1103A and 1105 PROGRAMMING

SPECIAL DISCUSSION OF COMMANDS

and

INPUT - OUTPUT EQUIPMENT

By
Guenther F. Paprotny
Univac Scientific and Univac 1105 Programming Training

INTRODUCTION

A. Who may read this manual?

During the discussion of the 1103A and 1105 basic commands, as presented in Chapter IV, it was assumed that the reader already possesses some basic knowledge of those commands and their applications. This knowledge must not be less than that of a student after 2 weeks of an 1103A or 1105 Programming course.

The material is presented to serve the following purposes:

1. as an aid in 5-week (or longer) 1103A or 1105 programming courses. It may be handed out to students as "class notes" at the beginning of the third week.

2. as a Reference Manual for experienced 1103A or 1105 programmers. In this respect the manual might answer questions with regard to special applications of commands and to programming input - output equipments.

3. as study material for experienced 1103 programmers who have to learn 1103A or 1105 programming.

4. as self-study material to 1103A and 1105 programmers who wish to increase their knowledge of the computer, its programming and internal operation (internal operation to the extent that only those machine operations are discussed which affect programming!)

B. Content of Manual

This manual contains a complete description of the 1103A and the 1105 computers and their Input - Output equipments.

Chapter I describes the Address System of an 1103A or 1105 computer.

Chapter II describes the functioning of the Program Address Counter. The use for extraction of commands is discussed as well as restrictions for carries in PAK and their consequences. This is explained for computers with one, two, or three banks of core storage.

Chapter III explains "One's Transmission" in order to enable the reader to understand the "Left Shift" and "Split" commands.

Chapter IV contains a special discussion of all 1103A and 1105 commands. This discussion presents exact sequences, where necessary, and points out "pitfalls" and computer faults because of the use of A and Q as operands.

Chapter V explains programming for the following Input-Output equipments:

The on-line electric Typewriter
The High Speed Punch Unit
The Ferranti Tape Reader
The on-line 80-column Card Unit
The Magnetic Tape System (Fixed and Variable Block Length)

Whenever necessary for programming the equipment is described. Furthermore, paragraph c) contains a description of the 1103A and 1105 lockout circuitry from a programmers point of view in order to enable the reader to understand why he has to program in this or that way during input-output operations.

Chapter VI presents a discussion of the 1105 Magnetic Tape and Buffer System and programming consequences.

Chapter VII discusses the 1103A and 1105 Interrupt Facility and its application for program controlled operations.

Chapter VIII describes the 1103A and 1105 Floating Point System.

C) Definition of Input-Output System for 1103A and 1105.

1103A:

Flexowriter
High Speed Punch Unit
Ferranti Reader
"Bull" Card Unit (optional)
Magnetic Tape System (≦10 Uniservos, One Tape Control Unit)

1105:

Flexowriter
High Speed Punch Unit
Ferranti Reader
"Bull"Card Unit (optional)
Magnetic Tape System (≦20 Uniservos, Two Tape Control Units, Two Buffer Units)

D) Which Chapters are to be read by 1105 Programmers; which ones by 1103A Programmers?

1105 Programmers:

In order to study programming for the 1105 Computer read the whole manual in the sequence presented. You may drop the description of "On-Line Card Unit" and/or "Floating Point System", if the computer you will work with does not possess these optional features.

1103A Programmers:

Read whole manual in the sequence presented except Chapter VI and Chapter VII, paragraph C, 2. Drop Floating Point and/or Card Unit, if the computer you will work with does not possess those.

Günther F. Tapolsky

References:

a) Univac Scientific General-Purpose Computer System, Programming, PX 18, Sept. 56.

b) Programming Manual, Univac Scientific Computing System Model 1103A, Manuscript Copy of section on magnetic tapes, Oct. 30, 57.

c) Programming Manual, Rough Draft, Univac Scientific Model 1103A

d) Preliminary Programming Manual for the Univac 1105 Computing System, US 108, March 7, 58

e) Univac Scientific General-Purpose Computer System, External Function Modification to the Basic Computer, PX 150, Sept. 57

f) Univac Scientific General-Purpose Computer System, Modification to Provide Two Additional Core Bays to the Basic Computer, PX 148, Sept. 57

g) Univac Scientific General-Purpose Computer System, Timing Sequences, PX 21, Oct. 56.

CONTENTS

I) The 1103A/1105 Address System

a) Magnetic Core Storage

The core storage of the 1103A and the 1105 computers consists of up to three (3) banks of cores. One bank of core storage holds 4096_{10} 36-bit-words and is a standard equipment.

The addresses are:

MCS 0 (first bank):	00000	thru	07777	(octal)
MCS 1 (second bank):	10000	thru	17777	(octal)
MCS 2 (third bank):	20000	thru	27777	(octal)

b) Arithmetic Registers

The addresses of the 36-bit Q-register are:

31000 thru 31777 (octal)

The addresses of the 72-bit Accumulator are:

32000 thru 37777 (octal)

c) Magnetic Drum Storage ("Double Drum")

The magnetic drum storage is divided into two (2) zones; zone A and zone B. Each zone possesses 16384_{10} registers with addresses

40000 thru 77777 (octal)

Therefore we have 2 registers 40000, 2 registers 40001, etc. If e.g. a reference to 40000 is made it depends upon the zone - selection made earlier, whether "40000 zone A" or "40000 zone B" is employed. This zone selection will be described later. (See page 50.)

d) Illegal Addresses

The illegal addresses of the computer are, if it is equipped with

one bank of core storage:	10000	thru	30777	(octal)
two banks of core storage:	20000	thru	30777	(octal)
Three banks of core storage:	30000	thru	30777	(octal)

II) The Program Address Counter PAK

a) General Remarks

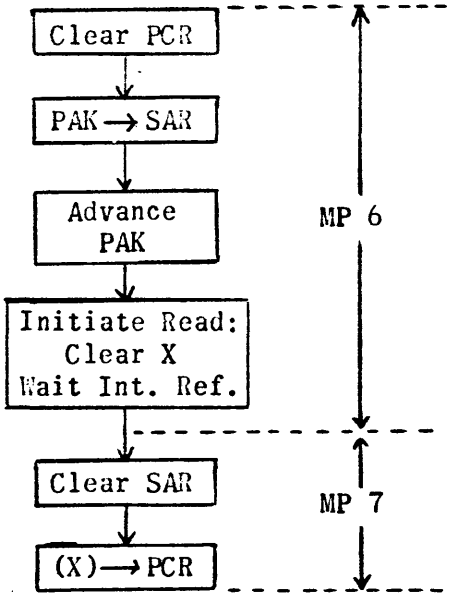
PAK is a 15-bit register which serves in two ways:

1) as a storage for the address from which the next command is to be extracted.
2) as a repeat counter during the execution of a repeat sequence also controlling the advancement of the u- and v-address of the repeated instruction. (This will be discussed under "Repeat command".)

The extraction of a command from storage and its execution is based upon eight (8) Main Pulses, MP 6, MP 7, and MP 0 thru MP 5.
MP 6 and MP 7 extract a command from storage,
MP 0 thru MP 5 execute this command.

A computer Master Clear which precedes all operations of the computer sets it automatically to MP 6 and PAK to 40000. Before depressing the Start button the operator has to manually insert the address at which the program starts, into PAK (if this address is different from 40000_8).

Now the following sequence of steps takes place upon starting operation:
(It is pointed out that one box of the flow diagram does not represent one clock pulse)



MP 6 essentially transfers the address from PAK to SAR, advances PAK by 1, and reads the word from the storage register whose address is now held in SAR, into X.

MP 7 brings the word from X into PCR, where it later will be interpreted as command.

MP 7 also clears SAR, since this will be used during the execution of the command.

SAR = Storage Address Register, a 15-bit register. When reading from or writing into a storage location the computer "looks" at SAR in order to determine the address of this storage location. Notice that SAR is not cleared, before the transfer PAK to SAR is made. However, each "Read into X" and "Write from X" sequence clears SAR immediately, after it used it. This involves transmissions between X on one side and core, drum, A, and Q on the other side.

But notice; <u>after a "Shift" sequence SAR is not cleared</u>. This is the reason for some rather unexpected results during special uses of "Left Shift" and "Split" commands, as explained later.

PCR = Program Control Register, a 36-bit register.

A word entering PCR is interpreted as command.

PCR consists of:

MCR = Main Control Register; 6 leftmost bits of PCR which hold the operation code.

UAK = U-Address Counter; 15 bits holding the u-address portion of the command.

VAK = V-Address Counter; 15 bits holding the v-address portion of the command.

Special use of UAK and VAK is explained later during the discussion of a command, whenever it is necessary.

b) Restrictions for Carries in PAK

When PAK is advanced a "1" is added to the number (address) held in PAK. The carry from one stage of PAK to the next stage is, however, restricted in the following way:

PAK = xxx xxx xxx xxx xxx (binary)

PAK_{14}, PAK_{13}, PAK_{12} ↑ PAK_{11} ↑ PAK_0 ↑

Restriction present with all 1103A/1105 computers:

There is never a carry from PAK_{13} to PAK_{14}.

In addition to this the following restriction is imposed on PAK:

1) Computer with one bank of core storage

If $PAK_{14} = 0$, there will be no carry from PAK_{11} to PAK_{12}.

If $PAK_{14} = 1$, there will be a carry from PAK_{11} to PAK_{12}.

Consequences for a computer with one bank of core storage:

Assume PAK = $100\ 000\ 000\ 000\ 000_2 = 40000_8$

Here PAK can be advanced up to

$111\ 111\ 111\ 111\ 111_2 = 77777_8$,

because the carry from PAK_{11} to PAK_{12} can be made.

Advancing PAK again by 1 we have:

$$\begin{array}{r} 111\ 111\ 111\ 111\ 111 \\ +\qquad\qquad\qquad\qquad\ 1 \\ \hline 100\ 000\ 000\ 000\ 000 \end{array} \qquad \text{(binary)}$$

That means that in PAK

$$\begin{array}{r} 77777_8 \\ +\quad 1 \\ \hline 40000_8 \end{array} \qquad \text{(octal)}$$

Assume PAK = $000\ 000\ 000\ 000\ 000_2 = 00000_8$

Here PAK can be advanced to

$000\ 111\ 111\ 111\ 111_2 = 07777_8$

Because of $PAK_{14} = 0$ there will be no carry from PAK_{11} to PAK_{12}. Therefore, adding a "1" again we have:

$$\begin{array}{r} 000\ 111\ 111\ 111\ 111 \\ +\qquad\qquad\qquad\qquad\ 1 \\ \hline 000\ 000\ 000\ 000\ 000 \end{array} \qquad \text{(binary)}$$

That means that now in PAK

$$\begin{array}{r} 07777 \\ +\quad 1 \\ \hline 00000 \end{array} \qquad \text{(octal)}$$

2) Computer with two or three banks of core storage.
In this case the restriction mentioned under 1) depends upon a switch set on the Supervisory Control Console. This is the so-called MCS-Section Switch. It will be set to one of the following two positions:

SINGLE - If the switch is set to this position the restriction for the carry is imposed on PAK as discussed under 1) above, i.e. there will be no carry from PAK_{11} to PAK_{12}, if $PAK_{14} = 0$.

NORMAL - Setting the switch to this position means to drop the restriction mentioned under 1). That means that now a carry from PAK_{11} to PAK_{12} can be made

regardless of the value of PAK_{14}. (Only exception: see "Repeat Command").

Let us look at the Normal setting of the switch. Here the programmer has to keep in mind the following fact:

a) Computer with two banks of core storage:

PAK can be advanced from 00000_8 thru 17777_8.

But $17777_8 + 1$

results in PAK = 00000_8.

b) Computer with three banks of core storage:

PAK can be advanced from 00000_8 thru 27777_8.

But $27777_8 + 1_8$

results in PAK = 00000_8

(In order to find out how this is accomplished by the machine refer to Block Diagrams)

III) Remarks on "One's Transmission"

Experience shows that some confusion exists even among experienced programmers of the 1103/1103A/1105, if the word "one's transmission" is mentioned. However, as pointed out during the discussion of PAK (see: PAK → SAR), it is absolutely necessary that a programmer understands the meaning of this word and the results caused by a one's transmission, if he really wants to understand the Shift-and Split-Commands (which will be discussed later in details).

Without going into engineering details let us consider the final results after a one's transmission from PAK to SAR:

Here One's Transmission means:

If PAK_i = 1, switch SAR_i to "1",

If PAK_i = 0, do not touch SAR_i.

(i = 1,2, ...14)

Assume PAK = $000\ 110\ 100\ 010\ 011_2 = 06423_8$

After one's transmission: SAR = $011\ 001\ 011\ 110\ 010_2 = 31362_8$

SAR = $011\ 111\ 111\ 110\ 011_2 = 37763_8$

This result can easily be obtained applying the above rule.

The example also shows that the result may be obtained by applying the following logical addition in binary:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1

This means that you merely may add the two numbers in binary in the normal way with the exception that 1 + 1 = 1. No carry will be produced.

It should be noticed that all transfers as e.g. storage→X, X→storage, X→Q, etc. are made by one's transmission. However, in order to obtain the correct result in the register into which has to be written the computer automatically clears this register first. Thus, if e.g.

(X) = 0——0100
(v) = 1——1100

and (X)→v has to occur, the computer produces.

Clear v.

One's Transmission from X to v.

This is merely stated here. The only exception from this general rule which is the concern of the programmer is the above mentioned one's transmission from PAK (UAK, VAK) to SAR or SAR to PAK.

IV) Special Discussion of Basic 1103A/1105 Commands

The following paragraphs represent a discussion of all commands except Floating-Point-Commands and the Input/Output-commands EF -v, ER jv, EW jv. This discussion refers in particular to programming situations which either would cause a computer fault or an unusual result; it requires, however, a basic knowledge of these commands on the part of the reader.

a) General Remarks

During most of the commands numbers are transferred in the computer from

storage to storage
storage to A or Q
A or Q to storage

plus certain transfers to input/output registers (discussed later). All these transfers are made via the X-register (Exchange Register).

Two of the above transfers are of special interest: Accumulator to X, X to Accumulator. These are transfers between a 72-bit register and a 36-bit register requiring special discussion.

1) Accumulator to X:

With the exception of one command, the LT jk v, there will always be a transfer of (A_R) to X. This is done automatically by the computer. Thus, the computer never picks up (A_L) except in a LT jk v with j = 0.

2) X to Accumulator:

Here the situation is somewhat different. A is an additive register which means that a number may be added to (A), but not transferred to A. If a transfer is to be obtained the computer automatically executes the following steps:

Clear A
$(A) + D(X) \longrightarrow A$

This results in the double extension of (X) in A, i.e. (A_R) = X and (A_L) contains sign-bits.

However, some commands do not use the double extension. These are the three (3) Q-Controlled commands and the four (4) Split-commands. They use the single extension S(X), i.e. (A_R) = X and (A_L) contains zeros.

Keep in mind that the Q-Controlled and Split commands are the only ones which make use of S(X) instead of D(X). All other commands <u>always</u> use D(X), if (X) has to be added to (A).

b) <u>"Transmit" Commands</u>

TP u v Sequence: $(u) \longrightarrow X$ *
$(X) \longrightarrow v$ **

** If v = A: Clear A
$(A) + D(X) \longrightarrow A$

* If u = A: $(A_R) \longrightarrow X$

TM u v
TN u v

These two commands are executed like the TP except that $|(u)|$ or $(u)^1$, respectively, are transferred

Notice that no faults ever occur during the execution of one of these three commands because of the use of A or Q as u- or v-addresses. (Here, and from now on, it is assumed that no illegal addresses like 30000_8 etc. are used as u or v. The use of an illegal u- or v-address, naturally, results in an SCC-Fault, SCC being Storage Class Control.)

TU u v

Sequence:

$u_{29} \ldots\ldots u_{15} \longrightarrow X_{29} \ldots\ldots X_{15}$

$X_{29} \ldots\ldots X_{15} \longrightarrow v_{29} \ldots\ldots v_{15}$

To write it in a different way:

$(u)_u \longrightarrow v_u$

Keep in mind:

$v_{35} \ldots\ldots v_{30}$ and $v_{14} \ldots\ldots v_0$ remain undisturbed

If u = A; $(A_R)_u \longrightarrow v_u$

u = Q; $(Q)_u \longrightarrow v_u$

However, v = A or Q results in an SCC-Fault.

TV u v

Sequence:

$u_{14} \ldots\ldots u_0 \longrightarrow X_{14} \ldots\ldots X_0$

$X_{14} \ldots\ldots X_0 \longrightarrow v_{14} \ldots\ldots v_0$

or, written in a different way:

$(u)_v \longrightarrow v_v$

Again, $v_{35} \ldots\ldots v_{15}$ remain undisturbed

If u = A: $(A_R)_v \longrightarrow v_v$

u = Q: $(Q)_v \longrightarrow v_v$

Like in the TU u v v = A or Q results in an SCC-Fault.
Therefore keep the general rule in mind:

It is not possible to transfer parts of a 36-bit number into A or Q.

c) "Arithmetic" Commands

RA u v

Sequence:
$(u) \longrightarrow X$
Clear A
$(A) + D(X) \longrightarrow A$
$(v) \longrightarrow X$
$(A) + D(X) \longrightarrow A$
$(A_R) \longrightarrow X$
$(X) \longrightarrow u$ *
* omit, if u = A

If v = A the above sequence shows that then

$$(u)_f = 2 \cdot (u)_i$$

provided that no "overflow" into the sign position (bit i_{35}) occured during the addition.

RS u v — The sequence of this command is equal to that of the RA u v except that (v) is subtracted from (u).

If v = A: $(u)_f = 0$
$(A)_f = 0$

AT u v

Sequence:
$(u) \longrightarrow X$
$(A) + D(X) \longrightarrow A$
$(A_R) \rightarrow X$
$(X) \longrightarrow v$ *
* omit, if v = A

ST u v — The sequence is equal to that of AT u v except that D(u) is subtracted from (A).

Notice that the above four commands RA, RS, AT, and ST never result in a computer fault. An "overflow" into the sign position i_{35} occuring during an addition or subtraction will, therefore, not be noticed during machine operations. If the programmer suspects such a possibility for an overflow he may e.g. apply the method suggested in the following paragraph:

Assume $(u)_i$ = 01————1 (binary)

$(v)_i$ = 0———— 01 (binary)

The command RA u v results in the following:

D(u) ⟶ A: (A) = 0———— 001 ——— 1

D(v) = 0———— 000 —— 01 + (binary)

$(A)_f$= 0———— 010 —— 00

A_R

$(A_R)_f$ ⟶ u: $(u)_f$ = 10———0 (binary)

As it can be seen: The sum of the largest positive number in u, $(u)_i = 2^{35} - 1$, and a "1" results in a negative number in u, $(u)_f = -(2^{35} - 1)$, which is the negative number with the largest absolute value.

> Assume the command following the RA u v is an EJ u w. This will test whether or not D(u) is equal to (A). In the above case this equality would not occur.

Therefore: Whenever the programmer suspects an overflow during an addition he may test this by giving the following commands:

```
 a   RA  u v
a+1  EJ  u w
a+2  ........
```

Upon jumping to w the programmer knows that no overflow occured. But if the computer proceeds with the next instruction in sequence (e.g. at a+2 in the above example) the programmer can provide some means which will indicate to him that an overflow occured.

MP u v

Sequence:
(u)⟶X
Clear A
(X)⟶ Q
(v)⟶ X
Form in A the (true) product of (Q) and (X).

Giving this command the programmer has to keep in mind two things:

$(Q)_i$ is destroyed and replaced by (u)

$(A)_i$ is destroyed and replaced by (u) · (v)

Moreover: if v = A: $(A)_f = 0$

if v = Q: $(A)_f = (u)_i^2$

It is obvious that this command never results in an overflow into the sign-position A_{71}.

MA u v

Sequence:

$(u) \longrightarrow X$

$(X) \longrightarrow Q$

Shift (A) left 36 places

$(v) \longrightarrow X$

Add the product (Q) · (X) to (A) using an addition process.

This command results, as it is well known, in $(A)_f = (A)_i + (u)\cdot(v)$. However, during the addition an overflow into the sign-position might occur depending upon the values of $(A)_i$, (u), and (v). The overflow might occur, if $(A)_i$ is very large such that $A_{71} \neq A_{70}$. The computer tests this condition <u>after</u> the above shift of 36 places in A has been made, i.e. it tests, whether or not $A_{35} \neq A_{34}$. If this is the case computation stops with an Overflow-Fault ("A"-Fault) <u>indicating the possibility of an overflow</u>. Notice that $A_{35} \neq A_{34}$ or, originally, $A_{71} \neq A_{70}$ does <u>not</u> mean that an overflow will occur in any case.

DV u v

Sequence:

$(u) \longrightarrow X$

$\frac{(A)_i}{(X)} \rightarrow Q$

$(Q) \longrightarrow v$ *, $(A)_f \geqq 0$ is Remainder

* If v = A:

Clear A

$(A) + D(Q) \longrightarrow A$,

i.e. remainder is lost.

Notice that the remainder in A always is positive. This can cause differnt results in Q and A during two divisions performed with the same arithmetic number.

To illustrate this let us take the number $-\frac{7}{3}$.

First case: $(A)_i = +7$

$(u)_i = -3$

here: $-\frac{7}{3} = +2 + \frac{1}{-3}$

Therefore:

quotient $(Q)_f = -2$

remainder $(A)_f = +1$

Second case: $(A)_i = -7$

$(u)_i = +3$

here: $\frac{-7}{3} = -3 + \frac{2}{3}$

Therefore:

quotient $(Q)_f = -3$

remainder $(A)_f = +2$

It is clear that the quotient of a division might consist of more than 35 significant bits, as e.g.

$$(A)_i = 2^{60},\ (u) = 2^2,\ \text{quotient} = 2^{58}$$

Such a number cannot be placed into Q. Therefore, if this situation occurs, i.e. an "overflow" in Q is about to take place, the computer stops with a Divide Fault ("A" Fault). Notice that at this time $(A)_i$ will be already destroyed.

d) Jump and Stop Commands

The following commands cause the computer to "ask the question"

EJ u v	Is D(u) = (A)?
TJ u v	Is D(u) > (A)?
ZJ u v	IS (A) = 0 ?
SJ u v	Is (A) $\geqq$ 0 ?
QJ u v	Is (Q) $\geqq$ 0 ? *

If the answer to any of the above questions is "yes" a jump to address v occurs.

* Keep in mind that after the decision "(Q) $\geqq$ 0 or not" is made the content of Q is shifted one place to the left in any case.

If the answer turns out to be "no" the sequence of steps which has to follow depends upon the nature of the command:

One-way-jump (EJ, TJ): take next instruction in sequence

Two-way-jump (SJ,ZJ,QJ): jump to u.

The above commands do not alter the contents of registers involved in their execution.

IJ u v

Sequence:

$(u) \longrightarrow X$

Clear A

$(A) + D(X) \longrightarrow A$

$(A) - 1 \longrightarrow A$

Is $A_{71} = 0$?

If yes:

$(A_R)_f \longrightarrow u$

jump to v

If no:

take next instruction in sequence

The reader will probably know that this command is mainly used for performing "loops" in the program, i.e. for executing a part of a program several times. Keep in mind that, if a part of the program has to be executed n times and the IJ is at the end of this part (where it will be in almost all cases), the "index" (u) has to be = n-1.

Also notice: after the loop has been performed n times and the computer continues with the instruction immediately following the IJ, $(u)_f = 0$.

MJ j v

The sequence of steps resulting from the execution of this command depends upon the value "j". This j is represented by the leftmost octal digit of the u-portion of the command, exactly $i_{29}\ i_{28}\ i_{27}$ in binary. Let us discuss the different values for j:

j = 0, i.e. MJ 00000 v

This is an unconditional jump to address v.

j = 1, i.e. MJ 10000 v

j = 2, i.e. MJ 20000 v

j = 3, i.e. MJ 30000 v

In these three cases there are two possibilities.

Either the switch on the console which corresponds to the number used in the command (1,2, or 3) is set

then: undonditional jump to v

or the corresponding switch is not set

then: take next instruction in sequence.

Keep in mind: the above mentioned switch on the console can be set or released, if and only if the computer is <u>not</u> operating. During computer operation a setting or releasing of these switches is blocked.
Therefore: if one part of your program makes use of a j = 1 set, and another part requires j = 1 to be released then you have to stop compuéer operation with a MS jv (see below). Now the switch may be released, and operation can be resumed.

j = 4, i.e. MJ 40000 v
j = 5, i.e. MJ 50000 v
j = 6, i.e. MJ 60000 v
j = 7, i.e. MJ 70000 v

These values of j do not possess any corresponding switch on the console. The execution of these four (4) commands has to be discussed for two cases:

1) 1103A, i.e. computer without Buffer System:
Here the j is actually determined by the bits i_{28} i_{27} which means that i_{29} is disregarded by the machine. Therefore, we have the following situation:

If i_{29} i_{28} i_{27} equals	it results in a "machine j" of
000 or 100	j = 0
001 or 101	j = 1
010 or 110	j = 2
011 or 111	j = 3

As you can see: a MJ 00000 v is equivalent to a MJ 40000 v, ..., a MJ 30000 v is equivalent to a MJ 70000 v.

2) 1105, i.e. computer with Buffer System:
In this case any MJ jv with a j of 4,5,6 or 7 represents a completely different kind of command used for Buffer operations. This will be discusséd under "Buffer System".

The following remarks refer to all jump commands:

If v = A and a jump to v occurs: SCC-Fault

For two-way-jumps in addition:

If u = A and a jump to u occurs: SCC-Fault

Notice that the fault occurs, if and only if a jump to A is made, i.e. if the programmer tries to extract the next instruction from A. However, the command EJ u A with (A) = 0, (u) = 1 will, for instance, not result in a fault.

Notice: If a jump to Q is made in any jump-command a fault is not generated. The machine will pick up (Q), send it to PCR, and interpret this as command. The programmer will certainly never try to jump to Q. If this, however, happens because of programming errors, there are three possibilities:

(Q) contains an illegal operation code: MCT-Fault

(Q) contains a jump command: Jump will be performed normally.

(Q) contains a legal, but not jump, command: This will be executed. Since PAK is advanced, say from 31000 to 31001, the next command is again taken from (Q), etc. Notice: you might advance PAK, until it reads 32000. Then: SCC-Fault!

MS j v — There exist again switches on the console for j = 1, 2, 3.

j = 0: v → PAK
Stop

Notice: v → PAK indicates: erase the address held in PAK, and replace it by v. Therefore, a "jump to address v" has been set up by the machine, but before continuing at v a stop is made.

j = 1,2, or 3 and corresponding switch set: v → PAK
Stop

and corresponding switch not set: v → PAK

As you see: v → PAK, i.e. jump to v, takes place in any case. j controls stopping or not stopping only.
Refer to MJ j v in order to see the difference between that command and the MS j v!

j = 4,5,6,7: For <u>both computers</u>, 1103A and the 1105, these values of j correspond to the values 0,1,2,3 in such a way that

a j = 4 results in a "machine-j" of 0

a j = 5 results in a "machine-j" of 1

a j = 6 results in a "machine-j" of 2

a j = 7 results in a "machine-j" of 3

PS - - This is the Program Stop command. If this command is given computer operation can be resumed after a computer Master Clear only.

e) <u>Commands Referencing Subroutines</u>:

Two commands are used for referencing subroutines. These are the Return Jump RJ u v and the Interpret IP - -.

RJ u v Sequence:

$PAK \longrightarrow X_{14} \ldots\ldots X_0$

Clear PAK

$v \longrightarrow PAK$

$X_{14} \ldots\ldots X_0 \rightarrow u_{14} \ldots\ldots u_0$

The sequence for this command is given here in details in order to inform you about the <u>real</u> facts, sinee the explanation to be found usually might cause confusion on the part of the reader or might mislead him. This <u>incorrect</u> explantion I refer to is:

If y is the address of the RJ uv, then $y + 1 \longrightarrow u_v$, $v \longrightarrow PAK$

To state it again: <u>this last mentioned explanation of the RJ uv is incorrect</u>.

