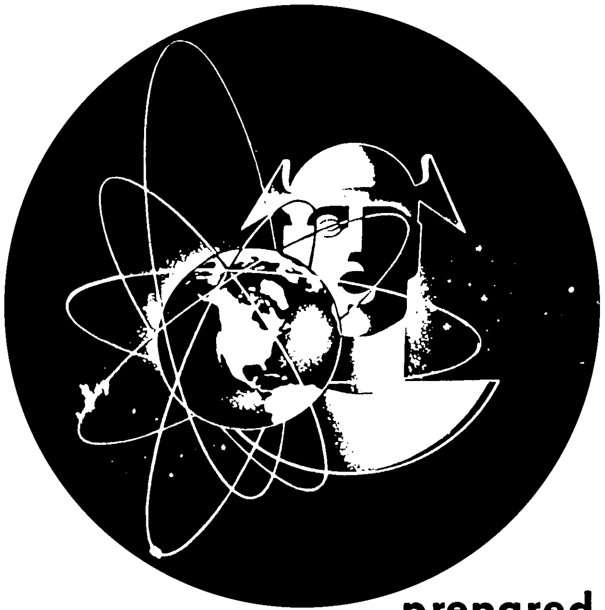
class notes

COURSE

003

ADVANCED PROGRAMMING



prepared by the Training Section,
Electronic Computer Department

Remington Rand INC

OUTLINE FOR COURSE 003

ADVANCED PROGRAMMING

The purpose of Advanced Programming Course 003 is to complete the basic description of the Univac System and describe the advanced programming techniques required for the preparation of efficient data processing routines. Each student is required to have satisfactorily completed Course 002 which is concerned with a description of the Univac central computer and basic programming.

The subject matter of Course 003 consists of four major sections, each of which is briefly described below:

1. Advanced Programming - This area of study covers the programming techniques commonly employed in efficient use of input-output buffers, problem reconstitution, item layouts, relative coding, reduction of instruction execution times, obtaining increases in available memory space, use of service routines for detection and correction of programming errors.

2. Programming for Heavy Auxiliaries - This area covers by description and illustrative problems, programming for the heavy auxiliaries of the Univac System, such as the Card-to-Tape Converter and the High-Speed Printer.

3. Central Computer Logic - A detailed description of the central computer's logical construction is the subject matter of this study area. Particular attention is given to the detailed analysis of how each instruction is executed. The checking circuits of the Univac and the method whereby the supervisory control may be used as a programmer aid in debugging and efficiently running data processing routines.

4. Thesis Problem - This area consists of workshop sessions wherein each student selects a moderately complex problem for his thesis. Each student will analyze, flow-chart, code, and debug on the Univac System (Univac #7 - Service Bureau, Remington Rand Inc., New York, New York) a problem involving from 600-800 instructions. This thesis work is designed to provide further experience in all phases of computer programming and is conducted under the supervision of the instruction staff assigned to his course.

A topical outline is presented below. The amount of time spent on each subject may vary somewhat among the different instructors and classes. Items 1 and 2 above are designated under A in the outline, 3 above is topic B, and 4 is topic C.

WEEK 1

MONDAY: A. ANALYSIS OF COMMON CODING ERRORS
1. Types of errors and their manifestations
2. Errors due to neglected instruction characteristics
3. Improper overflow counts
4. Errors in flow chart logic

B. REVIEW AND INTRODUCTION TO CIRCUITRY
1. Memory and word structure
2. Binary and XS-3 notation
3. Four-stage cycle of operation
4. Description of logical circuit elements

C. THESIS PROBLEM

TUESDAY: A. DESCRIPTION OF BIOR RELATIVE CODING

B. TIMING AND SYNCHRONIZATION IN UNIVAC
1. Minor cycle
2. Cycling unit
3. p and t notation
4. Synchronization of memory and registers
5. Maintenance of synchronization in memory to register, register to register, and register to memory transfers.

C. THESIS PROBLEM

WEDNESDAY: A. BIOR RELATIVE CODING (CON'T)

B. FOUR-STAGE CYCLE
1. Description of Chart U 102 and EBU-100
2. α time
3. HSB O-E checker, differences between positive and negative type checkers.

C. THESIS PROBLEM

THURSDAY: A. BIOR RELATIVE CODING (CONCLUDED)

B. FOUR-STAGE CYCLE (CON'T)
1. β time
2. Channel selection circuits
3. Time selection circuits

C. THESIS PROBLEM

FRIDAY: A. SPECIAL INPUT-OUTPUT TECHNIQUES
1. Preselector
2. 2-way automatic
3. Reversal
4. 3-way automatic

B. FOUR-STAGE CYCLE (CON'T)
1. γ and δ time
2. Start circuit

C. THESIS PROBLEM

WEEK 2

MONDAY:

A. INPUT-OUTPUT (CONCLUDED)
   1. Efficient transfers for items a sub-multiple of 60
   2. Efficient transfers for items not a sub-multiple of 60
   3. Variable length items

B. TRANSFER ORDERS - TYPE 1
   1. B, H, K as typical of 1 word transfers
   2. Error indications MDS and DSS

C. THESIS PROBLEM

TUESDAY:

A. IDENTIFICATION BLOCKS AND RERUNS
   1. Standard data tape format
   2. Generalized two-way merge

B. TRANSFER ORDERS - TYPE 1 (CON'T)
   1. V, W, Y, Z as typical of multi-word transfers
   2. Interchange in V-W transfers
   3. R order

C. THESIS PROBLEM

WEDNESDAY:

A. IDENTIFICATION BLOCKS AND RERUNS (CON'T)
   1. Generalized two-way merge (concluded)
   2. Rerun procedure for 1 output

B. TRANSFER ORDERS - TYPE 2
   1. .n, On as typical of shifts
   2. Improper shifts causing a stall
   3. Extract order

C. THESIS PROBLEM

THURSDAY:

A. IDENTIFICATION BLOCKS AND RERUNS (CONCLUDED)
   1. Reruns with multiple output

B. LOGICAL CHOICE
   1. Skip and U instructions

C. THESIS PROBLEM

FRIDAY:

A. REDUCING PROBLEM RUNNING TIME
   1. Minimum latency tables

B. LOGICAL CHOICE (CONCLUDED)
   1. Q and T instructions

C. THESIS PROBLEM

WEEK 3

MONDAY:

A. REDUCING PROBLEM RUNNING TIME (CON'T)
   1. Minimum latency coding

B. INTERRUPTED OPERATION
   1. Qn, Tn, 9 and , instructions
   2. Interrupted operation switch

C. THESIS PROBLEM

TUESDAY:

A. REDUCING PROBLEM RUNNING TIME (CONCLUDED)
   1. R-U counters
   2. Straight line coding

B. ARITHMETIC CIRCUITS
   1. A and S instructions

C. THESIS PROBLEM

WEDNESDAY:

A. CONSERVATION OF MEMORY SPACE
   1. Overlays
   2. Sub-dividing runs

B. ARITHMETIC CIRCUITS (CON'T)
   1. Overflow
   2. A- and S- instructions
   3. 12-place addition in β time

C. THESIS PROBLEM

THURSDAY:

A. COLLATION METHOD OF SORTING

B. ARITHMETIC CIRCUITS (CON'T)
   1. Multiplication as repeated addition
   2. P instruction

C. THESIS PROBLEM

FRIDAY:

A. COLLATION METHOD OF SORTING (CONCLUDED)

B. ARITHMETIC CIRCUITS (CON'T)
   1. P instruction (concluded)
   2. M and N instruction
   3. Action of MQC with multiplier digits not numbers

C. THESIS PROBLEM

WEEK 4

MONDAY:

A. FUNCTION TABLE SORTING

B. ARITHMETIC CIRCUITS (CON'T)
   1. Theory of non-restoring division
   2. D instruction

C. THESIS PROBLEM

TUESDAY:

A. SERVICE ROUTINES
   1. Classifications
   2. Locator
   3. Mark VIII

B. ARITHMETIC CIRCUITS (CON'T)
   1. D instruction (concluded)
   2. D- instruction

C. THESIS PROBLEM

WEDNESDAY:

A. SERVICE ROUTINES (CON'T)
   1. Mark VIII (con't)

B. ARITHMETIC CIRCUITS (CONCLUDED)
   1. Periodic memory check
   2. Memory clear

C. THESIS PROBLEM

THURSDAY:

A. SERVICE ROUTINES (CON'T)
   1. Mark VIII (concluded)
   2. AC-3
   3. AC-4

B. INPUT-OUTPUT CIRCUITS
   1. Input-output problem
   2. Uniservo principles
   3. Additional logical elements (thyratrons, relays, autosyns, transformers, motors)

C. THESIS PROBLEM

FRIDAY:

A. SERVICE ROUTINES (CON'T)
   1. AM-2 (line merge)
   2. Herb I

B. INPUT-OUTPUT CIRCUITS (CON'T)
   1. Step-down buffering
   2. 5n instruction

C. THESIS PROBLEM

WEEK 5

MONDAY: A. SERVICE ROUTINES (CON'T)
1. Code edit
2. Analyzer

B. INPUT-OUTPUT CIRCUITS (CON'T)
1. 5n instruction (con't)

C. THESIS PROBLEM

TUESDAY: A. SERVICE ROUTINES (CON'T)
1. Auto-monitors

B. INPUT-OUTPUT CIRCUITS (CON'T)
1. 5n instruction (concluded)
2. Low-density recording

C. THESIS PROBLEM

WEDNESDAY: A. SERVICE ROUTINES (CONCLUDED)
1. Follower
2. Code search
3. Word search

B. INPUT-OUTPUT CIRCUITS (CON'T)
1. Step-up buffering
2. 1n instruction

C. THESIS PROBLEM

THURSDAY: A. HEAVY AUXILIARIES
1. Card-to-Tape Converter

B. INPUT-OUTPUT CIRCUITS (CON'T)
1. 1n instruction (con't)

C. THESIS PROBLEM

FRIDAY: A. HEAVY AUXILIARIES
1. Card-to-Tape Converter (concluded)
2. Tape-to-Card Converter

B. INPUT-OUTPUT CIRCUITS
1. 1n instruction (concluded)
2. 2n instruction

C. THESIS PROBLEM

WEEK 6

MONDAY: A. HEAVY AUXILIARIES (CON'T)
1. High-speed printer

B. INPUT-OUTPUT CIRCUITS (CON'T)
1. 30 instruction
2. 3n instruction

C. THESIS PROBLEM
1. Unityping, code-edit and analysis

TUESDAY: A. HEAVY AUXILIARIES (CON'T)
1. High-speed printer (con't)

B. INPUT-OUTPUT CIRCUITS (CON'T)
1. 4n instruction
2. 6n and 8n instructions

C. THESIS PROBLEM
1. Proof-reading
2. Computer testing

WEDNESDAY: A. HEAVY AUXILIARIES (CON'T)
1. High-speed printer (con't)

B. INPUT-OUTPUT CIRCUITS (CONCLUDED)
1. 50 instruction
2. 10 instruction

C. THESIS PROBLEM
1. Computer testing

THURSDAY: A. HEAVY AUXILIARIES (CONCLUDED)
1. High-speed printer (concluded)

B. SUPERVISORY CONTROL OPERATIONS
1. SCI FILL
2. EMPTY
3. Clear C
4. General clear

C. THESIS PROBLEM
1. Computer testing

FRIDAY: A. -

B. SUPERVISORY CONTROL OPERATIONS (CONCLUDED)
1. Starting and stopping Univac
2. Simple error analysis

C. THESIS PROBLEM
1. Post Mortem

ANALYSIS OF COMMON CODING ERRORS

Practical problems prepared for digital computers involve upwards of several thousand instructions. While the UNIVAC computer is self-checking, it basically detects errors in performing the indicated operation and does not judge whether that operation, or series of operations, is correct in the context of the complete problem. Errors may occur in these categories:

1. Improper execution of an instruction due to mechanical or electronic failure of the computer.
2. Failure of the operator to properly initiate the problem or in performing rerun procedures.
3. Failure of the programmer to provide coding to carry out the assigned task.

As mentioned in the first paragraph, errors of type 1 are checked in UNIVAC by duplication of circuits or by redundant codes (odd-even check, etc). We need not consider them further.

Errors of type 2 are not inherently checked by the computer, and can cause disastrous consequences if allowed to perpetuate themselves. To a large extent these errors may be avoided by requiring the program to check operator intervention. Thus, the rewinding of multiple output or input tapes with 8n orders insures that these tapes will be removed before those servos are again called for further operations. Use of rings in those reels that contain permanent information during a run prevents the accidental destruction of information by Supervisory Control action.

Positive checks can be incorporated in the program to insure that the correct tapes are mounted, and in their correct order. This is the purpose of identification blocks and sequence checks. Since Supervisory Control operations are infrequent and the operator often unfamilar with the problem and its coding, programs should be prepared in such a way as to require little operator intervention; and when this is necessary, it should be of extreme simplicity. This is especially true in setup and rerun procedures.

Errors of type 3 are the main concern of this section. The first concern is how do we recognize that a coding error exists? Assuming that a normal procedure has been followed, the prepared coding will have been desk-checked (i.e., reviewed by one or more different coders), Unityped, and the Unityped tape printed and checked against the original copy. Thus, we can assume that the information on tape agrees with the coding sheets and that a preliminary check has been performed.

A coding error may manifest itself in one or more of the following ways after the program has been tried on the computer, or by none of them:

A. Overflow where it has not been anticipated (i.e., on A-, S-, X-, or D- orders)

B. Closed loops producing a characteristic sound from the HSB speaker.

C. Adder alphabetic errors.

D. Improper sequence of tape movements observable by operator or programmer. This includes doing 3n orders when rI is empty. In orders, when rI is filled followed by 3n, reading a rewound tape backwards, failure to detect end of tape information, and others.

E. Improper commands causing computer to stall.

F. Output results of test data not agreeing with pre-calculated results.

One should not always expect coding errors to be easily recognized, and often the manifestation of an error may occur in a section of the problem remote from where the error itself was made or even from the nature of the error. For example, the following error was abstracted from a mathematical subroutine designed to calculate log $A \cdot 10^a = B \cdot 10^b$. Where is the error and how was it recnized?

```
000  11  000
                 30  060     instructions ⟶ memory
001  81  000
                  -
 .
 .
 .
____________________________________________________________________________
006  K   000                 a ⟶ rL
                 F   119     +.03000   000000 ⟶ rF
007  B   104                 +.18835   453000 ⟶ rA
                 T   036     if a <.18835  453000 transfer control to
                             line 036.
____________________________________________________________________________
008  03  000                 SL3 (rA) = + .35453  000000 ⟶ rA
                 T   037     if a <.35453  000000 transfer control to
                             line 037.
____________________________________________________________________________
009  03  000                 SL3 (rA) = + .53000  000000 ⟶ rA
                 T   038     if a <.53000  000000 transfer control to
                             line 038.
____________________________________________________________________________
 .
 .
 .
026  K   000                 zero ⟶ rL
                 F   118     E = 010000  000000 ⟶ rF
____________________________________________________________________________
```

027	E	100			m.s.d. of $(\log_{10}x)/1000 \longrightarrow rA$
			Q	031	if m.s.d. = 0 transfer control to line 031.
028	B	125			$\pm$ .ee 000000 $\longrightarrow rA$
			.9	000	SR_9 (rA) = 0.00000 000$\pm$ee $\longrightarrow rA$
029	F	102			E_1 = 111111 111000 $\longrightarrow rF$
			E	100	$\log_{10}x \longrightarrow rA$
030	00	000			Skip
			U	(R+1)	Unconditional transfer of control to main routine.
031	S	118			-.10000 000000 $\longrightarrow rA$
			-1	000	SR_1 (rA) = -.01000 000000 $\longrightarrow rA$
032	A	125			+.02000 000000 $\longrightarrow rA$
			C	125	+.02000 000000 $\longrightarrow$ 125, zero $\longrightarrow rA$
033	K	000			zero $\longrightarrow rL$
			B	100	$(\log_{10}x)/1000 \longrightarrow rA$
034	01	000			SL_1 (rA) = $(\log_{10}x)/100 \longrightarrow rA$
			Q	030	if $\log_{10}x/100$ = zero, transfer control to line 030.
035	C	100			$(\log_{10}x)/100 \longrightarrow$ 100, zero $\longrightarrow rA$
			U	027	Unconditional transfer of control to line 027.
036	B	112			
			U	039	$-\log_{10}k_1 + k_1 \longrightarrow rA$
					Unconditional transfer of control to line 039.
037	B	113			$-\log_{10}k_2 + k_2 \longrightarrow rA$
			U	039	Unconditional transfer of control to line 039.
038	B	114			$-\log_{10}k_3 + k_3 \longrightarrow rA$
			U	039	Unconditional transfer of control to line 039.
039	H	100			$-\log_{10}k + k \longrightarrow$ 100, rA
			-3	000	SR_3 (rA) = $-(\log_{10}k)/1000 \longrightarrow rA$
040	A	101			$(b - \log_{10}k)/1000 \longrightarrow rA$
			C	101	$(b - \log_{10}k)/1000 \longrightarrow$ 101
041	B	100			$\log_{10}k + k \longrightarrow rA$
			08	000	SL_8 (rA) = $-k/10 \longrightarrow rA$

```
042   C    100                     -k/10 ⟶ 100
                  N     100        k_i a/10 ⟶ rA
043   01   000                     SL_1 (rA) = k_i a ⟶ rA
                  H     100        k_i a ⟶ 100, rA
044   K    000                     k_i a ⟶ rL
                  G     125        +.03000  000000 ⟶ 125
045   00   000
                  U     010        To continue computation
```

The error made itself known by a second attempt to read the instruction tape. Since this tape had been rewound with interlock, the computer stalled. How was the coding error located? The reasoning went something like this: The control was somehow transferred to line 000. Since this was not intentional, it might have been due to an unexpected overflow. As the problem analysis seemed to be correct, at least as an initial assumption, the actual calculation of the log in steps 010 through 025 probably did not give the overflow. The next point examined was the normalization subroutine designed to calculate the zeros lying to the left of the most significant digit. The subroutine is in lines 026 to 035. The add order in line 032 was suspect. This order adds 1 to a counter for each zero found to the left of MSD in $\log_{10} x$. The largest number of zeros would, of course, be 11. A check showed that (118) was indeed correct (-10 000 000 000); thus, (125) must not have been set correctly. On line 006 is an F 119 instruction placing the constant 003 000 000 000 in rF, and the G 125 on line 044 is to place this constant in cell 125 thus setting (125) to its proper initial value. The odd splitting of the F and G was due to the original omission on the coder's part of the resetting of 125. After recognizing the omission, he went back to make the transfer and filled in the F-G in whatever skips were available. The error, of course, is that the multiplication of line 042 destroys the 003 000 000 000 placed in rF, and places in its stead 3|rL . Fortunately, this number caused an overflow on line 032. The error was made on line 044 but manifested itself by a read instruction on line 000.

Many coding errors occur because a programmer has ignored or forgotten the peculiarities of certain UNIVAC orders. This subroutine is designed to transfer 30 two-word items from memory locations 301 to 360 inclusive, to memory locations 451 to 510 inclusive. What is the error?

```
000   [V    301
                  W     451]
001   B     000
                  A     005
002   L     006
                  Q     004
003   C     000
                  U     000
004   00    000
                  90    000
005   000   002
                  000   002
006   V00   361
                  W00   511
```

The obvious error, of course, is that a consecutive series of 2-word items starting at an odd memory location cannot be transferred by the V register. In spite of the fact that the routine avoids the interchange in rV by starting from an odd location and transferring to an odd location, the item in 309, 310 splits across two channels and so 309 and 300 go into 459 and 450 rather than 309, 310 into 459, 460.

The following error may be more difficult to recognize, and its variants are among the most frequent errors in editing and mathematical routines: Memory locations 900-939 contain 40 quantities Y_i ($i = 0, 1, 2, \ldots, 39$). A quantity $Z_i = .0065i$ is computed and added to Y_i (i.e., $Y_i + Z_i \longrightarrow Y_i$).

```
000   L     008
                   M     009
001  [A-    939
                   C     939 ]
002   B     001
                   S     010
003   C     001
                   K     000
004   B     009
                   Q     007
005   S     011
                   C     009
006   00    000
                   U     000
007   00    000
                   90    000
008   000   650
                   000   000
009  [039   000
                   000   000 ]
010   000   001
                   000   001
011   001   000
                   000   000
```

The error in this subroutine lies in reducing the instruction word A-0 939 C 00939.

It would seem to be perfectly permissible to subtract one's from the memory location digits successively reducing the instructions to

```
A-0 938 C00 938
A-0 937 C00 937
       .
       .
A-0 900 C00 900
```

But note that when the adder performs algebraic subtraction it complements the smaller (in absolute value) and adds. Thus

```
A-0 939 C00 939
-00 001 000 001
_______________
A09 062 C99 062
```

The presence of the minus sign following the "A" forces complementation on A-0 939 C00 939. A method of doing this problem involves adding the complement of 1:

```
A-Z 939 C00 939
000 999 000 999
_______________
A-Z 938 C01 938
```

The "Z" prevents a carry from adding to the minus sign. The correct program is as follows:

```
000   L     008
                  M    009
001   A-Z   939
                  C    939]
002   B     001
                  A    010
003   C     001
                  K    000
004   B     009
                  Q    007
005   S     011
                  C    009
006   00    000
                  U    000
007   00    000
                  90   000
008   000   650
                  000  000
009   039   000
                  000  000]
010   000   999
                  000  999
011   001   000
                  000  000
```

Use of overflow for control purposes has produced great simplification in coding and decreased the running time of routines. It has also brought a certain headache with it -- improper counting. The example shown below is a quite common error among beginning programmers. This fragment is from a routine that is processing 10-word items. The item counter is increased after transferring the new item to working storage. Unfortunately, the counter is initially set too high. When the last item is transferred, the counter overflows causing the next block of 6 items to be transferred into the current block. Item 6 is thus lost in each block.

```
050 [Y5   (610)                  A_i ⟶ WS
               Z     600]
051  B    050
               A*    055         i +1 ⟶ i
052  C    050
               U     ---         Transfer control to process A_1^o
053  B    056                    Return here if overflow from 051
               C     050         1 ⟶ i
054  31   600                    T_1 ⟶ A
               U     ---         Transfer control to process A_i
055  010  010
               000   000
056  Y5   610
               Z00   600
```

There is a simple scheme for correctly determining the initial value of the counter. Suppose we use the above example. After having processed the last item of the **block**, we will transfer a useless 10-word item into working **storage and augment the counter, which** should then overflow to indicate that a new block is to be read. The counter reading just before overflow should then be: Y9 660 Z 600. Initially, the counter must be Yx 610 Z 600, set to transfer the second item into working storage. If we add once to x each time we add 1 to the channel selection digits of the Y order, then 9-x 6-1 = 5; therefore, x = 4 is the initial setting. This method can be easily extended to any size item, for the case of 2-word items V99 660 W 600 is the reading after processing the 30th or last item. If we use 002 002 000 000 as the augmenting constant, 99-x = 58 and x = 41. The initial setting being V41 602 W 600.

