequence with both the u and v-addresses of the repeated instruction being advanced upon each execution. Since this is the case it becomes comparatively simple to read back into Q for checking purposes the words just written. This procedure is

Clear UAK

Set UAK to u where u is the first address which was written into.

Clear VAK

Set VAK to 10000

Press Start button
Press Step button
} Repeat

If the data that is to be written or to be read is not in consecutive memory locations, the following steps must be added at the end of both of the above repeated steps.

Clear VAK (UAK if reading)

Set VAK (UAK if reading) to the address to be written into or read from.

The process of program debugging has been described in terms of the programmer himself operating the computer. Manual debugging clearly requires a thorough knowledge of the program. However, the service routines may be applied by an operator with perhaps instructions from the programmer. The extent to which programmers will operate the computer for program debugging will vary from one installation to the next.

(8) OPERATING PROCEDURE. - The service routines which have been described as aids in debugging should be available for use whenever trouble arises. Since the need for these routines cannot always be anticipated, it is desirable that they be at all times in the computer memory. Many installations have reserved areas on the magnetic drum and/or magnetic tape for permanent storage of the service library. Except when necessary, programs are not stored in these areas. This practice eliminates the need for frequent reloading of the service routines.

In addition to the service routines for debugging there are service routines which facilitate operating the computer. Chief among these, in the case of a computer with a Ferranti Tape Reader, is the program for reading information from paper tape. Such a program may be coded to accept information punched in any form whatsoever. In particular, the program might be designed to load bioctal tapes with the 7th level control configurations described in Paragraph 4.

Routine for Loading Bioctal Tapes with a Ferranti Reader

00000	45	00000	00006	Jump to start
00001	45	30000	00003	Optional jump to bypass stop
00002	17	00005	00002	Stop reader
00003	11	00035	40000	Enter data
00004	21	00003	00037	Advance address
00005	45	30000	00007	Optional jump to bypass start
00006	17	00006	00006	Start reader
00007	76	00000	10000	Read to Q
00010	31	00035	00006	Isolate data levels
00011	52	00027	00035	
00012	31	00036	00001	Isolate seventh level
00013	52	00030	10000	
00014	51	00030	00036	
00015	43	00032	00001	Test for enter data
00016	43	00031	00021	Test for insert address
00017	43	00033	00023	Test for check address
00020	45	00000	00007	

PX 71871

00021	16	00035	00003	Insert address
00022	45	00000	00007	
00023	11	00003	20000	Check address
00024	36	00034	20000	
00025	43	00035	00007	
00026	57	77777	77777	
00027	00	00000	00077	Data levels mask
00030	00	00000	17700	Seventh level mask
00031	00	00000	11100	Insert address code
00032	00	00000	10100	Enter data code
00033	00	00000	10500	Check address code
00034	11	00035	00000	Constant for check address
00035	--	-----	-----	Word assembly
00036	--	-----	-----	Instruction code assembly
00037	00	00000	00001	Constant for advance address

If the program to be loaded is short enough to allow the storage of a copy, it is helpful to have an "image" of the program stored on the Magnetic drum (MD). Usually the image on MD contains the two extra instructions

```
RP   jn   w
TP   MD   RAS
```

which cause the program to be transferred to RAS at the start of the computations. Thus, whenever it is necessary to restart the program, the unaltered version of the program may be transferred from MD without reloading the computer from paper tape or cards. This drum image is also available for use by service routines such as a "changed word post-mortem" routine.

A Ferranti Loading Routine may be used which automatically provides such a drum image of the program. Of greater importance, such a loading routine permits loading anywhere in RAS; the previously listed loading routine does not allow RAS addresses 00000 through 00037 to be loaded. Provided that the loading routine contains an "end of tape" detection, the loading routine may be supplemented to operate under the following conditions:

PX 71871

(a) The loading routine is stored on the magnetic drum.

(b) The loading routine proper is prefixed with instructions which

(1) store the current contents of RAS as a drum image and

(2) "bootstrap" the loading routine proper into RAS for operation.

(c) All data being loaded into RAS under control of the loading routine is in fact loaded in the drum image.

(d) Upon detecting "end of tape", control is transferred to suffixed instructions (on the drum) which transfer the drum image to RAS and stop the computer.

The coding for such a bootstrap operation follows:

L_0	11	00000	76000	store F_1
L_1	75	31777	L_3	store (RAS) in drum image 76000 - 77777
L_2	11	00001	76001	
L_3	75	3 n	00000	transfer loading routine proper to RAS for operation
L_4	11	L_5	00000	
L_5	45	00000	L_x	entrance to loading routine
L_6				loading routine proper

L_{n+4}				
L_{n+5}	75	31777	L_{n+7}	transfer of drum
L_{n+6}	11	76001	00001	image to RAS
L_{n+7}	11	76000	00000	
L_{n+10}	56	00000	L_0	stop; re-entry to loading routine.

It is desirable that a bootstrap procedure similar to this be used with most service routines so that operation of the service routine does not destroy the status quo of RAS.

Another practice which many consider helpful is to "erase" the computer memory before loading a program for debugging. This means that erroneous jumps to an address not used by the program result immediately in an MCT fault. Also program sections separated by erased portions of RAS are easily recognized in a memory dump. This erasing, or writing of zeros in every location, may be

PX 71871

effected by a repeated Transmit Positive from Q with (Q) = 0. This may be set up manually, as previously explained, or included in the service library.

Frequently service librarys include routines to simplify manually reading information out of storage or writing information into storage. If these routines print out the word read or entered, a written record is provided of all corrections made in the course of running a program.

When a library of subroutines is kept on magnetic tape, the service library usually includes a routine to extract the subroutines from file. Input translation routines may also be in the service library.

It is important that a programmer be thoroughly familiar with any service routines in use at his installation. There are restrictions to the use of many of these routines of which he must be aware. For instance, the "Routine for Loading Bioctal Tapes with a Ferranti Reader" cited previously does not permit loading in addresses 00000-00037.

Before coding any routine, a programmer should acquaint himself with the operating conventions of his installation. He should know such particulars as the following:

(a) The service routines available and the facilities and restrictions of each.

(b) The input translation routines in existence.

(c) The procedures necessary in order to use library subroutines: their assembly, their reference to a constant pool, a temporary storage pool, an alarm print, etc.

(d) The areas, if any, of the computer memory which are reserved for special purposes.

c. NUMBER NOTATION.

(1) INTRODUCTION. - Because the theory of operation of the computer is based on binary logic, the coder should become thoroughly familiar with one's complement binary arithmetic. The following list of publications are recommended as references to supplement the discussion of binary arithmetic presented in Paragraph 2 and Appendix 6A of this volume.

(a) Stifler, W. W. Jr., (ed), High Speed Computing Devices by the staff of Engineering Research Associates, pp. 74-99, (McGraw-Hill, New York, 1950).

(b) Richards, R. K., Arithmetic Operations in Digital Computers, pp. 1-22, (D. Van Nostrand Co., Inc., New York, 1955).

(c) Uspensky, J. V., Elementary Number Theory, (McGraw-Hill, 1939).

PX 71871

As stated previously, for convenience octal instead of binary notation is used to describe computer words and addresses. It should be remembered, however, that the logic of computer operation is based on the binary system.

(2) RADIX CONVERSION. - In most problems the data for a program is presented in some form other than binary. In particular, data is frequently decimal. In Appendix 6A procedures for converting numbers from one radix (base) to another are described. It is usually desired that the computer perform the necessary radix conversions of input and output data.

If, for example, the computer were used for a problem involving large amounts of decimal data, an efficient method for converting from decimal to binary would be necessary. A "scalar product routine", such as that which follows, could be used for this conversion provided it is known that all decimal data are integers.

Let N be the desired binary number; then

$$N = a_0 + 10a_1 + 10^2 a_2 + \ldots + 10^n a_n$$

To form this scalar product, the individual products of the binary-coded decimal digits of the decimal number and their corresponding powers of ten, binary coded, are accumulated.

Routine to Convert Decimal Integers to Binary Integers.

e1	RS	A	A	Clear A to zero
e2	RP	3,n+1	e4	Form N
e3	MA	u_1	v_1	
e4	--	--	--	Next instruction
u1			1	10^0
u2			12	10^1
.			.	.
.			.	. Powers of ten coded
.			.	. in binary.
un+1				10^n
v1			a_0	Binary - coded decimal digits
v2			a_1	
.			.	
.			.	
.			.	
vn+1			a_n	

PX 71871

The result N (in binary) appears in the Accumulator.

The reverse operation, conversion from binary to decimal, can be accomplished with a repeated division. Assuming that the positive integral number N to be converted to decimal is contained in the Accumulator, the steps are:

f1	RP	3,n+1	f3	Form binary - coded decimal digits
f2	DV	u1	v1	
f3	--	--	--	Next instruction
u1				10^n
.			.	Powers of
.			.	ten coded in binary
.			.	
un			12	10^1
un+1			1	10^0
v1			a_n	
v2			a_{n-1}	
.			.	
.			.	Binary - coded decimal digits
.			.	
vn+1			a_1	

The Divide instruction (DVuv) causes the number in the 72-bit Accumulator to be divided by the number at address u. The quotient is deposited at address v, and a positive remainder is left in the Accumulator. In the above example, descending powers of ten are brought in as successive divisors of the diminishing remainder, then the series of quotients are the binary-coded digits of the decimal representation of the number initially contained in the Accumulator. These quotients are deposited at a series of consecutive addresses v_1, v_2, ... v_{n+1}.

To illustrate binary to decimal integer conversion, consider the following:

Convert n binary words from any n consecutive memory locations to decimal, and print these decimal integers on the typewriter with appropriate sign. This subroutine assumes that the Q-register, upon entry into the subroutine, contains a code word which specifies the number n and the address of the first binary number; i.e.,

(Q) = 00 xxxxx -----

where ----- specifies the number n and xxxxx specifies the address of the first datum.

PX 71871

Y0	MJ	0	30000	Exit line of subroutine
Y1	PR	0	b 6	Shift typewriter to lower case
Y2	TU	Q	Y 7	Set in initial location of data
Y3	TV	Q	b 2	Set in counter n
Y4	IJ	b2	Y 6	Test n: if n=0, no numbers are printed
Y5	MJ	0	Y 0	Exit - completion of routine
Y6	PR	0	b 1	Perform carriage return
Y7	TP	x_i	A	Place i^{th} number to be printed in A
Y10	SJ	Y11	Y14	Test sign of number
Y11	PR	0	b 3	Print negative sign
Y12	TM	A	A	Put absolute value of number in A
Y13	MJ	0	Y15	Skip next instruction
Y14	PR	0	b 4	Print plus sign
Y15	PR	0	b 5	Print space
Y16	RP	3,13	Y20	Form binary-coded decimal digits
Y17	DV	b7	Y22	
Y20	RP	2,13	Y22	Form print instructions
Y21	RA	Y22	b 0	
Y22			0	Print 11 decimal digits for each number. (Initially the binary-coded decimal digits are stored here until they are replaced by the appropriate print instructions.)
Y23			0	
Y24			0	
Y25			0	
Y26			0	
Y27			0	
Y30			0	
Y31			0	
Y32			0	
Y33			0	
Y34			0	

PX 71871

Y35	MJ	0	Y4	
b0	PR	0	d0	Dummy print command
b1			45	Flexowriter code for carriage return
b2			0	Counter n, zero initially
b3			56	Flexowriter code for negative sign
b4			54	Flexowriter code for positive sign
b5			04	Flexowriter code for space
b6			57	Flexowriter code for shift to lower case
b7			-	10^{10}
b10			.	10^{9}
b11			.	10^{8}
b12			.	10^{7}
b13			.	10^{6}
b14			.	10^{5} } Binary-coded powers of ten
b15			.	10^{4}
b16			.	10^{2}
b17			12	10^{1}
b20			1	10^{0}
d0			37	
d1			52	
d2			74	
d3			70	
d4			64	} Flexowriter codes for the decimal digits, 0 through 9.
d5			62	
d6			66	
d7			72	
d10			60	
d11			33	

PX 71871

Attention is called to the four instructions (Y17), (Y20), (Y21) and (Y22). The first two of these instructions form by repeated divisions the binary-coded decimal digits of each number to be printed. The second pair of instructions adds to each of these binary-coded decimal digits a dummy print instruction which addresses in its v-address portion the starting address of a table of Flexowriter codes for the digits 0 through 9. For example, suppose that the first binary-coded decimal digit is a five. This is stored at Y22 as the result of the first division. Then, as a result of the Replace Add instruction at Y21, the content of Y22 would be replaced by PR 0 d5. Thus, when this instruction is executed by the computer, the digit "5" would be printed by the typewriter.