Let us follow the correct sequence with an example:

Assume you have the following program:

```
Start   00150   RA 01000 02000
        00170   RJ 00170 00150
        00171   -- ----- -----
```

At the very beginning of the execution of the RJ uv you have:

PAK = 00171 (since it is advanced by 1 already)

PCR = RJ 00170 00150

Executing the RJ the computer saves the address held in PAK by

placing it into X, erases PAK, and continues with: 00150 $\longrightarrow$ PAK

$(X)_v$ = 00171 $\longrightarrow$ 00170_v

Thus the RJ at 00170 is now modified and reads

RJ 00170 00171

and a jump back to 00150 is made. If the RJ at 00170 is executed again later (and has not been changed by some other means in the meantime) it will not jump you back to 00150 again, but you will proceed with the next instruction in sequence.

As it can be seen: the explanation y + 1 $\longrightarrow$ u_v would mean that 00171 is sent to 00170_v changing the RJ before the jump is initiated. According to this explanation you had to pick up 00171 and to place it into PAK (by v $\longrightarrow$ PAK). This is not the case.

You might, however, say: at least the explantion is correct as far as y + 1 is concerned, because the above example picks up 00171, and this is y + 1.

This is right in the above case and will always be so as long as no Interrupt Signal is generated because of the use of the Interrupt Feature, However, if a RJ is executed right after the generation of an Interrupt Signal it will pick up the address held in PAK which will <u>not</u> be the above mentioned y + 1. (See under "Interrupt Feature")

Thus keep in mind:

Executing a RJ u v means to place the address held in PAK at that time into X_v, placing v into PAK and, finally, $(X)_v$ into u_v.

Usually the Return Jump is not used in the way as described in the above example. Normally the v-address of the RJ u v denotes the entrance of a subroutine, and the u-address denotes the exit of this subroutine, where e.g. (u) = MJ 00000 30000.

Keep in mind: using the RJ u v means that the entrance and exit of the subroutine referenced may be anywhere in the core or drum.

As already discussed earlier an SCC-Fault occurs in the following cases:

If v = A: SCC-Fault (do not jump to A)

If u = A <u>or</u> Q: SCC-Fault (do not try to write parts of a 36-bit number into A or Q)

For v = Q refer to the explanation given under "Jump and Stop Commands".

IP - -

The ten (10) octal digits which form the u-and v-portion of the IP-command are insignificant. They are completely disregarded by the machine during the execution of the IP --.
The sequence of steps which takes place can briefly be described as follows:

PAK⟶ v-portion of F_1

F_2 = 00001 ⟶ PAK

Address F_1 is determined by a switch on the Supervisory Control Console which can be set either to "00000" or "40001". Its normal setting is "00000" (also refer to "Repeat Command" which uses the F_1-Switch).

Assume F_1 = 00000. In this case the address held in PAK at the beginning of the execution of IP (which will be = address of IP-command + 1 in almost all cases; see remarks under RJ uv) is transferred to the v-portion of the content of 00000, and a jump to F_2 = 00001 is initiated. Thus the IP referenced a subroutine whose entrance is 00001, exit 00000.

If F_1 = 40001 keep in mind that now the entrance is again F_2=00001, but the exit of the subroutine will be 40001.

The usefulness of this command is based on the fact that the ten octal digits of its u-and v-portion may be used for storing other information as e.g. parameters, pseudo-codes, addresses, etc. which may be used by the subroutine to which the IP -- refers (Interpretive System).

f) The "Left Shift" Commands

As you know shifting can be performed in two registers: in the Accumulator and in the Q-register. There is only a shift to the left. Moreover in either register we have the so-called "end around shift", i.e. you do not "drop off" bits at the left end of the registers.
The two "Left Shift" commands are LA u k and LQ u k. The v-portion of either command contains a number k which determines how many places a word in A or Q is to be shifted to the left. Do not forget: a binary word is shifted by the computer, i.e. k refers to "binary places".

Example:

Assume $(Q)_i$ = 0———01001 in binary, k = 4.

After shifting has been performed $(Q)_f$ is:

$(Q)_f$ = 0———010010000 in binary.

Let us examine the sequence of the LA u k:

(u) ⟶ X
Clear A
} omit, if u = A

(A) + D(X) ⟶ A

Shift (A) k places left

$(A_R)_f$ ⟶ r

where address r is given by the Boolean logical sum of (u) + (v-k)
v denoting the v-portion of the LA u k.

The first part of the execution of this command is probably well known to you: take the double-extension of (u) place it into A, and shift k places left. (If u = A, just shift $(A)_i$). But now comes a point which has to be discussed in details, because the steps to follow depend entirely upon the number contained in the v-part of the instruction LA u k. This v-part consists of 15_{10} bits. However, just 7 bits are used for the representation of the number k, as indicated below:

v-part of LA u k: xxx xxx xxx | x xxx xxx
8 bits | k

How does the machine determine how many places it has to shift? This is done in the following way:

After D(u) is in A, the v-part of the LA u k is transferred to SAR (from VAK). The rightmost 7 bits of SAR are used as a Shift-Counter, SK, and the machine performs the sequence:

Is SK = 0 000 000 ? (binary)

If no: Shift (A) one place, subtract one from SK, and go back to above question.

If yes; continue with the following steps:
Transfer the u-address (from VAK) by one's transmission into SAR. Now write $(A_R)_f$ into the storage given by the number contained in SAR. If this address is an accumulator address do not touch $(A)_f$, but leave it as it is.

As you can see: the transmission of u to SAR will result in SAR = u, if the leftmost 8 bits (as shown above) were all zeros. If they are not all zeros, you generate the Boolean logical sum (" 1 + 1 = 1") in SAR between u and what was left in SAR at the end of the shifting. This might generate an address completely different from u.

Let us follow three examples:

1) Shift (01050) 17_{10} places and place $(A_R)_f$ back into 01050.

   Here: LA 01050 00021

   v-part of LA-command

   Before the shifting of D(01050) in A takes place $0002l_8 \rightarrow$ SAR.

   Therefore, SAR = 000 000 000 010 001 (binary)

   k

   After the shift SAR = 00000_8. The transfer u = 01050_8 to SAR results in SAR = 01050_8, and thus $(A_R)_f \rightarrow$ back to 01050.

2) Shift (01050) 17_{10} places in A and send result to 05250.

   Here we will use the command LA 01050 05221

   v-part!

   Before the shifting

   SAR = 000 101 010 010 001 (binary)

   k

   After the shift of 17_{10} places (notice that k = $21_8 = 17_{10}$) we have:

   SAR = 000 101 010 000 000 (binary)

   The one's transmission $01050_8 \rightarrow$ SAR results in the following:

+ (one's transmission)	000	101	010	000	000	(binary)
	000	001	000	101	000	(binary)
	000	101	010	101	000	(binary)
or	0	5	2	5	0	(octal)

   Therefore, we now send the answer $(A_R)_f$ to 05250_8, as intended. Notice that you always have to carefully figure out the v-part of the LA-command, if you wish to send the answer to a register different from u. Also notice that this register must have an address which has to be at least larger than u by 200_8. In fact, it has to be larger by $n \cdot 200_8$ than u, n = 0, 1, 2, 3, ...

3) Shift (01050) 17_{10} places in A and leave the answer in A.

Here the command LA 01050 32021 will give us the desired result, because

SAR after the shift	= 011 010 000 000 000	(binary)
$u = 01050_8$	= 000 001 000 101 000	(binary)
	011 011 000 101 000	(binary)

As you see SAR = 33050_8 which denotes A.

Notice that <u>any</u> core-register may be used for this purpose, i.e. for shifting in A and leaving the answer in A. (Compare this with the remarks made on the LQ u k for leaving the answer in Q. See below).

Generally speaking the above mentioned address r can be easily found by subtracting at first k from the number which makes up the v-part of the LA u k. Then address u has to be added to this in binary such that 1 + 1 = 1. Doing so with the above examples we have:

1) $v = 00021_8$, $k = 21_8$, $v - k = 00000_8$, $u = 01050_8$.

Therefore $u + (v - k) = 01050_8$

(↑ logical sum, 1 + 1 = 1)

2) $v = 05221_8$, $k = 21_8$, $v - k = 05200_8$, $u = 01050_8$.

Therefore $u + (v - k)$: $000\ 001\ 000\ 101\ 000_2$

$+\ 000\ 101\ 010\ 000\ 000_2$

$000\ 101\ 010\ 101\ 000_2$

$u + (v - k) = 05250_8$

(↑ logical sum, 1 + 1 = 1)

3) $v = 32021_8$, $k = 21_8$, $v - k = 32000_8$, $u = 01050_8$.

Therefore $u + (v - k) = 33050_8 = A$

(↑ logical sum, 1 + 1 = 1)

The LQ uk instruction works in the same way with the exception that u is now placed into Q and shifted there. Again SAR is used like in the LA u k. Thus you can send results to registers different from u.

There is only one situation which requires special discussion. This is the case that you wish to shift a number in Q and leave it there.

You have seen that you may use any core-address in order to shift a number in A and leave the result there. You will do that by giving a LA u 32000 + k command, as e.g. LA 01050 32021. This may give you the idea that you may do the same with a LQ u k command, i.e. may try to shift (02000) in Q and leave the result in Q by giving a LQ 02000 31003 (shift 3 places). This, however, is <u>not</u> true. The result will be sent to A (that it also stays in Q is beyond any doubt). The reason is that

$$\begin{array}{rl} u & : \ 000\ 010\ 000\ 000\ 000_2 \\ \text{SAR after shift} & : \ \underline{011\ 001\ 000\ 000\ 000_2} \\ & \quad 011\ 011\ 000\ 000\ 000_2 \end{array} \quad + \text{(one's transmission)}$$

results in 33000_8 in SAR which is the address of A.

It can easily be seen that the following addresses may be used in order to <u>leave the result in Q without sending it to any other place</u>:

00000	thru	01777	(octal)
10000	thru	11777	(octal)
20000	thru	21777	(octal)

No other addresses will accomplish this.

g) <u>The "Left Transmit" Command</u>

As it was said earlier: the computer always picks up (A_R), if a number has to be obtained from A. There is only one command which takes the content of A_L provided that the programmer specifies this. It is the LT jk v.

Sequence:

Shift $(A)_i$ k places left

If j = 1: $(A_R)_f \longrightarrow X$

If j = 0: $(A_L)_f \longrightarrow X$

In either case:

$(X) \longrightarrow v$ *

* if v = A: Clear A

$(A) + D(X) \longrightarrow A$

The u-portion of the command contains the number jk, such that

$$\underbrace{xxx}_{j}\ \underbrace{xxx\ xxx\ xxx\ xxx}_{k}$$

As you see: the rightmost 7 bits of the u-portion denote the number k, but <u>one</u> bit is used for determining j. All other bits are disregarded by the machine during the execution of the LT-command. They also do <u>not</u> affect the LT itself or the sequence of steps to follow, if they are different from zeros.

The j is, programwise, given by the leftmost octal digit of the u-portion of the LT jk v. Therefore:

$0 = 000_2$ $2 = 010_2$ $4 = 100_2$ $6 = 110_2$	all result in a "machine - j" of "0"	$1 = 001_2$ $3 = 011_2$ $5 = 101_2$ $7 = 111_2$	result in a "machine - j" of "1"

This means:

j is an even digit: $(A_L) \longrightarrow X$

j is an odd digit: $(A_R) \longrightarrow X$

This decision "even or odd" is made by the machine such that it "examines" the bit UAK_{12} (do not forget: during its execution the LT is in PCR).

h) <u>The "Split" Commands</u>

All four split commands have in common that the single-extension of a number is added to (or subtracted from) A.

SP u k

Sequence:

$(u) \longrightarrow X$

Clear A

$(A) + S(X) \longrightarrow A$

Shift (A) k places left

The number k denoting the number of places (A) has to be shifted is, as in the LA-and LQ-commands, given by the rightmost 7 bits of the v-portion of the SP u k command. The remaining 8 bits of this v-part are insignificant for the execution of this and the other 3 split commands. They, however, affect the sequence which follows with the next MP 6 (see under "Program Address Counter"). This requires a detailed discussion.

The question is: what is left in SAR, after the shifting of k places has been made? Obviously SAR will contain all zeros in the rightmost 7 bits, but the leftmost 8 bits will be equal to those stated in the v-part of the SP u k. Example:

(05202) = SP 03025 02405

Here, k = 5. Thus, after the shift we have

SAR = 000 010 100 000 000 (binary)

SK, now = 0

It is important to understand that SAR is not cleared at the end of the SP u k. The computer proceeds with the normal MP 6 (see page 2). That means: PAK → SAR by one's transmission. If, as given above in the example, the SP-command is stored at address 05202_8, then PAK contains 05203_8. Let us see what number is generated in SAR by PAK → SAR:

```
    SAR = 000 010 100 000 000      (binary)
    PAK = 000 101 010 000 011      (binary)
final SAR = 000 111 110 000 011    (binary)
```

As MP 6 and MP 7 state: the next command is to be extracted from the location whose address is held in SAR. This is now 07603_8. At the same time PAK holds the address 05204_8 since it was advanced by "1".

Therefore we do the following:
At first the SP at 05202_8 is executed normally. The next instruction is extracted from 07603_8. If this is not a jump we proceed at 05204_8. As you see: only 05203_8 has been omitted! Naturally, if the v-part of the SP-command contains the number k only (i.e., the leftmost 8 bits of the v-portion are all zeros), SAR will be equal to "zero" after the shift, and we will proceed in sequence. This is the normal way of using the SP-command.
It is pointed out that you may not arrange the v-part such that the next instruction had to be taken from A or Q. If you do so, you get an SCC-Fault in case the address of the accumulator results from PAK → SAR. If you end up with Q, no fault will be generated. However, the steps to follow depend upon the content of this arithmetic register!

All following three split commands use SAR in the same way as the SP u k. The sequence following the shifting is, therefore, equal to that discussed for the SP-command.

In general one can say for <u>all</u> split commands:
The instruction following the split command is extracted from location

PAK + (v - k)

(↑ Boolean logical sum, 1 + 1 = 1)

where v denotes the 5 octal digits which make up the v-portion of the split command.

SA u k

Sequence:
(u) → X
(A) + S(X) → A
Shift (A) k places left

SN u k

Sequence:
(u) → X
Clear A
(A) - S(X) → A
Shift (A) k places left

This can also be denoted as

$[S(X)]^1 \rightarrow A$
Shift(A) k places left

but the explanation found sometimes which says

$S(X)^1 \rightarrow A$ etc.

is completely wrong. Reason:
Assume (X) = 7———76 (octal). Then:

$S(X) = 0———0\ \ 7———76$
$[S(X)]^1 = 7———7\ \ 0———01$
but $(X)^1 = \qquad\quad 0———01$
$S(X)^1 = 0———0\ \ 0———01$

Notice the difference between $[S(X)]^1$ and $S(X)^1$.

SS u k

Sequence:
(u) → X
(A) - S(X) → A
Shift (A) k places

A final evaluation of all four sequences shows:

The difference between SA and SP is:
in the SA the step "Clear A" is omitted.

The difference between SS and SN is:
the step "Clear A" is omitted in the SS-command.

Otherwise the command itself points out (Split Positive Entry, Split Subtract, etc.) whether a single extension is added to or subtracted from A.

i) The "Q-Controlled" Commands

These commands are merely mentioned here without any discussion of details. There are just three things you have to keep in mind when you perform a "masking" operation:

1) The logical product L(Q)(u) is a bit-by-bit product, such that

$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

This bit-by-bit product is a pure logical operation and may not be mixed up with a (true) product of two numbers.

2) If in any of the three "Q-Controlled" commands u or v is A or Q, then watch out. The result may be different from what you expect. Some examples are:

QT Q v	$(A)_f = S(Q)$ $(v)_f = (Q)$
QA Q v	$(A)_f = (A)_i + S(Q)$ $(v)_f = (A_R)_f$
QS u A	$(A)_f = L(Q)(u)$
QS u Q	$(Q)f = (Q)^1 + L(Q)(u)$
QS Q A	$(A)_f = S(Q)$
QS Q Q	$(A)_f = 2^{36} - 1$

3) The logical product L(Q)(u) is developed in X. Then the single-extension of this number is added to A, $S(X) + (A)_i \longrightarrow A$. Therefore, (A_L) is e.g. zero (36 zeros in binary) at the end of any QT-command. It is also zero at the end of any QS-command, regardless of (Q), (u), and (v).

j) The "Controlled Complement" Command

The CC u v makes use of A_R during its execution. It does not clear or use A_L. A_L is undisturbed by the CC-command.

A_R is used to develop the logical sum of (u) and (v). This sum is developed such that 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 0.

Remember 1 + 1 = 0 is different from the result of the Boolean logical sum. There we had 1 + 1 = 1.

The logical sum applied by the CC can also be denoted as "bit-by-bit sum without carries".

Example:

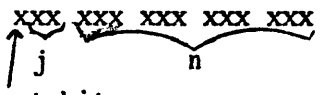
CC u A $\quad (A_R)_f = 0,\ (u)_f = 0$

k) The "Repeat" Command

The Repeat Command, RP jn w, repeats the next instruction several times, modifying it after each execution as specified.

The u-portion of the "Repeat" command contains the number jn, where n is given by the 12_{10} rightmost bits and j by the next two bits.

The leftmost bit of the u-portion is not used for the determination of j. It affects, however, the termination of the repeat sequence! This will be discussed later.



leftmost bit

n denotes the number which specifies, how many times the next instruction is to be executed.

j denotes how the u- and v-part of the next instruction is to be modified after each execution.

$j = 00_2$: do not modify NI (NI = next instruction)

$j = 01_2$: modify v-part of NI

$j = 10_2$: modify u-part of NI

$j = 11_2$ " modify u- and v-part of NI.

As you see: nothing has been said so far about the leftmost bit shown in the above picture!

The modification of NI is done by adding a "1".

Basically we have to distinguish between two cases:

the NI is a jump-command,

the NI is not a jump-command.

Let us begin with the second case.

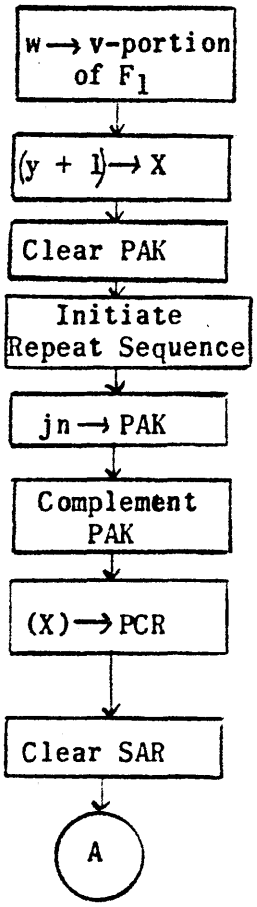
1) The repeated instruction is not a "jump" -command.

An example is e.g.

```
RP 10100     w
TP 05000 06000
```

(05000) = 0. Here registers 06000 thru 06077 will be cleared, since j = 1, n = 100_8.

In order to understand, why this is so and what happens in the machine that might affect programming we have to examine the sequence of the RP-command. This is given below in flow chart format and assumes that the Repeat command is stored at address y:



Address w is sent to the v-part of F_1, where F_1 = 00000 or 40001 as determined by the switch on console.

The NI is sent to X. PAK which holds y + 1 is now free for other operations.

PAK is cleared and the "Repeat" Sequence initiated (the latter affects several flip-flops)

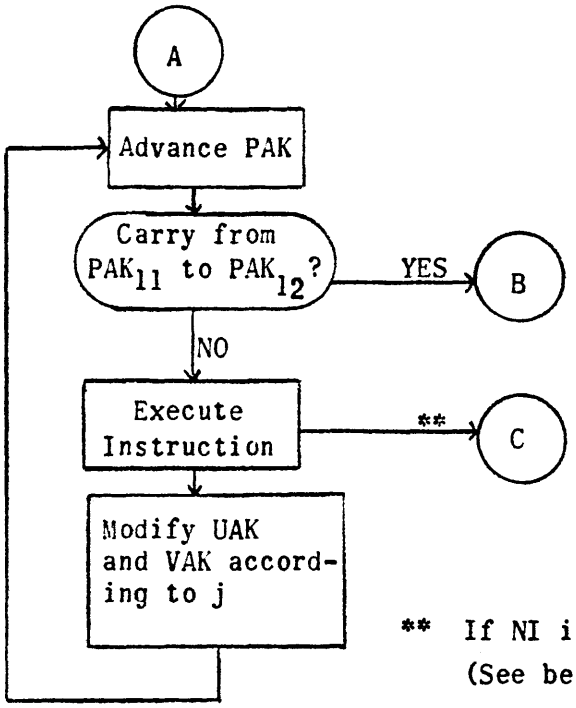
The u-part of the RP jn w is sent to PAK. Then this is complemented. (The above example would give us: $10100_8 \rightarrow$ PAK, $PAK^1 = 67677$)

The NI which was sent to X in step 2 is now placed into the Program Control Register and ready for execution.

SAR is cleared for later use.

At this point the execution of the Repeat command is terminated. Notice that we made proper use of w, and jn, and that we also "told" the machine to start with a Repeat sequence.

The repetition of the instruction following the RP jn w is performed by the se quence:



This sequence shows, how the machine determines, whether or not the NI has been executed n times. As you can see the carry from PAK_{11} to PAK_{12} tells the machine this fact. If it occurs the NI is not executed anymore, but the so called "Normal Repeat Termination Sequence" begins at (B). (see later)

** If NI is an EJ or TJ and a jump occurs! (See below)

Let us follow the above sequenee with the example

```
  a RP 10003   a + 2
a+1 TP 00005   06000
a+2 -- ------  ------
```

At first the address a+2 is sent to the v-portion of F_1. Then the complement of 10003_8 is sent to PAK, so that now PAK = 67774_8, and TP 00005 06000 → PCR. Starting at (A) we want to accomplish the following:

```
TP  00005  06000
TP  00005  06001
TP  00005  06002
```

and resume operation at a+2. Will we do that?

At first we advance PAK:

```
  67774
+     1
-----
  67775
```
(octal)

There was no carry. Therefore we execute TP 00005 06000, and then modify PCR such that now (PCR) = TP 00005 06001

Advancing PAK again we have:

```
  67775
+     1
-----
  67776
```
(octal)

No carry; therefore TP 00005 06001 is executed and PCR modified to TP 00005 06002.

Advancing PAK the third time we get:

$$\begin{array}{r} 67776 \\ +\quad 1 \\ \hline 67777 \end{array} \quad \text{(octal)}$$

Again there was no carry. We execute TP 00005 06002 and modify PCR to TP 00005 06003.

PAK is advanced again:

$$\begin{array}{r} 67777 \\ +\quad 1 \\ \hline 70000 \end{array} \quad \text{(octal)}$$

Here the carry from PAK_{11} to PAK_{12} occured. At this time we already executed the three TP-instructions as we intended to do. It is, therefore, all right that we do not go on executing the TP 00005 06003 which is in PCR, but go to (B).

Do not forget: PAK = 70000_8 at this time. This will be important later, if special cases are discussed!

You can see that the modification of the TP was done in PCR, not in the storage location a+1. Keep in mind that the content of the storage register holding the instruction to be repeated is not changed.

So far we have talked about a carry from PAK_{11} to PAK_{12}. This carry generates the "End Repeat Signal" which terminates the repeat sequence at once. Here you will probably remember the restrictions for carries in PAK as described under "Program Address Counter". How do these restrictions affect the above mentioned carry which is to terminate the repeat sequence?

Notice: the MCS-Section switch does not affect PAK as long as a "Repeat Sequence" is being performed. * (See under "Program Address Counter").

This means that a carry from PAK_{11} to PAK_{12} can be generated by the machine, if and only if $PAK_{14} = 1$, regardless of the number of core banks. Therefore, a Repeat Sequence can be terminated only, if the number j was a 0,1,2, or 3, because the complement of either of these four numbers results in a leftmost bit equal to "1".

Keep in mind: j = 4,5,6, or 7 results in an unterminated Repeat Sequence. The modification of the NI, however, will be performed such that a j = 4 equals a j = 0
j = 5 equals a j = 1
j = 6 equals a j = 2
j = 7 equals a j = 3, because this is determined by the machine by "looking" at PAK_{13} and PAK_{12}.

* To be precise: as long as the "End Repeat" flip-flop is "1".

As it was mentioned earlier: the repeated instruction is in PCR and the modification takes place there. It means that UAK and VAK are modified according to j. Both, UAK and VAK are counters, and the question arises, what restrictions for carries in these counters are established. The answer is:

UAK and VAK possess exactly the same restrictions for carries as PAK, including everything mentioned about the MCS-Section switch. The only difference is that during a Repeat Sequence and the MCS-Sect. switch in Normal, there will be a carry from UAK_{11} to UAK_{12} and VAK_{11} to VAK_{12} regardless of the value of UAK_{14} or VAK_{14}, respectively, but no carry from PAK_{11} to PAK_{12} unless $PAK_{14} = 1$.

Let us follow some examples: (assuming 3 cores)

```
1) a RP  17000  a+2
   a+1 TP  12000 05000          (12000)i = 0———0
   a+2 =-  ----- -----
```

First case: MCS-Section switch set to SINGLE:

Here we clear registers 05000 thru 07777
and 00000 thru 03777.

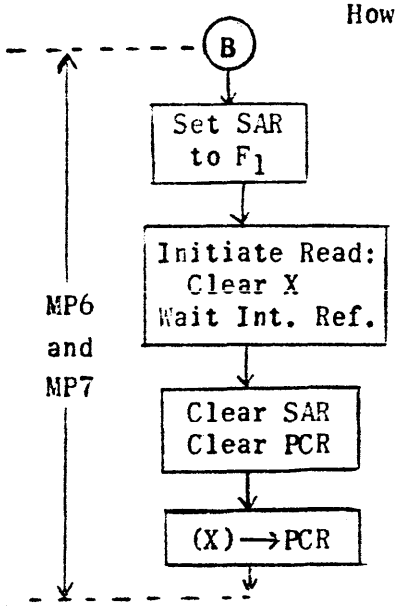
(v-part of TP is in VAK: VAK = 05000_8. Modifying this with the restriction that no carry from VAK_{11} to VAK_{12} occurs, gives us the above result.)

Second case: MCS-Section switch set to NORMAL:
Now we clear registers 05000 thru 13777, i.e. 7000_8 consecutive storages. (Carry in VAK is enabled by switch.)

As you see: If the machine possesses two or three banks of core storage, and you use the Repeat command, then give the operator a note, how you want the MCS-Section switch to be set (as you write down switches for Manual Jump and Manual Stop.) The normal case is, that it will be set to NORMAL. (Otherwise computer operation can be started in Test Mode only).

We are now ready to continue. We executed the NI n times and came to the "Normal Repeat Termination Sequence". Before we discuss it let us try to find out, what we have to do. We want to come back to the instruction at w, i.e. resume operation with the instruction whose address is given by w. (This will often be a+2, as shown in examples, but it need not be that).

How are we doing this?



The sequence shows: without using or changing PAK, we set SAR to the fixed address F_1. "Looking" at this address in SAR the computer reads the word from the storage into X and finally into PCR. Remember that the v-portion of this word which now is in PCR contains our address w.

The next Main Pulse is MP0, i.e. the word in PCR is executed. Remember what we tried to accomplish. We wanted to come to address w. This can be done only if PCR contains a jump command. Therefore:

Let (F_1) = MJ 00000 30000. The v-part of it was erased and replaced by w at an earlier time. This (F_1) which now reads MJ 00000 w is transferred to PCR (by the above termination sequence) and executed.
As you see: we jump to w and continue there.
Keep in mind: In order to continue at address w after the normal termination of a Repeat Sequence we have to have an unconditional jump at F_1. This jump will erase PAK and replace it by w.