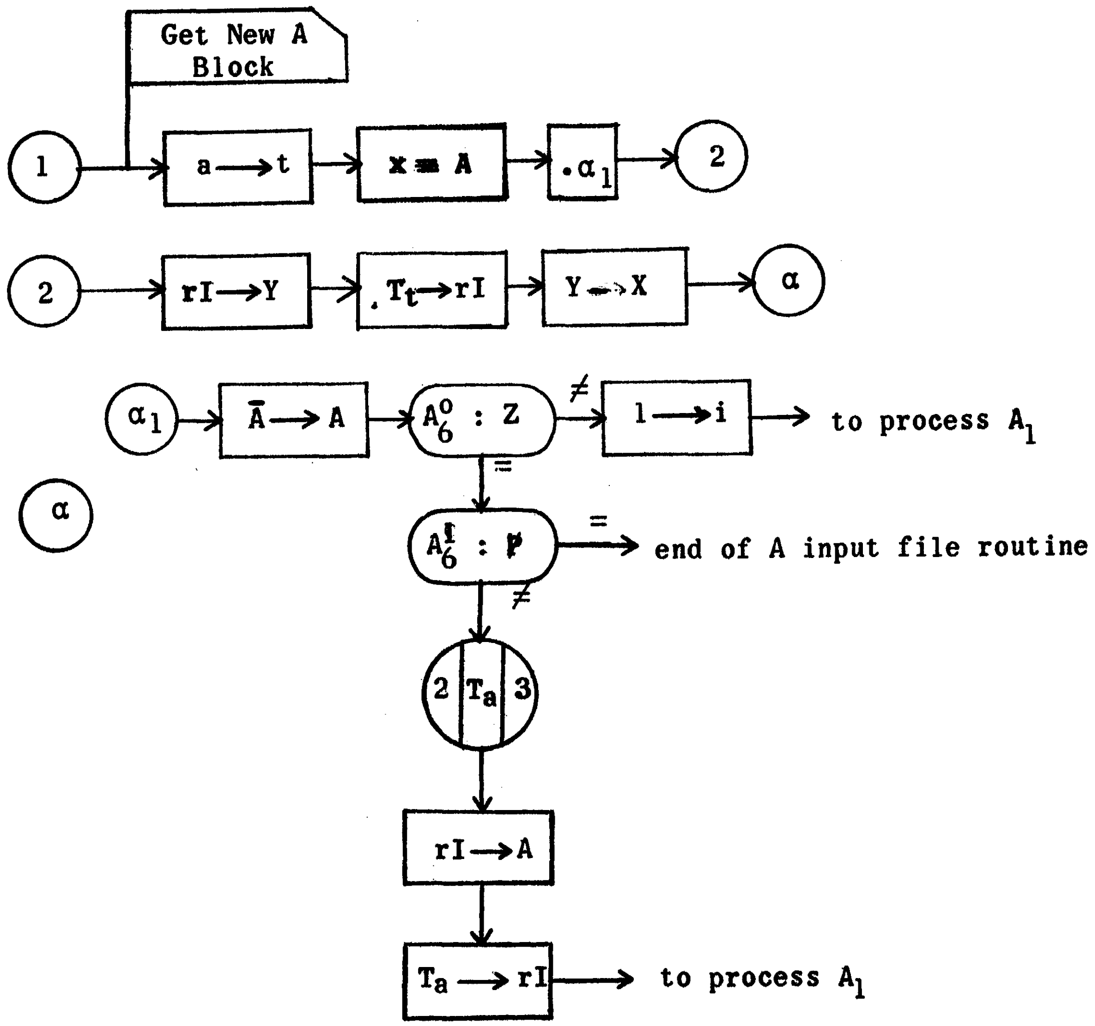
What is the possible indication of the coding error in the following routine? It is supposed to count the number of nonsignificant zeros of the number X. (Nonsignificant zeros would be those to the left of the 6 in the example: 000 006 350 010 except the sign).

```
000  K    000
               K    000
001  F    008
               E    009
002  00   000
               Q    004
003  00   000
               U    ---
004  B    010
               A    011
005  C    010
               B    008
006  -1   000
               C    008
007  00   000
               U    001
008 [010  000
               000  000]
```

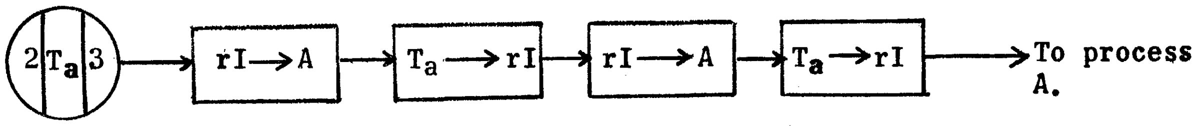
```
009       Quantity X

010 [000    000
                    000   000 ]
011  000    000
                    000   001
```

In this example the extractor is shifted right to examine each column successively. If the quantity X is zero, however, the routine will never transfer out and becomes a closed loop.

There is yet another class of errors belonging to section 3. These are errors of program logic and can be detected by an analysis of the flow chart. The illustration is that of a two-way merge with multiple data reels for each input. The standard sentinel convention is assumed for each input. The two standby block input procedure is used.



The error in this example is in the two steps following the servo switch symbol. As two sentinel blocks are at the end of each input tape, rI must have contained the second sentinel block and thus not a valid set of A items. The correct version would be:



STUDENT EXERCISES.

Find the coding or logical errors in the following problems:

This is intended to be an ending routine that will read into the memory the instructions for the next run. Assume the reading head on tape #1 is in the correct position and rI is erased.

```
049 ......
                ......
050 [B  940
                L  056 ]
051  00 000
                Q  053          ending test
052  00 000
                U  XXX          continue with Run 1
-----------------------
053  11 000
                30 000
054  50 055
                U  000          go to Run 2
-----------------------
055   Begin
                 Run  2
056  ZZZZZZ
                ZZZZZZ          Sentinels
057  ......
                ......
```

This is intended to be an input routine that will park the current 2-word input item in a working storage and read in a new input block when the current block is exhausted. It is to be entered from the main routine by R 305 U 300. Assume generalized overflow control with an increment of 000000 000001. Also rI contains the next input block from tape 2.

```
300   V60   925
                  W  925
301   B     300
                  A* 306
302   C     300
                  U  305
-------------------------
```

```
303   32    925
                  B     307
304   C     300
                  00    000
---------------------------
305   [input
                  return]
306   002   002
                  000000
307   V60   925
                  W     925
```

This output routine is supposed to transfer a 10-word item from an output working storage to the current position in the output block; then write the output block on tape after it is filled with 6 items. Assume generalized overflow control with an increment of 000000 000001. Output operation is to be accomplished in the main routine by R 430 U 426. Working storage is 600 ... 609.

```
425   000000
                  010   010
---------------------------
426   B     430
                  A*    425
427   00    000
                  U     429
---------------------------
428   72    940
                  B     432
429   H     430
                  00    000
430   [Y    600
                  Z40   940]
431   [output
                  return]
---------------------------
432   Y     600
                  Z40   940     Reset constant.
```

A SURVEY OF SPECIAL INPUT-OUTPUT TECHNIQUES

1. Additional methods for efficient use of the input buffer.

Chapter 10, Section 8, of the Univac Programming Manual describes a simple yet completely general method of keeping rI filled for multiple input tape problems. In this method each input tape requires the reserving of two blocks within the memory and the execution of six Y-Z transfers to put the "standby" block into working storage position. This paper assumes the reader is completely familiar with this method as described in the manual, which is hereinafter referred to as the "two-standby" technique.

Because of its simplicity and complete generality, the two-standby technique was described in the elementary programming manual. However, there are other techniques, Preselector, Reversal, and Automatic method which are of greater efficiency and in more general use.

The Preselector Method

This method is simple in concept and extremely conservative of memory requirements and time. It is the preferred method for all problems where the selection of the next item for processing is based solely upon the relative magnitudes of the sequencing fields within the items. The principle will be illustrated by the following example:

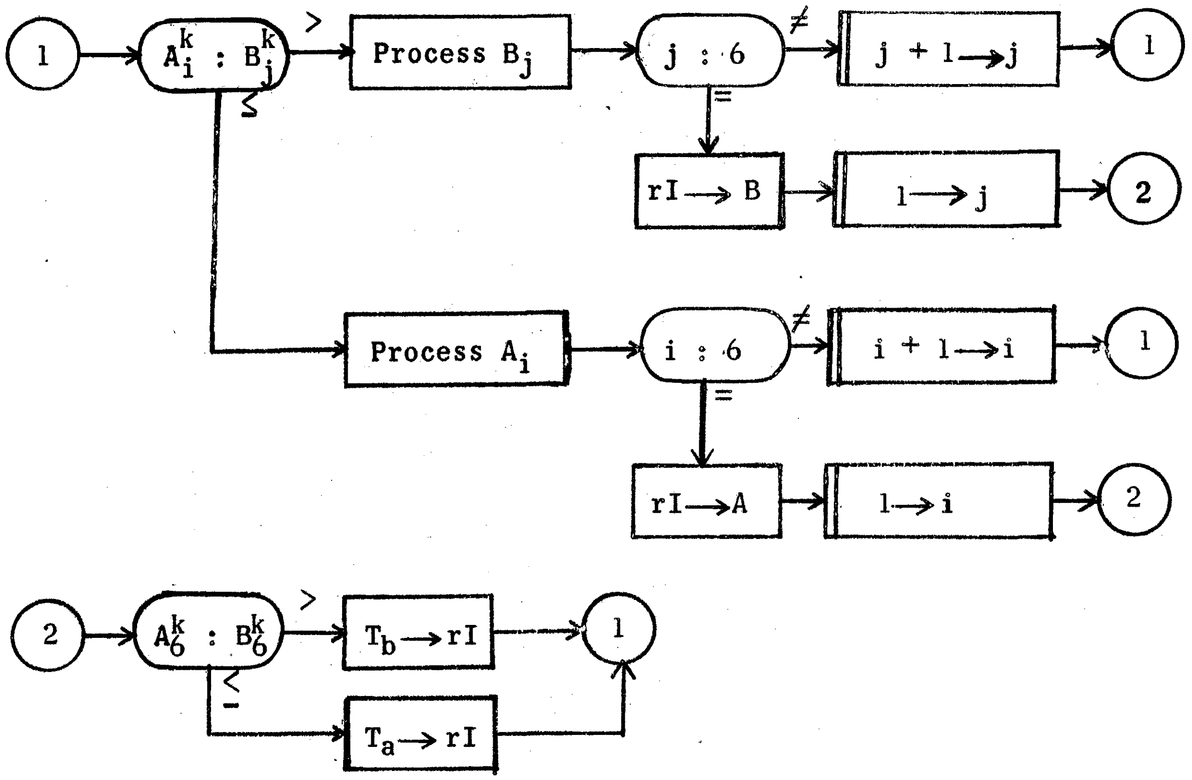
Suppose we have two files of input items to a processing run. Let us call the files A and B. Suppose further, that these are ten-word items, six to a block: that the items within each file are arranged in ascending sequence by a key within each item and, further, that the processing is always done on the item from either file which has the smallest (in magnitude) key.

At the start of our problem we will have the first A and B block in the memory and are concerned with what tape to read into rI. Let us write down the keys of the twelve items as they might appear:

A Item Keys	B Item Keys
1136	1000
1137	1010
2100	1011
2501	2050
2502	2161
2600	2163

In comparing the keys of the last A and B item of the block we note that the A item has a smaller key than B. Since the smallest item of A or B is processed first by our routine, it is evident that A_6 will be used before B_6. But since the A and B items are in ascending sequence within each file, we then know that the A block will be exhausted before B; therefore, we should order the next A block now. That is, fill rI with A. We can then begin processing A_1, A_2, ---, B_1, B_2,--- with the full confidence that when A_6 has been processed the new A block is in rI. If B_6 had been smallest we would fill rI with B items. Of course, after rI is transferred to the appropriate empty block, the last items must be compared again to determine what now should fill rI.

The method is simplicity itself as shown in the following flow chart:



The reader will note in the flow chart that if the keys of the current A and B items are identical, an A item is selected for processing first. Then, if A_6 and B_6 have like keys, and A block must be read into rI since it is the A block which is exhausted first. This can be a subtle point in some routines and care should be taken in the treatment of the case of equal keys.

Since no standby blocks nor block transfer instructions are required, the preselector is the best method for filling rI when it is applicable. There is, however, a class of problems with complex processing rules that do not admit the preselector method in these problems because the order of reading tape is either not dependent solely on the original contents of the tapes, or the order of processing the items cannot be easily determined as simply the smallest (or largest) item key first. In these cases one of the standby block methods, which are completely general, must be used.

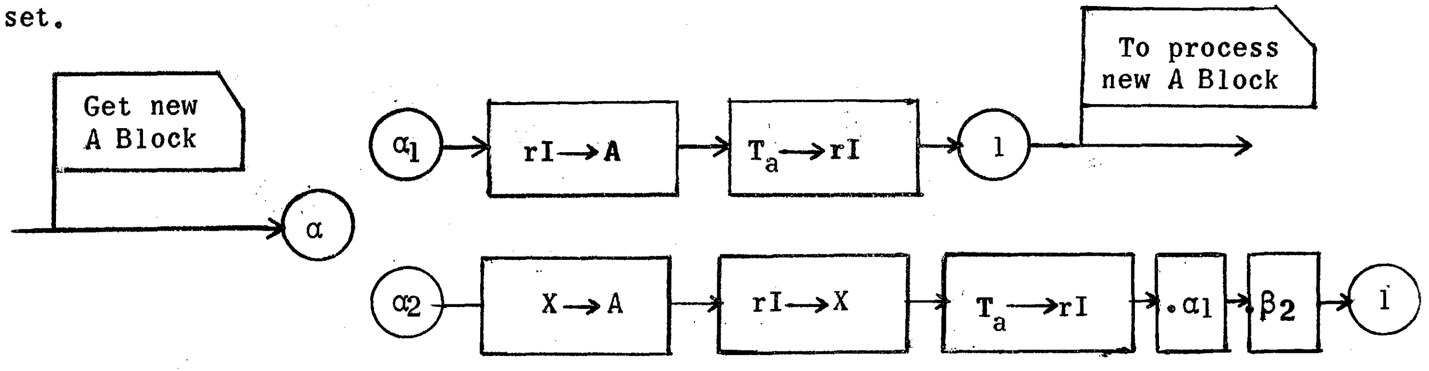
The Automatic Method For Two-way Input

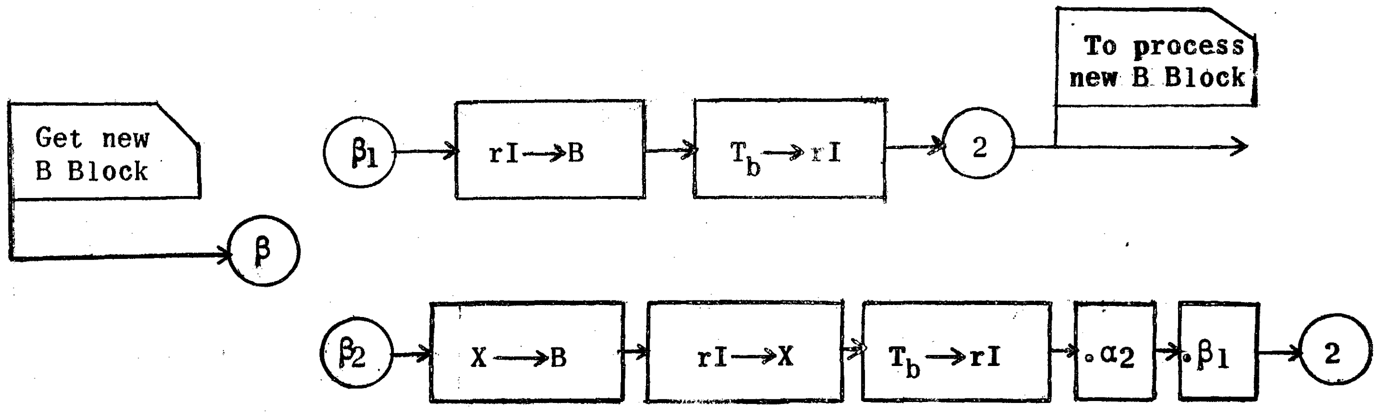
A study of the two-standby block method previously mentioned will make evident to the reader that one of the two standby blocks is always empty.

The Automatic Method uses only one standby block, the other block in rI is considered as the second. This method is thus somewhat more economical of memory space than the two-standby block method. The following notation is used in the flow chart and coding:

T_a	an input file on UNISERVO a
A	a block from tape a
rI	register I
X	designation of a standby block
$T_a \rightarrow rI$	a block from tape a is read into rI

Initially, the first block from T_a is read into A and the first block of T_b is read into B. The standby block X is filled with the second block from T_a and rI is filled with the second block from T_b. Variable connectors α_2 and β_1 are set.





In coding the automatic method the input blocks assigned were:

A -- 940 - 999

B -- 880 - 939

X -- 820 - 879

If a new A block is needed we go to line 100, and for a new B block to line 109.

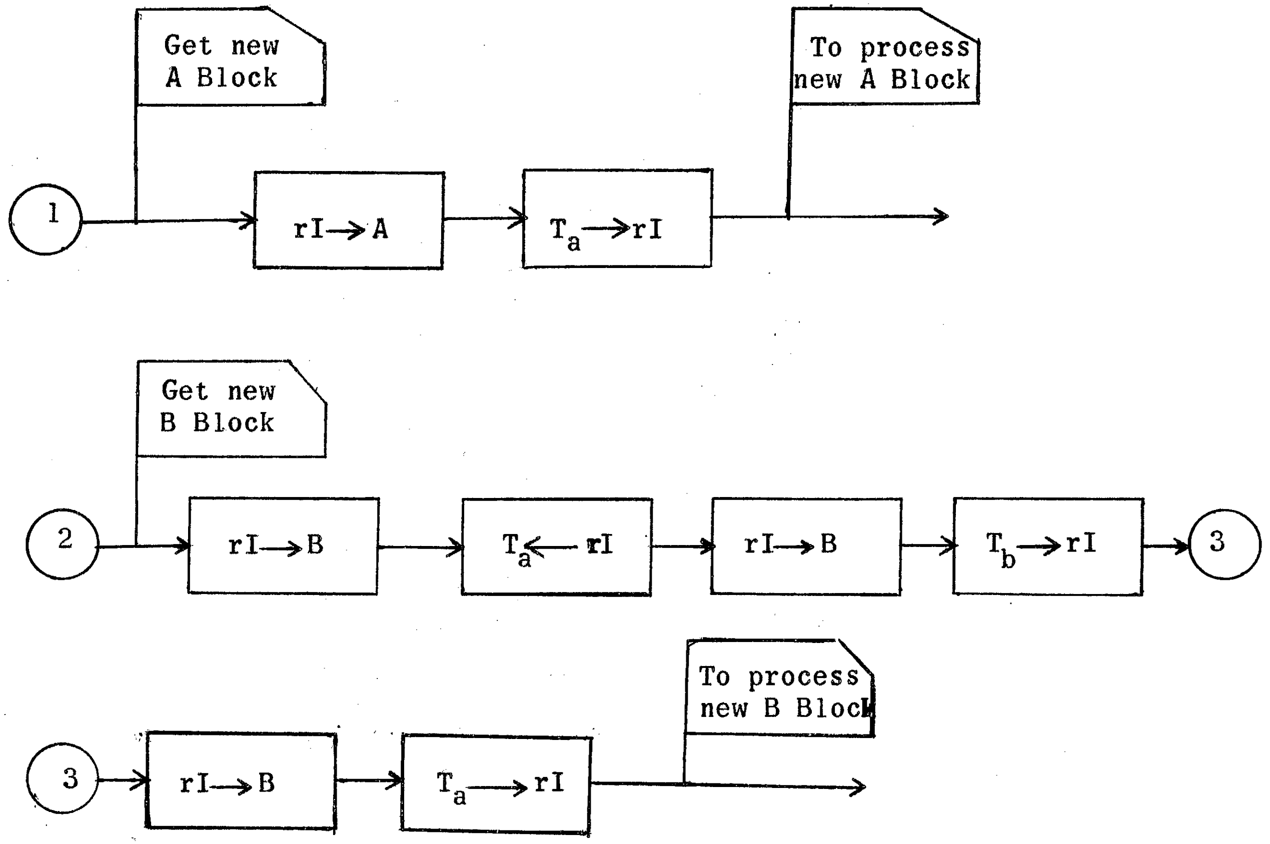
```
100  [Y    820
                Z  940]     Variable connector α  (α2 shown)
101   Y    830
                Z  950  \
102   Y    840           |
                Z  960   |
103   Y    850           }  x→A
                Z  970   |
104   Y    860           |
                Z  980   |
105   Y    870           |
                Z  990  /
106   3a   820              rI→X, Ta→rI
                B  118  }
107   C    100          }  •α1
                B  121  }
108   C    109          }  •β2
                U  ①
-------------------------------------------------------------
109  [3b   880
                U  ②]      Variable connector β  (β1 shown)
110   Y    830
                Z  890  \
111   Y    840           |
                Z  900   |
112   Y    850           }  X→B
                Z  910   |
113   Y    860           |
                Z  920   |
114   Y    870           |
                Z  930  /
115   3b   820              rI→X, Tb→rI
                B  119
116   C    100              •α2
                B  120
117   C    109              •β1
                U  ②
-------------------------------------------------------------
118   3a   940
                U  ①
119   Y    820
                Z  940
```

```
120   3b   880
                  U  (2)
121   Y    820
                  Z  880
```

The automatic method requires 22 words for instructions and constants, plus 180 words for the A, B, and X input blocks. The obtaining of the next A or B block will require either 3785 μs if the block is in rI, or 13,365 μs if the block is in X. If we assume a completely random reading sequence, then the average will be 8575 μs. Compare this to the two-standby block method described in Chapter 10, Section 8, UNIVAC PROGRAMMING MANUAL which requires 25 words of instructions and constants plus 240 words for the A and B blocks and their standbys. The time to obtain the next block is 15,395 μs. Obviously, the automatic method is superior to the method described in the programming manual.

Reversal Method for Two-way Input

The reversal method is feasible for those cases where the volumes of the two inputs are known to be very dissimilar. Assume the A input on tape a is very much larger than the B input on tape b. The initial setup is to fill an A block and a B block with the first blocks from their respective tapes, placing the second A block in rI. The following flow chart indicates the method of obtaining the next A and B blocks. (The symbol $T_a \longleftarrow rI$ is a shorthand notation for the statement "fill rI with a block from tape a, reading this tape backwards".)



In coding the reversal method, the A block is assumed to be in 940-999, the B block in 880-939. If a new A block is needed, we go to line 100; and if a new B block is needed, we go to line 101.

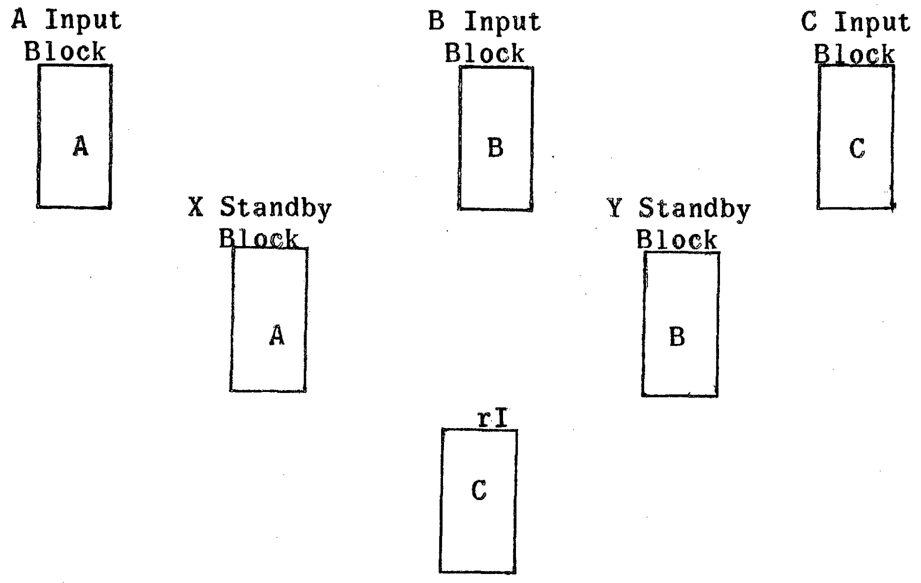
100	3a	940			$rI \rightarrow A$, $T_a \rightarrow rI$
			U	--	To process A block
101	4a	880			$rI \rightarrow B$, $T_a \leftarrow rI$
			3b	880	$rI \rightarrow B$, $T_b \rightarrow rI$
102	3a	880			$rI \rightarrow B$, $T_a \rightarrow rI$
			U	--	To process B block

The coding is small; 3 words, and an additional 120 words for the A and B block storage. The time to get the next A block is 3780 μs while the time to get the next B block is 1.53 seconds due to the tape reversals necessary to reposition T_a. A simple calculation shows that if the ratio of the size of the two files is $\frac{A}{B} \geq 130$, the reversal method is faster than the two-standby block method and requires less space than the automatic. Although the automatic method is always faster than the reversal for a very small B file, the few extra seconds required by the reversal method may be tolerable to gain the extra 79 memory cells.

