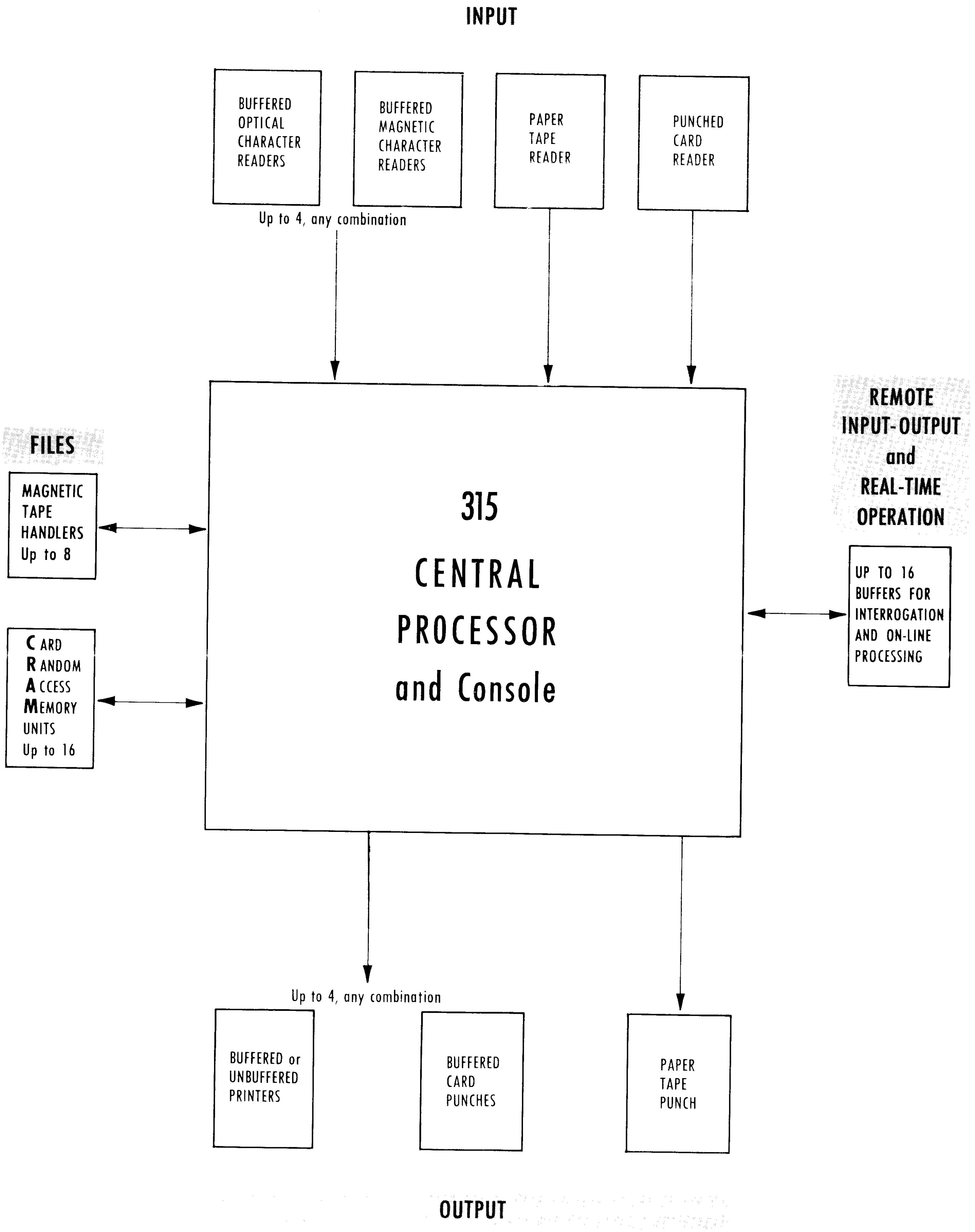
NCR

# REFERENCE MANUAL

PROGRAMMING<br>HANDBOOK

●

NCR 315


INPUT
BUFFERED OPTICAL CHARACTER READERS
BUFFERED MAGNETIC CHARACTER READERS
Up to 4, any combination
PAPER TAPE READER
PUNCHED CARD READER
FILES
MAGNETIC TAPE HANDLERS Up to 8
CARD RANDOM ACCESS MEMORY UNITS Up to 16
315
CENTRAL PROCESSOR
and Console
REMOTE INPUT-OUTPUT and REAL-TIME OPERATION
UP TO 16 BUFFERS FOR INTERROGATION AND ON-LINE PROCESSING
Up to 4, any combination
BUFFERED or UNBUFFERED PRINTERS
BUFFERED CARD PUNCHES
PAPER TAPE PUNCH
OUTPUT

# 315
# *PROGRAMMING HANDBOOK*

THE NATIONAL CASH REGISTER COMPANY
DAYTON 9, OHIO, U.S.A.

PRINTED IN U. S. A.

# TABLE OF CONTENTS

## *INTERNAL OPERATIONS*

## *INPUT, OUPTUT, FILE OPERATIONS*

## *APPENDIX*

# INDEX

## INTERNAL OPERATIONS

*These instructions permit a "literal" to be named as "A" if desired.

**These instructions *require* a "literal" to be named as "A".

| | | |
|---|---|---|
| SCND:G | Scan digits for greater, all positions. (Same as G7) | 32 |
| SCND:G1<br>SCND:G2<br>SCND:G3<br>SCND:G4<br>SCND:G5<br>SCND:G6<br>SCND:G7 | Selective Scans | |
| SCND:L | Scan digits for less, all positions. (Same as L7) | 32 |
| SCND:L1<br>SCND:L2<br>SCND:L3<br>SCND:L4<br>SCND:L5<br>SCND:L6<br>SCND:L7 | Selective Scans | |
| **SETF:– | Set Sign flag minus | 24 |
| **SETF:+ | Set Sign flag plus | 24 |
| **SETF:D | Set Demand Permit flag | 24 |
| **SETF:θ | Set Overflow flag | 24 |
| **SETF:T | Set Tracer Permit flag | 24 |
| *SETF:LH | Set Left-hand Memory flag | 24 |
| *SETF:RH | Set Right-hand Memory flag | 24 |
| *SHFT:AL | Shift alphas left | 28 |
| *SHFT:AR | Shift alphas right | 28 |
| *SHFT:DL | Shift digits left | 28 |
| *SHFT:DR | Shift digits right | 28 |
| *SHFT:RR | Shift digits right and roundoff | 28 |
| *SHFT:LC | Shift digits left circular | 28 |
| *SHFT:RC | Shift digits right circular | 28 |
| **SKIP: | Skip within the program | 26 |
| SLD :J | Spread-load J-registers | 29 |
| SLD :R | Spread-load R-registers | 29 |
| **SPRD:B | Spread a literal in Memory, start at beginning of area | 31 |
| **SPRD:E | Spread a literal in Memory, start at end of area | 31 |
| *ST : | Store Accumulator | 18 |
| ST :J | Store J-registers | 29 |
| ST :R | Store R-registers | 29 |
| STDA: | Store digit-to-alpha All characters | 36 |
| *SUB : | Subtract from Accumulator | 19 |
| SUPP: | Suppress | 24 |
| TEST:– | Test Sign flag negative | 25 |
| TEST:D | Test Demand Permit flag | 25 |
| TEST:θ | Test Overflow flag | 25 |
| TEST:T | Test Tracer Permit flag | 25 |
| TEST:G | Test Greater flag | 25 |
| TEST:L | Test Less flag | 25 |
| TEST:E | Test Equal flag | 25 |
| *TEST:LH | Test Left-hand Memory flag | 25 |
| *TEST:RH | Test Right-hand Memory flag | 25 |
| TEST:SL | Did Scan stop on left digit or alpha? | 25 |
| TEST:SM | Did Scan stop on middle digit? | 25 |
| TEST:SR | Did Scan stop on right digit or alpha? | 25 |
| *TEST:SW | Test Console Option switch | 25 |

*These instructions permit a "literal" to be named as "A" if desired.

**These instructions *require* a "literal" to be named as "A".

## INPUT, OUTPUT, FILE OPERATIONS

| | | |
|---|---|---|
| BACK: | Backup Magnetic Tape | 63 |
| *CLRU:C | Clear Unit Demand in CRAM | 49 |
| *CLRU:P | Clear Unit Demand in Printer or Card Punch | 49 |
| *CLRU:S | Clear Unit Demand in Sorter-Reader | 49 |
| *CLRU:Q | Clear Unit Demand in Inquiry buffer | 49 |
| HALT:A | Halt and accept Alpha input from Console | 56 |
| HALT:D | Halt and accept Digit input from Console | 56 |
| PNCH: | Punch a card | 55 |
| PPT :C | Punch Paper Tape, Character Mode | 50 |
| PPT :S | Punch Paper Tape, Slab Mode | 50 |
| PRNT: | Print, Buffered Printer | 56 |
| RCC : | Read CRAM Card | 64 |
| RCØL: | Read Columns of a Punched Card, 400 CPM Reader | 52 |
| | Read Columns of a Punched Card, 2000 CPM Reader | 54 |
| RCØL:F | Feed a Card, Read Columns, 400 CPM Reader | 52 |
| | Feed a Card, Read Columns, 2000 CPM Reader | 54 |
| RCØL:T | Read Columns, with translation by Reader | 54 |
| RCØL:TF | Feed a Card, Read Columns with translation | 54 |
| RMT : | Read Magnetic Tape | 60 |
| RPT :C | Read Paper Tape, Character Mode | 51 |
| RPT :CX | Read Paper Tape, Character Mode, Extra variation | 51 |
| RPT :S | Read Paper Tape, Slab Mode | 51 |
| SELC:DP | Select a CRAM, Drop a card, use Present card next | 48 |
| SELC:DN | Select a CRAM, Drop a card, use Next card next | 48 |
| *SELC:T | Select a CRAM, Test for Ready | 48 |
| *SELC:R | Select a CRAM, Release present card | 48 |
| *SELP: | Select a Printer or Card Punch | 49 |
| *SELS: | Select a Sorter-Reader | 49 |
| *SELQ: | Select an Inquiry buffer | 49 |
| *SETU:C | Set Unit Demand in CRAM | 49 |
| *SETU:P | Set Unit Demand in Printer or Card Punch | 49 |
| *SETU:S | Set Unit Demand in Sorter-Reader | 49 |
| *SETU:Q | Set Unit Demand in Inquiry buffer | 49 |
| TYPE:A | Type Alphas on Console | 60 |
| TYPE:AP | Type Alphas on Console, Programmed format | 60 |
| TYPE:D | Type Digits on Console | 60 |
| WIND: | Rewind Magnetic Tape | 63 |
| WIND:L | Rewind Magnetic Tape with Use Lockout | 63 |
| WCC : | Write CRAM Card | 66 |
| WMT : | Write Magnetic Tape | 62 |

*These instructions permit a "literal" to be named as "A" if desired.

## MEMORY, STORAGE OF INFORMATION:

A fundamental characteristic of any Electronic Data Processor is its internal information-storage, or *memory,* in which it is able to store both that part of the data which is being operated on at the moment, and also the program for processing that data. National's 315 Data Processor is available with memories of 2 000, 5 000, 10 000, 15 000, 20 000 or 40 000 permanently-numbered storage *locations,* which contain stored information. The number assigned to each location is its *address.* The range of addresses is:

| MEMORY SIZE | ADDRESSES |
|---|---|
| 2 000 | 00 000 thru 01 999 |
| 5 000 | 00 000 thru 04 999 |
| 10 000 | 00 000 thru 09 999 |
| 15 000 | 00 000 thru 14 999 |
| 20 000 | 00 000 thru 19 999 |
| 40 000 | 00 000 thru 39 999 |

Memory references are *cyclic modulo memory-size.* That is, if an address is used which is beyond the memory, the memory-size is automatically subtracted from this address again and again, until a new address is obtained which is within the memory. However, with 15 000-slab memory:

| | | |
|---|---|---|
| 15 000 thru 19 999 | interpreted as | 10 000 thru 14 999 |
| 20 000 thru 34 999 | interpreted as | 00 000 thru 14 999 |
| 35 000 thru 39 999 | interpreted as | 10 000 thru 14 999 |

Information is stored in memory by means of magnetic cores, which are tiny rings of ferrite material, strung on a lattice of wires. Each core may be selectively magnetized in either of two states which, for convenience, are designated **0** and **1**. These symbols are not numbers; they are merely convenient marks used to distinguish the two states of a single magnetic core, and any other pair of conventional symbols would serve as well. The marks **0** and **1**, corresponding to the two possible magnetized states of a core, are called *bits* and therefore a single core may store either a **0-bit** or a **1-bit**.

Information may be either numeric or alphanumeric. Numeric information (spoken of as "Digits") comprises the 10 decimal digits and the six symbols (the *non-decimal digits*) shown in the first row of the Language Code Table. Alphanumeric information (spoken of as "Alphas") comprises the entire set of 64 characters shown in the four rows of the table.

A Digit is represented by a combination of four bits, stored in four magnetic cores, whereas an Alpha is represented by a combination of six bits, stored in six magnetic cores. Of these six bits, the right-hand four are called *numeric bits* and the left-hand two are called *zone bits.*

It will be evident that the sixteen characters which appear in the first row of the table may be represented within the processor memory as either 4-bit Digits or 6-bit Alphas, and in practice they are stored in both forms at different times. All input-output communication with paper tape, punched cards, and printer is performed in terms of Alphas; all arithmetic operations are performed in terms of Digits; at other times, the convenience of the programmer will determine the form in which numeric information is stored. Special operations are included in the processor to condense and expand information from one form to the other.

The information stored in a single memory location is called a *slab,* and consists of 12 bits. Since these 12 bits may be divided into two groups of 6, or into three groups of 4, a slab may store either two Alphas or three Digits:

| | | | |
|---|---|---|---|
| B R | 3 A | 2 4 6 | • 4 9 |

Alpha slabs Digit slabs

The term *slab* is a contraction of "syllable": part of a word.

### LANGUAGE TABLE

| ZONE BITS | NUMERIC BITS | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| 00 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ | , | ⍁ | & | . | – |
| 01 | ! | A | B | C | D | E | F | G | H | I | ; | " | ? | : | ← | ↑ |
| 10 | + | J | K | L | M | N | θ | P | Q | R | % | = | $ | ( | ) | / |
| 11 | * | # | S | T | U | V | W | X | Y | Z | < | > | ' | [ | ] | \ |

The symbol ⍁ is conventionally used to represent *space.*

The basic unit of information for processing is the *word,* which contains a single item of information, such as Account Number, Name, Gross Pay Year-to-Date, Quantity on Hand, etc. A word may be up to 8 slabs long, and will usually contain all Digit, or all Alpha, information.

The algebraic sign of a Digit-word is determined by its extreme LH (left-hand) digit. If the LH digit is the character *hyphen,* then the word is negative; if the LH digit is anything else, then the word is positive. Thus the number +7968 would be stored in a 2-slab word as:

| | |
|---|---|
| 0 0 7 | 9 6 8 |

and in a 3-slab word as:

| | | |
|---|---|---|
| 0 0 0 | 0 0 7 | 9 6 8 |

whereas the number -7968 would be stored in a 2-slab word as:

| | |
|---|---|
| - 0 7 | 9 6 8 |

and in a 3-slab word as:

| | | |
|---|---|---|
| - 0 0 | 0 0 7 | 9 6 8 |

Therefore the longest negative number that can be stored in a given word is one digit shorter than the longest positive number that can be stored in the same word.

A word is referred to by naming the address of its LH slab, and by specifying its length (1 to 8 slabs). There are no markers within the information to designate the beginning and end of a word, and therefore the programmer may, at his convenience, regard a sequence of slabs sometimes as a single word, and at other times as several words.

Suppose the following three words are in memory, describing an item in the inventory file:

| 00 101 | 00 102 | 00 103 | 00 104 |
|---|---|---|---|
| | | | |
| STOCK | NUMBER | SIZE | COLOR |

The programmer may, if he chooses, compare the 4-slab word starting at location 00 101 with a similar 4-slab word elsewhere in memory (containing the same information about an item received) to see if they are the same. Then, for some succeeding operation, he may again regard this information as comprising three different words.

## *ACCUMULATOR:*

In addition to the numbered locations of memory, the processor contains an 8-slab storage called the Accumulator, and referred to as @. It is implicitly involved in almost every operation performed by the processor, although it is never named explicitly. The capacity of the Accumulator is 16 Alphas, or 24 Digits. Since the sign of the Accumulator is held in the Sign flag (described later) rather than in the Accumulator itself, the intermediate results of a computation may range up to 24 digits, positive or negative. However, the final result which is to be stored in memory may not exceed 24 digits positive, or 23 digits negative, unless double-precision techniques are used.

Consider the problem of adding the contents of two Digit-words, and storing their sum in a third word. Suppose the initial contents of the three words, and of the Accumulator are:

| 02 344 | 02 345 | 02 346 | 02 347 |
|---|---|---|---|
| 8 4 2 | 4 7 7 | 5 2 3 | 9 4 6 |

| 16 999 | 17 000 | 17 001 | 17 002 | 17 003 |
|---|---|---|---|---|
| 3 2 8 | 0 0 5 | 0 0 0 | 0 0 1 | 7 3 1 |

| 12 210 | 12 211 | 12 212 | 12 213 | 12 214 |
|---|---|---|---|---|
| 7 2 6 | 3 6 8 | 3 2 0 | 8 3 6 | 4 3 8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 4 0 8 | 4 6 9 | 2 5 0 | 8 6 2 | 4 3 6 | 7 9 9 | @ |

- First LOAD the Accumulator with the contents of the 2-slab word starting at location 02 345. The Accumulator now contains:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 0 0 | 0 0 0 | 0 0 0 | 0 0 0 | 4 7 7 | 5 2 3 | @ |

- Then ADD to the Accumulator the contents of the 3-slab word starting at location 17 000. The Accumulator now contains:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 0 0 | 0 0 0 | 0 0 0 | 0 0 5 | 4 7 7 | 5 2 4 | @ |

- Now STORE the contents of the Accumulator in the 3-slab word starting at location 12 211. The three words and the Accumulator now contain:

| 02 344 | 02 345 | 02 346 | 02 347 |
|---|---|---|---|
| 8 4 2 | 4 7 7 | 5 2 3 | 9 4 6 |

| 16 999 | 17 000 | 17 001 | 17 002 | 17 003 |
|---|---|---|---|---|
| 3 2 8 | 0 0 5 | 0 0 0 | 0 0 1 | 7 3 1 |

| 12 210 | 12 211 | 12 212 | 12 213 | 12 214 |
|---|---|---|---|---|
| 7 2 6 | 0 0 5 | 4 7 7 | 5 2 4 | 4 3 8 |

| 0 | 0 0 0 | 0 0 0 | 0 0 0 | 0 0 5 | 4 7 7 | 5 2 4 | @ |
|---|---|---|---|---|---|---|---|

This illustrates several important characteristics of the processor:

- The processor selects the desired words within memory, without being concerned about the information on either side of a word.
- "Reading" or copying information from a word, or from any register, does not alter the information in that word or register.
- Placing information into a word or a register completely replaces the information previously there, and the previous information is then lost.
- All transfers of information within the processor are *right-justified.* That is, the right-hand end of the information-source is always lined up with the right-hand end of the information-destination.
- If the destination is longer than the source, the destination is always filled out to the left with zeros.

### INDEX REGISTERS (R-registers):

An instruction of the type just illustrated names an **Operation**, an **Address** in memory, and a **Word Length.** The instruction also names one of 32 **Index Registers** which are always used by the processor in executing any instruction. An index register holds an address—a positive number up to 39 999.

In order to determine the actual address in memory to which the instruction refers, the processor automatically adds the address named in the instruction, plus the address stored in the index register.

Suppose that index registers 16, 17, 18 contained:

| | |
|---|---|
| **16** | 0 2 0 0 5 |
| **17** | 1 6 9 0 0 |
| **18** | 1 2 2 0 0 |

Then the three instructions just illustrated could have been written:

| | | | |
|---|---|---|---|
| LOAD | 2 slabs from | (R16) 340. | [from 02 345] |
| ADD | 3 slabs from | (R17) 100. | [from 17 000] |
| STORE | 3 slabs in | (R18) 011. | [in 12 211] |

### ADDRESSING METHODS:

An instruction names a positive 3-digit number in the address column, specifying the *position* of a word within an item of data. The index register names the *base* of the item—the address of the first slab of the item.

Suppose a series of transactions in memory which are to be posted by a Commercial Bank to its checking-account file. Each transaction comprises four words, occupying nine slabs of memory, and the first transaction starts in location 04 996:

| | Address | L | Pos. | D/A | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Remarks |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1st TRANSACTION | 04 996 | 3 | 0 | D | | | | | | | | | Account number |
| | 04 999 | 4 | 3 | D | | | | | | | | | Amount of trans. |
| | 05 003 | 1 | 7 | D | | | | | | | | | Transaction code |
| | 05 004 | 1 | 8 | D | | | | | | | | | Batch number |
| 2nd TRANSACTION | 05 005 | 3 | 0 | D | | | | | | | | | Account number |
| | 05 008 | 4 | 3 | D | | | | | | | | | Amount of trans. |
| | 05 012 | 1 | 7 | D | | | | | | | | | Transaction code |
| | 05 013 | 1 | 8 | D | | | | | | | | | Batch number |
| 3rd TRANSACTION | 05 014 | 3 | 0 | D | | | | | | | | | Account number |
| | 05 017 | 4 | 3 | D | | | | | | | | | Amount of trans. |
| | 05 021 | 1 | 7 | D | | | | | | | | | Transaction code |
| | 05 022 | 1 | 8 | D | | | | | | | | | Batch number |
| 4th TRANS. | 05 023 | 3 | 0 | D | | | | | | | | | Account number |
| | 05 026 | 4 | 3 | D | | | | | | | | | Amount of trans. |

In the program for posting these transactions, every instruction referring to Account Number will contain **000** as its address reference; every instruction referring to Amount will contain **003**; every instruction referring to Transaction Code will contain **007**; etc.

Before starting to post, an index register will be preset to contain **04 996**. Every instruction will then refer to the appropriate word in the first transaction. When that transaction is completely posted, the index register will be augmented by **9**, and will then contain **05 005** (the address of the first slab of the second transaction) whereupon each instruction will refer to the appropriate word in the second transaction. And so on.

In general, one Index Register will be used to control each data stream.

## *JUMP REGISTERS (J-registers):*

A total of 32 jump registers are also provided in the processor. Each of these holds an address—a positive number up to 39 999.

Many instructions provide for alternate *exits,* depending on conditions encountered while the instruction is being executed. If such a condition is found, then the processor will not proceed to the next instruction in sequence when the instruction is completed, but will instead *jump* to an instruction whose address is stored in one of the J-registers.

Such an instruction will name a J-register as being the first register in the *jump table* for that instruction. The jump table contains as many J-registers as there are possible exits from that particular instruction. Suppose, for example, that some instruction has three possible exits, corresponding to conditions "A", "B", "C"; and in writing the instruction the programmer specifies J23 as the beginning of the jump table. Then if the instruction finds condition "A", the processor will jump to the address stored in J23 after this instruction is complete; if condition "B", it will jump to the address stored in J24; if condition "C", it will jump to the address stored in J25. If none of these conditions exist, the processor will execute the next instruction in the normal program sequence.

Certain of the R-registers and J-registers perform special functions, and are not normally used in the fashion just described. A detailed discussion of the characteristics of those registers is given later.

## *SINGLE STAGE INSTRUCTIONS:*

A single stage instruction is written in the following format:

| Op | V | L | X | A |
|---|---|---|---|---|
| | | | | |

**Op** and **V** name the operation to be performed, and the variation if any. **L** indicates the length of the word (up to 8 slabs), **X** the index register to be used, and **A** the address reference. The processor obtains the actual address by adding **A** to the contents of the index register.

When this instruction is actually stored in the processor memory, as part of a program to be executed, it occupies two slabs of memory. The first slab contains **Op**, **V**, **L** and **X**, all condensed into 12 bits; the second slab contains **A**.

## *DOUBLE STAGE INSTRUCTIONS:*

A double stage instruction is written in the following format:

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| | | | | |
| | | | | |

Note that the instruction requires two lines, and that the **L** column is not used. **A** is the address reference; **B** specifies other information as required by each instruction. **X** *usually* specifies an R-register, and **Y** *usually* specifies a J-register, but not invariably. Therefore in the description of each instruction, an R-register will be called **RX** if it is specified by **X**, and **RY** if it is specified by **Y**; a J-register will be called **JX** if it is specified by **X**, and **JY** if it is specified by **Y**.

When this instruction is stored in memory, it occupies four slabs. **Op**, **V**, **X** and **Y** are condensed into the 24 bits of the first and third slabs; the second and fourth slabs contain **A** and **B** respectively.

## *FLAGS:*

A number of *flags* are provided in the processor. These may be turned on and off by the program at will, and perform certain automatic functions, as well as storing conditions which the program may test at a later time. The terms *on/off, set/cleared* are used interchangeably to describe the two states of a flag.

### SIGN FLAG:

This flag is associated with the Accumulator; it automatically indicates the algebraic sign of the Accumulator contents, and governs all arithmetic operations accordingly. However, for special purposes, it may be set *positive* or *negative* by the program, independently of the Accumulator contents.

Testing the Sign flag does not change its setting.

### OVERFLOW FLAG:

Whenever an attempt is made to store more information into a word than that word can hold, putaway stops at the LH (left hand) end of the word, and the remaining information is not stored. When this occurs, or when certain other conditions arise, the processor usually sets the Overflow flag. The precise circumstances under which each operation might set Overflow are described specifically for the individual operations.

The Overflow flag may also be set for special purposes by the program, independently of other operations.

Once Overflow has been set, the flag remains set until the program tests it, at which time it is automatically cleared.

### GREATER, LESS, EQUAL FLAGS:

These flags are set automatically, to indicate the result of a COMPARE or a COUNT instruction, and also to record detailed information about the execution of a SUPPRESS or a SCAN instruction.

The **G**, **L** and **E** flags are not independent; only one of them may be on at a time. However, they may all be off at the same time.

Testing these flags does not change their setting.

### MEMORY FLAGS:

It is often convenient for the programmer to designate flags of his own, to record information for later use. For example, it may take many tests to determine if an individual employee should receive overtime pay, and this *yes* or *no* answer may be required at two or more widely separated points in the payroll program. Rather than repeat the entire series of tests each time this answer is needed, the program obtains the answer the first time, and records it in a memory flag. Thereafter it is only necessary to test this flag each subsequent time the answer is needed.

Any slab in memory may be designated as containing a pair of flags, corresponding to the two Alphas which the slab can store. These then become the LH (left hand) and RH (right hand) flags in that slab. If a flag contains the Alpha character *zero* it is **off**; if it contains any other Alpha character it is **on**. These flags may be set, cleared, and tested independently of each other. Since any slab of memory may contain a pair of flags, the number of such flags available to the programmer is practically unlimited.

Testing a memory flag does not change its setting.

### DEMAND PERMIT FLAG:

Many of the peripheral units have the ability to interrupt the main program (*to demand* processor attention) when they have completed an operation previously assigned to them. In this fashion, the relatively slow input-output units may be kept running at maximum rate, while the processor is performing some other job; occasionally the main program will be interrupted for a brief interval to attend to one of the input-output units, and then immediately resume, while the slow unit continues to operate independently at its own speed.

The programmer will wish to permit Demand interruption during certain portions of his program, and to forbid it during other portions. The Demand Permit flag gives him this facility.

While the Demand Permit flag is on, any peripheral unit whose Unit Demand flag (see below) is on, will exercise Demand whenever it is ready to receive information from the processor, or to deliver information to the processor.

When Demand is exercised, the processor always completes the current instruction in the main program, then jumps to the demand program. The programmer may specify different demand programs for different portions of the main program, if he wishes. There is a single entry-point to any demand program, regardless of which among several possible peripheral units may have interrupted, and no indication is furnished to show which unit actually did exercise Demand. This gives the programmer complete flexibility in assigning priorities among competitively demanding units. The demand program merely attempts to SELECT each unit in turn, until it finds one which is in the *ready* state, and then gives attention to that unit. The assigning of priorities among units is performed in the simplest possible fashion—by specifying the sequence in which the units are tested.

Entering a demand program turns off the Demand Permit flag, since normally it is not desirable to have the demand program itself interrupted. However, the programmer may permit this if he chooses, merely by having the demand program turn the Demand Permit flag back on. Just before this, he will probably have specified a different demand program to be used for the time being.

The instructions TEST:D, TEST:T, TEST:SW, SETF:D, SETF:T, SETU, CLRU, SELect are *protected* against Demand; the processor never permits interrupt after completion of one of these instructions.

The program may set and test the Demand Permit flag. It is turned off automatically either when tested, or when Demand interrupt occurs.

### UNIT DEMAND FLAGS:

The following peripheral units are capable of exercising Demand: Printers, Card Readers, Card Punches, CRAMs (Card Random Access Memories), Magnetic Character Readers, Inquiry Stations. Each of these units has within it a Unit Demand flag, which may be set and cleared by the program.

The programmer will often wish to permit some of the peripheral units, but to forbid others, to exercise Demand during a particular part of the program. He then has the program turn the Unit Demand flags in the permitted units *on*, and those in the forbidden units *off*. Any device whose Unit Demand is off may still be used by the processor at any time; but it must await the program's convenience, rather than being able to demand attention at its own convenience.

### TRACER PERMIT FLAG:

To facilitate code-checking, it is customary to use supplemental *tracing* or *automonitoring* programs which permit the operator to follow the execution of the program being checked. In order to provide communication between the main program (the one being checked) and the tracing program, a Tracer interrupt facility is provided, controlled by the Tracer Permit flag.

When this flag is on, *and* the appropriate Console switch is also on, then at the conclusion of the execution of each instruction in the main program the processor automatically jumps to the tracing program.