This situation points out what will happen if F_1 does not contain a jump. Assume we have:

```
      a RP 17000  a+2
    a+1 TP 12000 05000
    a+2 -- ----- -----
```

and

```
    F1 TP 20000 30000
```

After executing the TP at a+1 7000_8 times we continue at F_1 according to the termination sequence. Remember that PAK = 70000_8. Executing (F_1) means to transfer a word from 20000_8 to a+2. PAK has not been changed. Since the computer continues with the normal MP 6 as shown under "Program Address Counter", the next instruction will be extracted from 70000_8. This is now the address where we continue our program after finishing the repeat sequence and the execution of F_1.
You can see:

If F_1 does not contain a jump-command the computer proceeds

```
at 70000,   if j = 1 was used
at 60000,   if j = 2 was used
at 50000,   if j = 3 was used
at 40000,   if j = 0 was used
```

At this point let us summarize what we know so far. We perform a Repeat Sequence, where the instruction to be repeated is not a jump-command. In any case this instruction is executed n times and the computer picks up the command stored at F_1 provided that a j = 0,1, 2, or 3 was used. If F_1 contains an uncoditional jump we will go to w and proceed there. If F_1 does not contain a jump, it is executed and the computer proceeds at one of the above mentioned drum addresses according to j.

A j = 4,5,6, or 7 sets up an unterminated repeat sequence and will normally not be used. There are, however, some situations where a programmer might use them with advantage.

It is also pointed out here that the whole repeat sequence is regarded by the computer as being completely finished, after the command at F_1 has been executed.* This is important for the Interrupt Feature which will be discussed later.

Let us discuss the second case:

2) The repeated instruction is a "jump"-command: We have to divide the jump-commands into two groups:
one group contains EJ, TJ
the second group contains all others.
If we forget about the EJ and TJ for a moment, we can see that in all other jump-commands there is only the alternative to jump immediately or never to jump. Take a MJ 10000 v. If the switch is set: unconditional jump. Therefore

```
RP j w
MJ 10000 v
```

would result in the following: during the very first execution of the MJ the jump occurs which erases PAK and replaces it by v. This means that the repeat sequence is terminated immediately. In general we can say: If the instruction to be repeated is a RJ, IP, QJ, SJ, ZJ, PS or MS the Repeat Sequence is automatically terminated.

* At that time the "Hold Repeat Flip-Flop" is finally "0" again. At the beginning of the "Normal Termination Sequence" which starts with a MPb, this Flip-Flop is still a "1". See: "Interrupt Feature".

These instructions behave, as if no RP precedes them. If the instruction to be repeated is an IJ or MJ and a jump is called for, the Repeat Sequence is terminated immediately. If no jump is called for (IJ uv with $(u)_i = 0$, MJ j v with j = 1, 2, 3 and switch not set) the instruction is repeated n times and the next instruction is taken from F_1, i.e. the repeated instruction is treated like a "normal" command.

The two commands EJ uv and TJ uv represent special cases. Assume you have a table of 100_8 numbers stored at 07000, 07001, ... etc. You wish to compare another number which is in A with this table in order to find out whether or not it is equal to at least one of the numbers in the table. At the same time you are interested in the address of the number in the table which is equal to (A). How can this be accomplished?

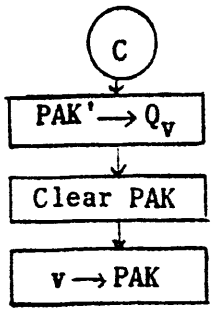
The answer is: (F_1) = MJ 00000 30000

a	RP	20100	a+2
a+1	EJ	07000	v
a+2	—	———	———

It might be easy for you to see that we compare (A) with (07000), (07001) etc. and that we jump out of the Repeat Sequence immediately, if an equality is found. Then we continue at address v. But how do we find the address of the number in the table which caused the jump?

Refer to the flow chart on page 29. There you see that we go to (C), if a jump occurs during the repetition of an EJ or TJ. Let us discuss the steps following at (C).

"Jump Termination Sequence" (for EJ and TJ only)



As you can see: the number which is in PAK at the time an equality is found, is complemented and sent to the v-portion of Q. (Before this is done Q is cleared, so that the operation part and the u-part of Q contain zeros.) Then PAK is cleared, and the address v from EJ u v is placed into PAK, i.e. a jump to v is completed.

As soon as this sequence is finished the whole Repeat Sequence is regarded by the machine as being terminated. * The next Main Pulse is a normal MP6. Keep this in mind for the Interrupt Feature.

What does the complement of PAK which now is in Q represent? Let us follow our example shown above and assume that (07003) = A.

At first (jn)' = 57677 is in PAK. Therefore:

	First Time	Second Time	Third Time	Fourth Time
Advance PAK	57700	57701	57702	57703
↓ Carry from PAK_{11} to PAK_{12}?	NO	NO	NO	NO
↓ Execute	EJ 07000 v	EJ 07001 v	EJ 07002 v	EJ 07003 v
↓ Jump?	NO	NO	NO	YES

As you see: PAK = 57703, when the jump occurs. The complement goes to Q which gives us (Q) = 00 00000 20074
We executed the EJ 4 times. Let us subtract 4 from jn:

$$\begin{array}{r} 20100_8 \\ -\quad 4 \\ \hline 20074_8 \end{array}$$

This is also our number in Q!

In general you execute the EJ exactly r times ($1 \leqq r \leqq n$). If an equality occurs during the r-th execution, then the number

$$jn-r$$

is sent to Q.

How do we get the address in the table? In our example we have to generate 07003_8. We could do that by $07000_8 + 4 - 1$. This would mean that we had

* As said earlier: the "Hold Repeat" Flip-Flop is "0".

$$(A_R)_f = (u) \cdot 2^s,$$

is given by

$$\underline{s = 72_{10} - k_{10}, \quad \text{if } 37 < k \leqq 71,}$$

If k = 0, then s = 0.

The above case makes use of an address u which is not the Accumulator address. Let us, therefore, give the command SF A v with

$$(A)_i = \underbrace{0010\text{——}0}_{A_L}\ \underbrace{0\text{——}0}_{A_R} \quad \text{(binary)}$$

Here $(A)_i$ is shifted, until $A_{35} \neq A_{34}$. As a result we obtain

$$(A)_f = \underbrace{0\text{————}0}_{A_L}\ \underbrace{010\text{————}0}_{A_R}$$

As you see: $(A)_i$ is "scaled down" in this particular case. This "scaling down" cannot happen in the case of SF u v, whereD(u)⟶A first.

What is the number k in the above example?

$k = 35_{10}$. We actually multiplied $(A)_i$ with 2^{-35}.

Therefore, s has to be = -35.

In general:

If SF A v is used and $\underline{0 \leqq k \leqq 36,}$ then

$\underline{s = - k.}$

If k turned out to be in the range

$37 < k \leqq 71$

then s = 72 - k as stated earlier.

We did not include the value $k = 37_{10}$ so far.

Notice: $\underline{\text{If } k = 37_{10}}$ then (A) contains "zero" or all 72 bits of (A) are "ones".

The number k which is sent to the v-portion of v denotes how many places $(A)_f$ would have to be shifted to the left, if one tried to bring it back to its original value D(u). Notice that the operation part and the u-part of (v) are not affected by the transmission $k \longrightarrow v_{14} \ldots v_0$.

Example:

(u) = 00 00000 00004

(v) = 13 00235 45670

What are the contents of A and v after the execution of SF uv? Answer:

$(A)_f$ = 0——0 010——0 (binary)

A_L A_R

or: $(A)_f$ = 00 00000 00000 20 00000 00000 (octal)

How many places would you have to shift $(A)_f$ to bring it back to the original value D(u)? Look at $(A)_f$ and count: shift 2 places and the "1" is in the rightmost position of A_L. Shift 36 more places and the "1" is in the rightmost position of A_R. Now shift 2 more places and you have $0——0100_2$. Altogether we shifted $36 + 2 + 2 = 40_{10}$ places.

Therefore $k = 40_{10} = 50_8$ and $(v)_f$ = 13 00235 00050.

You know that a shift of one place to the left is equivalent to a multiplication with 2 (provided that the most significant bit of the number is not shifted into the sign-position or even further). How did we multiply (u) during the application of the SF-command? We began with

0———0100 (binary)

and finished with

010———0 (binary)

As you see we actually shifted 32_{10} places. If you denote this number with s, then

$$s = 72_{10} - 40_{10} = 32_{10}$$

in our case.

In general:

Given is SF uv with u not A-address
Here D(u)⟶A and is shifted as stated above.
The only possibility for k is

$$k = 0 \text{ or } 37 < k \leq 71$$

The number (u) is "scaled up" in A. The number s, where

$$(A_R)_f = (u) \cdot 2^s,$$

is given by

$$\underline{s = 72_{10} - k_{10}, \quad \text{if } 37 < k \leqq 71,}$$

If k = 0, then s = 0.

The above case makes use of an address u which is not the Accumulator address. Let us, therefore, give the command SF A v with

$$(A)_i = \underbrace{0010\text{——}0}_{A_L}\ \underbrace{0\text{——}0}_{A_R} \quad \text{(binary)}$$

Here $(A)_i$ is shifted, until $A_{35} \neq A_{34}$. As a result we obtain

$$(A)_f = \underbrace{0\text{————}0}_{A_L}\ \underbrace{010\text{——}0}_{A_R}$$

As you see: $(A)_i$ is "scaled down" in this particular case. This "scaling down" cannot happen in the case of SF u v, whereD(u)⟶A first.

What is the number k in the above example?

$k = 35_{10}$. We actually multiplied $(A)_i$ with 2^{-35}.

Therefore, s has to be = -35.

In general:

If SF A v is used and $\underline{0 \leqq k \leqq 36,}$ then

$\underline{s = - k.}$

If k turned out to be in the range

$37 < k \leqq 71$

then s = 72 - k as stated earlier.

We did not include the value $k = 37_{10}$ so far.

Notice: $\underline{\text{If } k = 37_{10}}$ then (A) contains "zero" or all 72 bits of (A) are "ones".

V) The 1103A Input-Output System

a) The On-Line Electric Typewriter (Flexowriter)

The operation of this output-equipment is controlled by the command

PR - v

Generally speaking the execution of this Print-Command can be explained in the following way:

"Typewriter, perform one operation according to the two rightmost octal digits of the content of v."

As you see: the computer "looks" at the two rightmost octal digits of the number stored at address v. These two octal digits represent a code for the typewriter. The typewriter performs one operation which can be either a print-out of one character, i.e. one decimal digit or one letter or one sign

or a function as e.g. "Carriage Return", "Shift up", "Space," etc.

Notice: one Print-Command causes the print-out of one character

(or the performance of a function)

In order to print out the word "1103A" how do we have to program? Assume we store the codes for this word in register 02000_8 such that

```
(02000) = 52  52  37  70  47  30
          ↑   ↑   ↑   ↑   ↑   ↑
codes for 1   1   0   3       A
                              |
                          Shift Up
```

We want to make sure that typing will start in "shift down" position which gives us big numbers, but small letters. Therefore after the "3" has been typed we have to "shift up".

The program is:

```
        a   PR  00000  02001
      a+1   LQ  02000  00006
      a+2   PR  00000  31000
      a+3   IJ  02002  a + 1
      a+4   --  -----  ------
```

with

```
02001  00  00000  00057   "Shift Down" Code
02002  00  00000  00005   Index
```

The above example shows how you will print out some information, if you know this information at the time you write the program.

But how do we print out a number, say the content of a register in octal, if this number is unknown to us, because it is a result of some computation?

Assume we wish to print out all 12_{10} octal digits contained in 03000. We do it in the following way:

```
   a  LQ  03000  00003    Shift (03000) 3 places in Q
 a+1  TP   b-1  32000    Dummy print command ⟶ A
 a+2  QA   b-2   a+3     Set up print command in a+3
 a+3 [00  00000 00000]   Print one octal digit
 a+4  IJ   b-3     a     Printed all 12₁₀ digits?
 a+5  --  -----  -----

 b-3  00  00000  00013    Index
 b-2  00  00000  00007    Extractor mask
 b-1  PR  00000     b     "Dummy" print command
  b   00  00000  00037
 b+1  00  00000  00052    Codes for octal
 b+2  00  00000  00074
 b+3  00  00000  00070       digits
 b+4  00  00000  00064
 b+5  00  00000  00062    0 thru 7
 b+6  00  00000  00066
 b+7  00  00000  00072
```

In order to understand the method used here assume e.g. (03000) = 12 34507 65432 and follow the above program step by step.

At this time you will certainly like to know how the computer "tells" the typewriter what to print. The method used is also applied for other input-output equipments and is, therefore, discussed in the following paragraphs.

The Print-Command makes use of a special register, the so called Typewriter Register TWR. This is a "buffer" register, because it functions like a buffer between computer and typewriter. Actually the six rightmost binary bits from (v) are sent to TWR, and the typewriter performs its operation by sensing the combinations of "ones" and "zeros" in TWR. However, a six-bit-code can be sent to TWR, if and only if a previous print-operation has been completed by the typewriter. This machine possesses a certain speed. It prints out approximately 10 characters per second. This means that approximately 100 msec. are needed for the printing of one character. This is a very long time compared to the speed of the computer.

Assume you give a print-command, and 10 m sec. later you execute another print-command. The code sent by the first PR to TWR is still there, when the second code tries to enter TWR. Naturally this has to be avoided, since the second transfer to TWR would result in the logical sum of both codes. Therefore, TWR possesses a so called "lockout" which prevents a transmission from X to TWR as long as the previous operation has not

been completed, After the completion of it TWR is cleared and the "lockout" is removed. The next code may now enter TWR. Keep in mind: As long as a print operation is performed and a second print operation is about to take place the computer has to wait for the termination of the first print-out, until it can initiate the second one. The computer "hangs up" temporarily.

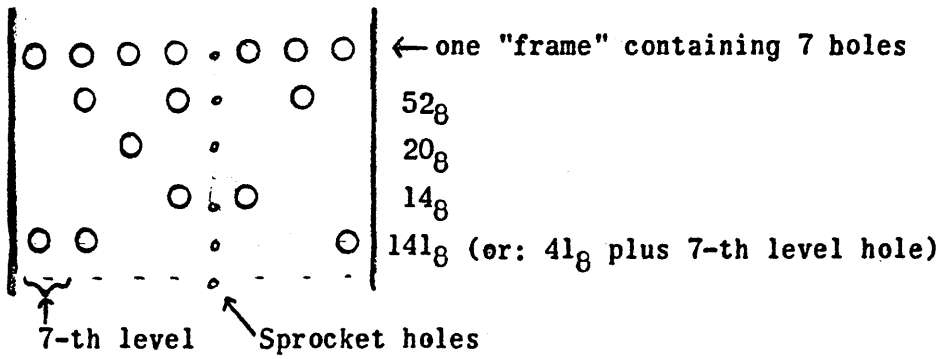
However, the execution of the Print-Command is performed within 34 micro-seconds, i.e. 34 micro-seconds after the beginning of a PR -v the computer can go on with the execution of another command. It always will do so! Only if this command (or one of the commands following within about 100 m sec.) is another PR -v the above mentioned situation (temporary "hanging-up") will occur.

The rather slow speed of the typewriter means that this output equipment may be used for a print-out of short results or some parameters which indicate the flow of operation in the computer to the operator (programmer). Never use it for printing out long tables of results etc. You waste computer time!

You have seen that you always have to send a code to the typewriter. But not all 6-bit-numbers are legal codes for this output equipment. The code 000 000_2 = 00_8 is e.g. no legal code. The question is: what will the typewriter do if a code is sent to it which he does not "understand"? In this case computer operation is stopped with a "Print Fault" ("A" Fault).

b) The High Speed Punch Unit

This output equipment punches holes into a 7-level paper tape. The format of the tape is described below:



The paper tape contains small holes which identify a "frame" on tape. These small holes are the so called Sprocket Holes.

One frame may contain up to 7 holes as shown above. A hole represents a binary "1", the absence of a hole denotes a binary "0".

The operation of the High Speed Punch Unit is controlled by the command

PU j v

Punch v_5 .....v_0 on one frame.

If j is an <u>odd</u> octal digit: <u>also</u> punch a 7-th level hole

If j is an <u>even</u> octal digit: do <u>not</u> punch a 7-th level hole.

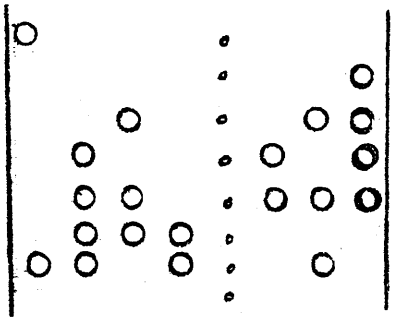
j is determindd by the binary digit in UAK_{12}, as explained under "Left Transmit" command.

Before we discuss how to punch out numbers or letters, we have to ask the question: If a paper tape has been punched out somehow, where can we read the information contained on it?

There exist two units which may be used: the Flexowriter and the Paper Tape Preparation Unit. Both are off-line equipments.

The Flexowriter is exactly the same electric typewriter which is used as on-line equipment to print out information by means of a PR -v. As off-line equipment it is capable of reading paper tapes and printing out the information <u>provided the paper tape has been printed out in Flex-Code</u> (as mentioned under "On-Line Electric Typewriter"). This means that the programmer when writing a Punch Routine has to make up his mind whether or not he wants to read the tape with the Flexowriter. If he intends to do so, he only needs to apply the programs given under "On-Line Electric Typewriter", where each Print-Command has to be replaced by a Punch-Command. j = 0, since Flexowriter cannot "read" a 7-th level hole.

The Paper Tape Preparation Unit is capable of punching, reproducing and reading paper tapes. The method used by this equipment for the representation of data on paper tape is the following:



00_8 plus 7-th level hole

01_8

23_8

45_8

67_8

70_8

52_8 plus 7-th level hole

<u>Each frame represents two octal digits (6 bits) of a 36-bit computer word.</u> Therefore 6 frames represent one 36-bit word. The 7-th level hole is used by the unit during the reading of the tape such that "finding" a 7-th level hole means to print out in octal all digits between the preceeding 7-th level hole and the one just found (the two octal digits being on the same frame as the last 7-th level hole are also printed out).

Without going into more details as far as the operation of the Tape Preparation Unit is concerned let us summarize the facts which are important for punching in this manner (so called "bioctal"):

One frame has to contain <u>two</u> octal digits. In order to initiate printing during the off-line reading procedure include a 7-th level hole (this will be at least on each 6-th frame!)

A typical example for punching in "bioctal" is the following program starting at a:

```
  a   PU  10000   a+2
a+1   LQ    u     00006
a+2   PU  00000   31000
a+3   IJ    b     a+1
a+4   LQ    u     00006
a+5   PU  10000   31000
 :
 :
 b    00  00000   00004
```

Follow this program step by step with any number (u) and prepare a picture of what will be punched out.

As you might guess: there does not exist any fault in connection with the Punch-Command, because you do <u>not</u> send a <u>code</u> to the High Speed Punch Unit. Whatever the last 6 bits of the content of v are they will be punched out on paper tape!

The speed of the High Speed Punch Unit is 60_{10} frames per second, i.e. one Punch-Command is completely executed and the frame punched out after approximately 16.7 m sec. This is about 6 times as fast as the typewriter. Therefore, whenever "flex code"output is desired use the High Speed Punch instead of the typewriter.

The execution of a Punch-Command is delayed, if a previous punch operati has not been finished. This is accomplished in the same way as for the PR -v; there is naturally a different "buffer" register which is used. This is the High Speed Punch Register HPR.

C) "Input-Output" Commands EF -v, ER jv, EW jv.

The Electric On-Line Typewriter and the High Speed Punch Unit are the only two external equipments which possess their own commands. All other external equipments have to be handled by the "external" commands EF, ER, and EW.

Basically there are two different operations which have to be made by programming:

Selection of an external equipment and specification of what it is to do (EF)

Program Control of the flow of information between computer and external equipment (ER, EW)

1) The EF-v instruction (general)

A general explanation of the execution of the EF -v is the following:

(v)⟶IOB (Input-Output Register B)

Select an external equipment and cause it to perform an operation. (Both selection and initiation of an operation are done by examining the content of IOB)

This means that the programmer specifies by the number he puts into v, which equipment and which operation he wants to select. The content of v has to be arranged by placing "ones" into special positions of the register. These are the so called "Bit Assignments". They will be explained later during the discussion of each equipment.

2) The Information Flow from and to External Equipments.

Very often the operation of an external equipment results in a flow of numbers from the external equipment to the computer (Input) or from the computer to the external equipment (Output). The numbers to be transferred may consist of up to 36 bits.

As you have seen during the discussion of the Print and Punch-Commands an exchange of numbers between computer and external equipment is made by using special registers, so called "buffer" registers. It is important that you understand this: All external equipments use either the IOB-register or the IOA-register or both. Only the Typewriter and the High Speed Punch possess their own buffer registers, TWR and HPR.

IOB (Input-Output Register B) is a 36-bit register

IOA (Input-Output Register A) is an 8-bit register

Both registers can be used for input or output purposes.

As it was said for TWR and HPR already: "buffer" registers possess a so called "lockout". This "lockout" is discussed in the following paragraphs.

It is obvious that the speed of an external equipment, any external equipment, is much slower than the speed of the computer. Because of that fact the following two situations will occur often:

During ext. equip. to computer transfer; the computer tries to pick up information from IOA or IOB, before the external equipment sent it to these registers,

During computer to ext. equip. transfer; the computer tries to send information to IOA or IOB, before the external equipment has picked up the previous information from these registers.

Let us assume the first of the above two situations occurs, i.e. the computer tries to pick up a number, say, from IOA, before the external equipment sent it there. Naturally, we have by all means to avoid the execution of the computers "intention". This is done automatically by the machine. A good understanding of the way it is done is absolutely necessary for a programmer, if he wants to be a good programmer.

The basic concept which led to building in a "lockout circuitry" is the following:

If the computer communicates with an external equipment, then

delay computer operation, if the computer is ahead of the external equipment; indicate a fault, if the external equipment is ahead of the computer.

Since the "lockout" sets up also certain situations for particular operations as e.g. parity error indication sent to IOA during reading of magnetic tape etc., it is necessary to discuss its functioning now.

3) The IOA-and IOB-Lockouts, ER jv, EW jv.

Let us look at the IOA-lockouts, since the IOB-lockouts are functioning in exactly the same way, and study one of the above mentioned two cases, namely an external equipment $\longrightarrow$computer transfer. The other transfer, computer $\rightarrow$ external equipment, is similar and will, therefore, be explained briefly only.

External Equipment $\longrightarrow$ Computer Transfer:

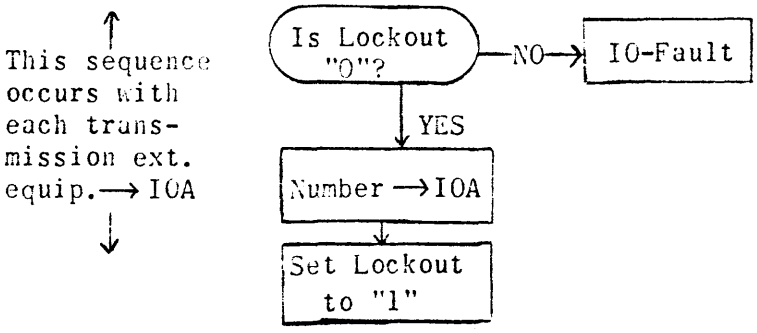
Assume you select an external equipment such that it will send one number after the other to IOA. The computer naturally has to pick up each number from IOA (by an ER-instruction; explained later) and place it into its memory. Say, the speed of the external equipment is such that one number is sent to IOA every 4.3 m sec. (speed of Ferranti Reader, discussed later).

What does this mean? It means that between the picking-up of two consecutive numbers the computer has a time of 4.3 m. sec. for other operations. It also means that the computer may not be too late, e.g. try to pick up a number every 10msec., because at that time two numbers tried to enter IOA already.

As programmers we can think of two different sequenees which take place in order to accomplish a delay of computer operation or to produce a fault: one sequence occurs automatically according to the speed of the external equipment and is, therefore, under ext. equip. control,
the other one takes place with each ER jv instruction and is, therefore, program controlled.

Under External Equipment Control:

Referring to the above mentioned example you see that once you select a data transfer to IOA to occur every 4.3 m sec. you set up an automatic sequence. Each time a number is sent to IOA by the external equipment, this equipment will also test whether or not the preceeding number has been taken away from IOA by the computer. With other words the external equipment finds out whether or not IOA is "empty" again. This "test" is made in a very simple way, namely by "examining" the state of one Flip-Flop, the so called "Wait Read IOA FF". Let us drop too many engineering details and discuss this "IOA Read Lockout" (how we will call it) from our programming point of view:



*

You see: if the "IOA Read Lockout" is in the "0" state, the external equipment is satisfied. A number is sent to IOA, and the lockout set to "1". If the lockout was not "0", i.e. it was a "1", then a special "B"-Fault is generated, called "IO"-Fault.

If you examine the above sequence closely, you will find that the computer, when placing(IOA) into its memory, must set the "IOA Read Lockout" back to "0", thus indicating to the external equipment that this transfer has been made. On the other hand the computer at first has to examine the state of the lockout. It may proceed, if this lockout is a "1", because this means that the external equipment sent a number to IOA already. Otherwise the computer has to wait, until this will be accomplished. All this is made by the sequence

* At this time the "second" number is also transferred to IOA. This means that at the time of the IO-Fault IOA contains the logical sum (1+1=1) between two numbers.

Under Program Control:

In order to pick up a number from IOA or IOB and to place it into a storage location, we possess the command

ER jv

If j = 0: (IOA) $\longrightarrow v_7 \ldots\ldots v_0$

$v_{35} \ldots\ldots v_8$ all zeros

If v = A:

(IOA) $\longrightarrow A_7 \ldots\ldots A_0$

$A_{71} \ldots\ldots A_8$ all zeros

If j = 1: (IOB) $\longrightarrow$ v

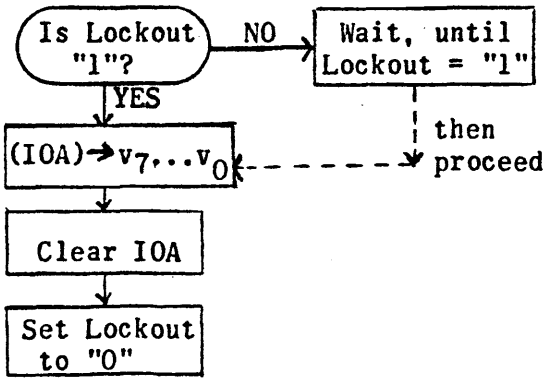
If v = A:

D(IOB) $\longrightarrow$ A

If v = Magnetic Drum location: SCC-Fault!

j is given by the leftmost octal digit of the u-part of the ER-command. It is determined exactly like in the Punch-Command, i.e. an even digit corresponds to j = 0, an odd digit corresponds to j = 1.

This command, as explained earlier, has to do some more things than just accomplish a transfer as stated above. It also has to examine the lockout and set it to "zero" at the end. Therefore, we have the following sequence for each ER 00000 v:



If the "IOA Read Lockout" is a "1" the computer "knows" that a number has been transferred to IOA by the ext. equip. It, therefore, picks it up, clears IOA (in preparation for the next transfer from ext. equip. to IOA) and sets the lockout to "0". If the lockout was a "0" at the moment the execution of the ER begins, this execution is delayed, until the ext. equip. sent a number to IOA.

You can also see that a failure to execute the ER-command in time will leave the lockout in its "1" state. The external equipment, when examining this lockout, thus finds out that an ER has not been executed in time, and therefore, the IO-Fault is generated.