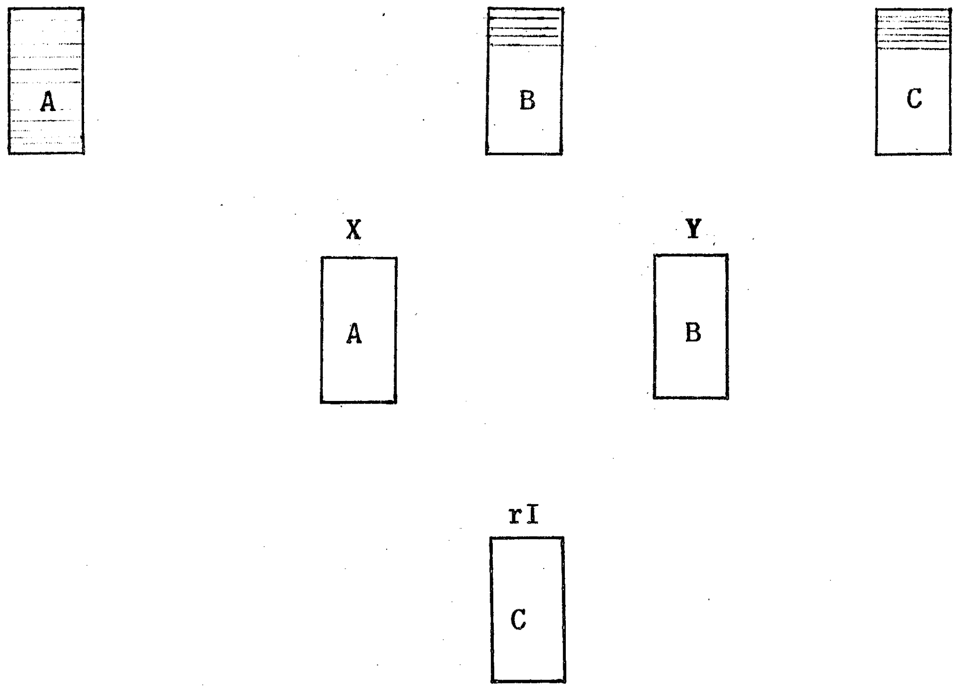
The Automatic Method for Three-way Input

This method is an extension of the automatic method described for two-way input. and uses two standby blocks in the memory with rI considered as the third. The following brief annalysis will show that the information in the standby blocks varies throughout the process.

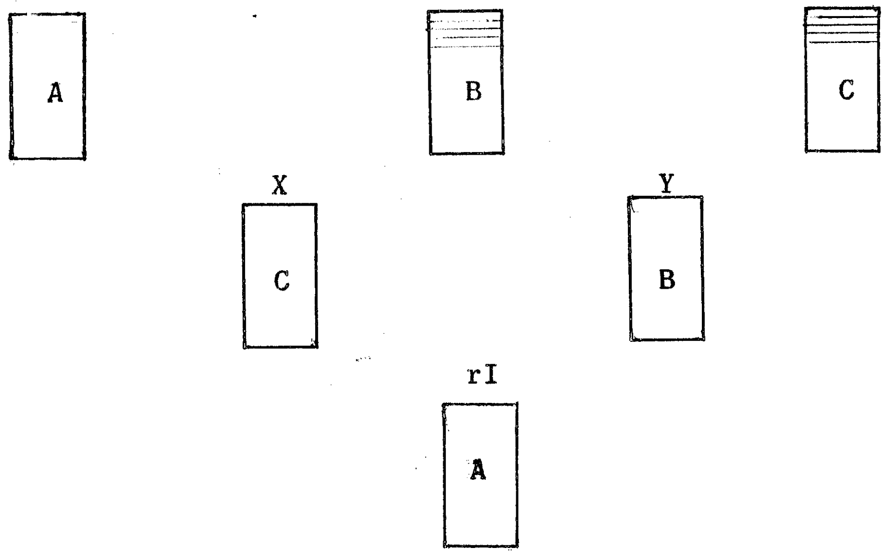
Assume that initially, the A, B, and C input blocks are filled from tapes a, b, and c, respectively, and that the two standby blocks labeled X and Y are filled with the second blocks from tapes a and b, respectively. The second C block is in rI:



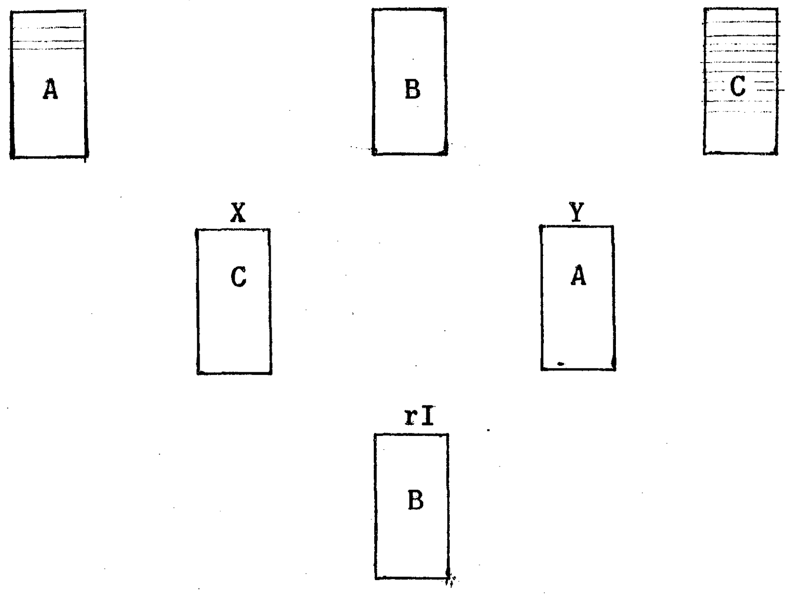
Suppose the A input block is exhausted first: (A shaded block indicates the portion of that block processed)



the contents of X are A items, so they are transferred to the now empty A input block. rI is transferred to X so as to be able to order the next A block from tape a.



If we again exhaust the A block, we can replace it directly from rI without disturbing the two standby blocks. If, however, the B block is exhausted next, we fill B from the Y standby, transfer rI to Y, and order the third B block from tape:

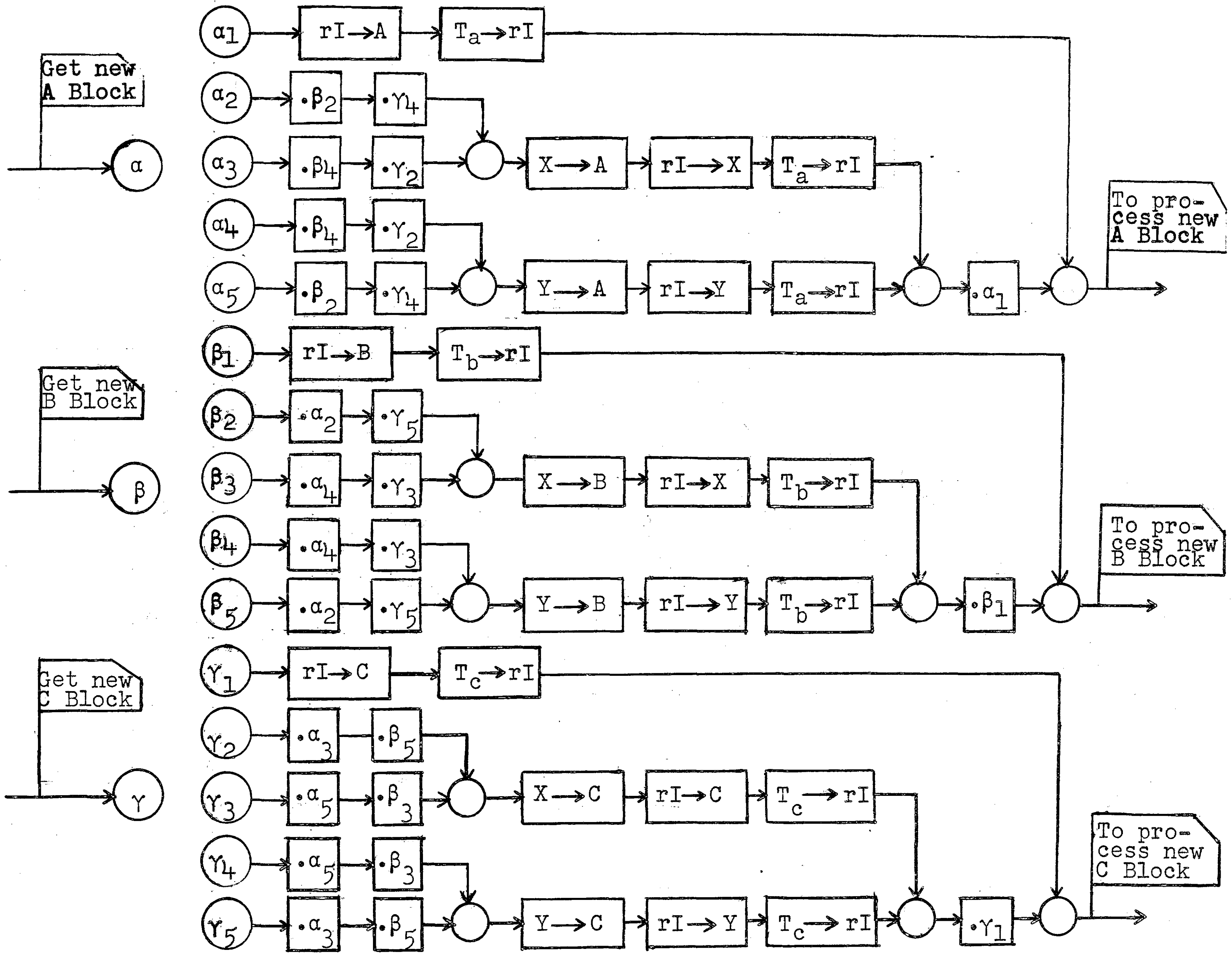


Thus, in this example, the standby blocks which start with A and B items soon are changed to C and A items. Unlike the simple procedure described in the Programming Manual where each standby block always contains only one type of item, this method permits "floating" information in the standbys. The attached flow chart indicates the general solution for this method. At the beginning, connectors α_3, β_5, γ_1, are set for the initial conditions already described.

Let us investigate the logic behind the flow chart by first making a table showing the possible configurations of X, Y, and rI:

Configuration Number	Contents of:		
	rI	X	Y
1	A	B	C
2	A	C	B
3	B	A	C
4	B	C	A
5	C	B	A
6	C	A	B

Now suppose an A block is needed; this block will always be available in either rI, X, or Y. After placing this A block in the A input position, we might ask ourselves: What is the new configuration? The following table shows the configuration numbers resulting from the act of obtaining an A block, B block, or C block from each of the six possible configurations:



Get new A Block
α
α1
rI→A
Ta→rI
α2
·β2
·γ4
α3
·β4
·γ2
X→A
rI→X
Ta→rI
α4
·β4
·γ2
α5
·β2
·γ4
Y→A
rI→Y
Ta→rI
·α1
To process new A Block
Get new B Block
β
β1
rI→B
Tb→rI
β2
·α2
·γ5
β3
·α4
·γ3
X→B
rI→X
Tb→rI
β4
·α4
·γ3
β5
·α2
·γ5
Y→B
rI→Y
Tb→rI
·β1
To process new B Block
Get new C Block
γ
γ1
rI→C
Tc→rI
γ2
·α3
·β5
γ3
·α5
·β3
X→C
rI→C
Tc→rI
γ4
·α5
·β3
γ5
·α3
·β5
Y→C
rI→Y
Tc→rI
·γ1
To process new C Block

Configuration Number Transformation Table			
Original	Configuration after obtaining an:		
	A Block	B Block	C Block
1	1	3	5
2	2	4	6
3	1	3	6
4	2	4	5
5	1	4	5
6	2	3	6

Note that obtaining an A block from configurations 1 or 2 leaves these configurations unaltered while configurations 3, 4, 5, and 6 are altered by obtaining an A block. By alteration is meant a change in position of either an A, B, or C block. The connector α_1 in the flow chart suffices to obtain the next A block when either configuration 1 or 2 exists. α_2 and α_3 are the connectors used where an A block is in X: α_2 for configuration 3, α_3 for configuration 6. Separate connectors are needed even though the A block is in X for each configuration since the act of moving the A block alters the position of the B and C blocks in different ways. α_4 and α_5 are connectors used when the A block is in Y: α_4 for configuration 4 and α_5 for 5. A similar assignment of connectors to configurations applies for obtaining B and C blocks. These are summariized in the following table:

Configuration Number	Connectors A	B	C
1	α_1	β_2	γ_4
2	α_1	β_4	γ_2
3	α_2	β_1	γ_5
4	α_4	β_1	γ_3
5	α_5	β_3	γ_1
6	α_3	β_5	γ_1

Now that we have assigned the variable connector numbers for each configuration, we note in the transformation table that obtaining an A block from configuration 6, for instance, will produce configuration 2. Now configuration 6 requires that α_3, β_5, and γ_1 be set, and the resulting configuration 2, will require connectors α_1, β_4, and γ_2 to be set. Thus, the connector α_3 must set β_4 and γ_2 as well as α_1. This kind of reasoning makes evident the rationale behind the flow chart.

The average time required to obtain an A, B, or C block is 11,622 μs. 387 words of storage are required for the three input blocks, the two standby blocks, and the coding and constants.

<u>Summary</u>

While it is evident from the foregoing discussion that the so-called preselection method of keeping rI filled is superior to the standby block schemes in both time and space requirements, there is occasionally a class of tape problems encountered for which preselection is not suitable. For these problems, one of the standby block methods is preferable. For two-way input the automatic method is preferred, the reversal method being used when the second input is of very small volume and memory space is at a premium.

For three-way input the automatic method is preferred over the two-standby block method (Chapter 10, Sec. 8). For more than three-way input with no great disparity in the relative volumes of the inputs an extension of the two-standby block method is superior in space and time to similar extensions in the methods described above.

2. <u>Arrangement of information for efficient transfer within the computer</u>.

<u>Item sizes a sub-multiple of 60</u>.

As a direct consequence of reading information in units of a block rather than an item, it is necessary to consider the most efficient arrangement of the items within the block. Where the item size is one, two, or ten words in length, the usual sequential arrangement of an entire item followed by the next one in succession clearly does not offer much room for improvement. In the case of one- or two-word items, minimum latency considerations might require that the order of items in the block be rearranged from the normal one for a particular run, but such a situation would be unusual.

The above statements follow directly from the existence of one-, two-, and ten-word registers for transferring information from an input block to an output block or to working storage as the case may be.

In dealing with twenty- and thirty-word items, or four- and six-word items, the same remarks apply with several transfers taking the place of the one required previously. There are cases, however, where the normal sequence of items can be vastly improved upon. Consider the twelve-word item for example. In general, to transfer a twelve-word item from an arbitrary position in the input block to working storage or to an arbitrary position in the output block requires six V-W instruction pairs.

If this is accomplished with iterative coding, the following routine will have to be executed six times for each item transferred:

```
100   V40200    W  300
101   B  100    A* 104
102   C  100    U  100
103   00 000    U  XXX   to processing routine
104   010002    000002
```

Straight line coding would offer some improvement in speed but would require 35 lines of coding per input to transfer the items to working storage and 35 lines of coding to transfer from working storage to the output block.

The following arrangement of the twelve-word items would be far more efficient, both with regard to memory space and execution time: (j, k is word k of item j)

Cell										
200	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10
210	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10
220	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	3,10
230	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	4,10
240	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9	5,10
250	1,11	1,12	2,11	2,12	3,11	3,12	4,11	4,12	5,11	5,12

For this configuration, an item transfer would require only a Y-Z and a V-W. Using straight line coding 19 lines per input will suffice to transfer the item to working storage and similarly 19 lines for the output block. The coding necessary to accomplish the transfer of items to working storage is illustrated below.

```
100   00 000
                U  101
101   R  100
                U  110
102   R  100
                U  113
103   R  100
                U  116
104   B  108
                C  100
105   V  258
                Y  240
106   W  250
                Z  200
107   00 000
                U  XXX   to processing routine
```

```
108   R  100
                U  109     constant
---------------------------------------------
109   30 200
                U  XXX     to preselector read
---------------------------------------------
110   V  252
                Y  210
111   W  250
                Z  200
112   00 000
                U  XXX     to processing routine
---------------------------------------------
113   V  254
                Y  220
114   W  250
                Z  200
115   00 000
                U  XXX     to processing routine
---------------------------------------------
116   V  256
                Y  230
117   W  250
                Z  200
118   00 000
                U  XXX     to processing routine
```

Use of minimum latency coding could further improve this routine. The following examples will illustrate further applications of this technique.

15-Word Item

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	3,10
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	4,10
1,11	1,12	1,13	1,14	2,11	2,12	2,13	2,14	3,11	3,12
3,13	3,14	4,11	4,12	4,13	4,14	1,15	2,15	3,15	4,15

5-Word Item

1,1	1,2	1,3	1,4	2,1	2,2	2,3	2,4	3,1	3,2
3,3	3,4	4,1	4,2	4,3	4,4	5,1	5,2	5,3	5,4
6,1	6,2	6,3	6,4	7,1	7,2	7,3	7,4	8,1	8,2
8,3	8,4	9,1	9,2	9,3	9,4	10,1	10,2	10,3	10,4
11,1	11,2	11,3	11,4	12,1	12,2	12,3	12,4	1,5	2,5
3,5	4,5	5,5	6,5	7,5	8,5	9,5	10,5	11,5	12,5

3-Word Item

1,1	1,2	2,1	2,2	3,1	3,2	4,1	4,2	5,1	5,2
6,1	6,2	7,1	7,2	8,1	8,2	9,1	9,2	10,1	10,2
11,1	11,2	12,1	12,2	13,1	13,2	14,1	14,2	15,1	15,2
16,1	16,2	17,1	17,2	18,1	18,2	19,1	19,2	20,1	20,2
1,3	2,3	3,3	4,3	5,3	6,3	7,3	8,3	9,3	10,3
11,3	12,3	13,3	14,3	15,3	16,3	17,3	18,3	19,3	20,3

Thus we see that a 15-word item can always be transferred using one Y-Z, two V-W's and one B-C. Similarly, a five-word item can be transferred by using two V-W's and a B-C.

It is to be noted that input data from non-Univac sources, in general, can not be obtained in the efficient configurations illustrated above. Therefore, the rearrangement of the information will have to be executed by the first Univac run on the data.

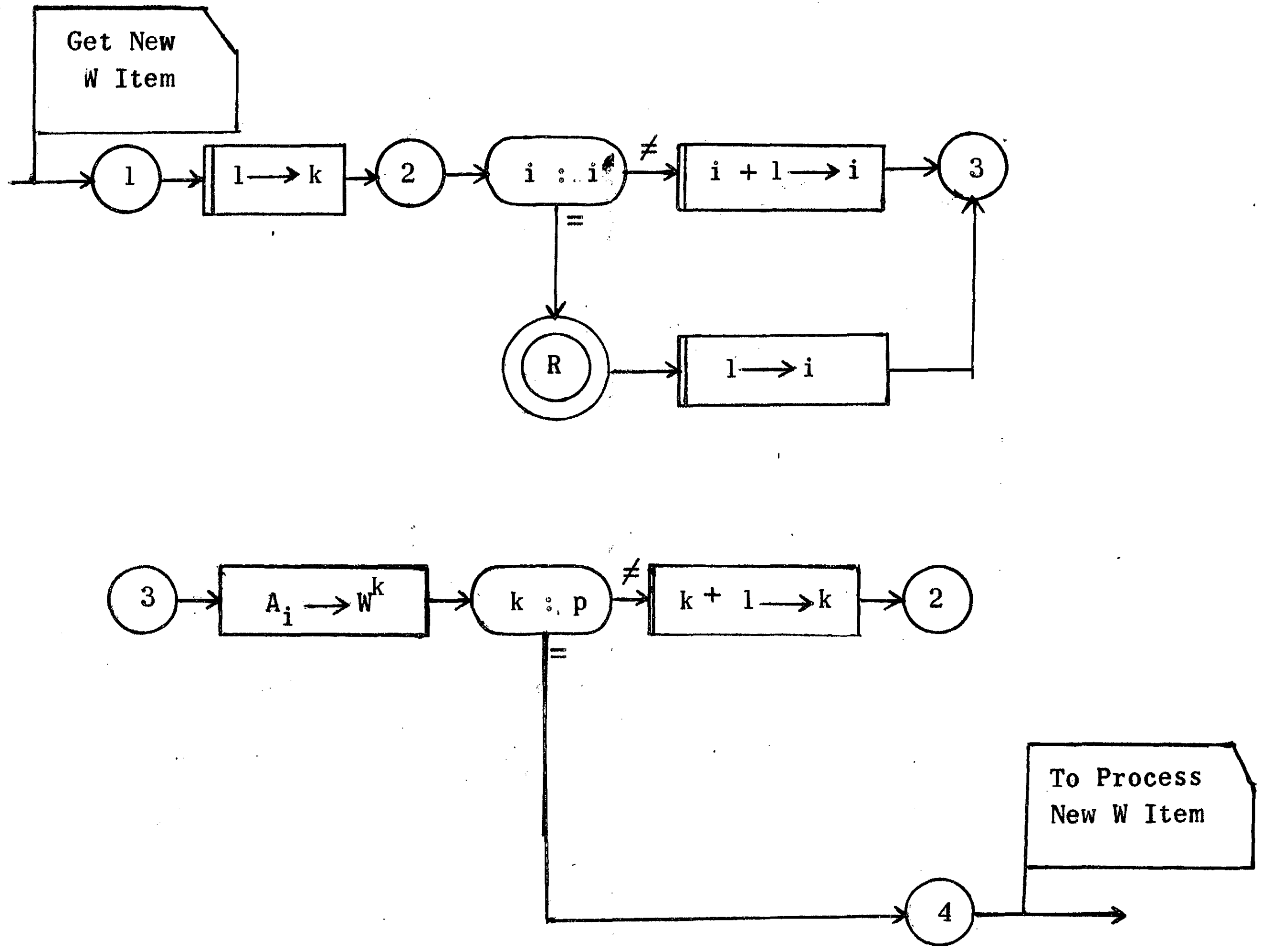
<u>Item sizes not a multiple or sub-multiple of 60</u>

Occasionally, an item size which cannot be contained an intergral number of times within a block is desirable. Generally, this would be true where the processing is done at less than tape time. Then, any reduction of the total amount of tape required will reduce the overall processing time.

An obvious solution to this problem is to consider the item as being composed of a series of sub-items, the length of the sub-items <u>being</u> a sub-multiple of a block. In the practical case the sub-item will be ten words in length as the time required to move many one- or two-word sub-items negates any advantage gained in reducing the amount of tape. For example, if the number of digits required in an item is 420 (35 words) an item size of 40 words could be used.

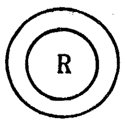
Each item is then composed of four ten-word sub-items. If it were not possible to handle items not multiples or sub-multiples of a block, the item size would have to be 60 words in length. The 40-word item thus saves 1/3 of the tape required and reduces the processing time up to 1/3 depending on how much the item processing time is under the tape time.

The following flow chart is an iterative solution to the selection of an N-word item, W, composed of p_k sub-items A_i. Each input block, A, containing i′ such sub-items. The k in W^k is shown as a superscript since W^1, W^2, ..., W^k are, in actuality, fields of the item W.



Note: initially i = 0

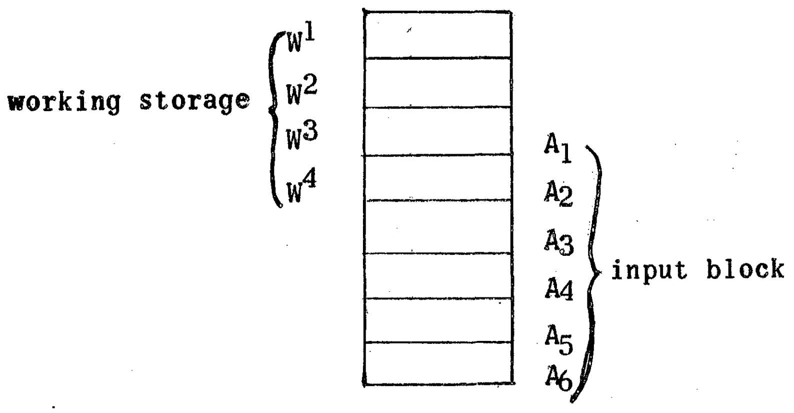
The symbol



represents an appropriate subroutine which will obtain the next A block. The reader should note that while it is not possible to use all of the input block A for the working storage W, it is possible to use a portion of the block for some item sizes and thereby conserving memory space. For example, consider the possible configurations of blocks containing a forty-word item:

Block 1	Block 2	Block 3
1-1	2-3	4-1
1-2	2-4	4-2
1-3	3-1	4-3
1-4	3-2	4-4
2-1	3-3	5-1
2-2	3-4	5-2

(The notation j-k means the kth sub-item of the item j). Thus, only two block configurations are possible. Now, if the working storage W^1, W^2, W^3, and W^4 is positioned as follows with respect to the input block A_1, A_2, A_3, A_4, A_5 and A_6:



Only twenty extra positions are necessary for the working storage W.