The instructions TEST:D, TEST:T, SETF:D, SETF:T, DLR are *protected* against tracing; the processor never permits Tracer interrupt after completion of one of these five instructions.

The program may set and test the Tracer Permit flag. It is turned off automatically either when tested, or when Tracer interrupt occurs.

## *REGISTERS:*

The 32 R-registers and 32 J-registers may be thought of as existing in the following array, which indicates the special functions assigned to some of the registers:

| RELATIVE ADDRESS (INDEX) R—REGISTERS | | | | JUMP J—REGISTERS | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 0 0 0 0 | 16 | 1 0 0 0 0 | 0 | STEP uses Registers 00 through 05 | 16 | MICR Sorter-Reader uses Registers 16 through 19 |
| 1 | 0 1 0 0 0 | 17 | 1 1 0 0 0 | 1 | | 17 | |
| 2 | 0 2 0 0 0 | 18 | 1 2 0 0 0 | 2 | | 18 | |
| 3 | 0 3 0 0 0 | 19 | 1 3 0 0 0 | 3 | | 19 | |
| 4 | 0 4 0 0 0 | 20 | 1 4 0 0 0 | 4 | | 20 | Inquiry System uses Registers 20 through 22 |
| 5 | 0 5 0 0 0 | 21 | 1 5 0 0 0 | 5 | | 21 | |
| 6 | 0 6 0 0 0 | 22 | 1 6 0 0 0 | 6 | PACE uses Registers 06 through 11 | 22 | |
| 7 | 0 7 0 0 0 | 23 | 1 7 0 0 0 | 7 | | 23 | |
| 8 | 0 8 0 0 0 | 24 | 1 8 0 0 0 | 8 | | 24 | |
| 9 | 0 9 0 0 0 | 25 | 1 9 0 0 0 | 9 | | 25 | |
| 10 | | 26 | | 10 | | 26 | |
| 11 | | 27 | | 11 | | 27 | |
| 12 | | 28 | Registers 28 and 29 are used by STEP, PACE, macros and subroutines. Their contents are not preserved. | 12 | | 28 | Registers 28 and 29 are used by macros and subroutines. Their contents are not preserved. |
| 13 | | 29 | | 13 | | 29 | |
| 14 | | 30 | Processor stores an address | 14 | Jump-Table Link | 30 | Demand-Program Jump |
| 15 | Main Link (Program Decision) | 31 | Sequence-Control Register | 15 | Demand-Program Link | 31 | Tracer-Program Jump |

## *USING THE REGISTERS:*

### LOADING THE REGISTERS:

An address is loaded into a register from a *memory pair*—a 2-slab word of memory. In this loading operation only the RH 18 bits (4½ digits) of the pair are placed into the register.

This is equivalent to saying that, if an attempt is made to load a negative number into a register, the negative sign is ignored, and the number is loaded as positive. If an attempt is made to load a number greater than 39 999, the processor automatically subtracts 40 000 from that number again and again until the result is less than 40 000.

### STORING THE REGISTERS:

An address is stored from a register into the RH 18 bits of a memory pair. The LH 6 bits (1½ digits) of the pair are automatically set to zero.

### ADDING AND SUBTRACTING IN THE REGISTERS:

Any addition and subtraction performed in the registers is modulo 40 000.

This is equivalent to saying that, if a number is added to the contents of a register and the sum is greater than 39 999, the processor automatically subtracts 40 000 again and again until the result is less than 40 000. If a number is subtracted from the contents of a register, and the result is negative, the processor automatically adds 40 000 again and again until the result is positive (or zero).

NOTE: The contents of a register is always a positive number from 00 000 thru 39 999, since each of the registers contains 18 bit-positions. In loading a register, only the RH 18 bits of the memory pair are loaded, with the LH 6 bits ignored. In storing a register, the RH 18 bits of the pair are stored, and the LH 6 bits set to zero. In adding and subtracting in a register, the augmenter is in a memory pair, and only the RH 18 bits of the augmenter are used; *except* that if the LH 4 bits of the augmenter form the Digit *hyphen* then the augmenter is treated as negative. The result of the addition or subtraction is stored in the register modulo 40 000 as a positive number.

However, once the processor is given an address to *lookup* in memory, that address is interpreted *modulo memory-size* in memories of less than 40 000 slabs, except for the special rule for 15 000-slab memory, stated on page 5.

## *SPECIAL FUNCTIONS OF SOME REGISTERS:*

In order to obtain cross-references within the program, the first 10 index registers (**R00** thru **R09**) always contain the addresses 00 000, 01 000, . . . . . . 09 000 respectively. Thus a jump to the instruction in location 06 785 would be written:

| Op | V | L | X | A |
|---|---|---|---|---|
| J U M P | | | 0 6 | 7 8 5 |

**R-30** When the processor performs certain operations whose scope is variable, it automatically stores an address in R30 to indicate where the operation terminated. Therefore R30 will not normally be used by the programmer for routine address-modification. There is a comment in the description of each instruction which stores information in R30.

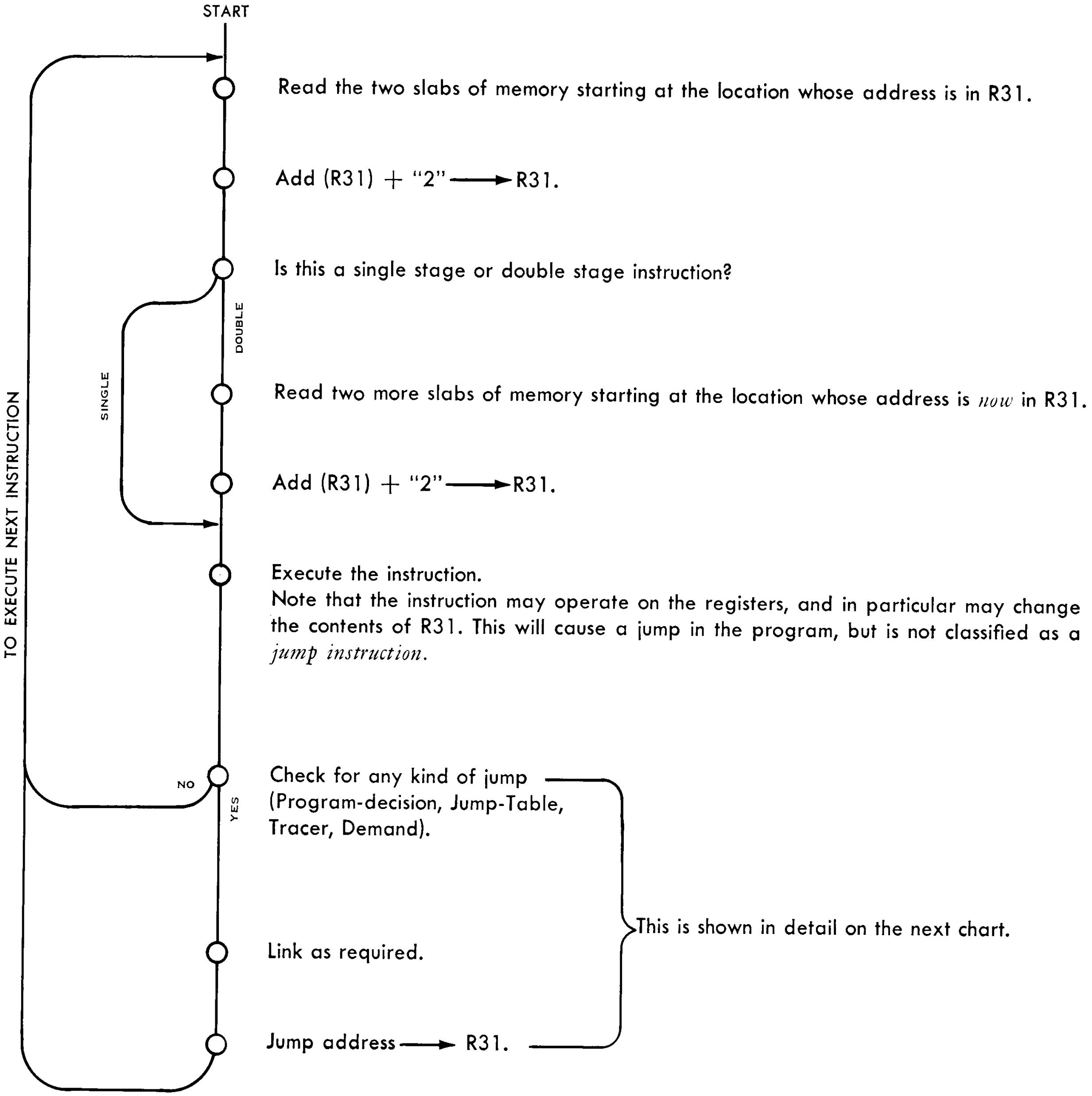
**R15** and **R31** may never be used for address-modification.

**R31** is used by the processor as its *Sequence-Control Register;* each time a new instruction is to be executed, the processor finds the address of that instruction in R31, and immediately replaces that address with the address of the next instruction in the normal sequence. If an instruction requires a *branch* or jump out of the normal program sequence, the processor saves the new contents of R31 as a *link* back to the normal sequence, and then plants the jump address into R31.

**R15** is used as the Link Register for program-decision jumps, and is called the *main* link. When the jump is the result of a program decision (such as TEST FLAG or Unconditional Jump) the contents of R31 are saved in R15 before the jump address is placed into R31. There are only three branching instructions (JUMP: IP; TEST: D; TEST: T) which do not link.

The programmer may at any time impose a jump on the program (without the use of a branching instruction) by changing or replacing the contents of R31. Since these are *operations* on one of the registers, they are not classified as jumps, and do not link. Similarly, the programmer is free to change or replace the contents of any of the link registers.

This process can best be illustrated by a flow chart which shows the detailed steps performed by the processor in executing an instruction. The notation (*R31*) means *the contents of R31.*



Read the two slabs of memory starting at the location whose address is in R31.

Add (R31) + "2" ⟶ R31.

Is this a single stage or double stage instruction?

Read two more slabs of memory starting at the location whose address is *now* in R31.

Add (R31) + "2" ⟶ R31.

Execute the instruction.
Note that the instruction may operate on the registers, and in particular may change the contents of R31. This will cause a jump in the program, but is not classified as a *jump instruction.*

Check for any kind of jump (Program-decision, Jump-Table, Tracer, Demand).

Link as required.

Jump address ⟶ R31.

This is shown in detail on the next chart.

**J30** and **J31** hold the Demand program and Tracer program jump addresses, respectively. The programmer stores the appropriate addresses in these registers at the beginning of the program, and of course he has the privilege of changing them at any time during the program if he sees fit to do so.

When either Demand interrupt or Tracer interrupt occurs, the contents of R31 are preserved in J15; then the contents of J30 or J31, as appropriate, are planted into R31, imposing a jump to the Demand or to the Tracer program.

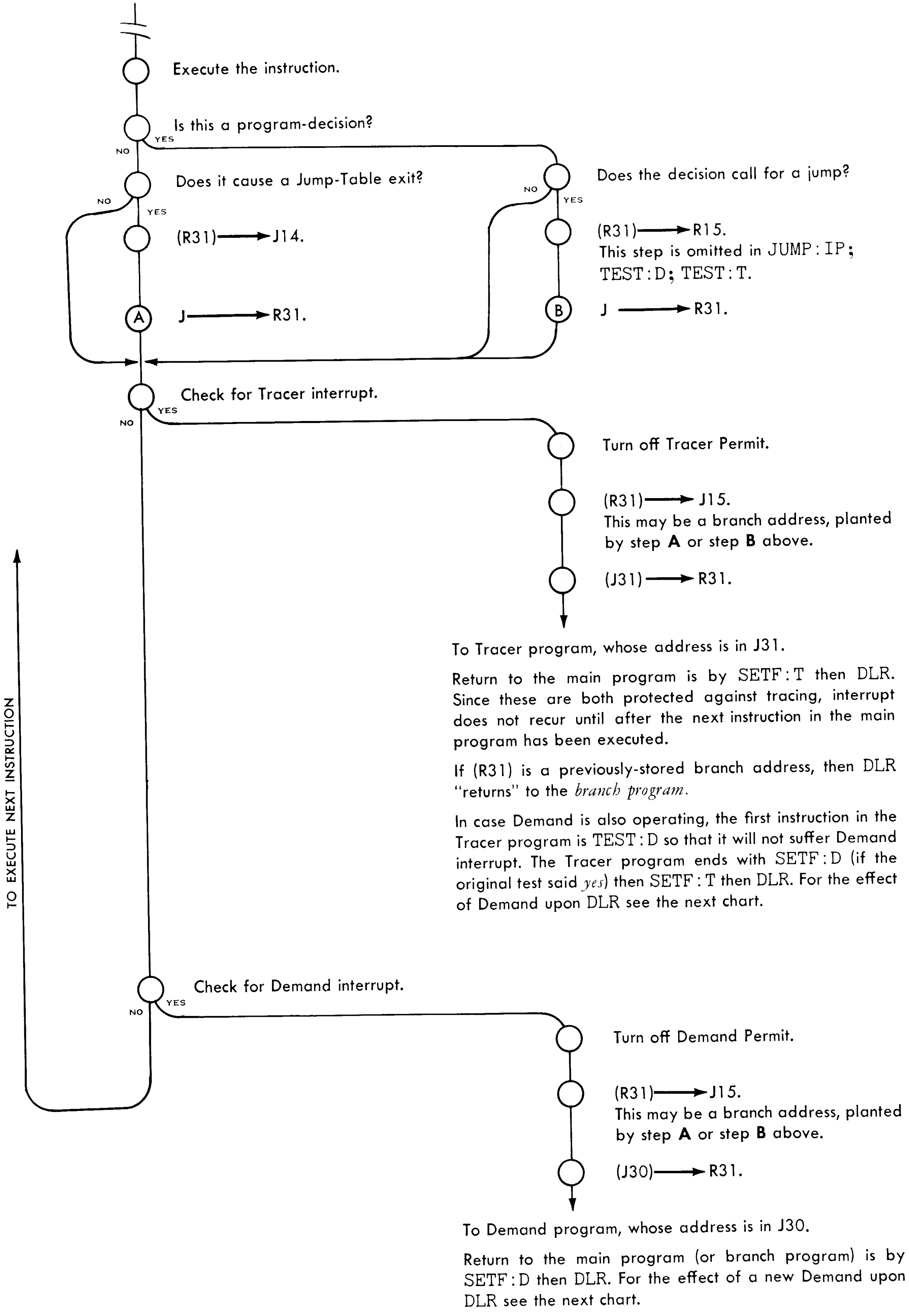
**J15** holds the link—the address of the next instruction in the interrupted program—for either Demand or Tracer interrupt.

**J14** holds the link if any instruction takes a jump table exit. The contents of R31 are saved in J14; then the contents of the designated one of the J-registers are planted into R31.

It occasionally happens that the programmer does not care whether an exit condition occurs or not during execution of an instruction. He can then suppress the exit by naming J14 as the jump register for that instruction. The sequence of events within the processor after detection of the exit condition then is:

- Plant the contents of R31 (the address of the next instruction in sequence) into J14.
- Plant the contents of the designated J-register (which is J14, and which now has the same contents as R31) into R31.
- The contents of R31 thus remain unchanged, and the processor takes the next instruction in sequence.

All these processes are illustrated on the following flow chart (an expansion of one section of the previous chart) which also shows the result if two, or all three, of these jumps are required simultaneously.



Execute the instruction.

Is this a program-decision?

Does it cause a Jump-Table exit?

(R31)→J14.

J→R31.

Does the decision call for a jump?

(R31)→R15.
This step is omitted in JUMP: IP; TEST: D; TEST: T.

J→R31.

Check for Tracer interrupt.

Turn off Tracer Permit.

(R31)→J15.
This may be a branch address, planted by step **A** or step **B** above.

(J31)→R31.

To Tracer program, whose address is in J31.

Return to the main program is by SETF: T then DLR. Since these are both protected against tracing, interrupt does not recur until after the next instruction in the main program has been executed.

If (R31) is a previously-stored branch address, then DLR "returns" to the *branch program.*

In case Demand is also operating, the first instruction in the Tracer program is TEST: D so that it will not suffer Demand interrupt. The Tracer program ends with SETF: D (if the original test said *yes*) then SETF: T then DLR. For the effect of Demand upon DLR see the next chart.

Check for Demand interrupt.

Turn off Demand Permit.

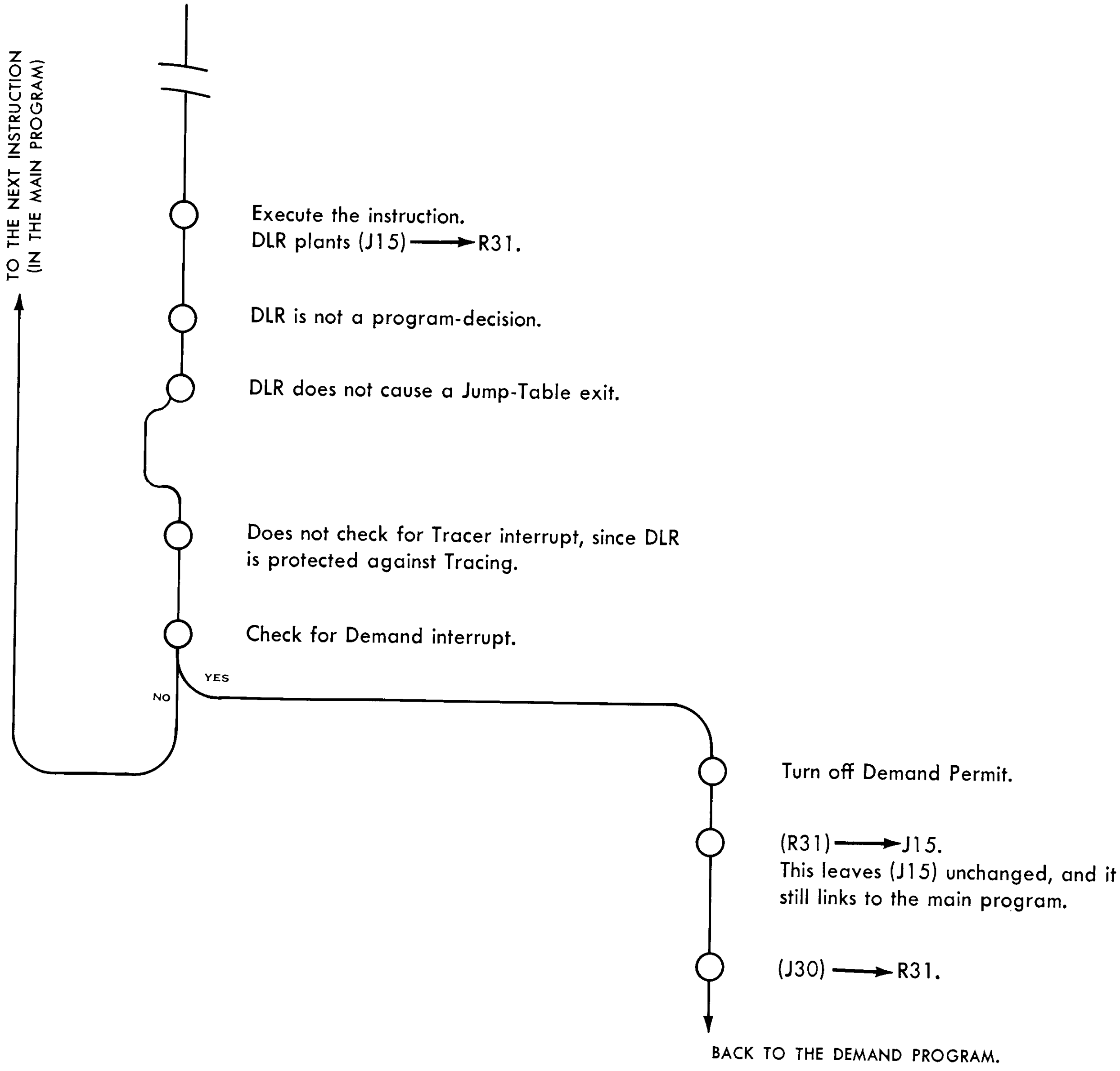
(R31)→J15.
This may be a branch address, planted by step **A** or step **B** above.

(J30)→R31.

To Demand program, whose address is in J30.

Return to the main program (or branch program) is by SETF: D then DLR. For the effect of a new Demand upon DLR see the next chart.

It is not immediately obvious that a Demand interrupt occurring at the conclusion of a DLR instruction will work correctly. It will, however, because DLR is an *operation* on the contents of R31 (thus imposing a jump) but is not a *jump instruction.* The following flow chart of part of the DLR instruction shows how it works:


TO THE NEXT INSTRUCTION
(IN THE MAIN PROGRAM)
Execute the instruction.
DLR plants (J15)⟶R31.
DLR is not a program-decision.
DLR does not cause a Jump-Table exit.
Does not check for Tracer interrupt, since DLR is protected against Tracing.
Check for Demand interrupt.
YES
NO
Turn off Demand Permit.
(R31)⟶J15.
This leaves (J15) unchanged, and it still links to the main program.
(J30)⟶R31.
BACK TO THE DEMAND PROGRAM.

## NAMING OF LITERALS:

Certain of the instructions permit naming index registers R15 and R31 *as though* they were to be used for address-modification, but when this is done, the processor accepts it as an indication to perform a special function.

Whenever R15 or R31 is named in the **X** column (in those instructions which permit it) then the processor interprets the contents of the **A** column not as the address of the data, but as the data itself. The **L** column is then not used.

Thus suppose it is desired to multiply the Accumulator contents by 17, and then to add 125 to the result. The instructions would be:

| Op | V | L | X | A |
|---|---|---|---|---|
| M U L T | | | 1 5 | 0 1 7 |
| A D D | | | 1 5 | 1 2 5 |

If one were writing programs in machine language, and wanted an Alpha literal, it would be necessary to write its Digit equivalent. However, the NEAT Assembly and the NEAT Compiler permit the programmer to designate a literal as either Digit or Alpha, and the Digit equivalent of an Alpha literal is produced automatically.

Those instructions, and variations, which permit naming a literal in this fashion are marked throughout this manual with an *.

## THE "EFFECTIVE LENGTH" OF THE ACCUMULATOR:

Although the actual length of the Accumulator is, and always remains, 8 slabs, the concept of its "effective length" is a useful one when discussing overflow conditions, and in the formulas for the execution times of the instructions.

The effective length of the Accumulator is simply the number of slabs, counting from the RH end, which contain all the non-zero information in the Accumulator. However, the effective length is never zero, so a cleared Accumulator has an effective length of one slab.

| ACCUMULATOR CONTENTS | | | | | | | | EFFECTIVE LENGTH |
|---|---|---|---|---|---|---|---|---|
| 0 0 0 | 0 0 0 | 0 0 1 | 0 0 0 | 0 0 0 | 0 0 0 | 3 1 5 | 0 0 0 | 6 |
| A B | C D | E F | G H | I J | K M | N Ø | P Q | 8 |
| 0 0 | 0 0 | 0 0 | 0 0 | 0 0 | 0 SP | N C | R SP | 3 |
| 0 0 0 | 0 0 0 | 0 0 0 | 0 0 0 | 0 0 0 | 0 0 0 | 0 0 0 | 0 0 0 | 1 |

# DEFINITIONS

LH Left-hand.

RH Right-hand.

@ The Accumulator.

(@) The contents of the Accumulator.

Op:V Operation and variation.

A The A-address named in an instruction. May range from 000 thru 999.
May not be negative.

(A) The contents of the A-word.

L The length of the word referred to in an instruction.

A-word The word referred to by the address A.
The address of the LH slab of the word is obtained by adding
A + contents of an index register.
The length of the word is L slabs.

RX The R-register specified by X.

RY The R-register specified by Y.

JX The J-register specified by X.

JY The J-register specified by Y.

G An augmenter named in an instruction. May range from —99 thru 999.

Alpha A 6-bit character.

Digit A 4-bit character.

slab The information stored in a single memory location.
A slab comprises 2 Alphas or 3 Digits.
The term "slab" is a contraction of "syllable": part of a word.

word A single unit of information, consisting of 1 to 8 slabs. A few operations permit longer words than this.

pair A 2-slab word containing an address or an augmenter in its RH 18 bit-positions. If the pair contains an augmenter, then it may be positive or negative.

N The number of times a sub-unit of an operation is to be performed.
(ie- Load N registers; Move N slabs; etc.)

J Jump address named in an instruction.

* Indicates an instruction which permits naming of a "literal" if desired.

** Indicates an instruction which requires naming of a "literal".

## *LOAD Accumulator

| Op | V | L | X | A |
|---|---|---|---|---|
| L D |  | L | X | A |

(**A**) replaces (@)

or "**A**" replaces (@)

This operation may be performed on either Digit or Alpha information.

The A-word is transcribed into the Accumulator, right-justified, and the Accumulator is filled out with zeros to the left.

SIGN FLAG: The processor does not "know" whether the programmer regards the A-word as made up of Digits or of Alphas. Therefore, if the left-hand four bits of the A-word are all 1-bits, the processor assumes a negative digit-word, sets the Sign flag negative, and replaces those four 1-bits in the Accumulator with four 0-bits.

Otherwise the Sign flag will be set positive.

OVERFLOW: Cannot occur.

## *STORE Accumulator

| Op | V | L | X | A |
|---|---|---|---|---|
| S T |  | L | X | A |

(@) replaces (**A**)

or (@) replaces "**A**"

This operation may be performed on either Digit or Alpha information.

The Accumulator is transcribed into the A-word right-justified. If the A-word is shorter than the effective length of the Accumulator, overflow occurs.

SIGN FLAG: If the Sign flag was previously set negative, and if the LH digit of the stored A-word is *zero,* then that digit is replaced by *hyphen.*

If the LH digit of the stored A-word is not *zero,* then there is no room for the negative sign, and overflow will occur.

In case of data overflow, the LH digit of the stored A-word might accidentally be *zero;* in this case a negative sign is not stored.

Setting of the Sign flag remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW:

1. The A-word is shorter than the effective length of the Accumulator.
2. Sign flag is negative, and there is not room to store the sign in the A-word, even though there is room for all the digits.

## *ADD to Accumulator

| Op | V | L | X | A |
|---|---|---|---|---|
| A D D |  | L | X | A |

(@) + (**A**) replaces (@)
or (@) + "**A**" replaces (@)

This operation may be performed only on Digit information.

If the LH digit of the A-word is *hyphen* then the word is considered negative and that digit is added as though it were a zero; otherwise the word is considered positive. The setting of the Sign flag designates the algebraic sign of the Accumulator.

The addition is performed according to the algebraic law of signs, and the sign of the result causes a new setting of the Sign flag.

NOTE: If any ADD operation yields a result of "zero", then the Sign flag remains unchanged.

SIGN FLAG: Designates the sign of the Accumulator;

Then is set by the sign of the result.

ACCUMULATOR: Holds the result of the operation.

OVERFLOW: If the result contains more than 24 digits.

The Accumulator will then hold the RH 24 digits of the result, and the Sign flag will be set correctly.

## *SUBTRACT from Accumulator

| Op | V | L | X | A |
|---|---|---|---|---|
| S U B |  | L | X | A |

(@) − (**A**) replaces (@)
or (@) − "**A**" replaces (@)

This operation may be performed only on Digit information.

If the LH digit of the A-word is *hyphen* then the word is considered negative and that digit is subtracted as though it were a zero; otherwise the word is considered positive. The setting of the Sign flag designates the algebraic sign of the Accumulator.

The Subtraction is performed according to the algebraic law of signs, and the sign of the result causes a new setting of the Sign flag.

NOTE: If any SUB operation yields a result of "zero", then the Sign flag remains unchanged.

SIGN FLAG: Designates the sign of the Accumulator;

Then is set by the sign of the result.

ACCUMULATOR: Holds the result of the operation.

OVERFLOW: If the result contains more than 24 digits.

The Accumulator will then hold the RH 24 digits of the result, and the Sign flag will be set correctly.

## * ADD to Memory

| Op | V | L | X | A |
|---|---|---|---|---|
| A D D | M | L | X | A |

(@) + (**A**) replaces (**A**)
or (@) + "**A**" replaces "**A**"

This operation may be performed only on Digit information.

If the LH digit of the A-word is *hyphen* then the word is considered negative and that digit is added as though it were zero; otherwise the word is considered positive. The setting of the Sign flag designates the algebraic sign of the Accumulator.

The addition is performed according to the algebraic law of signs, and the sign of the result is stored in the A-word with the result itself. However, the Sign flag is not changed by the result.

NOTE: If this operation yields a result of *zero* the original sign of the A-word remains unchanged.

SIGN FLAG: Designates the sign of the Accumulator;

Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW:

1. If the result contains more significant digits than the A-word can hold.
2. If the sign of the result is negative, and there is no room (see STORE) for the sign in the A-word. The sign is not stored.

## * COMPARE

| Op | V | L | X | A |
|---|---|---|---|---|
| C Ø M P |  | L | X | A |

(@) is compared with (**A**) } and G, L, E flag is
or (@) is compared with "**A**" } set accordingly.

This operation may be performed on either Digit or Alpha information.

The G-flag is set if (@) is Greater;
The L-flag is set if (@) is Less;
The E-flag is set if (@) is Equal.

With Digit information:

If the LH digit of the A-word is *hyphen* then the word is considered negative and that digit is compared as though it were a zero; otherwise the word is considered positive. The setting of the Sign flag designates the algebraic sign of the Accumulator.

The comparison is performed according to the algebraic law of signs, whereby any negative number is smaller than any positive number; and of two negative numbers, the larger magnitude is the smaller number.

If any non-decimal digits are present, the result can be predicted by giving the digits their binary values.