The same situation is true for IOB. Therefore, if a data transfer from external equipment to computer via IOB is made, the external equipment as

well as each ER 10000 v examine the "Wait Read IOB FF". This is handled in exactly the same way and need not be explained again. But keep in mind that both IOA and IOB possess a "Read Lockout".

Now let us briefly discuss the transfer Computer → External Equipment: The basic idea is this: At first the computer has to transfer a number to IOA (or IOB). The ext. equip. then has to pick up this number from IOA (or IOB). Again we want to make sure that the computer may not try to insert a number, say, into IOA, before the ext. equip. made use of the previous number, i.e. the computer may wait, if it is ahead of the external equipment. On the other hand the external equipment has by all means to find a number in IOA, when it is ready to receive it. Therefore, if such a number is not present in IOA, i.e. if the transfer computer → IOA did not occur in time, a fault is to be generated. This fault is a "B" Fault and represents a "No Information" Fault.

These situations are handled by another Flip-Flop, the so called "Wait Write IOA FF". There are again two sequences, one under Program Control (by an EW jv) and another sequence which is under External Equipment control. They are similar to the "Read" Sequences and are, therefore, not discussed here in details.

The data transfer from computer to IOA or IOB is made by the command

EW j v

If j = 0: $v_7 \ldots\ldots v_0 \longrightarrow$ IOA

If j = 1: (v) ⟶ IOB

If v = Magnetic Drum location: SCC-Fault!

As already said: in addition to the data transfer the EW-command also tests the "Write Lockout" condition and sets it at the end such that the external equipment will be "satisfied" when testing it.

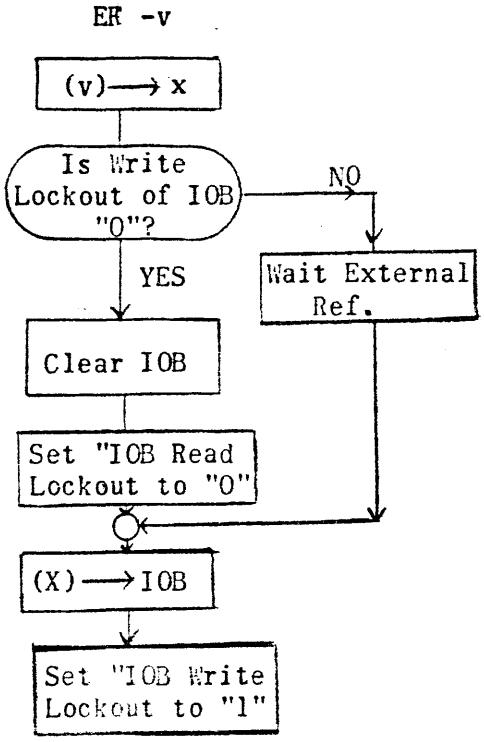
4) The EF -v instruction (details)

Now we are ready to discuss the EF-instruction in details. This instruction has been modified slightly during 1957 (See: External Function Modification, PX 150, Sept. 57). The only modification which was made refers to the following situation:

An EF -v is given at a time when IOB still contains "Read" information deposited there by an external equipment.

Previously, such a condition resulted in the Borlean logical sum (1 + 1 = 1) of both the "Read" information and the (v) which was supposed to select another equipment. Therefore, both numbers were lost, no fault was detected, and nothing could be said about the selection which took place.

At present the EF -v sequence prevents the above occurence of the above mentioned case. If it detects that IOB still contains "Read" information, it clears IOB, inserts (v) into IOB, and sets the "IOB Read Lockout" to "0" thus preventing the execution of an ER 10000 v, until new "Read" information has been placed into IOB. We can describe this situation by the following sequence:



As you see: the IOB Write lockout is tested. If it is "1", the execution of the EF is delayed, because IOB still contains "Write" information sent to it by a previous EW (or EF). If it is "0" no such "Write" sequence occured lately. However, in any case IOB is cleared and the IOB Read Lockout set to "0". Thus the machine keeps track of the above mentioned case (whether it occured or not). At the end the IOB Write Lockout is set to "1" preventing the execution of another EW (or EF), until this EF has been executed completely (actually: until the ext. equip. sent an IOB Resume signal).

d) The Ferranti Paper Tape Reader

This input device reads 7-level paper tape. As the tape moves through the reader 7 photocells associated with the seven levels of each frame sense the holes of the frame and send them as binary "ones" into IOA. Therefore, the frame

| O O ◦ O O |

would result in (IOA) = 00 101 110

(Notice: only the rightmost 7 bits of IOA are used by the reader, i.e. IOA_7 is always a "0"). The transmission of a frame to IOA is made, whenever the reader senses a sprocket hole.

Bit Assignments:

IOB_{33} = 1 Select Reader ("Master Bit")

IOB_{16} = 1 Start

IOB_{15} = 1 Stop

The following combinations are possible:

1) Start Reader, Free Run:

This selection is made by an EF -v instruction, where (v) = 10 00002 00000 (octal). It means that the Reader is started and will from now on send one frame after the other to IOA. The time between successive transfers is 4.3 m sec. (safe time).

2) Stop Reader:

Here (v) = 10 00001 00000 (octal)

This selection causes a stop of a Free Run operation. Keep in mind that one more frame is sent to IOA after the Stop instruction has been executed. If you want to use IOA again, make sure that you clear it and set the lockout to "0" by giving one more ER 00000 v.

3) Step Reader:

If it is desired to read just one frame and then stop the reader an EF -v may be executed with (v) = 10 00003 00000.

Notice that the "Master Bit" IOB_{33} has to be present in all codes which refer to the Ferranti Reader. You can keep in mind the general rule: the "Master Bit" of an equipment has to be present in (v) each time a reference to this equipment is made by an EF -v instruction.

The correct timing for start and stop delays may be found in the Univac Scientific Programming Manual, U 1519. (It is only pointed out here that e.g. punching with a PU jv may not be attempted during a Free Run operation of the Reader.)

dd) Drum Zone Selection:

The computer Master Clear selects Drum Zone A. A switching to Zone B is made by an

EF -v with $v_{26} = 1$, $v_{15} = 1$

i.e. (v) = 00 04001 02000

Switching to Zone A is made by an

EF -v with $v_{26} = 1$ only,

i.e. (v) = 00 04000 00000

After selection of one zone all commands referencing 40000 thru 77777 automatically refer to the registers of this zone, until the next selection is made.

e) The "Bull" Card Unit as On-Line Equipment

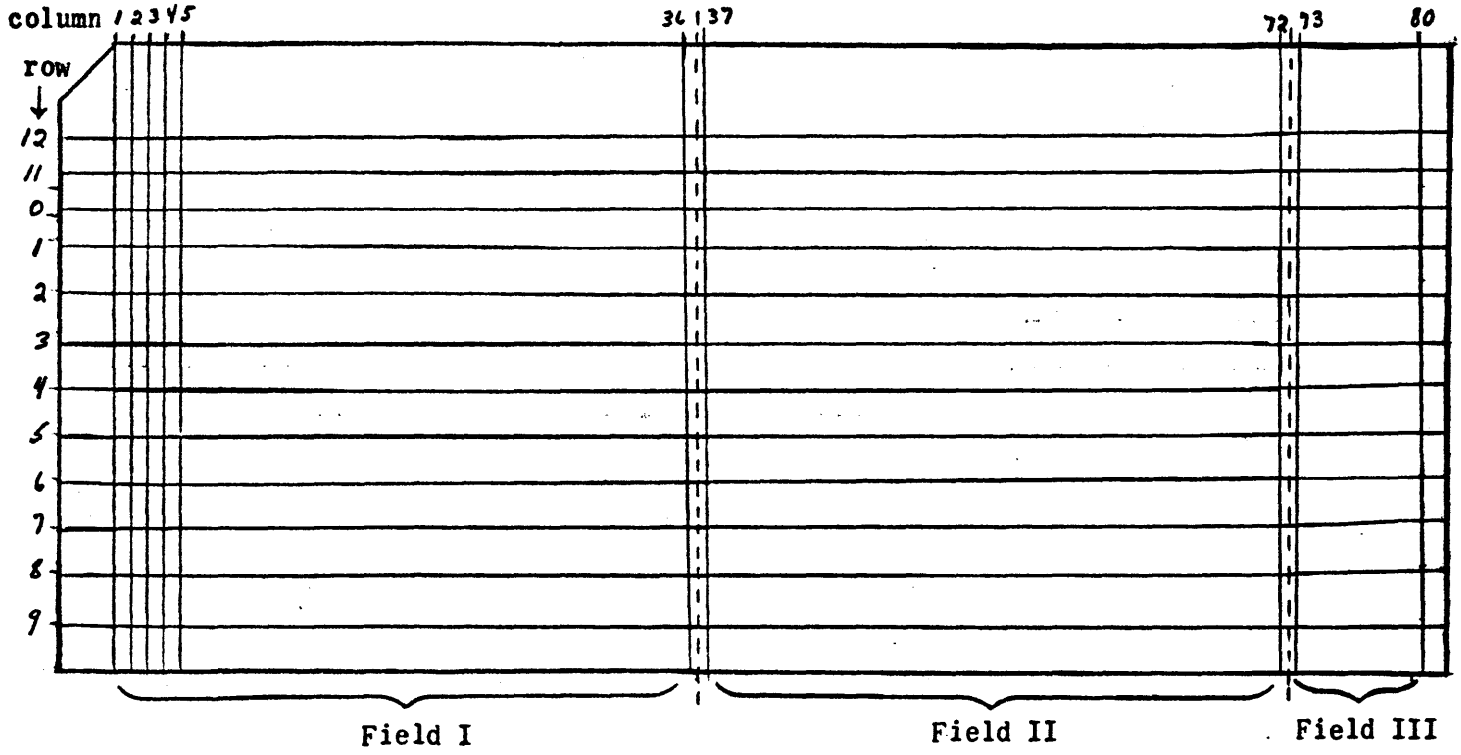
1) The 80-Column Card

The card is divided into 12 horizontal rows and 80 vertical columns. A rectangular hole may be punched at the intersection of each row and column.

Each card is divided into three fields:

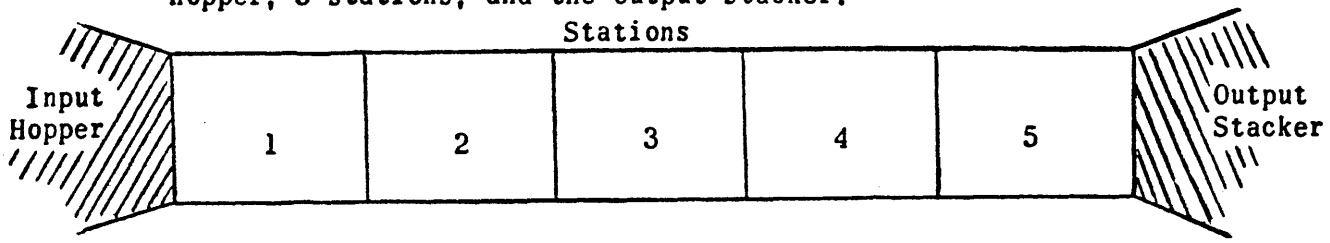
Field I = Columns 1 thru 36
Field II = Columns 37 thru 72
Field III = Columns 73 thru 80

The following figure shows columns, rows, and fields of a card:



2) General Description of Read-and Write-Channels

The "Bull" Card Unit possesses two channels, the right-hand "Read" channel and the left-hand "Write" channel. These channels consist of an input hopper, 5 stations, and the output stacker:



The card on the bottom of the input hopper is picked up and placed into station 1;
the card which is advanced from station 5 into the output stacker is placed on top of the stack.
The reading of a card is done in station 2 of the "Read" channel, whereas the punch mechanism has to be set after the card arrived in station 3 of the "Write" channel. Actual punching is performed during the advancing of the card from station 3 to station 4.

3) Selection of Card Unit Operations, Bit Assignment

Any operation of the card unit has to be initiated by an EF -v instruction, where (v) contains the proper bits for the operation to be performed.
The following operations are possible:

Select Card Unit (Start Cycle) - $v_{35} = 1$ ("Master Bit")

This bit selects the card unit as external equipment and must, therefore, be present in any (v) of an EF -v which refers to this unit. It also advances all cards in station 2 through 5 one station in both channels, i.e. card in station 2 goes into station 3, ..., card in station 5 is placed on top of stack in output stacker.

Pick Read Card - $v_2 = 1$

This bit causes the card unit to pick a card from the base of the "Read" channel and to place it into station 1. It also advances the card in station 1 of the "Read" channel into station 2.

Pick Punch Card - $v_3 = 1$

Similar to "Pick Read Card".

Read - $v_0 = 1$

This bit enables the sensing brushes to "read" the card which moves from station 1 to station 2 in the "Read" channel. It does not initiate a movement of the card from station 1 to station 2.

Punch - $v_1 = 1$

This selection enables the punch mechanism to receive information from IOA and IOB, and prepares it to punch the card at the beginning of its movement from station 3 to station 4.

Free Run - $v_5 = 1$

This bit causes a continuous operation of the card unit by retaining all selections made together with the Free Run bit in the same (v) of EF -v.
The Free Run has to be stopped by programming an EF -v, where (v) contains $v_{35} = 1$ and

Stop Free Run - $v_4 = 1$

All selection made for the Free Run operation are dropped by this bit. However, one more cycle of operations as performed during the Free Run will be executed after the "Stop" has been given.
Example: If cards are read in Free Run and the "Stop" is given by an EF -v instruction one more card will be read by the card unit. The program has, therefore, to provide the proper instructions for reading this card as described later.

Interrupt - $v_7 = 1$

This bit causes the Interrupt Facility of the 1103-A (or 1105) to work in connection with the card unit. An explanation of this will be given during the discussion of the Interrupt Feature.

4) Reading of Cards

A card is read row after row beginning with row 9. Thus the last row to be read is row 12.

A hole at the intersection of any row and column is transmitted to the computer as "1", the absence of a hole denotes a "0". Since one row contains 80 "ones" and "zeros" in any combination it is impossible to read the whole row at a time. The transmission of the data of one row is, therefore, made in three steps:

1. Field III = 8 bits → IOA
2. Field I = 36 bits → IOB
3. Field II = 36 bits → IOB

Thus the following three instructions have to be executed for each row:

ER 00000 u (IOA) → u_7 ..... u_0
ER 10000 v (IOB) → v
ER 10000 w (IOB) → w

To read the whole card this group of instructions has to be executed 12 times. The transmission of data from Field III to IOA may be omitted by manually setting the "Enable Field III" switch on the Card Unit Control Cabinet to its "out" position. In this case the ER 00000 u must not be executed, (see under "Faults"). For reading all three fields the above mentioned switch has to be set to its "Normal" position.

5) "Writing" on Cards

In order to punch data on a card the punch mechanism has to be set for each row beginning with row 9. Again three instructions have to transfer the information, which is to be punched into one row, to IOA and IOB, respectively, This information has to be given in the sequence:

for Field III
" I
" II

Thus the following group of three instructions has to be executed for each row:

1. EW 00000 u $u_7 \ldots\ldots u_0 \longrightarrow$ IOA
2. EW 10000 v (v) $\longrightarrow$ IOB
3. EW 10000 w (w) $\longrightarrow$ IOB

If the "Enable Field III" switch is set to its "out" position the EW 00000 u must not be executed (see under "Faults"). If this switch is set to "Normal" this EW 00000 u has to be executed.

6) Summary on Programming "Read" and "Write" Operations

a) Read Operation

1. The card to be read must be picked up from the read input hopper by an EF -v, where (v) = 40 00000 00004. This instruction places the card into station 1.
2. In order to advance the card from station 1 to station 2 the "Pick Read Card" must be given again. In addition the "Read" selection has to be made now. These operations are initiated by EF -v, where (v) = 40 00000 00005. Now the card which is to be read is placed into station 2, but another card is picked from the input hopper and placed into station 1 of the "Read" channel. This cannot be avoided even if it is not intended to read this second card.
3. Now the group of three instructions as described in paragraph 4) has to be executed 12 times.
4. In order to move the card just read, out of the read channel an EF -v with (v) = 40 00000 00000 has to be executed 4 times. Notice that the card which was read is placed into the output stacker, but the second card still remains in station 1.

b) Write Operation

1. The card to be punched must be picked up from the write input hopper and placed into station 1 by an EF -v, where (v) = 40 00000 00010.
2. Another EF -v with the same (v) has to place this card into station 2, but will also pick one more card and place it into station 1.
3. The card in station 2 must be placed into station 3 and the punch mechanism enabled for receiving information. This is done by an EF -v instruction, where (v) = 40 00000 00002.
4. Now the group of three instructions as described in paragraph 5) has to be executed 12 times.
5. In order to move the card just punched, out of the write channel an EF -v with (v) = 40 00000 00000 has to be executed 3 times.

   Notice that the card which was placed into station 1 during step 2 still remains in this station.
   Exact programming and timing for reading and punching cards is given in the appendix.

7) Faults

a) Program Faults:

During a "Read" Operation the following situations might occur:

1. If the 3 ER's are not given 12 times for each card, i.e. each row is not picked up from IOA and IOB, respectively, an IO-Fault (B-Fault) occurs. This is due to the fact that the first row for which no ER's are provided causes a transmission to IOB twice. Since the first data (from Field I) have not been removed the second transfer of data (from Field II) will find the IOB-lockout still in its "1" state thus causing the IO-Fault.

2. If the time available between rows is exceeded, i.e. the 3 ER's are not given early enough, the computer is also stopped with an IO-Fault due to the lockout condition of IOA.

3. If the Enable Field III switch is set to "out" and the ER 00000 v is initiated the execution of this instruction cannot be performed, since no data have been transferred to IOA. Thus the two following ER's for IOB cannot be executed which again results in an IO-Fault as described under 1.

4. If the Enable Field III switch is set to "normal" and the ER 00000 v instruction is omitted the second transfer of data to IOA causes an IO-Fault as mentioned under 2.

During a "Write" operation the same 4 situations might occur as during the reading process. They all result in a computer B-Fault, and the "No Information" Light on the Card Unit Control Cabinet is illuminated.

No Card in Reader:

If card reading is supposed to occur and no card is present in the reading station a computer B-Fault is generated and the "No Card in "Reader" light on the Card Unit Control Cabinet is illuminated.

No Card in Punch:

If punching is attempted and no card will be present beneath the punch die during the next cycle (i.e. no card is present in station 3 which could be punched during its movement from 3 to 4) a computer B-Fault is generated and the "No Card in Punch" light on the Card Unit Control Cabinet is illuminated.

b) Other Faults:

The following fault conditions will cause the card equipment to stop illuminating the "stop" light on the Card Unit Control Cabinet:

1. Read output stacker full
2. Write output stacker full
3. Read input hopper empty
4. Write input hopper empty
5. "Stop" button depressed
6. Punch jam occurs
7. "Standby" switch to forward position
   (i.e. away from operator of card unit)

Notice that these conditions stop the card unit only, but not the computer. As a result computer operation continues, e.g. if cards are read and the read input hopper is empty the card unit stops and the program hangs up, if the first attempt is made to read (IOA). Operation can be resumed after placing cards into the read input hopper.

Notice also that during a reading or writing operation at least one card has to be present in the input hopper not used during the operation.

8) Manual Preparation of Card Unit for Program Controlled Operation

The card unit is prepared for program controlled operation by following the steps below:

1. Set the "Enable Field III" switch in the Control Cabinet to the position as required by the program (see paragraphs 4) and 5)).
2. Place the cards into the read and write channel face (printed side) down, so that row 9 enters the channels first. Place the metal weights on top of decks.
3. Set the switches on the card unit in the following manner:

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

Notice: In order to clear both channels manually before starting the program the following operation has to be performed:

1. Remove all cards from input hoppers.
2. Set PICK READ and PICK PUNCH switches away from operator.
3. Press "Start" and "Clear" button on the card unit simultaneously, until all cards left the channels.
4. Place cards into channels and do not forget to set PICK READ and PICK PUNCH towards operator.

9) In addition to the timing mentioned in the appendix keep in mind that the theoretical time which is available to the computer between reading or writing consecutive rows is approximately 27,8 m sec.

f) The 1103A Magnetic Tape System, 1105 Bypass Mode Operations (Uniservo II)

The paragraphs below contain a description of the Magnetic Tape System, as it is used with an 1103A computer equipped with Uniservo II Magnetic Tape Handlers. However, they also represent a description of either one of the Magnetic Tape Control Units of the 1105 computer and the programming of Bypass Mode operations, as performed there. (1105 computers are always equipped with Uniservo II's). 1105 programmers may, therefore, carefully study the explanations given below, since they will not be repeated.

1103A computers equipped with Uniservo I Magnetic Tape Handlers require slightly different programming methods for reading tape in Fixed and Variable Block Length Format. Informations about these methods is omitted in this manual and may be found in the appropriate literature.

I. Fixed Block Length

A) Representation of Data on Tape

One line of magnetic tape consists of 8 bits: 6 data bits, a parity bit, and the sprocket bit.

The sprocket bit is used during the reading process. It is always a "1" and causes a sprocket pulse thus indicating that data (and not blank space) is being read.

The 6 data bits represent the information stored on the tape. They may represent two octal digits in 1103-A (or 1105) machine language or a 6 bit character in any code, e.g. in Univac Excess Three code.

The parity bit is automatically inserted on each line during the writing process. It makes the number of "ones" on each line odd (sprocket bit not included) enabling the equipment which reads Unitape to check the accuracy of the information on a line.

B) Tape Format

The tape is divided into blocks. Each block consists of 6 blockettes, one blockette consists of 120_{10} lines. Therefore; 1 block = 6 blockettes = 720_{10} lines = 120_{10} computer words.

Blocks are separated by block spaces of either 1.2" or 2.4".

The blockette space may be 0", .1" or 1.2". When writing on tape a high or low density (lines per inch) can be selected. It is 200_{10} and 128_{10} lines per inch, respectively.

C) Registers of the Magnetic Tape Control Unit

The following registers are automatically employed, if a magnetic tape operation is selected.

Tape Register TR	36 bits: 6 lines are assembled in TR during the reading process. It also holds a word which is to be written on tape.
Tape Control Register TCR	14 bits: when a tape operation is selected IOB_{25} thru IOB_{12} is transferred to TCR. It selects and controls the operation to be performed with a Uniservo.
Align Input Register AIR	7 bits: it holds the 6 data bits plus the parity bit during a reading operation.
Tape Shift Counter TSK	3 bits: counts the shifting of TR
Line Counter LK	3 bits: if LK = 6, it initiates a pulse which advances the WK by 1, and is cleared.
Word Counter WK	5 bits: if WK = 20_{10}, BTK is advanced by 1 and WK is cleared.
Blockette Counter BTK	3 bits: if BTK = 6, it initiates an "end of block" signal and is cleared. During the "move" operation it also sends a signal to BK decreasing it by 1.
Block Counter BK	12 bits: it holds IOB_{11} thru IOB_0 during a "move" operation, i.e. the number of blocks to be moved forward or backward.

D) Selection of Magnetic Tape Operations

A reference to the magnetic tape system has to be made by an EF -v instruction where (v) contains a "1" in v_{31} ("Master Bit").

Notice that the execution of an EF -v instruction just places (v) into IOB. $v_{31} = IOB_{31} = 1$ will then cause a transmission of IOB_{25} thru IOB_{12} to TCR which now can select the operation as specifed in (v).

The following tape operations and contents of v are possible:

Operation	Contents of v includes bits for
Read forward (or backward)	Select Magnetic Tape Read forward (or backward) Uniservo number Stop (see "Free Run")
Write	Select Magnetic Tape Write, Density Block space, Blockette space Uniservo number Stop (see "Free Run")

Operation	Contents of v includes bits for
Move forward (or backward)	Select Magnetic Tape Move forward (or backward) Uniservo number Number of Blocks to be moved
Rewind	Select Magnetic Tape Rewind (or Rewind with Interlock) Uniservo number
Change Bias	Select Magnetic Tape Change Bias to Low (or High or Normal) Notice: No other operations may be specified in this (v).

E) Discussion of Modes of Operation

1) Read Forward

6 data bits plus parity bit are sent to AIR. Now the parity check is made. In any case the 6 data bits are transmitted to TR_{35} thru TR_{30}, and AIR is cleared. (TR) is then shifted left 6 places and the next line sent to TR_{35} thru TR_{30} etc. 6 left shifts of 6 places each will take place altogether. After one complete word is assembled in TR it is sent to IOB and TR is cleared. Now (IOB) can be removed into a storage by an ER 10000 v instruction.

Since it is only possible to read complete (not part of a block as e.g. one word) blocks the programmer must provide 120_{10} ER's for each block he intends to read.

2) Read backwards

This operation is performed like the "read forward". Only exception is: (TR) is shifted right 6 places and only 5 right shifts will take place.

3) Write

There is only a "write forward" operation because of the physical structure of the Uniservo. This unit possesses an Erase Head which is located about 4" ahead of the Read/Write Head. During the "write" operation the Erase Head erases the old informations which might be on the tape. Therefore, writing may be started either at the very beginning of the tape or at a point where the previous writing operation has been stopped.

An EW 10000 v instruction has to transfer (v)$\longrightarrow$IOB. The computer then transfers (IOB)$\longrightarrow$TR. This is shifted left 6 places and TR_5 thru TR_0 written on one line of tape; a parity bit and sprocket bit is also generated for each line. After (TR) has been shifted 6 times and 6 lines have been written on tape TR is cleared and the next transfer (IOB)$\longrightarrow$ TR may occur. The machine also inserts block and blockette spaces as specified by the EF -v instruction which initiated the "write" operation. Notice that only complete blocks can be written on tape. Therefore, 120_{10} EW's have to be given for each block.

4) Move forward or backward

When this mode of operation is selected a transfer IOB_{11} thru $IOB_0 \rightarrow BK$ occurs. For each block passed a "1" is subtracted from (BK).

If (BK) = 0 tape movement stops automatically.

5) Rewind

The selection of a "Rewind" operation for a Uniservo causes this tape to be rewound to its very beginning and then to stop automatically. As soon as the "rewind" operation was initiated TCR is cleared and the tape control unit free for other tape operations. After a "Rewind with Interlock" no further reference to this Uniservo can be made by the computer until the interlock is removed manually at the Uniservo.

6) Change Bias

Provision has been made for a change of the "read" bias which enables the programmer to re-read a block after a parity error by either suppressing too strong signals ("noise") or reading weak signals also.

Notice that a "change bias" operation will change the bias for all Uniservos.

7) Free Run Operations

A "read" or "write" operation may be done either for one block or any number of blocks. If only one block is to be read (written) the "stop" bits may be included in (v) of the EF -v instruction which selects reading (writing). If more than one block is to be read (written), the operation can be done in Free Run such that no "stop" bits are included in (v). After the proper number of blocks has been read (written) another EF -v must be executed where (v) now contains the Select Magnetic Tape bit and the "stop" bits. The Uniservo number need not be selected again, since TCR still contains the bits which selected the read (write) operation, and the second transfer of IOB_{25} ...IOB_{12} to TCR ("one's transmission") will provide "stop" bits in TCR. Thus Uniservo number and stop bits are present in TCR causing a stop signal for the Uniservo in action which will be effective if the "end of block" signal from BTK is also present. After the stop TCR is cleared and another tape operation may be initiated.

F) Checks made during "Read" Operation

1) Parity check

During a "Read forward" or "Read backward" operation a parity check is made by the computer for each line on tape. If one or more parity errors occur within a block the following indication is given to the programmer at the end of the block:

a) "1" $\rightarrow IOA_0$ and IOA - lockout is set to "1"

b) Tape movement stops in interblock space.

Therefore the content of IOA has to be examined by the programmer at the end of each block, e.g. by giving an ER 00000 A and then testing (A). After a parity error was found the programmer may provide a change of the bias level and re-read the block.