Although it would be desirable to have the read routine be the preselector, it has not been found possible, up to this time, to suitably modify this technique to make it work properly for the "odd" item sizes. This means that one of the standby block methods described earlier must be employed in ordering blocks from tape.

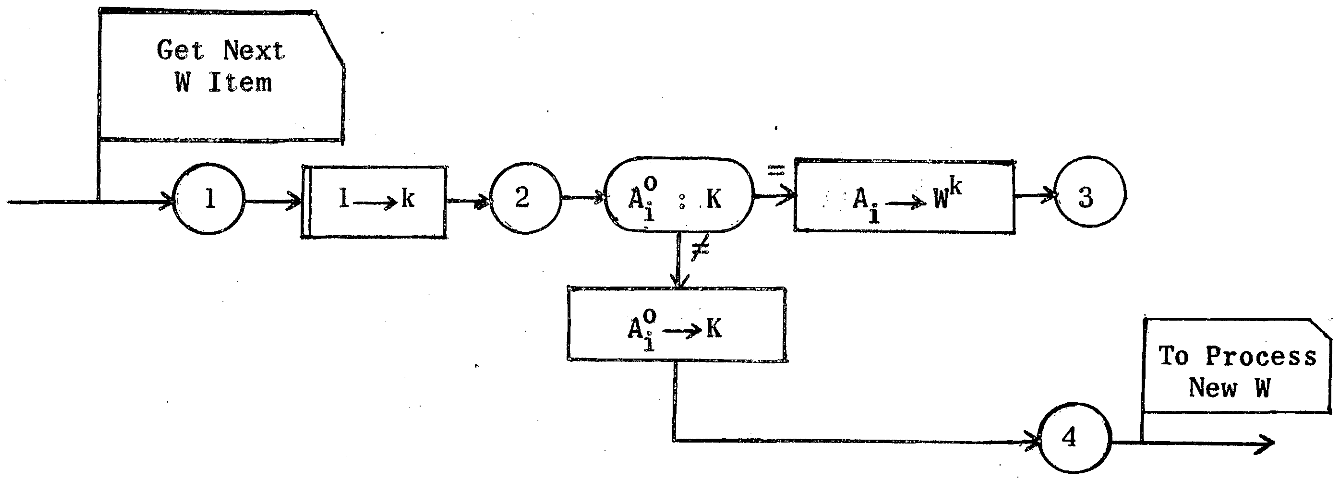
Item sizes not of fixed length.

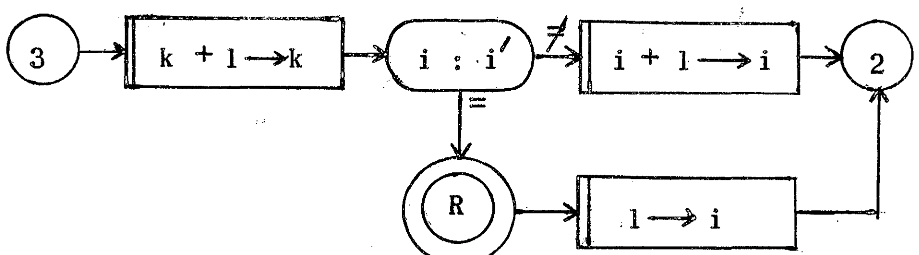
In some large volume tape problems the variation in minimum required digits for the items of a file may be very large. This case may occur when a master information file is laid out. If we allot to each item of the file the tape space required for the largest possible item that can occur, we may find that a very large portion of the file contains "blank" areas causing inefficient use of the tape and could greatly increase the processing time.

Hence, it is desirable to let each item use as little tape space as is feasible. If this is done, then the items on the tape may be of varying lengths. This variable item size can be easily handled by the computer by considering each item to be composed of a variable number of sub-items, each sub-item being of a fixed size. All that is then necessary is to read into the memory all of the sub-items comprising an item. The routine that selects the next sub-item from the block (or the next block from tape) is designed in the usual manner since the sub-item is of fixed size.

The problem next to be solved is how do we know when all of the sub-items for a given item have been assembled? This may be done in several ways, the one to be selected depending on the particular situation.

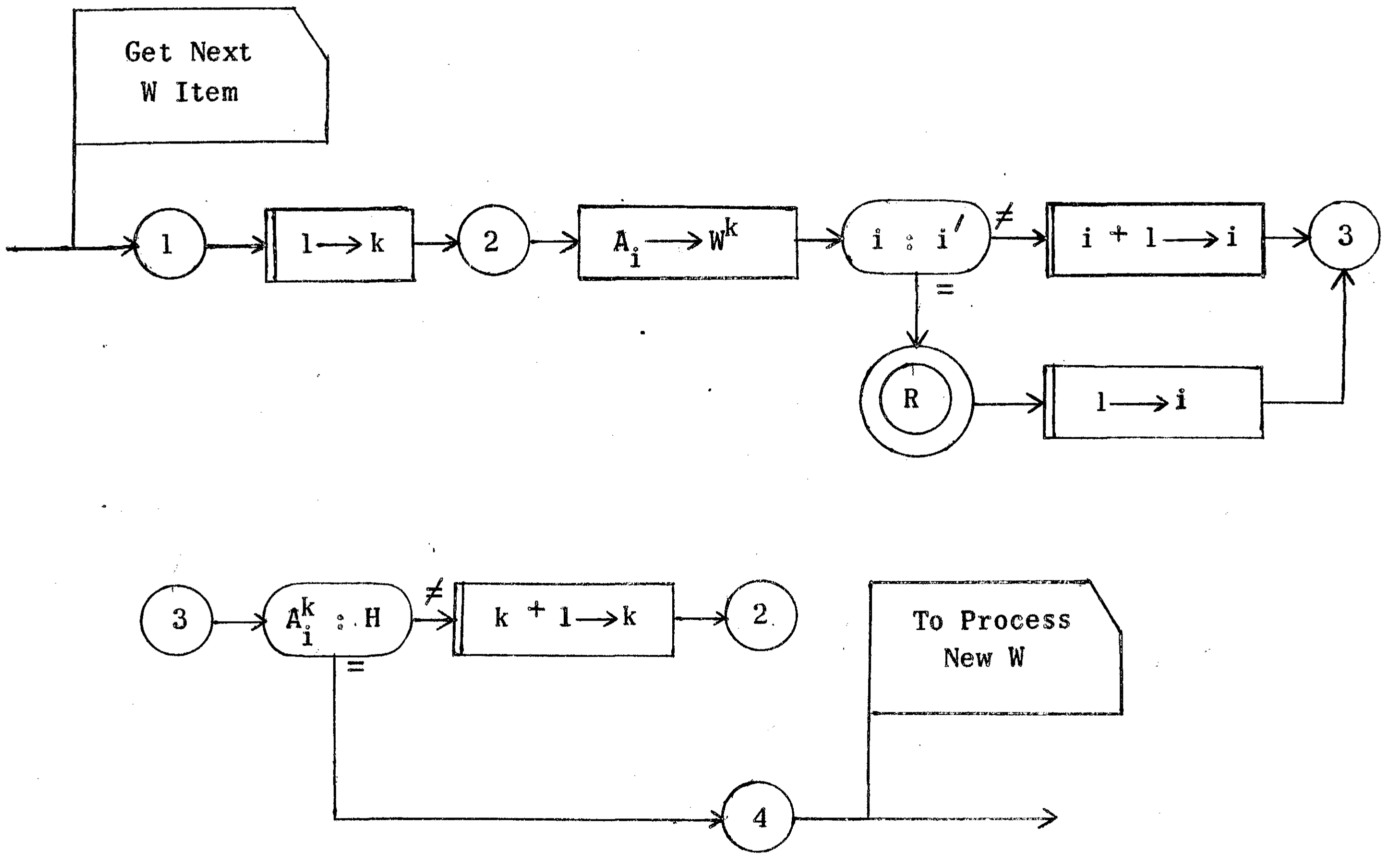
The first method to be described assumes that each sub-item carries the item key word. Then, as each sub-item is selected its key word is checked for equality with the current key word. The flow chart for this method is shown below. A is the input block containing i' sub-items A_i. W is the working storage which will contain the current tape a item; this item is composed of fields W^k. K stores the key word for the current input item. K is initially set to the value of the key word of the first input item. The key word of each sub-item is assumed to be field A_i^o.





This method assumes that two items with the same key word is not possible. It is also inherently assumed that the space required for repeating the key word is not unduly excessive.

Perhaps an inherently better method is to consider the sub-items of an item to be of two kinds. The first sub-item may be thought of as a "header" sub-item which contains the key word for the item. The remaining sub-items can be considered as "trailer" sub-items. Now, instead of requiring all sub-items to carry the item key word, we can simply require that each sub-item have a fixed digit position which will contain, say, the symbol "H" if the sub-item is a header and "T" if the sub-item is a trailer. The flow chart for selecting the N-word item W composed of a variable number of sub-items A_i, with key as A_i^k.



IDENTIFICATION BLOCKS AND RERUNS

1. Introductory Considerations

The governing characteristic of commercial data processing problems is the very large amounts of information (records or items) to be fed into the computer. Even some of the simpler problems may involve tape files of 20 to 25 reels per file, while large problems can demand nearly 300 reels per file. Most often the processing of these tape files requires that the items comprising the file be processed in a definite sequential manner. If this is true, then certain precautions must be exercised to insure that these items are processed in sequence. Two techniques in common use to guarantee this sequential processing are identification blocks and block counts.

The reader should bear in mind the situation encountered by an operator during the running of a large tape problem involving several hundred reels. Since the contents of a reel of tape are not directly visible to the operator, a gummed label is usually attached to each reel. Now, although each reel has a visible identification label, what assurance have we that when the computer calls for the fifth reel of file A to be mounted on a Uniservo that the operator does indeed mount reel #5 of file A on that Uniservo? Or, granting that a reel bearing a visible label "Reel #5 - File A" is mounted on the desired Uniservo, what guarantee is there that this label is a correct description of the contents of the reel? These questions arise because of the bias of human operations toward error. Since the computer is self-checked in its operation, it is desirable that human intervention in the processing chain be limited as much as possible and that such intervention be done in a simple fashion, subject to check.

Assuming, now, the correct data reel has been mounted on the desired Uniservo, can we rest assured that the items on that reel will be processed in their correct sequence by the computer? We can provided, again, that no operator intervention is necessary. Occasionally, it is necessary for the operator to affect the position of the tapes mounted on the Uniservos; for example, if a block on Uniservo #2 was read into rI incorrectly, the operator will set into SR from SCP instructions causing the block to be re-read.* If the number of data blocks recorded on the tapes is placed on the tape itself, then the computer can check that the operator did not misposition the tape. With this check made possible, a block of items cannot get lost in the processing run, nor can a block be processed twice.

To illustrate the use of identification blocks and block counts in checking the human interventions mentioned above, a general two-way merge for ten-word items will be described. First however, we shall describe the standard data tape format.

* This condition will not often arise if the computer is equipped with the automatic re-read device.

2. The Standard Data Tape Format

The first block recorded on a data tape will be the identification block of which the first word only will be of interest to us now.* This word is the reel identification number, designating the file of which this reel is a part, and the sequence number of the reel within the file. For our two-way merge the identification word has the following appearance (for the 13th reel of the file A):

Ɍ Δ Δ A 0 1 3Δ Δ Δ Δ Δ

Following this ID block are the data blocks. If the reel is not the last reel of the file, there may be up to 1981 full data blocks. Following the last data block there will be recorded a sentinel block having the following composition: word 00 (the first word of the block) consists of twelve Z's (the sentinel); word 50 also consists of twelve Z's; word 51 contains the number of data blocks recorded on tape plus this sentinel block. For example, if this tape contains 1981 data blocks, word 51 of the sentinel block looks like:

0000000001982

Word 59 (the last word of the block) will not consist of twelve Ᵽ's. Following this block will be other blocks which are provided for the possibility of the reruns to be described later. If the reel is the last reel of a file, its identification block is in the same format noted. However, being the last reel of a file, it may have less than 1981 full data blocks recorded on it. Following the ID block will be as many data blocks as needed (1981) to record the remaining items. If the items on the last reel are a multiple of six, they will exactly fill the last data block. In this case, the sentinel block already described will follow the last data block, except that word 59 of the sentinel block consists of twelve Ᵽ's which serve to identify the last reel of a file. But if the data items are not a multiple of six in number, the last data block will be only partially filled. In this case, a Z sentinel will be placed in the next word position of the partial data block, following the last word of the last data item. For example, if the partial block contains only three ten-word items, word 30 will contain all Z's. In addition, the 50th word of the block will also be a Z sentinel; amd word 51 will contain the number of full data blocks on this tape, plus the partial data block. Word 59 of this block will consist of twelve Ᵽ's. In either case, following the partial or full sentinel block, is a second full sentinel block.

3. Description of the Generalized Two-Way Merge

Briefly, the generalized two-way merge problem is this:

* Other information in the ID block is for use in re-runs.

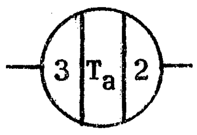
There are two multi-reel files, labeled for convenience, file A and B. Each file consists of a series of ten-word items, the first word of each item being a key word or serial number. The items in each file are recorded on tape in an ascending sequence by their key words. The reels comprising each file are recorded in the standard data tape format just described. The problem is to produce a new file, labeled C, also in standard data tape format, which contains every item on the A and B files, arranged in ascending sequence by their key words. Two Uniservos will be allotted for each file:

Uniservos	File
2,3	A
4,5	B
6,1	C

In this way, after reel 1 of file A, which is mounted on Uniservo 2, has been processed, it is given a rewind instruction. When this tape is rewound, the operator will replace it with reel 3 of file A. While reel 1 is being rewound, reel 2, which has been mounted on servo 3, is processed by the computer; thus, the computer need not wait for the rewinding of a reel and the mounting of a new reel of tape. This same prodedure is used for the B and C files as well.

The complete flow chart and coding of this problem is attached.

The following list is an explanation of the flow chart symbols.

T_a	=	Tape on Uniservo a
A	=	A block of 6 items from T_a
A_i	=	ith item (i = 1,2,...,6) of A
A_i^n	=	nth word (n = 0,1,2,...,9) of A_i
Ia	=	File and reel label for T_a
Δa	=	Block counter for T_a
$T_a \longrightarrow$ rI		A block from T_a is read into rI (forward read)
RWD* T_a		Rewind T_a with interlock set
$C \longrightarrow T_c$		The block C is written on T_c
		Servo interchange symbol, a, which was Uniservo 2, now becomes 3. Circle rotates 180° so that next trip through a, which was servo 3, becomes 2 again.
- - - - ->		Flow line followed by individual items
⟹		Flow line followed for end (or start) of block
⇛		Flow line followed for end (or start) of tape
⟶		Flow line followed for end (or start) of file

A brief description of the flow chart follows:

At the start of the problem the flag indicates that the input and output Block counters are set to zero, and the input and output tape labels are set to one. At connector $\textcircled{1}$ the input file labels with their appropriate Uniservo numbers are printed so that the operator can mount the first reel of the A file on the Uniservo 2, the second reel on 3, the first reel of the B file on Uniservo 4, the second on 5, and blanks on Uniservos 6 and 1. The next step, $\textcircled{21}$, is to write the identification block on the output tape. At $\textcircled{2}$ is conditional transfer breakpoint 1. If breakpoint 1 has been depressed, the computer will stop at this point. If the operator then forces transfer, the computer will go to $\textcircled{6}$ for the re-run procedure. The normal path through this breakpoint is to $\textcircled{16}$, where the tape label for tape b is tested. If the wrong T_b has been mounted, the tape will be rewound with interlock, the correct label printed on SCP, and the computer will stall on the next read until the new tape is mounted. When T_b is correct (or corrected), the first data block is read into the memory and the computer goes to $\textcircled{18}$ to test the T_a label. When the first data block of T_a and T_b has been read, the computer goes to $\textcircled{10}$ where the preselector method is used to fill rI. $\textcircled{11} \rightarrow \textcircled{12} \rightarrow \textcircled{\delta} \rightarrow \textcircled{11}$ is the merge proper. As each output block C is filled, the computer goes to $\textcircled{19}$ where the output item counter is reset, the block written on T_c, and the output block counter increased. When the 1981 data block has been written on the tape, a Z sentinel block is written and $\textcircled{\beta}$ set to indicate the contents of rI. The output tape label, Lc, is printed on SCP to enable the operator to label the current output tape correctly. Lc is then augmented for the next output reel, the output block counter, Δc, is reset, and the output tape rewound with interlock. The output servos are then switched and the augmented tape label is written on the new output tape. The output operations concerned with re-runs will be discussed later.

As each A block is exhausted, the computer goes to $\textcircled{17}$ where the A input item counter is reset and the next A block obtained from rI. The input block counter, Δa, is augmented and the new input block is examined for a sentinel in the key word position of the last item of the block. Should this be a sentinel block, the computer data block count, Δa, is tested against the block count on the tape. If they do not agree, the next operation is at $\textcircled{14}$ where all tapes are rewound and the error noted on SCP. If the counts are correct, T_a is rewound with interlock and a test is made to determine whether the last T_a has been processed. If it has not, the A input block counter is reset, the T_a input servos interchanged, and the T_a tape label augmented. At $\textcircled{18}$ the new T_a label is examined as noted before. If this was the last A tape, $\textcircled{\gamma 2}$ is set to send us later into the ending routine at $\textcircled{13}$. A similar procedure holds for exhausting a B input block.

4. Re-run Procedure

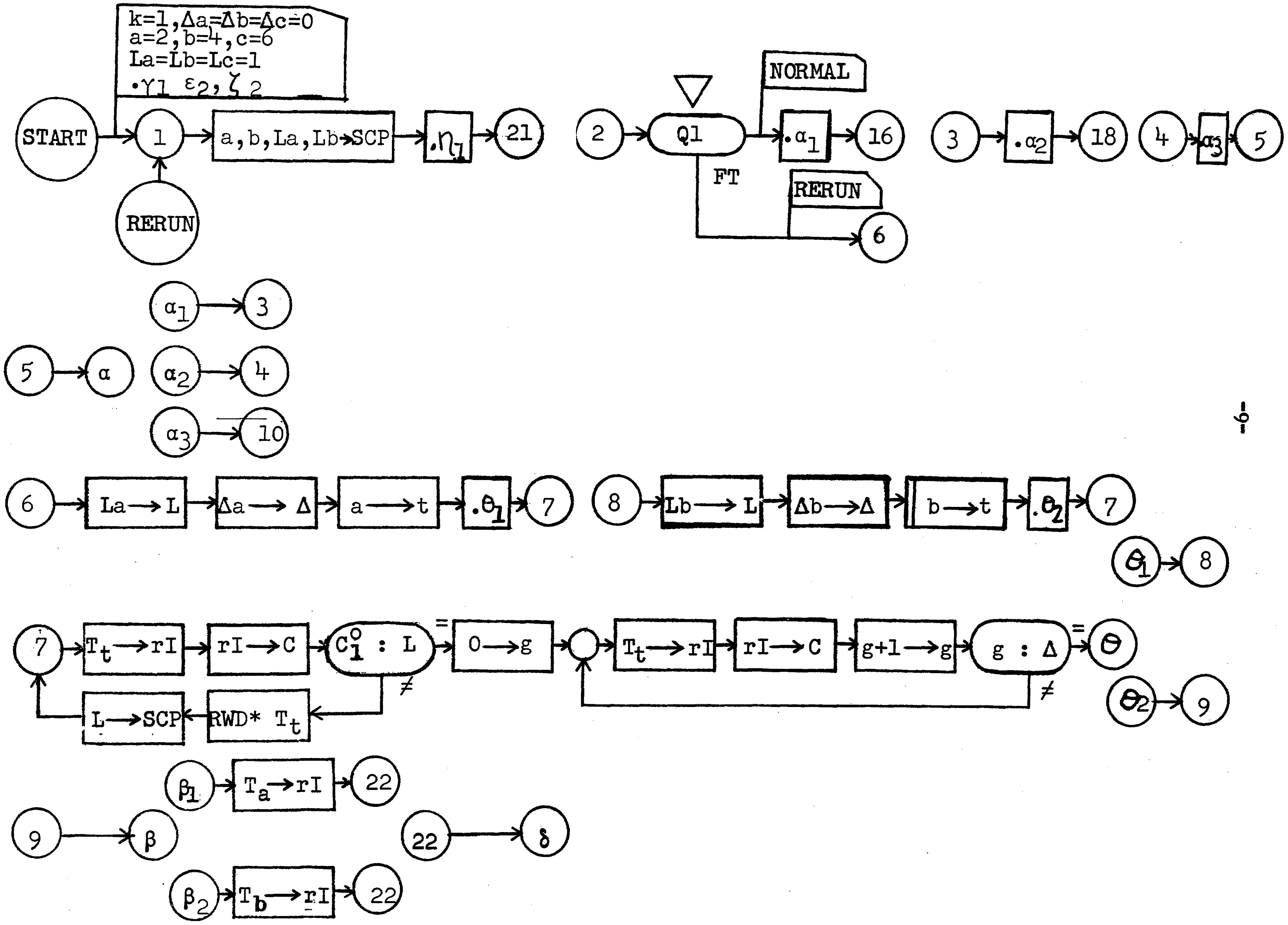
Next, let us examine the procedure planned for a possible re-run. A procedure must be prepared by the programmer for the continuation of computation on his problem in the event of a computer breakdown. It is always preferable to have a general re-run procedure which will be used for any re-run to avoid any possibility of errors on the part of the operator. Thus, the re-run (to be explained) is designed for even the extreme cases where nothing in the computer can be used.

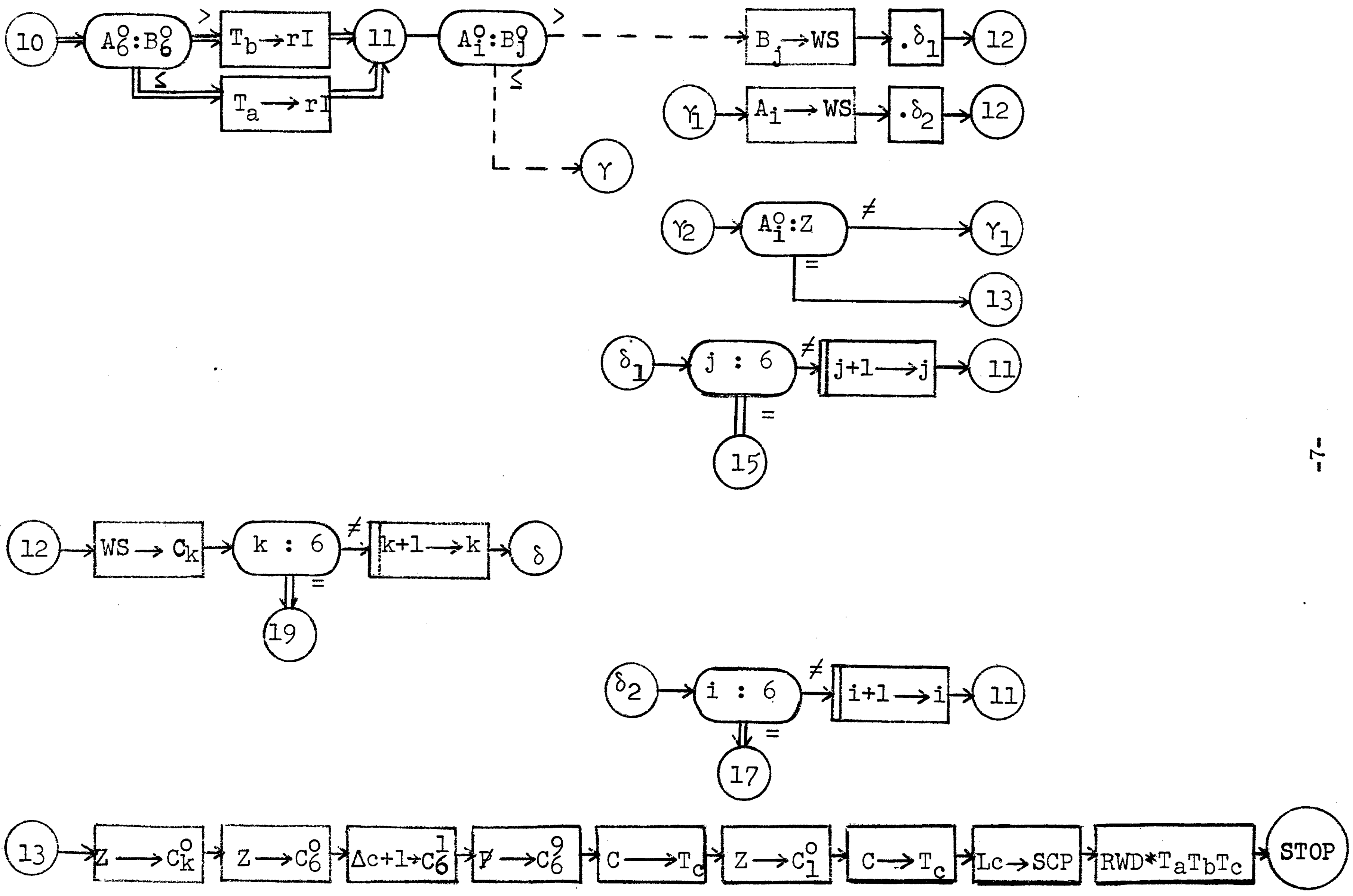
The theory of this re-run method is quite simple: We "photograph" the memory and register setup at the conclusion of each output tape. Thus, in the event of a computer breakdown while processing output tape k we need only restore the memory and reposition the tapes as they existed at the beginning of output tape k. At the end of output tape k-1, after we have written the sentinel block, we set a variable connector $\textcircled{\beta}$ in accordance with the contents of rI. Then, at $\textcircled{20}$ we augment the output tape label, L_c, and reset the output block counter, Δc. At this point, we write the entire contents of the memory onto the output tape and rewind it.