With Alpha information:

Since the operation does not distinguish between Digit and Alpha information (the actual comparison is bit-by bit) it is essential that the Sign flag be set positive before this operation is performed. In dealing with Alpha information, this will usually be the case anyway. However, note that if the LH character of the A-word is *apostrophe, left bracket, right bracket, reverse slant,* then the A-word will be considered negative.

NOTE: If the two numbers being compared are "positive zero" and "negative zero" then this operation sets the E-flag.

SIGN FLAG: Designates the sign of the Accumulator.

Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

## *DIVIDE Accumulator

| Op | V | L | X | A |
|---|---|---|---|---|
| D I V |  | L | X | A |

$\frac{(@)}{(\mathbf{A})}$ replaces LH (@)<br>
or $\frac{(@)}{``\mathbf{A}"}$ replaces LH (@)

Remainder replaces RH (@)

This operation may be performed only on Digit information.

The quotient appears, right-justified, in the LH 4 slabs of the Accumulator. The remainder appears, right-justified, in the RH 4 slabs of the Accumulator.

If the LH digit of the A-word is *hyphen* then the word is considered negative and the digit is treated as though it were a zero; otherwise the word is considered positive. The setting of the Sign flag designates the algebraic sign of the Accumulator.

The division is performed according to the algebraic law of signs, and the sign of the result causes a new setting of the Sign flag.

No sign is explicitly associated with the remainder; it is understood that the remainder has the same sign as the dividend (*initial* setting of the Sign flag).

After storing the result of the division, the quotient and remainder should thereafter be treated as separate words. The remainder word will appear to be positive, and the programmer must keep track of what its sign ought to be.

If for any reason the remainder alone is to be stored, observe the comments under STORE, with regard to overflow and storage of the Sign flag. Particular care must be taken at this point if there is a possibility that the quotient may be either zero or minus-zero.

SIGN FLAG: Designates the sign of the Accumulator;

Then is set by the sign of the Quotient.

ACCUMULATOR: Holds the dividend (numerator); Then receives the result of the operation.

OVERFLOW: No division is performed, and overflow occurs, in the following cases:

1. The A-word contains zero,
2. The A-word is more than 4 slabs long.
3. The effective length of the Accumulator is more than 4 slabs, *unless* the A-word contains a number of greater magnitude than the number in the LH 4 slabs of the actual Accumulator. This is the criterion for obtaining a quotient more than 4 slabs long, which is forbidden.

## *MULTIPLY ACCUMULATOR

| Op | V | L | X | A |
|---|---|---|---|---|
| M U L T |  | L | X | A |

(@) × (**A**) replaces (@)<br>
or (@) × "**A**" replaces (@)

This operation may be performed only on Digit information.

If the LH digit of the A-word is *hyphen* then the word is considered negative and that digit is multiplied as though it were a zero; otherwise the word is considered positive. The setting of the Sign flag designates the algebraic sign of the Accumulator.

The multiplication is performed according to the algebraic law of signs, and the sign of the result causes a new setting of the Sign flag.

SIGN FLAG: Designates the sign of the Accumulator;

Then is set by the sign of the result.

ACCUMULATOR: Holds the result of the operation.

OVERFLOW: If the sum of:
Length of the A-word and
Effective length of the
Accumulator.
is more than 8 slabs.

No multiplication will be performed, and the contents of the Accumulator will be unchanged.

## * *BINARY ADD to Accumulator*

| Op | V | L | X | A |
|---|---|---|---|---|
| B A D D | | L | X | A |

(@)+ (**A**) replaces (@) Addition is Mod-64 with no carry between Alpha positions.

or (@)+ "**A**" replaces (@) Addition is normal, with full carries.

SIGN FLAG: Ignored. Both operands considered positive.

Remains unchanged.

ACCUMULATOR: Holds result of the operation.

OVERFLOW: When adding (**A**), overflow cannot occur.

When adding "**A**", overflow will occur if the result exceeds the 96-bit capacity of the Accumulator, but this is unlikely.

ADDITION TABLE
FOR BINARY ADD

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | c0 |

"c" means carry

## *EDIT*

| Op | V | L | X | A |
|---|---|---|---|---|
| E D I T | | L | X | A |

Edit (**A**) into @ according to format-control previously in @.

This operation may be performed only on Alpha information.

The characters of the A-word are transcribed, one by one, from right to left, into the Accumulator, replacing certain of the characters previously in the Accumulator.

In order to do this, the characters in the Accumulator are scanned from right to left, until either *question* or *comma* is found.

If *question* Replace the *question* by the next (initially the rightmost) character of the A-word, and continue to scan the Accumulator.

If *comma* If the next character of the A-word is anything other than *asterisk* or *space* then leave the *comma* unchanged and continue to scan the Accumulator.

If the next character of the A-word is either *asterisk* or *space* then replace the *comma* with that character, but do not advance to the next character of the A-word.

The operation terminates when either the A-word or the Accumulator is exhausted.

SIGN FLAG: Remains unchanged.

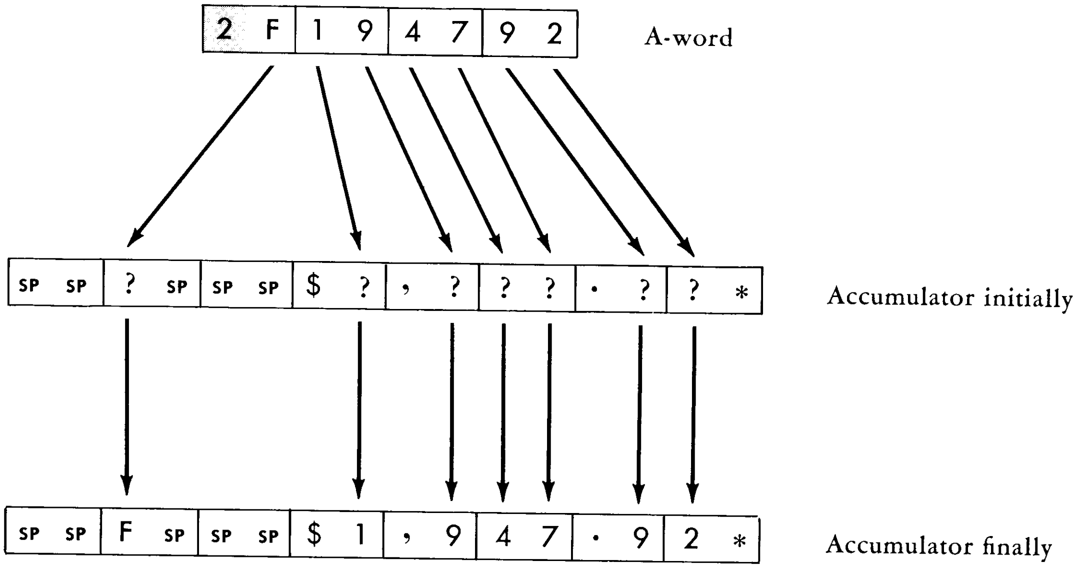
ACCUMULATOR: Holds the format-control pattern;

Then holds the edited A-word.

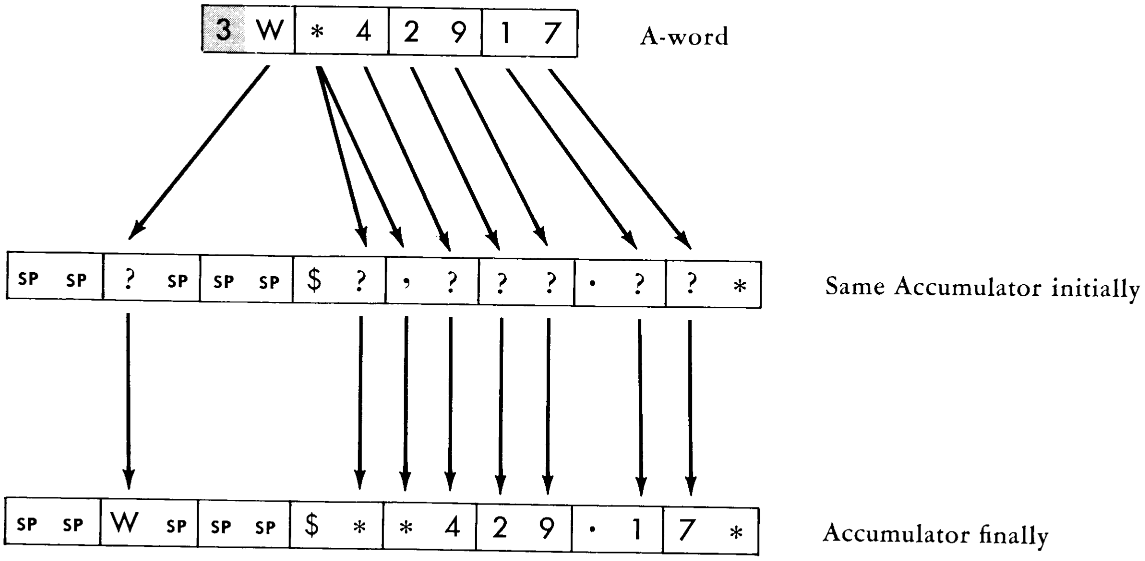
OVERFLOW: Cannot occur.

# EDIT

## Example 1:


2 F | 1 9 | 4 7 | 9 2
A-word
SP SP | ? SP | SP SP | $ ? | , ? | ? ? | . ? | ? *
Accumulator initially
SP SP | F SP | SP SP | $ 1 | , 9 | 4 7 | . 9 | 2 *
Accumulator finally

## Example 2:


3 W | * 4 | 2 9 | 1 7
A-word
SP SP | ? SP | SP SP | $ ? | , ? | ? ? | . ? | ? *
Same Accumulator initially
SP SP | W SP | SP SP | $ * | * 4 | 2 9 | . 1 | 7 *
Accumulator finally

## SUPPRESS

| Op | V | L | X | A |
|---|---|---|---|---|
| S U P P |  | L | X | A |

Leading zeros in the A-word are replaced with spaces.

This operation may be performed only on Alpha information.

The A-word is scanned from left to right. If the first character is a *zero* it is replaced by a *space* and the scanning proceeds to the next character; and so on.

The operation terminates when either:

1. The scanning process encounters any character other than *zero.*
2. The A-word has been exhausted, and now contains all spaces.

After the operation terminates:

The Accumulator contains the number of slabs *in which* suppression has occurred.

If an odd number of zeros were suppressed, the G-flag is turned on.

SIGN FLAG: Always set positive.

ACCUMULATOR: Holds tally of number of slabs now containing spaces.

G-FLAG: ON if odd number of spaces.
OFF if even number of spaces.

L-FLAG; E-FLAG: Always turned OFF.

OVERFLOW: Cannot occur.

## SET and CLEAR Processor Flags

FORMAT A: All but Memory Flags

| Op | V | L | X | A |
|---|---|---|---|---|
| S E T F | (V) |  |  |  |

FORMAT B: Memory Flags

| Op | V | L | X | A |
|---|---|---|---|---|
| S E T F | (V) |  | X | A |

| Op | V | L | X | A |
|---|---|---|---|---|
| C L R F | (V) |  | X | A |

|  |  |  |
|---|---|---|
| **SETF:+ | Set Sign flag plus. |  |
| **SETF:– | Set Sign flag minus. |  |
| **SETF:ϑ | Set Overflow flag on. |  |
| **SETF:D | Set Demand Permit flag on. | Protected |
| **SETF:T | Set Tracer Permit flag on. |  |
| *SETF:LH | Set LH Memory flag on, in 1-slab A-word. | The character *space* is inserted |
| *SETF:RH | Set RH Memory flag on, in 1-slab A-word. |  |
| *CLRF:LH | Set LH Memory flag off, in 1-slab A-word. | The character *zero* is inserted |
| *CLRF:RH | Set RH Memory flag off, in 1-slab A-word. |  |

SIGN FLAG: Remains unchanged, except by SETF: + and SETF: –.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur, except as result of SETF:ϑ

### *TEST (single-stage)*

| Op | V | L | X | A |
|---|---|---|---|---|
| T E S T | (V) | | X | J |

Each of these may cause a jump to the instruction whose address is: "J" + (contents of RX)

TEST:G Jump if G-flag is on.
Set Link in R15

TEST:L Jump if L-flag is on.
Set Link in R15

TEST:E Jump if E-flag is on.
Set Link in R15

TEST:– Jump if Sign flag negative.
Set Link in R15

Flag not disturbed

TEST:θ Jump if Overflow flag is on.
Set Link in R15

Protected

TEST:D Jump if Demand Permit flag is on..
Does not link.

TEST:T Jump if Tracer Permit flag is on.
Does not link.

Flag turned off

The programmer will use TEST:D and TEST:T, with jumps suppressed, to turn off the Demand and Tracer flags. Aside from that, these instructions are used only for housekeeping purposes in the "canned" portions of the Tracer and Demand programs.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

NOTE: When testing the G, L, E flags after SCAN, it is more convenient to use the following alternative mnemonics.

TEST:SL Did SCAN stop on left digit or alpha?
Same command as TEST:E

TEST:SM Did SCAN stop on middle digit?
Same command as TEST:L

TEST:SR Did SCAN stop on right digit or alpha?
Same command as TEST:G

### * *TEST (double-stage)*

FORMAT A:
Test Memory flags.
Jump to address named in instruction.
Link in R15.

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| T E S T | (V) | | X | A |
| | | | JUMP ADDRESS (ALL 5 DIGITS) | |

FORMAT B:
Test Console Switches.
Jump to address "J" + (contents of RY).
Link in R15.

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| T E S T | S W | | X | A |
| | | | Y | J |

*TEST:LH Jump if LH flag in 1-slab A-word is on (not equal to zero).

*TEST:RH Jump if RH flag in 1-slab A-word is on (not equal to zero).

*TEST:SW Jump if Console Option Switch number (**A**) or "**A**" is on.
Switch number may be 000-007.

Protected against Demand

SIGN FLAG: Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

LH, RH FLAGS: Remain unchanged.

SWITCHES: Remain unchanged. They are set by the console operator.

### *JUMP*
### *JUMP INDIRECT*
### *JUMP INDIRECT, keep PREVIOUS Link*

| Op | V | L | X | A |
|---|---|---|---|---|
| J U M P | | | X | J |

| Op | V | L | X | A |
|---|---|---|---|---|
| J U M P | I | | X | A |

| Op | V | L | X | A |
|---|---|---|---|---|
| J U M P | I P | | X | A |

JUMP: Unconditional jump to address "J" + (contents of RX). Link in R15. (R31) replaces (R15). Then Jump address replaces (R31).

JUMP:I Unconditional jump to address stored in 2-slab A-word. Link in R15. (R31) replaces (R15). Then A-word replaces (R31).

JUMP:IP Unconditional jump to address stored in 2-slab A-word. **Does not link.** A-word replaces (R31).

SIGN FLAG Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW Cannot occur.

### ***SKIP within the program*

| Op | V | L | X | A |
|---|---|---|---|---|
| S K I P | | | | G |

This instruction causes an unconditional jump to a point up to 999 slabs after, or up to 99 slabs before, the next instruction in the normal sequence.

(R31) + "**G**" replaces (R31).
**G** may range to 999 or to −99.

This is an *operation* upon (R31), and is not a *jump instruction.* Therefore it does not set any link.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

## **DEMAND LINK RETURN

| Op | V | L | X | A |
|---|---|---|---|---|
| D L R | | | | |

This instruction causes a return to the Demand Link.

(J15) replaces (R31).

This is an *operation* upon (R31), and is not a *jump instruction.* Therefore it does not set any link.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

NOTE: This instruction is protected against Tracing. It is not protected against Demand.

## **MAIN LINK RETURN, and AUGMENT Control Register

| Op | V | L | X | A |
|---|---|---|---|---|
| M L R A | | | | G |

This instruction causes an unconditional jump back to the Main Link, with the option of skipping instead to a point up to 999 slabs after the link, or up to 99 slabs before the link.

(R15) + "**G**" replaces (R31).
**G** may range to 999 or −99.

This is an *operation* upon (R31), and is not a *jump instruction.* Therefore it does not set any link.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

## * SHIFT Accumulator

| Op | V | L | X | A |
|---|---|---|---|---|
| S H F T | (V) | | X | A |

This operation may be performed on either Digit or Alpha information, as specified by the Variation.

The contents of the Accumulator are shifted N places. N is equal to: The contents of the A-word (always one slab long); or "**A**" itself.

Except for SHFT:LC and SHFT:RC all shifts are linear, ie- "off the end".

*SHFT:DL Shift digits left,
enter zeros at the right.

*SHFT:DR Shift digits right,
enter zeros at the left.

*SHFT:RR Shift digits right and roundoff,
enter zeros at the left.

Before the shift, 5 is lined up with the Nth character from the right, and added to the Accumulator contents.

*SHFT:LC Shift digits left circular.†

*SHFT:RC Shift digits right circular.†

The two circular shifts operate within the *effective length* of the Accumulator. They leave the effective length unchanged even though the circulation may bring zeros into the leftmost positions.

*SHFT:AL Shift alphas left,
enter spaces at the right.

If, before the shift, there were zeros at the RH end of the Accumulator, these are shifted unchanged just like any other characters; they *are not replaced* by spaces.

*SHFT:AR Shift alphas right,
enter zeros at the left.

**NOTE**: During an Alpha shift, all zeros within the previous effective length of the Accumulator remain significant to the new effective length.

Before a left or right Alpha shift of an odd number of positions, a slab containing **0 SP** is inserted in the Accumulator, just to the left of the previous effective length.

After an Alpha Right shift equal to or more than the previous effective length, the Accumulator will contain the single slab **SP SP**.

SIGN FLAG: Remains unchanged.

OVERFLOW: Cannot occur.

If N = 0, these operations do nothing.

## * COUNT

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| C N T | | | X | A |
| | | | Y | G |

This instruction performs two distinct operations.

1. First ADD: (RY) + "**G**"
replaces (RY)
**G** ranges to 999 or −99.

2. Then COMPARE: (RY) vs (**A**) completely
or (RY) vs "**A**" mod 1000
and set G, L, E flag.

If the LH digit of "**G**" is the character *hyphen* then "**G**" is considered negative; otherwise "**G**" is considered positive. The contents of an Index Register are always positive.

If an A-word in Memory is named, it is always a "pair": a 2-slab word containing a number no greater than 39 999.

If "**A**" itself is used for the comparison, then only the RH 3 digits of the Index Register are used in the comparison.

SIGN FLAG: Remains unchanged.

G-FLAG: ON if (RY) is greater.
L-FLAG: ON if (RY) is less.
E-FLAG: ON if (RY) is equal.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

NOTE: Addition is always performed modulo 40 000 regardless of the memory size of the Processor.

†Circular Shift and the Accumulator Length

If the effective length of the Accumulator is more than one slab, a circular shift that brings one or more slabs of zeros to the left will leave the Accumulator length "unstable." This means that, if the Accumulator is stored and then reloaded, its effective length will have been decreased by the number of zero slabs on the left, and another circular shift will not now behave the way it would have behaved if the Accumulator had been undisturbed.

The only significance of this condition is the possible effect upon a circular shift following a circular shift.

The Demand macros, Trace, etc., do not check for this condition. If Demand interrupt might occur while the Accumulator length is unstable, or if Tracing is to be performed, the program should be protected against Demand or Tracer while the condition exists. Since the condition is a rare one, it will not be mentioned in any published discussions of Demand, Trace, or service routines, and this present discussion will serve as notice to anyone using circular shift.

### LOAD Registers
### SPREAD-LOAD Registers

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| L D | (V) | | X | A |
| | | | Y | N |

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| S L D | (V) | | X | A |
| | | | Y | N |

LD :R Transcribe N successive Memory
pairs (A), (A+2), etc.
into N successive R-registers
starting with RY.

LD :J Transcribe N successive Memory
pairs (A), (A+2), etc.
into N successive J-registers
starting with JY.

SLD :R Transcribe one Memory pair (A)
into each of N successive
R-registers starting with RY.

SLD :J Transcribe one Memory pair (A)
into each of N successive
J-registers starting with JY.

If any of these operations remains incomplete after referring to R31 or J31, there will be an error halt.

If any of these operations carries past the end of memory, it will cycle back to location 00 000 and continue.

NOTE: Only the RH 18 bits (4½ digits) of each pair are loaded. The LH 6 bits (1½ digits) are irrelevant.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

If N = 0, these operations do nothing.

### STORE Registers

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| S T | (V) | | X | A |
| | | | Y | N |

ST:R Transcribe N successive R-registers
starting with RY
into N successive Memory
pairs (A), (A+2,) etc.

ST:J Transcribe N successive J-registers
starting with JY
into N successive Memory
pairs (A), (A+2), etc.

If either of these operations remains incomplete after referring to R31 or J31, there will be an error halt.

If either of these operations carries past the end of memory, it will cycle back to location 00 000 and continue.

NOTE: The contents of a register are stored as the RH 18 bits (4½ digits) of the Memory pair. The LH 6 bits of the pair are set to zero.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

If N = 0, these operations do nothing.

## *AUGMENT Registers*
## *SPREAD-AUGMENT Registers*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| A U G | (V) | | X | A |
| | | | Y | N |

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| S A U G | (V) | | X | A |
| | | | Y | N |

AUG :R The contents of each of N successive R-registers starting with RY is augmented by the contents of the corresponding one of N successive Memory pairs (A), (A+2), etc.

AUG :J The contents of each of N successive J-registers starting with JY is augmented by the contents of the corresponding one of N successive Memory pairs (A), (A+2), etc.

SAUG:R The contents of each of N successive R-registers starting with RY is augmented by the contents of Memory pair (A).

SAUG:J The contents of each of N successive J-registers starting with JY is augmented by the contents of Memory pair (A).

Only the RH bits (4½ digits) of a Memory pair are used in augmenting a register.

If the LH digit of any Memory pair is *hyphen*, then the contents of that pair is a negative number and the contents of the register are diminished. The contents of the register are always positive.

Addition is performed modulo 40 000 regardless of memory size.

If any of these operations remains incomplete after referring to R31 or J31, there will be an error halt.

If any of these operations carries past the end of memory, it will cycle back to location 00 000 and continue.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

If N = 0, these operations do nothing.

### Examples

RESULT OF

| AUG | R | | 01 | 711 |
|---|---|---|---|---|
| | | | 13 | 006 |

RESULT OF

| SAUG | R | | 01 | 711 |
|---|---|---|---|---|
| | | | 13 | 006 |

| | REGISTERS INITIALLY | | MEMORY PAIRS | | RESULT OF AUG | | RESULT OF SAUG |
|---|---|---|---|---|---|---|---|
| R13 | 0 0 0 4 7 | 01711 | 0 0 0 1 0 0 | R13 | 0 0 1 4 7 | R13 | 0 0 1 4 7 |
| R14 | 2 7 6 3 5 | 01713 | 0 0 0 2 0 0 | R14 | 2 7 8 3 5 | R14 | 2 7 7 3 5 |
| R15 | 3 9 9 9 9 | 01715 | - 0 0 0 0 2 | R15 | 3 9 9 9 7 | R15 | 0 0 0 9 9 |
| R16 | 1 5 0 0 0 | 01717 | 0 3 0 0 0 0 | R16 | 0 5 0 0 0 | R16 | 1 5 1 0 0 |
| R17 | 0 4 2 9 6 | 01719 | 0 0 0 0 0 1 | R17 | 0 4 2 9 7 | R17 | 0 4 3 9 6 |
| R18 | 0 2 5 0 0 | 01721 | - 7 0 0 0 0 | R18 | 1 2 5 0 0 | R18 | 0 2 6 0 0 |

### *MOVE information between Registers*

**Move from Y to X**

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| M Ø V E | (V) | | X | |
| | | | Y | N |

MØVE:RR Transcribe N successive R-registers
starting with RY
into N successive R-registers
starting with RX.

MØVE:JR Transcribe N successive J-registers
starting with JY
into N successive R-registers
starting with RX.

MØVE:RJ Transcribe N successive R-registers
starting with RY
into N successive J-registers
starting with JX.

MØVE:JJ Transcribe N successive J-registers
starting with JY
into N successive J-registers
starting with JX.

If any of these operations remains incomplete after referring to R31 or J31, there will be an error halt.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: Remains unchanged.

OVERFLOW: Cannot occur.

If N = 0, these operations do nothing.

### *MOVE memory* } { *Start at Beginning* } up to
### ** *SPREAD memory* } { *Start at End* } 999 slabs

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| M Ø V E | (V) | | X | A |
| | | | Y | B |

Y specifies Index Register RY, modifying address B.

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| S P R D | (V) | | | A |
| | | | Y | B |

Y specifies Index Register RY, modifying address B.

These operations may by performed on either Digit or Alpha information.

***MOVE*** transcribes N (up to 999) consecutive slabs from an A-area to a B-area in Memory.

N is in the RH slab of the Accumulator.

MOVE:B Start at the beginning of each area:
(A) replaces (B)
then (A+1) replaces (B+1) etc.

MOVE:E Start at the end of each area:
(A) replaces (B)
then (A−1) replaces (B−1) etc.

***SPREAD*** transcribes "A" itself into each slab of an N-slab B-area in Memory.

**SPRD:B "A" replaces (B)
then "A" replaces (B+1) etc.

**SPRD:E "A" replaces (B)
then "A" replaces (B−1) etc.

If any of these operations carries past the end of Memory, it will cycle back to location 00 000 and continue.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: RH slab holds N: the number of slabs to be transcribed (up to 999).

Remains unchanged.

OVERFLOW: Cannot occur.

If N = 0, these operations do nothing.

## SCAN

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| S C N D | (V) | | X | A |
| | | | Y | L |

L is length of A-word: up to 999
Y specifies Jump Register JY

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| S C N A | (V) | | X | A |
| | | | Y | L |

L is length of A-word: up to 999
Y specifies Jump Register JY

SCND: Scan Digits.

SCNA: Scan Alphas.

This operation may be performed on either Digit or Alpha information, as specified by the Variation.

Consider the Accumulator as being of the same length as the A-word, which in this operation may be up to 999 slabs long. Consider that every slab in the Accumulator exactly duplicates the RH slab.

The Accumulator and the A-word are then simultaneously scanned, from *left* to *right*, and compared Digit by Digit, or Alpha by Alpha, until one character in the A-word is found which bears the specified relationship to the corresponding character in the Accumulator. The relationship may be specified as Greater than, Less than, or Equal to, the corresponding character in the Accumulator.

The LH character of the Variation specifies whether the SCAN shall seek a character in the A-word which is:

G: greater than
L: less than
E: equal to

the corresponding character in the Accumulator.

The operation terminates when either:

1. A character in the A-word meets the test. R30 then contains the address of the slab in which this character appears.

   Flags are set to indicate the position of this character in that slab.

   | | |
   |---|---|
   | RH Digit or RH Alpha | G-flag set |
   | Middle Digit | L-flag set |
   | LH Digit or LH Alpha | E-flag set |

   The Processor proceeds to the next instruction in normal sequence.

   NOTE: It may be inconvenient to remember the correspondence of G, L, E flags to right, middle, left positions within the slab. Therefore three alternative mnemonics are provided.

   TEST:SL Did SCAN stop on left digit or alpha?
   Same command as TEST:E

   TEST:SM Did SCAN stop on middle digit?
   Same command as TEST:L

   TEST:SR Did SCAN stop on right digit or alpha?
   Same command as TEST:G

2. No character in the A-word meets the test.
   R30 remains unchanged.
   G, L, E flags are all turned off.
   The Processor takes its next instruction from the address in JY. Link in J14.

If L = 0 the Processor immediately takes termination 2.

SIGN FLAG: Remains unchanged.

ACCUMULATOR: RH slab contains the Scan Key, which is *considered* to be duplicated in every slab of the Accumulator.

The entire Accumulator remains unchanged by the SCAN operation.

R30: After termination 1, R30 holds address of slab containing the successful character of the A-word. After termination 2, R30 remains unchanged.

G, L, E FLAGS: After termination 1, these indicate the position of the successful character within its own slab. After termination 2, all these flags are off.

JY: After termination 2, Processor jumps to address stored in JY.

OVERFLOW: Cannot occur.

SELECTIVE SCANNING:

As the characters in memory are compared with the characters in the RH slab of the Accumulator, it is not necessary that *all* the Digits or Alphas of each slab be examined. In order to specify which positions are actually to be scanned, the positions in a slab are given "scan-values":

| | | | |
|---|---|---|---|
| Digits | 4 | 2 | 1 |
| Alphas | 2 | | 1 |

In the RH character of the variation, the programmer writes the sum of the scan-values of those positions which he wishes the SCAN to examine.

For example—

SCND:G3 means Scan Digits for Greater, examining only the RH and middle digits of each slab (ignoring the LH digit).

SCND:E6 means Scan Digits for Equal, examining only the LH and middle digits of each slab (ignoring the RH digit).

SCNA:L2 means Scan Alphas for Less, examining only the LH alpha of each slab (ignoring the RH alpha).

A "blank" as the RH character of the variation means that all characters are significant. Thus:

SCND:G means the same as SCND:G7

SCNA:E means the same as SCNA:E3

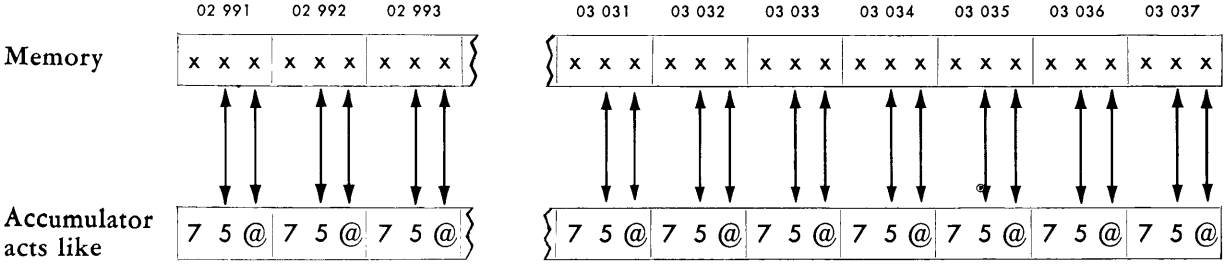
As an example, consider the instruction:

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| SCND | L3 | | 02 | 991 |
| | | | 12 | 047 |

When the Accumulator contains:

| | | |
|---|---|---|
| 1 2 3 | 4 5 6 | 7 5 @ |

The Processor behaves (for the duration of this instruction) *as though* the Accumulator were 47 slabs long, corresponding to the 47-slab length of the memory word to be scanned.