(3) SCALING. - It should be emphasized that the above routines are for binary/decimal conversion of integers. Words in the 1103 are considered to be integers in the sense that the computer has been designed so that for all arithmetic operations the binary point is thought to be to the right of the right-most bit. This does not mean, however, that only computations with integral numbers can be performed in the 1103.

Whenever a problem involves a quantity s which either (a) exceeds the range of integers which may be represented in a 36 bit word ($1-2^{35} \leq s \leq 2^{35}-1$), or (b) has a fractional part, that quantity s may be represented as $s = s_1 \cdot 2^{s_2}$, where s_1 is an integer within the range of a computer word. The quantity 2^{s_2} is called a "scale factor", s_1 is called the "mantissa" of the number s, and s_2 is called the "characteristic" of s. A positive characteristic is used to represent integers outside the range ($1-2^{35}$, $2^{35}-1$); a negative characteristic is used to express numbers with fractional parts.

Thus, in a computer operating upon integers, the machine representation of a number is the actual value of the number multiplied ("scaled") by some constant 2^l. It must be remembered that when the machine executes an arithmetic operation involving scaled numbers, the scaling may change. Suppose s_m and t_m are the machine representations, each scaled 2^{20}, of s_e and t_e. The result r_m of a Multiply instruction MP $\{s_m\}$ $\{t_m\}$ is then the machine representation of $r_e = s_e \cdot t_e$ scaled 2^{40}.

$$s_m = s_e \cdot 2^{20} \qquad t_m = t_e \cdot 2^{20}$$

$$s_e \cdot 2^{20} \cdot t_e \cdot 2^{20} = s_m \cdot t_m = r_m = r_e \cdot 2^{40}$$

It may then be necessary to "scale down" r so that its machine representation does not exceed the capacity of a 36 bit word. Thus the choice of scaling for a number depends not only on the number of fractional bits to be represented but also on the machine restriction of 36 bits per word. In general the coder must consider three things in scaling:

(a) All numbers must be scaled so that the "machine representations" of these numbers are integers.

(b) The routine should be coded so that the scaling is always known.

(c) Frequent rescaling may be necessary in the routine if the values of the numbers are not to exceed the capacity of an 1103 word.

PX 71871

In many computations involving scaled numbers, the desire to retain the maximum amount of "precision", maximum number of significant bits, creates a problem. To solve this problem, the numbers that enter into the calculations, as well as the numbers resulting from these calculations, are "normalized". A normalized number is a number which has been scaled up in a register so that the left-most stage of the register contains the sign bit of the number with the most significant bit of the number contained in the next adjacent stage.

The following are examples of normalized numbers:

30 00000 00000 is 00 00000 00003, (+3) normalized

47 77777 77777 is 77 77777 77774, (-3) normalized

Numbers are normalized easily in the 1103 by using the Scale Factor (SFuv) instruction. This instruction normalizes the number $(u)_i$ in A_R and stores the number indicating the left shifts necessary to restore the number $(u)_f$ to its original state. This shift count k is stored in the v-portion of the register whose address is given by the v of the Scale Factor instruction.

In case the u-address of the Scale Factor instruction is the Accumulator, the relationship between the initial and final contents of the Accumulator is as follows:

$(A)_f = (A)_i \cdot 2^s$ where s is the scale factor.

The relationship between s and k is

$s = -k$ if $0 \leq k \leq 36$

$s = 72-k$ if $37 \leq k \leq 71$

If $k = 0$, $(A)_i$ was properly positioned before shifting.

If $k = 37$, $(A)_i$ is all ones or zeroes.

To illustrate the use of the Scale Factor instruction, consider the problem of normalizing the product of two numbers x_m and y_m where these numbers are themselves scaled machine copies as follows:

$x_m = x \cdot 2^r$

$y_m = y \cdot 2^s$

The routine would be as follows:

```
C1    MP    v1    v2    form product
C2    SF    A     t1    normalize (A)
C3    TP    A     t2    place normalized product in t2
```

C4	TP	t1	A	place k in A
C5	TJ	t3	C7	is k < 37 ?
C6	RA	v3	t4	add 72_{10} to (v3)
C7	RS	v3	t1	subtract k from (v3)
C10	--	--	--	next instruction
t1			0	k, (zero initially)
t2			0	normalized product
t3			45	constant, 37_{10}
t4			110	constant, 72_{10}
v1	--	--	--	x_m
v2	--	--	--	y_m
v3	--	--	--	r + s initially

The product formed by the first instruction is

$$(v_1) \cdot (v_2) = x \cdot y \cdot 2^{r+s}$$

The remaining instructions form the product

$$(t_2) = x \cdot y \cdot 2^{(v3)_f}$$

where $(v3)_f$ is

$$r + s - k \text{ if } k < 37$$

$$r + s + 72 - k \text{ if } k \geq 37$$

An overall relationship would be expressed as follows:

$$x \cdot y = (v_1) \cdot (v_2) \cdot 2^{-(v3)_i} = (t_2) \cdot 2^{-(v3)_f}$$

It should be noted that the 21 leftmost stages of t1 must initially contain zero since the Scale Factor instruction writes only into the lower order 15 stages of t1.

If a routine is coded such that only the mantissas of scaled numbers are stored in the computer and operated upon by the program, the routine is said to be coded in <u>fixed point</u>. In this case the programmer must himself keep an account of the characteristics associated with each mantissa at every step of the computation. The programmer must also be aware of the size of the mantissas and scale them down when there is danger of exceeding the capacity of

PX 71871

a computer word. On the other hand, if the mantissas are carried in normalized form, the result of every arithmetic operation must be normalized.

The programmer must be exceedingly careful in the record which he keeps of the characteristics. A mistake may result in his having to rescale the problem from that point on. This accounting is generally listed on the coding sheets; the characteristic is recorded beside each instruction which changes the scaling of a mantissa. Great care must also be exercised in judging the magnitude of the mantissas; again, if unanticipated overflow occurs, the problem may have to be rescaled.

It is evident that the fixed point coding of a problem can be complex. Consequently it is sometimes preferable to store in the computer the characteristics as well as the mantissas and use subroutines which, before each arithmetic operation, effect the scaling of the mantissas on the basis of their associated characteristics. The result of each arithmetic operation is then stored as a mantissa with associated characteristic. This manner of coding is known as floating point coding. Various schemes may be used for the storage of the mantissa and characteristic of a floating point number. If the mantissa and characteristic are stored at separate locations, the floating point number is said to be "unpacked"; if the mantissa and characteristic are stored together at one location, the number is said to be in "packed" form. A common floating point packed form for the 1103 allows 28 bits for the mantissa and eight bits for the characteristic. Generally the mantissa is normalized although this is not necessary. Should one wish to retain an indication of the number of significant bits in the results, the mantissas may be carried unnormalized.

As an example of one floating point packed form which has been used on the 1103, consider the following representation of the number s by the components s_1 and s_2 of 28 bits and eight bits, respectively:

$$\text{If } s \neq 0 \quad 1 > |s_1| \geq 1/2, \quad 256 > s_2 + 128 \geq 0$$

$$\text{If } s = 0 \quad s_1 = s_2 + 128 = 0$$

The number s is represented in the computer by $s_1 \cdot 2^{35}$ and $s_2 + 128$, the machine forms of its mantissa and characteristic. Note that the mantissa is normalized. When the characteristic is represented in this way, as $s_2 + 128$, it is said to be a "biased" characteristic. (The characteristic might also be represented by eight bits as a "signed" characteristic, $128 \geq s_2 \geq -128$.) The packed representation of the number s is positioned at a location as follows:

28 bits	8 bits
normalized mantissa	biased characteristic

The choice of a representation for floating point numbers is influenced by many considerations which may vary from installation to installation.

Elaborate systems of subroutines have been developed to perform arithmetic operations with floating point numbers. The price which the programmer pays

PX 71871

for relief from the complexities of fixed point scaling is increased execution time for the arithmetic operations. For instance with a floating point system which uses a packed representation, the execution times for the operations of addition, multiplication, and division may be on the order of 60 times that of the corresponding machine operations. Floating point operations with unpacked operands are considerably faster; here the multiples of the increase in time over the computer operations are approximately 20, 3, and 3 for addition, multiplication, and division, respectively. In general, floating point coding is used to reduce the elapsed time between receipt of a problem and the production of answers. Floating point notation is particularly useful in problems in which the numbers involved vary widely and where only crude predictions can be made of the amount of variation.

(4) MULTIPLE PRECISION. - In addition to scaling in order to modify the range of numbers which may be represented in the computer, it is possible to extend the range by representing numbers in a double precision, triple precision or, in general, n-precision form. In this case two, three, or n computer words are used for the storage of a single number. For example, two locations are reserved in storage to represent a double precision number; the representation may be (A_L) and (A_R) when the number is in the Accumulator. The addition of two numbers so represented is easy and fast using the Split instructions. In the case of multiplication or division, an end correction must be provided for each word whose sign bit is "1". Another method is to select a positive constant k and represent a double precision number N as

$$N = q \cdot k + r$$

with a side condition on r to require unicity. Both q and r are single precision numbers. For example, let $k = 2^{34}$ with $0 \leq r < 2^{34}$. With this representation addition is slower but multiplication and division are markedly easier.

Similar schemes may be used for n-precision computing. Again, a number of considerations enter in the choice of a representation of multiple precision numbers.

(5) CHOICE OF NUMBER NOTATION. - In choosing the number representation to be used in solving a given problem on the computer, the accuracy which is required is first considered. If a large number of significant bits is demanded, a multiple precision representation may be required. Secondly, the choice between a fixed point and a floating point notation is made on the basis of running time on the computer versus programming time. Floating point routines are frequently used to simplify the coding of "one shot" programs (the program is to be used once) where the emphasis is on elapsed time from problem formulation to solution. Generally if a program is to be run repeatedly, it is coded in fixed point in order to minimize the running time.

d. NOTES ON THE INSTRUCTIONS IN THE 1103 REPERTOIRE. - In the following paragraphs, pecularities, tricks, and pitfalls which confront a programmer are discussed. In almost all cases the unusual circumstances which must be noted are due to particular features of the 1103 logic; certain of these features create problems which are not immediately obvious.

(1) OPERATIONS INVOLVING THE ACCUMULATOR.

(a) NEGATIVE ZERO. - Since the basic arithmetic operation of the 1103 is subtraction and the "1's" complement number system is used, negative zero (72 "ones" in the Accumulator) cannot be generated by any series of arithmetic operations. Consider the operations performed by the instructions below:

```
C1      RA    e1      e2        form sum
C2      RS    A       e3        form difference
.
.
.
e1      77 77777 77776          -1 in decimal
e2      00 00000 00001          +1 in decimal
e3      77 77777 77777          -0 in decimal
```

Using for exemplary purposes single length registers of four bits and an Accumulator of eight bits, the operations can be shown as follows:

Contents of Accumulator	Operations	
0000 0000	clear A	
0000 0001	"add D(e1)" by subtracting D(e1)'	Operations of RA at C1
1111 1111		
1	end around borrow	
1111 1110		
1111 1110	"add D(e2)"by subtracting D(e2)'	
0000 0000	result of executing (C1)	
0000 0000	transmit (A_R) to X, then clear (A)	Operations of RS at C2
1111 1111	"add D(X)" by subtracting D(X)'	
0000 0001		
1	end around borrow	
0000 0000		
1111 1111	subtract D (e3)	
0000 0001		
1	end around borrow	
0000 0000	result after executing (C2)	

Continuing the above example, negative zero may be formed in the Accumulator using one of the "logical" instructions:

```
C3      SS    e3      0         form - (2^36 -1) in A
C4      CC    A       e3        form negative zero in A
```

i.e.,

0000 0000	result after executing (C2)	
0000 1111	form (A) - S(e3)	Operations of SS at C3
1111 0001		
1	end around borrow	
1111 0000	result of executing (C3)	

1111 0000	transmit (A_R) to X, then clear (A_R)	} Operations of CC at C4
1111 0000	form logical sum of (X) and (A_R)	
1111 1111	form logical sum of (A_R) and (e_3)	
1111 1111	result after executing (C4)	

Negative zero can be generated in the Accumulator by other series of instructions concluded by the Controlled Complement instruction (CCuv). Note that in the above example the content of e3 (normally considered a negative zero, mod $2^{36}-1$) was considered as $2^{36}-1$ (mod $2^{72}-1$). Such is the case when any of the "split" or "logical" instructions are used.