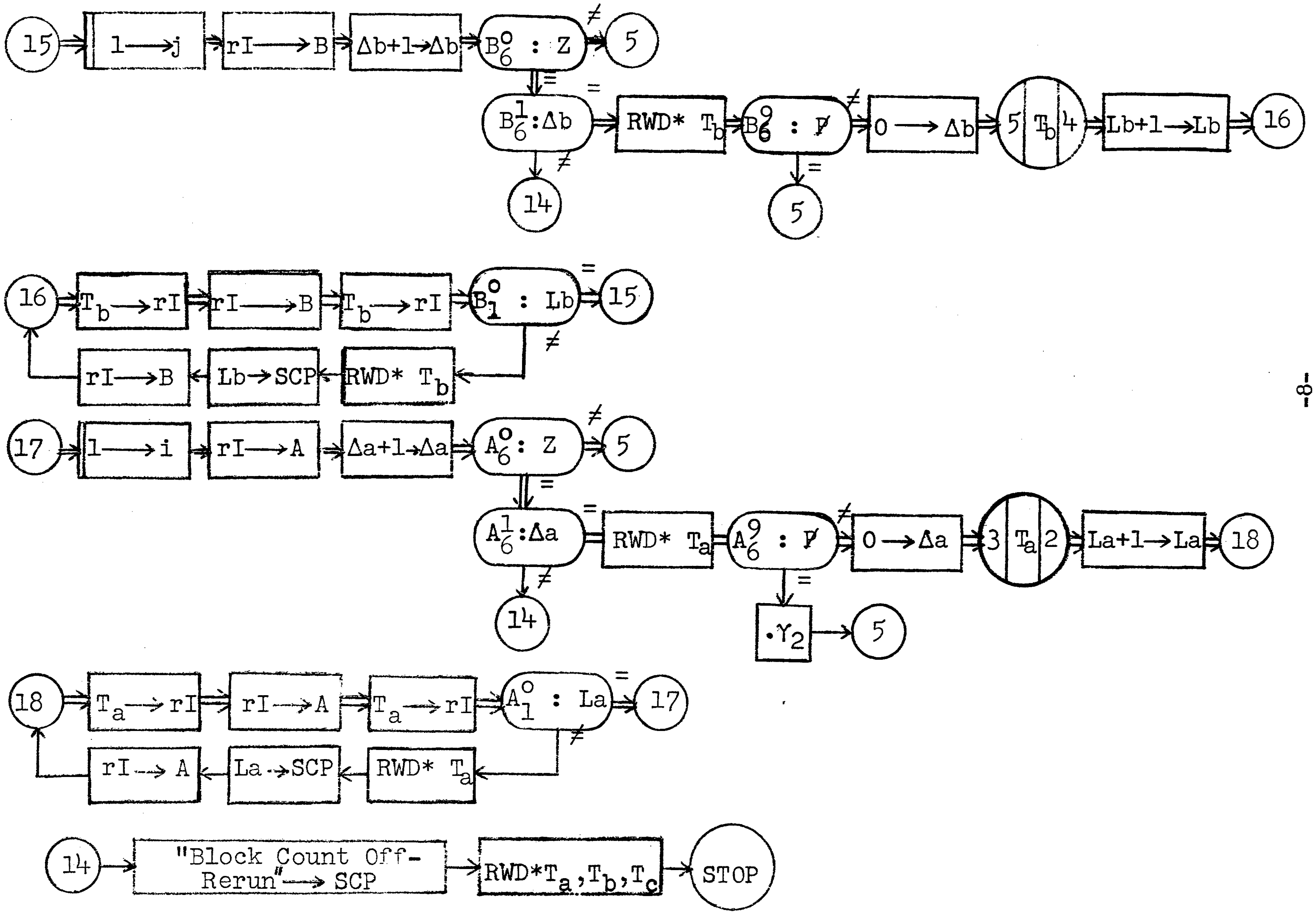
Let us now suppose a computer breakdown or an operator error occurs, or even in some succeeding run one of the output tapes of this run proves unreadable. In all of these cases, we need only locate the memory dump recorded on the previous output tape. By reading this memory dump back into the computer, we will restore the memory to the state which existed at the beginning of the output tape which later proved unreadable, or during the writing of which the computer or operator error occurred. The input tape labels and block counters are used to reposition the input tapes. Hence, we have reconstructed the problem at the beginning of an output tape.

If a re-run is to be done, breakpoint 1 is depressed and the last good output tape is read down past the sentinel blocks (a search routine is placed in the output tape ID block for this purpose).* The memory dumped on this output tape is then read into the computer and control transferred to the start of the merge routine. After writing the output tape ID block, the computer stops on Q1 where we release the breakpoint and force transfer. At $\textcircled{6}$ the input tape label for T_a is checked and this tape positioned in accordance with the reading of a obtained from the memory dump. At $\textcircled{8}$ the T_b label is checked and this tape positioned. At $\textcircled{\beta}$ rI is filled from the appropriate tape, control then being transferred to $\textcircled{\delta}$ to begin the reprocessing of this new output tape.

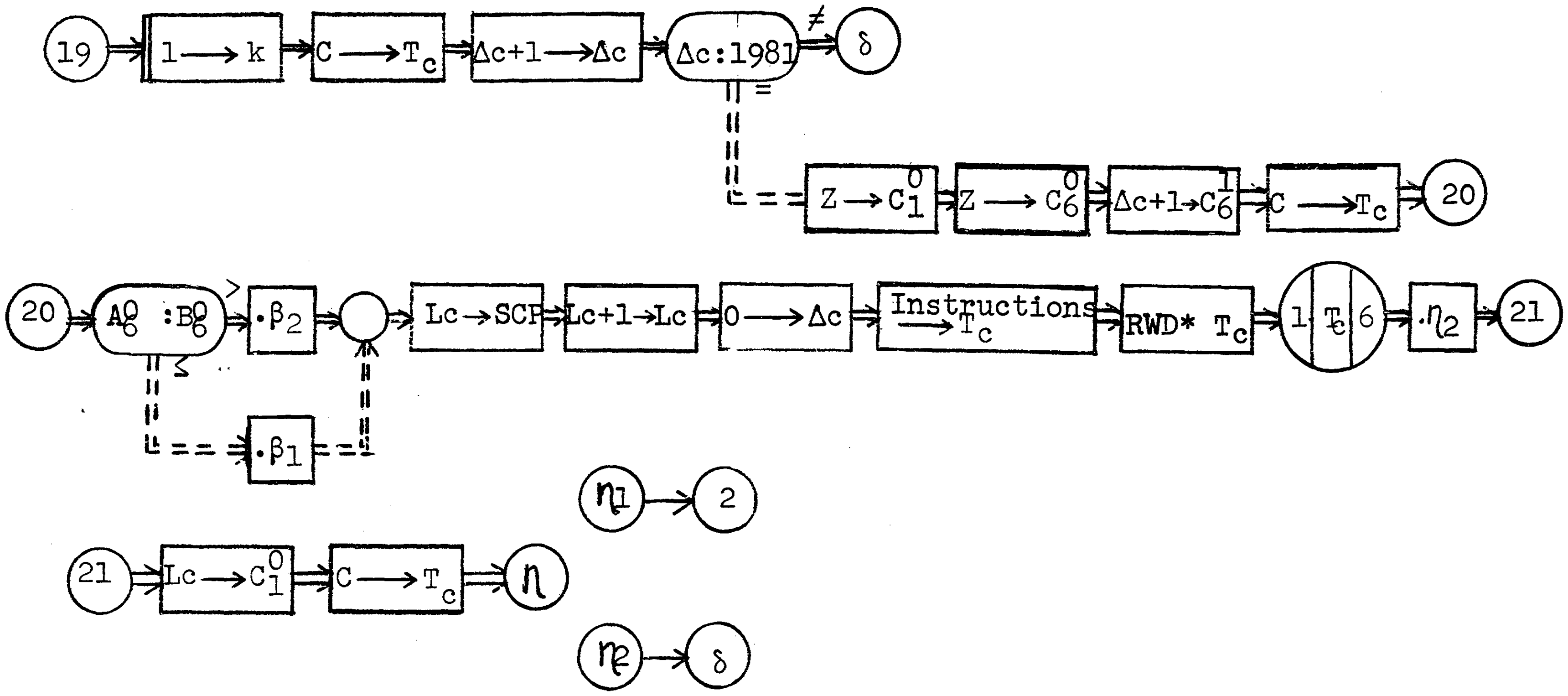
* As indicated in the coding for this example, the operator will mount on servo 1 the last completed output tape. This tape is easily identified by the Lc last printed on SCP. He then does Initial Read on this tape. The instructions in cell 000 which set up in SR are ignored through manual clearing of SR. The computer then executes the instructions in cell 001 of the ID block which locates the memory dump at the end of this tape.


k=1, Δa=Δb=Δc=0
a=2, b=4, c=6
La=Lb=Lc=1
•γ1 ε2, ζ 2
START
1
RERUN
a, b, La, Lb→SCP
.η1
21
2
Q1
NORMAL
.α1
16
FT
RERUN
6
3
.α2
18
4
α3
5
α1
3
5
α
α2
4
α3
10
6
La→L
Δa→Δ
a→t
.θ1
7
8
Lb→L
Δb→Δ
b→t
.θ2
7
θ1
8
7
Tt→rI
rI→C
C°1 : L
=
≠
0→g
Tt→rI
rI→C
g+1→g
g : Δ
=
≠
θ
θ2
9
L→SCP
RWD* Tt
β1
Ta→rI
22
9
β
22
δ
β2
Tb→rI
22


10
$A_6^0 : B_6^0$
>
≤
$T_b \rightarrow rI$
$T_a \rightarrow rI$
11
$A_i^0 : B_j^0$
>
≤
$B_j \rightarrow WS$
$.\delta_1$
12
γ
γ_1
$A_i \rightarrow WS$
$.\delta_2$
12
γ_2
$A_i^0 : Z$
≠
=
γ_1
13
δ_1
j : 6
≠
=
$j+1 \rightarrow j$
11
15
12
$WS \rightarrow C_k$
k : 6
≠
=
$k+1 \rightarrow k$
δ
19
δ_2
i : 6
≠
=
$i+1 \rightarrow i$
11
17
13
$Z \rightarrow C_k^0$
$Z \rightarrow C_6^0$
$\Delta c+1 \rightarrow C_6^1$
$F \rightarrow C_6^9$
$C \rightarrow T_c$
$Z \rightarrow C_1^0$
$C \rightarrow T_c$
$Lc \rightarrow SCP$
$RWD^* T_a T_b T_c$
STOP



15
1 → j
rI → B
Δb+1 → Δb
B⁰₆ : Z
≠
5
=
B¹₆ : Δb
=
RWD* T_b
B⁹₆ : F̸
≠
0 → Δb
5 T_b 4
Lb+1 → Lb
16
≠
14
=
5
16
T_b → rI
rI → B
T_b → rI
B⁰₁ : Lb
=
15
≠
RWD* T_b
Lb → SCP
rI → B
17
1 → i
rI → A
Δa+1 → Δa
A⁰₆ : Z
≠
5
=
A¹₆ : Δa
=
RWD* T_a
A⁹₆ : F̸
≠
0 → Δa
3 T_a 2
La+1 → La
18
≠
14
=
·Y₂
5
18
T_a → rI
rI → A
T_a → rI
A⁰₁ : La
=
17
≠
RWD* T_a
La → SCP
rI → A
14
"Block Count Off-Rerun" → SCP
RWD* T_a, T_b, T_c
STOP



19
1 ⟶ k
C ⟶ T_c
Δc+1 ⟶ Δc
Δc:1981
≠
δ
=
Z ⟶ C_1^0
Z ⟶ C_6^0
Δc+1 → C_6^1
C ⟶ T_c
20
20
A_6^0 : B_6^0
>
≤
·β_2
·β_1
Lc → SCP
Lc+1 → Lc
0 ⟶ Δc
Instructions ⟶ T_c
RWD* T_c
1 T_c 6
·η_2
21
η_1 → 2
21
Lc ⟶ C_1^0
C ⟶ T_c
η
η_2 → δ

5. Coding for Generalized Two-way Merge

```
     000   R2W   007
                       UMG   005      transfer control to generalized overflow
     ----------------------------------------------------------------------
     001   000   000
                       000   001
     ----------------------------------------------------------------------
     002   11    000                 \
                       31    060      |
     003   31    120                  } instructions ——> 000-239
                       30    180      |
     004   81    000                  |
                       U     008     /
     ----------------------------------------------------------------------
     005   B     007                 \
                       A-    001      |
     006   C     007                  } generalized overflow subroutine
                       00    000      |
     007  [CCC   CCC           /      |
                       CCC   CCC]    /
     ----------------------------------------------------------------------
 1   008   F     193
                       B     189
     009   E     194
                       F     195
     010   E     183
                       C     820
     011   B     190
                       E     185
     012   F     193
                       E     194
     013   C     821
                       F     082
     014   B     196
                       E     184
     015   H     822
                       E     186
     016   C     823
                       50    197      R̸2-way△ merge          \
     017   50    198                  R̸MOUNT△ TAPES           |
                       50    820      R̸△ △ Axxx△ ON△x         |
     018   50    822                  . △ ALT△ W/xi̸i̸i̸          } ——> SCP
                       50    821      R̸ △△ Bxxx △ON△ x        |
     019   50    823                  . △ ALT△ W/xi̸i̸i̸          |
                       50    199      R̸ △△ BLANKS△ ON        /
```

```
        020   50    217
                           00    000      Δ6Δ ANDΔ 1.~~RRR~~
        021   R     142                   •h_1
                           U     141      transfer control to connector 21
        ----------------------------------------------------------------------
        022   L     000
   (2)                     Q1    156      force transfer for rerun
        ----------------------------------------------------------------------
        023   R     026                   •α1
                           U     098      transfer control to connector 16
        ----------------------------------------------------------------------
        024   R     026                   •α2
   (3)                     U     065      transfer control to connector 18
        ----------------------------------------------------------------------
        025   B     218
(5)-(α)                    C     026      •α3
        ----------------------------------------------------------------------
        026   [CCC  CCC
                           CCC   CCC]
  (10)  027   00    000
                           T     029      transfer control if A_6^ρ > B_6^ρ
        ----------------------------------------------------------------------
        028   [12   000                   T_a ⟶ rI
                           U     030]     transfer control to connector 11
        ----------------------------------------------------------------------
        029   [14   000                   T_b ⟶ rI
                           00    000]
        ----------------------------------------------------------------------
        030   B     880
  (11)                     L     940
        031   00    000
                           T     073      transfer control if A_i^ρ > B_j^ρ
        ----------------------------------------------------------------------
   (γ)  032   [Y    880                   A_i ⟶ ws
                           00    000]
        ----------------------------------------------------------------------
        033   R     108                   •δ2
                           U     106      transfer control to connector 12
        ----------------------------------------------------------------------
        034   [CCC  CCC                 }
  (δ2)                     CCC   CCC]   } i + 1 ⟶ i   (overflow if i = 6)
        035   B     034                 }
                           A*    219
        ----------------------------------------------------------------------
        036   C     034
                           U     030      transfer control to connector 11
        ----------------------------------------------------------------------
        037   B     220
                           C     034      1 ⟶ i
  (17)  038   30    880                   rI ⟶ A
                           B     191
        039   A01   195                 }
                           C11   191    } Δa + 1 ⟶ Δa
                                        }
```

```
     040   B     930
                             L     230
     041   00    000
                             Q     043        transfer control if $A_6^0$ = Z
------------------------------------------------------------------------------
     042   00    000
                             U11   026        transfer control to connector 5
------------------------------------------------------------------------------
     043   B     931
                             L     191
     044   00    000
                             Q     054        transfer control if $A_6^1$ = Δa
------------------------------------------------------------------------------
     045   50    221                          BLOCK Δ COUNT }
(14)                         50    222        Δ OFF-RERUN   } → SCP
     046   F     157
                             B     050
     047   E     183
                             H     050
     048   E     185
                             H     051
     049   E     187
                             C     052
     050  [800   000
                             000   000]   }
     051  |CCC   CCC                      }
                             CCC   CCC|   }  rewind with interlock $T_a$, $T_b$, $T_c$
     052  [CCC   CCC                      }
                             CCC   CCC]   }
     053   90    000                          stop
                             00    000
------------------------------------------------------------------------------
     054  [82    000                          rewind with interlock $T_a$
                             B     939]
     055   L     223
                             Q     144        transfer control if $A_6^9$ = ₱
------------------------------------------------------------------------------
     056   K     000
                             C     191        0 → Δ a
     057   F     183
                             B     184    }
     058   G     184                      }
                             H     183    }
     059   F     224                      }
                             E     028    }  interchange $T_a$ uniservos
     060  H    028
                     E    054
     061  H    054
                     E    065
     062  H    065
                     E    068
     063  C    068
                     B    189
     064  A-   225                La + 1 ⟶ La
                     C    189
---------------------------------------------------------------
(18) 065  [12  000                Ta ⟶ rI
                     32   880]    rI ⟶ A, Ta ⟶ rI
     066  B    880
                     L    189
     067  00   000
                     Q    037     transfer control if A_1^0 = La
---------------------------------------------------------------
     068  [B2  000                rewind T_a with interlock
                     B    226]
     069  F    216
                     E    183
     070  C    881
                     50   881     R̸Ta△LABEL△SH  }
     071  50   227                OULD△BE--i̸i̸i̸ } → SCP
                     50   189     R̸△△Axxx△△△△△  }
     072  30   880                rI ⟶ A
                     U    065
---------------------------------------------------------------
     073  Y    940                B_j ⟶ ws
                     00   000
     074  R    108                •δ1
                     U    106     transfer control to connector 12
---------------------------------------------------------------
(δ1) 075  [CCC CCC
                     CCC  CCC]
     076  B    075
                     A*   219     j + 1 ⟶ j (overflow if j = 6)
---------------------------------------------------------------
     077  C    075
                     U    030     transfer control to connector 11
---------------------------------------------------------------
(15) 078  B    228
                     C    075     1 ⟶ j
     079  30   940                rI ⟶ B
                     B    192
     080  A-   195                      } $\Delta b + 1 \longrightarrow \Delta b$
                         C     192      }
     081  B    990
                         L     230
     082  00   000
                         Q01   084      transfer control if $B_6^0 = Z$
     ------------------------------------------------------------------------
     083  00   000
                         U     026      transfer control to connector 5
     ------------------------------------------------------------------------
     084  B    991
                         L     192
     085  00   000
                         Q     087      transfer control if $B_6^1 = \Delta b$
     ------------------------------------------------------------------------
     086  00   000
                         U     045      transfer control to connector 14
     ------------------------------------------------------------------------
     087 [84   000                      rewind $T_b$ with interlock
                         B     999]
     088  L    223
                         Q     026      transfer control if $B_6^9 = ₱$
     ------------------------------------------------------------------------
     089  K    000
                         C     192      $0 \longrightarrow \Delta b$
     090  F    185
                         B     186    }
     091  G    186                    }
                         H     185    }
     092  F    224                    }
                         E     029    }
     093  H    029                    }
                         E     087    }  interchange $T_b$ uniservos
     094  H    087                    }
                         E     098    }
     095  H    098                    }
                         E     101    }
     096  C    101                    }
                         B     190    }
     097  A-   225                        $Lb + 1 \longrightarrow Lb$
                         C     190
     ------------------------------------------------------------------------
(16) 098 [14   000
                         34    940]
     099  B    940
                         L     190
```

```
      100  000   000
                          Q    078       transfer control if $B_1^0$ = Lb
--------------------------------------------------------------------------
      101 [84    000
                          B    226 ]     rewind Tb with interlock
      102  F     216
                          E    185
      103  C     991
                          50   991       ~~R~~Tb ΔLABELΔ SH  }
      104  50    227                     OULDΔ BE--~~iii~~   } ⟶ SCP
                          50   190       ~~R~~ΔΔ BxxxΔΔΔΔΔ   }
      105  30    940                     rI ⟶ B
                          U    098
--------------------------------------------------------------------------
(12)  106 [Z4    820                     ws ⟶ $C_k$
                          B    106 ]
      107  A*    219                     k + 1 ⟶ k (overflow if k = 6)
                          C    106
--------------------------------------------------------------------------
(δ)   108 [CCC   CCC
                          CCC  CCC ]
--------------------------------------------------------------------------
(19)  109  B     229
                          C    106       1 ⟶ k
      110 [56    820                     C ⟶ $T_c$
                          B    231 ]
      111  A-    195                   } Δc + 1 ⟶ Δc
                          H    231
      112  L     232
                          Q    114       transfer control if  c = 1981
--------------------------------------------------------------------------
      113  00    000
                          U    108       transfer control to δ
--------------------------------------------------------------------------
      114  F     230
                          G    820       Z ⟶ $C_1^0$
      115  G     870                     Z ⟶ $C_6^0$
                          A-   195
      116  C     871                     Δc + 1 ⟶ $C_6^1$
                          56   820       C ⟶ $T_c$
(20)  117  B     930
                          L    990
      118  F     042
                          T    120       transfer control if $A_6^0 > B_6^0$
--------------------------------------------------------------------------
      119  B     028
                          U    121       •$\beta_1$
--------------------------------------------------------------------------
```

```
     120   B     029                          ·β2
                          00    000
     ------------------------------------------------------------------
     121   .6    000
                          C     162
     122   B     235
                          E     187
     123   H     128
                          F     210
     124   E     233
                          C     821
     125   50    821                          R̸LABELΔ Tc---    }
                          50    200           R̸ ΔΔCxxxΔΔΔΔΔ    }→ SCP
     126   B     200                        }
                          A-    225         } Lc + 1 → Lc
     127   C     200                        }
                          C     231           0 → Δc
     ------------------------------------------------------------------
     128 [ CCC   CCC
                          CCC   CCC ]
     129   A*    234                          Memory → Tc
                          C     128
     ------------------------------------------------------------------
     130   00    000
                          U     128
     ------------------------------------------------------------------
     131 [86     000                          Rewind Tc with interlock
                          B     188]
     132   F     187
                          G     188  }
     133   H     187                 }
                          F     224  }
     134   E     110                 }
                          H     110  }
     135   E     116                 }        Interchange Tc servos
                          H     116  }
     136   E     131                 }
                          H     131  }
     137   E     141                 }
                          H     141  }
     138   E     152                 }
                          H     152  }
     139   E     153                 }
                          C     153  }
     140   C     142                          ·η2
                          00    000
     ------------------------------------------------------------------
(21) 141 [76     200                          Lc → C⁰₁,  C → Tc
                          00    000]
(η)  142 [CCC    CCC
                          CCC   CCC]
     ------------------------------------------------------------------
(η2) 143   00    000
                          U     108           transfer control to δ
     ------------------------------------------------------------------
```