If the memory word contains—

a digit from 0-4 in the middle position of any slab, or

a digit from 0-9 in the right-hand position of any slab,

the Scan will indicate the position of the *first* such digit, in R30 and in the Scan-middle or the Scan-right flag. The Processor will then execute the next instruction in the normal sequence.

If the memory word contains no such digit in either of these positions, the Processor will clear the G, L, E flags, leave (R30) unchanged, and take its next instruction from the address stored in J12.

In either case, the original 3-slab Accumulator contents remain unchanged.

## PARTIAL ALPHA STORE

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| P A S T | (V) | | X | A |
| | | | | L |

L is length of A-word: up to 8 (or 9)

This operation may be performed only on Alpha information.

The Accumulator is stored, right-justified, in a part of the A-word.

If L = 0, this operation does nothing.

PAST:XL Except the LH character of the A-word.

PAST:XR Except the RH character of the A-word.

PAST:XB Except both.

SIGN FLAG: Remains unchanged. The Sign flag is not stored in the A-word.

ACCUMULATOR: Remains unchanged.

OVERFLOW: XL: L greater than 8; 16 characters stored in A-word.
XR: L greater than 8; 16 characters, plus a zero, stored in A-word.
XB: L greater than 9; 16 characters, plus a zero, stored in A-word.

**Examples:** Shaded areas are unchanged.

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 0 | A B | C D | E F | Contents of Accumulator |
| | | | ▨ B | C D | E F | 3-slab A-word after PAST:XL |
| ▨ 0 | 0 0 | A B | C D | E F | 5-slab A-word after PAST:XL |
| | | B C | D E | F ▨ | 3-slab A-word after PAST:XR |
| 0 0 | 0 A | B C | D E | F ▨ | 5-slab A-word after PAST:XR |
| | | ▨ C | D E | F ▨ | 3-slab A-word after PAST:XB |
| ▨ 0 | 0 A | B C | D E | F ▨ | 5-slab A-word after PAST:XB |

## *LOAD ALPHA-TO-DIGIT*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| L D A D | (V) | | X | A |
| | | | Y | L |

L is length of A-word: up to 12 (or 13)
Y specifies Jump Register JY

This operation may be performed only on Alpha information in Memory, which then becomes Digits in the Accumulator.

The A-word is Loaded into the Accumulator, right-justified, with its Alphas transformed into Digits.

The Alphas of the A-word are transcribed, from right to left, into the Accumulator. As each character is transcribed, it is stripped of its zone bits, and stored in the Accumulator as a 4-bit Digit.

The operation terminates when either:

1. The A-word is exhausted; the Accumulator is filled out to the left with zeros.
2. The Accumulator is filled, and the A-word is not exhausted. Overflow then occurs.

If any of the discarded zone bits of the A-word are 1-bits, the operation still proceeds to completion; then the Processor takes its next instruction from the address in JY, and sets link in J14. Otherwise the Processor proceeds in sequence.

If L = 0, this operation clears the Accumulator.

LDAD: Load and condense the entire A-word.

LDAD:XL Except the LH character of the A-word.

LDAD:XR Except the RH character of the A-word.

LDAD:XB Except both. L may be equal to 13.

SIGN FLAG: Set positive by this operation.

ACCUMULATOR: Contains the condensed A-word.

JY: Processor jumps to address stored in JY if any 1-bits in zones. If desired, this jump can be suppressed by naming J14 as JY.

OVERFLOW: If the A-word is too long.

### Example 1:

| | | | | |
|---|---|---|---|---|
| 1 2 | 3 4 | 5 F | P X | A-word |
| 0 0 0 0 | 0 1 2 | 3 4 5 | 6 7 7 | Accumulator after LDAD<br>Processor jumps to address in JY. |

### Example 2:

| | | | | | |
|---|---|---|---|---|---|
| | 1 2 | 3 4 | 5 6 | 7 8 | A-word |
| 0 | 0 0 0 | 0 1 2 | 3 4 5 | 6 7 8 | Accumulator after LDAD |
| 0 | 0 0 0 | 0 0 2 | 3 4 5 | 6 7 8 | Accumulator after LDAD:XL |
| 0 | 0 0 0 | 0 0 1 | 2 3 4 | 5 6 7 | Accumulator after LDAD:XR |
| 0 | 0 0 0 | 0 0 0 | 2 3 4 | 5 6 7 | Accumulator after LDAD:XB |

## STORE DIGIT-TO-ALPHA

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| S T D A | | | X | A |
| | | | | L |

L is length of A-word: up to 12

This operation may be performed only on Digit information in the Accumulator, which then becomes Alphas in Memory.

The Accumulator is stored in the A-word, right-justified, with its Digits transformed into Alphas.

The digits of the Accumulator are transcribed, from right to left, into the A-word. As each digit is transcribed, a pair of 0-bit zone bits are attached to it, and it is stored in the A-word as a 6-bit Alpha.

The operation terminates when either:

1. The A-word is filled. If any non-zero digits of the Accumulator cannot fit into the A-word, overflow occurs.
2. The A-word has received 24 characters. If the A-word is more than 12 slabs long, overflow occurs.

If L = 0, this operation does nothing.

SIGN FLAG: Remains unchanged.

The Sign flag is not stored in the A-word.

ACCUMULATOR: Remains unchanged.

OVERFLOW:

1. If the A-word is too small for all the non-zero characters in the Accumulator.
2. If the A-word is more than 12 slabs long.

# INPUT, OUTPUT, FILE DEVICES

# INPUT, OUTPUT, FILE DEVICES

## INTRODUCTION

Processor communication with the "outside world" is performed in Alpha form (6 bits per character, 2 characters per slab) except for the Console Typewriter which accepts either Alphas or Digits for input-output. File information is recorded on magnetic tape and CRAM in "image alpha" form, whether it is actually Alpha or Digit.

Remember Except as specifically noted, none of these operations change the Sign flag, the Overflow flag, or the contents of the Accumulator.

The operations for Inquiry Units and for Magnetic Character Sorter-Readers are the subject of another publication, and are not discussed here.

## PERIPHERAL UNITS

### CONSOLE TYPEWRITER:

The Console includes an electric typewriter which permits modest amounts of information to be typed out by the program, or to be entered by the operator, in either Digit or Alpha form.

During normal operations the Console Typewriter will be used to create a Log of operations, and for occasional input of information such as Today's Date, etc.

### CARD READER:

Two models of Card Reader are available, one operating at 400 cards per minute, the other at 2000 cards per minute. Both are serial readers, reading one card column at a time photoelectrically.

The 400 cpm reader stores an image of each column as a pattern of 12 bits in one slab of memory. There is sufficient time between columns for the processor program to translate the column images into the actual characters. Any kind of non-standard punching or binary punching may be read, by storing appropriate tables in memory.

The 2000 cpm reader has automatic translation circuitry within it, and stores the actual characters in memory after checking for invalid configurations. The automatic translation may be by-passed if desired, to read non-standard or binary punching with programmed translation; the reader can switch from translate to non-translate and back any number of times within a card.

Both readers leave the processor free between columns, during the unread portion of each card (if less than the entire card is read), and between cards. They both exercise Demand as each new card reaches the reading station.

### PAPER TAPE READER:

The Paper Tape Reader operates at 1000 characters per second, and will stop between characters at any time, under control of the processor program. It stores an image of each row of punching as a binary configuration in memory, and there is sufficient time between characters for the program to translate these images into the actual characters, and to do validity-checking.

Any code system whatever, up to 8 channels, will be accepted by the reader, and can be translated by program, with appropriate tables.

### BUFFERED LINE PRINTERS:

The processor can operate up to four printers or card punches, in any combination. Each printer is completely buffered, and performs all printing and paper movement under its own control, with automatic end-of-page detection. It has a 56-character font, and prints a 120-character line.

The printer has a 2-speed switch for printing at 680 alphanumeric (940 numeric) lines per minute, or at 380 alphanumeric (470 numeric) lines per minute. The lower speed provides superior print quality for photo-offset reproduction. The higher printing rates (at both speeds) for numeric information arise automatically out of the characteristics of the printer itself, and are obtained whenever numeric lines are printed; no special programming or special adjustment of the printer is required.

When printing at either speed, paper movement over blank lines is at the rate of 15 inches per second (5400 lines per minute) regardless of the number of blank lines.

After each line has been printed, the printer is READY and can exercise Demand, so that the processor program may immediately transmit the information for another line.

## CARD PUNCHES:

The processor can operate up to four card punches or printers, in any combination. The punches may be: IBM 523-1 or 523-2 (100 cards per minute); or IBM 7550 (250 cards per minute). Each punch is completely buffered, and performs all punching, and all plugboard-programmed editing and checking, under its own control.

Each Card Punch Buffer automatically translates the characters from the processor memory into standard card code for punching. There is a switch on the buffer, permitting the translation to be by-passed, in which case each column of the card is punched as an image of the bit-configuration in one slab of memory. This feature permits the output of cards with any kind of non-standard or binary punching.

When each card is completely punched, the punch is READY and can exercise Demand, so that the processor program may immediately transmit the information for another card.

## PAPER TAPE PUNCH:

The paper tape punch operates at 110 characters per second, and can punch any code system whatever, up to 8 channels. The time between characters is available to the processor program for translation of the characters into the required hole-configurations, and for any other processing work.

## MAGNETIC TAPE HANDLERS:

The processor may accommodate up to 8 handlers, each holding a reel of tape. The tape moves at 120 inches per second, and three recording densities are available:

200 alphanumeric characters per inch
24,000 characters per second (24 kc)
Compatible with IBM tapes.

333 alphanumeric characters per inch
40,000 characters per second (40 kc)
60,000 digits per second

500 alphanumeric characters per inch
60,000 characters per second (60 kc)
90,000 digits per second

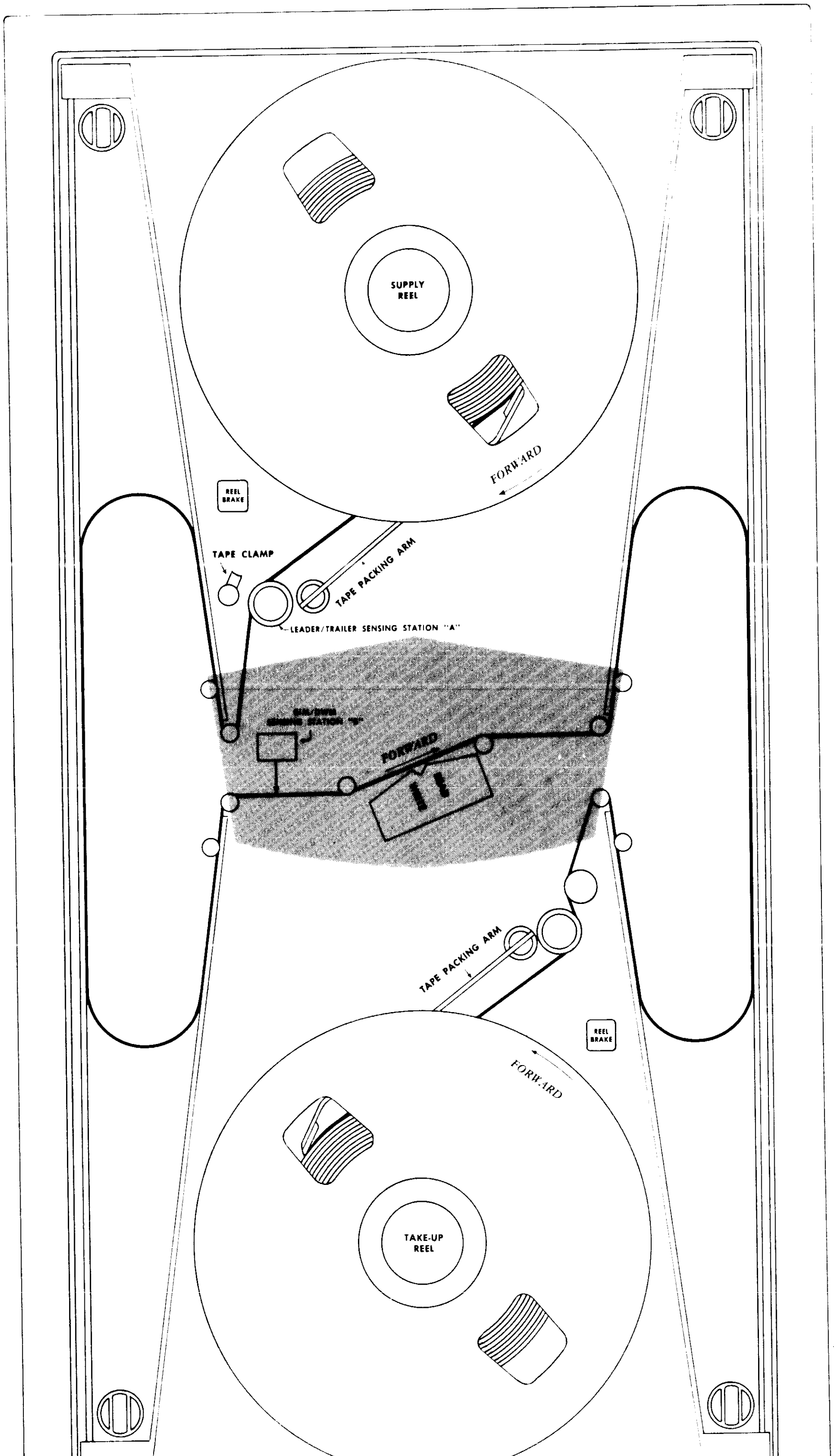
The Standard Handler can read and write tape at 24 and 40 kc. The High Speed Handler can read and write at all three speeds. Each handler has a switch to determine the density of writing; while reading, this switch must correspond to the density at which the tape was recorded.

Information is recorded on magnetic tape in "image alpha" form. Images of the left-hand and right-hand six bits of each slab are recorded as alpha characters on the tape. If the slab contains alpha information, these will be the actual characters; if the slab contains digit information, these will be pseudo-characters, each comprising 1½ digits, which will recreate the image of the slab in memory when the tape is read. Therefore three digits take the same amount of tape, and the same read-write time, as two alphas, so that information transfer rates for numeric information are 1½ times those for alpha-numeric information.

An information block on tape may contain 1 to 7,999 slabs (2 to 15,998 alphanumeric characters, 3 to 23,997 digits or any proportional mixture). Between blocks there is a three-quarter-inch gap (length of blank tape). It is clearly advantageous to *block* records—to put as many accounts as possible into a single block—to minimize the number of gaps. The large number of index registers in the NCR 315, and their flexibility in use, make this procedure extremely convenient, whether the blocks are fixed or variable in length.

As the tape is being written, the processor automatically adds a parity-bit to each character, and records 7 bit-channels on the tape. Even-parity is established for 24-kc tapes, odd-parity for 40-kc and 60-kc tapes. At the end of the block, the processor automatically adds a parity-character to the block, establishing lengthwise parity along each channel. A reading head on the handler, immediately behind the writing head, automatically reads back the information just written, and checks the 2-dimensional parities; if there has been any error in writing the block, there is an automatic branch in the processor program, so that the tape may be backed up, and the block re-written. This branch actually goes into a master program called **STEP** (Standard Tape Executive Program), which makes several attempts to write the block, and if still unsuccessful, lays down a unique *skip* block and then writes the desired block on the tape. When the tape is later read, this *skip* block will cause another automatic branch into **STEP**, which will discard it and proceed with the processing.

**STEP** is an extremely sophisticated program, which is held in memory at all times when magnetic tapes are used, and which handles automatically *all* housekeeping chores connected with tapes. These chores include label-checking, error-correction, block counts and check sums, end-of-tape alternation of handlers, end-of-file detection, run-to-run supervisor, etc.



MAGNETIC TAPE TRANSPORT

When a tape is mounted on a handler, it follows the path shown in the diagram. The loops at the sides are vacuum chambers, and the grey area represents a cover which should be removed only by maintenance personnel. A leader is permanently fastened to the take-up reel, and when the tape is completely rewound, the tape-to-leader splice is between the tape clamp and the supply reel, with the leader threaded through the mechanism. In order to change tapes, the operator opens the door, closes the tape clamp, and disconnects the mechanical splice. He changes the reel, connects the splice, releases the clamp, and closes the door. Electronic interlocks prevent any use of the handler until the clamp is released and the door closed.

After the tape has been loaded, the first Read or Write instruction addressed to the handler causes the take-up reel to wind the leader, and the tape, until the Beginning-of-Information Marker (BIM) is detected at the BIM/DWM Sensing Station "B" and then reading or writing begins. When a Rewind instruction is addressed to the handler, the mechanism is reversed and the supply reel winds the tape until the BIM is detected at "B". The tape is now ready to be re-read, or re-written, if desired.

When recording tape, station "B" is alert for Destination Warning Marker (DWM) and upon detecting it, signals the processor that the physical end of the tape is approaching. The processor program should now write an end-of-tape Control Mark. The BIM and DWM are reflective spots on the back of the tape, 10 feet and 18 feet from the beginning and end, respectively, and station "B" detects them photo-electrically.

The leader and the trailer ("trailing leader") are made of electrically-conductive material, and are detected at the Leader-Trailer sensing station "A". If writing continues too long after the DWM, or if an attempt is made to read past the end of the recorded information, station "A" will detect the trailer before any physical damage can result, and the handler will go out of the "operate" status.

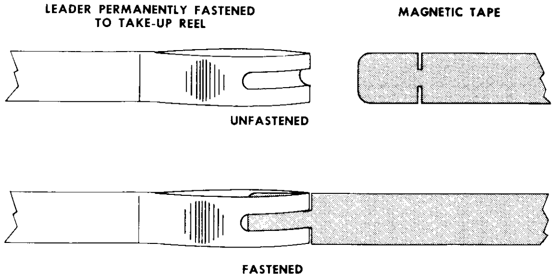
When a reel of tape is to be changed, after being rewound to the BIM, the operator presses the manual Rewind button on the handler, causing additional rewind until the leader reaches station "A" and passes a few inches beyond, in position to be clamped conveniently.

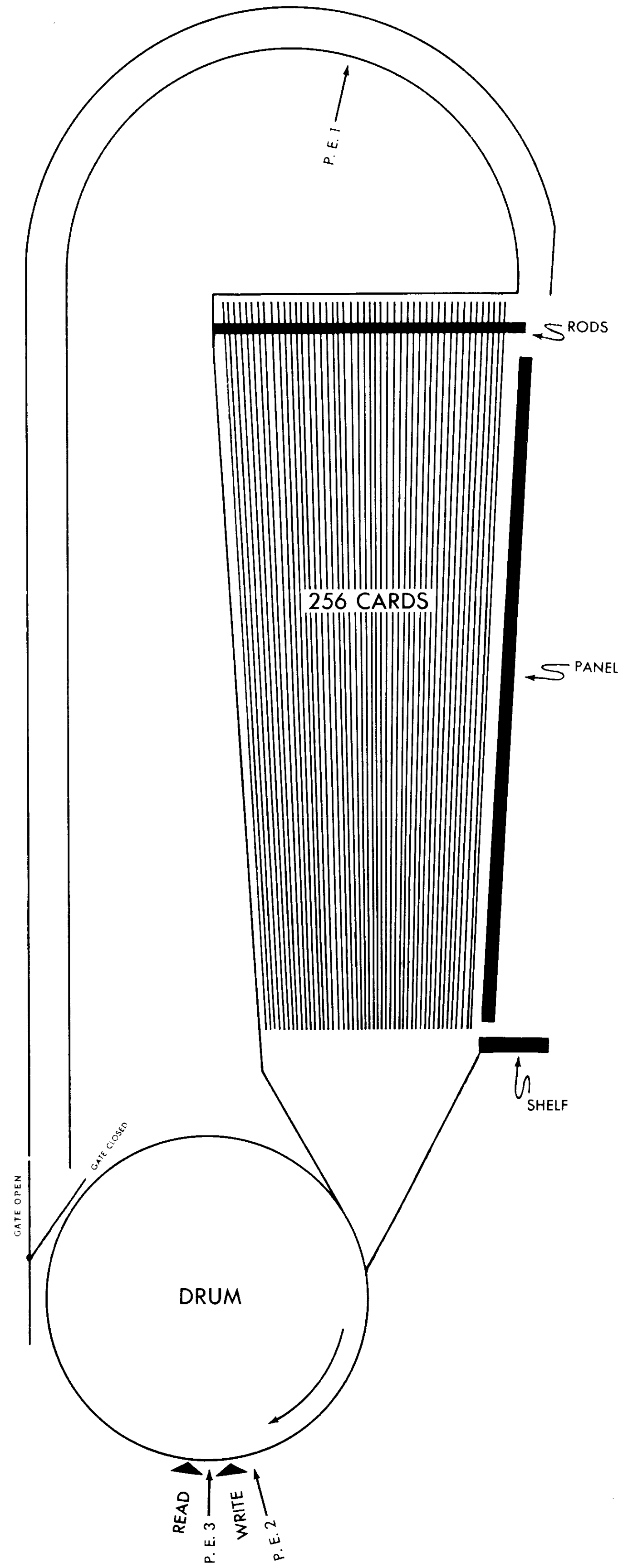
The forward speed of the tape while reading and writing is 120 inches per second. Rewind speed is 240 inches per second. The lower Tape Packing Arm detects the point at which the tape is almost completely rewound, and reduces the rewind speed to 120 inches per second for the last few feet of tape.

The Standard Handler (24/40 kc) may use either 1 mil tape (3600-foot reels) or 1½ mil tape (2400-foot reels). The High Speed Handler (24/40/60 kc) requires 1½ mil tape. All tapes recorded for IBM-compatibility must be 1½ mil. All 1½ mil tape must be "hard binder," rather than "sandwich."

## TAPE-TO-LEADER SPLICE

### One-Half Size




P. E. 1
RODS
256 CARDS
PANEL
SHELF
GATE CLOSED
GATE OPEN
DRUM
READ
P. E. 3
WRITE
P. E. 2

## *CARD RANDOM ACCESS MEMORY* (CRAM):

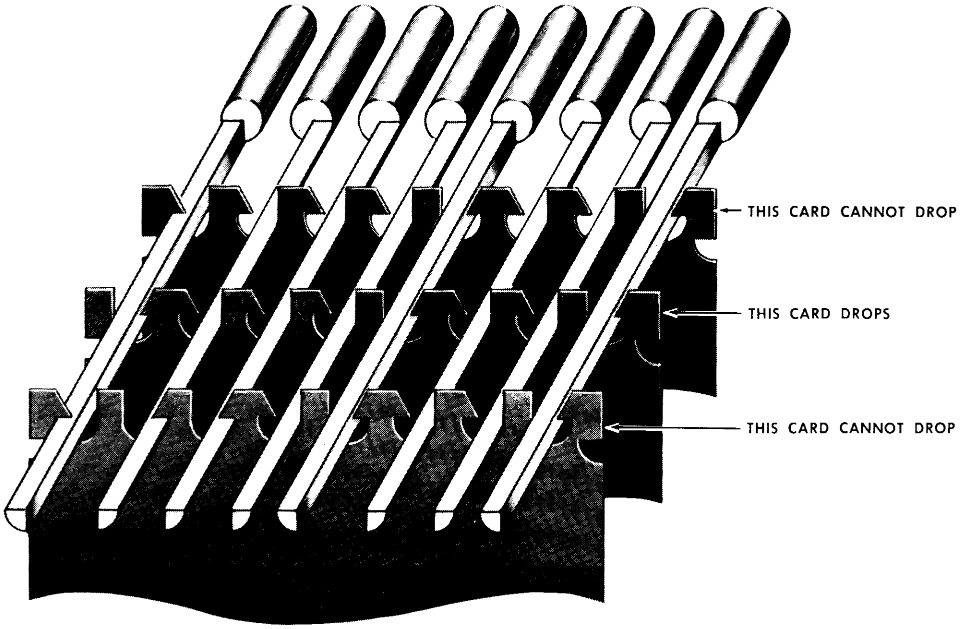
A CRAM unit holds a deck of 256 magnetic cards. When the processor program calls for a particular card, that card drops out of the chamber and wraps around a rotating drum, where it is held by air suction. The card now has the properties of a magnetic drum memory, in which the recording occupies $^2/_3$ of the drum circumference. One-third of each drum-revolution is free for updating the information so that it is ready for re-recording during the next revolution.

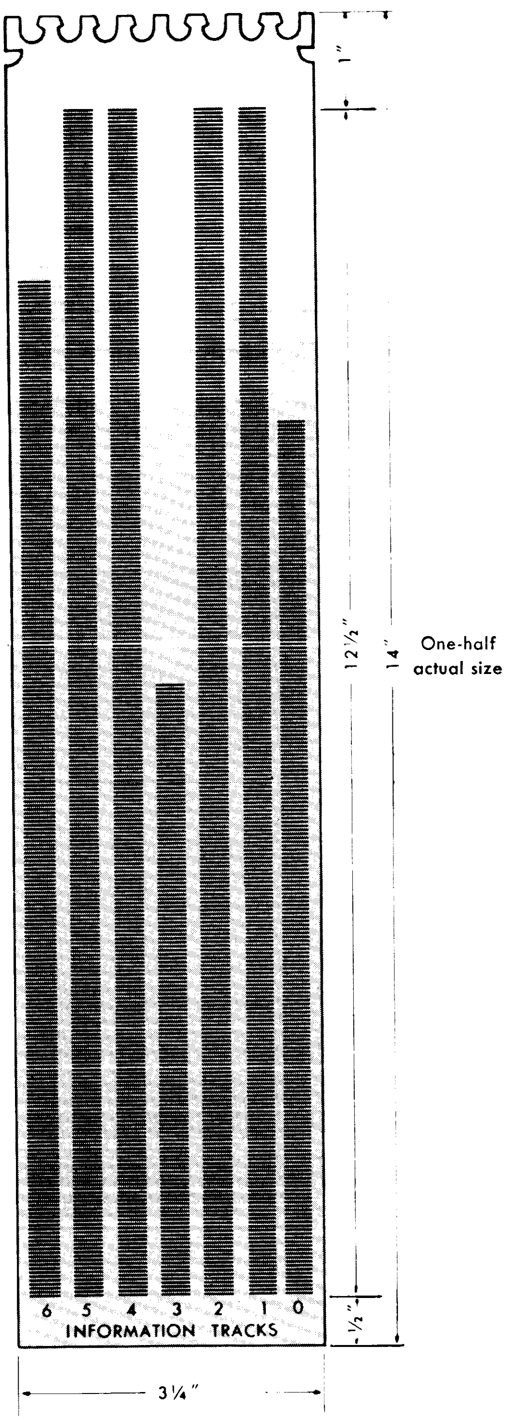
When the program has finished with this card the exit gate opens, the card is released from the drum, and it goes up through a chute back to the chamber. When the leading edge of the card reaches a photo-electric cell (P.E. 1 in the diagram opposite) the panel retracts, and the card drops into the chamber. It is slowed by a braking mechanism (not shown), and is stopped by the shelf. Then the panel comes forward to put the card into the chamber, where it fits on the rods.

Each card has eight notches across the top, cut in a pattern which is unique to that card, and the cards hang in the chamber from a row of eight selection rods. The diagram below shows the principle of card selection. When the processor calls for a particular card, the rods rotate into the combination of positions corresponding to that card. A pair of gating rods, which fit into notches at the sides of the cards, support the entire deck while the rods are coming into position; then they swing back, and the selected card drops out of the chamber.

This achieves a true random selection of cards, as each card is chosen by the selection rods regardless of its position within the deck. In fact, the physical sequence of the cards is itself random, since the last-used card is always replaced at the back of the deck.

PRINCIPLE OF CARD SELECTION





EACH TRACK CONTAINS UP TO 1550 SLABS
CAPACITY OF 1 CARD: OVER 32,500 DIGITS
CAPACITY OF 1 CARTRIDGE: OVER 8,300,000 DIGITS

While the card is on the drum it may be written or read in any of seven information-tracks. No switching time is involved in changing from one track to another, as the CRAM is provided with individual read-write heads for each track. The pattern of tracks on the card is shown in the diagram. Each information-track holds one block of information which may contain 1 to 1,550 slabs (2 to 3,100 alphanumeric characters, 3 to 4,650 digits or any proportional mixture). Recording is performed in "image alpha" form, in seven bit-channels per track, with character-parity and block-parity, as on magnetic tape. And as with magnetic tape, each track has a read head immediately behind the write head, so that all recording is automatically checked while it is being performed. Information transfer rate is 100 kc (100,000 alphanumeric characters or 150,000 digits per second).

A Read or Write operation always starts at the beginning of a track, and terminates as soon as information transfer is complete, even if the block is less than 1,550 slabs long. The CRAM also has the ability to execute a partial-read, which terminates at any desired point within the block. Termination of the operation at end of information, rather than at the end of the track, allows a greater portion of the drum-revolution to be shared with other processor operations.

A master program called **PACE** (PAckaged CRAM Executive), which is analagous to **STEP**, performs all housekeeping chores for CRAM. The scope of **PACE** is similiar to that of **STEP**.

When the processor calls for a CRAM to drop a particular card, the selection rods release that card and it starts to drop. From the moment the processor calls for the card to the moment the leading edge of the card reaches photo-electric cell P.E. 2 (see diagram on page 42) the CRAM is in a DROPPING status, and is inhibited from accepting a command to drop another card.