If the above routine is continued with the execution of the following instruction, note that the content of the Accumulator (all "ones") must be considered as zero.

C5 LA A k shift (A) left k places

i.e.,

1111 1111	result after executing (C4)

1111 1111	do not clear (A), but clear (X)	} Operations of LA at C5
1111 1111	"add D(X)" to A by subtracting D(X)' from A	
0000 0000	(A) is now all zeros	
0000 0000	shift (A) left k places	
0000 0000	result after executing (C5)	

Thus, if the contents of the Accumulator are all "one's", its contents must be considered to be all zeroes except in the following cases:

If (C5) had been either an Equality Jump or Threshold Jump instruction, the result of testing the contents of the Accumulator, (negative zero), would be as follows:

If (u) is $2^{36}-1$ (negative zero),

EJuv would show an equality, take (v) as NI
TJuv would show (A) positive, continue present sequence

If (u) is all zeroes (positive zero),

EJuv would not show an equality, continue sequence
TJuv would show (A) negative, take (v) as NI

In none of these cases will the content of A be restored to negative zero; the Accumulator will be set to all zeros after the execution of any of these instructions.

If C5 had contained a Zero Jump (ZJuv), the Accumulator would test as not being equal to zero, but as the result of executing the ZJ the Accumulator would be set to all zeros.

(b) SINGLE OR DOUBLE LENGTH EXTENSIONS. - A common error involving transmissions to the Accumulator and a subsequent testing of its contents is illustrated by the following example:

```
C1      SP   e1   0          enter variable in A
C2      EJ   e2   -          compare (A) and (e2)
C3      --   --   -
.
.
.
e1      40   00000 00000    variable (here shown as a constant)
e2      40   00000 00000    a constant
```

After (C1) is executed, (A) is 00 00000 00000 40 00000 00000. The EJ at C2 tests to determine if (A) and D (e2) are equal which is not true in this case since D(e2) = 77 77777 77777 40 00000 00000. A correct method of testing for the equality of these two values, (e1) and (e2), could be coded as follows:

```
C1      TP   e1   a
C2      EJ   e2   -
C3      --   --   -
.
.
.
```

By using the Transmit Positive instruction, the double length extension of (e1) is formed in the Accumulator.

(c) COMMON PITFALLS. - Many instructions make use of the Accumulator without explicitly referencing it as one of the execution addresses.

It should be remembered that the initial contents of the Accumulator may be destroyed when executing certain of these instructions which use the Accumulator. In certain cases this may lead to erroneous results when the Accumulator is used as the u or v address of the instruction. A study of the Sequential Listing of Instructions, Paragraph 3.c., will indicate the peculiarities arising from such situations. The Content of Registers, Appendix A, Volume I, presents in tabular form the results of such peculiarities.

Similar peculiarities result from addressing the Q-Register as u or v of an instruction.

(2) SHIFT INSTRUCTIONS. - All shifts performed are <u>left</u> <u>end-around</u> shifts: when a shift is performed the left-most portion of the number shifted in the Accumulator (or Q-register) becomes the right-most portion of the number. It should be remembered that a left shift of "k" places is equivalent to a <u>modular</u> multiplication by 2^k. Depending upon the significance attached to the sign-bit, the modular value of the shifted number will be in the range of the system of <u>signed</u> numbers possible to represent in the register or in the range of the system of binary numbers directly represented in the register. If a significant "1" (or "0" if the number has a negative

representation) is shifted into the sign-bit position of the register, the "sign" of the shifted number must be regarded as opposite in sign to the initial number unless the shifted number is referenced for later use by one of the Split instructions.

The mathematical statement expressing a left shift of (u) by k is:

$$(u)\cdot 2^k - \left[(u)\cdot 2^{k-m}\right] 2^m + (u)\cdot 2^{k-m}$$

where the brackets mean the greatest integer contained in the register and

m = 72 for the Accumulator where $k \leq 72$
m = 36 for the Q Register where $k \leq 36$

An equivalent right shift can be obtained from a left shift by observing the following:

Right shift of k places = m-k left shifts.

When the shift count k of the Left Shift in A, LAuk, and Left Shift in Q, LQuk, instructions is in the range $k > 177_8$, transmission back to the u-address of the shifted quantity contained in A or Q can be stopped. The peculiarities resulting from a k of this size are discussed in the Sequenced Listing of Instructions, Paragraph 3.c.

This property of these shift instructions is useful whenever it is desired to leave the original contents of u undisturbed. To make the best use of this property, the u-address should be an RAS address and the v-address portion should be k plus 10000_8 or 20000_8 (the address of Q or A) depending upon in which register the programmer wishes the shifted result to be stored. For example, consider the instruction

LA 01000 20002

The operations resulting from this instruction are as follows:

Clear A

D (01000) → A

(A) is shifted two places to the left

The shifted result is <u>not</u> transmitted to 01000 but left in A.

A useful operation, possible because of this peculiarity of these Shift instructions, is

Transmit (positive) to A and Q the contents of u (for this case u must be an RAS address), after (u) has been shifted k places to the left; e.g.,

LQ 01000 20005 or also LA 01000 10005

PX 71871

The following operations result from these instructions:

(01000) → Q or (01000) → A

(Q) or (A) is shifted 5 places to the left

D (Q) → A or (A_R) → Q

If the u-address is not an RAS address when using this property of the Shift instructions, one of the following conditions will arise:

(a) If the u-address is a magnetic drum address, (A) or (Q) is returned to the drum but not to the original address; e.g.,

LA 41234 20011

resulting in

Clear A

D(41234) → A

(A) is shifted 9 places to the left

(A_R) → 61234

(b) If the u-address is an address of the Accumulator or the Q-register, care must be exercised to prevent the creation of an address not allowable, which will cause an SCC fault. Such would be the case in the following instructions:

LQ	10000	20003
LQ	20000	10107
LA	10000	20176
LA	20000	10177

(3) "ROUND OFF" AND "SCALE DOWN" OPERATIONS. - During the course of many arithmetic operations it becomes necessary to change the scaling of a number. A simple method of scaling a number down would be to perform a left shift of k places equivalent to the desired right shift. For example, the instructions below accomplish the following: multiply two numbers (u) and (v) and scale the resulting product down 5 places.

MP	u	v	form product (u) · (v)
LA	A	00103	right shift 5 or (72-k) places
TP	A	A	eliminate unwanted bits

PX 71871

The left shift instruction LA A 00103 (where k = $103_8 = 67_{10} = 72-5$) performs an end-around left shift of the product 67 places, equivalent to an end-around right shift of five places. This shifting operation is equivalent to a division by 2^5 if the Transmit Positive instruction is included and if it is assumed that the multiplication of (u) and (v) did not generate more than 40 significant bits. The instruction TP A A performs the operation of clearing the least significant bits which appear in (A_L) after the shift operation. If these least significant bits were left in (A_L) and further arithmetic operations were performed on the shifted product, they would no longer be considered as least significant bits but as the most significant bits of the contents of the Accumulator. In fact, the bits that were shifted into (A_L) could also effect the sign of the Accumulator in any further arithmetic operations.

In the above example the product was shifted down without any consideration to the arithmetic error (truncation or round-off error) that the process may generate. Usually before a "shift down" operation is performed in arithmetic problems, a "round-off" step is first programmed. In the above example the maximum error generated would be plus or minus one in the least significant place. The following method will minimize the round-off error to be in the range of plus or minus one-half in the least significant place. For simplicity let us assume that (u) and (v) are positive; then, to round-off and scale down, the instructions could be coded as

```
C1      MP    u      v        form product (u) · (v)
C2      SA    d1     00103    round-off and scale down
C3      TP    A      A        eliminate unwanted bits
.
.
.
d1                   20       binary 10000
```

In this example the product was scaled down five places after it was "rounded off" at the sixth place. The constant added would increase the number to be shifted down by one if the fifth bit of the number was "1". The general rule for rounding-off and scaling down by k places is to add 2^{k-1} (or subtract 2^{k-1} if the number to be rounded is negative) before the scaling operation occurs.

In "scale up" operations (multiplication by 2^k), round-off operations are not necessary, but care must be taken not to scale a number up beyond 71 bits, if the number is in the Accumulator, and 35 bits if the number is in the Q-register.

(4) ACCUMULATIVE OVERFLOW. - The possibility of an overflow beyond the stage A_{71} of the Accumulator during repeated executions of a Multiply Add instruction is automatically checked by the computer, but often it is desirable to know if an overflow occurs into A_{35} and beyond during repeated executions of Multiply Add or Add and Transmit operations. An overflow into A_{35} can be determined by executing the Equality Jump instruction with its u address referencing the Accumulator; e.g., EJ 20000 v. The equality of $D(A_R)$ and (A) is tested by this instruction; if there has been an overflow into A_{35}, $D(A_R) \neq (A)$. For example, if $(A)_i$ = 0000 1011, $D(A_R)$ = 1111 1011, indicating an overflow to or beyond A_{35}.

PX 71871

(5) DIVIDE OVERFLOW. - The Divide instruction (DVuv) divides the 72-bit integer in the Accumulator by the 36-bit integer at address u, placing the quotient in the Q-register and at address v, and leaving in A a positive remainder. Therefore, care must be exercised to insure that the quotient does not exceed 36 bits in length (35 bits plus a sign bit). If the quotient should exceed 36 bits, the computer will stop, giving a divide overflow indication on the control panel. The divide operation as performed in the 1103 is defined as

$$(A)_i = (Q) \cdot (u) + R \text{ where } 0 \leq R < |u|$$

It should be remembered that in the above definition, (Q), R, (u), and (A) are all integers.

Also to be noted is the following condition taken into account by the Divide instruction: Consider the definition for all divide cases when (A) is negative and when R, the remainder, is not zero. In order that R be positive, the magnitude of the quotient, (Q), must be equal to the magnitude of the "true quotient" plus one. The true quotient is the quotient as derived from the expression

$$|(A)| = (Q) \cdot (u) + R.$$

To obtain the true rounded quotient after a division, for all cases of the Divide instruction, the following expression may be used:

$$(A) + 1/2\,|(u)| = (Q) \cdot (u) + R,$$

and, assuming d1 contains the dividend,

```
C1    TM   u   A     form |(u)| in A
C2    LA   A   107   form 1/2 |(u)| in A
C3    RA   A   d1    form (d1) + 1/2 |(u)| in A
C4    DV   u   q     form rounded division
```

It should be noted that the instruction at C2 puts an unwanted bit into A_{71}, but by using the Replace Add instruction, which, as addressed, only uses 36-bits of (A) to perform the addition, the bit in the A_{71} position is cleared out.

Since the Divide instruction produces as its result an integer quotient and remainder, the loss of significant bits may be troublesome in "fixed" or "floating point" operations if a scaling operation is not performed before a division. To illustrate how to perform this scaling operation, assume that the division b/c is desired where

$$\frac{1}{2} \leq |b| \leq 1$$

and

$$\frac{1}{2} \leq |c| \leq 1$$

and assume that b and c are scaled up in the registers e1 and e2 as

$$(e1) = b \cdot 2^{35}$$

and

$$(e2) = c \cdot 2^{35}$$

PX 71071

or $b = b_m \cdot 2^{-35}$
$c = c_m \cdot 2^{-35}$

where b_m and c_m are the machine copies (integers) of the external true fractional numbers. The instructions would be coded as follows:

f1	LA	e1	20043	Scale b up 35 places and leave the result in A.
f2	DV	e2	Q	Divide by c and put the result in Q.

The Q-register now contains the quotient scaled up 35 places; i.e.,

$$2^{35} > |q_m| \geq 2^{34} \text{ where } q_m = (Q)$$

or $1 > |q| \geq 1/2.$

In the above example it was tacitly assumed that the magnitude of b was greater than the magnitude of c. If this were not the case, the result of executing (f2) would cause a divide overflow since,

$$2^{36} > |q_m| \geq 2^{35}$$

(6) REPEAT INSTRUCTION. - Because of the complexity of the RP instruction, the details of the repeat operation are restated below.