144	R	032			$\cdot\Upsilon 2$
			U	026	transfer control to 5
145	L	230			
			Q	147	transfer control if $A_1^0 = Z$
146	Y	880			$A_i \longrightarrow ws$
			U	033	
147	Y	230			$Z \longrightarrow rY$
			00	000	
148	R	107			
			U	106	transfer control to do $Z \longrightarrow C_k^0$
149	Z	870			$Z \longrightarrow C_6^0$
			B	231	
150	A-	195			
			C	871	$\Delta c + 1 \longrightarrow C_6^1$
151	B	223			
			C	879	$\not P \longrightarrow C_6^0$
152	[56	820			$C \longrightarrow T_c$
			Z	820]	$Z \longrightarrow C_1^0$
153	[56	820			$C \longrightarrow T_c$
			B	187]	
154	R	126			
			U	123	transfer control to do $Lc \longrightarrow SCP$
155	50	236			R̸END MERGE.R̸ $\longrightarrow$ SCP
			U	046	tran:fer control to RWD* tapes
156	B	189			$La \longrightarrow L$
			L	191	$\Delta a \longrightarrow \Delta$
157	F10	183			$a \longrightarrow t$
			00	000	
158	R	182			$\cdot\theta_1$
			U	164	transfer control to 7
159	B	190			$Lb \longrightarrow L$
			L	192	$\Delta b \longrightarrow \Delta$
160	F	185			$b \longrightarrow t$
			00	000	
161	R	182			θ_2
			U	164	transfer control to 7
162	[CCC	CCC			β_1: 00 000 1a 000
			CCC	CCC]	β_2: 00 000 1b 000
163	00	000			
			U	108	transfer to control to δ

```
(7)
164  C     237
                  J     238
165  G     239
                  F     039
166  B     239
                  E     169
167  H     169
                  H     178
168  E     172
                  C     172
---------------------------------------------------------------
169 [1t    000
                  30    820]      Tt ──→ rI
170  B     820                     rI ──→ C
                  L     237
171  00    000
                  Q     177       transfer control if C⁰₁ = L
---------------------------------------------------------------
172 [8t    000                     RWD Tt with interlock
                  B     226]
173  F     216
                  E     239
174  C     821
                  50    821       RTtΔLABELΔSH  }
175  50    227                     OULDΔBE--iii   } ──→ SCP
                  50    237       R ΔΔLxxxΔΔΔΔΔ  }
176  00    000
                  U     169
---------------------------------------------------------------
177  K     000
                  C     237       0 ──→ g
---------------------------------------------------------------
178 [1t    000                     Tt ──→ rI
                  30    820]      rI ──→ C
179  B     237
                  A-    195       } g + 1 ──→ g
180  H     237
                  K     000
181  B     238
                  T     178       transfer control if g= Δ
---------------------------------------------------------------
182 [CCC   CCC
                  CCC   CCC]
---------------------------------------------------------------
183 [222   222                     } a
                  222   222]
184 [333   333                     } alternate a
                  333   333]
185 [444   444                     } b
                  444   444]
186 [555   555                     } alternate b
                  555   555]
187 [666   666                     } c
                  666   666]
188 [111   111                     } alternate c
                  111   111]
```

```
189 [ɌΔΔ   A00                  }La
                 1ΔΔ  ΔΔΔ ]
190 [ɌΔΔ   B00                  }Lb
                 1ΔΔ  ΔΔΔ ]
191 [000   000                  }a
                 000   000]
192 [000   000                  }b
                 000   000]
193  000   000
                 011   110
194  000   000
                 0Δ0   NΔ0
195  000   000
                 000   001
196  .ΔA   LTΔ
                 W/0   ɨɨɨ
197  Ɍ2-   WAY
                 ΔME   RGE
198  ɌMO   UNT
                 ΔTA   PES
199  ɌΔΔ   BLA
                 NKS   ΔON
200  ɌΔΔ   C00                  }Lc
                 1ΔΔ   ΔΔΔ
----------------------------------------
201  11    000
                 30    100
202  B     100
                 L     009
203  00    000
                 Q     005
----------------------------------------
204  00    000
                 U     001
----------------------------------------
205  21    000
                 30    100
206  Y     010
                 Z     990
207  L     008
                 U     990
----------------------------------------
208  31Z   960
                 B00   991
209  R2W   007
                 UMG   005
----------------------------------------
210  111   111
                 C11   111
----------------------------------------
211  31Z   000
                 B     991
212  A-    996
                 Q     994
----------------------------------------
213  C     991
                 U     991
----------------------------------------
```

```
214  81   000
                 00   000
215  30   960
                 U    008
```

```
216  001  060
                 000  000
217  Δ6Δ  AND
                 Δ1.  R̸R̸R̸
218  B00  930
                 L00  990
219  010  010
                 000  000
220  Y40  890
                 Z00  880
221  R̸BL  OCK
                 ΔCO  UNT
222  ΔOF  F-R
                 FRU  NR̸R̸
223  P̸P̸P̸  P̸P̸P̸
                 P̸P̸P̸  P̸P̸P̸
224  101  111
                 101  111
225  000  000
                 100  000
226  R̸Ta  ΔLA
                 BEL  ΔSH
227  OUL  DΔB
                 E--  1̸1̸1̸
228  Y40  950
                 Z00  940
229  Z40  820
                 B00  106
230  ZZZ  ZZZ
                 ZZZ  ZZZ
231 [000  000
                 000  000]      } Δc
232  000  000
                 001  982
233  R̸LA  BEL
                 ΔTc  ---
234  001  000
                 000  060
235  B83  128
                 500  000
236  R̸EN  DΔM
                 ERG  E.R̸
237 ┌ -
                       -  ┐   ┐
238 │ -                   │   │
                       -  │   ├  working storage
239 └ -                   │   │
                       -  ┘   ┘
```

6. Reruns with Two Outputs

The rerun procedure for multiple outputs described below is very similar to the single output rerun previously described. The only complexities introduced are the problems of repositioning one of the output tapes and of differentiating between the output controlling the rerun and the tape that must be repositioned.

The latter problem is dealt with by writing on the identification block of every output tape the letter identifying the output file. When the rerun is initiated, this information is made available to the rerun routine by the memory dump locator (also in the identification block).

The output tape which is not being rerun is treated as an input tape for repositioning purposes.

During the repositioning, all tapes are read into the output block of the file being rerun. The contents of the input and the other output block at the end of the rerun routine are the same as they were at the time the memory dump was written. rI is filled by the read subroutine of the normal instructions.

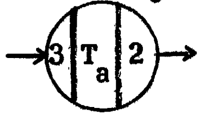
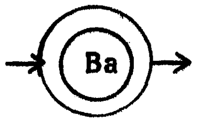
It is to be noted that the rerun must occur at a point where the contents of the computer registers (except rI) are of no consequence as the rerun makes no provision for restoring them. **(Please refer to the coding in Section 5.)**

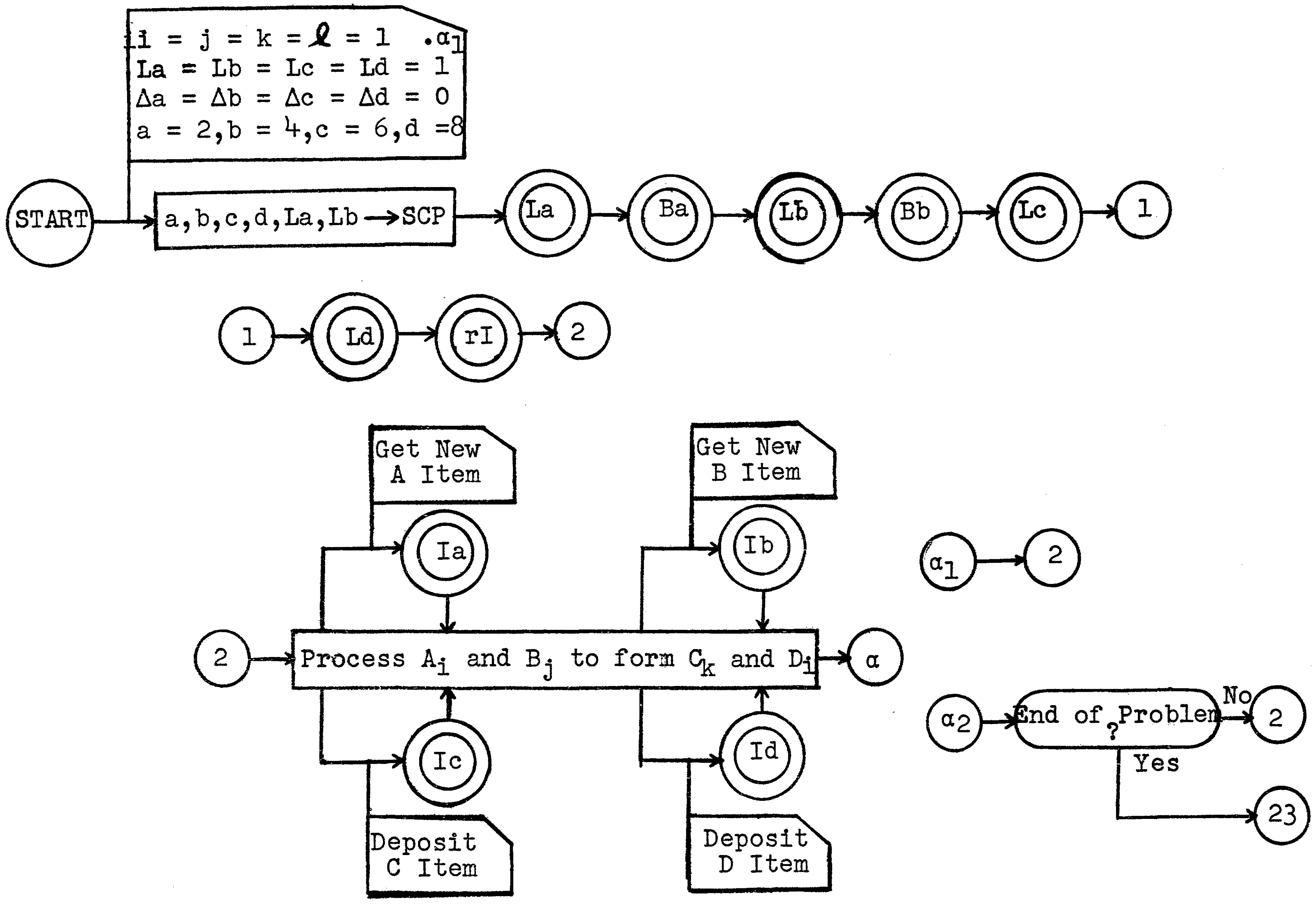
Unlike the single output rerun this routine requires no breakpoint option to rerun. Breakpoint options represent a manual operation and, therefore, a possible operator error. Since one of the output tapes contains good information, a manual error can ruin this tape and necessitate rerunning from an earlier point.

To rerun, the operator merely mounts the last good output tape of the file and does an initial read. If this is done correctly, no other manual operations are required and, therefore, no further possibility of error exists.

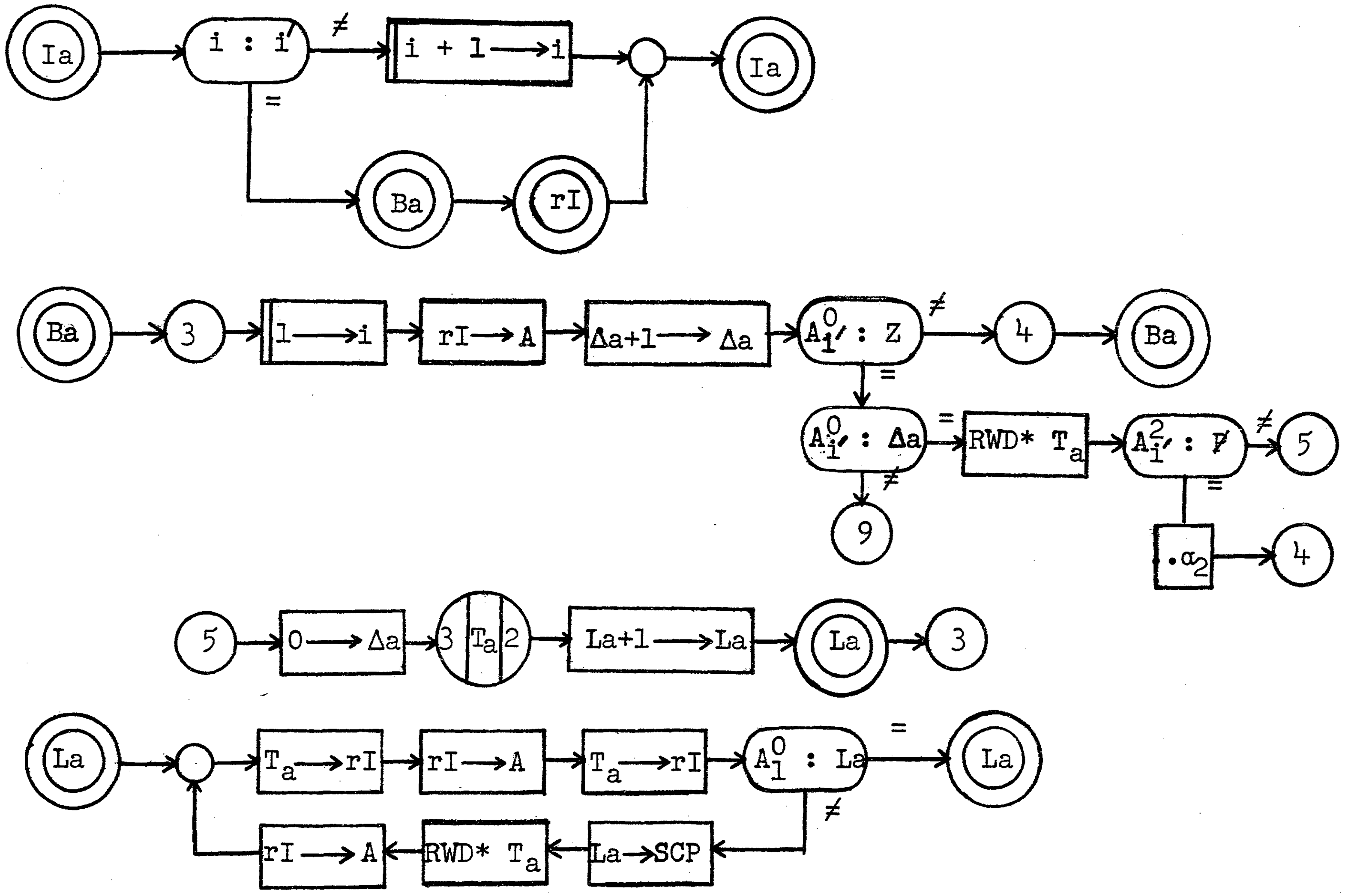
Explanation of Flow Chart Symbols

T_a	Tape mounted on Uniservo a
$T_a \longrightarrow rI$	A block from T_a is read into input buffer (forward)
$rI \longrightarrow A$	The block in the input buffer is transferred to A
A_i	The ith item comprising the block A ($i = 1,2,\ldots,i'$)
A_i^o	The keyword field of item A_i
La	The label identifying the current A file reel being processed.

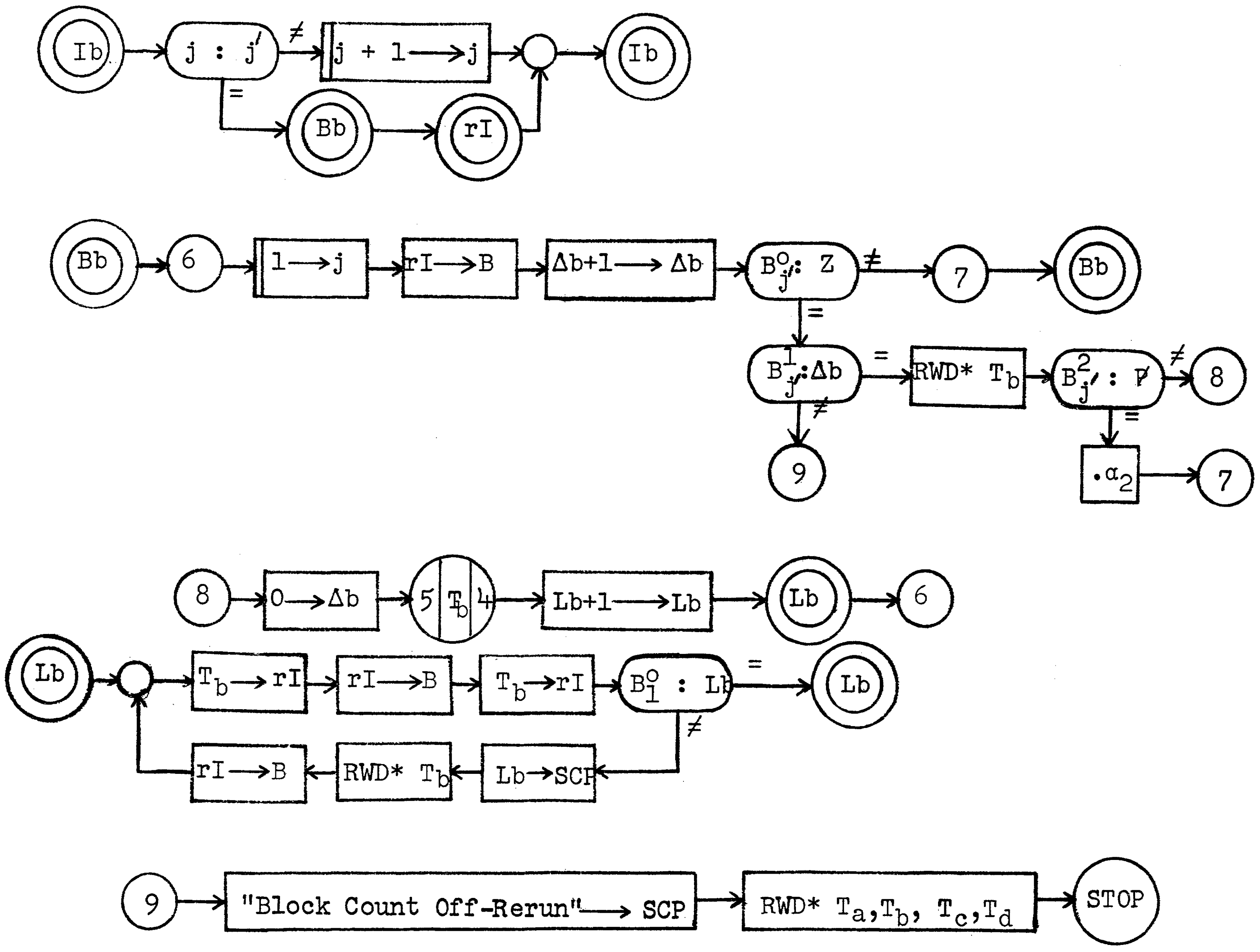
Δa	The number of A blocks processed on T_a.
$C \longrightarrow T_c$	The C block is written on T_c.
$C_I^0 \equiv C$	The symbol C_I^0 is made identical with the symbol C.
RWD* T_c	Tape C is rewound and an interlock set preventing further tape motion until the tape reel is removed.
	Servo interchange symbol. a which initially read 2 is changed to 3 and the symbol rotated 180^o.
	Do operations specified by subroutine Ba.
SCP	Supervisory Control Printer



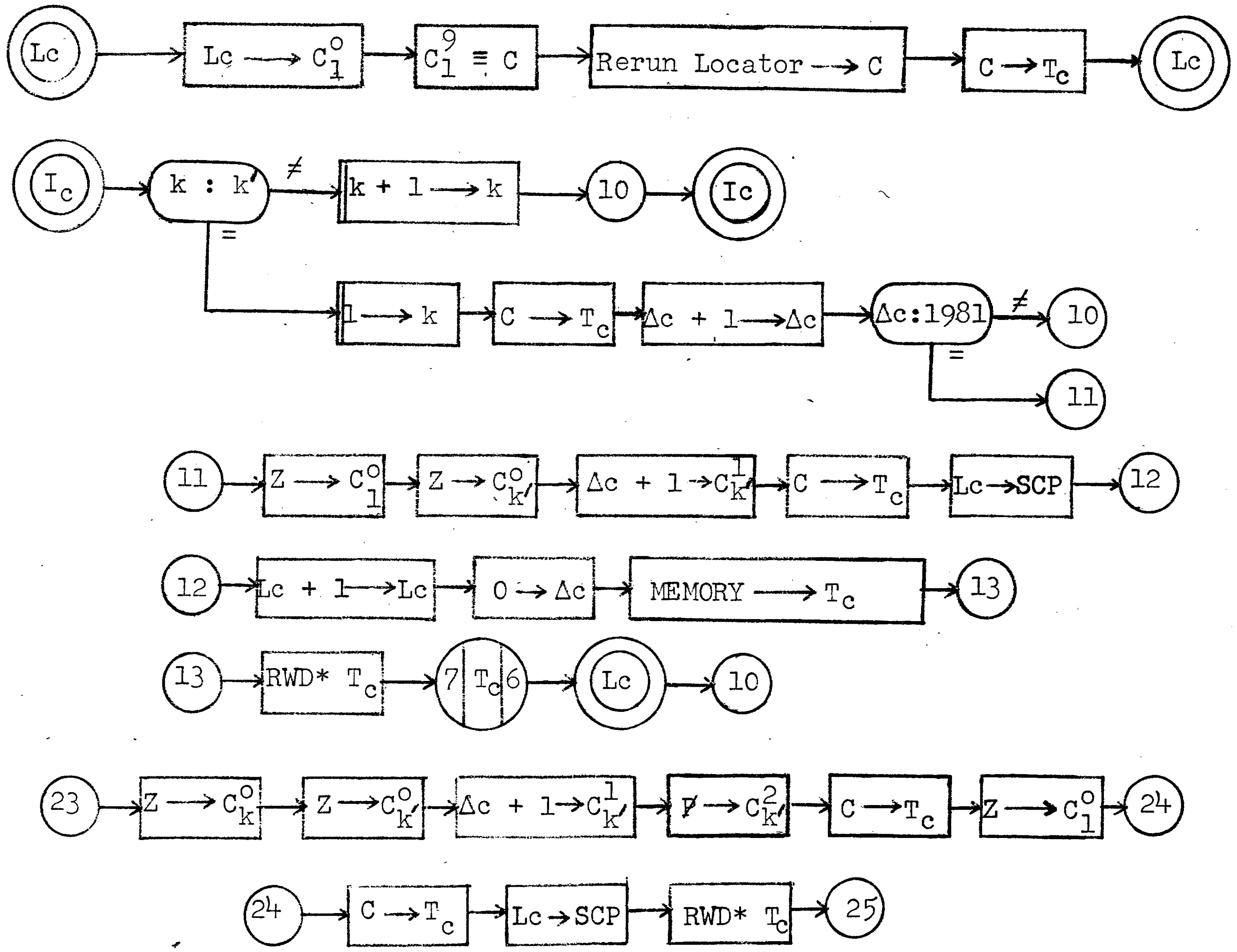
i = j = k = ℓ = 1 .α1
La = Lb = Lc = Ld = 1
Δa = Δb = Δc = Δd = 0
a = 2, b = 4, c = 6, d = 8
START
a, b, c, d, La, Lb → SCP
La
Ba
Lb
Bb
Lc
1
1
Ld
rI
2
Get New A Item
Get New B Item
Ia
Ib
α1
2
2
Process A_i and B_j to form C_k and D_l
α
Ic
Id
α2
End of Problem?
No
2
Yes
23
Deposit C Item
Deposit D Item