As soon as the leading edge of the card reaches P.E. 2 the DROPPING signal is turned off, and the LOADED signal goes on, indicating that a card is loaded on the drum. Only a portion of the card is actually in contact with the drum at this point, but reading or writing may now begin.

When a card is released from the drum, the CRAM is no longer LOADED, and it will not accept a Read or Write instruction.

The CRAM will accept a Read or Write instruction at any time a card is LOADED, but the instruction will be executed with minimum delay only while the leading edge of the card is between P.E. 2 and P.E. 3. Once the leading edge has passed P.E. 3, it is too late to read or write during this revolution, since the beginning of the track has passed the write head; and until the leading edge reaches P.E. 2, it is too early to read or write during the next revolution. When the leading edge is between the two P.E. cells, the card is in *minimum-access position* and can exercise Demand so that the processor program may immediately read or write.

A good deal of time-sharing is possible with the CRAM. As soon as a card is no longer DROPPING, the unit may be instructed to drop the *next* card, and then to read or write the card which has just been loaded (the *present* card). In this case, the unit is both DROPPING and LOADED at the same time. This situation will remain for one revolution of the card, and then the present card will automatically be released from the drum, and return to the chamber.

When processing CRAM files, the program will determine when it is about to perform the last Read or Write on a Card, and will call for the next card before performing that last operation on the present card. There will be a significant interval between completion of that operation and the moment when the CRAM is LOADED with the next card; the processor will utilize that interval to perform other work, and as soon as the next card reaches P.E.2 (the CRAM is LOADED) it will exercise Demand.

If an interval of 750 milli-seconds passes without any command being addressed to a CRAM, the card on the drum is released and returned to the chamber.

### UNBUFFERED PRINTERS: BUFFERED NUMERIC LISTERS:

Any or all of the Line Printers connected to an NCR 315 system may optionally be unbuffered. In the absence of the buffer, the Processor is tied to the Printer during the print cycle, but is free for other work while the paper is moving.

The unbuffered Printer operates at 600 alphanumeric lines per minute, 790 numeric lines per minute, intermixed in any fashion.

The same Printer may be optionally ordered with a type line consisting of 96 alphanumeric positions, and 24 numeric-only positions.

Under program control, the Printer with the optional type line may operate in an unbuffered mode, printing the full 120-position line (with the RH 24 positions limited, of course, to numeric information) or as a BUFFERED numeric lister, using only the RH 24 print positions, operating at the rate of 1750 lines per minute.

### MAGNETIC CHARACTER READERS: OPTICAL CHARACTER READERS: REMOTE INPUT-OUTPUT SYSTEM:

These devices are discussed in separate pamphlets.

## READY STATUS AND DEMAND INTERRUPT

As a general principal, if a peripheral unit is READY, and if its Unit Demand flag is set, it will transmit a *demand signal* to the processor; if the processor's Demand Permit flag is set, Demand interrupt will occur.

When interrupt occurs, the demand program must determine which of several peripheral units caused the interrupt. This is done by attempting to *select* each unit in turn, since the Select operation includes a test for READY. However, each unit has its individual characteristics, which are summarized in the table on the next page.

Note that the card reader does not have a Unit Demand flag. When reading cards there will usually be no occasion to have any other units demanding, since the timing of the card reader will govern other peripheral operations.

Priorities among competitively-demanding units are determined simply by the sequence of the SELECT commands in the program.

## READY STATUS AND DEMAND INTERRUPT

| | READY | | DEMAND SIGNAL<br>Exercise Demand if Demand Permit flag on | |
|---|---|---|---|---|
| | ON | OFF | ON | OFF |
| BUFFERED PRINTER | Finished printing a line unless instruction took JY + 1 | PRNT instruction loads the buffer | READY on and Unit Demand on | READY off or Unit Demand off |
| CARD PUNCH | Finished punching a card<br>or<br>Could not punch a card | PNCH instruction loads the buffer<br>PNCH instruction takes branch | READY on and Unit Demand on | READY off or Unit Demand off |
| CARD READER | Card leading edge reaches reading station | Card column 1 reaches reading station | READY on | READY off |
| CRAM | Card in minimum access position AND Unit Demand on; unless present card after SELC:DN | WCC or RCC instruction or card passed P.E. 3 or Unit Demand off | READY on (includes Unit Demand on) | SELC instruction or READY off |

# INPUT, OUTPUT, FILE OPERATIONS

## *SELECT a CRAM*

| OP | V | L | X/Y | A/B |
|---|---|---|---|---|
| SELC | (V) |  | X | A |
|  |  |  | Y | J |

Jump is to the instruction whose address is "J" + (contents of RY).
Link in R15.

SELC:DP **Select** a **CRAM**, **drop** a card.

The next Read or Write is intended for the **present** card.

Jump if a card is now DROPPING on this unit.

The present card remains available for one more Read or Write before it is released from the drum.

This unit will become READY (if its Unit Demand flag is set) next time the present card reaches minimum-access position.

If the next Read or Write is issued too late for the present card, *that* operation will take the "wrong card" jump.

Selecting a CRAM has no effect on its Unit Demand flag.

Selecting a CRAM turns off the Demand signal in that CRAM.

**NOTE**: All SELect commands are protected against Demand.

Processor hangs up if:
  Two or more units assigned same number;
  No unit assigned this number;
  Power turned off in this unit.

SELC:DN **Select** a **CRAM**, **drop** a card.

The next Read or Write is intended for the **next** card.

Jump if a card is now DROPPING on this unit.

If the next Read or Write is issued too soon for the next card, *that* operation will take the "not loaded" jump.

As soon as this instruction is given, the LOADED status is terminated, and the present card becomes inaccessible. The CRAM will become LOADED again only when the next card reaches the drum and, if the Unit Demand flag is on, it will become READY at the same time.

**(A)** is a 2-slab word:

| CRAM No. | Card No. |
|---|---|

CRAM numbers: 0- 7
10-17
only RH 5 bits are used

Card numbers: 000-255
expressed as an 8-bit binary number. A standard subroutine is provided for the decimal-to-binary conversion.

*SELC:T **Select** a **CRAM** and **test** for READY (card in minimum-access position, and Unit Demand set).

Jump if unit is READY.

**(A)** or "**A**" is CRAM number.

*SELC:R **Select** a **CRAM** and **release** present card after one more opportunity to Read or Write.

Jump if unit is not LOADED (no card on drum, or present card after SELC:DN).

**(A)** or "**A**" is CRAM number.

### *SELECT other input-output devices

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| SELP | | | X | A |
| | | | Y | J |

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| SELS | | | X | A |
| | | | Y | J |

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| SELQ | | | X | A |
| | | | Y | J |

Jump is to the instruction whose address is "J" + (contents of RY).
Link in R15.

*SELP: Select a Printer or Card Punch.
(**A**) or "**A**" is unit number: 0-3
Only RH 2 bits are used.
Jump if unit is READY.

*SELS: Select a Sorter.
(**A**) or "**A**" is unit number: 0-3
Only RH 2 bits are used.
Jump if unit is READY.

*SELQ: Select an Inquiry buffer.
(**A**) or "**A**" is unit number.
Jump if unit is READY.

Selecting a unit has no effect on its Unit Demand flag.

These instructions do not turn off the Demand signal in the peripheral unit; the instructions which operate the units do turn it off.

**NOTE**: All SELect commands are protected against Demand.

Processor hangs up if:
Two or more units assigned same number;
No unit assigned this number;
Power turned off in this unit.

### *SET and CLEAR Unit Demand Flags

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| SETU | (V) | | X | A |
| | | | 0 0 | |

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| CLRU | (V) | | X | A |
| | | | 0 0 | |

The second line of the instruction format is not used. It is suggested that zeros be entered in the Y column to indicate on a coding sheet that this is a double-stage instruction.

*SETU:C Set
*CLRU:C Clear } Unit Demand in CRAM.
(**A**) or "**A**" is unit number: 0-7
10-17
Only RH 5 bits are used.

*SETU:P Set } Unit Demand in Printer
*CLRU:P Clear } or Card Punch.
(**A**) or "**A**" is unit number: 0-3
Only RH 2 bits are used.

*SETU:S Set
*CLRU:S Clear } Unit Demand in Sorter.
(**A**) or "**A**" is unit number: 0-3
Only RH 2 bits are used.

*SETU:Q Set } Unit Demand in Inquiry
*CLRU:Q Clear } buffer.
(**A**) or "**A**" is unit number.
**B** is used to specify R-Demand and/or W-Demand.

Setting or clearing Unit Demand has no effect upon Selection of units.

**NOTE**: All modes of SETU and CLRU are protected against Demand.

Processor hangs up if:
Two or more units assigned same number;
No unit assigned this number;
Power turned off in this unit.

## PUNCH PAPER TAPE:

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| P P T | (V) | | X | A |
| | | | | N |

Punch N rows of paper tape from memory, starting with the LH end of the A-area.
N may be 000-999.

In each mode, an image of the bit-configuration in memory is punched into the tape, with holes corresponding to 1-bits. Channel 1 on the tape (see illustration below) corresponds to the RH bit of the character or the slab. Before punching, the program must translate each character into the bit-configuration used for that character in the paper tape code. Standard subroutines and macro-instructions are furnished for this purpose.

It is customary to translate and punch one character at a time, since there is sufficient time between characters to perform the table-lookup while the Punch is operating at its full speed of 110 characters per second.

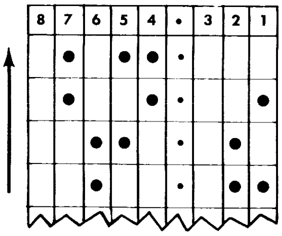
When punching tape for communication between Processors, the Character mode is used and no translation is required, as the instruction will punch an exact image of the bit-configurations stored in memory. In that situation, N may be made as large as convenient.

PPT :C Each alpha **character** is punched into the RH 6 channels of a row on the tape. The Processor automatically punches *hole* or *no-hole* into the seventh channel to establish odd-parity (an odd number of holes in each row).

Thus the two slabs

| H 9 | S L |
|---|---|

would be punched as:



**H** (01 1000)
**9** (00 1001)
**S** (11 0010)
**L** (10 0011)

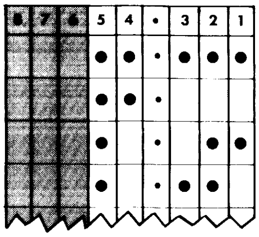
The punching corresponds to QRST in tape-to-card conversion code (odd-parity). Before punching, the translation subroutine for this output code will have translated QRST into H9SL.

PPT :S The RH 8 bits of each **slab** are punched into the 8 channels of a row on the tape.

Thus the four slabs

| 0 ↑ | 0 H | 0 C | 0 F |
|---|---|---|---|

would be punched as:



**0 ↑** (00 0000 01 1111)
**0 H** (00 0000 01 1000)
**0 C** (00 0000 01 0011)
**0 F** (00 0000 01 0110)

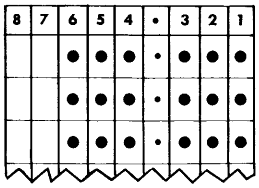
The punching corresponds to letter-shift, ABC in 5-channel telegraph code (no parity). Before punching, the translation subroutine for this output code will have translated ABC into 0↑0H0C0F, and the translation would have included insertion of the required letter-shift.

NOTE: If, for any reason, the program issues a PPT too late to maintain full punching speed, the Punch will stop and wait. The only time lost is the waiting time.

If N = 0, these operations do nothing.

TAPE FEED:

When the Tape Feed button on the Punch is depressed, the Punch will continuously emit the following configuration until the button is released:



When punching in any given code, the program will .initially punch a foot or two of the *run-in* configuration defined for that particular code, as a leader; and again at the end, as a trailer. After the tape has been punched, the operator may tear off the ends of the tape which contain the *tape feeds.*

When punching without translation, for communication between Processors, the *tape feed* itself will be used as the *run-in* configuration.

Special facilities have been provided in the Read Paper Tape instruction to accommodate the fact that this is an even-parity configuration.

Processor hangs up 6 inches from end of tape, if no tape, if tape breaks, or if power off in Punch.

## READ PAPER TAPE

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| R P T | (V) | | X | A |
| | | | Y | N |

Two-way jump table starts in JY.
Link in J14.

Start the Reader (unless it is already running) and read N rows of paper tape into memory, starting at the LH end of the A-area.

N may be 000-999.

If the program issues another RPT instruction soon enough, the tape will continue moving at full speed. If the program does not issue another RPT instruction by the time the Reader reaches the next character, the tape will stop in position to read that character; this will cost an acceleration time when the tape is restarted.

In each mode, an image of the row is stored as a bit-configuration in memory, with 1-bits corresponding to holes. Channel 1 on the tape (see illustration under PPT) corresponds to the RH bit of the character or the slab. After reading, the program must translate each bit-configuration into the actual character. Standard subroutines and macro-instructions are furnished for this purpose.

It is customary to read and translate one character at a time, since there is sufficient time between characters to perform the table-lookup while the Reader is operating at its full speed of 1000 characters per second.

When tape has been punched without translation, for communication between Processors, the tape contains an image of the bit-patterns which were in memory at the time of punching; reading is done in the Character mode and no translation is required. In that situation, N may be made as large as convenient.

If N = 0, these operations do nothing.

JY: Parity error branch for RPT:C and RPT:CX.

JY + 1: Tape feed branch for RPT:CX if N=1.

Processor hangs up if no tape, broken tape, end of tape, or if power off in Reader.

RPT :C The RH 6 channels of successive rows are read into successive **character**-positions in memory. Channel 7 is used by the Processor to verify odd-parity (an odd number of holes in each row) and is not stored in memory. Channel 8 is ignored.

Complete slabs are always stored. If N is an odd number, then a *zero* will be stored as the RH character of the last slab.

If any row on the tape contains an even number of holes, the Processor will store the character **x** in memory for that row. After all N rows have been read, the Processor will then take its next instruction from the address named in JY (parity error branch).

*Tape feed* (see PPT) will be ignored if it precedes data during any execution of the instruction. That is, *tape feeds* will be passed over without reading, and without counting, until some other configuration is found on the tape.

*Tape feed* is treated as a parity error if it occurs within the data.

RPT :CX This is an **extra** variation of the character mode, permitting use of *tape feed* as an end-of-item code. This special variation is identical with the Character mode, except when N = 1.

If N = 1, and if the first row on the tape is a *tape feed,* only that character is passed on the tape. Nothing is stored in memory, and the Processor takes its next instruction from the address named in (JY + 1).

RPT :S The 8 channels of each row on the tape are read into the RH 8 bit-positions of a **slab**, with the LH 4 bits of the slab set to 0 bits.

*Tape feed* is just another 8-bit configuration in this mode.

Y is not used, as this variation does not branch.

## *FEED a Punched Card and READ Columns*
## *READ Columns from a Punched Card*

### 400 card per minute reader

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| R C θ L | F | | X | A |
| | | | Y | N |

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| R C θ L | | | X | A |
| | | | Y | N |

Jump is to address in JY.
Link in J14.

Punched cards are read serially, column by column, beginning with card-column 1. N columns of a card are read into memory, starting at the LH end of the A-area. If columns 81 and 82 have been punched, they are ignored.

N may be 000-159.

The image of each card column is stored as a bit-configuration in one slab of memory, with 1-bits corresponding to holes in the card. Card row 9 is stored as the RH bit of the slab; card row 12 is stored as the LH bit of the slab (see illustration on the opposite page). After reading, the program must translate each bit-configuration into the actual character. Standard subroutines and macro-instructions are furnished for this purpose.

It is customary to read and translate one column at a time, since there is sufficient time between card columns to perform the table-lookup while the card is passing.

When cards have been punched in the Direct mode, for communication between Processors, each card column contains an image of the bit-patterns which were in memory at the time of punching, and no translation is required when reading. In that situation, N may be made as large as convenient.

When the leading edge of each card reaches the reading station, the Card Reader will exercise Demand if the Processor's Demand Permit flag is on. Note that there is no Unit Demand flag in the Card Reader, and when reading cards, all other Unit Demand flags will usually be turned off.

Processor hangs up if input stacker empty, if a card is mis-fed, if power off in Reader, or if the instruction calls for reading columns from a card that has not been fed.

RCθL:F Feed a card, read N columns.

If N=0, feed only; do not test for previous *missed column.*

If a FEED is issued after column 1 and before column 25 of the present card, then the Reader will remain in *continuous feed* and the next card will be fed with minimum interval between cards.

A FEED issued before column 1 can apply only to the present card. If this card has already been fed, the new FEED does nothing.

RCθL: Read N columns.

If N=0, this instruction does nothing.

JY:

Missed-column jump. Branch without execution. One or more columns of the present card have passed the reading station without being captured by an RCθL or RCθL:F instruction. The current instruction will not be executed, but will branch instead to the address named in JY.

Only the instruction RCθL:F with N=0 may be executed when the missed-column condition exists. This instruction will not branch; it *will* set the feed signal for the next card.

The missed-column condition is automatically reset when the trailing edge of the card passes the reading station. Thus reading may terminate when the desired portion of a card has been read, and resume with the next card without encountering the JY branch.

HOW TO USE THESE OPERATIONS:

The feeding and reading functions are, to some extent, independent of each other. When the Card Reader is at rest a FEED will, of course, cause a card to be fed past the reading station. However, once column 1 of this card has been read, the Reader will accept and store a signal to feed the next card as soon as this card has passed.

Thus, purely as an illustrative example, the following sequence would cause the entire contents of two cards to be read.

FEED and READ 1 column.

A card is fed; the Processor waits for column 1, reads it, and terminates the instruction.

The card continues to move past the reading station.

FEED and READ 159 columns.

> The Reader accepts and stores the feed signal for the second card.
>
> The Processor waits for column 2 of the first card, reads it, waits for column 3, reads it; and so on through column 80.
>
> The Reader feeds the next card immediately.
>
> The Processor waits for column 1, reads it; and so on through column 80. Then the instruction terminates.

However, this would be a very inefficient way to use the Processor, since all the time between columns, and between cards, would be wasted in waiting. A more efficient procedure, and the one actually used, is outlined:

- Feed and Read zero columns.
- Set Demand Permit flag.

A ● Begin to execute some other program until the leading edge of the card reaches the reading station. The Reader will then exercise Demand.

- Feed and read 1 column into working storage.
- Translate that column into a character, and store it in memory.
- Repeat the previous two operations 79 more times to complete the card. In the operation that reads column 2, the Feed is significant; all the other Feeds are redundant.
- Set Demand Permit flag.
- Demand Link Return. Resume execution of the other program (point A in this outline).

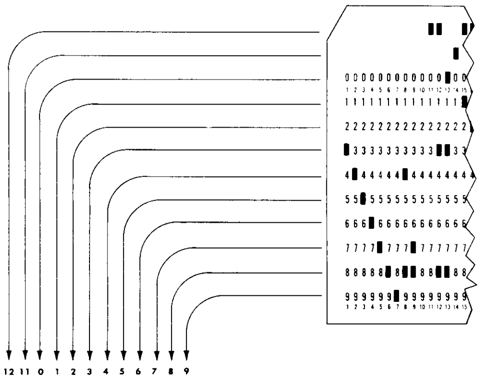
And so on for as many cards as will fit into the input area that the programmer has assigned. The last card is read with the READ mode in order to terminate feeding of cards.

Further time-sharing may be achieved if the input requires only the early columns of each card. The program stops the column-by-column reading after the last wanted column, and proceeds to other work.

The missed-column condition is set when the first unwanted column passes the reading station without being read. But the program never encounters this condition since there is no further attempt to read the current card, and the condition is automatically reset by the trailing edge of that card.

The leading edge of the next card causes the Reader to exercise Demand, informing the program that it is time to begin reading the columns of the next card.

The formation of the card-column image in memory is illustrated in the following example:



| Card Rows | 12 | 11 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Card Columns |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SUCCESSIVE SLABS OF MEMORY | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 3 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 4 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 5 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 6 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 7 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 8 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 |
| | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 |
| | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 12 |
| | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 13 |
| | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 14 |
| | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 15 |

Bit-patterns in memory

### *FEED a Punched Card and READ Columns*
### *READ columns from a Punched Card*

**2000 card per minute reader**
*without* **translation by the reader**

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| R C θ L | F | | X | A |
| | | | Y | N |

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| R C θ L | | | X | A |
| | | | Y | N |

Jump is to address in JY.
Link in J14.

These two instructions work for the 2000 cpm reader exactly as they do for the 400 cpm reader, except that at the higher reading speed there is not time to perform any significant amount of processor work between columns.

### *FEED a Punched Card and READ Columns*
### *READ Columns from a Punched Card*

**2000 card per minute reader**
*with* **translation by the reader**

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| R C θ L | T F | | X | A |
| | | | Y | N |

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| R C θ L | T | | X | A |
| | | | Y | N |

Jump is to address in JY.
Link in J14.

When these two instructions are used with the 2000 cpm reader, they work just like RCθL:F and RCθL except:

- The reader translates each card column into a character.
- N is the number of **slabs** to be stored, two characters (two columns) to a slab.

If the reader detects a configuration in any column which does not correspond to one of the 64 characters it can translate, it will store the character **x** in the processor and set the Overflow flag.

The program may switch from translate to non-translate and back as many times as desired within a card, if the card fields make it appropriate to do so. However, if an odd number of untranslated columns have been read before switching, the last untranslated column will appear a second time as a character.

Suppose the first half-dozen columns of a card contained **A B C D E F**, and that three columns were read untranslated, then two slabs with translation. Five slabs of memory would receive information; the first three would contain **A B C** as images of the columns, and the next two would contain **CD EF** as translated characters.

If these instructions are addressed to the 400 cpm reader, they will work just like RCθL:F and RCθL.

OVERFLOW: If, during the execution of either of these instructions, the reader found an unallowable configuration and substituted **x** for it.

Processor hangs up if input stacker empty, if a card is mis-fed, if power off in Reader, or if the instruction calls for reading columns from a card that has not been fed.

### *PUNCH a card*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| P N C H | | | X | A |
| | | | Y | S |

S controls use of Selectors on punch plugboard.
Jump is to address in JY. Link in J14.

The instruction automatically addresses itself to whichever Punch was last selected by SELP. If a Printer was selected last, this instruction will be interpreted as PRNT and will operate the selected Printer.

The 40-slab or 80-slab A-area, containing the edited information to be punched, is transmitted to the selected Card Punch Buffer, together with the Selector control. This terminates the instruction within the Processor, and the program immediately resumes.

Under its own independent control, the Card Punch Buffer operates the Punch and punches one card. When it has finished this task, it reports READY.

The control panel on the Card Punch Buffer includes a 2-position switch, labelled TRANSLATE and DIRECT.

When the switch is in the T position, the Processor transmits 40 slabs (80 characters) and the card is punched in the *Character* mode, one alpha to a column, with each alpha automatically translated into conventional card-code. The translation is performed by an encoding matrix in the Card Punch Buffer, and does not affect the timing of the operation in any way.

When the switch is in the D position, the Processor transmits 80 slabs (160 characters) and the card is punched in the *slab* mode, one slab to a column, with each column punched as an exact image of the bit-pattern in the slab: holes correspond to 1-bits. The correspondence between bit-positions in memory, and rows in the card column, is the same as in RCØL; subsequent reading of the card without translation will duplicate memory as it was when the card was punched.

SELECTOR CONTROL:

Optional with the Model 523 punch, and standard with the Model 7550 punch, are three pairs of selector hubs on the plugboard, marked 1, 2, 3. The 3 RH bits of S are regarded as correspondingly numbered:

Bits of S
x 3 2 1

and each bit-position controls the correspondingly-marked pair of selector hubs. Thus up to 8 different combinations may be programmed, corresponding to 8 plugboard operations.

JY:

*This* instruction will abort if the *previous* card on this Punch could not be punched. The information for that previous card remains in the buffer. Instead of executing *this* instruction, the Processor will take its next instruction from the address named in JY.

The following conditions will prevent punching of a card. Each of them causes the Punch to become READY, even though the card has not been punched.

- Power has gone off in this Punch.

  The next card cannot be punched, and the PNCH instruction after that will abort.

- Card misfeed or card jam.

  A misfed card cannot be punched at all; a jammed card cannot be punched completely. The next PNCH instruction will abort.

- Input hopper empty or output stacker full.

  The next card cannot be punched, and the PNCH instruction after that will abort.

- Plugboard programmed halt, such as detection of double-punch or blank-column.

  Card "1" contains a punching error;

  Card "2" is being punched while "1" is being checked. After "2" is completely punched, the Punch halts because of the error in "1".

  Card "3" cannot be punched.

  The PNCH instruction for card "4" will abort.

  The error-card "1" will be the last in the output stacker. Card "2" will be in the read-check station.

When a PNCH instruction aborts and takes the JY branch, the Punch becomes NOT READY, and remains in that state until the condition is corrected by the operator. He must clear the buffer, usually by punching its contents into a card; operator controls are provided for this purpose.

READY:

A Punch is READY when it has completed the punching of a card, or when it has failed to punch a card and has set the branch indicator.

It is NOT READY while it is punching a card, or after the Processor detects a branch condition and resets the branch indicator.

Ordinarily the program will test the READY state by means of SELP before issuing PNCH. However, if PNCH is issued to a Punch which is NOT READY, the Processor will wait until the Punch becomes READY.

DEMAND INTERRUPT:

Whenever a Punch is READY, it will exercise Demand if its own Unit Demand flag, and the Processor's Demand Permit flag, are both on.

Processor error-halts if no Punch has been selected; hangs up if Punch is not ready, or if power off.

### *HALT and accept Console input*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| H A L T | (V) | | X | A |
| | | | 0 0 | |

The second line of the instruction format is not used. It is suggested that zeros be entered in the Y column to indicate on a coding sheet that this is a double-stage instruction.

Execution of the program is halted, and the operator is free to perform any Console operation he chooses. Any information entered on the Console Keyboard will be stored in memory, starting at the LH end of the A-area. However, the operator may change the putaway address at his option. As many slabs as the operator desires may be entered.

HALT:D  Store input in memory as Digits. If a non-digit character is entered, the Processor will ignore that character, and error-halt.

HALT:A  Store input in memory as Alphas.

Before entering any information, or after pressing REST after some information has been entered, the operator may press the ALPHA or DIGIT button to change the mode of input, if desired.

Keyboard entries are held in a 1-slab input register, which is automatically putaway into successive slabs of memory each time it is filled with 2 Alphas or 3 Digits.

Execution of the program may be resumed by pressing the COMPUTE button on the keyboard. However, this button is inoperative unless the input register is empty. Pressing the REST button will clear the input register without putaway, and will permit use of the COMPUTE button, or any other Console operation.

ERROR HALT:  If a non-Digit character is typed while in HALT:D. The character is not entered.

### *PRINT (buffered printer)*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| P R N T | | | X | A |
| | | | Y | M F |

This digit always zero

M is Mode of Format Control.
F is Vertical Format Control.

JY is beginning of 2-way Jump Table.
Link is J14.

The instruction automatically addresses itself to whichever Printer was last selected by SELP. If a Card Punch was selected last, this instruction will be interpreted as PNCH and will operate the selected Card Punch.

The 60-slab A-area, containing the edited information to be printed, is transmitted to the selected Printer's Buffer together with the Mode and the Vertical Format Control. This terminates the instruction within the processor, and the program immediately resumes.

Under its own independent control, the Printer moves the paper as specified by M and F, then prints the line. When it has finished this task, it reports READY.

MODE:

| | OUT-OF-PAPER JUMP (JY+1) | |
|---|---|---|
| | Print this line | Do not print this line |
| F is number of lines | M=0 | M=2 |
| F is recognition code | M=1 | M=3 |

If M is 0 or 2, then F is the actual number of lines of paper to be moved before printing this line. If F is a non-decimal digit, then:

| F | means |
|---|---|
| @ | 10 |
| , | 11 |
| space | 12 |
| & | 13 |
| . | 14 |
| - | 15 |

F=0 means do not move paper at all; this will cause this line to be overprinted on the previously printed line.

If M is 1 or 3, paper is moved until a punched configuration on the VFU Tape (see below) exactly matches the bit-configuration of F, where holes in the tape correspond to 1-bits of F. This line is then printed at the point on the paper opposite that configuration on the tape. Do not use *period* as a recognition code, as this is reserved for possible special function in future model printers.

If F=0 the paper will stop at the first unpunched position on the VFU Tape.

If there is no VFU configuration which matches F, the Printer moves paper until the VFU "special" code (4 holes) is reached the *second* time. Paper movement then stops, and the Printer error-halts.

NON-PRINTING CHARACTERS:

There are 8 "non-printing" characters which are merely convenient marks for their respective binary configurations, and which will not appear in data. However, in producing program listings, the Printer will be called upon to print them.

A non-printing character will appear on the printed page as a capital letter, overprinted with a *plus* sign:

| | | | |
|---|---|---|---|
| ? | M | ' | U |
| : | N | [ | V |
| ← | @ | ] | W |
| ↑ | P | \ | X |

†JY (end of page):

If paper movement for any PRNT command reached or passed a "special" code (all 4 holes) on the VFU Tape, the JY indicator is set in the Printer. The *next* PRNT command will take the JY branch instead of being executed, the JY indicator is cleared, and the Printer remains in the READY state.

†JY+1 (out of paper):

At the time the JY indicator is set, the Printer also checks whether it is printing on the last page of the continuous-form paper. If so, the JY+1 indicator is also set in the Printer. The next PRNT command will take the JY branch and turn off the JY indicator in the Printer.

The next PRNT command *after that* will detect the JY+1 indicator, turn it off, and take the JY+1 branch. The line will be printed or not, depending on the Mode of the command, but the branch is always taken. The Printer becomes NOT READY, and remains in that state until the operator loads more paper.

VERTICAL FORMAT UNIT (VFU):

This unit, which is part of the Printer, holds a loop of paper tape into which 15 different code-configurations can be punched. The loop is the same length as the form being printed, and the punched codes exactly correspond to the lines which they designate.