(a) FORMAT OF RP INSTRUCTION. - The RP instruction has the form RPjnw. The normal values of j are 0 through 3 and determine the advance of the execution addresses of the repeated instruction as follows:

if j=0, neither the u nor v address portion of the repeated instruction is advanced.

if j=1, the v address of the repeated instruction is advanced by one after each execution.

if j=2, the u address of the repeated instruction is advanced by one after each execution.

if j=3, both the u and v addresses of the repeated instruction are advanced by one after each execution.

The advance of the addresses occurs only in the Program Control Register (PCR) where the repeated instruction is held during the repeat operations; therefore, the original form of the repeated instruction in storage is unaltered.

The value n determines the number of times the repeated instruction is to be executed. This value can vary through the range 0 through 4095 (decimal). If n is zero, the repeat operations are terminated without execution of the "repeated" instruction.

PX 71871

The "repeat termination address", w, replaces the v-address portion of fixed address F_1 (00000 or 40001) during the execution of the RP instruction. (Normally, F_1 contains an MJjv instruction with a j of 0.) This feature provides the means of "jumping out" of the repeated operations after n executions of the repeated instruction.

(b) REPEATED NON-JUMP INSTRUCTION. - The repeated instruction is held in the Program Control Registers (PCR) and is executed n times. After the n^{th} execution, a jump to F_1 occurs. Because this instruction is usually an MJjv with a j of zero and a v of w, it produces a jump to the address originally specified by w of the RP, instruction.

(c) REPEATED JUMP INSTRUCTION. - If the repeated instruction is a Threshold Jump (TJ) or Equality Jump (EJ), the jump to w may not occur. The repeated instruction is held in PCR and is executed not more than n times. If n executions occur with no jump condition being fulfilled, the normal termination, consisting of a jump to F_1, followed by a jump to w, occurs. However, if any execution of the repeated instruction fulfills a jump condition, the value j, n-r is stored in the v portion of Q (where j = j of RP instruction, and n-r = n of RP instruction minus r, the number of executions performed), and a jump is made to the address specified by v of the TJ or EJ instruction.

Similar circumstances are created by the Index Jump and Manually Selective Jump instructions with the exception that no indication of the number of executions is given by a storage in Q.

Other instructions (Interpret, Return Jump, Q-Jump, Sign Jump, Zero Jump, Manually Selective Stop, and Final Stop) produce a jump or stop on the first execution, and thus behave as if no RP instruction preceeded them.

(d) EXAMPLES USING THE REPEAT INSTRUCTION. - In many problems which are programmed for computer execution, it is important that the most efficient use of operating time and storage capacity be made. The Repeat Instruction, an unusual logical feature of the 1103, makes it possible to realize a sizeable reduction in the time spent referring to storage for instructions and in the storage space devoted to holding instructions. By using the RP instruction, long sequences of operations can be governed by only two or three instructional references to storage. This is evidenced by the following examples:

The instruction Transmit Positive (TPuv) causes the word at address u to be transmitted to address v. The simple two-instruction routine

C1	RP	3,n	C3	}	Transfer n words $(u_1) \rightarrow v_1$,
C2	TP	u1	v1	}	$(u_2) \rightarrow v_2$, ..., $(u_n) \rightarrow v_n$
C3	-	-	-		next instruction

effects a "block transfer" of n words from registers u1 through un to registers v1 through vn. If the transmissions are from registers u_i in Rapid Access Storage to registers v_i in Rapid Access Storage, the transfer rate is about 30,000 words per second.

PX 71871

Consider now the accumulation of products, such as,

$$S = a_1b_1 + a_2b_2 + \ldots + a_nb_n = \sum_{i=1}^{n} a_ib_i$$

The Multiply Add instruction (MAuv) causes the product of the numbers located in storage with the addresses u and v to be added to the number already in the Accumulator, leaving the total result in the 72-bit Accumulator. If the MA instruction is repeated as shown below, the scalar product of two n-vectors is generated.

d1	RS	A	A	Clear Accumulator to zero
d2	RP	3,n	d4	Form sum of products
d3	MA	u_1	v_1	$S = (u1)(v1) + (u2)(v2) + \ldots + (un)(vn)$
d4	--	--	--	Next instruction

u1	a_1	Vector a
.	.	
.	.	
.	.	
un	a_n	

v1	b_1	Vector b
.	.	
.	.	
.	.	
vn	b_n	

Note the instruction at d1 which clears the Accumulator to insure that its initial content is zero.

Care should be taken that the contents of F1 are not inadvertently altered during a block transfer of n words to Rapid Access Storage. This pitfall is illustrated by the following example:

RP	30400	w
TP	50000	01600

The sequence of events which occurs during the execution of these instructions is as follows:

1 Replace the lower order 15 bits of (F_1) with the address w.

2 Proceed to transmit the contents of

PX 71871

50000 to 01600
50001 to 01601
. .
. .
. .
50177 to 01777
50200 to 00000
. .
. .
. .
50377 to 00177

3 Extract the next instruction to be executed from address F_1.

Note that the contents of the address 00000, (F_1), have been replaced by the contents of address 50200 during the repeated transmissive operations. Because of this, the desired jump to w no longer exists at this address.

The following example illustrates further the care that must be exercised when using the Repeat instruction:

00100	RP	30200	00303
00101	TP	45000	00200

The operations performed by this pair of instructions are

1 Replace the lower order 15 bits of (F_1) with the address 00303

2 Proceed to transmit the contents of

45000 to 00200
45001 to 00201
. .
. .
. .
45102 to 00302
45103 to 00303
. .
. .
. .
45177 to 00377

3 Extract the next instruction to be executed from address F_1. Note that in this example the original contents of 00303 have been replaced by the contents of address 45103. Therefore the next instruction to be executed after the jump to 00303 is the instruction which was originally stored at 45103.

(e) The initial operations of the RP instruction replace the lowest order 15 bits of (F_1), the contents of the address 00000 (or 40001), with the address w of the RPjnw instruction. The terminating operations of the repeat sequence extract the next instruction to be executed from the address F_1

PX 71871

(unless the repeated instruction called for a jump to v). The instruction contained in F_1 is normally a jump instruction (usually an MJjv) referencing its v-address, the address w written in by the RP. However, if the instruction contained in F_1 is not a jump instruction, or if it is a conditional jump (IJ, MJ, TJ or EJ) where the jump condition is not fulfilled, the next instruction to be executed after the execution of the instruction at F_1 is taken from one of the following addresses:

If j was 0, NI from address 40000
If j was 1, NI from address 70000
If j was 2, NI from address 60000
If j was 3, NI from address 50000

These are the addresses that are left in the Program Address Counter (PAK) after the RP instruction, because of the use of PAK as the counter for the number of executions to be performed on the repeated instruction.

(7) PRINT AND PUNCH INSTRUCTION, PR-v AND PUjv. - Because of the operands of the Print and Punch instructions requiring only a six bit storage space, these instructions can be coded so that a savings of storage space is effected. The v-address of PUjv or PR-v may reference any instruction whose v-address references the Accumulator or Q-Register, or whose v-address portion is not used in the instruction, e.g., AMjn-, BMjn-, or FS--. For example, note the following instructions:

```
01001    PR    00000    01050    Print the number "1"
  .                .
  .                .
  .                .
01050    MP    00010    10052
```

Note that only one character is printed or punched with each output reference. In order to print or punch out in octal a 36 bit word, twelve Print or Punch instructions would be needed.

To illustrate the use of these instructions, consider the following routine which prints out on the typewriter one word in octal.

```
C1    PR    0     e10    Print carriage return
C2    LQ    d1    3      Shift word to be printed 3 places
                           to the left
C3    QT    d2    A      Mask out 3 bits or one octal digit.
C4    AT    d3    C5     Compute print instruction which is NI.
C5    PR    0     ei     Print one octal digit.
C6    IJ    d4    C2     Test for end, if not finished go to C2
C7    FS    --    --     End.
```

PX 7187

d1	xx	xxxxx	xxxxx	Number to be printed
d2	00	00000	00007	Mask
d3	PR	0	e0	Dummy command
d4	00	00000	00013	Counter, n-1 = 11_{10}
e0	00	00000	00037	(0)
e1	00	00000	00052	(1)
e2	00	00000	00074	(2)
e3	00	00000	00070	(3) Flexowriter codes for the digits 0 through 7.
e4	00	00000	00064	(4)
e5	00	00000	00062	(5)
e6	00	00000	00066	(6)
e7	00	00000	00072	(7)
e10	00	00000	00045	Flexowriter code for carriage return.

PX 71871

APPENDIX 6A

NUMBER SYSTEMS

1. GENERAL.

Any positive integer N can be expressed in the form

$$N = A_n r^n + A_{n-1} r^{n-1} + \ldots + A_1 r^1 + A_0 r^0$$

where $0 \leq A_i < r$ and r is any integer greater than 1. The integer r is called the base or radix of the number system. The only radix values that need to be considered for programming for the 1103 are 10, 8 and 2. The number systems with these radices are called decimal, octal, and binary systems, respectively.

An integer N expressed as a number in the usual decimal notation implies that the number is a refinement of a polynomial in powers of 10 with coefficients which satisfy the relation $0 \leq A_i < 10$. For example, N = 308 implies the polynomial $N = 3 \cdot 10^2 + 0 \cdot 10^1 + 8 \cdot 10^0$. Accordingly, an integer N = 1011 in the binary system with $0 \leq A_i < 2$ may be expressed as the polynomial $N = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ (= 11 decimal); and an integer N = 126 in the octal system with $0 \leq A_i < 8$ may be expressed as the polynomial $N = 1 \cdot 8^2 + 2 \cdot 8^1 + 6 \cdot 8^0$ (= 86 decimal).

Because electronic and magnetic circuits consist primarily of bi-stable elements, the ERA 1103 uses the binary representation in all internal manipulations of numbers. As will be seen later, there is a very simple relationship between the binary and octal representations of a number. Since conversion between these two systems is almost immediate and since the octal notation is shorter, programs are encoded with octal notation representing binary numbers. It is desirable for people who prepare programs for the ERA computers to be

familiar with methods of converting numbers from decimal to binary to octal form and vice versa. When the base of a number is not apparent from the context, it will be specified by a decimal subscript as shown in the following example:

$$34_{10} = 42_{8} = 100010_{2}$$

2. CHANGE OF BASE.

Since the manual conversion of numbers from one system to another involves the arithmetic operations of addition and multiplication, it is necessary to know the sums and products of certain pairs of integers expressed in the octal and binary systems. In doing arithmetic operations by hand, the ability to remember the sums and products of pairs of decimal digits is utilized, although these sums and products are tabulated in decimal addition and multiplication tables. Similar tables presented below are necessary for arithmetic operations with octal and binary digits.

OCTAL ADDITION TABLE

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

OCTAL MULTIPLICATION TABLE

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

The addition table and the multiplication table for binary numbers are shown below.

BINARY ADDITION TABLE

	0	1
0	0	1
1	1	10

BINARY MULTIPLICATION TABLE

	0	1
0	0	0
1	0	1

PX 71871

a. CONVERSION OF INTEGERS. - The simplest way of changing a binary or octal integer to its decimal equivalent is to expand it to, and evaluate the sum of the terms of, the polynomial expression given on a previous page. The conversion from decimal to octal or binary presents more of a problem. Two methods are given which can be used for these conversions. The first of these is as follows:

(1) Given two integers N and D, the latter being positive, it can be shown that there exist unique integers Q and R such that $N = QD + R$ where $0 \leq R < D$. This is, of course, true regardless of the choice of base used in expressing N. Hence, if N_{10} and N_8 are the decimal and octal expressions for the same integer and if D_{10} and D_8 are the decimal and octal expressions for another (positive) integer, we can compare the two equations

$$N_{10} = Q_{10} \cdot D_{10} + R_{10}, \quad 0 \leq R_{10} < D_{10}$$
$$N_8 = Q_8 \cdot D_8 + R_8, \quad 0 \leq R_8 < D_8$$

and conclude that $Q_{10} = Q_8$ and $R_{10} = R_8$. This fact is used in the process of converting an integer from base 10 to base 8 in the following way: The decimal expressions for an integer N expressed as N_{10} and N_8 are

$$N_{10} = d_n 10^n + d_{n-1} 10^{n-1} + \ldots + d_1 10 + d_0, \quad 0 \leq d_i < 10$$
$$N_8 = e_m 8^m + e_{m-1} 8^{m-1} + \ldots + e_1 8 + e_0, \quad 0 \leq e_i < 8$$

Dividing the polynomials by decimal 8 gives the same remainder in either case, but from the second expression, it is obvious that this remainder is e_0 while the quotient is $e_m 8^{m-1} + e_{m-1} 8^{m-2} \ldots + e_2 8 + e_1$. Dividing this quotient by 8 gives the new quotient $e_m 8^{m-2} + \ldots e_3 8 + e_2$ and the new remainder e_1. Continuing division results in the successive remainders e_0, e_1, e_2, ..., e_m which

PX 71871

are the digits of N_8 in reverse order. In a numerical case, the successive divisions of N_{10} by decimal 8 yield remainders which are actually the decimal equivalents of the digits $e_0, e_1, \ldots, e_m$.