2) Sprocket Error Check

In addition to the parity check the computer counts the number of lines N of a block. If $N > 720_{10}$ or $N < 720_{10}$, a socalled Sprocket Error occurs. In this case:

a) $1 \longrightarrow IOA_3$, set IOA-Read Lockout to "1"
b) Tape stops automatically

Reading and testing (IOA) at the end of each block reveals the occurrence of this error.

3) End of Block without Error:

If neither a parity nor a sprocket error occurred, the IOA-Read Lockout is merely set to "1" in order to enable the program to execute the ER 00000 v at the end of each block. Thus, finding (IOA) = 0 the programmer knows that the current block was read successfully.

4) Occurrence of Parity and Sprocket Error:

If both errors are detected for a block, IOA_0 and IOA_3 are set to "1". Thus, finding $(IOA) = 11_8$ the programmer knows that this situation occurred. Tape movement was stopped automatically.

G) Most Frequent Programming Faults

1) An ER 10000 v instruction has to transfer (IOB) $\longrightarrow$ Core before the next transmission of (TR) $\longrightarrow$ IOB occurs. If (IOB) has not been removed at the time when (TR) $\longrightarrow$ IOB is due a computer B-Fault (IO-Fault) is initiated and the tape movement stops at the end of the block.

2) During the "Write" operation an EW 10000 v has to transfer a word to IOB thus enabling a transmission (IOB) $\longrightarrow$ TR. If the Tape Control Unit is about to execute this transfer and no word has been sent to IOB a computer B-Fault (MT-Fault) is initiated, the "No Information" light in the Tape Control cabinet or on the Desk Console is illuminated, and the tape movement stops at the end of the block.

3) "Too Many" EW 10000 v instructions will result either in a "No Information" Fault or in a "wait" position of the computer (i.e. computation "hangs up"). If more than $120_{10} \cdot n$ EW's are programmed for writing n blocks on tape in Free Run the first extra External Write is executed at the end of writing the n^{th} block and the word written on tape, after the block space was inserted behind this block since tape movement was not stopped by an EF stop instruction. Writing continues with the execution of the Extra EW's until an EF stop instruction is initiated. This stop, however, cannot become effective, because a transmission $IOB_{25} \ldots IOB_{12} \longrightarrow TCR$ is not possible except between blocks. Tape movement continues, and the "too many" EW's are now interpreted as "too few" EW's (for the $(n+1)^{st}$ block)

resulting in a No Information Fault.

If a "Write on Block and Stop" operation is programmed followed by more than 120_{10} EW's tape movement is stopped at the end of the block but the 121^{st} EW initiates an IOB lockout condition which is established when the next EW or EF instruction is attempted. Then the computer "hangs up".

4) "Too Many" or "Too Few" ER's also set up either faults or cause the computer to "hang up". Which situation occurs depends upon the operation and the amount of ER's executed. They are not discussed here, but the reader should try to find out what happens for various cases as e.g. Read one block and stop using 119_{10} ER's; read one block and stop using 118_{10} (or less) ER's etc. This should not be too hard for the reader if he understands the functioning of the IOB lockouts and the sequence of the EF - instruction as explained earlier, and will be a good exercise. If questions arise, refer to Programming Manual U 1519 and/or Manuscript Copy of Section on Magnetic Tapes, Oct. 30, 57.

5) If two Number Selection switches for the Uniservos are set to the same number a computer B-Fault (MT-Fault) is generated at the time the computer is started. In addition the "Selection Error" light on the Tape Control Cabinet is illuminated.

6) If a tape is "Rewound with Interlock" and a reference to this tape is attempted by an EF -v the following occurs:

   (v) of this EF is sent to IOB. Then IOB_{25} thru IOB_{12} is transferred to TCR. The selection of the tape operation, however, cannot be initiated because of the interlock. On the other hand the computer program continues with the instruction immediately following the above EF -v.

   What happens in the future depends entirely upon <u>further</u> references of any of the tape units (for 1105: tape units of the same TCU as the tape rewound with interlock). Sooner or later the computer will hang up with an EF, ER, or EW, if such references are made.

H) Miscellaneous

A "Rewind" operation initiated for a tape which is rewound already will not cause any fault. Computation proceeds normally.

A computer "Master Clear" sets the bias to normal.

The speed of the tape is 100_{10} inches per second. Notice that in Fixed Block Mode tape movement can be stopped in an interblock space only, never within a block.

J) Sample Programs (for Bit Assignments refer to the table in the appendix)

1) Rewind Uniservo 3. Then write one block of data from 05000, 05001, etc. on Uniservo 3 in low density, 1.2" Blockette and 1.2" Block Spaces. (High Speed Printer Format)

```
   a   EF  00000    b                      b    02  00200  30000
  a+1  EF  00000   b+1          1103A:    b+1   02  00656  30000
  a+2  RP  10170   a+4           1105:    b+1   02  00646  30000
  a+3  EW  10000  05000
  a+4  NI                                F1   MJ  00000  30000
```

2) Write 10_{10} blocks in Free Run, density etc. as in example 1. Then move back 10_{10} blocks.

```
   a   EF  00000    b          1103A:    b     02  00056  30000
  a+1  RP  12260   a+3          1105:    b     02  00046  30000
  a+2  EW  10000  05000                  b+1   02  00600  00000
  a+3  EF  00000   b+1                   b+2   02  00014  30012
  a+4  EF  00000   b+2
```

3) Read forward one block from Uniservo 8 into 06000, 06001, etc.

```
   a   EF  00000    b                      b    02  00603  00000
  a+1  RP  10170   a+3                    b+1   00  00000  00001
  a+2  ER  10000  06000                   b+2   00  00000  00010
  a+3  ER  00000  32000
  a+4  ZJ   a+5    OK
  a+5  EJ   b+1     P
  a+6  EJ   b+2     S
  a+7  Beginning of "Parity and Sprocket Error"
```

As you see: at the end of the block (IOA) is tested. If it is = 0, a jump to "OK" occurs, i.e. everything is fine and we continue in the normal program.

If (IOA) = 1, we know a parity error occurred. We, therefore, jump to the place where this Parity Error Routine begins (at address "P").

If (IOA) = 10_8, a sprocket error existed and a jump to "S" is made.

If (IOA) $\neq$ 0, $\neq$ 1, and $\neq 10_8$, it must be = 11_8. In this case a parity and sprocket error occurred. Starting at a+7 we, therefore, have the steps provided for this situation.

The steps to be taken after occurrence of errors will generally be the following: Re-read the block which caused the error by changing the bias level to high, low, and possibly normal again. If after several tries the data can not be read successfully, one can indicate this (possibly by a short Print-out on flexowriter) and stop the program.

II) Variable Block Length

A) Data Representation and Tape Format

In Variable Block Length one line on magnetic tape consists of 6 data bits, a parity bit, and a sprocket bit, i.e. it is equal to a line recorded in Fixed Block Length Mode.

Since only full 36_{10} bit words can be transmitted between the computer and the magnetic tape, each block must contain an integral number of words. Its length may, therefore, vary from one computer word up to the capacity of core storage.

Blocks are recorded with a high density, the inter-block space is always 1.4".

B) "Write Operation"

A "Write" operation is initiated by the execution of an EF -v instruction, where (v) has to contain the following information:

Select Magnetic Tape bit
Variable Block Length/Continous Data Input bits
Write Selection bits
Uniservo Number

One EW 10000 v instruction has to be executed for each word to be written on tape.

Notice that "stop" bits must not be included in the EF-instruction which selects the "write" operation. This is due to the fact that TCR is inspected for stop bits after each word. A "Write" operation has, therefore, to be stopped by programming an EF "stop" instruction. (See under D) after the execution of the desired number of EW's.

Also notice that density and inter-block spaces need not be selected. However, if bits specifying other density or block spaces than high density and 1.4" space are contained in (v) of EF -v which selected the "Write" operation, these bits are ignored.

C) "Read" Operation

A "Read Forward" (or "Read backward") operation is initiated by an EF -v instruction, where (v) has to contain the following bits:

Select Magnetic Tape
Read Forward (or Backward)
Variable Block Length/Continous Data Input
Uniservo Number
Stop Bits (if no Free Run desired)

One word is assembled in TR, and then (TR) is transferred to IOB. Thus, one ER 10000 v has to be executed for each word of a block.

Due to the nature of the Variable Block Length Mode the occurrence of the following situations requires a special indication in IOA:

Parity error, end of block detection, mod 6 error, and end of record detection.

1) End of Block Detection

The number of words contained in a block is variable and in general unknown to the programmer. However, each word has to be picked up by an ER 10000 v instruction. There may not be less or more ER's than words in the block. In order to enable the program to decide whether or not a word has been assembled the following indications are sent to IOA:

> If one word has been assembled from tape, the IOA-Read Lockout is set to "1". Thus testing IOA and finding (IOA) = 0 the program knows: there is a word in IOB. Therefore, pick it up by ER 10000 v. In other words:
>
> Read IOA first. If and only if (IOA) = 0, a word is in IOB and can be picked up. If (IOA) $\neq$ 0, no word is in IOB.

The end of block is detected after a lack of sprocket pulses for approximately 600 μ sec. If this time elapses without occurrence of a sprocket pulse

$1 \rightarrow IOA_1$; Set IOA-Read Lockout to "1".

Assume there is only one word in a block. You test IOA first, and since a word (the only one) has been assembled (IOA) = 0. Finding this you execute ER 10000 v. Now you return to your test of IOA. Because of the assumption that there is no other word (and provided no error occurs during reading) 600 μ sec. will elapse. Then $1 \rightarrow IOA_1$. Reading and testing IOA you find (IOA) = 2. This indicates to you that there is no word, but that the end of the block has been reached without a "Read Error". Therefore, do not execute an ER 10000 v!

During reading the tape parity or sprocket errors (here called: mod 6 errors) might occur. This is discussed in the following paragraphs.

2) Parity Error

If during reading of a block one or more parity errors occur, an indication of this situation is given to the program at the end of the block. This indication is:

$$1 \rightarrow IOA_0$$

This is made precisely at the time the end of block is detected, i.e. if a parity error (but not a mod 6 error) occurred, one will find

$$(IOA) = 3$$

because of $IOA_1 = 1$ (end of block)
$IOA_0 = 1$ (parity error)

Contrary to Fixed Block Length: a parity error in Variable Block Mode does not cause an automatic stop of tape movement.

3) Mod 6 Error (Sprocket Error)

A block has to contain an integral multiple of 6 lines. If 1 thru 5 lines are missing (a situation detected at the end of the block, naturally)

$1 \longrightarrow IOA_3$

Thus, assuming this error occurs only, one will find

$(IOA) = 12_8$

because of

$IOA_3 = 1$ (mod 6 error)

$IOA_1 = 1$ (end of block)

Tape movement is <u>not</u> stopped automatically.

4) Parity <u>and</u> Mod 6 Error

If both errors occur in a block,

$(IOA) = 13_8$

because of

$IOA_3 = 1$ (mod 6 error)

$IOA_1 = 1$ (end of block)

$IOA_0 = 1$ (parity error)

Again tape movement is <u>not</u> stopped automatically.

5) End of Record Detection

If during a "Read" operation, a lack of sprocket pulses is detected for a distance of approximately 4", an end of record signal generates

a) $1 \longrightarrow IOA_2$

b) Stop Tape movement automatically

6) Summary

The following contents of IOA are possible:

<u>Content of IOA (octal)</u>	<u>Condition</u>
0	Word in IOB
2	End of good block
3	End of block with parity error
12	End of block with mod 6 error
13	End of block with parity <u>and</u> mod 6 error.
4	End of Record

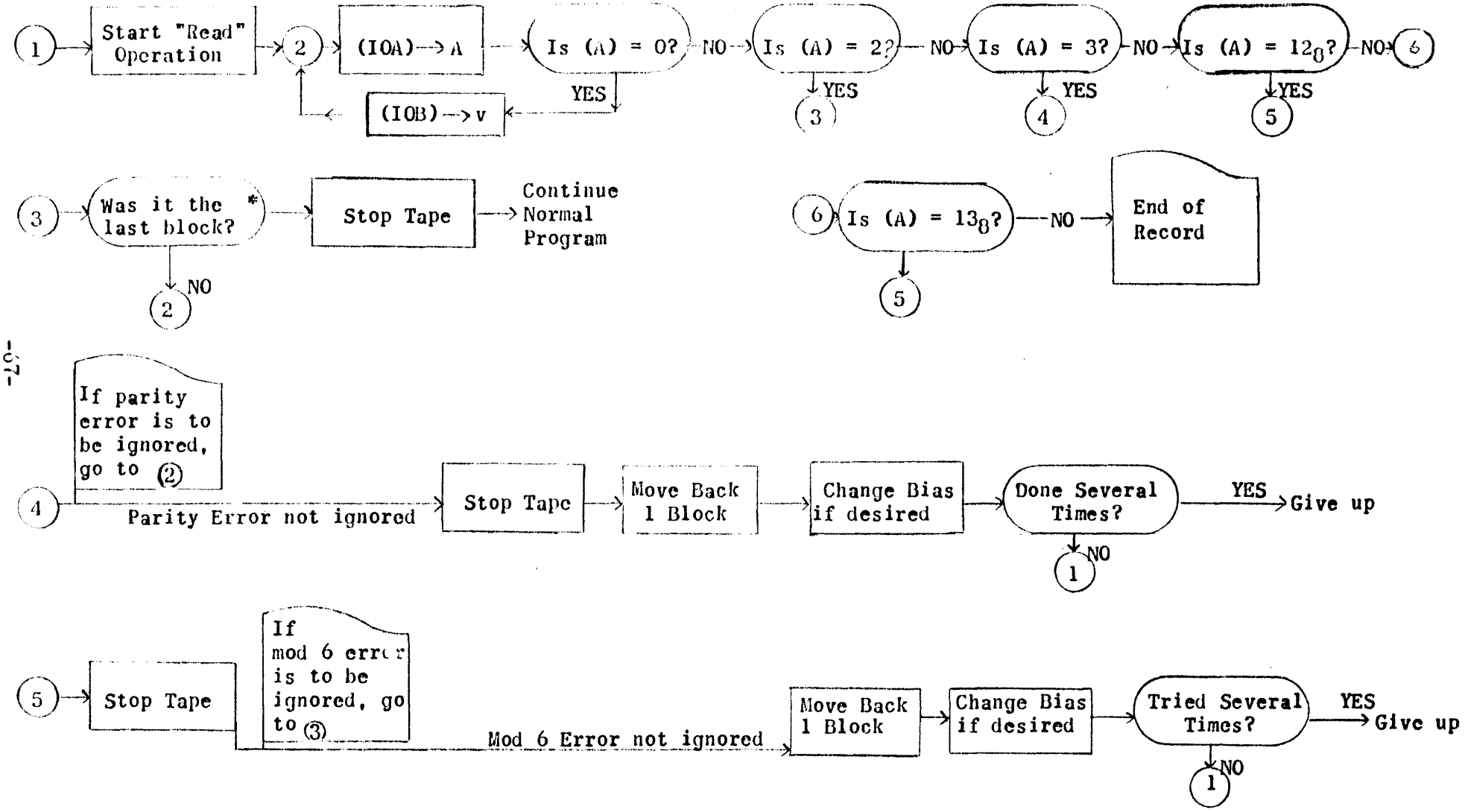
Thus there are 3 different possibilities:

(IOA) = 0 : Word in IOB

(IOA) = 2 or 3 or 12_8 or 13_8: No Word in IOB, but End of block reached without or with error(s).

(IOA) = 4 : End of Record, Tape stopped automatically.

Flow Diagram for "Read Forward" Operation
in Variable Block Length Mode



* If reading of n blocks is desired

Notice: This flow diagram represents a logical solution only.
The necessary housekeeping instructions are not shown here.

D) Stop Tape Operation

Since the only automatic stop of tape movement occurs after the detection of an end of record, i.e. approximately 4" of blank space, * the programmer has to provide an EF -v instruction, which stops tape movement after a "Write" or "Read" operation. Here (v) has to contain the following bits:

Select Magnetic Tape

Stop Code

After a "Write" operation the EF "Stop" instruction produces the inter-block space.

Since the EF "Stop" instruction also clears TR no mis-assembly of the words contained in the next block will occur after a mod 6 error.

E) Move Forward (or Backward)

This operation is initiated by an EF -v instruction, where (v) has to contain the following bits:

Select Magnetic Tape
Variable Block Length/Continuous Data Input
Move Forward (or Backward)
Uniservo Number
Number of Blocks to be moved.

The number of blocks specified in (v) is sent to BK. An end of block detection is used to decrease (BK) by 1. If (BK) = 0 moving is stopped automatically.

No parity or mod 6 check is made during a "Move" operation. There is also no end of block or end of record indication in IOA. Therefore (IOA) must not be read during or after a "Move" operation.

F) Rewind, Rewind with Interlock, Change Bias Operations

The Variable Block Length/Continuous Data Input bits have no significance for these operations. They need, therefore, not be included in (v) of the EF -v instructions which select these operations.

G) Selection of Variable or Continuous Input Mode

Since the bits IOB_{20} $IOB_{19} = 11_2$ cause an operation in either Variable or Continuous Input mode, the selection of one of these two modes is discussed in the following paragraphs.

* Except "Move" and "Rewind"

1) A preceding computer Master Clear determines that Variable Block Length mode is chosen by the Variable/Continuous bits

2) A preceding computer Master Clear followed by an EF "Change Mode" instruction determines that Continuous Data Input is chosen by the Variable/Continuous bits.

3) If Continuous Data Input was selected another EF "Change Mode" instruction executed later switches the computer back to Variable Block Length mode, etc.

Notice: Any EF -v instruction for magnetic tape which does not specify Variable/Continuous mode, automatically causes an operation in Fixed Block Length.

H) Sample Programs (Variable Block Length Format)

1) Write 20_{10} words into one block on Uniservo 5. (Assume tape is rewound or writing starts where previous writing operation had been stopped.) Words to be taken from 15000, 15001, etc.

```
  a  EF  00000   b          b   02  00066  50000
a+1  RP  10024   a+3        b+1 02  00600  00000
a+2  EW  10000   15000
a+3  EF  00000   b+1
```

2) Read forward two blocks from Uniservo 7 into 20000, 20001, etc. After parity error re-read the block one more time without changing the bias. If again unsuccessful, stop with MS j = 3.

If a "mod 6" error occurs stop computation with a MS j = 1. In case of a "parity and mod 6" error, step with a MS j = 2.

```
a-1  EF  00000   b
 a   TV  a+3     b+6        b   02  00062  70000
a+1  ER  00000   32000      b+1 00  00000  00001
a+2  EJ  a+6     a+3        b+2 00  00000  00002
a+3  ER  10000   [20000]     b+3 00  00000  00003
a+4  RA  a+3     b+1        b+4 00  00000  00012
a+5  MJ  00000   a+1        b+5 00  00000  00001
a+6  EJ  b+2     a+12       b+6 00  00000  20000
a+7  EF  00000   b+7        b+7 02  00600  00000
a+10 EJ  b+3     c          b+10 02 00074  70001
a+11 MJ  00000   d          b+11 00 00000  00001
a+12 TP  b+1     b+11
a+13 IJ  b+5     a          d   EJ  b+4    d+2
a+14 EF  00000   b+7        d+1 MS  20000  ----
a+15 NI                     d+2 MS  10000  ----

 c   EF  00000   b+10
c+1  TV  b+6     a+3
c+2  IJ  b+11    a-1
c+3  MS  30000   ----
```

J) Continuous Data Input

This mode of operation which is available automatically, if the computer is equipped with Variable Block Length Format, represents a special input method. A discussion of it is omitted here. However, it is pointed out that the data representation on tape is completely different from Fixed or Variable Format. One line on tape consists of 4 data bits, 2 code bits, one parity, and one sprocket bit. 3 lines are called a Data Entry Word, etc.

Detailed information may be found in the Manuscript Copy of Section on Magnetic Tapes, Programming Manual, Oct. 57.

III. Tape Format Required by Off-line High Speed Printer

In order to prepare a magnetic tape which is to be read by the off-line High Speed Printer the tape has to be prepared in the following format: *

Density: 128 lines per inch
Block Space: 1.2"
Blockette Space: 1.2"

Furthermore the High Speed Printer requires a special code for each number, letter, etc. This is the so called "Univac Excess Three" code. Each line of the tape results in the printing of one character, i.e. one 36-bit computer word results in printing of 6 characters. One Blockette gives us 120 characters. These are printed on one line of the High Speed Printer paper. This unit is capable of printing up to 600_{10} of such lines per minute.

The "Univac Excess Three" Code and the conversion from binary to excess-three and visa versa is omitted here, since it represents a special programming situation which can be solved easily by any person who knows how to program the 1103A or 1105.

* as of May, 58

VI) The 1105 Magnetic Tape and Buffer System

A) General Introduction to the 1105 Buffer System

As explained under "1103A Magnetic Tape System, 1105 Bypass Mode Operations" the communication between computer and Uniservos is made word by word using the buffer-register IOB. During transfers of data the computer cannot be used for other operations except during the time which is available between transfers of two words. This time, however, is very short, and a significant part of calculations cannot be performed in many kinds of applications.

Because of this fact a "Buffer System" has been added to the computer which will allow computer operations to occur during a transfer from Buffer to Tape Units. These operations, Computer to Buffers and Buffers to Uniservos, are explained in the following paragraphs. However, it is pointed out that the representation of data on tape, the selection of operations of Uniservos, and the physical structure of the 1105 Magnetic Tape System are identical to those used with the 1103A computer. The only difference is that the 1105 computer possesses two Magnetic Tape Control Units which are independent from each other. But each of these is built exactly as explained for the 1103A.* 1105 programmers may, therefore, carefully study chapter f) before continuing reading at B) chapterVI).

As already said: A data transfer from computer to Uniservos using the Buffers is accomplished in two steps: at first all data (up to 120_{10} computer words) are placed into a buffer memory, and then the contents of this buffer is written on a selected Uniservo. The transfer computer to Buffer is equivalent with a core-to-core transfer and very fast. The transfer Buffer to Uniservo is initiated by one command and then performed independently from the computer. Thus, the time needed for writing up to 120_{10} words on tape is completely available for other computer operations.

The same situations occur, if a transfer tape $\rightarrow$computer is made. All these facts are discussed in the following paragraphs. But to say it again: read them only if you understand the structure, processes and operations of the Magnetic Tape System as explained under f), Input-Output Section.

B) Physical Structure of the Buffer System

The 1105 Buffer System comprises two Buffers.
Each Buffer contains

- a core memory of 120_{10} registers
- a word counter
- an Input-Output Transfer Register IOT

and communicates directly with one Magnetic Tape Control Unit (TCU).

* as far as programming is concerned.

Therefore we possess 2 Buffers, 2 Tape Control Units (one for each Buffer), and 20_{10} Uniservos (10_{10} Uniservos for each TCU).*
In order to be able to distinguish between buffers, tape control units, and Uniservos the following numbering is applied:

Buffer #1, TCU1, Uniservos 1 thru 10_{10}

Buffer #2, TCU2, Uniservos 1 thru 10_{10}

Keep in mind: Buffer 1 can communicate with TCU 1 only, never with TCU 2. In the same way, Buffer 2 can communicate with TCU 2 only, never with TCU 1. Since the Uniservos have numbers from 1 thru 10, we have always to specify whether we mean e.g. Uniservo 3 connected with TCU 1, or Uniservo 3 connected with TCU 2.
The IOT-register is a 36-bit register. Each word transferred to or from the buffer has to go via IOT.

The Buffer Word Counter is increased by one, if a word enters the buffer, and decreased by one if a word leaves the buffer. Its use during machine operations and programming situations will be explained later.

C) Information Flow via Buffers

1) Computer ⟷ Buffer

Assume 120_{10} words are to be sent to a buffer, say buffer 1. In order to accomplish that we have to make sure that the buffer is ready to receive data (this is made by instructions explained later). If it is ready the following flow of data occurs:

One word has to be sent to IOB. This is done by an EW 10000 v instruction. Since the buffer is "informed" that we send words to IOB, it picks the word up from IOB, puts it into IOT, and finally places it into one buffer memory register increasing its Word Counter by "1". In order to fill the buffer completely 120_{10} words have to be sent to it, i.e. 120_{10} EW's must be executed to fill the buffer. As you see: this is the amount of words needed to write one block on tape in Fixed Block Length.

Assume buffer 1 is filled completely, and all words have to be placed into the computer. At first we have to "tell" the buffer to send the words to the computer (by an instruction explained later). The buffer then sends one word at a time to IOT and from there to IOB. The computer has to pick up each word from IOB by an ER 10000 v instruction.

As the above mentioned cases indicate: the computer program controls only the transfer from computer to IOB and from IOB to computer. The other transfer, buffer ⟷ IOB, occurs automatically, whenever the buffer "realizes" that either IOB contains a word which can be picked up (IOB → Buffer), or IOB is empty and the next word can be placed into it (Buffer → IOB), provided the proper selection of the buffer operation has been made.

* in the future, up to 12_{10} Uniservos per TCU!

2) Buffer ↔ Tape

Assume buffer 1 is filled, i.e. contains 120_{10} words. We intend to write these words on tape in Fixed Block Length. In order to do so we have to give a command which initiates writing. This command, however, has to specify: TCU 1, Uniservo No, Write, density, block and blockette spaces, and stop bits, if no free run is desired. Compare this with an EF -v instruction for Bypass Mode. There we have to specify the same information to write one block on tape. The only exception is the TCU No. instead of the former "Master Bit". As you probably guess: the instruction which initiates writing of the contents of a buffer on tape, is an EF "Write" instruction, where (v) contains the same bit selections as explaine under Bypass Mode, except "Master Bit". This "Master Bit" is now replaced by two bits, one for TCU 1 and another bit for TCU 2.

If we give this command: Buffer 1 write on Uniservo ....., the buffer begins the information transfer to the tape and continues to do so, until the whole content of the buffer is written on tape. This means: one command initiates the writing of up to 120_{10} words on tape. This writing process is automatically performed by the buffer and need not (and cannot) be controlled by the computer program. Notice that the computer, therefore, is free for other operations and can use the whole time needed for the writing process. The information flow is: buffer memory → IOT → TR → tape. Keep in mind: IOB is not used during a transfer buffer → tape.

Now we want to examine the transfer tape → buffer. This transfer is initiated by an EF "Read" instruction where (v) has to specify TCU No., Read (forward or backward), Uniservo No, and stop bits, if necessary. If e.g. TCU 2 was selected, then word after word enters buffer 2, until one block (up to 120_{10} words) has been transferred to it. IOB Is not used during this transfer, since one word goes from tape → TR → IOT → buffer memory. The operation is, as the buffer → tape transfer, initiated by one command, and from now on it is performed automatically, i.e. the computer is free for other operations.

D) Buffer States

1) "Load" and "Unload"

As you saw in the preceding paragraphs: each buffer possesses an IOT-regist Buffer memory communicates with this register directly.

Let us examine the case that a buffer has to receive data, say from tape, As already explained one instruction (an EF "Read" instruction) initiates the transfer of one block to the buffer. Word after word goes from TR to IOT and has to enter the buffer memory automatically. This can be done only if the buffer is really ready to receive data from IOT.

The other case is that the buffer has to transfer words, say to the computer. This means that one word after the other has to be sent from buffer memory to IOT (and then to IOB).

Again this automatic sequence (buffer memory →IOT) can take place only if the buffer is ready to do so.

The result of these considerations is that a buffer has to be in one of two states:

in the "load" state, if it is to receive data regardless of whether the data comes from computer or tape

in the "unload" state, if it is to transfer data to computer or tape.

The question is: how is the switching from one state to the other accomplished? Upon starting computer operations the programmer expects that the buffers are completely empty and ready to receive data, either from tape or from the computer. This is the case.

A computer Master Clear clears both buffer memories, sets both Word Counters to "0", and sets both buffers to the "load" state.

From now on switching to "unload" and "load" depends upon the Word Counter or the End of Block Signal. It may be said now, that buffer operations in Fixed and Variable Block Length Mode are possible. Let us discuss switching for both.