As the paper moves vertically, the VFU, and its tape loop, move with it so that all paper movement is controlled by the VFU tape as specified by M and F in the instruction.

READY:

A Printer is READY when it has advanced the paper and completed the printing of a line unless an "out of paper" condition exists.

Ordinarily the program will test the READY state by means of SELP before issuing PRNT. However, if PRNT is issued to a Printer which is NOT READY, the Processor will wait until the Printer becomes READY.

DEMAND INTERRUPT:

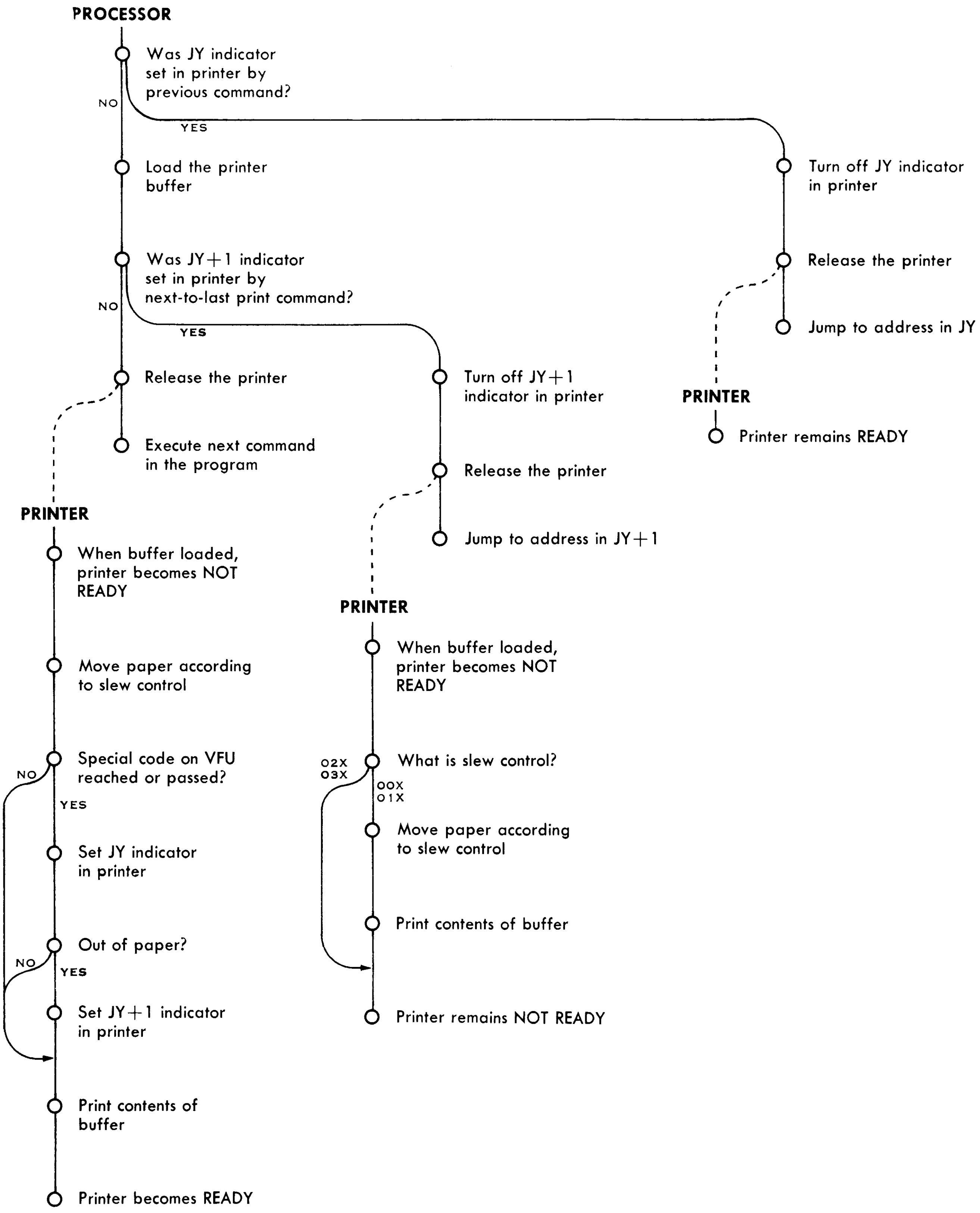
Whenever a Printer is READY, it will exercise Demand if its own Unit Demand flag, and the Processor's Demand Permit flag, are both on.

†See flow chart on next page for detail operation of PRNT and the jumps.

See Appendix for discussion of VFU conventions and programming techniques for using PRNT.

Processor error-halts if no Printer has been selected; hangs up if Printer not ready, or if power off.

# DETAIL OPERATION OF PRINT COMMAND (Buffered Printer)


PROCESSOR
Was JY indicator set in printer by previous command?
NO
YES
Load the printer buffer
Was JY+1 indicator set in printer by next-to-last print command?
NO
YES
Release the printer
Execute next command in the program
Turn off JY+1 indicator in printer
Release the printer
Jump to address in JY+1
Turn off JY indicator in printer
Release the printer
Jump to address in JY
PRINTER
Printer remains READY
PRINTER
When buffer loaded, printer becomes NOT READY
Move paper according to slew control
Special code on VFU reached or passed?
NO
YES
Set JY indicator in printer
Out of paper?
NO
YES
Set JY+1 indicator in printer
Print contents of buffer
Printer becomes READY
PRINTER
When buffer loaded, printer becomes NOT READY
What is slew control?
02X
03X
00X
01X
Move paper according to slew control
Print contents of buffer
Printer remains NOT READY

## STUDENT'S NOTES ON PRINT COMMAND

### *TYPE on Console Typewriter*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| T Y P E | (V) | | X | A |
| | | | | N |

Type N slabs from memory, starting with the LH end of the A-area. N may be 000-999.

TYPE:D Type digits.

TYPE:A Type alphas.

There is no format control in these modes, and the Console Typewriter is not provided with an automatic carriage-return.

TYPE:AP Type alphas with programmed format.

Whenever the characters ] and \ appear in the A-word, the Processor automatically substitutes the format-control operations "tab" and "carriage return" respectively.

] becomes Tab

\ becomes Carriage Return

If N=0, these operations do nothing.

### *READ MAGNETIC TAPE*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| R M T | | | X | A |
| | | | Y | I |

JY is the beginning of a 6-way Jump Table used for all Tape operations on a given file. RMT uses only four of the jumps.
Link in J14.

All or part of one block on magnetic tape is read into the A-area of memory. The information is automatically checked for 2-dimensional parity while the reading is being performed. Checking is for odd-parity at 60 kc and 40 kc, and for even-parity at 24 kc.

If the block contains an odd number of characters (as can occur with IBM-recorded tapes) a zero is automatically stored in memory to complete the last slab of the block.

I: Actual address (000-999) in memory of 3-slab Information Table, containing Handler-number and number of slabs to be read from the block.

| I | I+1 | I+2 |
|---|---|---|
| Ha | ← | N → |

Ha: Handler-number: 0-7
only RH 3 bits are used

N: Number of slabs: 0000-7999
only RH 15 bits are used

Since the Handler-number is referenced in the instruction, there is no explicit operation to select the Handler.

If the block is more than N slabs long, the first N slabs of the block are stored in memory, while the rest of the block is used only for checking but is not stored in memory. The entire block is always checked, and therefore the time to perform a partial read is the same as to perform a complete read.

If the block is N slabs long, or less, the entire block is stored in memory.

If N=0, the tape does not move; the instruction merely tests for BUSY and USE LOCKOUT.

INDEX:

Indexing a tape forward is accomplished by Reading with N = 1.

CM:

Control Mark. Any 1-slab record serves as a CM.

Conventions have been established for a number of CMs, as listed. The first three are used by **STEP**, and the programmer need be aware of them only to be sure that he never uses one of them for a CM which he may record himself.

| | |
|---|---|
| ←← | Skip Block follows |
| CC | Rescue Dump follows |
| TT | End of Tape |
| FF | End of File |
| HH | Hash-totals follow |

BRANCHES:

| | READ | WRITE | BACK | WIND |
|---|---|---|---|---|
| JY | Read Error | | | |
| JY+1 | | Write Error | | |
| JY+2 | CM | | | |
| JY+3 | | DWM | | |
| JY+4 | Busy Rewinding | | | |
| JY+5 | Write Lockout (WRITE only)<br>or Use Lockout | | | |

The complete Jump Table is shown, since the same table will normally be used for all operations on the same file.

Priority of branches:
- Use Lockout
- Busy
- Write Lockout
- Read or Write Error
- CM or DWM (Destination Warning Marker)

If two branch conditions arise simultaneously, the Overflow flag is set.

Processor hangs up if there is a Handler malfunction.

Since all error-detection, end-of-tape alternation, etc., are handled by **STEP** (Standard Tape Executive Program), and a single Magnetic Tape Jump Table for the entire program is constructed by ***NEAT*** (National's Electronic Autocoding Technique), the programmer normally remains unaware of the branch conditions. He provides only for end-of-file CM and for any special purposes for which he may choose to use CMs.

READ ERROR:

When the Processor detects a read-error, it still reads the full N slabs or the entire block, whichever is less,' into memory.

LOCKOUT:

One of the modes of REWIND will place a Use Lockout on the Handler. The Processor may not use this Handler again until the Use Lockout has been cleared by the operator, usually when he changes reels of tape.

R30: When the operation is complete, R30 contains the address of the first memory slab *following* the information read.

If N = 0, or if unable to read because of Busy or Use Lockout, (R30) remains unchanged.

OVERFLOW: If the Overflow flag is on, this instruction turns it off before the operation begins.

If the operation encounters a Read Error and a CM simultaneously, it takes the Read Error branch and sets Overflow.

ERROR HALT: Attempting to execute a READ after a WRITE on the same tape. No tape movement.

Attempting to READ the Trailer (ie- past the physical end of the tape). No tape movement.

Attempting to READ blank tape. Tape runs to the Trailer, then error halts.

## WRITE MAGNETIC TAPE

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| W M T | | | X | A |
| | | | Y | I |

JY is the beginning of a 6-way Jump Table used for all Tape operations on a given file. WMT uses only four of the jumps. Link in J14.

One block is written on magnetic tape from the A-area of memory. The information has 2-dimensional parity automatically added to it, then it is immediately read back from the tape, and the parities checked. The operation uses odd-parity at 60 kc and 40 kc, and even-parity at 24 kc.

I: Actual address (000-999) in memory of 3-slab Information Table, containing Handler-number and number of slabs to be written as a block.

| I | I+1 | I+2 |
|---|---|---|
| Ha | ← N | → |

Ha: Handler-number: 0-7
only RH 3 bits are used

N: Number of slabs: 0000-7999
only RH 15 bits are used

Since the Handler-number is referenced in the instruction, there is no explicit operation to select the Handler.

If N=0, the tape does not move; the instruction merely tests for BUSY and LOCKOUT.

CM:

If N=1, the resulting record is a CM. See definitions and discussion under RMT.

DWM:

Destination Warning Marker. Approximately 18 feet from the physical end of the tape (the Trailer) the Handler will encounter the DWM, indicating that the end of the tape is approaching; it is safe to finish the current block, and to record (if desired) a few more blocks of control information, but no more file information should be written on this reel.

Once the tape passes the DWM, subsequent WMT instructions will not take this branch. If the programmer does any writing beyond the DWM, it is his responsibility not to write so much that he reaches the trailer.

The DWM is significant only when writing; it is ignored when reading.

BRANCHES:

| | |
|---|---|
| (JY+1) | Write Error. |
| (JY+3) | DWM. |
| (JY+4) | Handler busy rewinding. |
| (JY+5) | Handler in Use Lockout or Write Lockout. |

See complete table, and discussion, under RMT.

LOCKOUT:

One of the modes of REWIND will place a Use Lockout on the Handler. The Processor may not use this Handler again until the Use Lockout has been cleared by the operator, usually when he changes reels of tape.

Each Handler is normally in a Write Lockout state, making it impossible to destroy information on a tape by accidently recording over it. When it is determined that a given reel is to be recorded, the operator releases the Lockout on that Handler; as soon as the reel is recorded, rewound and changed, the Lockout is automatically established again.

OVERFLOW: If the Overflow flag is on, this instruction turns it off before the operation begins.

If the operation encounters a Write Error and a DWM simultaneously, it takes the Write Error branch and sets Overflow.

ERROR HALT: Attempting to WRITE when positioned on the Trailer. No tape movement.

Processor hangs up if there is a Handler malfunction.

### *BACKUP Magnetic Tape one Block*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| B A C K | | | | |
| | | | Y | I |

JY is the beginning of a 6-way Jump Table used for all Tape operations on a given file. BACK uses only two of the jumps. Link in J14.

The designated magnetic tape is moved backward one block, so that it is now positioned to re-read or re-write the last block which was read or written.

I: Actual address (000-999) in memory of 1-slab Information Table (first slab of 3-slab table used for RMT or WMT) containing Handler-number.

| I |
|---|
| Ha |

Ha: Handler-number: 0-7
only RH 3 bits are used

Since the Handler-number is referenced in the instruction, there is no explicit operation to select the Handler.

BRANCHES:

(JY+4) Handler busy rewinding.
(JY+5) Handler is in Use Lockout.
See complete table, and discussion, under RMT.

OVERFLOW: If the Overflow flag is on, this instruction turns it off before the operation begins.

This instruction never sets Overflow.

ERROR HALT: Tape positioned on Trailer, Leader, or BIM (Beginning of Information Marker).

Processor hangs up if there is a Handler malfunction.

### *REWIND Magnetic Tape*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| W I N D | (V) | | | |
| | | | Y | I |

JY is the beginning of a 6-way Jump Table used for all Tape operations on a given file. WIND uses only two of the jumps. Link in J14.

WIND: Rewind the designated magnetic tape to the Beginning-of-Information Marker (BIM).

WIND:L Rewind, and set Use Lockout in the Handler.

I: Actual address (000-999) in memory of 1-slab Information Table (first slab of 3-slab table used for RMT or WMT) containing Handler-number.

| I |
|---|
| Ha |

Ha: Handler-number: 0-7
only RH 3 bits are used

Since the Handler-number is referenced in the instruction, there is no explicit operation to select the Handler.

BRANCHES:

(JY+4) Handler already busy rewinding.
(JY+5) Handler already in Use Lockout.
See complete table, and discussion, under RMT.

NOTE: If tape is already positioned on Leader or BIM, the operation does nothing.

OVERFLOW: If the Overflow flag is on, this instruction turns it off before the operation begins.

Processor hangs up if there is a Handler malfunction.

## READ CRAM CARD

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| R C C | | | X | A |
| | | | Y | I |

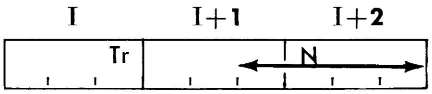
JY is the beginning of a 6-way Jump Table used for both Reading and Writing. RCC uses 4 of the jumps.
Link in J14.

The instruction automatically addresses itself to whichever CRAM was last selected by SELC, and reads one block from the card which is already on the drum.

Information is recorded on the card with one block in each of 7 tracks. Block length may be 1 to 1550 slabs.

All or part of one block on the card is read into the A-area of memory. The information is automatically checked for 2-dimensional parity while the reading is being performed.

I: Actual address (000-999) in memory of 3-slab Information Table containing track-number and number of slabs to be read from the block.



Tr: Track-number: 0-6
only RH 3 bits are used

N: Number of slabs: 0000-1550
only RH 15 bits are used

If the block is more than N slabs long, the first N slabs of the block are stored in memory, while the rest of the block is discarded. See comments under PARTIAL READ.

If the block is N slabs long, or less, the entire block is read.

If Tr=7, or if N=0, the instruction does nothing.

If N is greater than 1550, the Processor error-halts.

CM:

Control Mark. Any 1-slab block serves as a CM.

BRANCHES:

| | READ | WRITE |
|---|---|---|
| JY | Read Error | |
| JY+1 | | Write Error |
| JY+2 | CM | |
| JY+3 | Not Loaded | |
| JY+4 | Wrong Card | |
| JY+5 | | Write Lockout |

The complete Jump Table is shown, since the same table will be used for reading and writing.

Attempting to read a blank track (one which has never before been recorded) will cause the Processor to take the "wrong card" branch.

Priority of branches:
Write Lockout
Wrong Card
Not Loaded
Read or Write Error
CM

If two branch conditions arise simultaneously, the Overflow flag is set.

Since all error-detection and housekeeping are handled by **PACE** (PAckaged CRAM Executive), and a single CRAM Jump Table for the entire program is constructed by ***NEAT*** (National's Electronic Autocoding Technique), the programmer normally remains unaware of the branch conditions. He provides only for end-of-file CM, and even that only when using CRAM Cards as a serial file.

PARTIAL READ:

If the block is more than N slabs long, reading terminates with the Nth slab, and the Processor immediately proceeds with the program without waiting for the rest of the block. In this case, while each character read into memory is checked for its parity bit, the block-parity character is not verified.

When writing a block which is expected to be subject to a Partial Read, it is customary to place an additional slab in the block at the point where the partial reading will terminate. This slab contains a programmed check-sum of that part of the block which precedes it, and the check-sum is verified after the partial read. Standard subroutines are provided for creating and verifying this check-sum.

READ ERROR:

When the Processor detects a read-error, it still reads the full N slabs or the entire block, whichever is less, into memory.

R30: When the operation is complete, R30 contains the address of the first memory slab *following* the information read.

If Tr=7, or N=0, or if unable to read because of NOT LOADED or WRONG CARD, (R30) remains unchanged.

OVERFLOW: If the Overflow flag is on, this instruction turns it off before the operation begins.

If the operation encounters a Read Error and a CM simultaneously, it takes the Read Error branch and sets Overflow.

Processor error-halts if no CRAM has been selected; hangs up if CRAM not ready, or if power off.

See next page for
WRITE CRAM CARD

## *WRITE CRAM CARD*

| Op | V | L | X/Y | A/B |
|---|---|---|---|---|
| W C C | | | X | A |
| | | | Y | I |

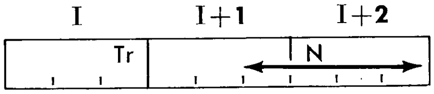
JY is the beginning of a 6-way Jump Table used for both Reading and Writing. WCC uses 4 of the jumps.
Link in J14.

The instruction automatically addresses itself to whichever CRAM was last selected by SELC, and writes one block on the card which is already on the drum.

Information is recorded on the card with one block in each of 7 tracks. Block length may be 1 to 1550 slabs.

One block is written in the designated track from the A-area of memory. The information has 2-dimensional parity automatically added to it, and is immediately read back from the card, and the parities checked.

I: Actual address (000-999) in memory of 3-slab Information Table containing track-number and number of slabs to be written as the record.



Tr: Track-number 0-6
only RH 3 bits are used

N: Number of slabs: 0000-1550
only RH 15 bits are used

After the block has been written, and the block-parity character recorded and checked, the Processor terminates the operation, and proceeds to the next instruction in the program. The CRAM, under its own independent control, then erases the rest of the track.

If Tr=7, or if N=0, the instruction does nothing.

If N is greater then 1550, the Processor error-halts.

CM:
If N=1, the resulting block is a CM.

BRANCHES:

| | |
|---|---|
| (JY+1) | Write Error |
| (JY+3) | Not Loaded |
| (JY+4) | Wrong Card |
| (JY+5) | Write Lockout |

See complete table, and discussion, under RCC.

LOCKOUT:
Each CRAM is normally in a Write Lockout state, making it impossible to destroy information on a card by accidently recording over it. When it is determined that a deck of cards is permitted to have new information recorded on it, the operator releases the Lockout on that CRAM; as soon as the deck is removed, and a new deck is placed in the CRAM, the Lockout is automatically established again.

OVERFLOW: If the Overflow flag is on, this instruction turns it off before the operation begins.

This instruction never sets Overflow.

Processor error-halts if no CRAM has been selected; hangs up if CRAM not ready, or if power off.

# APPENDIX

## PROCESSOR FORMAT OF COMMANDS

When a program has been translated by the NEAT Assembly or by the NEAT Compiler into actual processor code, each single-stage command occupies 2 slabs of memory, and each double-stage command occupies 4 slabs of memory.

The command format in the NCR 315 has been made very "tight" in order to make most economical use of memory space for programs. For this reason, all the bits of the command have been used as much as possible, and a given set of bits within the command format may have different significance in different commands.

The formats of processor commands may be represented as:

| | | | | | |
|---|---|---|---|---|---|
| Single Stage | X | x | F | C | A |

| | | | | | |
|---|---|---|---|---|---|
| Double Stage | X | x | F | C | A |
| | Y | y | Q | G | B |

The A and B fields in the commands are identical to A and B as described in this manual for each command, and need not be discussed further.

In a single-stage command, C represents the Operation Code (eg. ADD, SUB, etc) and F often represents the Field Length (reduced by 1). However, in those commands (such as TEST) where no field length is appropriate, F is used to specify additional variations of the operation.

In some cases also, where it is not appropriate to permit the designation of a "literal" operand, the designation of index register R15 (or optionally R31) is used to distinguish a whole new set of variations.

In a double-stage command, C is always either *zero* or *hyphen,* and F is the actual operation code. Q and G are used to specify variations.

Referring to the diagram of R-registers and J-registers on page 11, it will be seen that the 32 registers of each type, which are there numbered from 0 through 31, could just as well have been numbered—

0 through 15, left column,
0 through 15, right column.

They are actually so numbered in the processor format of a command. The symbols X and Y in the processor command format stand for "hexadecimal" configurations whose values are:

| CHARACTER | VALUE |
|---|---|
| 0 – 9 | 0 – 9 |
| @ | 10 |
| , | 11 |
| ⍁ | 12 |
| & | 13 |
| . | 14 |
| – | 15 |

In the single bit-positions marked x and y, left-column is indicated by a 0-bit and right-column by a 1-bit.

Recalling that the Operation Code for ADD is 3, and that F contains L-1, the following command:

| Op | V | L | X | A |
|---|---|---|---|---|
| A D D | | 6 | 1 0 | 7 8 9 |

would appear in processor format as:

| | |
|---|---|
| @ 5 3 | 7 8 9 |

If the command has called for R26 (register 10, right-column) it would have appeared as:

| | |
|---|---|
| @ & 3 | 7 8 9 |

The following is the complete list of commands in the 315 vocabulary, showing the entries for F, C, Q, G. Wherever there is no entry for F, it is L-1. Wherever there is no entry for Q or G, the corresponding bit-positions in the command are irrelevant to its execution. But wherever values for F, Q, or G are shown, then any value other than those listed will cause the processor to halt on "programmer error." A double asterisk indicates that this is an R15 variation, in which F has an entirely different significance than if some other index register is named. As elsewhere, an asterisk preceding the command indicates that a literal operand is permitted by naming R15.

| CODE | F | C | Q | G |
|---|---|---|---|---|
| *LD | | 1 | | |
| *ST | | 2 | | |
| *ADD | | 3 | | |
| *SUB | | 4 | | |
| *MULT | | 5 | | |
| *COMP | | 6 | | |
| TEST : G | 0 | 7 | | |
| TEST : SR | 0 | 7 | | |
| TEST : L | 1 | 7 | | |
| TEST : SM | 1 | 7 | | |
| TEST : E | 2 | 7 | | |
| TEST : SL | 2 | 7 | | |
| TEST : – | 3 | 7 | | |
| TEST : σ | 4 | 7 | | |
| JUMP | 5 | 7 | | |
| TEST : D | 6 | 7 | | |
| TEST : T | 7 | 7 | | |
| **DLR | 0 | 7 | | |
| **SETF : + | 1 | 7 | | |
| **SETF : σ | 2 | 7 | | |
| **SETF : – | 3 | 7 | | |
| **SETF : D | 6 | 7 | | |
| **SETF : T | 7 | 7 | | |
| *SHFT : AR | 0 | 8 | | |
| *SHFT : DR | 1 | 8 | | |
| *SHFT : RR | 2 | 8 | | |
| *SHFT : DL | 3 | 8 | | |
| *SHFT : RC | 4 | 8 | | |
| *SHFT : LC | 6 | 8 | | |
| *SHFT : AL | 7 | 8 | | |
| *ADD : M | | 9 | | |
| *BADD | | @ | | |
| *DIV | | , | | |

| | CODE | F | C | Q | G |
|---|---|---|---|---|---|
| Single Stage | *SETF : LH | 2 | ¢ | | |
| | *SETF : RH | 3 | ¢ | | |
| | *CLRF :LH | 4 | ¢ | | |
| | *CLRF :RH | 5 | ¢ | | |
| | JUMP : I | 6 | ¢ | | |
| | JUMP :IP | 7 | ¢ | | |
| | **MLRA | 6 | ¢ | | |
| | **SKIP | 7 | ¢ | | |
| | EDIT | | & | | |
| | SUPP | | . | | |
| Double Stage | *TEST : LH | 0 | ¢ | | |
| | *TEST : RH | 1 | ¢ | | |
| | *CNT | 1 | 0 | | |
| | LD :R | 2 | 0 | 0 | 0 |
| | LD :J | 2 | 0 | 0 | 1 |
| | SLD :R | 2 | 0 | 0 | 2 |
| | SLD :J | 2 | 0 | 0 | 3 |
| | MOVE :RR | 2 | 0 | 0 | 4 |
| | MOVE :JR | 2 | 0 | 0 | 5 |
| | MOVE :RJ | 2 | 0 | 0 | 6 |
| | MOVE :JJ | 2 | 0 | 0 | 7 |
| | ST :R | 2 | 0 | 0 | 8 |
| | ST :J | 2 | 0 | 0 | 9 |
| | AUG :R | 2 | 0 | 1 | 0 |
| | AUG :J | 2 | 0 | 1 | 1 |
| | SAUG :R | 2 | 0 | 1 | 2 |
| | SAUG :J | 2 | 0 | 1 | 3 |
| | MOVE :B | 3 | 0 | 0 | |
| | MOVE :E | 3 | 0 | 1 | |
| | **SPRD :B | 3 | 0 | 0 | |
| | **SPRD :E | 3 | 0 | 1 | |
| | SCND :G$v$ | 4 | 0 | $v$ | 1 |
| | SCND :L$v$ | 4 | 0 | $v$ | 2 |

| CODE | F | C | Q | G |
|---|---|---|---|---|
| SCND :E$v$ | 4 | 0 | $v$ | 4 |
| SCNA :G$n$ | 4 | 0 | $m$ | 9 |
| SCNA :L$n$ | 4 | 0 | $m$ | @ |
| SCNA :E$n$ | 4 | 0 | $m$ | ¢ |
| LDAD | 5 | 0 | 0 | |
| LDAD :XR | 5 | 0 | 1 | |
| LDAD :XL | 5 | 0 | 2 | |
| LDAD :XB | 5 | 0 | 3 | |
| STDA | 6 | 0 | 0 | |
| PAST :XR | 6 | 0 | 1 | |
| PAST :XL | 6 | 0 | 2 | |
| PAST :XB | 6 | 0 | 3 | |
| SELC :DN | 7 | 0 | 0 | 0 |
| SELC :DP | 7 | 0 | 0 | 1 |
| *SELC :R | 7 | 0 | 0 | 2 |
| *SELC :T | 7 | 0 | 0 | 3 |
| *TEST :SW | 7 | 0 | 1 | 0 |
| *SELP | 7 | 0 | 1 | 1 |
| *SELS | 7 | 0 | 1 | 2 |
| *SELQ | 7 | 0 | 1 | 3 |
| *CLRU :C | 7 | 0 | 2 | 0 |
| *CLRU :P | 7 | 0 | 2 | 1 |
| *CLRU :S | 7 | 0 | 2 | 2 |
| *CLRU :Q | 7 | 0 | 2 | 3 |
| *SETU :C | 7 | 0 | 3 | 0 |
| *SETU :P | 7 | 0 | 3 | 1 |
| *SETU :S | 7 | 0 | 3 | 2 |
| *SETU :Q | 7 | 0 | 3 | 3 |
| HALT :D | 1 | — | 0 | |
| HALT :A | 1 | — | 1 | |
| TYPE :D | 1 | — | 2 | |
| TYPE :A | 1 | — | 3 | |

| CODE | F | C | Q | G |
|---|---|---|---|---|
| PPT :C | 1 | — | 4 | |
| PPT :S | 1 | — | 5 | |
| TYPE :AP | 1 | — | 7 | |
| RPT :S | 2 | — | 1 | |
| RPT :C | 2 | — | 2 | |
| RPT :CX | 2 | — | 3 | |
| RCOL :F | 2 | — | 4 | |
| RCOL | 2 | — | 5 | |
| RCOL :TF | 2 | — | 6 | |
| RCOL :T | 2 | — | 7 | |
| STRT :S | 3 | — | 0 | |
| RCK | 3 | — | 1 | |
| PKT | 3 | — | 2 | |
| STOP :S | 3 | — | 3 | |
| PRNT | 3 | — | 4 | |
| PNCH | 3 | — | 4 | |
| RMT | 4 | — | 0 | $g$ |
| WMT | 4 | — | 1 | $g$ |
| BACK | 4 | — | 2 | $g$ |
| WIND | 4 | — | 3 | $g$ |
| WIND :L | 4 | — | 4 | $g$ |
| RCC | 5 | — | 0 | $g$ |
| WCC | 5 | — | 1 | $g$ |
| RQ | 6 | — | 0 | |
| WQ | 6 | — | 1 | |

LEGEND:
*R = 15 or ≠ 15
**R = 15
$v$ = 1-7
$n$ = 1,2,3 $m$ = 1,4,7
$g$: "Thousands" digit for I

| | | |
|---|---|---|
| ABSOLUTE | X xF C | A |
| FORMAT | Y yQ G | B |

| PRINTER REPRESENTATION OF "NON-PRINTING" CHARACTERS | OVERPRINTED WITH A "+" SYMBOL | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | M | N | O | P | U | V | W | X |
| | ? | : | ← | ↑ | ' | [ | ] | \ |

| xF or yQ | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | @ | , | ¢ | & | . | — |

| X or Y | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ | , | ¢ | & | . | — |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| REGISTER NUMBER | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

## PAPER TAPE INPUT

Paper tape for input of *programs* to the NCR 315 must be punched in NCR General-Purpose Code, illustrated on the next page. All the facilities of NEAT (Assembler, Compiler, COBOL, Librarian, etc) expect the source program to be punched in this code.