To illustrate this method, the number expressed as 1492 in decimal notation is changed to its octal equivalent as follows:

Decimal Expression		Octal Equivalent
$8\overline{)1492}$,	Remainder 4	4
$8\overline{)186}$,	Remainder 2	2
$8\overline{)23}$,	Remainder 7	7
$8\overline{)2}$,	Remainder 2	2
0		

Thus $1492_{10} = 2724_8$

Conversion of an integer N from base 8 to base 10 can be carried out in a similar fashion by dividing N_8 by 12_8 ($= 10_{10}$) to yield the octal equivalents of the digits of N_{10}.

The conversion of an integer N from base 10 to base 2 or from base 2 to base 10 can be discussed according to the method outlined above with N_{10} and N_2 represented as follows:

$$N_{10} = d_n 10^n + \ldots + d_1 10 + d_0 \ , \ 0 \leq d_i < 10$$

$$N_2 = b_k 2^k + \ldots + b_1 2 + b_0 \ , \ 0 \leq b_i < 2$$

N_{10} would be divided by 2 to yield the decimal equivalents of the digits of N_2, and N_2 would be divided by 1010 ($= 10_{10}$) to yield the binary equivalents of the digits of N_{10}.

(2) The method that follows has the advantage that it yields the desired digits in normal order, but it has the disadvantage that it requires a knowledge of the values of powers of one base in terms of another. This method requires the procedure below:

Given a number N_r to express as $N_b = a_n b^n + a_{n-1} b^{n-1} + \ldots + a_1 b^1 + a_0 b^0$, the coefficients $a_n \ldots a_0$, $0 \leq a_i < b$, may be found as follows:

1. Determine the highest power n such that $b^{n+1} > N_r$. Then, let $N_r = N_n$. Divide N_n by b^n, yielding the quotient a_n and the remainder $N_{n-1} = a_{n-1} b^{n-1} + \ldots + a_0 b^0$.

2. Repeat the divisions of N_i ($i = n, n-1, \ldots, 0$) by b^i until the last division yields the quotient a_0.

3. A quotient (or coefficient) a_i may be zero and must be represented in its proper order in the final result.

As an example of this method the number expressed decimally as 137 is changed to its octal equivalent as follows:

$$N_n = a_n b^n + a_{n-1} b^{n-1} + \ldots + a_0 b^0 = N_b$$

$r = 10$, $b = 8$, $N_r = N_n = 137$, $n = 2$ $(8^3 > 137)$

$137 = 2 \cdot 64 + 9$ $\qquad a_2 = 2$, $N_1 = 9$

$9 = 1 \cdot 8 + 1$ $\qquad a_1 = 1$, $N_0 = 1$

$1 = 1 \cdot 1 + 0$ $\qquad a_0 = 1$

The coefficients derived above are the digits of N_8; thus $N_{10} = 137 = N_8 = 211$.

PX 71871

(3) Although it would be possible to use the above methods in converting an integer from base 8 to base 2 and vice versa, it is easier to convert by inspection. This can be done because of simple relationship, discussed subsequently, between N_8 and N_2, dependent upon the fact that $2^3 = 8$.

The equivalent polynomials for an integer N expressed as N_8 and N_2 are

$$N_8 = e_m 8^m + e_{m-1} 8^{m-1} + \ldots + e_2 8^2 + e_1 8^1 + e_0 8^0, \quad 0 \le e_i < 8$$

$$N_2 = b_k 2^k + b_{k-1} 2^{k-1} + \ldots + b_2 2^2 + b_1 2^1 + b_0 2^0, \quad 0 \le b_i < 2$$

The expression for N_2 may be factored as follows:

$$N_2 = \ldots + (b_{3m+2} 2^2 + b_{3m+1} 2^1 + b_{3m}) 2^{3m} + \ldots + (b_2 2^2 + b_1 2^1 + b_0) 2^0$$

with the integral range $m = 0, 1, 2, \ldots, p/3$, $k \ge p \ge k-2$

The value of p which is a multiple of three is chosen. It is seen by comparing termwise the expressions for N_2 factored and N_8 that a common component exists, $(2^3)^m = 8^i$. Hence, the coefficients of each term are equal since the polynomial expressions themselves are equal. Thus the digits of N_8, given its binary equivalent, are as follows:

$$e_0 = (b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0)$$

$$e_1 = (b_5 \cdot 2^2 + b_4 \cdot 2^1 + b_3)$$

$$e_2 = (b_8 \cdot 2^2 + b_7 \cdot 2^1 + b_6)$$

$$\vdots \qquad \vdots$$

$$e_m = (b_{p+2} 2^2 + b_{p+1} 2^1 + b_p)$$

Note that the condition $0 \le e_i < 8$ is satisfied by the above relationship.

PX 71871

The simple procedure of the conversion from N_2 to N_8, according to the preceding method, is shown by the following example:

Given $N_2 = 1101001111$, the digits of its equivalent in octal notation, N_8, are derived by evaluating each group of three binary digits, beginning at the right. Thus, N_2 above yields $N_8 = 1517$. (The binary number may be expanded by assuming zeros at the left.)

To convert a number from octal to binary notation, each octal digit is replaced by its binary equivalent.

b. CONVERSION OF FRACTIONS. - Fractions in the binary and octal number systems are represented in a fashion similar to fractions in the decimal system. For example, the number N, represented by the mixed decimal number 5.78125, is equal to the fraction $\frac{578125}{10^5}$. The number N in binary notation, represented by the mixed number 101.11001, is equal to the binary fraction $\frac{10111001}{100000}$. To represent a fraction, base r, whose denominator is equal to ar^p with $0 \leq a < r$, as a number with a radical point, place the radical point p places to the left of the right-most digit of the numerator.

Numbers represented by the fractions of one system may be translated to their representations in other systems by applying the methods described previously to both the integral numerator and denominator. For example, the fraction in the binary notation above may be converted to decimal notation as follows:

$$\frac{2^7 + 2^5 + 2^4 + 2^3 + 2^0}{2^5} = \frac{185}{32} = 5.78125$$

PX 71871

A fraction in the binary system may be changed to its octal notation by the grouping method described previously. Applying this method to the same fraction, base 2, yields the fraction, base 8, $\frac{271}{40} = \frac{27.1}{4} = 5.62$. This octal number may be expressed as the fraction $\frac{562}{100}$ which is equal to the decimal fraction $\frac{5 \cdot 8^2 + 6 \cdot 8 + 2}{8^2} = 5.78125$.

Fractions, expressed with a radical point, of one number system may be translated to numbers of another system by the method described subsequently. If the fraction is a mixed number, the integral portion is converted by one of the methods for integral conversion previously explained. The fractional conversion may be explained by establishing the polynomial expressions for a number F expressed as a fraction in two systems. The expansions of F_{10} and F_8 will be used for exemplary purposes. The expressions for a fraction F are as follows:

$$F_{10} = d_{-1}10^{-1} + d_{-2}10^{-2} + \ldots + d_{1-n}10^{1-n} + d_{-n}10^{-n} \quad , \quad 0 \leq d_i < 10$$

$$F_8 = e_{-1}8^{-1} + e_{-2}8^{-2} + \ldots + e_{1-m}8^{1-m} + e_{-m}8^{-m} \quad , \quad 0 \leq e_i < 8$$

Multiplication by 8 of the polynomials above, yields the expressions

$$F_8 \times 8 = e_{-1}8^0 + e_{-2}8^{-1} + \ldots + e_{-m}8^{1-m}$$

and $$F_{10} \times 8 = d_{-1}10^{-1}.8 + d_{-2}10^{-2}.8 + \ldots + d_{-n}10^{-n}\cdot 8$$

The multiplication of $F_8 \times 8$ results in a mixed number with $e_{-1}8^0$ as the integral portion; therefore, the number to the left of the decimal point of $F_{10} \times 8$ must be the decimal equivalent of e_{-1}. This number may be a zero or an integer depending on the value of the first decimal digit d_{-1} of F_{10}. To find the decimal digit equivalent to the second octal digit e_{-2}, multiply

by 8 the fractional portion of $F_{10} \times 8$. This multiplication will result in a number whose portion to the left of the decimal point is equivalent to the integral portion, $e_{-2}8^0$, of 8 times the fractional portion of $F_8 \times 8$. Thus, successive multiplications by 8 of the fractional portions of F_{10} yield the decimal equivalents of the octal digits e_{-1}, e_{-2}, ..., e_{-m}. As an example, to convert the fraction 5.78125, base 10, to its octal equivalent the procedure would be as follows:

	Decimal expression	Octal equivalent
	5.78125	5
.78125 x 8 =	6.25	6
.25 x 8 =	2.00	2

Thus $5.78125_{10} = 5.62_8$.

Similar conversions are made with other number systems by using the appropriate number as the multiplier. Note that a terminating fraction of one base need not lead to a terminating fraction of another base. For example, $0.10_{10} = .0631463146314 \ldots_8$.

PX 71871

3. REPRESENTATION OF SIGNED NUMBERS.

The modulus of a number system is the discrete number of quantities which can be represented by the system. The modulus of a number system to be represented by the elements of a computer is fixed by the design of the machine. This discussion will consider only the representation of a binary system by k parallel stages of bi-stable elements. Each stage represents a digit of N_2: the state of each stage (indicating a 0 or 1) represents a coefficient of a term of the polynomial

$$N_2 = b_{k-1}2^{k-1} + \ldots + b_1 2^1 + b_0 2^0 \ , \ 0 \leq b_i < 2$$

For exemplary purposes the value k = 4 stages will be used. Thus if a computer had four bi-stable elements the modulus of the binary system which could be represented by the machine would be $2^4 = 16$, with the range of binary numbers from 0000 to 1111. It is desirable that this range of binary numbers from 0 to 2^k-1 represent both positive and negative values since it is not possible machine-wise to indicate negative numbers by a minus sign designation. Therefore, the binary digit represented by the stage 2^{k-1} is reserved to indicate the sign of a number, and negative numbers are represented machine-wise by a complement form. The simplest way to represent a negative number in such a manner in a computer with bi-stable elements is to use the one's complement of its absolute value. This entails subtracting the binary representation of the absolute value of a negative number of a system of signed numbers, modulus 2^k, from the binary representation of 2^k-1. Thus, the process of forming the one's complement representation of the negative number -3, with k = 4, would be as follows:

```
      1111
minus 0011  binary representation of |-3|
      1100  machine representation of -3
```

PX 71871

Note that forming the one's complement representation of a negative number can be accomplished machine-wise, if the absolute value is represented by k stages, by merely reversing the state of a bi-stable element: each representation of one is replaced by a zero representation and vice versa.

In a one's complement binary system the range of values of signed numbers that can be represented by k stages is $1-2^{k-1}$ to $2^{k-1}-1$ with the left-most stage indicating the sign of the number represented. The table below shows the correspondence of signed numbers to their representation in one's complement system with k = 4.