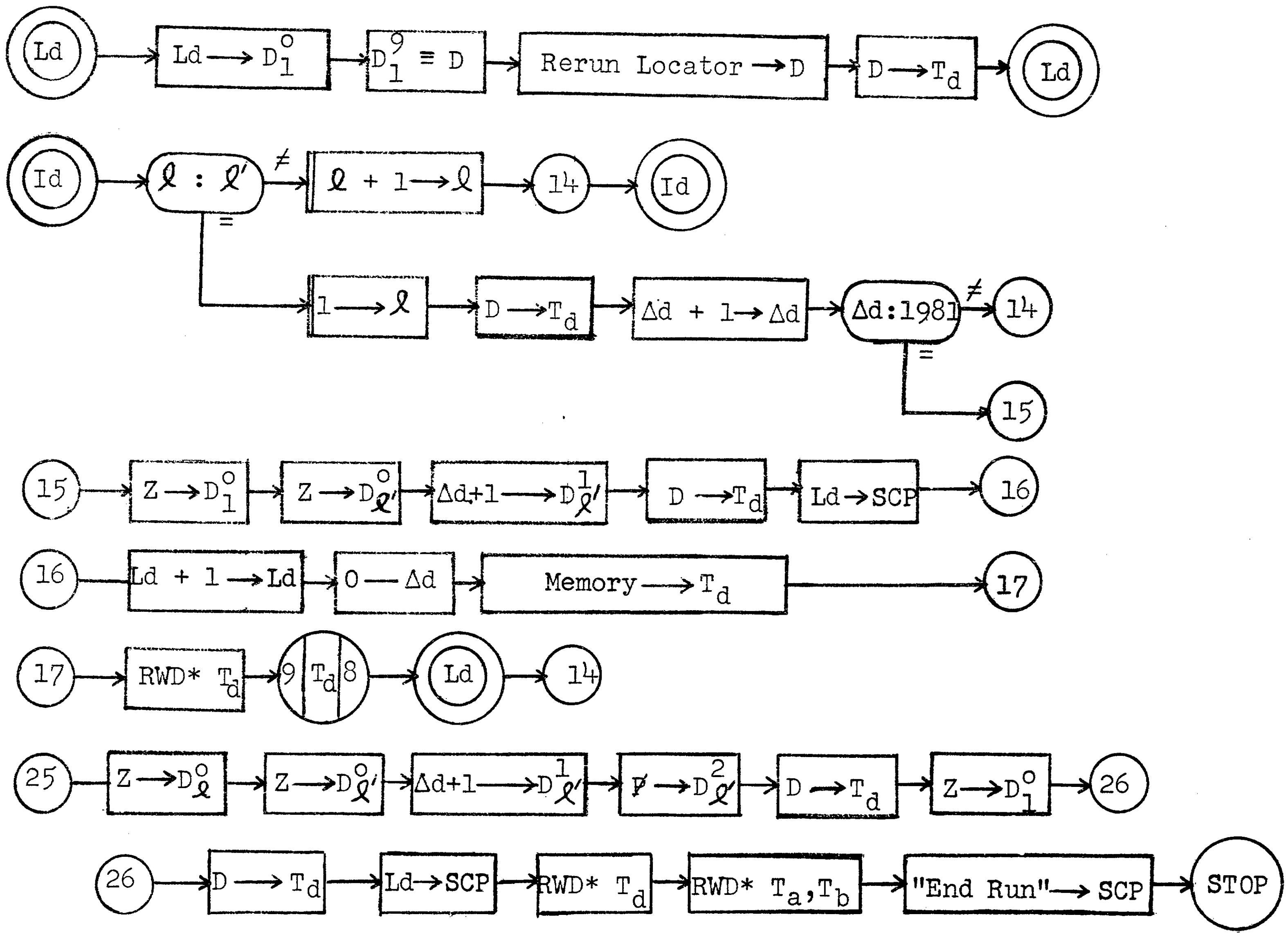


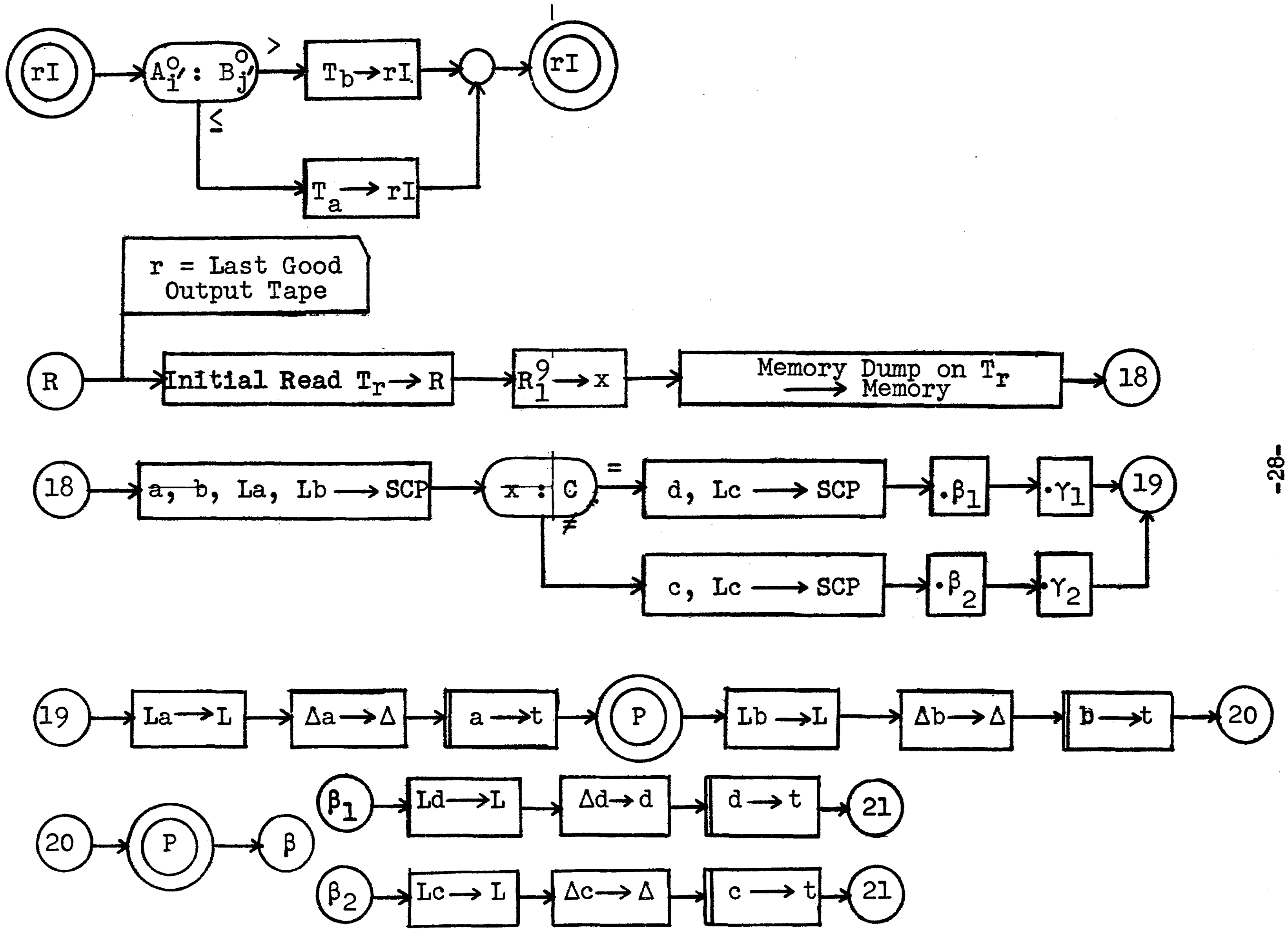
Ia
i : i′
≠
=
i + 1 → i
Ia
Ba
rI
Ba
3
1 → i
rI → A
Δa+1 → Δa
$A^0_{i'}$: Z
≠
4
Ba
=
$A^0_{i'}$: Δa
=
RWD* T_a
$A^2_{i'}$: P′
≠
5
≠
9
=
$\cdot\alpha_2$
4
5
0 → Δa
3 T_a 2
La+1 → La
La
3
La
T_a → rI
rI → A
T_a → rI
A^0_1 : La
=
La
≠
rI → A
RWD* T_a
La → SCP

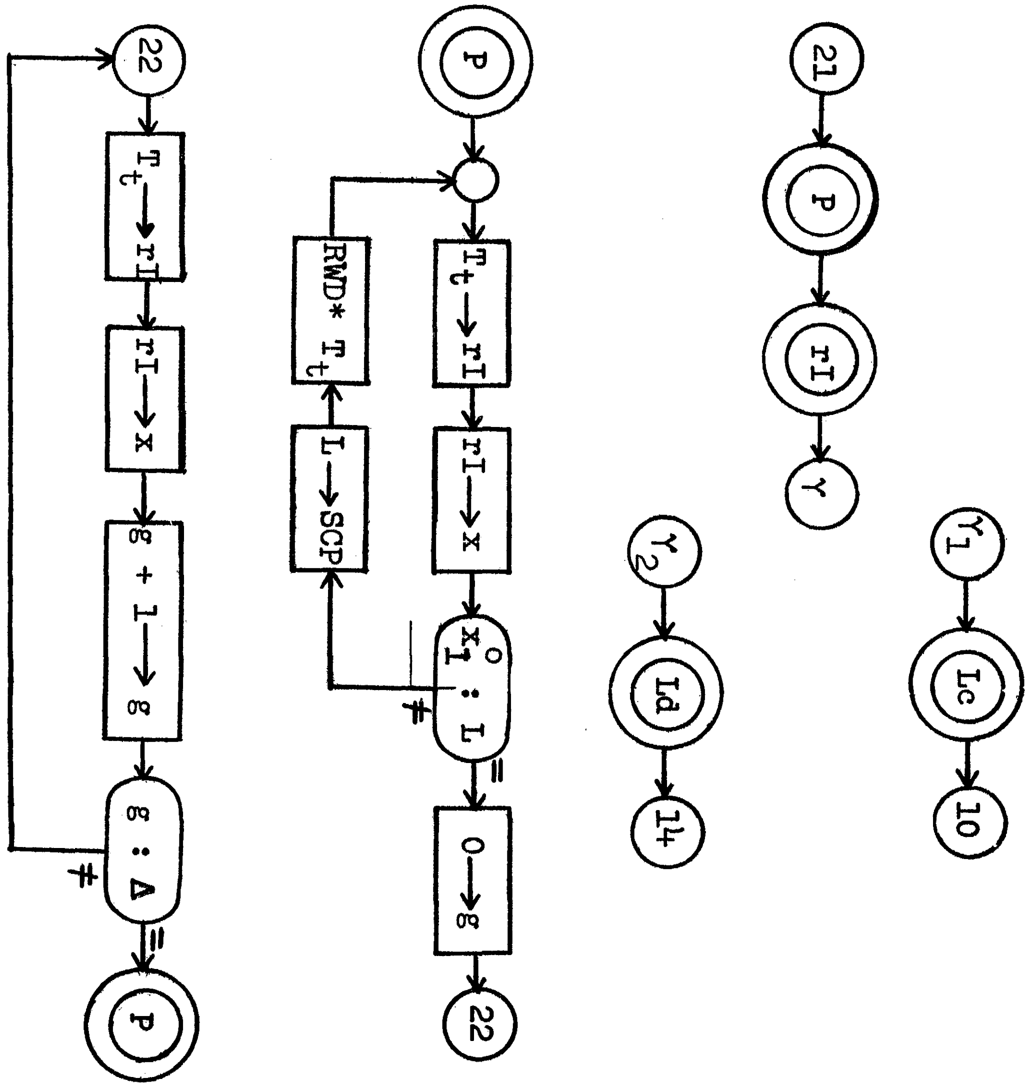


Ib
j : j'
≠
j + 1 → j
=
Bb
rI
Ib
Bb
6
1 → j
rI → B
Δb+1 → Δb
B°j' : Z
≠
7
Bb
=
B¹j' : Δb
=
RWD* Tb
B²j' : P
≠
8
=
≠
9
·α2
7
8
0 → Δb
5 Tb 4
Lb+1 → Lb
Lb
6
Lb
Tb → rI
rI → B
Tb → rI
B°1 : Lb
=
Lb
≠
rI → B
RWD* Tb
Lb → SCP
9
"Block Count Off-Rerun" → SCP
RWD* Ta, Tb, Tc, Td
STOP


Lc
Lc → C₁⁰
C₁⁹ ≡ C
Rerun Locator → C
C → Tc
Lc
Ic
k : k′
≠
k + 1 → k
10
Ic
=
1 → k
C → Tc
Δc + 1 → Δc
Δc:1981
≠
10
=
11
11
Z → C₁⁰
Z → Cₖ′⁰
Δc + 1 → Cₖ′¹
C → Tc
Lc → SCP
12
12
Lc + 1 → Lc
0 → Δc
MEMORY → Tc
13
13
RWD* Tc
7 Tc 6
Lc
10
23
Z → Cₖ⁰
Z → Cₖ′⁰
Δc + 1 → Cₖ′¹
F → Cₖ′²
C → Tc
Z → C₁⁰
24
24
C → Tc
Lc → SCP
RWD* Tc
25


Ld
Ld → D$_1^0$
D$_1^9$ ≡ D
Rerun Locator → D
D → T$_d$
Ld
Id
ℓ : ℓ′
≠
ℓ + 1 → ℓ
14
Id
=
1 → ℓ
D → T$_d$
Δd + 1 → Δd
Δd:1981
≠
14
=
15
15
Z → D$_1^0$
Z → D$_{ℓ'}^0$
Δd+1 → D$_{ℓ'}^1$
D → T$_d$
Ld → SCP
16
16
Ld + 1 → Ld
0 — Δd
Memory → T$_d$
17
17
RWD* T$_d$
9 T$_d$ 8
Ld
14
25
Z → D$_ℓ^0$
Z → D$_{ℓ'}^0$
Δd+1 → D$_{ℓ'}^1$
F → D$_{ℓ'}^2$
D → T$_d$
Z → D$_1^0$
26
26
D → T$_d$
Ld → SCP
RWD* T$_d$
RWD* T$_a$,T$_b$
"End Run" → SCP
STOP


rI
$A_i^{o\prime} : B_j^{o\prime}$
>
≤
$T_b \rightarrow rI$
$T_a \rightarrow rI$
rI
r = Last Good Output Tape
R
Initial Read $T_r \rightarrow R$
$R_1^9 \rightarrow x$
Memory Dump on T_r Memory
18
18
~~a, b,~~ La, Lb → SCP
x : C
=
≠
d, Lc → SCP
$\cdot\beta_1$
$\cdot\gamma_1$
19
c, Lc → SCP
$\cdot\beta_2$
γ_2
19
La → L
Δa → Δ
a → t
P
Lb → L
Δb → Δ
b → t
20
20
P
β
β_1
Ld → L
Δd → d
d → t
21
β_2
Lc → L
Δc → Δ
c → t
21


21
P
rI
γ
γ1
Lc
10
γ2
Ld
14
P
$T_t \rightarrow rI$
$rI \rightarrow x$
$x_1^0 : L$
=
≠
$0 \rightarrow g$
22
RWD* T_t
$L \rightarrow SCP$
22
$T_t \rightarrow rI$
$rI \rightarrow x$
$g + 1 \rightarrow g$
$g : \Delta$
≠
=
P

EFFICIENT CODING TECHNIQUES

1. Introduction

Many data processing applications of the UNIVAC System involve routines that are to be used repeatedly or which must process large volumes of data. It is frequently desirable to reduce the running time of these routines. This can be accomplished by:

1. Reducing the number of instructions executed by the computer. (Linear Coding)

2. Reducing the time spent in execution of the individual instructions. (Minimum Latency Coding)

In the discussion to follow we will consider techniques to accomplish these savings in time. It should be noted, however, that maximum success in reducing the running time of an average routine can be achieved only by increasing the amount of memory space devoted to the program.

2. Linear Coding

The instructions of a typical program can be considered to belong in one of the following categories:

1. Instructions for starting and ending the program.

2. Instructions which read and write blocks of data on tape.

3. Instructions which move items to (or from) working storage from (or to) data blocks.

4. Instructions which process the item.

Instructions in the first category are executed only once per run and, therefore, no effort need be exerted to optimize them. For a discussion of instructions in the second category the reader is referred to the paper "A Survey of Input-Output Techniques". The following will, therefore, be restricted to a discussion of the instructions in categories three and four.

Let us assume first that a particular sequence of instructions has been developed which will process an item. There are then three choices available in the development of the routines which will process a block of such items. These are:

1. Repetition of the instructions for processing an item with the addresses modified for each separate item in the block.

2. An auxiliary routine to modify the addresses of the processing instructions.

3. An auxiliary routine to transfer successive items into a working storage from which the item is processed.

Clearly the first of these alternatives will involve the execution of the fewest number of instructions since every instruction pair executed will be processing items. Note, however, that it will also require the most memory space.

Where the available memory space does not permit using straight line processing instructions (method 1 above), a choice will have to be made between methods 2 and 3. In general, it has been found more efficient to transfer items to working storage than to modify processing instructions; there are some exceptions to this rule. These exceptions occur generally where "odd" sized items are being processed which cannot be transferred very efficiently within the computer and where the processing instructions have very few references to the memory location of the item.

The reader should note that straight line techniques are also available for the movement of items into or out of working storage. The following examples will illustrate this:

Example 1: The R-U Counter

```
100   R  100
                 U  xxx     To transfer the first item
101   R  100
                 U  xxx     To transfer the second item
102   R  100
                 U  xxx     To transfer the third item
 .       .
 .       .          .
                    .
      R  100
                 U  xxx     To transfer the last item
      B  xxx
                 C  100     Reset line 100
         -
                    -       End of block routine
```

Example 2: The Function Table

```
100   B  101
                 A  102
101   C  101
                 U  104
102   000000
                 000004
 .       .
                   .
104   x  xxx
                 x  xxx     Transfer the first item
 .       .
                    .
108   x  xxx
                 x  xxx     Transfer the second item
 .       .
                    .
```

```
112    x  xxx
                 x  xxx      Transfer the third item
 .         .
                    .
       x  xxx
                 x  xxx      Transfer the last item
 .         .
                    .
       B  xxx
                 C  101      Reset line 101
           -
                    -        End of block routine
```

Both of the methods illustrated are perfectly general although the method in example 1 requires more memory space than does the method of example 2. The number of lines of instructions executed in moving each item is the same* for both. Example 2 has a disadvantage in some cases (as we shall see in the discussion on minimum latency coding) in that it requires that each set of coding handling successive items be stored in the memory with a fixed spacing between the addresses.

A further example will aid in fixing these principles in the reader's mind.

Example 3

Select from the four-word stock items on T_a all those items for which the expected requirements is less than the sum of the on hand and on order and write them on T_b. The item format is:

```
SSSSSS RRRRRR
HHHHHH 000000
-other  data-
-other  data-
```

where:

SSSSSS	is the stock number
RRRRRR	is the expected requirements
HHHHHH	is the on hand amount
000000	is the on order amount

The coding which will process one item is in cells 800 - 803. The coding is shown below.

```
010    F  100
                 B  800
011    E  101
                 K  000
012    B  801
                 E  101
013    C  102
                 X  000
```

*One line less is used in the R-U Counter than in the Function Table method in transferring the first item only.

```
014   .6 000
                A  102
015   R   xxx                Set exit of output routine
                T  098       Transfer control if OH + 00 > R
 .     .
 .     .          .
 .     .          .
                  .
098   F  803
                B  802
099   V  800
                U  xxx       Transfer control to output routine
100   111111
                000000
101   000000
                000000
102      -
                  -          Temporary storage
```

This routine requires that at least 6 lines of coding be executed for each item. Two extra lines must be provided for the item in case it is to be placed on the output tape. A block of such items might be processed as follows:

If linear coding were to be used, a total of 122 lines of coding (15x8 plus the two constants in cells 100 and 101) would have to be stored in the memory exclusive of the output routine. The total instruction pairs executed per block would be only 90 (6x15) plus 2 for each output item.

If an auxiliary routine were provided to modify the addresses of the instructions on lines 010, 012, 098, and 099 so as to be able to process the succeeding items, the routine would begin on line 016 and might look as follows:

```
016   B  010
                A* 097
017   C  010
                B  012
018   A  096
                C  012
019   B  098
                A  095
020   C  098
                B  099
021   A  096
                C  099
022   00 000
                U  010
023   B  091                 From overflow on line 016
                C  010
024   B  094
                C  012
025   V  092
                W  098
```

```
026      -
                -           End of block routine
 .       .
 .       .      .
 .       .      .
                .
091    F40100
              B  800
092    F  803
              B  802
093    V  800
              U  xxx
094    B  801
              E  101
095    000004
              000004
096    000004
              000000
097    004000
              000004
```

Here the total number of lines required for storage is 27, but the total instruction pairs executed per block of data is (including 4 for generalized overflow) 196 exclusive of 2 lines for each output item.

Another alternative is to provide an iterative routine that will move items into a working storage, such as:

```
016    V40804
              W  800
017    V  806
              W  802
018    B  016
              A* 095
019    C  016
              B  017
020    A  095
              C  017
021    00 000
              U  010
022    V  096                From overflow on line 018
              W  016
023      -                   End of block routine
                -
 .       .
 .       .      .
 .       .      .
                .
095    004004
              000000
096    V40804
              W  800
097    V  806
              W  802
```

This method requires only 20 lines of storage. A total of 182 lines of coding are executed for each block of data exclusive of output.

A last possibility is to again make use of the working storage principle, but uses linear coding for the routine transferring each item to the working storage. The following is an example using the function table technique:

```
016   B  017
                A  096
017   C  017
                U  018
018   V  804
                W  800
019   V  806
                W  802
020   00 000
                U  010
021   V  808
                W  800
022   V  810
                W  802
023   00 000
                U  010
024   V  812
                W  800
 .        .
 .        .       .
 .        .       .
                  .
057   V  856
                W  800
058   V  858
                W  802
059   00 000
                U  010
060   B  097
                C  017      Reset line 017
061       -
                  -         End of block routine
 .        .
 .        .       .
 .        .       .
                  .
096   000000
                000003
097   B  017
                A  096
```

This method requires 57 lines of coding to be stored, but only 163 instruction pairs are executed for each block of items processed (plus 2 lines for each output item).

These results are summarized in the following table. Only the total lines for storage or execution required in processing a block of input items are considered.

Method	Total Lines Stored (excluding Gen OF)	Total Lines Executed (No Output Items)
1. Linear Coding	122	90
2. Iterative Alteration of Instruction Addresses	27	196
3. Iterative Transfer to Working Storage	20	182
4. Linear Transfer to Working Storage	57	163

It is generally true, as the previous discussion has pointed out, that the minimum number of instructions to be executed per routine can be obtained by exchanging memory space for time. The relative merits of each method discussed depends to a large extent on the amount of processing required for each item and on the item size. Where memory space is not too critical, the programmer would do well to investigate linear or semi-linear (methods 1 and 4 of the above table) techniques for reducing problem running time.

3. Introduction To Minimum Latency

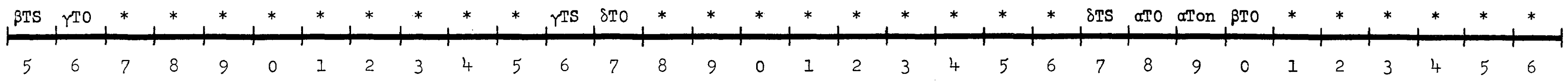
A study of UNIVAC logic reveals that the time required to execute an instruction pair falls into two categories:

1. Time spent in actual execution of the instructions.
2. Time spent in waiting for information to appear at the read out or erase gates of the memory.

The first of these is constant and fixed by the logic of the computer. The second of these, called latency time, depends for its duration upon the relation between the Time Selection Counter (TSC) and the Time Selection Digit (TSD) of the memory address involved. By judicious selection of memory locations, the latency time or a routine can be materially reduced.

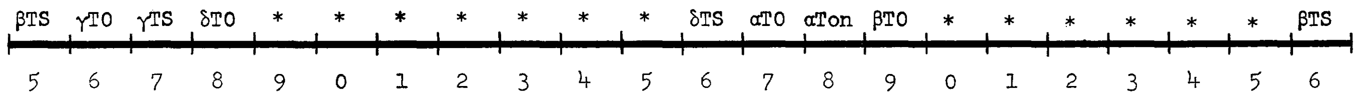
Consider first an instruction pair that does not involve a transfer of control. It is desirable to minimize the elapsed time between the selection of the instruction pair at address m and the selection of the next pair at address $m + 1$.

If the TSD of m is k, then the β Time Selection Minor Cycle occurs when the TSC is reading k. For the instruction pair in $m + 1$ this will occur at a TSC reading of $k + 1$. An instruction pair cannot be executed in one minor cycle; therefore, the minimum time between β time selection will be 11 minor cycles: ten minor cycles to step the TSC around to k and one more to $k + 1$. If the instruction pair is not executed in this time, additional time, in units of ten minor cycles (one Major Cycle), will elapse before the β Time Selection Minor Cycle of $m + 1$.