Fixed Block Length:

Here one block consists of 120_{10} computer words, i.e. is equal to the capacity of one buffer. As long as this mode of operation is selected switching of the buffer from "load" to "unload" and visa versa is fully automatic. It means:

After computer → buffer transfer:

If the Word Counter is = 120_{10}, the buffer is switched to "unload".

After tape → buffer transfer:

If the "End of Block" signal is generated, the buffer is switched to "unload".

After buffer → computer and buffer → tape transfers:

If the Word Counter is = 0, the buffer is switched to "load".

Variable Block Length:

Because of the capacity of each buffer, a block in Variable Format may not exceed 120_{10} words, i.e. it may contain 1,2,3,.....,120 words, but not more. Variable Blocks which contain more than 120_{10} words have to be read in Bypass Mode.

The switching of the buffer from "load" to "unload" and visa versa is fully automatic after the following data transfers:

buffer → computer
tape ⟶ buffer
buffer ⟶ tape.

The transfer computer → buffer, however, does not switch the buffer to "unload" automatically. This switching of the buffer has to be accomplished by an EF "End Transfer" command as explained later.

The following may be pointed out here:
The EF "End Transfer" has to be given, if less than 120_{10} words are sent to the buffer (from the computer). If exactly 120_{10} words are sent to the buffer and are to be written on tape in Variable Format, we have to distinguish between the following two situations:

Either the EF "Write on Tape in Variable Format" has been given before the transfer computer → buffer was made: then an EF "End Transfer" has to be made (this will be the normal case, namely Free Run).

Or the EF "Write on Tape in Variable Format" is given after the transfer of 120_{10} words to the buffer: then an EF "End Transfer" need not be given. However, if it is executed the program proceeds normally, This means for a programmer:

If you wish to write in Variable Format always give an EF "End Transfer" after the computer → buffer transfer.

2) Buffer "Activity"

As you have seen earlier: the transfers buffer → tape and tape → buffer are made independently from the computer as soon as a command initiated them. The computer program will naturally use the time which elapses during such a transfer in order to perform other operations. After a certain time has been used in this way the computer might try to refer to a buffer which just communicated (or still communicates) with a tape unit.

In order to handle such situations in a proper way the following has been made:

A buffer is "active", if it
either receives data from tape,
or transmits data to a Uniservo,
or if one of its Uniservos performs a "Move" operation.

A buffer is "inactive", if its memory
either is not receiving data from tape and the switch "load" to "unload" has been completed,

or is not transmitting data to a Uniservo and the switch from "unload" to "load" has been completed,
or if none of its Uniservos is performing a "Move" operation.

A detection of buffer activity is made by MJ jv instructions with j = 4 or 5, as explained below.

E) Programming for Write Operations using Buffers

A "Write" operation using a buffer involves the following program steps in the sequence given:

α) Fixed Block Length

1) Test for Buffer Activity

```
MJ 40000  v     If buffer 1 is active, jump to v.
                If buffer 1 is inactive, take N.I.

MJ 50000  v     Same as above for buffer 2.
```

These instructions have to be applied, whenever the programmer suspects that a previous buffer↔tape data transfer or a Move Tape operation might not be finished at the time he tries to initiate another operation involving the same buffer and/or TCU.

2) EF "Write Buffer" instruction

This instruction prepares the buffer for the computer→buffer data transfer. The (v) of the EF -instruction contains bits for

"Buffer No"
"Write Buffer"

3) Data Transfer from Computer to Buffer

120_{10} words have to be sent to the buffer, i.e. we have to execute 120_{10} EW's. The fastest way to do it is by using a RP -command, i.e.

```
RP 10170  w
EW 10000  v
```

4) Buffer→Tape Transfer

In order to write 120_{10} words of a buffer on tape an EF "Write Tape" instruction has to be given, where (v) contains bits for

TCU No. (corresponding to Buffer No.)
Write, Density
Uniservo No.
Spaces
Stop (if no Free Run desired)

This instruction is identical in function and coding to the EF "Write Tape" instruction used for the 1103A computer with the exception: the old "Master Bit" is now replaced by a TCU No.

Notice: The EF "Write Tape" instruction sets the buffer to "active". *

β) Variable Block Length

In order to write one block on tape in Variable Format apply the following steps:

1) as in Fixed Block Length
2) as in Fixed Block Length

* provided the buffer is in the "unload" state

3) the data transfer to a buffer involves the transfer of less than 120_{10} words. But as in Fixed Block Length the proper number of words can be transmitted by the sequence:

```
RP    jn     w
EW    10000  v
```

where $n \leq 120_{10}$.

3a) EF "End Transfer"

(v) of this EF contains bits for

"Buffer No"
"End Transfer"

This command switches the buffer to "unload".

4) as in Fixed Block Length with the addition that bits for Variable Format have to be included in (v). If writing of one block is attempted only, then "Stop" bits can be included in (v). (This is different from the 1103A and 1105 Bypass operations in Variable Block Length Mode.)

F) Programming for Read Operations using Buffers

The program steps which have to be executed in order to read information from tape into the computer via a buffer are the following:

α) Fixed Block Length

1) EF "Read Tape" instruction

This information which initiates the transfer of one (or more) blocks from tape into a buffer makes use of a (v) with the selection bits for

TCU No
Read forward (or backward)
Uniservo No
Stop (if no Free Run desired)

Thus, if TCU 1 is selected, a block goes into buffer 1; if TCU 2 is selected, a block is sent to buffer 2.

Notice: The EF "Read Tape" instruction sets the buffer to "active". * The transfer of one block from tape to buffer is now an automatic sequence. The time necessary to perform this transfer will, in most cases, be used by the programmer. He, therefore, has to test for buffer activity, when he wants to continue with an operation involving the buffer which was used for the tape → buffer transfer.

2) Read Error

If the buffer was found to be inactive the programmer may test whether or not an error occurred during the reading process. This is done with an MJ j = 6 or 7:

```
MJ  60000  v  If an error occurred during reading into buffer 1, take
                 N.I. If there was no error, jump to v.
MJ  70000  v  Equivalent to above command, but refers to buffer 2.
```

The Read Error can be a parity error, sprocket error, or end of record. If either one occurs the tape movement is automatically stopped in the following block space. It does not result in a computer fault.

*provided the buffer is in the "load" state

In order to determine the nature of the error the following method has to be applied:

Read the corresponding Buffer Word Counter (see page 79) and test it.

If (BWK) = 120, there was a parity error

If (BWK) ≠ 120 and ≠ 0, there was a sprocket error *

If (BWK) = 0, there was an end of record, i.e. no words are in the buffer.

Keep in mind: In normal mode (buffer mode) IOA is not used for error indications.

3) EF "Read Buffer" instruction **

This command initiates the transfer buffer → computer. (v) of this EF -v has to contain bits for

"Buffer No"
"Read Buffer"

4) (IOB) → Computer Transfer

Each word sent to IOB by the buffer has to be picked up by an ER 10000 v.

If all 120_{10} words are to be read, the fastest way of doing it is

RP 10170 w
ER 10000 v

However, keep in mind that you need not pick up all words! If e.g. reading of the first word is intended only, then one ER is sufficient. But the second word is sent to IOB from the buffer! There are now two possibilities: either the first word is only operated on and, after some time, the reading of the buffer is continued. In this case, the programmer has to give the remaining 119_{10} ER's.

The second possibility is that after reading the first word another EF-instruction is given in order to select a different input/output equipment. In this case (IOB) = second word is erased, i.e. it is lost. It may be said that an EF-instruction may not select a read operation with the second buffer or any other equipment.

4a) In any case: if only part of the content of a buffer is read into the computer and the remaining words are not wanted the switch from "unload" to "load" and a clearing of the buffer memory is accomplished by an

EF "Clear Buffer" instruction:

(v) contains bits for

"Buffer No"

"Clear Buffer"

The instruction accomplishes:

Set Buffer Word Counter to "0"

Clear Buffer Memory

Switch Buffer from "unload" to "load"

* Notice: If more than 720_{10} lines per block are detected, each additional line advances the Buffer Word Counter by one, until it reads $127_{10} = 177_8$

** See remarks on bottom page 84

β) Variable Block Length

1) as in Fixed Block Length; only include bits for Variable Block Length.

2) Test for Read Error like in Fixed Mode. In Variable Mode there are, however, three types of Read Errors:

parity error, sprocket ("mod 6") error, end of record.

The End of Record is detected by the machine like in Bypass Mode, i.e. after approximately 4" of blank space have been passed following the end of the last block. This causes the tape to stop automatically. It can be found by reading the Buffer Word Counter and testing it. If it is = 0, there was an end of record.

If the BWK ≠ 0, the nature of the read error (parity or sprocket error) can be determined only by re-reading the block in bypass mode. The tape movement, however, stopped automatically in the following block space.

Notice: IOA is not used for error indications.

2a) EF "Read Word Counter"

Since the length of the block read into the buffer from the tape is unknown to the programmer, but the exact number of ER's has to be given for the transfer buffer → computer, an instruction is provided which picks up the content of a Buffer Word Counter. This is the EF "Read Word Counter". (v) of this instruction has to contain:

"Buffer No"
"Read Word Counter" bits

The sequence of steps set up by this command is:

Content of Buffer Word Counter $\rightarrow IOB_u$

Therefore, an ER 10000 v can take (IOB) and place it into the computer.

It is obvious that this number represents the exact number of ER's which have to be executed. It is actually our "n" for a RP-command, and we have only to put it into this place. This, by the way, is the reason why the Buffer Word Counter is sent to the u-part of IOB.

3) as in Fixed Block Length

4) as in Fixed Block Length. The "n" of the RP-command has been found out by the method explained under 2a) EF "Read Word Counter". If all words which entered the buffer from tape are placed into the computer the switch from "unload" to "load" is made automatically.

4a) If only part of the content of a buffer is read into computer memory, the switch from "unload" to "load" has to be made by the EF "Clear Buffer" instruction as explained for Fixed Block Length.

G) Stop after Read or Write Operations

If the tape is to be written or read a block at a time, the stop bits may be included in the EF "Write Tape" or EF "Read Tape" instruction (Fixed or Variable Format).

When reading or writing in free run mode, tape movement must be stopped by an EF "Stop Tape" instruction. (v) of this command has to contain

"TCU No"
"Stop bits"

When has this command to be executed?

In order to terminate a "Read" operation, the EF "Stop Tape" must be executed either before or after the instructions which complete the transfer of the last block from buffer to computer.

In order to terminate a "Write" operation, the EF "Stop Tape" must be executed after the last block has been written on tape, i.e. after the last transfer buffer → tape.

A rule which covers both, reading and writing, is:
In order to terminate a Free Run operation, execute the EF "Stop Tape" instruction not earlier than at the time, when the last transfer tape → buffer or buffer → tape is finished.

It is not possible in Buffer Mode to execute an EF "Stop Tape" instruction too late. This is explained under "Automatic Tape Controller".

H) Selection of "Bypass Buffer Mode"

Tape operations can be performed without using a buffer, i.e. it is possible to "bypass" or disregard either of the buffers. If this is intended, an EF "Bypass Buffer" instruction has to be executed, where (v) contains bits for

"Buffer No"
"Bypass Buffer"

Assume that this instruction is given in order to bypass Buffer 1. As a result all following tape operations referring to TCU 1 will be made in exactly the same way as described under Vf) "The 1103A Magnetic Tape System, 1105 Bypass Mode Operations". This means that the flow of information is

computer → IOB → TR (of TCU 1) → tape
or tape → TR (of TCU 1) → IOB → computer.

However, tape operations referring to TCU 2 still employ buffer 2. It is naturally possible to bypass both buffers simultaneously. But keep in mind that in this case reading and/or writing cannot occur simultaneously, as it is possible in normal (buffer) mode.

In order to return to the normal mode, (i.e. to use buffer 1 again in above example), one of the following instructions has to be executed:

EF "Clear Buffer"
EF "End Transfer"
EF "Read Word Counter"
EF "Write Buffer"
EF "Read Buffer"

Which of these instructions is used depends upon the current state of the buffer, i.e. upon the programming situation given by the problem to be solved.

The "Bypass" Mode has to be selected:

in order to read or write blocks in Variable Format consisting of more than 12^{10} words;

in order to distinguish between parity and sprocket error in Variable Format.

It may be selected for any other tape operation.

J) Automatic Tape Controller ATC

Let us assume we perform a Free Run operation, say Writing, in Bypass Mode. Block after block is written on tape. The tape does not stop, until we give an EF "Stop Tape" instruction. This forces the programmer to carefully calculate the timing for his problem. Between the last word of one block and the first word of the next block the programmer has a certain time which he may not exceed. There is also a limit for the time which may pass, until an EF "Stop Tape" instruction has to be given after writing the last word on tape.

During Free Run operation which employs a Buffer the situation is completely different.

Reading in Free Run via a buffer:

If one block of data entered the buffer and is not picked up by the computer tape movement is stopped temporarily. This stop occurs after a nominal 3 m sec. time lapse following the detection of the end of the block, if during this time an EF "Read Buffer" has not been given.

Tape movement is automatically started again, when the EF "Read Buffer" instruction is executed.

If the block transferred from tape to buffer was the last block and the EF "Stop Tape" instruction has not been executed within 3 m sec. following the end of block detection the automatic stop is also initiated. An EF "Stop Tape" instruction executed later will effect a permanent halt of tape movement.

Writing in Free Run via a buffer:

If one block of data has been written on tape and the transfer of the next block from computer to buffer is not initiated within a

nominal 3 m sec. time lapse following writing of the last word on tape, an automatic stop of tape movement is made. Tape movement is started automatically, when the transfer computer → buffer is initiated by an EF "Write Buffer" instruction.

If the block written on tape was the last block and the automatic stop occurred an EF "Stop Tape" will cause a permanent halt of tape movement.

The following example will illustrate the situation completely. Assume you write in Free Run via buffer 1. You send the first block to the buffer, and writing on tape is started. This takes 36 m sec., if a density of 200 lines per inch is selected and 120_{10} words are written. The program naturally uses this time for other operations. Assume further, that 50 m sec. elapse before you initiate the next transfer computer → buffer. What happened? After 36 m sec. the last word of the first block was written on tape. Now the block space is inserted. If 1.2" are selected, it takes 12 m sec. to pass this space. However, after 36 + 3 = 39 m sec. the Tape Control realizes that the next block is not being transferred from computer to buffer 1. It, therefore, initiates the automatic stop of the tape. Just imagine for a moment this would not be so. In this case the tape would require the first word of the second block not later than 48 m sec. after the start of the writing of the first block. Since it was assumed that 50 m sec. pass this example would cause troubles for the programmer. Therefore, keep in mind: you can never be too late during Free Run operations. "Too early" is naturally a different situation. Properly programmed MJ's (j = 4 or 5) prevent a too early reference to a buffer.

K) Some Timing *

a) Transfer Buffer ↔ Tape

The tape moves with a speed of 100" per sec. Therefore, the transfer of 120_{10} words with 200 lines per inch density requires 36 m sec.

b) Spaces

To pass a block space of 1.2" or 2.4" a time of 12 m sec. or 24 m sec., respectively, is required. Similar for Blockette Spaces.

c) Transfer Buffer ↔ Computer

Buffer → Computer: If repeated ER's are used one word is transferred every 16 μ sec. 120_{10} words are, therefore, placed into the computer within approximately 1.9 m sec.

Computer → Buffer: If repeated EW's are used one word is transferred every 20 μ sec., 120_{10} words within approx. 2.4 m sec.

For more timing refer to "1105 Prog. Manual, U 1513".

* See "Preliminary Programming Manual for the 1105 Computer", US 108, page 20 and 21

L) Faults

All faults occurring during Buffer operations are "B" Faults. The following classes of faults are possible:

"Select Fault":

This fault occurs in either one of the following two cases:

a) Buffer is in the "load" state and inactive, but an EF "Read Buffer" is executed.

b) Buffer is in the "unload" state and inactive, but an EF "Write Buffer" is executed.

"IO-Fault":

This fault occurs, if a reading of tape is attempted, but the buffer is already filled.

Examples: Reading of a block of more than 120_{10} words.
EF "Read Tape" and Buffer is in "unload" state.

The above fault is generated at the time when the second word from tape tries to enter IOT, while the first word is still there.
An "Address Fault" (see below) will automatically select an "IO-Fault".

"Address-Fault":

As a checking device a buffer counts the words as they enter the buffer. If a word has not been stored in its proper buffer memory location the "Address Fault" is generated.

"Transfer Fault":

Each "B-Fault" generated during computer and/or buffer operations is sent to the Tape System where it generates a "Transfer Fault" which will stop tape movement in the next block space.

M) Sample Programs (for Bit Assignments see Appendix)

1) Write one block of numbers stored at 20000, 20001, ... on Uniservo 5, TCU 1 using high density, 1.2" Blockette Space, 1.2" Block Space, Fixed Block Length. At first test for Buffer Activity. If Buffer 1 is active wait until it is available. (It is assumed that the operation of the buffer is a "Write on Tape" operation. If buffer is inactive you know that it must be in the "load" state.)

```
  a   MJ  40000    a      Wait until buffer 1 is inactive.
a+1   EF  00000    b      Select "Write into Buffer 1"
a+2   RP  10170   a+4\    Send 120(10) words
a+3   EW  10000  20000/     to Buffer 1
a+4   EF  00000   b+1      Write on tape
a+5   NI

  b   00  10000  00400    Constant
b+1   01  00646  50000    Constant
```

Notice how we accomplish the waiting for a previous buffer → tape operation. (a) is an MJ 40000 a. This means: as long as Buffer 1 is active, we jump to a, i.e., to the same command. In the moment the buffer is inactive, no such jump will occur, but we now continue in sequence and perform the buffer and tape operations.

2) Read forward one block from Uniservo 3, TCU 2, into 10000, 10001, ... Fixed Block Length. Test for Buffer Activity first and wait, if not available. (It is assumed that a previous operation leaves the buffer in the "load" state.)

```
   a   MJ   50000    a       Wait, until Buffer 2 is inactive
 a+1   EF   00000    b       Read Tape into Buffer 2
 a+2 \
  :   } available for computation;
a+n-1/  at least about 46 m sec.
                    may be used
 a+n   MJ   50000   a+n      Wait, until Buffer 2 is inactive
a+n+1  MJ   70000  a+n+3     Test for Read Error
a+n+2  RJ     u      v       Return Jump to Error Routine
a+n+3  EF   00000  b+1 \     Select "Read from Buffer 2"
a+n+4  RP   10170 a+n+6 }    Buffer 2 → Computer
a+n+5  ER   10000  10000/

   b   02   00602  30000     Constant
 b+1   00   20000  00200     Constant
```

3) Write one block of data consisting of 19_{10} words on Uniservo 1, TCU 2, in Variable Block Length. Assume Buffer 2 is ready for the operation.

```
   a   EF   00000     b \     Select "Write into Buffer 2"
 a+1   RP   10023    a+3 }    Send words to Buffer 2
 a+2   EW   10000   05000/     from 05000, 05001, ......
 a+3   EF   00000    b+1      Switch Buffer 2 to "unload"
 a+4   EF   00000    b+2      Write on Tape

   b   00   20000   00400     Constant
 b+1   00   20000   01000        "
 b+2   02   00666   10000        "
```

Notice the EF "End Transfer" at a+3 which accomplishes a switching of Buffer 2 from "load" to "unload".

Remarks: During a "Read Backwards" operation the word which was recorded as last word is read as first word into the buffer, the word which was recorded as first word (word #1) is read as last word. However, when reading this block from buffer → computer, word #1 is picked up first etc. This means that a "Read Backward" operation essentially transfers all words of a block to the computer in exactly the same order as they were written on tape, if a buffer is used!

VII) The 1103A and 1105 Interrupt Feature

A) General Explanation

The Program Interrupt Feature of the 1103A/1105 permits an external equipment to interrupt the computer program automatically, when the external equipment is ready to communicate with the computer. Proper programming can interrupt the computer program upon occurrence of the following situations:

Data sent to IOB or IOA by an ext. equip. have to be picked up by the computer.

Data have to be sent to IOB or IOA by the computer, because the external equipment is ready to receive them.

Either (or both) of the Buffers of the 1105 is ready to communicate with the computer.

There is also the possibility to interrupt the computer by manually pushing two buttons on the computer console.

Both the manual and the program controlled interruption generate a so-called "Interrupt Signal" which results in a modification of the normal sequence of steps the computer follows. This modification affects Main Pulse 6 and is explained in the following paragraph.

B) Modification of the Computer Program by an Interrupt Signal and Programming Consequences

1) When does the Interrupt become effective?

As already said: the Interrupt is nothing else but a signal (pulse) Assume data are read into the computer, say from cards. After one row is placed into IOB and IOA the Card Unit emits a signal thus informing the computer that data are available in IOB and IOA. This signal is the Interrupt Signal. What is the situation now? The computer is operating, i.e. it is either picking up a command from storage (MP 6 and 7) or executing a command (MP 0 thru 5). The Interrupt Signal arrives at any of these Main Pulses, and the question is: will it be effective at once, or will it have to wait?

The answer to this question is:

> The Interrupt Signal has to wait, until the computer executed a command completely or until a Repeat Sequence is terminated completely. *

To make this absolutely clear: we have to distinguish between two situations:

Either the interruption is attempted at a time a Repeat Sequence is being executed. In this case the Interrupt Signal becomes effective, after the command at address F_1 has been completely

* To be precise: until the "Hold Repeat FF is 0" (refer to "Repeat Command")

executed or a jump has been performed during repetition of an EJ or TJ.

Or the interruption is attempted and no Repeat Sequence is being executed. In this case the Interrupt Signal becomes effective after the execution of the command which the computer is operating on.

The following program might illustrate this:

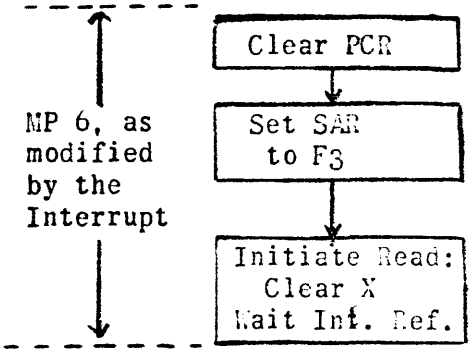
```
   a   TP    b     b+1
 a+1   RA    a      b
 a+2   RP  10100  a+4
 a+3   TP   b+2     c
 a+4   NI
```

First case: The Interrupt Signal arrives during the execution of the TP at a. In this case it has to wait, until this command is executed completely. Then the signal is effective and results in a modification of steps as explained later.

Second case: The Interrupt Signal arrives during the execution of the Repeat Sequence, say after the TP at a+3 has been executed 5 times. Now the interruption is delayed not only until the TP at a+3 is executed 100_8 times, but also, until the command at F_1 is executed completely. (See under "Repeat Command" for details regarding the execution of an RP jn w)

2) Modification of MP 6 by the Interrupt Signal

Now we are ready for the discussion of the modifications performed by the Interrupt Signal. At the moment this signal becomes effective the computer is actually ready to continue with MP 6, i.e. the computer tries to pick up the next command from a storage location given by PAK. (Refer to "Program Address Counter", page 2 of this manual. There is the discussion of the computer operations performed during MP 6 and MP 7). The Interrupt modifies this MP 6 in the following way:



Essentially SAR is set to address F_3 = 00002, and the word stored at this location is placed into X.
This is made without changing or using the address currently held in PAK.

Example: Assume the Interrupt is generated during the execution of (06235) = TP A Q. At this time PAK = 06236. At the end of the execution of the TP, the Interrupt becomes effective. This means that the next command to be executed is extracted from 00002. PAK is left unchanged and reads 06236 at the beginning of the execution of (00002).

3) Programming Consequences

The above example shows that the steps the computer has to follow after picking up (00002) entirely depend upon the command stored there. There are two possibilities:

α) (00002) is not a jump command

In this case (00002) is executed and the computer proceeds with extracting the next command from 06236, since PAK has not been changed.

β) (00002) is a jump command

Here PAK is erased by the new address to which we jump, and the computer proceeds there. On the other hand it is obvious that we want to come back to our main program sooner or later, and especially to the point, where we left it (06236 in example). We also want to accomplish by the interruption a communication with external equip., i.e. starting at 00002 there has to be a small subroutine which has to perform the communication. How can we do this? We have to save the address of our main program held in PAK, and must execute a jump, since the above mentioned subroutine hardly consists of (00002) only.

This can be done by one command, the RJ uv. Therefore:

If the Interrupt Feature is to be used, then

(00002) = RJ u v,

where v denotes the entrance of the subroutine, u denotes its exit.

Here you see the importance of a complete understanding of the RJ- command. Its execution is made in the way:

$PAK \longrightarrow U_v$

$v \longrightarrow PAK$

Following our example and assuming the subroutine to be executed after an Interrupt starts at 05000 and end at 05500, we have to write:

(00002) = RJ 05500 05000

with (05500) = MJ 00000 30000.

In this case $06236 \longrightarrow 05500_v$ so that now

(05500) = MJ 00000 06236.

Then $05000 \longrightarrow$ PAK, i.e. a jump to 05000 is made. After execution of the whole subroutine we come to 05500, which brings us back to our main program, namely 06236.

So far we have explained how to save the address of the main program to which we intend to return, and how to jump to the subroutine which has to be executed. However, you can see that this subroutine probably uses A and Q during the execution of its commands. On the other hand valuable numbers might be left in A and Q at the moment the main program

is left. We certainly cannot afford to lose these numbers. Moreover, before returning to the main program we have to restore A and Q such that they contain the same numbers as at the moment we left the main program. The result of these considerations is the following:

```
(00002) = RJ    u       v

  (v)   = TP    A       ws        (A_R) ⟶ ws
 (v+1)  = LT    00000   ws+1      (A_L) ⟶ ws+1
 (v+2)  = TP    Q       ws+2      (Q)   ⟶ ws+2
   :      :       :       :      }
   :      :       :       :      }  Program which actually
   :      :       :       :      }  performs the communication
   :      :       :       :      }  with ext. equip.
 (u-3)  = TP    ws+2    Q         Restore Q
 (u-2)  = SP    ws+1    00044     Restore A_L
 (u-1)  = SA    ws      00000     Restore A_R
  (u)   = MJ    00000   30000     Exit
```

The above program is a typical example for a subroutine which has to be executed after the generation of an Interrupt Signal. It stores $(A)_i$ and $(Q)_i$, performs its specific job, and restores (A) and (Q).

C) The Program Controlled Interrupt

1) The "Bull" Card Unit

The Interrupt Feature can be selected in connection with the on-line card unit. One bit ($v_7 = 1$) included in (v) of the EF-v which selects a card unit operation enables this equipment to emit the Interrupt Signal.

Reading of Cards:

When cards are read the Interrupt Signal is emitted at the time IOB and IOA contain data from one row, i.e. one row is ready to be picked up from IOB and IOA. This means that the subroutine to which the program jumps has to read IOB and IOA as explained under "Bull Card Unit."

Punching of Cards:

When cards are punched the Interrupt Signal is emitted at the time the card unit requires new data for one row in IOB and IOA, i.e. the subroutine has to send data for one row to IOA and IOB as explained earlier.

In either case it is important to understand that the interruption is made for each row of a card.

The usefulness of the Interrupt Feature in conjunction with the card unit is evident. It eliminates the task for the programmer to carefully calculate the time he uses for other computer operations between successive rows. The Interrupt Signal automatically "informs" the computer that either (IOB) and (IOA) have to be picked up (reading)

or data have to be sent to IOA and IOB (punching). This means that now these operations cannot be made too late. On the other hand it also means that the programmer need not initiate these operations too early and, therefore, "hang up" temporarily thus wasting valuable computer time.

It is, however, pointed out that the main program may not contain any Repeat Sequences which need more time for their executions than 1.5 m sec. (if Interrupt for punching is used) or 10 m sec. (if Interrupt for reading is used). These are the so called "receptive times" for the card unit, i.e. time which may elapse between the beginning of an input or output cycle and the execution of the first EW or ER.