Signed Decimal Number	Signed Binary Number	One's Complement Binary Number
7	+111	0111
6	+110	0110
5	+101	0101
4	+100	0100
3	+011	0011
2	+010	0010
1	+001	0001
0	+000	0000
-0	-000	1111
-1	-001	1110
-2	-010	1101
-3	-011	1100
-4	-100	1011
-5	-101	1010
-6	-110	1001
-7	-111	1000

PX 71871

Note that the moduli of the preceding systems of signed numbers, as shown represented by a one's complement binary system, may be reduced from 2^4 to 2^4-1 because of the representation of two zeros. In actual arithmetic operations by a computer the occurrence of only one of these representations of zero, 00...00 or 11...11, is possible. This reduces the modulus of a one's complement system represented by k stages to 2^k-1. The fundamental arithmetic operation of the computer determines which of these two states will represent zero. If the basic arithmetic operation is subtraction, as it is in the ERA 1103, a result of zero derived from any internal arithmetic operation will be represented by the simultaneous zero state of each stage. (Addition is performed by a subtractive process by subtracting the one's complement of the addend from the augend to form the difference, which will, in this case, be the sum.)

Thus the representation of a zero occurring when a number is subtracted from itself or when the absolute value of a negative number is added (by a subtractive process) to its negative value will be a series of zeros if these operations are performed by a subtractive process. The following examples illustrate this process with a four-stage one's complement binary number representation.

	0001		1		1110		-1
minus	0001	minus	1	minus	1110	plus	|-1|
	0000		0		0000		0

When the subtraction of two numbers of a binary system, modulus 2^k-1, necessitates a borrow from an implied 2^k stage such that the minuend exceeds the modulus of the system, the resulting (partial) remainder will be numerically greater than the true modular difference by the amount by which the borrow alone increased the value of the minuend beyond the value of the modulus. Since the borrow in itself always exceeds the modulus by a value of one, this value must be subtracted from the partial remainder to yield the modular difference.

Machinewise, the borrow propagated from the left-most stage is actually applied to the right-most stage and is known as an end-around or circular borrow. Using the notation of the one's complement system, modulus 2^4-1, the subtraction of a value of four from a value of two proceeds as follows:

```
      (1)  0010          2
minus      0100    minus 4
           1110
minus       1
           1101         -2
```

The subtraction was enabled by applying the borrow, propagated from the 2^3 stage, to the 2^0 stage; or, by increasing the minuend to exceed the value of the modulus, the excessive value of one derived by the borrow must be subtracted from the partial remainder to yield the modular difference.

Another example is provided by the simple process of forming the sum of zero and one by a subtractive process. Thus, using the same notation

```
      (1)  0000          0
minus      1110    plus+ 1
           0010
minus       1
           0001         +1
```

In this case the end-around borrow is continued past the right-most stage until it can be made. It may continue only as far as the stage where the conditions first necessitated the borrow.

PX 71871

APPENDIX 6B

TABLE OF POWERS OF TWO

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

PX 71871

APPENDIX 6C

DECIMAL TO OCTAL CONVERSION TABLE
(0-4096)

0 Through 199

		UNITS									
		0	1	2	3	4	5	6	7	8	9
00	0	0	1	2	3	4	5	6	7	10	11
	1	12	13	14	15	16	17	20	21	22	23
	2	24	25	26	27	30	31	32	33	34	35
	3	36	37	40	41	42	43	44	45	46	47
	4	50	51	52	53	54	55	56	57	60	61
TENS	5	62	63	64	65	66	67	70	71	72	73
	6	74	75	76	77	100	101	102	103	104	105
	7	106	107	110	111	112	113	114	115	116	117
	8	120	121	122	123	124	125	126	127	130	131
	9	132	133	134	135	136	137	140	141	142	143
01	0	144	145	146	147	150	151	152	153	154	155
	1	156	157	160	161	162	163	164	165	166	167
	2	170	171	172	173	174	175	176	177	200	201
	3	202	203	204	205	206	207	210	211	212	213
	4	214	215	216	217	220	221	222	223	224	225
TENS	5	226	227	230	231	232	233	234	235	236	237
	6	240	241	242	243	244	245	246	247	250	251
	7	252	253	254	255	256	257	260	261	262	263
	8	264	265	266	267	270	271	272	273	274	275
	9	276	277	300	301	302	303	304	305	306	307

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE(Con't)
(0-4096)

200 Through 399

		UNITS									
		0	1	2	3	4	5	6	7	8	9
02	0	310	311	312	313	314	315	316	317	320	321
	1	322	323	324	325	326	327	330	331	332	333
	2	334	335	336	337	340	341	342	343	344	345
	3	346	347	350	351	352	353	354	355	356	357
	4	360	361	362	363	364	365	366	367	370	371
TENS	5	372	373	374	375	376	377	400	401	402	403
	6	404	405	406	407	410	411	412	413	414	415
	7	416	417	420	421	422	423	424	425	426	427
	8	430	431	432	433	434	435	436	437	440	441
	9	442	443	444	445	446	447	450	451	452	453
03	0	454	455	456	457	460	461	462	463	464	465
	1	466	467	470	471	472	473	474	475	476	477
	2	500	501	502	503	504	505	506	507	510	511
	3	512	513	514	515	516	517	520	521	522	523
	4	524	525	526	527	530	531	532	533	534	535
TENS	5	536	537	540	541	542	543	544	545	546	547
	6	550	551	552	553	554	555	556	557	560	561
	7	562	563	564	565	566	567	570	571	572	573
	8	574	575	576	577	600	601	602	603	604	605
	9	606	607	610	611	612	613	614	615	616	617

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

400 Through 599

		UNITS									
		0	1	2	3	4	5	6	7	8	9
04	0	620	621	622	623	624	625	626	627	630	631
	1	632	633	634	635	636	637	640	641	642	643
	2	644	645	646	647	650	651	652	653	654	655
	3	656	657	660	661	662	663	664	665	666	667
	4	670	671	672	673	674	675	676	677	700	701
TENS	5	702	703	704	705	706	707	710	711	712	713
	6	714	715	716	717	720	721	722	723	724	725
	7	726	727	730	731	732	733	734	735	736	737
	8	740	741	742	743	744	745	746	747	750	751
	9	752	753	754	755	756	757	760	761	762	763
05	0	764	765	766	767	770	771	772	773	774	775
	1	776	777	1000	1001	1002	1003	1004	1005	1006	1007
	2	1010	1011	1012	1013	1014	1015	1016	1017	1020	1021
	3	1022	1023	1024	1025	1026	1027	1030	1031	1032	1033
	4	1034	1035	1036	1037	1040	1041	1042	1043	1044	1045
TENS	5	1046	1047	1050	1051	1052	1053	1054	1055	1056	1057
	6	1060	1061	1062	1063	1064	1065	1066	1067	1070	1071
	7	1072	1073	1074	1075	1076	1077	1100	1101	1102	1103
	8	1104	1105	1106	1107	1110	1111	1112	1113	1114	1115
	9	1116	1117	1120	1121	1122	1123	1124	1125	1126	1127

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

600 Through 799

		UNITS									
		0	1	2	3	4	5	6	7	8	9
06	0	1130	1131	1132	1133	1134	1135	1136	1137	1140	1141
	1	1142	1143	1144	1145	1146	1147	1150	1151	1152	1153
	2	1154	1155	1156	1157	1160	1161	1162	1163	1164	1165
	3	1166	1167	1170	1171	1172	1173	1174	1175	1176	1177
	4	1200	1201	1202	1203	1204	1205	1206	1207	1210	1211
TENS	5	1212	1213	1214	1215	1216	1217	1220	1221	1222	1223
	6	1224	1225	1226	1227	1230	1231	1232	1233	1234	1235
	7	1236	1237	1240	1241	1242	1243	1244	1245	1246	1247
	8	1250	1251	1252	1253	1254	1255	1256	1257	1260	1261
	9	1262	1263	1264	1265	1266	1267	1270	1271	1272	1273
07	0	1274	1275	1276	1277	1300	1301	1302	1303	1304	1305
	1	1306	1307	1310	1311	1312	1313	1314	1315	1316	1317
	2	1320	1321	1322	1323	1324	1325	1326	1327	1330	1331
	3	1332	1333	1334	1335	1336	1337	1340	1341	1342	1343
	4	1344	1345	1346	1347	1350	1351	1352	1353	1354	1355
TENS	5	1356	1357	1360	1361	1362	1363	1364	1365	1366	1367
	6	1370	1371	1372	1373	1374	1375	1376	1377	1400	1401
	7	1402	1403	1404	1405	1406	1407	1410	1411	1412	1413
	8	1414	1415	1416	1417	1420	1421	1422	1423	1424	1425
	9	1426	1427	1430	1431	1432	1433	1434	1435	1436	1437

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

800 Through 999

		UNITS									
		0	1	2	3	4	5	6	7	8	9
08	0	1440	1441	1442	1443	1444	1445	1446	1447	1450	1451
	1	1452	1453	1454	1455	1456	1457	1460	1461	1462	1463
	2	1464	1465	1466	1467	1470	1471	1472	1473	1474	1475
	3	1476	1477	1500	1501	1502	1503	1504	1505	1506	1507
	4	1510	1511	1512	1513	1514	1515	1516	1517	1520	1521
TENS	5	1522	1523	1524	1525	1526	1527	1530	1531	1532	1533
	6	1534	1535	1536	1537	1540	1541	1542	1543	1544	1545
	7	1546	1547	1550	1551	1552	1553	1554	1555	1556	1557
	8	1560	1561	1562	1563	1564	1565	1566	1567	1570	1571
	9	1572	1573	1574	1575	1576	1577	1600	1601	1602	1603
09	0	1604	1605	1606	1607	1610	1611	1612	1613	1614	1615
	1	1616	1617	1620	1621	1622	1623	1624	1625	1626	1627
	2	1630	1631	1632	1633	1634	1635	1636	1637	1640	1641
	3	1642	1643	1644	1645	1646	1647	1650	1651	1652	1653
	4	1654	1655	1656	1657	1660	1661	1662	1663	1664	1665
TENS	5	1666	1667	1670	1671	1672	1673	1674	1675	1676	1677
	6	1700	1701	1702	1703	1704	1705	1706	1707	1710	1711
	7	1712	1713	1714	1715	1716	1717	1720	1721	1722	1723
	8	1724	1725	1726	1727	1730	1731	1732	1733	1734	1735
	9	1736	1737	1740	1741	1742	1743	1744	1745	1746	1747

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

1000 Through 1199

		UNITS									
		0	1	2	3	4	5	6	7	8	9
10	0	1750	1751	1752	1753	1754	1755	1756	1757	1760	1761
	1	1762	1763	1764	1765	1766	1767	1770	1771	1772	1773
	2	1774	1775	1776	1777	2000	2001	2002	2003	2004	2005
	3	2006	2007	2010	2011	2012	2013	2014	2015	2016	2017
	4	2020	2021	2022	2023	2024	2025	2026	2027	2030	2031
TENS	5	2032	2033	2034	2035	2036	2037	2040	2041	2042	2043
	6	2044	2045	2046	2047	2050	2051	2052	2053	2054	2055
	7	2056	2057	2060	2061	2062	2063	2064	2065	2066	2067
	8	2070	2071	2072	2073	2074	2075	2076	2077	2100	2101
	9	2102	2103	2104	2105	2106	2107	2110	2111	2112	2113
11	0	2114	2115	2116	2117	2120	2121	2122	2123	2124	2125
	1	2126	2127	2130	2131	2132	2133	2134	2135	2136	2137
	2	2140	2141	2142	2143	2144	2145	2146	2147	2150	2151
	3	2152	2153	2154	2155	2156	2157	2160	2161	2162	2163
	4	2164	2165	2166	2167	2170	2171	2172	2173	2174	2175
TENS	5	2176	2177	2200	2201	2202	2203	2204	2205	2206	2207
	6	2210	2211	2212	2213	2214	2215	2216	2217	2220	2221
	7	2222	2223	2224	2225	2226	2227	2230	2231	2232	2233
	8	2234	2235	2236	2237	2240	2241	2242	2243	2244	2245
	9	2246	2247	2250	2251	2252	2253	2254	2255	2256	2257