Paper tape for input of *data* may be punched in any code whatever, and read by a macro-instruction that translates the character-images on the tape into 315 internal configurations. Certain codes have been designated as standard, and translators have been written for them; these are provided as macros in the NEAT Compiler, and as subroutines for the NEAT Assembler. These standard codes are shown on the succeeding pages.

The macros operate by reading and translating one character at a time, for as many characters as the programmer has specified. They utilize the sharable time between the characters, and perform their functions while the Reader operates at full speed.

> Programmers sometimes wish to specify that data be punched in 315 binary-image code, which needs no translation, in order to use the time between characters for other processing. This advantage is illusory, since when reading one character at a time for this purpose, a subroutine is still needed to store successive characters alternately LH and RH in successive memory slabs. As a result, the units of time available between characters for other processing are too small to be of practical use.

When fields, or items, or batches, are of varying lengths the programmer may save himself some effort by specifying the use of *control* configurations as data markers. When he specifies a number-of-characters to the macro, this is really a maximum size for the "gulp," since the macro will terminate the gulp if a control configuration is found before the specified number of characters have been read. Thus each field, or each record, would be read in isolation regardless of its length, and would always start at the same position in memory.

The macros set program indicators, which may be tested by the main program, to show: whether the gulp ran to the specified maximum length or was terminated by a control configuration, how many characters the gulp contained, and whether any parity error or undefined configuration was found in the tape.

## PAPER TAPE OUTPUT

A similar group of macros is available for punching information in any of the standard codes. These also operate on a character-for-character basis.

Because there is quite a substantial amount of free time between characters when punching, the macros return control to the main program after each character. Further computation may be performed during free time, and the main program permits the macro to resume for each additional character to be punched.

On occasion, information will be punched by the Processor, for the sole purpose of returning that information to the Processor at a later time. (For example: for transfer of low-volume data from one Processor to another, paper tape is more economical to use and to ship than punched cards.) For this purpose, the programmer may punch the tape in 315 binary-image, using the Processor instruction PPT:C, with any convenient N in the instruction. When this tape is later read, the Processor instruction RPT:C, with any convenient N, would be used. In neither of these cases would translation be required, and no macro need be used.

# PAPER TAPE INPUT-OUTPUT

## NCR GENERAL-PURPOSE CODE

(Odd-Parity)

| TYPEWRITER CHARACTER | | CHANNELS | | | | | | | | | IMAGE IN 315 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LOWER SHIFT | UPPER SHIFT | 8 | 7 | 6 | 5 | 4 | • | 3 | 2 | 1 | |
| 0 | 0 | | ● | | ● | | • | ● | ● | ● | G |
| 1 | = | | ● | | ● | ● | • | | ● | ● | " |
| 2 | # | | ● | ● | | | • | ● | ● | ● | P |
| 3 | ! | | ● | | | | • | | ● | ● | 3 |
| 4 | $ | | | | ● | | • | ● | | ● | E |
| 5 | % | | ● | ● | | | • | | | ● | J |
| 6 | & | | ● | ● | | ● | • | | ● | ● | = |
| 7 | " | | ● | | | ● | • | ● | ● | ● | – |
| 8 | ( | | | | | ● | • | ● | | ● | & |
| 9 | ) | | | ● | ● | | • | | | ● | # |
| A | A | | ● | | | | • | ● | ● | | 6 |
| B | B | | | ● | ● | | • | | ● | | S |
| C | C | | | | ● | ● | • | ● | | | ? |
| D | < | | ● | | ● | | • | | ● | | B |
| E | E | | | | | | • | | ● | | 2 |
| F | F | | | | ● | ● | • | | ● | | ; |
| G | G | | | ● | ● | | • | ● | | | U |
| H | * | | ● | ● | | ● | • | | | | Q |
| I | • | | ● | | | ● | • | ● | | | sp |
| J | + | | | | ● | | • | ● | ● | | F |
| K | – | | ● | | ● | ● | • | ● | ● | | ← |
| L | , | | ● | ● | | | • | ● | | | M |
| M | ? | | | ● | ● | ● | • | | | | Y |
| N | : | | ● | | ● | ● | • | | | | H |
| ɵ | ← | | ● | ● | ● | | • | | | | * |
| P | ↑ | | | ● | | ● | • | ● | | | $ |
| Q | Q | | ● | ● | | ● | • | ● | ● | | ) |
| R | R | | ● | | ● | | • | ● | | | D |
| S | > | | ● | | | ● | • | | ● | | @ |
| T | / | | | ● | | | • | | | | + |

| | TYPEWRITER CHARACTER | | CHANNELS | | | | | | | | | IMAGE IN 315 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | LOWER SHIFT | UPPER SHIFT | 8 | 7 | 6 | 5 | 4 | • | 3 | 2 | 1 | |
| | U | ' | | | | | ● | • | ● | ● | | • |
| | V | [ | | ● | ● | ● | | • | ● | ● | | W |
| SYMBOL ROW | W | ] | | | ● | | | • | ● | ● | | ɵ |
| | X | \ | | ● | ● | ● | ● | • | | ● | | < |
| | Y | ; | | | ● | | ● | • | | ● | | % |
| | Z | @ | | ● | ● | | | • | | ● | | K |
| 4 | SPACE | SPACE | | | | | ● | • | | | | 8 |
| 8 | COMP<br>ꓘ ① | COMP<br>ꓘ ① | | | | ● | ● | • | ● | ● | ● | ↑ |
| 7 | CLEAR<br>⊗ ① | CLEAR<br>⊗ ① | | | ● | ● | | • | ● | ● | ● | X |
| 6 | PUT<br>ʎ ① | PUT<br>ʎ ① | | | | | ● | • | | ● | ● | , |
| 5 | PDISC<br>Ɉ ① | PDISC<br>Ɉ ① | | ● | | | | • | ● | | ● | 5 |
| 9 | STOP<br>ə ① | STOP<br>ə ① | | ● | ● | ● | ● | • | ● | | | ' |
| 3 | CARET<br>① | CARET<br>① | | | | ● | | • | | | | ! |
| 2 | TAB<br>① | TAB<br>① | | | | | | • | | | ● | 1 |
| 1 | LINE<br>ȝ ① | LINE<br>ȝ ① | | | | | | • | ● | | | 4 |
| 0 | RUNIN<br>② | RUNIN<br>② | ● | | ● | ● | ● | • | ● | ● | ● | TAPE FEED |
| | DELE<br>② | DELE<br>② | | ● | ● | ● | ● | • | ● | ● | ● | \ |
| | UP SHIFT | UP SHIFT | | | ● | ● | ● | • | ● | ● | | ] |
| | DOWN SHIFT | DOWN SHIFT | | | ● | ● | ● | • | ● | | ● | [ |

## SEPARATE CODES
### FOR NUMERIC KEYBOARDS

| AMOUNT KEY | CHANNELS | | | | | | | | | IMAGE IN 315 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 8 | 7 | 6 | 5 | 4 | • | 3 | 2 | 1 | |
| 0 | | ● | ● | | ● | • | ● | | ● | ( |
| 1 | | | ● | ● | ● | • | | ● | ● | > |
| 2 | | | | ● | | • | | ● | ● | C |
| 3 | | ● | | ● | ● | • | ● | | ● | : |
| 4 | | | | | | • | ● | ● | ● | 7 |
| 5 | | | ● | | ● | • | ● | ● | ● | / |
| 6 | | | ● | | | • | | ● | ● | L |
| 7 | | | | ● | ● | • | | | ● | I |
| 8 | | ● | ● | ● | ● | • | | | ● | Z |
| 9 | | ● | | ● | | • | | | ● | A |

NOTE: This code is compatible with Addressograph-Graphotype.

① STOP. Terminates the RPT Macro.

② SKIP. Ignored by the RPT Macro.

# PAPER TAPE INPUT-OUTPUT

## I. B. M. 046-047 8-CHANNEL CODE

(Odd-Parity)

| CHARACTERS | | CHANNELS | | | | | | | | | | IMAGE IN 315 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 046-047 | 315 | NCR | 8 | 7 | 6 | 5 | 4 | • | 3 | 2 | 1 | |
| | | IBM | EL | X | 0 | \ | 8 | • | 4 | 2 | 1 | |
| 0 | 0 | | | | ● | | | • | | | | 0+ |
| 1 | 1 | | | | | | | • | | | ● | 01 |
| 2 | 2 | | | | | | | • | | ● | | 02 |
| 3 | 3 | | | | | ● | | • | | ● | ● | 0C |
| 4 | 4 | | | | | | | • | ● | | | 04 |
| 5 | 5 | | | | | ● | | • | ● | | ● | 0E |
| 6 | 6 | | | | | ● | | • | ● | ● | | 0F |
| 7 | 7 | | | | | | | • | ● | ● | ● | 07 |
| 8 | 8 | | | | | | ● | • | | | | 08 |
| 9 | 9 | | | | | ● | ● | • | | | ● | 0I |
| A | A | | | ● | ● | | | • | | | ● | 1J |
| B | B | | | ● | ● | | | • | | ● | | 1K |
| C | C | | | ● | ● | ● | | • | | ● | ● | 1T |
| D | D | | | ● | ● | | | • | ● | | | 1M |
| E | E | | | ● | ● | ● | | • | ● | | ● | 1V |
| F | F | | | ● | ● | ● | | • | ● | ● | | 1W |
| G | G | | | ● | ● | | | • | ● | ● | ● | 1P |
| H | H | | | ● | ● | | ● | • | | | | 1Q |
| I | I | | | ● | ● | ● | ● | • | | | ● | 1Z |
| J | J | | | ● | | ● | | • | | | ● | 1A |
| K | K | | | ● | | ● | | • | | ● | | 1B |
| L | L | | | ● | | | | • | | ● | ● | 13 |
| M | M | | | ● | | ● | | • | ● | | | 1D |
| N | N | | | ● | | | | • | ● | | ● | 15 |
| O | O | | | ● | | | | • | ● | ● | | 16 |
| P | P | | | ● | | ● | | • | ● | ● | ● | 1G |
| Q | Q | | | ● | | ● | ● | • | | | | 1H |
| R | R | | | ● | | | ● | • | | | ● | 19 |
| S | S | | | | ● | ● | | • | | ● | | 0S |
| T | T | | | | ● | | | • | | ● | ● | 0L |
| U | U | | | | ● | ● | | • | ● | | | 0U |
| V | V | | | | ● | | | • | ● | | ● | 0N |
| W | W | | | | ● | | | • | ● | ● | | 0O |

| CHARACTERS | | CHANNELS | | | | | | | | | | IMAGE IN 315 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 046-047 | 315 | NCR | 8 | 7 | 6 | 5 | 4 | • | 3 | 2 | 1 | |
| | | IBM | EL | X | 0 | \ | 8 | • | 4 | 2 | 1 | |
| X | X | | | | ● | ● | | • | ● | ● | ● | 0X |
| Y | Y | | | | ● | ● | ● | • | | | | 0Y |
| Z | Z | | | | ● | | ● | • | | | ● | 0R |
| . | . | | | ● | ● | | ● | • | | ● | ● | 1= |
| , | , | | | | ● | ● | ● | • | | ● | ● | 0> |
| SPACE | SPACE | | | | | ● | | • | | | | 0 ! |
| - | - | | | ● | | | | • | | | | 10 |
| & | & | | | ● | ● | ● | | • | | | | 1* |
| @ | @ | | | | | ● | ● | • | ● | | | 0? |
| ⌑ | ; | | | ● | ● | ● | ● | • | ● | | | 1' |
| % | % | | | | ● | | ● | • | ● | | | 0$ |
| $ | $ | | | ● | | ● | ● | • | | ● | ● | 1" |
| / | / | | | | ● | ● | | • | | | ● | 0# |
| * | * | | | ● | | | ● | • | ● | | | 1 sp |
| # | # | | | | | | ● | • | | ● | ● | 0 ’ |
| PI1 | ① | | | | | ● | ● | • | | ● | | 0 ; |
| PI2 | ① | | | ● | | | ● | • | | ● | | 1@ |
| PI3 | ① | | | | ● | | ● | • | | ● | | 0% |
| PI4 | ① | | | | ● | ● | ● | • | ● | | ● | 0[ |
| PI5 | ① | | | ● | ● | | ● | • | ● | | ● | 1 ( |
| PI6 | ① | | | ● | | ● | ● | • | ● | | ● | 1 : |
| PI7 | ① | | | | | | ● | • | ● | | ● | 0& |
| EC1 | ① | | | | | | ● | • | ● | ● | | 0 • |
| EC2 | ① | | | | ● | | ● | • | ● | ● | ● | 0 / |
| SP1 | ① | | | ● | ● | ● | ● | • | | ● | | 1< |
| SP2 | ① | | | ● | ● | | ● | • | ● | ● | | 1 ) |
| ENDLI | ① | | ● | | | | | • | | | | 20 |
| ERROR | ① | | | ● | | | ● | • | ● | ● | ● | 1 – |
| SKIP | ① | | | | ● | ● | ● | • | ● | ● | | 0] |
| RUNIN | ② | | | ● | ● | ● | ● | • | ● | ● | ● | 1\ |
| CORR | ① | | | | | ● | ● | • | ● | ● | ● | 0↑ |
| CARET | ① | | | ● | | ● | ● | • | ● | ● | | 1← |

① STOP. Terminates the RPT Macro.

② SKIP. Ignored by the RPT Macro.

# PAPER TAPE INPUT-OUTPUT

## 5-CHANNEL TELEGRAPH CODE

(Non-Parity)

| CHARACTERS | | CHARACTERS | | CHANNELS | | | | | | IMAGE IN 315 |
|---|---|---|---|---|---|---|---|---|---|---|
| TELEGRAPH LETTER SHIFT | 315 | TELEGRAPH FIGURE SHIFT | 315 | NCR 5 | 4 | • | 3 | 2 | 1 | |
| | | | | TEL. 1 | 2 | • | 3 | 4 | 5 | |
| A | A | - | - | ● | ● | • | | | | H |
| B | B | ? | COMP ꓘ ① | ● | | • | | ● | ● | C |
| C | C | : | + | | ● | • | ● | ● | | . |
| D | D | $ | $ | ● | | • | | ● | | B |
| E | E | 3 | 3 | ● | | • | | | | ! |
| F | F | ! | PUT ʎ ① | ● | | • | ● | ● | | F |
| G | G | & | & | | ● | • | | ● | ● | , |
| H | H | = | # | | | • | ● | | ● | 5 |
| I | I | 8 | 8 | | ● | • | ● | | | space |
| J | J | ' | * | ● | ● | • | | ● | | ; |
| K | K | ( | ( | ● | ● | • | ● | ● | | ← |
| L | L | ) | ) | | ● | • | | | ● | 9 |
| M | M | . | . | | | • | ● | ● | ● | 7 |
| N | N | , | , | | | • | ● | ● | | 6 |
| O | O | 9 | 9 | | | • | | ● | ● | 3 |
| P | P | 0 | 0 | | ● | • | ● | | ● | & |

| CHARACTERS | | CHARACTERS | | CHANNELS | | | | | | IMAGE IN 315 |
|---|---|---|---|---|---|---|---|---|---|---|
| TELEGRAPH LETTER SHIFT | 315 | TELEGRAPH FIGURE SHIFT | 315 | NCR 5 | 4 | • | 3 | 2 | 1 | |
| | | | | TEL. 1 | 2 | • | 3 | 4 | 5 | |
| Q | Q | 1 | 1 | ● | ● | • | ● | | ● | : |
| R | R | 4 | 4 | | ● | • | | ● | | @ |
| S | S | BELL | @ | ● | | • | ● | | | D |
| T | T | 5 | 5 | | | • | | | ● | 1 |
| U | U | 7 | 7 | ● | ● | • | ● | | | ? |
| V | V | ; | ; | | ● | • | ● | ● | ● | – |
| W | W | 2 | 2 | ● | ● | • | | | ● | I |
| X | X | / | / | ● | | • | ● | ● | ● | G |
| Y | Y | 6 | 6 | ● | | • | ● | | ● | E |
| Z | Z | " | " | ● | | • | | | ● | A |
| SPACE | SPACE | SPACE | SPACE | | | • | ● | | | 4 |
| FIGURE SHIFT | FIGURE SHIFT | FIGURE SHIFT | FIGURE SHIFT | ● | ● | • | | ● | ● | " |
| LETTER SHIFT | LETTER SHIFT | LETTER SHIFT | LETTER SHIFT | ● | ● | • | ● | ● | ● | ↑ |
| RUNIN | ② | RUNIN | ② | | | • | | | | 0 |
| LINE | ① | LINE | ① | | ● | • | | | | 8 |
| CARET | ① | CARET | ① | | | • | | ● | | 2 |

① STOP. Terminates the RPT Macro.

② SKIP. Ignored by the RPT Macro.

## PAPER TAPE INPUT-OUTPUT

### NCR 304 TYPEWRITER CODE

(Odd-Parity)

| TYPEWRITER KEY | | CHANNELS | | | | | | | | | IMAGE IN 315 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LOWER SHIFT | UPPER SHIFT | 8 | 7 | 6 | 5 | 4 | • | 3 | 2 | 1 | |
| 0 | ) | | ● | | | ● | • | | ● | | (a |
| 1 | 1 | | | | | | • | | | ● | 1 |
| 2 | ! (=) | | | | | | • | | ● | | 2 |
| 3 | # | | ● | | | | • | | ● | ● | 3 |
| 4 | $ | | | | | | • | ● | | | 4 |
| 5 | % | | ● | | | | • | ● | | ● | 5 |
| 6 | / | | ● | | | | • | ● | ● | | 6 |
| 7 | & | | | | | | • | ● | ● | ● | 7 |
| 8 | * | | | | | ● | • | | | | 8 |
| 9 | ( | | ● | | | ● | • | | | ● | 9 |
| A | A | | ● | | ● | | • | | | ● | A |
| B | B | | ● | | ● | | • | | ● | | B |
| C | C | | | | ● | | • | | ● | ● | C |
| D | d(<) | | ● | | ● | | • | ● | | | D |
| E | E | | | | ● | | • | ● | | ● | E |
| F | F | | | | ● | | • | ● | ● | | F |
| G | G | | ● | | ● | | • | ● | ● | ● | G |
| H | H | | ● | | ● | ● | • | | | | H |
| I | I | | | | ● | ● | • | | | ● | I |
| J | J | | ● | ● | | | • | | | ● | J |
| K | K | | ● | ● | | | • | | ● | | K |
| L | L | | | ● | | | • | | ● | ● | L |
| M | m(?) | | ● | ● | | | • | ● | | | M |
| N | n(:) | | | ● | | | • | ● | | ● | N |
| Ơ | ơ(←) | | | ● | | | • | ● | ● | | Ơ |
| P | p(↑) | | ● | ● | | | • | ● | ● | ● | P |

| TYPEWRITER KEY | | CHANNELS | | | | | | | | | IMAGE IN 315 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LOWER SHIFT | UPPER SHIFT | 8 | 7 | 6 | 5 | 4 | • | 3 | 2 | 1 | |
| Q | Q | | ● | ● | | ● | • | | | | Q |
| R | R | | | ● | | ● | • | | | ● | R |
| S | s(>) | | | ● | ● | | • | | ● | | S |
| T | T | | ● | ● | ● | | • | | ● | ● | T |
| U | u(') | | | ● | ● | | • | ● | | | U |
| V | v([) | | ● | ● | ● | | • | ● | | ● | V |
| W | w(]) | | ● | ● | ● | | • | ● | ● | | W |
| X | x(\) | | | ● | ● | | • | ● | ● | ● | X |
| Y | Y | | | ● | ● | ● | • | | | | Y |
| Z | Z | | ● | ● | ● | ● | • | | | ● | Z |
| @ | + | | | ● | | | • | | | | + |
| ¢(!) | □(;) | | ● | ● | ● | | • | | | | * |
| - | △(") | | | | ● | | • | | | | ! |
| . | , | | | | | ● | • | ● | ● | | . |
| SPACE | SPACE | | ● | | | ● | • | ● | | | space |
| PUT ↗ ① | PUT ↙ ① | | | | | ● | • | ● | | ● | & |
| COMP > ① | STOP ⊐ ① | | ● | ● | | ● | • | ● | | ● | ( |
| CLEAR ⊠ ① | CLEAR ⊠ ① | | ● | | | ● | • | ● | ● | ● | − |
| UP SHIFT | UP SHIFT | | ● | | ● | ● | • | | ● | ● | " |
| DOWN SHIFT | DOWN SHIFT | | | | ● | ● | • | | ● | | ; |
| DELE ② | DELE ② | | ● | ● | ● | ● | • | ● | ● | ● | \ |
| RUNIN ② | RUNIN ② | | ● | | | | • | | | | 0 |
| CARET ① | CARET ① | | | | ● | ● | • | ● | ● | ● | ↑ |
| TAB ① | TAB ① | | ● | | ● | ● | • | ● | ● | | ← |
| BACKSPACE | | DOES NOT PUNCH | | | | | | | | | |

① STOP. Terminates the RPT Macro.

② SKIP. Ignored by the RPT Macro.

## PROGRAMMING THE BUFFERED PRINTER

The following conventions have been adopted for punching the VFU (Vertical Format Unit) control tape on the Printer, assuming a page 11 inches high.

1. A punch in the P-column on VFU line 27. After loading paper in the Printer's rear tractors, the operator presses the SET LOOP button, advancing the paper into the front tractors and moving the VFU tape to the P-Punch. Then he moves the paper so that the page-perforation is aligned with the Printer's perforation guide. The paper is now aligned with the VFU tape, so that line 1 on the tape corresponds to the first printing line on the page.
2. Punch configurations 2, 4, 6, 8 on VFU lines 2, 4, 6, 8 respectively.
3. Punch "special" (4 holes) on VFU line 61.
4. Punch configuration 1 on VFU line 65.

The requirements of specific printed forms, or the use of a page longer or shorter than 11 inches (66 lines), may indicate the placing of "special" and "1" at somewhat different positions; any *other* configurations may be punched elsewhere on the tape to provide additional format control. These changes will still permit the VFU loop to be used for Compiler listings and other service routine printouts.

However, the loop may not be used as standard for these purposes if any of the specified configurations is repeated elsewhere on the VFU tape.

Do not use *period* as a value for F, as this is reserved for possible special function in future model Printers.

The first printed line on each page uses slew control 032, 034, 036, 038 as appropriate. Succeeding lines use 03**x** or 02**x**, depending on whether **x** is a recognition code or a number of lines.

The last normal line on the page will correspond to "special," while a line of Totals (for example) will correspond to the line punched "1".

NOTE: The programmer must use caution in calling for a paper slew from above "special," direct to the next page. If the Printer is out of paper, this condition will not be known to the program until the attempt to print the *second* line on the non-existent next page. The programmer must, therefore, arrange to save the first line of each page, and return to the command that prints it when resuming after the paper has been replenished.

As an alternative, the programmer may wish to print a blank line (all spaces) with slew control 031 at the bottom of each page. Then the *first* line on the next page will discover an out-of-paper condition. The programmer must determine, for each printing job, whether the additional printing time for this blank line on every page is warranted by the modest programming convenience.

Consider the printing of invoices, department store bills, bank statements, or similar forms with variable number of line-items. (See the accompanying flow chart.)

A. One of the line-items (not the last on the bill) falls opposite "special". The print command for the *next* line-item takes JY branch without printing or moving paper.

   Print page subtotals if desired, slew control 011. If out of paper, this command takes JY+1 after execution.

   Print first line of abbreviated heading on next page, using 032. If out of paper and subtotals not printed, this command takes JY+1 branch without execution. Print any additional lines of heading; then repeat the aborted line-item using appropriate slew control to place it properly on the page; then resume printing line-items.

B. Last line-item of the bill is in the middle of the page. Print totals (if desired) using 011. This command will not branch in this circumstance; mode "1" slew is used because of case C.

   Then print the next bill using 032 (or 034, etc) for the first line. If totals were printed, this command will take JY branch (since previous line passed "special") and is programmed to repeat the command. If previous page was the last sheet, and Printer is now out of paper, this command will take JY+1 the *second* time it is executed.

   If totals were not printed, then the command to print the first line on this next bill will not branch, but the command to print the second line will branch. Observe that if the printer is out of paper, the first line for this next bill will have been "lost," since the out-of-paper condition is not known to the program till it attempts to print the second line. Therefore the programmer must save the first line in memory, so that it can be printed when paper has been reloaded.

C. Last line-item of the bill falls opposite "special." Procedure is same as case B, but note that now the print command for totals *will* take JY branch and be repeated; and if out of paper, it will take JY+1 the *second* time it is executed. The print command for first line of next heading will not branch in this case.

If totals are not printed, then the print command for first line of next heading will take JY and be repeated, then take JY+1.

Some of the print commands on the accompanying flow chart show the possibility of both JY and JY+1 exits. In each of these cases, the JY branch repeats the command, and JY+1 is therefore possible when the command is executed the second time.

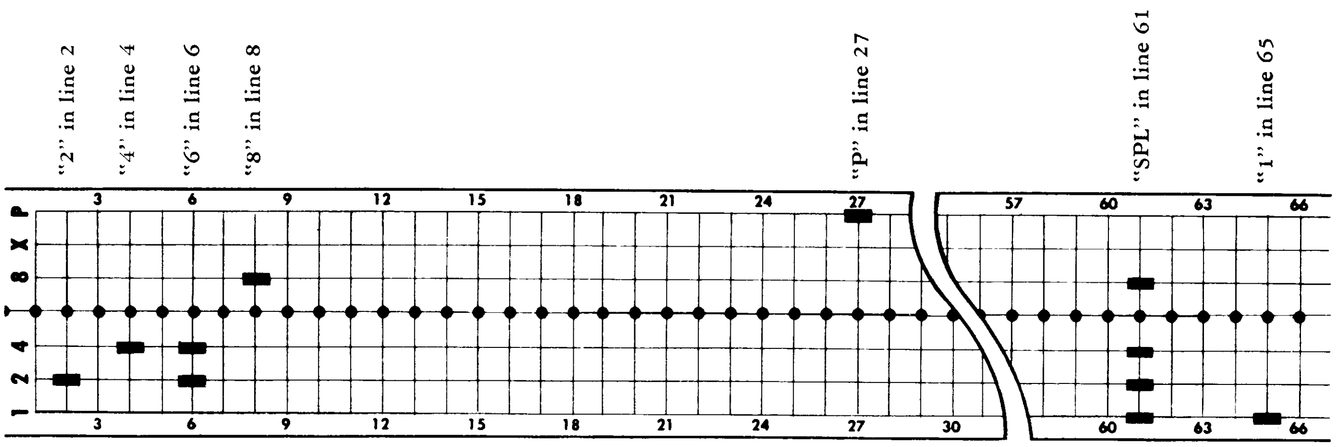
Remember that no print command can take JY+1 unless the previously-executed print command has taken JY.

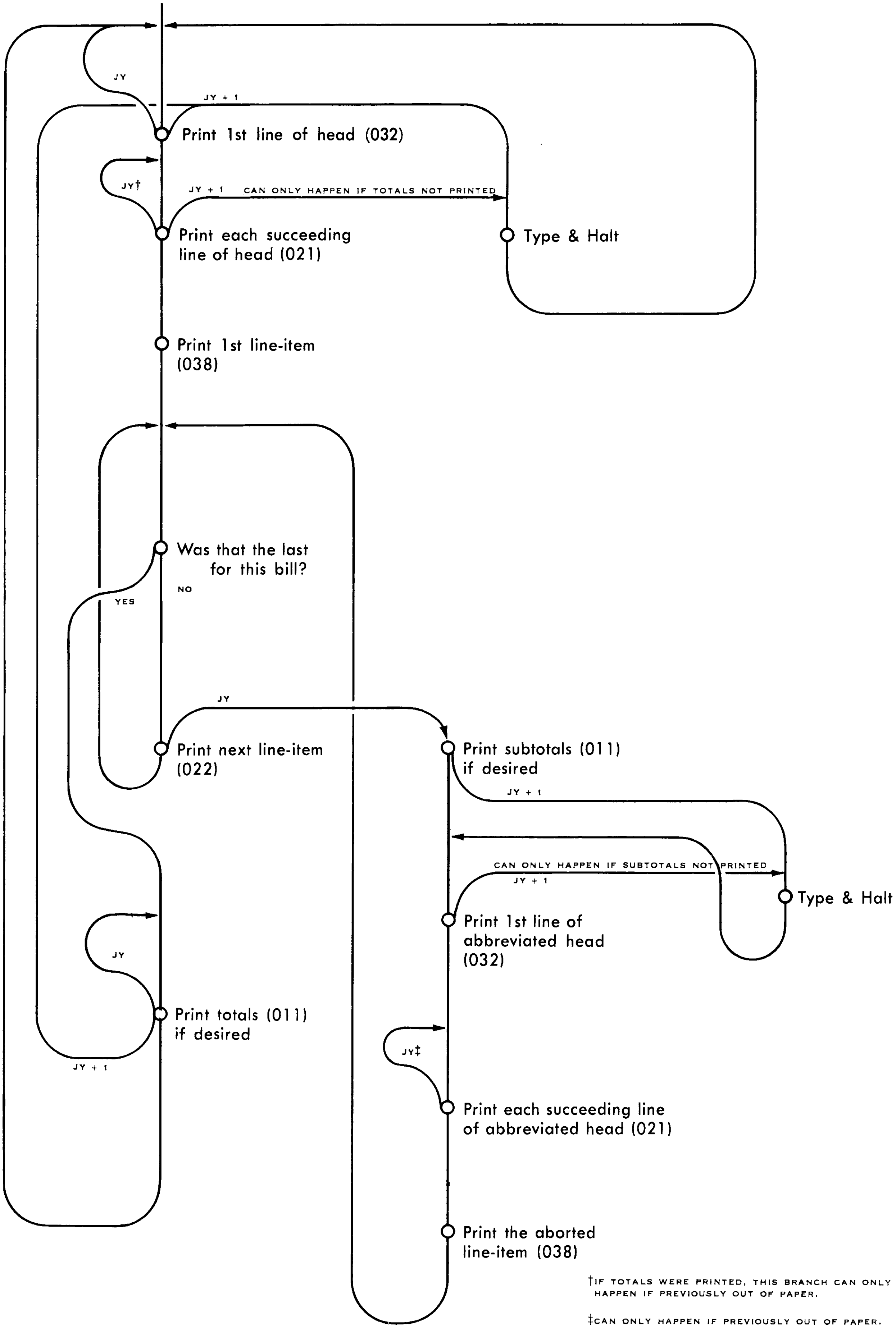
NOTE: Slew controls 00x and 01x should be used with the utmost caution. They are appropriate only when the programmer knows that a print command may take JY+1, and he specifically wants the line printed anyway.