2) The 1105 Buffer System

The Interrupt can be used in connection with the 1105 Buffer System. It is possible to select the Interrupt for either buffer or both buffers. This selection is made by including bit $v_{24} = 1$ into (v) of one of the following instructions: EF "Read Tape", EF "Write Tape", EF "Move Tape",

Reading from Tape via a Buffer:

The Interrupt Signal is generated at the time a whole block has been transferred from tape to buffer. Actually the interruption is made by the "end of block" signal, and at the same time the buffer is switched from "load" to "unload". The subroutine to which you jump because of the RJ uv at 00002 can, therefore, perform the transfer buffer →computer.

Writing on Tape via a Buffer:

In this case the Interrupt Signal is generated, after the content of the buffer has been written on tape and the buffer has been switched to the "load" state. The subroutine to which you jump because of the Interrupt can, therefore, perform the transfer of the next block from computer to buffer.

In both cases you see that the Interrupt only means: A previous tape ↔ buffer operation is completed. Therefore, the buffer is ready to communicate with the computer. However, the interruption does not mean that the computer has to perform a buffer↔computer transfer, since even in Free Run the Automatic Tape Controller would stop the tape, if the next reading or writing is not initiated in time. With other words, the Interrupt for the Buffer System merely prevents the computer from waiting for Buffer Inactivity. Thus it gives the programmer the possibility to make the most efficient use of the time needed for tape↔buffer transfers.

As long as the Interrupt is chosen for use with one buffer at a time, no MJ with j = 4 or 5 need to be executed after the jump to the subroutine, because such a jump means that the buffer is inactive. However, if the Interrupt is selected for both buffers simultaneously, these MJ's have

to be executed in order to detect which buffer emitted the Interrupt Signal. After the main program is interrupted once, all subsequent Interrupt Signals initiated within 0.5 m sec are ignored.

Moving the Tape:

The Interrupt may also be selected for a "Move n blocks" operation. In this case the main program is interrupted after the tape has been moved the specified number of blocks. (This does not apply for Bypass Mode Move operations!)

3) Other Equipments

The Interrupt may also be chosen for other equipments as e.g. the Ferranti Tape Reader and the On-Line High Speed Printer. A discussion of these operations is omitted. The existence of these possibilities is merely mentioned here.

VIII) The 1103A/1105 Floating Point System

A) Representation of Numbers

A decimal number N is represented in the form

$$N = \pm m \cdot 2^c$$

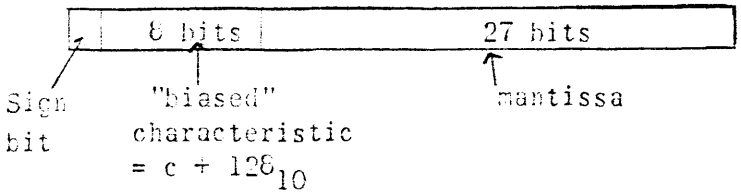
where $\frac{1}{2} \leq m < 1$.

With this restriction for m, there exists one and only one value for c.

m is called "mantissa",

c is called "characteristic".

Since both, mantissa and characteristic, are to be placed into one 36-bit register, the following format has been chosen:



α) Representation of positive Numbers (N > 0)

The sign bit of a number N > 0 is a "0".

The mantissa is placed into the rightmost 27_{10} bits of the register such that its decimal point is assumed to be between bit i_{27} and i_{26}, i.e. all 27 bits are fractional bits with the leftmost bit (i_{26}) being a "1".

As shown in the above figure the 8 bits reserved for the characteristic actually represent the so-called "biased" characteristic C, where $C = c + 128_{10}$.

The reason for this method is the following:
The 8 bits reserved for the characteristic can lie between 00 000 000 and 11 111 111 (binary). It is, however, obvious that a positive number can possess a negative characteristic, as e.g. $\frac{1}{4} = \frac{1}{2} \cdot 2^{-1}$, i.e. $c = -1$. This means that we have to be able to represent negative exponents for a positive number. If we assume that 00 000 000 represents a $c = 0$, we cannot represent a $c < 0$, unless we assign the leftmost of these 8 bits the function of a sign for the characteristic. This, however, would over-complicate the whole matter, since we then had to deal with two sign bits in one register.

Therefore, the following representation for the characteristic has been chosen:

$$8 \text{ bits} = 10\ 000\ 000_2 = 200_8 = 128_{10}$$

corresponds to $c = 0$.

Then:

$10\ 000\ 001 = 201_8$ represents $c = 1$

$10\ 000\ 010 = 202_8$ represents $c = 2$

$10\ 000\ 011 = 203_8$ represents $c = 3$

$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$

$11\ 111\ 111 = 377_8$ represents $c = 127_{10}$

$01\ 111\ 111 = 177_8$ represents $c = -1$

$01\ 111\ 110 = 176_8$ represents $c = -2$

$01\ 111\ 101 = 175_8$ represents $c = -3$

$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$

$00\ 000\ 000 = 0$ represents $c = -128_{10}$

Applying this method we are able to represent exponents c in the range

$$-128_{10} \leqq c \leqq +127_{10}$$

Thus all numbers $N > 0$ in the range

$$2^{-129} \leqq N < 2^{127}$$

can be represented using this "Floating Point" Format. (The range in decimal is roughly $10^{-38} \leqq N \leqq 10^{+38}$)

Examples:

$N = 3$:

Here, $3 = 3/4 \cdot 2^2$, i.e. $m = 3/4$, $c = 2$.

Since $3/4 = 6/8 = .6_8 = 110_2$, we have:

$3 \equiv$ 0 (Sign) 10 000 010 (C) 110 $\longleftrightarrow$ 0 (mantissa, 27 bits)

In octal we have: $3 \equiv 202\ 60 \longleftrightarrow 0$

$N = \frac{1}{4}$:

Since $\frac{1}{4} = \frac{1}{2} \cdot 2^{-1}$, we obtain $m = \frac{1}{2}$, $c = -1$.

Therefore, $\frac{1}{4} \equiv 0\ 01\ 111\ 111\ 10 \longleftrightarrow 0$ (binary)

or, $\frac{1}{4} \equiv 177\ 40 \longleftrightarrow 0$ (octal)

40 $\longleftrightarrow$ 0: mantissa.

Notice: $\frac{1}{2} = 4/8 = .4_8$

$N = 1/3$:

$1/3 = 2/3 \cdot 2^{-1}$, i.e. $m = 2/3$, $c = -1$

In this case the conversion of the decimal mantissa $m = 2/3$ into binary (or octal) is not made as simply as in the above examples, where the denominator always was 2^x ($x = 2$ for $N = 3$, $x = 1$ for $N = 1/4$). Therefore, we have to convert a decimal fraction to an octal fraction by multiplying

with 8:

$1/3 = .666\ 666\ 666\ ....$

$$5 \overline{)\ .333\ 328} \quad .666\ 666 \cdot 8$$

you see: the result is actually 5.333 333 ..... Therefore, we go on with

$$2 \overline{)\ .666\ 664} \quad .333\ 333 \cdot 8$$

Here the result is 2.666 666 ..... This means we have to go on with .666 666... which brings us back to where we started. Thus

$2/3 = .525\ 252\ 525\ 5\overline{25}_8$

Since the mantissa consists of 27_{10} bits we have to restrict ourselves to 9 octal digits.

$1/3 \equiv 0\ 01\ 111\ 111\ \ 101\ 010\ 101\ 010\ 101\ 010\ 101\ 010\ 101_2$

or $1/3 \equiv 177\ 525\ 252\ 525_8$

β) Representation of negative Numbers $N < 0$.

A negative number is given by the complement of all 36 bits of its absolute value.

Example:

$N \equiv -3$

Since $+3 \equiv 0\ 10\ 000\ 010\ 110 \longleftrightarrow 0$, we obtain

$-3 \equiv 1\ 01\ 111\ 101\ 001 \longleftrightarrow 1$ (binary)

Notice: Sign bit, mantissa, and characteristic are complemented. On the other hand, +3 has a characteristic c = 2, and $-3 = -3/4 \cdot 2^2$ has also a characteristic c = 2. But nevertheless the 8 bits comprising the characteristic portion of a negative number are given by the one's complement of the true biased characteristic.

γ) Representation of N = 0.

The number "zero" is represented by 36_{10} "zeros", i.e. all 36 bits of the register are zeros.

A number N being represented in the above mentioned format is said to be "normalized" and "packed".

"Normalized" means: the most significant bit of the mantissa is in stage i_{26}.

"Packed" means: mantissa, biased characteristic, and sign bit are placed into one register in the 1—8—27 format.

B) Floating Point Commands

During all Floating Point arithmetic processes the programmer possesses the option to round or not to round the rightmost bit of the mantissa of the result of such a process. This is handled by the command

FR j- (Floating Point Round Option)

Octal operation code = 03

Function:

If j = 1: do not round results of all following floating point operations

If j = 0: from now on round again all results of all subsequent floating point operations. This is also accomplished by a computer Master Clear, i.e. rounding is the normal operation of the machine.

j is determined as in PU jv etc.

FA uv (Floating Point Add)

Octal operation code = 64

Function:

(u) : normalized, packed Floating Point number

(v) : normalized, packed Floating Point number

Then $(u) + (v) \longrightarrow Q$

where $(Q)_f$ = normalized, packed and rounded (optional) Floating Point result.

FS uv (Floating Point Subtract)

Octal operation code = 65

Function:

$(u) - (v) \longrightarrow Q$

where (u), (v), and $(Q)_f$ are normalized, packed numbers, and $(Q)_f$ is rounded (optional).

FM uv (Floating Point Multiply)

Octal operation code = 66

Function:

$(u) \cdot (v) \longrightarrow Q$

where (u), (v) and $(Q)_f$ are normalized, packed numbers, and $(Q)_f$ is rounded (optional).

FD uv (Floating Point Divide)

Octal operation code = 67

Function:

$\frac{(u)}{(v)} \longrightarrow Q$

(u), (v) and $(Q)_f$ are normalized, packed numbers, and $(Q)_f$ is rounded (optional).

FP uv (Floating Point Polynomial Multiply)

Octal Operation code = 01

Function:

$$(u)\cdot(Q)_i + (v) \longrightarrow Q$$

(u), $(Q)_i$, and (v) have to be normalized and packed Floating Point numbers. $(Q)_f$ is also normalized and packed, and it is rounded (optional). This rounding process is performed in the following way: At first the product $(u)\cdot(Q)_i$ is rounded. Then the sum $(u)\cdot(Q)_i + (v)$ is rounded again. If no rounding is wanted, neither the product nor the sum are rounded.

The usefulness of this command is illustrated by the following example:

Compute

$$a_6x^6 + a_7 x^7 + \ldots\ldots a_1 x + a_0,$$

where

(00100) = a_6

(00101) = a_7

(00102) = a_6

⋮

(00107) = a_1

(00110) = a_0

and (00077) = x

Solution:

b	TP	00100	0		$a_6 \longrightarrow Q$
b+1	RP	10010	b+3	}	Develop
b+2	FP	00077	00101		$a_6 x^6 + \text{-----} + a_0 \longrightarrow Q$
b+3	NI				

FI uv (Floating Point Inner Product)

Octal operation code = 02

Function:

$$(u)\cdot(v) + (Q)_i \longrightarrow Q$$

(u), (v), and $(Q)_i$ have to be normalized and packed. $(Q)_f$ is also normalized, packed and rounded (optional). As in the FP uv, either both, product $(u)\cdot(v)$ and sum $(u)\cdot(v) + (Q)_i$, are rounded, or neither one is rounded.

Notice: Location F_4 = 00003 is used for temporary storage of $(Q)_i$.

This command is useful for computing sums of products as e.g.

$$\sum_{i=0}^{n} a_i\, b_i$$

Example: Let n = 5 and

(00100) = a_0 (00200) = b_0

(00101) = a_1 (00201) = b_1

⋮ ⋮

(00105) = a_5 (00205) = b_5

To develop the sum $\sum_{i=0}^{5} a_i\, b_i = a_0\, b_0 + a_1\, b_1 + \text{----} + a_5\, b_5$

we write the program:

```
    c   FM  00100  00200        a0 b0 → Q
  c+1   RP  30005  c+3    }     compute
  c+2   FI  00101  00201  }     a0 b0 + ---- + a5 b5 → Q
  c+3   NI
```

UP uv (Unpack)

Octal operation code = 03

Function:

Assume (u) is a packed, normalized Floating Point number, Unpack (u) in the following way:

$(u)_m \rightarrow u_m$, sign bit u_{35} is also placed into u_{34} thru u_{27}

If $(u) \geq 0$: $(u)_c \rightarrow v_c$ } v_{35} and v_{26} thru v_0 are all "zeros"

If $(u) < 0$: $(u)'_c \rightarrow v_c$ }

Definition:

$(u)_m$ is the mantissa part of the number

$(u)_c$ is the characteristic part of the number

u_m means: bits u_{26} ..... u_0

v_c means: bits v_{34} ..... v_{27}

Example:

$(u)_i = 0\ \underbrace{10\ 000\ 011}_{C}\ \underbrace{101\ 0\longleftrightarrow 0}_{m} \equiv +5 = \frac{5}{8} \cdot 2^3$

Unpack (u) gives the result:

$(u)_f = 0\ 00\ 000\ 000\ 101\ 0\longleftrightarrow 0$

$(v)_f = 0\ 10\ 000\ 011\ 0\longleftrightarrow 0$

$(u)_i = 1\ 01\ 111\ 100\ 010\ 1\longleftrightarrow 1 \equiv -5 = -\frac{5}{8} \cdot 2^3$

Unpack (u):

$(u)_f = 1\ 11\ 111\ 111\ 010\ 1\longleftrightarrow 1$

$(v)_f = 0\ 10\ 000\ 011\ 0\longleftrightarrow 0$

As you see: When unpacking (u) the rightmost 27_{10} bits of $(u)_f$ (mantissa part) are equal to those of $(u)_i$, but the other 9 bits are all sign bits. However, $(v)_f$ contains always the true biased characteristic of the number $(u)_i$. This means that the characteristic part of $(u)_i < 0$ is at first complemented and then placed into the proper part of v (in above example $(u)_i = -5$: $(u)_c$ = 01 111 100 is complemented and 10 000 011 $\rightarrow v_c$)

NP uv (Normalize Pack)

Octal operation code = 04

Function:

Assume $(u)_i$ is an unpacked mantissa, normalized or not, i.e. the binary point is assumed to be between u_{27} and u_{26}, but the most significant bit of the mantissa may or may not be at u_{26}.

Assume further that $(v)_i$ contains an unpacked true biased characteristic in the characteristic part.

Normalize and pack in the following way:

$(u)_i$ is normalized and rounded (optional) such that now the mantissa is in u_m.

$(v)_i$ is adjusted according to the number of shifts which were needed to normalize $(u)_i$, and sent to u_c. If $(u)_i$ was negative, complemtnt $(v)_c$ and then send it to u_c.

As you see: $(u)_i$ contains a mantissa which may or may not be normalized. However the binary point is assumed to be between u_{27} and u_{26}. Therefore, if e.g. the most significant bit of the mantissa was in u_{28}, two right shifts are needed to bring it into u_{26}. This is equivalent to a multiplication with 2^{-2}. Therefore the true biased characteristic is increased by two. Then the machine determines whether $(u)_i$ is positive or negative. If positive the adjusted characteristic is sent to u_c. If negative the complement of the adjusted characteristic is sent to u_c. In any case $(u)_f$ is a normalized, packed and rounded (optional) Floating Point number.

Example:

Let $(u)_i$ = 000 000 101 0———0 = m

binary point

$(v)_i$ = 010 000 001 0———0

Normalizing $(u)_i$ means: shift it 3 places to the right, i.e. multiply mantissa m with 2^{-3}. Now $(u)_i$ = 000 000 000 101 0———0 contains $m \cdot 2^{-3}$. Therefore a "3" is added to the biased characteristic in v: $(v)_f$ = 0 10 000 100 0———0.

Then $(v)_c \longrightarrow u_c$:

$(u)_f$ = 0 10 000 100 101 0———0

= 10_{10}

What we actually produced was this:

We have m and C = 201_8, i.e. c = 1.

The number we are looking for is, therefore, $m \cdot 2^1$

But m was not in the range $\frac{1}{2} \leq m < 1$, since it was = 5. As a result $m \cdot 2^{-3} = \frac{5}{8}$ was produced. However, in order to keep the value $m \cdot 2^1$, we now have to add 3 to the characteristic, since

$m \cdot 2^1 = \underbrace{m \cdot 2^{-3}}_{\frac{1}{2} \leq \text{mantissa} < 1} \cdot 2^{\overbrace{1+3}^{\text{characteristic}}}$

C) Some Remarks on Machine Operations occurring during Floating Point Arithmetic Processes.

Assume we give the command FA uv. In order to add (u) and (v) the machine obviously has to unpack both numbers, align the mantissas according to the difference of the characteristics, add the mantissas, round the result (pptional), and normalize and pack the result in Q. Without going into details the following is stated:

> During Floating Point Arithmetic operations the Accumulator is used. This means that $(A)_i$ is destroyed, whereas $(A)_f$ contains the mantissa of the result such that the most significant bit of the mantissa is in A_{62} (i.e. m is given by the rightmost 27_{10} bits of A_L). If a result is a "zero", then $(A)_f = 0$, $(Q)_f = 0$.

During the "unpack" procedure the two characteristics are saved in two special registers, C and D. The characteristic of the result is developed in another register, called S-register.

Since there is no room for going into details of sequences of Floating Point commands, the use of A and Q as operands and the results of such operations are not explained here. It is, therefore, pointed out that the reader may check the appropriate literature on the Floating Point System before using A and/or Q as u and/or v in these instructions.

During addition, subtraction, multiplication, division, and normalizing, one of the two following situations may occur:

1) The <u>characteristic</u> of the result is <u>too large</u>, i.e. c would be $> 127_{10}$. If this is detected, a computer "A" Fault is generated, and the "char. overflow" light on the console is illuminated.
2) The <u>characteristic</u> of the result is <u>too small</u>, i.e. c would be <-128. In this case the result is replaced by "0", i.e. A and Q are cleared.

D) Use of "Transmit" and "Compare" Instructions for Floating Point Numbers.

Because of the format chosen for the representation of numbers in Floating Point, the use of the following instructions is preserved <u>for operations with Floating Point</u> numbers:

TP uv	EJ uv
TM uv	TJ uv
TN uv	SJ uv
	ZJ uv

Examples:

Assume (u) = 010 000 011 101 0———0 ≡ + 5

After TN u v we have

(v) = 101 111 100 010 1———1 ≡ −5

Assume you want to compare (u) and (u+1).

```
Then:   a   TP   u   A
      a+1   EJ  u+1  v
```

Assume you performed an arithmetic operation in Floating Point. In order to find out whether or not this number is positive, just give a SJ uv without transferring the result from Q to A. This is possible, because $(A)_f$ contains the resulting mantissa and naturally Sign bits. In order to find out whether the result is "0", just give a ZJ uv, since $(A)_f$ has to be = 0, if result = 0.

TABLE I

Sample Card Unit Routines

READ SINGLE CARDS

The computer instructions below withdraw two cards from the read card feed hopper, position the first card for reading and transmit its contents to the computer, and continue advancing it through the read channel until it reaches its final position in the receiving stacker. (The second card withdrawn from the hopper is left in the first station.)

EF-v	(v) = 40 00000 00004 (START, PICK READ CARD)	1 cycle
EF-v	(v) = 40 00000 00005 (START, PICK READ CARD, READ) Within 140 ms. of the execution of this instruction, the execution of the following three instructions should be initiated:	1 cycle
ERjv (j=0) ERjv (j=1) ERjv (j=1)	Repeat for each card row, each repetition being initiated not later than 10 ms. after the beginning of the corresponding row point. *	
EF-v	(v) = 40 00000 00000 (START)	1 cycle
EF-v	(v) = 40 00000 00000 (START)	1 cycle
EF-v	(v) = 40 00000 00000 (START)	1 cycle
EF-v	(v) = 40 00000 00000 (START) (card read now placed in receiving stacker)	1 cycle

* This means (here and in following tables): After you read one row and used already 27.8 m sec. you may give the first ER after an additional 10 m sec. However, as you probably see: do not try to do this for each row again and again, because these additional m sec. will soon increase such that a "B"-Fault has to occur!

TABLE II

READ CONSECUTIVE CARDS

A. Single Card Mode

To transmit information from n consecutive cards, without selecting the FREE RUN bit, a routine similar to the one above for READ SINGLE CARD is executed:

EF-v	(v) = 40 00000 00004 (START, PICK READ CARD)	1 cycle
EF-v	(v) = 40 00000 00005 (START, PICK READ CARD, READ) Within 140 ms. of the execution of this instruction the execution of the following three instructions should be initiated:	1 cycle
ERjv (j=0) ERjv (j=1) ERjv (j=1)	Repeat for each card row, each repetition being initiated not later than 10 ms. after the beginning of the corresponding row point.	
EF-v	(v) = 40 00000 00005 (START, PICK READ CARD, READ) Execute this instruction within 10 ms. of the beginning of row point 12 of the previous cycle. Within 170 ms. of the execution of this instruction, the execution of the following three instructions should be initiated:	n-1 cycles
ERjv (j=0) ERjv (j=1) ERjv (j=1)	Repeat for each card row, each repetition being initiated not later than 10 ms. after the beginning of the corresponding row point.	
	Execute this same instruction (n-2) more times, initiating each execution with 10 ms. of row point 12 of the previous card cycle. After each such instruction is executed, initiate execution of its associated External Read Instructions within 170 ms.	
EF-v *	(v) = 40 00000 00000 (START)	1 cycle
EF-v *	(v) = 40 00000 00000 (START)	1 cycle
EF-v *	(v) = 40 00000 00000 (START)	1 cycle
EF-v *	(v) = 40 00000 00000 (START)	1 cycle

At the conclusion of the above program, the n^{th} card (or last card read) is found in the receiving stacker; an $(n+1)^{th}$ card is in the first reading station

* Execute these instructions within 10 ms. of row point 12 of the previous cycle to obtain continuous card cycles from the Card Unit.

TABLE III

B. Free Run Mode

The computer instructions below withdraw n+1 cards from the read card feed hopper, one at a time, position n of them for reading and transmit their information content to the computer, and continue advancing these n cards through the read channel until the last card reaches its final position in the receiving stacker. (The $n+1^{st}$ card is left in the first reading station.)

EF -v	(v) = 40 00000 00004 (START, PICK READ CARD)	1 cycle
EF-v	(v) = 40 00000 00045 (START, FREE RUN, PICK READ CARD, READ) Within 140 ms. of the execution of this instruction, the execution of the following three instructions should be initiated.	1 cycle (first free run cycle)
ERjv (j=0) ERjv (j=1) ERjv (j=1)	Repeat for each card row, each repetition being initiated not later than 10 ms. after the beginning of the corresponding row point. (A series of card cycles is initiated.)	
	170 ms. from the completion of the External Read Instruction for row 12 of the previous card cycle, execution of the following three instructions should be initiated.	(n-3) cycles (intermediate free run cycles)
ERjv (j=0) ERjv (j=1) ERjv (j=1)	Repeat for each card row each repetition being initiated not later than 10 ms. after the beginning of the corresponding row point: Same procedure for next (n-4) cycles	
	170 ms. from the completion of the External Read Instructions for row 12 of the last previous card cycle, execution of the following three instructions is initiated.	1 cycle (the next to last cycle of free run)
ERjv (j=0) ERjv (j=1) ERjv (j=1)	Repeat for each card row, each repetition being initiated not later than 10 ms. after the beginning of the corresponding row point.	
EF-v	(v) = 40 00000 00020 (START, STOP) This instruction must be executed within 10 ms. of the beginning of row point 12 of this cycle.	

CON'T.

TABLE III (Continued)

ERjv (j=0) ERjv (j=1) ERjv (j=1)	170 ms. from the completion of the External Read Instructions for row 12 of the previous card cycle, execution of the following three instructions is initiated: Repeat for each card row, each repetition being initiated not later than 10 ms. after the beginning of the corresponding row point. (nth card read during this cycle and free run selections are dropped.)	1 cycle (last cycle of free run)
EF-v *	(v) = 40 00000 00000 (START)	1 cycle
EF-v *	(v) = 40 00000 00000 (START)	1 cycle
EF-v *	(v) = 40 00000 00000 (START)	1 cycle
EF-v *	(v) = 40 00000 00000 (START)	1 cycle

* Execute these instructions within 10 ms. of row point 12 of the previous card cycle to obtain continuous card cycles from the Card Unit.

TABLE IV

PUNCH SINGLE CARDS

The computer instructions below withdraw two cards from the punch card feed hopper, position the first card for punching and punch information in it, and continue advancing it through the punch channel until it reaches its final position in the receiving stacker. (The second card withdrawn from the hopper is left in Station 1.)

EF-v	(v) = 40 00000 00010 (START, PICK PUNCH CARD)	1 cycle
EF-v	(v) = 40 00000 00010 (START, PICK PUNCH CARD)	1 cycle
EF-v EFjv (j=0) EFjv (j=1) EFjv (j=1)	(v) = 40 00000 00002 (START, PUNCH) Within 140 ms. of the execution of this instruction, the execution of the following three instructions should be initiated: Repeat for each card row, each repetition being initiated not later than 1.5 ms. after the beginning of the corresponding row point.*	1 cycle
EF-v	(v) = 40 00000 00000 (START) (Card is punched in this cycle)	1 cycle
EF-v	(v) = 40 00000 00000 (START)	1 cycle
EF-v	(v) = 40 00000 00000 (START)	1 cycle

* See remarks on Table 1

TABLE V

A. SINGLE CARD MODE

To punch n cards without selecting the FREE RUN bit.

EF-v	(v) = 40 00000 00010 (START, PICK PUNCH CARD)	1 cycle
EF-v	(v) = 40 00000 00010 (START, PICK PUNCH CARD)	1 cycle
EF-v EWjv (j=0) EWjv (j=1) EWjv (j=1)	(v) = 40 00000 00012 (START, PICK PUNCH CARD, PUNCH) Within 140 ms. of the execution of this instruction, the execution of the following three instructions should be initiated. Repeat for each card row. Each repetition being initiated no later than 1.5 ms. after beginning of each row point. Execute the above sequence of instructions n-1 times. Each sequence starting with the EF, and given no later than 170 ms. of the previous row point.	(n-1) cycles
EF-v EWjv (j=0) EWjv (j=1) EWjv (j=1)	(v) = 40 00000 00002 (START PUNCH) Timing as above. nth card punched in this cycle.	1 cycle
EF-v EF-v EF-v	(v) = 40 00000 00000 (START) Each EF within 10 ms. of row point 12 of previous instruction	3 cycles

At the conclusion of the above program a single card is left in Station 1, and all cards punched are in the stacker.

TABLE VI

B. FREE RUN MODE

The computer instructions below withdraw n+2 cards from the punch feed hopper, one at a time: the first n cards are advanced through the punching channel, punched and sent to the receiving stacker; the $(n+1)^{st}$ card is advanced to the Stacker; the $(n+2)^{nd}$ card is left in Station 1.