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

1200 Through 1399

		UNITS									
		0	1	2	3	4	5	6	7	8	9
12	0	2260	2261	2262	2263	2264	2265	2266	2267	2270	2271
	1	2272	2273	2274	2275	2276	2277	2300	2301	2302	2303
	2	2304	2305	2306	2307	2310	2311	2312	2313	2314	2315
	3	2316	2317	2320	2321	2322	2323	2324	2325	2326	2327
	4	2330	2331	2332	2333	2334	2335	2336	2337	2340	2341
TENS	5	2342	2343	2344	2345	2346	2347	2350	2351	2352	2353
	6	2354	2355	2356	2357	2360	2361	2362	2363	2364	2365
	7	2366	2367	2370	2371	2372	2373	2374	2375	2376	2377
	8	2400	2401	2402	2403	2404	2405	2406	2407	2410	2411
	9	2412	2413	2414	2415	2416	2417	2420	2421	2422	2423
13	0	2424	2425	2426	2427	2430	2431	2432	2433	2434	2435
	1	2436	2437	2440	2441	2442	2443	2444	2445	2446	2447
	2	2450	2451	2452	2453	2454	2455	2456	2457	2460	2461
	3	2462	2463	2464	2465	2466	2467	2470	2471	2472	2473
	4	2474	2475	2476	2477	2500	2501	2502	2503	2504	2505
TENS	5	2506	2507	2510	2511	2512	2513	2514	2515	2516	2517
	6	2520	2521	2522	2523	2524	2525	2526	2527	2530	2531
	7	2532	2533	2534	2535	2536	2537	2540	2541	2542	2543
	8	2544	2545	2546	2547	2550	2551	2552	2553	2554	2555
	9	2556	2557	2560	2561	2562	2563	2564	2565	2566	2567

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

1400 Through 1599

		UNITS									
		0	1	2	3	4	5	6	7	8	9
14	0	2570	2571	2572	2573	2574	2575	2576	2577	2600	2601
	1	2602	2603	2604	2605	2606	2607	2610	2611	2612	2613
	2	2614	2615	2616	2617	2620	2621	2622	2623	2624	2625
	3	2626	2627	2630	2631	2632	2633	2634	2635	2636	2637
	4	2640	2641	2642	2643	2644	2645	2646	2647	2650	2651
TENS	5	2652	2653	2654	2655	2656	2657	2660	2661	2662	2663
	6	2664	2665	2666	2667	2670	2671	2672	2673	2674	2675
	7	2676	2677	2700	2701	2702	2703	2704	2705	2706	2707
	8	2710	2711	2712	2713	2714	2715	2716	2717	2720	2721
	9	2722	2723	2724	2725	2726	2727	2730	2731	2732	2733
15	0	2734	2735	2736	2737	2740	2741	2742	2743	2744	2745
	1	2746	2747	2750	2751	2752	2753	2754	2755	2756	2757
	2	2760	2761	2762	2763	2764	2765	2766	2767	2770	2771
	3	2772	2773	2774	2775	2776	2777	3000	3001	3002	3003
	4	3004	3005	3006	3007	3010	3011	3012	3013	3014	3015
TENS	5	3016	3017	3020	3021	3022	3023	3024	3025	3026	3027
	6	3030	3031	3032	3033	3034	3035	3036	3037	3040	3041
	7	3042	3043	3044	3045	3046	3047	3050	3051	3052	3053
	8	3054	3055	3056	3057	3060	3061	3062	3063	3064	3065
	9	3066	3067	3070	3071	3072	3073	3074	3075	3076	3077

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

1600 Through 1799

		UNITS									
		0	1	2	3	4	5	6	7	8	9
16	0	3100	3101	3102	3103	3104	3105	3106	3107	3110	3111
	1	3112	3113	3114	3115	3116	3117	3120	3121	3122	3123
	2	3124	3125	3126	3127	3130	3131	3132	3133	3134	3135
	3	3136	3137	3140	3141	3142	3143	3144	3145	3146	3147
	4	3150	3151	3152	3153	3154	3155	3156	3157	3160	3161
TENS	5	3162	3163	3164	3165	3166	3167	3170	3171	3172	3173
	6	3174	3175	3176	3177	3200	3201	3202	3203	3204	3205
	7	3206	3207	3210	3211	3212	3213	3214	3215	3216	3217
	8	3220	3221	3222	3223	3224	3225	3226	3227	3230	3231
	9	3232	3233	3234	3235	3236	3237	3240	3241	3242	3243
17	0	3244	3245	3246	3247	3250	3251	3252	3253	3254	3255
	1	3256	3257	3260	3261	3262	3263	3264	3265	3266	3267
	2	3270	3271	3272	3273	3274	3275	3276	3277	3300	3301
	3	3302	3303	3304	3305	3306	3307	3310	3311	3312	3313
	4	3314	3315	3316	3317	3320	3321	3322	3323	3324	3325
TENS	5	3326	3327	3330	3331	3332	3333	3334	3335	3336	3337
	6	3340	3341	3342	3343	3344	3345	3346	3347	3350	3351
	7	3352	3353	3354	3355	3356	3357	3360	3361	3362	3363
	8	3364	3365	3366	3367	3370	3371	3372	3373	3374	3375
	9	3376	3377	3400	3401	3402	3403	3404	3405	3406	3407

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

1800 Through 1999

		UNITS									
		0	1	2	3	4	5	6	7	8	9
18	0	3410	3411	3412	3413	3414	3415	3416	3417	3420	3421
	1	3422	3423	3424	3425	3426	3427	3430	3431	3432	3433
	2	3434	3435	3436	3437	3440	3441	3442	3443	3444	3445
	3	3446	3447	3450	3451	3452	3453	3454	3455	3456	3457
	4	3460	3461	3462	3463	3464	3465	3466	3467	3470	3471
TENS	5	3472	3473	3474	3475	3476	3477	3500	3501	3502	3503
	6	3504	3505	3506	3507	3510	3511	3512	3513	3514	3515
	7	3516	3517	3520	3521	3522	3523	3524	3525	3526	3527
	8	3530	3531	3532	3533	3534	3535	3536	3537	3540	3541
	9	3542	3543	3544	3545	3546	3547	3550	3551	3552	3553
19	0	3554	3555	3556	3557	3560	3561	3562	3563	3564	3565
	1	3566	3567	3570	3571	3572	3573	3574	3575	3576	3577
	2	3600	3601	3602	3603	3604	3605	3606	3607	3610	3611
	3	3612	3613	3614	3615	3616	3617	3620	3621	3622	3623
	4	3624	3625	3626	3627	3630	3631	3632	3633	3634	3635
TENS	5	3636	3637	3640	3641	3642	3643	3644	3645	3646	3647
	6	3650	3651	3652	3653	3654	3655	3656	3657	3660	3661
	7	3662	3663	3664	3665	3666	3667	3670	3671	3672	3673
	8	3674	3675	3676	3677	3700	3701	3702	3703	3704	3705
	9	3706	3707	3710	3711	3712	3713	3714	3715	3716	3717

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

2000 Through 2199

		UNITS									
		0	1	2	3	4	5	6	7	8	9
20	0	3720	3721	3722	3723	3724	3725	3726	3727	3730	3731
	1	3732	3733	3734	3735	3736	3737	3740	3741	3742	3743
	2	3744	3745	3746	3747	3750	3751	3752	3753	3754	3755
	3	3756	3757	3760	3761	3762	3763	3764	3765	3766	3767
	4	3770	3771	3772	3773	3774	3775	3776	3777	4000	4001
TENS	5	4002	4003	4004	4005	4006	4007	4010	4011	4012	4013
	6	4014	4015	4016	4017	4020	4021	4022	4023	4024	4025
	7	4026	4027	4030	4031	4032	4033	4034	4035	4036	4037
	8	4040	4041	4042	4043	4044	4045	4046	4047	4050	4051
	9	4052	4053	4054	4055	4056	4057	4060	4061	4062	4063
21	0	4064	4065	4066	4067	4070	4071	4072	4073	4074	4075
	1	4076	4077	4100	4101	4102	4103	4104	4105	4106	4107
	2	4110	4111	4112	4113	4114	4115	4116	4117	4120	4121
	3	4122	4123	4124	4125	4126	4127	4130	4131	4132	4133
	4	4134	4135	4136	4137	4140	4141	4142	4143	4144	4145
TENS	5	4146	4147	4150	4151	4152	4153	4154	4155	4156	4157
	6	4160	4161	4162	4163	4164	4165	4166	4167	4170	4171
	7	4172	4173	4174	4175	4176	4177	4200	4201	4202	4203
	8	4204	4205	4206	4207	4210	4211	4212	4213	4214	4215
	9	4216	4217	4220	4221	4222	4223	4224	4225	4226	4227

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

2200 Through 2399

		UNITS									
		0	1	2	3	4	5	6	7	8	9
22	0	4230	4231	4232	4233	4234	4235	4236	4237	4240	4241
	1	4242	4243	4244	4245	4246	4247	4250	4251	4252	4253
	2	4254	4255	4256	4257	4260	4261	4262	4263	4264	4265
	3	4266	4267	4270	4271	4272	4273	4274	4275	4276	4277
	4	4300	4301	4302	4303	4304	4305	4306	4307	4310	4311
TENS	5	4312	4313	4314	4315	4316	4317	4320	4321	4322	4323
	6	4324	4325	4326	4327	4330	4331	4332	4333	4334	4335
	7	4336	4337	4340	4341	4342	4343	4344	4345	4346	4347
	8	4350	4351	4352	4353	4354	4355	4356	4357	4360	4361
	9	4362	4363	4364	4365	4366	4367	4370	4371	4372	4373
23	0	4374	4375	4376	4377	4400	4401	4402	4403	4404	4405
	1	4406	4407	4410	4411	4412	4413	4414	4415	4416	4417
	2	4420	4421	4422	4423	4424	4425	4426	4427	4430	4431
	3	4432	4433	4434	4435	4436	4437	4440	4441	4442	4443
	4	4444	4445	4446	4447	4450	4451	4452	4453	4454	4455
TENS	5	4456	4457	4460	4461	4462	4463	4464	4465	4466	4467
	6	4470	4471	4472	4473	4474	4475	4476	4477	4500	4501
	7	4502	4503	4504	4505	4506	4507	4510	4511	4512	4513
	8	4514	4515	4516	4517	4520	4521	4522	4523	4524	4525
	9	4526	4527	4530	4531	4532	4533	4534	4535	4536	4537

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

2400 Through 2599

		UNITS									
		0	1	2	3	4	5	6	7	8	9
24	0	4540	4541	4542	4543	4544	4545	4546	4547	4550	4551
	1	4552	4553	4554	4555	4556	4557	4560	4561	4562	4563
	2	4564	4565	4566	4567	4570	4571	4572	4573	4574	4575
	3	4576	4577	4600	4601	4602	4603	4604	4605	4606	4607
	4	4610	4611	4612	4613	4614	4615	4616	4617	4620	4621
TENS	5	4622	4623	4624	4625	4626	4627	4630	4631	4632	4633
	6	4634	4635	4636	4637	4640	4641	4642	4643	4644	4645
	7	4646	4647	4650	4651	4652	4653	4654	4655	4656	4657
	8	4660	4661	4662	4663	4664	4665	4666	4667	4670	4671
	9	4672	4673	4674	4675	4676	4677	4700	4701	4702	4703
25	0	4704	4705	4706	4707	4710	4711	4712	4713	4714	4715
	1	4716	4717	4720	4721	4722	4723	4724	4725	4726	4727
	2	4730	4731	4732	4733	4734	4735	4736	4737	4740	4741
	3	4742	4743	4744	4745	4746	4747	4750	4751	4752	4753
	4	4754	4755	4756	4757	4760	4761	4762	4763	4764	4765
TENS	5	4766	4767	4770	4771	4772	4773	4774	4775	4776	4777
	6	5000	5001	5002	5003	5004	5005	5006	5007	5010	5011
	7	5012	5013	5014	5015	5016	5017	5020	5021	5022	5023
	8	5024	5025	5026	5027	5030	5031	5032	5033	5034	5035
	9	5036	5037	5040	5041	5042	5043	5044	5045	5046	5047