These slew controls should never be used when the programmer is "sure" JY+1 cannot happen, since programmers have been known to be mistaken in such matters.

The following flow chart indicates the logic for the above example. Headings begin on line 2, single-spaced; line-items begin on line 8, double-spaced. A branch line swinging upward on the chart indicates that the branch is taken without executing the command.

## STANDARD PUNCHING for VERTICAL FORMAT TAPE




JY
JY + 1
Print 1st line of head (032)
JY†
JY + 1 CAN ONLY HAPPEN IF TOTALS NOT PRINTED
Print each succeeding line of head (021)
Type & Halt
Print 1st line-item (038)
Was that the last for this bill?
NO
YES
JY
Print next line-item (022)
Print subtotals (011) if desired
JY + 1
CAN ONLY HAPPEN IF SUBTOTALS NOT PRINTED
JY + 1
Type & Halt
Print 1st line of abbreviated head (032)
JY
Print totals (011) if desired
JY + 1
JY‡
Print each succeeding line of abbreviated head (021)
Print the aborted line-item (038)

†IF TOTALS WERE PRINTED, THIS BRANCH CAN ONLY HAPPEN IF PREVIOUSLY OUT OF PAPER.

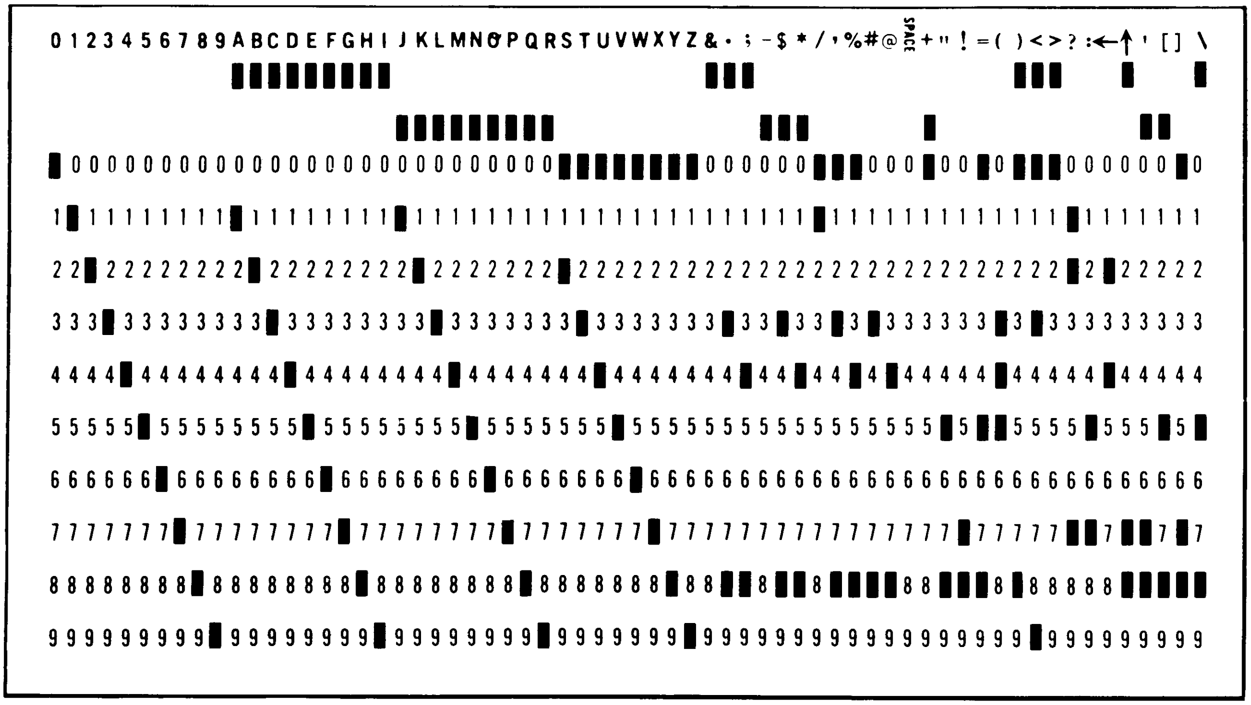
‡CAN ONLY HAPPEN IF PREVIOUSLY OUT OF PAPER.

# CORRESPONDENCE BETWEEN NCR 315 CODES and IBM BCD CODES ON MAGNETIC TAPE

| 315 IMAGE | IBM CHARACTER | BCD Code | | | | | |
|---|---|---|---|---|---|---|---|
| | | B | A | 8 | 4 | 2 | 1 |
| @ | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 3 | 0 | 0 | 0 | 0 | 1 | 1 |
| 4 | 4 | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | 5 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | 6 | 0 | 0 | 0 | 1 | 1 | 0 |
| 7 | 7 | 0 | 0 | 0 | 1 | 1 | 1 |
| 8 | 8 | 0 | 0 | 1 | 0 | 0 | 0 |
| 9 | 9 | 0 | 0 | 1 | 0 | 0 | 1 |
| # | A | 1 | 1 | 0 | 0 | 0 | 1 |
| S | B | 1 | 1 | 0 | 0 | 1 | 0 |
| T | C | 1 | 1 | 0 | 0 | 1 | 1 |
| U | D | 1 | 1 | 0 | 1 | 0 | 0 |
| V | E | 1 | 1 | 0 | 1 | 0 | 1 |
| W | F | 1 | 1 | 0 | 1 | 1 | 0 |
| X | G | 1 | 1 | 0 | 1 | 1 | 1 |
| Y | H | 1 | 1 | 1 | 0 | 0 | 0 |
| Z | I | 1 | 1 | 1 | 0 | 0 | 1 |
| J | J | 1 | 0 | 0 | 0 | 0 | 1 |
| K | K | 1 | 0 | 0 | 0 | 1 | 0 |
| L | L | 1 | 0 | 0 | 0 | 1 | 1 |
| M | M | 1 | 0 | 0 | 1 | 0 | 0 |
| N | N | 1 | 0 | 0 | 1 | 0 | 1 |
| Ө | Ө | 1 | 0 | 0 | 1 | 1 | 0 |
| P | P | 1 | 0 | 0 | 1 | 1 | 1 |
| Q | Q | 1 | 0 | 1 | 0 | 0 | 0 |
| R | R | 1 | 0 | 1 | 0 | 0 | 1 |
| B | S | 0 | 1 | 0 | 0 | 1 | 0 |
| C | T | 0 | 1 | 0 | 0 | 1 | 1 |
| D | U | 0 | 1 | 0 | 1 | 0 | 0 |
| E | V | 0 | 1 | 0 | 1 | 0 | 1 |

| Translated by macro-command into: | 315 IMAGE | IBM CHARACTER | BCD Code | | | | | | 1401 | 7070 | 7080 | 7090 | 704 | 705 | 705-III | 709 | 650 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | B | A | 8 | 4 | 2 | 1 | | | | | | | | | |
| | F | W | 0 | 1 | 0 | 1 | 1 | 0 | | | | | | | | | |
| | G | X | 0 | 1 | 0 | 1 | 1 | 1 | | | | | | | | | |
| | H | Y | 0 | 1 | 1 | 0 | 0 | 0 | | | | | | | | | |
| | I | Z | 0 | 1 | 1 | 0 | 0 | 1 | | | | | | | | | |
| | * | & | 1 | 1 | 0 | 0 | 0 | 0 | | | | | | | | | |
| | > | • | 1 | 1 | 1 | 0 | 1 | 1 | | | | | | | | | |
| | + | – | 1 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | |
| | = | $ | 1 | 0 | 1 | 0 | 1 | 1 | | | | | | | | | |
| | $ | * | 1 | 0 | 1 | 1 | 0 | 0 | | | | | | | | | |
| | A | / | 0 | 1 | 0 | 0 | 0 | 1 | | | | | | | | | |
| | " | , | 0 | 1 | 1 | 0 | 1 | 1 | | | | | | | | | |
| | ? | % | 0 | 1 | 1 | 1 | 0 | 0 | | | | | | | | | |
| | , | # | 0 | 0 | 1 | 0 | 1 | 1 | | | | | | | | | |
| | Space | @ | 0 | 0 | 1 | 1 | 0 | 0 | | | | | | | | | |
| | ! | Blank | 0 | 1 | 0 | 0 | 0 | 0 | | | | | | | | | |
| ; | ' | ⌑ | 1 | 1 | 1 | 1 | 0 | 0 | | | | | | | | | |
| ← | – | Tape Mk. (TM) | 0 | 0 | 1 | 1 | 1 | 1 | | | | | | | | | |
| \ | \ | Grp. Mk. (GM) | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | |
| ↑ | ↑ | Seg. Mk. (SM) | 0 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | |
| " | / | △ | 1 | 0 | 1 | 1 | 1 | 1 | | | | | | | | | |
| + | < | +0 | 1 | 1 | 1 | 0 | 1 | 0 | | | | | | | | | |
| < | % | –0 | 1 | 0 | 1 | 0 | 1 | 0 | | | | | | | | | |
| ? | ; | Rec. Mk. (RM)‡ | 0 | 1 | 1 | 0 | 1 | 0 | | | | | | | | | |
| : | : | Wd. Separ. | 0 | 1 | 1 | 1 | 0 | 1 | | | | | | | | | |
| | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | |
| | & | | 0 | 0 | 1 | 1 | 0 | 1 | | | | | | | | | |
| | • | | 0 | 0 | 1 | 1 | 1 | 0 | | | | | | | | | |
| | ← | | 0 | 1 | 1 | 1 | 1 | 0 | | | | | | | | | |
| | ( | | 1 | 0 | 1 | 1 | 0 | 1 | | | | | | | | | |
| | ) | | 1 | 0 | 1 | 1 | 1 | 0 | | | | | | | | | |
| | [ | | 1 | 1 | 1 | 1 | 0 | 1 | | | | | | | | | |
| | ] | | 1 | 1 | 1 | 1 | 1 | 0 | | | | | | | | | |
| | Tape Handlers Used | | | | | | | 729-II | ✓ | ✓ | ✓ | ✓ | | | | | |
| | | | | | | | | 729-IV | ✓ | ✓ | ✓ | ✓ | | | | | |
| | | | | | | | | 729-I | | | | | | | ✓ | | ✓ |
| | | | | | | | | 727 | | | ✓ | | ✓ | ✓ | | ✓ | |

## PUNCHED CARD CODE


0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N Ø P Q R S T U V W X Y Z & • ; – $ * / , % # @ SPACE + " ! = ( ) < > ? : ← ↑ ' [ ] \
0
1
2
3
4
5
6
7
8
9

| CARD CHARACTER | IMAGE IN 315 |
|---|---|
| 0 | 8 0 |
| 1 | 4 0 |
| 2 | 2 0 |
| 3 | 1 0 |
| 4 | 0 + |
| 5 | 0 ! |
| 6 | 0 8 |
| 7 | 0 4 |
| 8 | 0 2 |
| 9 | 0 1 |
| @ | 0 K |
| , | 9 2 |
| SP | 0 0 |
| & | + 0 |
| • | J 2 |
| – | ! 0 |
| ! | 0 6 |
| A | M 0 |
| B | K 0 |
| C | J 0 |
| D | + + |
| E | + ! |

| CARD CHARACTER | IMAGE IN 315 |
|---|---|
| F | + 8 |
| G | + 4 |
| H | + 2 |
| I | + 1 |
| ; | + K |
| " | 0 B |
| ? | 6 4 |
| : | 0 D |
| ← | 2 + |
| ↑ | + 6 |
| + | H 0 |
| J | D 0 |
| K | B 0 |
| L | A 0 |
| M | ! + |
| N | ! ! |
| Ø | ! 8 |
| P | ! 4 |
| Q | ! 2 |
| R | ! 1 |
| % | 8 K |
| = | 8 B |

| CARD CHARACTER | IMAGE IN 315 |
|---|---|
| $ | A 2 |
| ( | 1 * |
| ) | Q 2 |
| / | SP 0 |
| * | ! K |
| # | 1 2 |
| S | @ 0 |
| T | 9 0 |
| U | 8 + |
| V | 8 ! |
| W | 8 8 |
| X | 8 4 |
| Y | 8 2 |
| Z | 8 1 |
| < | R 1 |
| > | Q 0 |
| ' | ! 6 |
| [ | ! B |
| ] | 8 6 |
| \ | + B |

| IMAGE IN 315 | CARD CHARACTER |
|---|---|
| 0 0 | SP |
| 0 1 | 9 |
| 0 2 | 8 |
| 0 4 | 7 |
| 0 6 | ! |
| 0 8 | 6 |
| 0 ! | 5 |
| 0 B | " |
| 0 D | : |
| 0 + | 4 |
| 0 K | @ |
| 1 0 | 3 |
| 1 2 | # |
| 1 * | ( |
| 2 0 | 2 |
| 2 + | ← |
| 4 0 | 1 |
| 6 4 | ? |
| 8 0 | 0 |
| 8 1 | Z |
| 8 2 | Y |
| 8 4 | X |

| IMAGE IN 315 | CARD CHARACTER |
|---|---|
| 8 6 | ] |
| 8 8 | W |
| 8 ! | V |
| 8 B | = |
| 8 + | U |
| 8 K | % |
| 9 0 | T |
| 9 2 | , |
| @ 0 | S |
| SP 0 | / |
| ! 0 | – |
| ! 1 | R |
| ! 2 | Q |
| ! 4 | P |
| ! 6 | ' |
| ! 8 | Ø |
| ! ! | N |
| ! B | [ |
| ! + | M |
| ! K | * |
| A 0 | L |
| A 2 | $ |

| IMAGE IN 315 | CARD CHARACTER |
|---|---|
| B 0 | K |
| D 0 | J |
| H 0 | + |
| + 0 | & |
| + 1 | I |
| + 2 | H |
| + 4 | G |
| + 6 | ↑ |
| + 8 | F |
| + ! | E |
| + B | \ |
| + + | D |
| + K | ; |
| J 0 | C |
| J 2 | • |
| K 0 | B |
| M 0 | A |
| Q 0 | > |
| Q 2 | ) |
| R 1 | < |

***NOTE***
***for Addition and Subtraction***
If the memory word is longer than the effective length of the Accumulator, switch Memory and Accumulator for the excess memory slabs.

# ADDITION

## NO CARRY FROM PREVIOUS DIGIT-POSITION

***Use this table for:***
Add like signs
Subtract unlike signs
Add to Memory, like signs
Augment, add positive number
Count, add positive number

*Digits from Memory*

*Digits from Accumulator or Register*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ | , | SPACE | & | • | - |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ | , | SPACE | & | • | - |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | c0 | , | SPACE | & | • | - | 0 |
| 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | c0 | c1 | SPACE | & | • | - | 0 | 1 |
| 3 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | c0 | c1 | c2 | & | • | - | 0 | 1 | 2 |
| 4 | 4 | 5 | 6 | 7 | 8 | 9 | c0 | c1 | c2 | c3 | • | - | 0 | 1 | 2 | 3 |
| 5 | 5 | 6 | 7 | 8 | 9 | c0 | c1 | c2 | c3 | c4 | - | 0 | 1 | 2 | 3 | 4 |
| 6 | 6 | 7 | 8 | 9 | c0 | c1 | c2 | c3 | c4 | c5 | 0 | 1 | 2 | 3 | 4 | 5 |
| 7 | 7 | 8 | 9 | c0 | c1 | c2 | c3 | c4 | c5 | c6 | 1 | 2 | 3 | 4 | 5 | 6 |
| 8 | 8 | 9 | c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7 | 2 | 3 | 4 | 5 | 6 | 7 |
| 9 | 9 | c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | 3 | 4 | 5 | 6 | 7 | 8 |
| @ | c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | 4 | 5 | 6 | 7 | 8 | 9 |
| , | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c@ | 5 | 6 | 7 | 8 | 9 | c0 |
| SPACE | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c@ | c, | 6 | 7 | 8 | 9 | c0 | c1 |
| & | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c@ | c, | cSPACE | 7 | 8 | 9 | c0 | c1 | c2 |
| • | c4 | c5 | c6 | c7 | c8 | c9 | c@ | c, | cSPACE | c& | 8 | 9 | c0 | c1 | c2 | c3 |
| - | c5 | c6 | c7 | c8 | c9 | c@ | c, | cSPACE | c& | c• | 9 | c0 | c1 | c2 | c3 | c4 |

F

G

## CARRY FROM PREVIOUS DIGIT-POSITION

*Digits from Memory*

*Digits from Accumulator or Register*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ | , | SPACE | & | • | - |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | c0 | , | SPACE | & | • | - | 0 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | c0 | c1 | SPACE | & | • | - | 0 | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | c0 | c1 | c2 | & | • | - | 0 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 | c0 | c1 | c2 | c3 | • | - | 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | c0 | c1 | c2 | c3 | c4 | - | 0 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | c0 | c1 | c2 | c3 | c4 | c5 | 0 | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | c0 | c1 | c2 | c3 | c4 | c5 | c6 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | 4 | 5 | 6 | 7 | 8 | 9 |
| @ | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c@ | 5 | 6 | 7 | 8 | 9 | c0 |
| , | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c@ | c, | 6 | 7 | 8 | 9 | c0 | c1 |
| SPACE | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c@ | c, | cSPACE | 7 | 8 | 9 | c0 | c1 | c2 |
| & | c4 | c5 | c6 | c7 | c8 | c9 | c@ | c, | cSPACE | c& | 8 | 9 | c0 | c1 | c2 | c3 |
| • | c5 | c6 | c7 | c8 | c9 | c@ | c, | cSPACE | c& | c• | 9 | c0 | c1 | c2 | c3 | c4 |
| - | c6 | c7 | c8 | c9 | c@ | c, | cSPACE | c& | c• | c- | c0 | c1 | c2 | c3 | c4 | c5 |

F

G

*Use this table for:*
Add unlike signs
Subtract like signs
Add to Memory, unlike signs but interchange Accumulator and Memory for entry to the table
Augment, add negative number
Count, add negative number

# SUBTRACTION

If the Subtraction Table indicates that the result of an operation ends with a borrow (for example **b823**) the result is automatically complemented and appears in the processor as negative (-177).

## NO BORROW BY PREVIOUS DIGIT-POSITION

*Digits from Memory*

*Digits from Accumulator or Register*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ | , | SPACE | & | • | - |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- | b• | b& | bSPACE | b, |
| **1** | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- | b• | b& | bSPACE |
| **2** | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- | b• | b& |
| **3** | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- | b• |
| **4** | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- |
| **5** | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| **6** | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 |
| **7** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 |
| **8** | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 |
| **9** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 |
| @ | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 |
| , | , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 |
| SPACE | SPACE | , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 |
| & | & | SPACE | , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 |
| • | • | & | SPACE | , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 |
| - | - | • | & | SPACE | , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

F

G

## BORROW BY PREVIOUS DIGIT-POSITION

*Digits from Memory*

*Digits from Accumulator or Register*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ | , | SPACE | & | • | - |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- | b• | b& | bSPACE | b, | b@ |
| **1** | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- | b• | b& | bSPACE | b, |
| **2** | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- | b• | b& | bSPACE |
| **3** | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- | b• | b& |
| **4** | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- | b• |
| **5** | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b- |
| **6** | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| **7** | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 |
| **8** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 |
| **9** | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 | b3 |
| @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 | b4 |
| , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 | b5 |
| SPACE | , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 | b6 |
| & | SPACE | , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 | b7 |
| • | & | SPACE | , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 | b8 |
| - | • | & | SPACE | , | @ | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | b9 |

F

G

# EXECUTION TIMES OF 315 INTERNAL COMMANDS

Times are stated as number of blocks of 6 μsec (0.000 006 seconds).

To obtain total processing time in minutes, add up total blocks and move decimal point 7 places to the left.

Unless otherwise specified, time for R15 is time for L=1, less 1 block.

L means length of the memory word (number of slabs)

@ means length of the Accumulator (number of slabs)

£ means the longer of $\begin{cases} @ \\ L-1 \end{cases}$

N means the "N" of the command (number of slabs, characters, registers)

S means the sum of the quotient digits

| | | |
|---|---|---|
| ADD | 6+ £<br>7+2L | if no sign change<br>if sign change |
| ADD:M | | |
| signs alike | 6+ L | +1 if negative |
| signs unlike | 6+ L<br>7+2L<br>9+2L | if no sign change<br>if sign change to plus<br>if sign change to minus |
| AUG:R,J | 9+4N | |
| BADD R≠15<br>R=15 | 5+ L<br>5+ £ | |
| CLRF:LH,RH | 8 | |
| CLRU | 13 | |
| CNT | 10 | |
| CØMP | 6<br>6+ L | if signs unlike<br>if signs alike |
| DIV | 116+37L—@+S(L+1) | |
| DLR | 7 | +2 if interrupted |
| EDIT | 5+3@ | +1 for each LH character of memory slab into a RH position of an Accumulator slab.<br>+1 for each comma in format. |
| JUMP | 8 | |
| JUMP:I | 10 | |
| JUMP:IP | 9 | |
| LD | 6+ L | |
| LD:R,J | 9+2N | |
| LDAD | 11+2L—@ | +8 if JY exit |
| MLRA | 9 | |
| MØVE Memory | 9+2N | |
| MØVE Registers | 7+2N | |
| MULT | | See Table, next page |
| PAST | 9+ L | |
| SAUG:R,J | 10+2N | |
| SCND, SCNA | 9 | +1 per Digit or Alpha actually scanned<br>+7 if JY exit |
| SELC:DN,DP | 15 | +1 if jump |
| SELC:R | 16 | +1 if jump |
| SELC:T | 13 | +3 if jump |
| SELect other | 14 | +3 if jump |
| SETF:LH,RH | 9 | |
| SETF:Ø,D,T | 7 | |
| SETF:+,- | 6 | |
| SETU | 13 | |

| | | |
|---|---|---|
| SHFT:AL<br>SHFT:AR<br>SHFT:DL<br>SHFT:DR | | See notes and tables, next page. |
| SHFT:LC | 5+ N((@+2) | |
| SHFT:RC | 6+ N((@+2) | |
| SHFT:RR | | Same as SHFT:DR |
| SKIP | 9 | |
| SLD:R,J | 11+ N | |
| SPRD Memory | 10+ N | |
| ST | 6+ L | +1 if sign negative |
| ST:R,J | 8+3N | |
| STDA | 9+ L | |
| SUB | 6+ £<br>7+2L | if no sign change<br>if sign change |
| SUPP | 7 | +1 for each slab in which suppression occurs |
| TEST:D,T | 8 | +1 if jump |
| TEST:G,L,E | 6 | +2 if jump |
| TEST:LH,RH | 9 | +5 if jump |
| TEST:SW | 11 | +3 if jump |
| TEST:Ø | 8 | +2 if jump |
| TEST:- | 6 | +2 if jump |

## MULTIPLICATION TIMES

| L \ @ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 46 | 77 | 117 | 166 | 224 | 291 | 367 |
| 2 | 63 | 97 | 140 | 192 | 253 | 323 | |
| 3 | 86 | 123 | 169 | 224 | 288 | | |
| 4 | 115 | 155 | 204 | 262 | | | |
| 5 | 150 | 193 | 245 | | | | |
| 6 | 191 | 237 | | | | | |
| 7 | 238 | | | | | | |

The shorter number should be in the Accumulator to obtain the best execution time.

## NOTES ON SHIFT TIMES

**Alpha** For shifts of 4 or more positions, it is faster to store and reload, plus (if necessary) shift 1 position.

**Digit** For shift of exactly 3 positions, it is faster to shift 2 Alphas.

For shifts of 6 or more positions, it is faster to store and reload, plus (if necessary) shift 1 or 2 positions.

**Round** When shifting exactly 6 places, or more than 8 places, it is faster to add, store, and reload, plus (if necessary) shift 1 or 2 positions.

### SHFT:AL

| N \ @ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 2 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 23 |
| 3 | 14 | 17 | 20 | 23 | 26 | 29 | 32 | 32 |
| 4 | 19 | 23 | 27 | 31 | 35 | 39 | 41 | 41 |
| 5 | 23 | 28 | 33 | 38 | 43 | 48 | 50 | 50 |
| 6 | 29 | 35 | 41 | 47 | 53 | 57 | 59 | 59 |
| 7 | 34 | 41 | 48 | 55 | 62 | 66 | 68 | 68 |
| 8 | 41 | 49 | 57 | 65 | 71 | 75 | 77 | 77 |
| 9 | 47 | 56 | 65 | 74 | 80 | 84 | 86 | 86 |
| 10 | 55 | 65 | 75 | 83 | 89 | 93 | 95 | 95 |
| 11 | 62 | 73 | 84 | 92 | 98 | 102 | 104 | 104 |
| 12 | 71 | 83 | 93 | 101 | 107 | 111 | 113 | 113 |
| 13 | 79 | 92 | 102 | 110 | 116 | 120 | 122 | 122 |
| 14 | 89 | 101 | 111 | 119 | 125 | 129 | 131 | 131 |
| 15 | 98 | 110 | 120 | 128 | 134 | 138 | 140 | 140 |

### SHFT:AR

| N \ @ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 2 | | 13 | 15 | 17 | 19 | 21 | 23 | 25 |
| 3 | | 15 | 18 | 21 | 24 | 27 | 30 | 33 |
| 4 | | | 22 | 26 | 30 | 34 | 38 | 42 |
| 5 | | | 24 | 29 | 34 | 39 | 44 | 49 |
| 6 | | | | 33 | 39 | 45 | 51 | 57 |
| 7 | | | | 35 | 42 | 49 | 56 | 63 |
| 8 | | | | | 46 | 54 | 62 | 70 |
| 9 | | | | | 48 | 57 | 66 | 75 |
| 10 | | | | | | 59 | 71 | 81 |
| 11 | | | | | | 61 | 74 | 85 |
| 12 | | | | | | | 76 | 90 |
| 13 | | | | | | | 78 | 93 |
| 14 | | | | | | | | 95 |
| 15 | | | | | | | | 97 |

### SHFT:DL

| N \ @ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 14 |
| 2 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 24 |
| 3 | 15 | 18 | 21 | 24 | 27 | 30 | 33 | 34 |
| 4 | 20 | 24 | 28 | 32 | 36 | 40 | 43 | 44 |
| 5 | 24 | 29 | 34 | 39 | 44 | 49 | 53 | 54 |
| 6 | 29 | 35 | 41 | 47 | 53 | 59 | 63 | 64 |
| 7 | 35 | 42 | 49 | 56 | 63 | 69 | 73 | 74 |
| 8 | 40 | 48 | 56 | 64 | 72 | 79 | 83 | 84 |
| 9 | 46 | 55 | 64 | 73 | 82 | 89 | 93 | 94 |
| 10 | 53 | 63 | 73 | 83 | 92 | 99 | 103 | 104 |
| 11 | 59 | 70 | 81 | 92 | 102 | 109 | 113 | 114 |
| 12 | 66 | 78 | 90 | 102 | 112 | 119 | 123 | 124 |
| 13 | 74 | 87 | 100 | 112 | 122 | 129 | 133 | 134 |
| 14 | 81 | 95 | 109 | 122 | 132 | 139 | 143 | 144 |
| 15 | 89 | 104 | 119 | 132 | 142 | 149 | 153 | 154 |
| 16 | 98 | 114 | 129 | 142 | 152 | 159 | 163 | 164 |

### SHFT:DR

| N \ @ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 2 | 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 |
| 3 | | 17 | 20 | 23 | 26 | 29 | 32 | 35 |
| 4 | | 19 | 23 | 27 | 31 | 35 | 39 | 43 |
| 5 | | 21 | 26 | 31 | 36 | 41 | 46 | 51 |
| 6 | | | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | | | 32 | 39 | 46 | 53 | 60 | 67 |
| 8 | | | 34 | 42 | 50 | 58 | 66 | 74 |
| 9 | | | | 46 | 55 | 64 | 73 | 82 |
| 10 | | | | 48 | 58 | 68 | 78 | 88 |
| 11 | | | | 50 | 61 | 72 | 83 | 94 |
| 12 | | | | | 65 | 77 | 89 | 101 |
| 13 | | | | | 67 | 80 | 93 | 106 |
| 14 | | | | | 69 | 83 | 97 | 111 |
| 15 | | | | | | 87 | 102 | 117 |
| 16 | | | | | | 89 | 105 | 121 |

An Educational Publication
Marketing Services Department
THE NATIONAL CASH REGISTER COMPANY — DAYTON 9, OHIO

NCR