EF-v	(v) = 40 00000 00010 (START, PICK PUNCH CARD)	1 cycle
EF-v	(v) = 40 00000 00010 (START, PICK PUNCH CARD)	1 cycle
EF-v EWjv (j=0) EWjv (j=1) EWjv (j=1)	(v) = 40 00000 0005_2 (START, FREE RUN, PICK PUNCH CARD, PUNCH) Within 140 ms. of the execution of this instruction, the execution of the following three instructions should be initiated: Repeat for each card row, each repetition being initiated not later than 1.5 ms. after the beginning of the corresponding row point. (A series of card cycles is initiated.)	1 cycle (first free run cycle)
    EWjv (j=0) EWjv (j=1) EWjv (j=1)	170 ms. from the completion of the first External Write Instruction (j=1) for row 12 of the previous card cycle, execution of the following three instructions should be initiated: Repeat for each card row each repetition being initiated not later than 1.5 ms. after the beginning of the corresponding row point. (Same procedure for next (n-4) cycles)	(n-3) cycles (intermediate free run cycles)
    EWjv (j=0) EWjv (j=1) EWjv (j=1) EF-v	170 ms. from the completion of the External Write Instructions for row 12 of the last previous card cycle, execution of the following three instructions is initiated. Repeat for each card row, each repetition being initiated not later than 1.5 ms. after the beginning of the corresponding row point. (v) = 40 00000 00020 (START, STOP) This instruction must be executed within 10 ms. of the beginning of row point 12 of this cycle.	1 cycle (next to last cycle of free run)

CON'T.

TABLE VI (Continued)

EWjv (j=0) EWjv (j=1) EWjv (j=1)	170 ms. from the completion of the External Write Instructions for row 12 of the previous card cycle, execution of the following three instructions is initiated: Repeat for each card row, each repetition being initiated not later than 1.5 ms. after the beginning of the corresponding row point. (FREE RUN selections are dropped)	1 cycle (last cycle of free run)
EF-v *	(v) = 40 00000 00000 (START) (n^{th} card is punched)	1 cycle
EF-v *	(v) = 40 00000 00000 (START)	1 cycle
EF-v *	(v) = 40 00000 00000 (START) (nth card put in punch receiving stacker)	1 cycle
EF-v *	(v) = 40 00000 00000 (START) $(n + 1)^{st}$ Card is put in receiving stacker	1 cycle

* Execute these instructions within 10 ms. of row point 12 of the previous card cycle to obtain continuous card cycles from the Card Unit.

TABLE VII

READ AND PUNCH SIMULTANEOUSLY

The computer instructions below:

1) Withdraw two cards, one at a time, from the punch feed hopper; the first card withdrawn is punched and placed into the output stacker; the second card withdrawn is left in Station 1.

2) Withdraw two cards, one at a time, from the read feed hopper; the first card withdrawn is read and sent to the receiving stacker; the second card withdrawn is left in Station 1.

EF-v	(v) = 40 00000 00010 (START, PICK PUNCH CARD)	1 cycle
EF-v	(v) = 40 00000 00014 (START, PICK PUNCH CARD, PICK READ CARD)	1 cycle
EF-v EWjv (j=0) EWjv (j=1) EWjv (j=1) ERjv (j=0) ERjv (j=1) ERjv (j=1)	(v) = 40 00000 00007 (START, PUNCH, PICK READ CARD, READ) Within 140 ms. of the execution of this instruction, the execution of the following six instructions should be initiated: Repeat for each card row, each repetition being initiated not later than 1.5 ms. after the beginning of the corresponding row point. (READ CARD read in this cycle)	1 cycle
EF-v	(v) = 40 00000 00000 (START) (PUNCH CARD punched in this cycle)	1 cycle
EF-v	(v) = 40 00000 00000 (START)	1 cycle
EF-v	(v) = 40 00000 00000 (START) (PUNCH CARD put in punch receiving stacker)	1 cycle
EF-v	(v) = 40 00000 00000 (START) (READ CARD put in receiving stacker)	1 cycle

Simultaneous Reading and Punching of n cards can be done by applying a similar method, either in Single Card Mode or in Free Run!

TABLE VIII

1103A and 1105

Octal Magnetic Tape Functions

1103A

The table given below shows the octal equivalent of the bit assignments for the selection of Magnetic Tape operations:

02	00000	00000	Select Magnetic Tape ("Master Bit")
00	00000	10000	Uniservo Number 1
00	00000	20000	" " 2
	⋮		⋮ ⋮ ⋮
00	00000	70000	" " 7
00	00001	00000	" " 8
	⋮		⋮ ⋮ ⋮
00	00001	20000	" " 10
00	00006	00000	Write in high density | Write in low density } 1105!
00	00016	00000	Write in low density | Write in high density } 1105!
00	00000	00000	0.0" Blockette Space
00	00020	00000	0.1" " "
00	00040	00000	1.2" " "
00	00000	00000	1.2" Block Space
00	00100	00000	2.4" " "
00	00002	00000	Read forward
00	00012	00000	Read backward
00	00600	00000	Stop (after "Read" or "Write" operation only)
00	00004	00000	Move forward
00	00014	00000	Move backward
00	00000	0xxxx	xxxx = Number of blocks ("Move" operation only)
00	00200	00000	Rewind
00	00400	00000	Rewind with interlock
00	00001	50000	Set "Read Bias" to Normal
00	00001	60000	" " " " Low
00	00001	70000	" " " " High
00	00060	00000	Select Var. Bl. Length or Cont. Data Input
00	00010	00000	Change Mode

Examples:

Write in high density on Uniservo 3 using Free Run, 1.2" Blockette Space, 1.2" Block Space:

(v) = 02 00046 30000

Write in low density one block on Uniservo 9 using 0.0" Blockette Space, 2.4" Block Space:

(v) = 02 00717 10000

TABLE VIII (Continued)

Read forward in Free Run from Uniservo 6, Fixed Block Length:

(v) = 02 00002 60000

Move backward Uniservo 8 19_{10} blocks; Fixed Block Length:

(v) = 02 00015 00023

Read forward from Uniservo 4 in Var. Block Length:

(v) = 02 00062 40000

1105

a) Bypass Mode

In order to select tape operations without using a buffer, i.e. to bypass a buffer, give an EF -v, where

(v) = 00 10000 04000 to bypass buffer 1

(v) = 00 20000 04000 to bypass buffer 2

Tape operations in Bypass Mode are now selected as explained for 1103A with the addition, that

i_{31} = 1 is Master Bit for TCU 2,

i_{30} = 1 is Master Bit for TCU 1.

Either one of these bits can be used depending upon the buffer which is to be bypassed (Both bits may not be specified!)

Therefore:

If Buffer 1 is to be bypassed, EF "Tape" operations are selected with a (v) = 01 xxxxx xxxxx.

If Buffer 2 is to be bypassed, EF "Tape" operations are selected with a (v) = 02 xxxxx xxxxx.

b) Normal (Buffer) Mode

Tape operations are selected as explained for 1103A, with the addition that either one of the Tape Control Units can be specified (see under Bypass Mode). Notice that "Write Density" for 1105 is reverse to 1103A.

Example:

Write in high density, Fixed Block Length, Unis. 3, Free Run, 1.2" Blockette Space, 1.2" Block Space,
TCU 1: EF -v with
(v) = 01 00056 30000

The same for Unis. 3, TCU 2: EF -v with (v) = 02 00056 30000.

Interrupt Selection:

Bit v_{24} = 1 in (v) of EF "Tape" instruction

Buffer operations

(v) of EF -v			Function
00	10000	02000	Clear Buffer 1 and switch to "load"
00	20000	02000	" " 2 " " " "
00	10000	01000	"End Transfer", Buffer 1 switch to "unload"
00	20000	01000	" " " 2 " " "
00	10000	00400	"Write Buffer" 1 (similar for Buffer 2)
00	10000	00200	"Read Buffer" 1 (similar for Buffer 2)
00	10000	00100	Read Word Counter of Buffer 1 into IOB_u
00	20000	00100	" " " " " 2 " "

TABLE IX (Continued)

TAPE OPERATION	DO	DON'T
Move Forward/ Backward	Designate in (v) of EF instruction: 1) Magnetic Tape Master Bit 2) Move Forward/Backward 3) Number of blocks to be moved. 4) Uniservo Number	Program a Move Forward/ Backward of more blocks than have been recorded. Terminate a move operation with an EF Stop Tape instruction.
Rewind	Designate in (v) of EF instruction: 1) Magnetic Tape Master Bit 2) Rewind 3) Uniservo Number	
Rewind/ Interlock	Designate in (v) of EF instruction: 1) Magnetic Tape Master Bit 2) Rewind/Interlock 3) Uniservo Number	Reference a uniservo rewound with interlock until the uniservo door interlock switch has been opened and closed.
Change Bias	Designate in (v) of EF instruction: 1) Magnetic Tape Master Bit 2) Change bias level bits Program a return to normal bias level unless a computer Master Clear occurs which accomplishes this.	Include any other tape operation in the EF Change Bias instruction. Program two successive changes to high or low bias level.

TABLE X

AVAILABLE COMPUTATION TIMES

Fixed Block Length Mode

The following table shows the recommended programming times allowable for computation between tape operations. The safe times listed are based on theoretical timing conditions and then adjusted to include a safety factor. The safety factor takes into consideration fluctuations in normal operating characteristics and unpredictable component variations. The theoretical times do not include any execution time for the instructions effecting the particular tape operation under consideration. Because of possible adjustments on different machines, the theoretical times may vary slightly from installation to installation.

Situation	Theoretical Time	Safe Time (based on theoretical time)
READ OPERATION		
Between EF Start Tape and first ER		
1.2" block space	46.1 ms	31 ms
2.4" block space	58.9 ms	40 ms
(See also leader and reversal delay)		
Between successive ER instructions		
at 120 lines per inch	468 μs	350 μs
at 50 lines per inch	1200 μs	900 μs
Across the blockette space		
1.2" blockette space	12 ms	9 ms
0.1" blockette space	1 ms	750 μs
Across the block space		
1.2" block space	12 ms	9 ms
2.4" blockette space	24 ms	18 ms
Between last ER and EF Stop Tape		
1.2" block space	3.75 ms	1 ms *
2.4" Block space	17.25 ms	14 ms *

* Observance of the stated time for this interval is important if reading is continued. If the stop occurs too close to the first lines of the next block, not enough time is allowed for the tape to accelerate before the first lines are encountered in the following reading operation.

TABLE X (Continued)

If this should be the case, the first lines will not be sensed from the tape. For a 1.2 inch block space, a zero time delay would be recommended if it were feasible, as this would stop tape movement such that the read/write head would be positioned midway through the inter-block space. Where other instructions must be executed before the EF Stop Tape instruction, it is recommended that the minimum time does not exceed the value shown in the table. The 2.4 inch case is not as critical, but again it is recommended that the value shown is not exceeded.

Situation	Theoretical Time	Safe Time (based on theoretical time)
WRITE OPERATION		
Between EF Start Tape and first EW		
1.2" block space	46.1 ms	31 ms
2.4" block space	58.9 ms	40 ms
(See also leader and reversal delays)		
Between successive EW instructions		
at 128 lines per inch	468 μs	350 μs
at 50 lines per inch	1200 μs	900 μs
Across the blockette space		
1.2" blockette space	12 ms	9 ms
0.1" blockette space	1.0 ms	750 μs
Across the block space		
1.2" block space	12 ms	9 ms
2.4" block space	24 ms	18 ms
Between last EW and EF Stop Tape		
1.2" block space	12 ms	1 ms **
2.4" block space	24 ms	1 ms **
LEADER DELAY		
Starting reading or writing from rewound position		
1.2" block space	1546.1 ms	1237 ms
2.4" block space	1558.9 ms	1247 ms
REVERSAL DELAYS		
Reading or writing in opposite direction from previous tape movement	611.1 ms	458 ms

** Next Page

TABLE X (Continued)

** During writing, the limitation on this time is actually dependent upon the block spacing. If the time allowed for block spacings of 1.2 inch or 2.4 inch is exceeded, a No Information fault occurs. For the sake of uniformity of block spacings, however, it is recommended that the time does not exceed one ms for either of the two spacings between blocks.

Situation	Theoretical Time	Safe Time (based on theoretical time)
REWIND INITIATION DELAYS (preventing another IOB to TCR transmission)		
Previous forward direction	35 ms	22 ms
Previous backward direction	600 ms	450 ms
CHANGE BIAS DELAY (preventing another IOB to TCR transmission)	35 ms	22 ms

TABLE XI

SYNOPSIS OF
VARIABLE BLOCK LENGTH MODE OPERATION

TAPE OPERATION	DO	DON'T
Read Forward/ Backward	Designate in (v) of EF instruction: 1) Magnetic Tape Master Bit 2) Read Forward/Backward 3) Uniservo Number 4) Variable/Continuous Bits Read and check content of IOA before IOB is read; then, read IOA again or read IOE.	Exceed allowable computation time between the EF Start instruction and the 1st ER, between successive ER instructions, or between the last ER and EF Stop Tape instruction. Include uniservo number in EF Stop Tape instruction.
Write	Designate in (v) of EF instruction: 1) Magnetic Tape Master Bit 2) Variable/Continuous Bits 3) Uniservo Number 4) Write Operation Program an EW instruction for each word to be written. Program an EF Stop Tape instruction immediately after the last word to be written in <u>each</u> block.	Exceed allowable computation time between the EF Start and 1st EW, between successive EW's, and between the last EW and EF Stop Tape instructions. Include uniservo number in EF Stop Tape instruction.
Stop	Designate in (v) of EF instruction: 1) Magnetic Tape Master Bit 2) Stop Bits Program an EF Stop Tape during reading for all desired stops, except at end of record; during writing, to terminate tape movement and create block spacing.	Designate a uniservo in (v).

TABLE XI (Continued)

TAPE OPERATION	DO	DON'T
Move Forward/ Backward	Designate in (v) of EF instruction: 1) Magnetic Tape Master Bit 2) Move Forward/Backward 3) Variable/Continuous Bits 4) Number of blocks to be moved 5) Uniservo Number	Terminate Move Operation with an EF Stop Tape instruction. Program a Move Forward/ Backward of more blocks than are recorded in the direction of the movement.
Rewind	See Fixed Block Length Mode Operation	
Rewind Interlock		
Change Bias		

TABLE XII

AVAILABLE COMPUTATION TIMES

Variable Block Length Mode

The following table shows the recommended programming times allowable for computation between tape operations. The safe times listed are based on theoretical timing conditions and then adjusted to include a safety factor. The safety factor takes into consideration fluctuations in normal operating characteristics and unpredictable component variations. The theoretical times do not include any execution time for the instructions effecting the particular tape operation under consideration. Because of possible adjustments on different machines, the theoretical times may vary slightly from installation to installation.

Situation	Theoretical Time	Safe Time (Based on theoretical time)
READ OPERATION		
Between EF Start Tape and first ER (1.4" block space)	50.85 ms	33 ms
(See also leader and reversal delays)		
Between successive ER instructions (at 128 lines per inch)	468 μs	350 μs
Across the block space (1.4" block space)	14 ms	10 ms
Between last ER and EF Stop Tape		
Inter-block stop	7.0 ms	1 ms *
Intra-block stop	468 μs	250 μs

* Observance of the stated time for this interval is important if reading is continued. If the stop occurs too close to the first lines of the next block, not enough time is allowed for the tape to accelerate before the first lines are encountered in the following reading operation. If this should be the case, the first lines will not be sensed from the tape. For a 1.4 inch block space, a zero time delay would be recommended if it were feasible, as this would stop tape movement such that the read/write head would be positioned midway through the interblock space. Where other instructions must be executed before the EF Stop Tape instruction, it is recommended that the minimum time does not exceed the value shown in the table.

TABLE XII (Continued)

Situation	Theoretical Time	Safe time (based on theoretical time)
WRITE OPERATION		
Between EF Start Tape and first EW (1.4" block space)	50.85 ms	33 ms
(See also leader and reversal delays)		
Between successive EW instructions (at 128 lines per inch)	468 μs	350 μs
Across the block space (1.4" block space)	14 ms	10 ms
Between last EW and EF Stop Tape (1.4" block space)	468 μs **	250 μs
LEADER DELAY		
Starting reading or writing from rewound position	1550.85 ms	1240 ms
REVERSAL DELAY		
Reading or writing in opposite direction from previous tape movement	615.85 ms	462 ms
REWIND INITIATION DELAYS (preventing another IOB to TCR transmission)		
Previous forward direction	35 ms	22 ms
Previous backward direction	600 ms	450 ms
CHANGE BIAS DELAY		
(preventing another IOB to TCR transmission)	35 ms	22 ms

** During writing, the limitation on this time is actually dependent upon the block spacing. If the time allowed for the block spacing of 1.4 inch is exceeded, a No Information occurs.

Special Remarks on the Buffer System

The following remarks dedicated to the design of the buffer system might, at first glance, seem to be unnecessary from a programmer's point of view. However, these engineering details (modified to "programmer's language" as much as possible) affect the programming of buffer and tape operations to a great extent. They are, therefore, presented here and might help to clarify some peculiar programming situations occuring during 1105 magnetic tape and buffer operations.

I. Handling of Buffer Activity and Inactivity

Each buffer possesses a so-called "Buffer Active Flip-Flop". If it is in the "1" state, the buffer is "active". If it is in the "0" state, the buffer is "inactive".

a) Which operations set a buffer to "active", and when?
A buffer is set to "active" in either one of the following three situations:

1) "Move Enable" from TCU
2) "Write Enable" from TCU AND buffer "Unload FF" in its "1" state (i.e. buffer in "unload" state)
3) "Read Enable" from TCU AND buffer "Unload FF" in its "0" state (i.e. buffer in "load" state)

Programming consequences:
A buffer is set to "active" by one of the following three instructions:

1) EF "Move Tape n blocks"
2) EF "Write Tape" provided the buffer is in the "unload" state.
3) EF "Read Tape" provided the buffer is in the "load" state.

(It is assumed that the buffer was "inactive" upon execution of these instructions, and that "Bypass Mode" has not been selected.)

b) By what means is an active buffer set to "inactive"?

The "Buffer Active FF" can be set to "0", if and only if a computer Main Pulse 5 occurs.

This MP 5 in conjunction with some other pulses (which essentially mean that the tape operation is completed, see below) set a buffer to "inactive".

Programming consequence:
Assume the computer stalled, i.e. hung up at one instruction such that no MP 5 can be generated by the machine. In this case a buffer cannot be set to "inactive", even if the corresponding tape operation which caused buffer activity is finished!

1) Read Tape operation:

Assume a tape is read (Free Run or not). One block enters a buffer. During this time the buffer is "active". When is it set to "inactive"?

> The buffer is set to "inactive" during the occurrence of the first MP 5 which follows the detection of an interblock space.

This is accomplished in the following way:

The above mentioned interblock space generates an End of Block signal which in turn generates a "Stop" signal in the Buffer Stop Control. This sets the "Run FF" (which was set to "1" earlier, essentially at the time the "Active FF" was set to "1") to "0". The fact "Run FF ≠ 1 and Buffer in Unload State" means that the "Active FF" is set to "0".

When is buffer activity resumed, if we are in Free Run?

> The buffer is set back to "active" after its content has been transferred to the computer and the switch from "unload" to "load" has been accomplished.

This is almost obvious because of a) #3. After the transfer buffer → computer and the switching to the "load" state we possess the "Read Enable" from TCU (because of the Free Run) and the "Unload FF = 0" pulse (because buffer is in "load" state). These two will cause a pulse "Set Buffer Active", and activity is, therefore, resumed.

(At the same time the Run FF is set to "1" again, since buffer is in "load" state again.)

2) Write Tape operation:

Switching of the buffer from "active" to "inactive" and back to "active" (if Free Run) during a Write Tape operation is accomplished in a manner similar to that of a Read Tape operation.

> The buffer is set to "inactive" during the occurrence of the first MP 5 which follows the detection of a buffer "Limit Count" (meaning: buffer is empty, i.e. BWK = 1 and "Back BWK" pulse).
> The buffer is set back to "active" (Free Run operation) after the transfer of the next block of data from computer to buffer and the switch from "load" to "unload".

3) Move Tape operation

Assume a tape is moved n blocks. The buffer connected with the same TCU is set to "active" at the start of this operation, as we saw earlier.

When is buffer activity dropped?

> The buffer is set to "inactive" after the completion of the Move operation.

This is done in the following way: The "Move" operation placed the number n (= number of blocks to be moved) into the Block Counter. As long as this BLK ≠ 0, the buffer cannot be set to "inactive". But precisely at the time BLK = 0, the buffer "Active FF" is set to "0". As you see: BLK = 0 also means: Move operation is finished.

Notice: Assume a moving of 3 blocks is attempted. Although TCU detects an Interblock Space after the first and second block, buffer activity is not dropped, since BLK ≠ 0.

c) Special programming situations during reading and writing:

1) Writing on Tape

As explained the EF "Write Tape" instruction sets the buffer connected with the same TCU to "active" only if the buffer is in the "unload" state. If the buffer is in the "load" state and such an EF-instruction is given the buffer will not be set to "active". The tape, however, starts right away.

Let us assume we have the following situation:
The buffer is in the "load" state (say, it is empty and inactive) and an EF "Write Tape" is executed (Free Run or not). What Happens? The tape movement is started, and within a certain time the TCU will require data from the buffer. This means that the buffer must be filled with data (from the computer) and must have completed the switch from "load" to "unload", before the TCU requires the first word. If the switch to "unload" is made, the buffer "Active FF" will now be set to "1" according to a) #2.

How much time is available between the EF "Write Tape" and EF "Write Buffer" cannot be determined exactly at the present time. It depends upon the start delay of the Uniservo II plus the time needed for moving to the beginning of the block which is to be written. However, 6 m sec. seem to be a safe time in any case. This time might be larger, but has to be determined by examining the appropriate literature, if available.

Assume the EF "Write Tape" is given, the buffer is in the "load" state, and the program does not fill the buffer with the first block (to be written) within the time available. In this case a No Information ("B") Fault is generated in the tape system, when the TCU requires the first word from the buffer (from IOT), and there is no word. This shows that a programmer who gives an EF "Write Tape" with the buffer in the "load" state creates a timing problem for himself, (namely for the first block), because before the first block is written on tape the Automatic Tape Control will not be effective, i.e. will not stop the tape! (Refer to "Automatic Tape Control")

2) Reading of Tape

During reading magnetic tape a situation similar to that of writing on tape can occur, i.e. an EF "Read Tape" is given at the time the buffer

involved is in the "unload" state (and inactive). As already explained the buffer will not be set to "active", but the tape movement is started. This means that the program has a certain time (see: c) #1) to transfer the content of the buffer to the computer. If this and the switch to "load" is accomplished in time, the buffer will be set to "active" because of a) #3.

If, however, the buffer content is not removed, before the transmission from tape to buffer is started, an "IO"-Fault occurs at the time the second word tries to enter IOT, whereas the first word is still there.

This again creates a timing problem for the programmer, namely for the first block to be read, since the Automatic Tape Control does not become effective before reading the first block.

II. Selection of Buffer Operations (EF "Buffer" instructions)

Serious trouble can occur for a program, if an EF "Buffer" instruction is given during the time the buffer is "active". It is true that this situation should never occur, because a programmer may execute a command referencing a buffer, if and only if he knows that the buffer is "inactive". However, errors in a program might lead to such a situation.

a) How can the selection attempted by an EF "Buffer" instruction become effective?

The EF "Buffer" command sends (like any EF) a 36-bit word to IOB. The computer is now satisfied, i.e. continues with the next instruction in sequence. The code word in IOB, however, is used to send pulses to the Start Control of the buffer referenced which are to set up an operation of this buffer.

If the buffer is "inactive", the selection is performed and IOB cleared for further use.

If, however, the buffer is "active", the selection cannot be performed, and IOB will not be cleared, but the program continues in sequence!

It is, therefore, very important that a programmer keeps in mind:

> An EF "Buffer" instruction can perform the selection, if and only if the buffer is "inactive" upon its execution. In any case (whether the selection could be made or not) the program continues in sequence.

A short explanation of how this is made in the buffer control is given below:

A pulse from "Buffer Active FF = 0" is used in the buffer Start Control to allow the pulses from IOB to pass through the appropriate gates and initiate those pulses which will set up the buffer for its operation. If the buffer is "active", the "Active FF" is in its "1" state and the buffer Start Control can, therefore, not initiate the operation.

On the other hand the Start Control initiates a "Resume" signal, if the operation attempted could be selected properly. This "Resume" signal is

used (via Buffer Command Timing Circuitry) to generate a "Clear IOB and Resume" pulse. This means that IOB can be cleared only if the operation selected by the EF "Buffer" instruction can be performed. Otherwise IOB continues holding the code, and any further reference of IOB establishes an IOB Lockout condition.

b) Illustrative Example:

The example below is a program which essentially is used after detection of a Read Error, i.e. it tries to accomplish the following: Move back one block; Read forward one block; Test for Read Error; if none, place buffer content into computer and proceed. (TCU 1, Uniservo 2)

a	EF	—	b
a+1	EF	—	b+1
a+2	MJ	40000	a+2
a+3	MJ	60000	a+5
a+4	MJ	u	v
a+5	EF	—	b+2
a+6	MP	10170	a+10
a+7	ER	10000	c
b	01	00014	20001
b+1	01	00002	20000
b+2	00	10000	00200

At first glance nothing seems to be wrong, but careful examination reveals that the computer will hang up continuously at the first of the 120_{10} ER's with BAR = $67e10_8$. The reason is the following:

The EF at address a is executed properly, i.e. the "Move" code is sent to TCR and moving will be started. At the same time the buffer is set to "active".

The EF at address a+1 will now be executed to the extent that the code (b+1)⟶IOB, but not further, since TCR is locked out because of the "Move" operation. The computer, however, proceeds.

The MJ 4 at a+2 finds the buffer in its "active" state. It was programmed to wait for the completion of the "Read Tape" operation, However, the computer now waits at this point, until the "Move" operation will be finished.

As soon as moving is completed, the computer program continues at a+3, where it tries to detect a Read Error. Since at this time reading has not even been started, no Read Error can be detected and the program will proceed at a+5. At the same time (b+1) which was in IOB has been sent to TCR, where it set up the "Read" operation, i.e. the buffer is "active" again.

The EF at a+5 is executed such that the code (b+2)⟶IOB. The proper selection of the buffer operation ("Read Buffer") cannot be made because of the fact that the buffer is "active".

The computer, however, continues at a+6, a+7. The first ER will, therefore, hang up, since no word entered IOB from the buffer. This hanging-up occurs at MP 1, i.e. no MP 5 can be issued! This in turn means that the buffer never can be set to "inactive", and a permanent hanging-up is accomplished!

The above program is wrong because of the fact that another tape operation was initiated at the time the buffer was still "active" with moving. The instruction at a+1 should be executed only if the buffer was found to be "inactive" i,e. an MJ 4 between a and a+1 would correct the program.

On the other hand you saw that the EF at a+1 itself did not stall the computer, not even the EF at a+5. Had we arrived at a+5 at the time when reading was finished, there would not have been any hanging-up! Only the fact that (a+5) was executed during buffer activity caused the computer to stall at the first ER.

The above example will certainly stress the fact that the programmer has to test buffer activity first and initiate an operation which involves this buffer, if and only if he found the buffer to be inactive. It will also illustrate, how careful a faulty program has to be examined in order to find the real trouble source.

III The Automatic Tape Controller

It is well known that the Automatic Tape Controller is used to stop tape movement temporarily during "Read Tape" or "Write Tape" operations. The following remarks are made to define the exact situation which causes the ATC to become effective.

a) Read Tape Free Run

During this operation the ATC becomes effective after the first (second, third, ...) block has been read. If within approximately 3 m sec. from the detection of the End of Block no EF "Read Buffer" has been executed, ATC issues a "Stop" signal to TCU which causes a temporary stop of tape movement. An EF "Read Buffer" will then cause the tape to start again.

b) Write Tape Free Run

Here the Automatic Tape Controller becomes effective after the first (second, third, ,....) block has been written on tape. If within approximately 3 m sec. from the time at which the last word has been written on tape an EF "Write Buffer" has not been given, ATC stops tape movement. An EF "Write Buffer" will then initiate the restart of the tape.

c) Consequences

Keep in mind: when you Read Tape or Write Tape, ATC will not stop the tape in front of the first block, since neither an "End of Block" signal (during reading) nor a "Last Word Write" signal from TCU was present.

Guenther F. Paprotny
September 22, 1958