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

2600 Through 2799

		UNITS									
		0	1	2	3	4	5	6	7	8	9
26	0	5050	5051	5052	5053	5054	5055	5056	5057	5060	5061
	1	5062	5063	5064	5065	5066	5067	5070	5071	5072	5073
	2	5074	5075	5076	5077	5100	5101	5102	5103	5104	5105
	3	5106	5107	5110	5111	5112	5113	5114	5115	5116	5117
	4	5120	5121	5122	5123	5124	5125	5126	5127	5130	5131
TENS	5	5132	5133	5134	5135	5136	5137	5140	5141	5142	5143
	6	5144	5145	5146	5147	5150	5151	5152	5153	5154	5155
	7	5156	5157	5160	5161	5162	5163	5164	5165	5166	5167
	8	5170	5171	5172	5173	5174	5175	5176	5177	5200	5201
	9	5202	5203	5204	5205	5206	5207	5210	5211	5212	5213
27	0	5214	5215	5216	5217	5220	5221	5222	5223	5224	5225
	1	5226	5227	5230	5231	5232	5233	5234	5235	5236	5237
	2	5240	5241	5242	5243	5244	5245	5246	5247	5250	5251
	3	5252	5253	5254	5255	5256	5257	5260	5261	5262	5263
	4	5264	5265	5266	5267	5270	5271	5272	5273	5274	5275
TENS	5	5276	5277	5300	5301	5302	5303	5304	5305	5306	5307
	6	5310	5311	5312	5313	5314	5315	5316	5317	5320	5321
	7	5322	5323	5324	5325	5326	5327	5330	5331	5332	5333
	8	5334	5335	5336	5337	5340	5341	5342	5343	5344	5345
	9	5346	5347	5350	5351	5352	5353	5354	5355	5356	5357

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

2800 Through 2999

		UNITS									
		0	1	2	3	4	5	6	7	8	9
28	0	5360	5361	5362	5363	5364	5365	5366	5367	5370	5371
	1	5372	5373	5374	5375	5376	5377	5400	5401	5402	5403
	2	5404	5405	5406	5407	5410	5411	5412	5413	5414	5415
	3	5416	5417	5420	5421	5422	5423	5424	5425	5426	5427
	4	5430	5431	5432	5433	5434	5435	5436	5437	5440	5441
TENS	5	5442	5443	5444	5445	5446	5447	5450	5451	5452	5453
	6	5454	5455	5456	5457	5460	5461	5462	5463	5464	5465
	7	5466	5467	5470	5471	5472	5473	5474	5475	5476	5477
	8	5500	5501	5502	5503	5504	5505	5506	5507	5510	5511
	9	5512	5513	5514	5515	5516	5517	5520	5521	5522	5523
29	0	5524	5525	5526	5527	5530	5531	5532	5533	5534	5535
	1	5536	5537	5540	5541	5542	5543	5544	5545	5546	5547
	2	5550	5551	5552	5553	5554	5555	5556	5557	5560	5561
	3	5562	5563	5564	5565	5566	5567	5570	5571	5572	5573
	4	5574	5575	5576	5577	5600	5601	5602	5603	5604	5605
TENS	5	5606	5607	5610	5611	5612	5613	5614	5615	5616	5617
	6	5620	5621	5622	5623	5624	5625	5626	5627	5630	5631
	7	5632	5633	5634	5635	5636	5637	5640	5641	5642	5643
	8	5644	5645	5646	5647	5650	5651	5652	5653	5654	5655
	9	5656	5657	5660	5661	5662	5663	5664	5665	5666	5667

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

3000 Through 3199

		UNITS									
		0	1	2	3	4	5	6	7	8	9
30	0	5670	5671	5672	5673	5674	5675	5676	5677	5700	5701
	1	5702	5703	5704	5705	5706	5707	5710	5711	5712	5713
	2	5714	5715	5716	5717	5720	5721	5722	5723	5724	5725
	3	5726	5727	5730	5731	5732	5733	5734	5735	5736	5737
	4	5740	5741	5742	5743	5744	5745	5746	5747	5750	5751
TENS	5	5752	5753	5754	5755	5756	5757	5760	5761	5762	5763
	6	5764	5765	5766	5767	5770	5771	5772	5773	5774	5775
	7	5776	5777	6000	6001	6002	6003	6004	6005	6706	6007
	8	6010	6011	6012	6013	6014	6015	6016	6017	6020	6021
	9	6022	6023	6024	6025	6226	6027	6030	6031	6032	6033
31	0	6034	6035	6036	6037	6040	6041	6042	6043	6044	6045
	1	6046	6047	6050	6051	6052	6053	6054	6055	6056	6057
	2	6060	6061	6062	6063	6064	6065	6066	6067	6070	6071
	3	6072	6073	6074	6075	6076	6077	6100	6101	6102	6103
	4	6104	6105	6106	6107	6110	6111	6112	6113	6114	6115
TENS	5	6116	6117	6120	6121	6122	6123	6124	6125	6126	6127
	6	6130	6131	6132	6133	6134	6135	6136	6137	6140	6141
	7	6142	6143	6144	6145	6146	6147	6150	6151	6152	6153
	8	6154	6155	6156	6157	6160	6161	6162	6163	6164	6165
	9	6166	6167	6170	6171	6172	6173	6174	6175	6176	6177

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

3200 Through 3399

		UNITS									
		0	1	2	3	4	5	6	7	8	9
32	0	6200	6201	6202	6203	6204	6205	6206	6207	6210	6211
	1	6212	6213	6214	6215	6216	6217	6220	6221	6222	6223
	2	6224	6225	6226	6227	6230	6231	6232	6233	6234	6235
	3	6236	6237	6240	6241	6242	6243	6244	6245	6246	6247
	4	6250	6251	6252	6253	6254	6255	6256	6257	6260	6261
TENS	5	6262	6263	6264	6265	6266	6267	6270	6271	6272	6273
	6	6274	6275	6276	6277	6300	6301	6302	6303	6304	6305
	7	6306	6307	6310	6311	6312	6313	6314	6315	6316	6317
	8	6320	6321	6322	6323	6324	6325	6326	6327	6330	6331
	9	6332	6333	6334	6335	6336	6337	6340	6341	6342	6343
33	0	6344	6345	6346	6347	6350	6351	6352	6353	6354	6355
	1	6356	6357	6360	6361	6362	6363	6364	6365	6366	6367
	2	6370	6371	6372	6373	6374	6375	6376	6377	6400	6401
	3	6402	6403	6404	6405	6406	6407	6410	6411	6412	6413
	4	6414	6415	6416	6417	6420	6421	6422	6423	6424	6425
TENS	5	6426	6427	6430	6431	6432	6433	6434	6435	6436	6437
	6	6440	6441	6442	6443	6444	6445	6446	6447	6450	6451
	7	6452	6453	6454	6455	6456	6457	6460	6461	6462	6463
	8	6464	6465	6466	6467	6470	6471	6472	6473	6474	6475
	9	6476	6477	6500	6501	6502	6503	6504	6505	6506	6507

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

3400 Through 3599

		UNITS									
		0	1	2	3	4	5	6	7	8	9
34	0	6510	6511	6512	6513	6514	6515	6516	6517	6520	6521
	1	6522	6523	6524	6525	6526	6527	6530	6531	6532	6533
	2	6534	6535	6536	6537	6540	6541	6542	6543	6544	6545
	3	6546	6547	6550	6551	6552	6553	6554	6555	6556	6557
	4	6560	6561	6562	6563	6564	6565	6566	6567	6570	6571
TENS	5	6572	6573	6574	6565	6576	6577	6600	6601	6602	6603
	6	6604	6605	6606	6607	6610	6611	6612	6613	6614	6615
	7	6616	6617	6620	6621	6622	6623	6624	6625	6626	6627
	8	6630	6631	6632	6633	6634	6635	6636	6637	6640	6641
	9	6642	6643	6644	6645	6646	6647	6650	6651	6652	6653
35	0	6654	6655	6656	6657	6660	6661	6662	6663	6664	6665
	1	6666	6667	6670	6671	6672	6673	6674	6675	6676	6677
	2	6700	6701	6702	6703	6704	6705	6706	6707	6710	6711
	3	6712	6713	6714	6715	6716	6717	6720	6721	6722	6723
	4	6724	6725	6726	6727	6730	6731	6732	6733	6734	6735
TENS	5	6736	6737	6740	6741	6742	6743	6744	6745	6746	6747
	6	6750	6751	6752	6753	6754	6755	6756	6757	6760	6761
	7	6762	6763	6764	6765	6766	6767	6770	6771	6772	6773
	8	6774	6775	6776	6777	7000	7001	7002	7003	7004	7005
	9	7006	7007	7010	7011	7012	7013	7014	7015	7016	7017

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

3600 Through 3799

		UNITS									
		0	1	2	3	4	5	6	7	8	9
36	0	7020	7021	7022	7023	7024	7025	7026	7027	7030	7031
	1	7032	7033	7034	7035	7036	7037	7040	7041	7042	7043
	2	7044	7045	7046	7047	7050	7051	7052	7053	7054	7055
	3	7056	7057	7060	7061	7062	7063	7064	7065	7066	7067
	4	7070	7071	7072	7073	7074	7075	7076	7077	7100	7101
TENS	5	7102	7103	7104	7105	7106	7107	7110	7111	7112	7113
	6	7114	7115	7116	7117	7120	7121	7122	7123	7124	7125
	7	7126	7127	7130	7131	7132	7133	7134	7135	7136	7137
	8	7140	7141	7142	7143	7144	7145	7146	7147	7150	7151
	9	7152	7153	7154	7155	7156	7157	7160	7161	7162	7163
37	0	7164	7165	7166	7167	7170	7171	7172	7173	7174	7175
	1	7176	7177	7200	7201	7202	7203	7204	7205	7206	7207
	2	7210	7211	7212	7213	7214	7215	7216	7217	7220	7221
	3	7222	7223	7224	7225	7226	7227	7230	7231	7232	7233
	4	7234	7235	7236	7237	7240	7241	7242	7243	7244	7245
TENS	5	7246	7247	7250	7251	7252	7253	7254	7255	7256	7257
	6	7260	7261	7262	7263	7264	7265	7266	7267	7270	7271
	7	7272	7273	7274	7275	7276	7277	7300	7301	7302	7303
	8	7304	7305	7306	7307	7310	7311	7312	7313	7314	7315
	9	7316	7317	7320	7321	7322	7323	7324	7325	7326	7327

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Con't)
(0-4096)

3800 Through 3999

		UNITS									
		0	1	2	3	4	5	6	7	8	9
38	0	7330	7331	7332	7333	7334	7335	7336	7337	7340	7341
	1	7342	7343	7344	7345	7346	7347	7350	7351	7352	7353
	2	7354	7355	7356	7357	7360	7361	7362	7363	7364	7365
	3	7366	7367	7370	7371	7372	7373	7374	7375	7376	7377
	4	7400	7401	7402	7403	7404	7405	7406	7407	7410	7411
TENS	5	7412	7413	7414	7415	7416	7417	7420	7421	7422	7423
	6	7424	7425	7426	7427	7430	7431	7432	7433	7434	7435
	7	7436	7437	7440	7441	7442	7443	7444	7445	7446	7447
	8	7450	7451	7452	7453	7454	7455	7456	7457	7460	7461
	9	7462	7463	7464	7465	7466	7467	7470	7471	7472	7473
39	0	7474	7475	7476	7477	7500	7501	7502	7503	7504	7505
	1	7506	7507	7510	7511	7512	7513	7514	7515	7516	7517
	2	7520	7521	7522	7523	7524	7525	7526	7527	7530	7531
	3	7532	7533	7534	7535	7536	7537	7540	7541	7542	7543
	4	7544	7545	7546	7547	7550	7551	7552	7553	7554	7555
TENS	5	7556	7557	7560	7561	7562	7563	7564	7565	7566	7567
	6	7570	7571	7572	7573	7574	7575	7576	7577	7600	7601
	7	7602	7603	7604	7605	7606	7607	7610	7611	7612	7613
	8	7614	7615	7616	7617	7620	7621	7622	7623	7624	7625
	9	7626	7627	7630	7631	7632	7633	7634	7635	7636	7637

PX 71871

DECIMAL TO OCTAL CONVERSION TABLE (Concl.)
(0-4096)

4000 Through 4096

		UNITS									
		0	1	2	3	4	5	6	7	8	9
40	0	7640	7641	7642	7643	7644	7645	7646	7647	7650	7651
	1	7652	7653	7654	7655	7656	7657	7660	7661	7662	7663
	2	7664	7665	7666	7667	7670	7671	7672	7673	7674	7675
	3	7676	7677	7700	7701	7702	7703	7704	7705	7706	7707
	4	7710	7711	7712	7713	7714	7715	7716	7717	7720	7721
TENS	5	7722	7723	7724	7725	7726	7727	7730	7731	7732	7733
	6	7734	7735	7736	7737	7740	7741	7742	7743	7744	7745
	7	7746	7747	7750	7751	7752	7753	7754	7755	7756	7757
	8	7760	7761	7762	7763	7764	7765	7766	7767	7770	7771
	9	7772	7773	7774	7775	7776	7777	10000			

PX 71871