βTS γTO * * * * * * * * * γTS δTO * * * * * * * * * δTS αTO αTon βTO * * * * * *
5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6

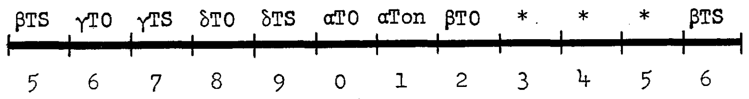
Total Time 31 MC
Latency Time 23 MC

Ratio L.T./Eff.T. 2.9:1


βTS γTO γTS δTO * * * * * * * δTS αTO αTon βTO * * * * * * βTS
5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6

Total Time 21 MC
Latency Time 13 MC

Ratio L.T./Eff.T. 1.6:1



Total Time 11 MC
Latency Time 3 MC

Ratio L.T./Eff.T .38:1

*Latency Time

The following examples will illustrate these concepts: (see Figure 1)

1.	025	B	056	L	057	31 MC
2.	025	L	057	B	056	21 MC
3.	025	L	057	B	059	11 MC

Example 2 interchanges the instructions in example 1 which results in substantial time savings. In example 3 a further increase in speed comes from changing the address of the B instruction.

The critical parameters involved in a calculation of the latency time are:

1. The difference between TSD's of the address of the LH instruction and the instruction line number (n).

2. The difference between TSD's of the address of the RH instruction and the instruction line number (m).

3. The amount of time taken up in the actual execution of the Left and Right Hand Instructions individually.

4. Construction of the Tables

The following will refer to a limited number of instructions which will act as prototypes. For each UNIVAC instruction (except tape orders) there is a prototype which has the same execution time.

TABLE OF PROTOTYPE INSTRUCTIONS

Prototype	Instructions
B	B C F G H J L R E
A	A S
K	K X 00
U	U
T	Q T
Ox	Ox -x .x ;x
W	V W
Y	Y Z
M	M N P
D	D

Thus, the previous examples would be written:

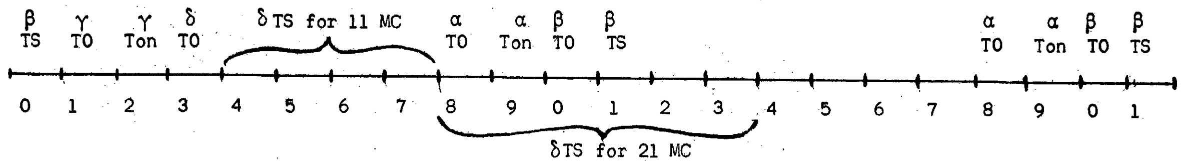
1. B 1 B 2
2. B 2 B 1
3. B 2 B 4

To simplify tables of latency times, references to prototype instruction pairs such as BB, AT, WU, etc., will be made as illustrated above.

EXAMPLE PROBLEM:

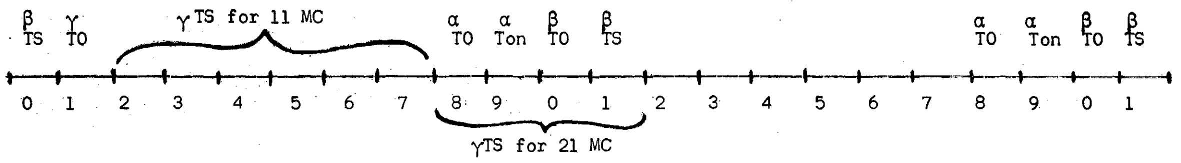
Prepare a table of latency times for the instruction pairs K B and B K.

<u>K B</u>



The earliest time at which TS could occur is at TSC 4. The last point at which it could occur and still leave enough time (3 MC for TO, Ton, and TO) for the instruction pair to be executed in 11 MC is 7. If m is 8, 9, 0, 1, 2, or 3, then the second TS is the one that occurs and the operation takes 21 MC.

<u>B K</u>



For n between 2 and 7, the instruction takes 11 MC; and for n between 8, 9, 0 and 1, the instruction takes 21 MC. This problem is relatively simple since the latency time is independent of the address of the K instruction.

STUDENT EXERCISES:

1. An A order takes 3 MC for execution, a B order takes 1 and an X order takes 1. How long does it take to execute the following instruction pairs? (Do not forget the TO minor cycles which follows each instruction.)

025	B 030	A 056
026	H 058	X 000
027	C 056	B 025

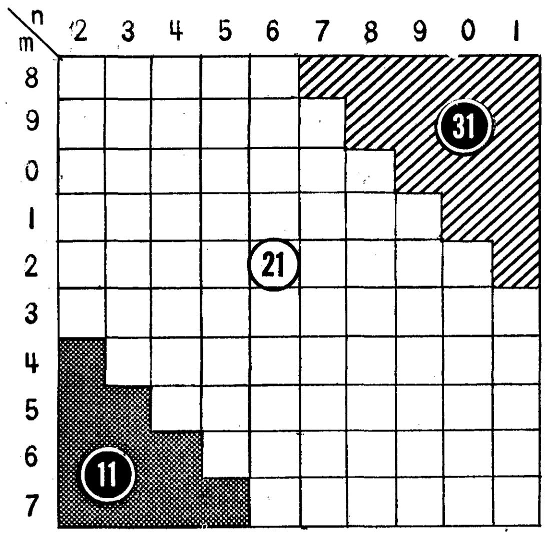
2. Prepare a table of instruction times for the prototype pair B n B m.

PART I

MINIMUM LATENCY

TABLES

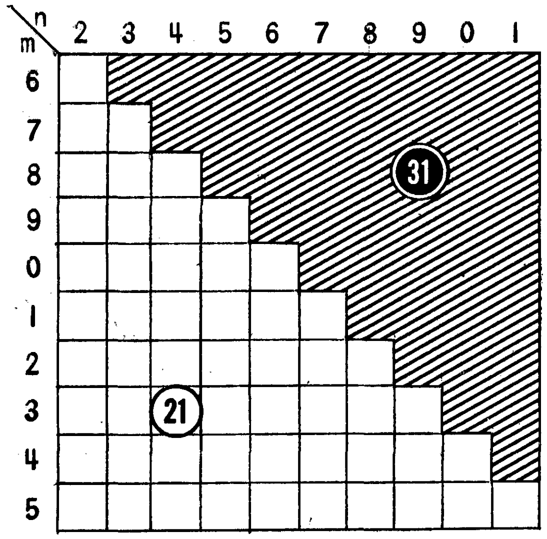
Line L = [B L + n B L + m]



BB

Av = 21.5

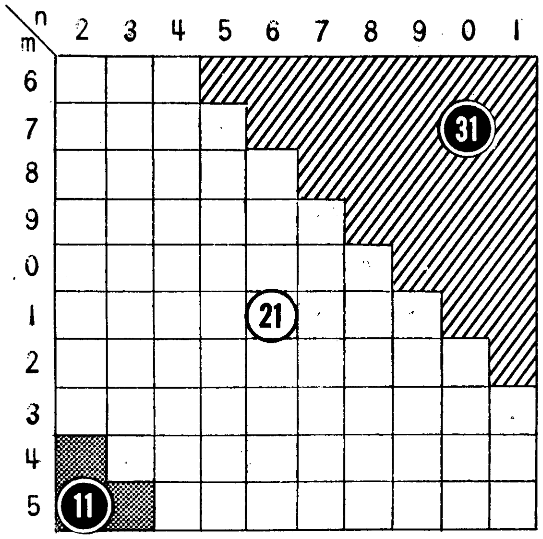
Line L = [A L + n A L + m]



AA

Av = 25.5

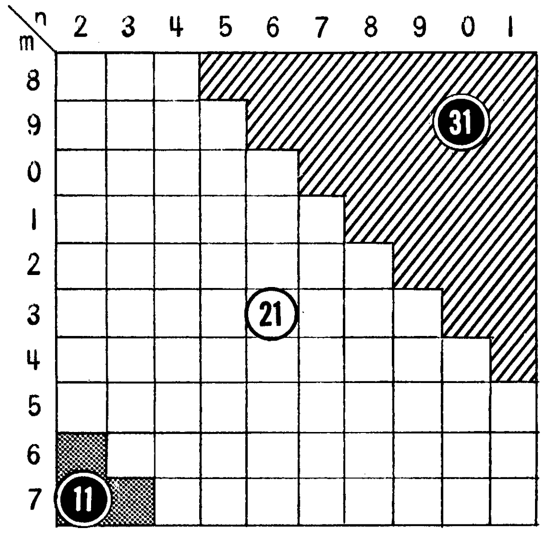
Line L = [B L+n A L+m]



BA

Av = 23.5

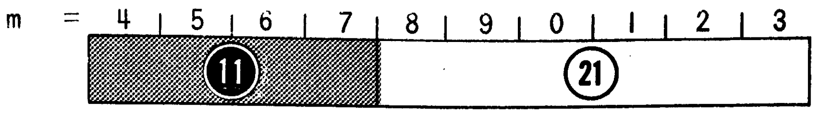
Line L = [A L+n B L+m]



AB

Av = 23.5

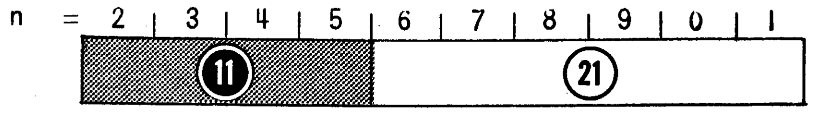
Line L = [K B L + m]



KB

Av = 17

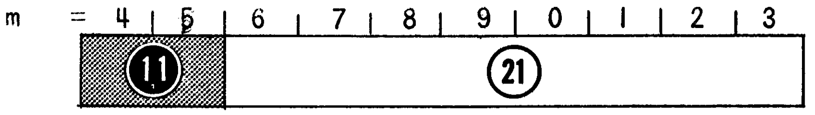
Line L = [B L + n K]



BK

Av = 17

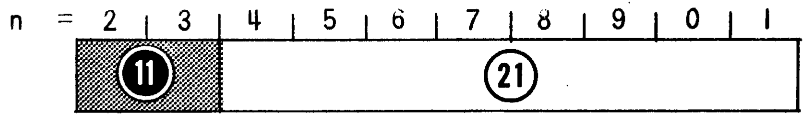
Line L = [K A L + m]



KA

Av = 19

Line L = [A L + n K]



AK

Av = 19

Line L = [K U L + m]

m =	8	9	0	1	2	3	4	5	6	7
	8	9	10	11	12	13	14	15	16	17

KU

Av = 12.5

Line L = [K K]

11

KK

Av = 11

Line L = [B L + n U L + m]

m \ n	2	3	4	5	6	7	8	9	0	1
8	8	18	18	18	18	18	18	18	18	18
9	9	9	19	19	19	19	19	19	19	19
0	10	10	10	20	20	20	20	20	20	20
1	11	11	11	11	21	21	21	21	21	21
2	12	12	12	12	12	22	22	22	22	22
3	13	13	13	13	13	13	23	23	23	23
4	14	14	14	14	14	14	14	24	24	24
5	15	15	15	15	15	15	15	15	25	25
6	16	16	16	16	16	16	16	16	16	26
7	17	17	17	17	17	17	17	17	17	17

BU

Av = 17

Line L = [A L + n U L + m]

m \ n	2	3	4	5	6	7	8	9	0	1
0	10	20	20	20	20	20	20	20	20	20
1	11	11	21	21	21	21	21	21	21	21
2	12	12	12	22	22	22	22	22	22	22
3	13	13	13	13	23	23	23	23	23	23
4	14	14	14	14	14	24	24	24	24	24
5	15	15	15	15	15	15	25	25	25	25
6	16	16	16	16	16	16	16	26	26	26
7	17	17	17	17	17	17	17	17	27	27
8	18	18	18	18	18	18	18	18	18	28
9	19	19	19	19	19	19	19	19	19	19

AU

Av = 19

a) TRANSFER

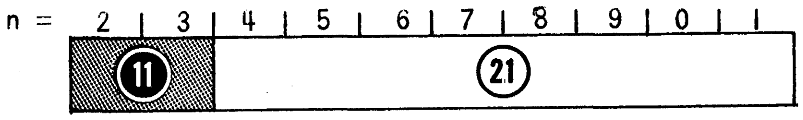
Line L = [B L + n T L + m]

m \ n	2	3	4	5	6	7	8	9	0	1
0	10	20	20	20	20	20	20	20	20	20
1	11	11	21	21	21	21	21	21	21	21
2	12	12	12	22	22	22	22	22	22	22
3	13	13	13	13	23	23	23	23	23	23
4	14	14	14	14	14	24	24	24	24	24
5	15	15	15	15	15	15	25	25	25	25
6	16	16	16	16	16	16	16	26	26	26
7	17	17	17	17	17	17	17	17	27	27
8	18	18	18	18	18	13	18	18	18	28
9	19	19	19	19	19	19	19	19	19	19

BT

Av = 19

b) NO TRANSFER



a) Transfer

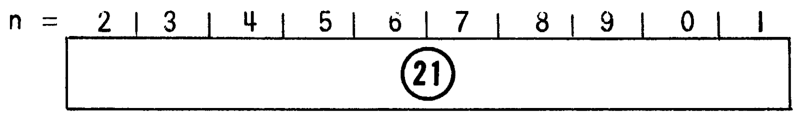
Line L = [A L + n T L + m]

m \ n	2	3	4	5	6	7	8	9	0	1
2	12	22	22	22	22	22	22	22	22	22
3	13	13	23	23	23	23	23	23	23	23
4	14	14	14	24	24	24	24	24	24	24
5	15	15	15	15	25	25	25	25	25	25
6	16	16	16	16	16	26	26	26	26	26
7	17	17	17	17	17	17	27	27	27	27
8	18	18	18	18	18	18	18	28	28	28
9	19	19	19	19	19	19	19	19	29	29
0	20	20	20	20	20	20	20	20	20	30
1	21	21	21	21	21	21	21	21	21	21

$\boxed{\textbf{AT}}$

Av = 21

b) NO TRANSFER



a) TRANSFER

Line L = [K T L + m]

m =	0	1	2	3	4	5	6	7	8	9
	10	11	12	13	14	15	16	17	18	19

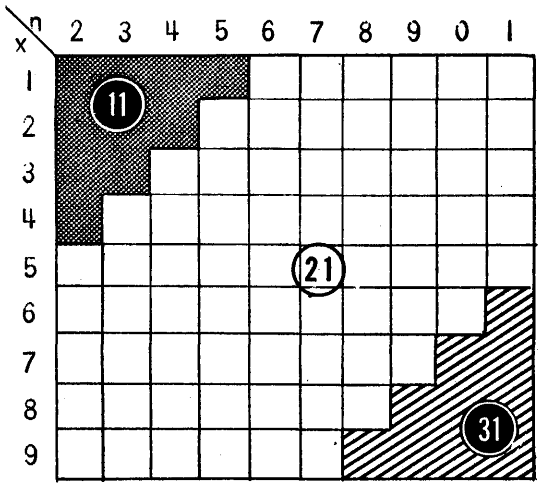
$\boxed{\textbf{KT}}$

Av = 12.75

b) NO TRANSFER

$\boxed{11}$

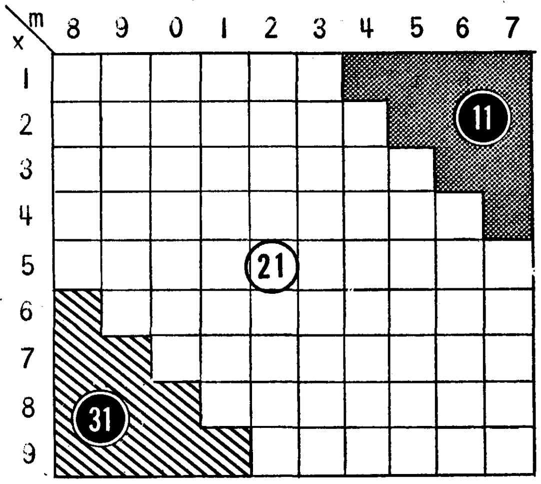
Line L = [B   L + n   0 x   ]



BO

Av = 16 + X

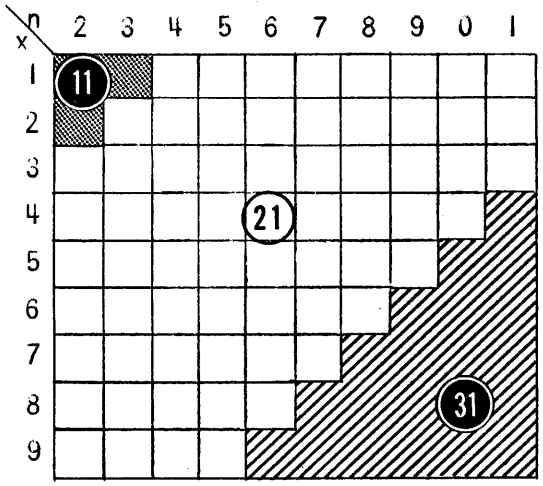
Line L = [0 x   B   L + m]



OB

Av = 16 + x

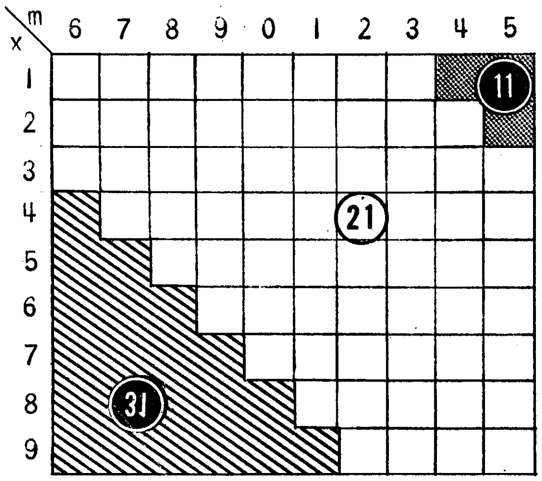
Line L = [A L + n 0 x]



A0

Av = 18 + X

Line L = [0 x A L + m]

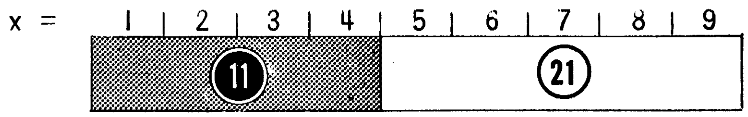


0A

Av = 18 + x

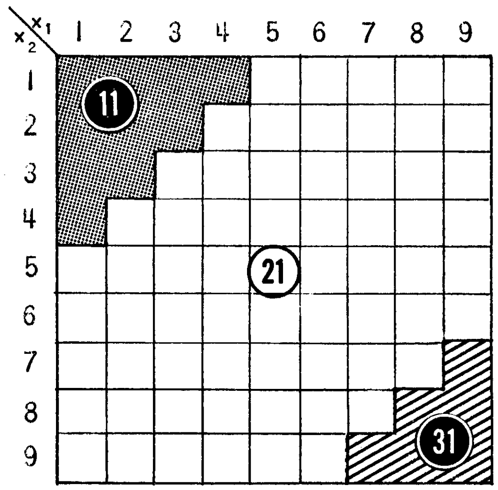
Line L = [0 x K]
Line L = [K 0 x]

OK



KO

Line L = [0 x_1 0 x_2]



00

Line L = [0 x U L + m]

m \ x	1	2	3	4	5	6	7	8	9
8	8	18	18	18	18	18	18	18	18
9	9	9	19	19	19	19	19	19	19
0	10	10	10	20	20	20	20	20	20
1	11	11	11	11	21	21	21	21	21
2	12	12	12	12	12	22	22	22	22
3	13	13	13	13	13	13	23	23	23
4	14	14	14	14	14	14	14	24	24
5	15	15	15	15	15	15	15	15	25
6	16	16	16	16	16	16	16	16	16
7	17	17	17	17	17	17	17	17	17

$\boxed{\text{0U}}$

Av = 11.5 + X

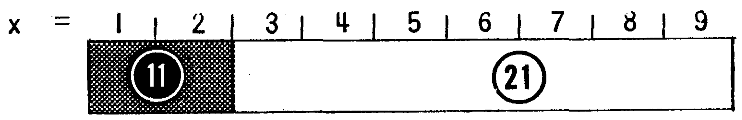
a) TRANSFER

Line L = [0 x T L + m]

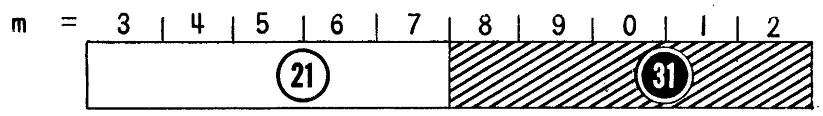
m \ x	1	2	3	4	5	6	7	8	9
0	10	20	20	20	20	20	20	20	20
1	11	11	21	21	21	21	21	21	21
2	12	12	12	22	22	22	22	22	22
3	13	13	13	13	23	23	23	23	23
4	14	14	14	14	14	24	24	24	24
5	15	15	15	15	15	15	25	25	25
6	16	16	16	16	16	16	16	26	26
7	17	17	17	17	17	17	17	17	27
8	18	18	18	18	18	18	18	18	18
9	19	19	19	19	19	19	19	19	19

$\boxed{\text{0T}}$

b) NO TRANSFER



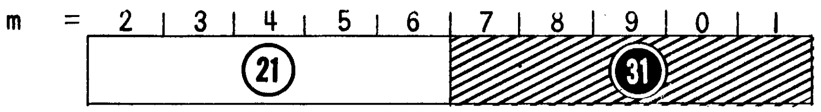
Line L = [Y B L + m]



$\boxed{\text{YB}}$

Av = 26

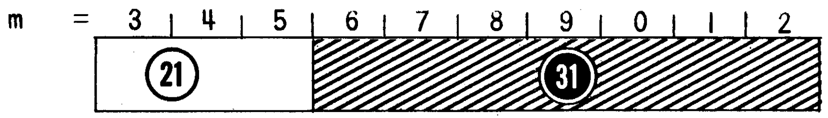
Line L = [B L + m Y]



$\boxed{\text{BY}}$

Av = 26

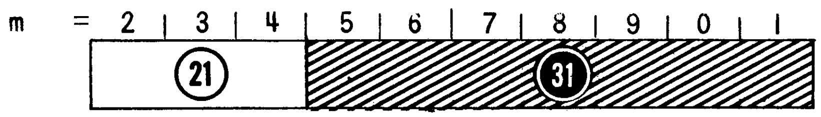
Line L = [Y A L + m]



$\boxed{\text{YA}}$

Av = 28

Line L = [A L + m Y]



$\boxed{\text{AY}}$

Av = 28

Line L = [K Y]
Line L = [Y k]

$\boxed{21}$

$\boxed{KY}$

$\boxed{YK}$

Av = 21

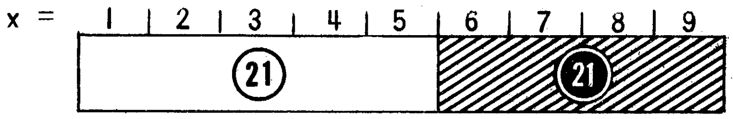
Line L = [Y Y]

$\boxed{31}$

$\boxed{YY}$

Av = 31

Line L = [Y 0 x]
Line L = [0 x Y]



$\boxed{Y0}$

$\boxed{0Y}$

a) TRANSFER Line L = [Y T L + m]

m =	9	0	1	2	3	4	5	6	7	8
	19	20	21	22	23	24	25	26	27	28

$\boxed{\mathbf{YT}}$

Av = 22.25

b) NO TRANSFER

$\boxed{\mathbf{21}}$

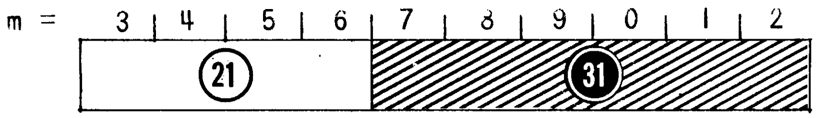
Line L = [Y U L + m]

m =	6	7	8	9	0	1	2	3	4	5
	16	17	18	19	20	21	22	23	24	25

$\boxed{\mathbf{YU}}$

Av = 20.5

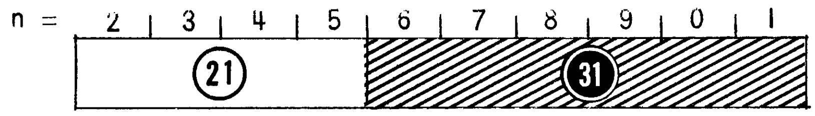
Line L = [Y W L + m]



$\boxed{\mathbf{YW}}$

Av = 27

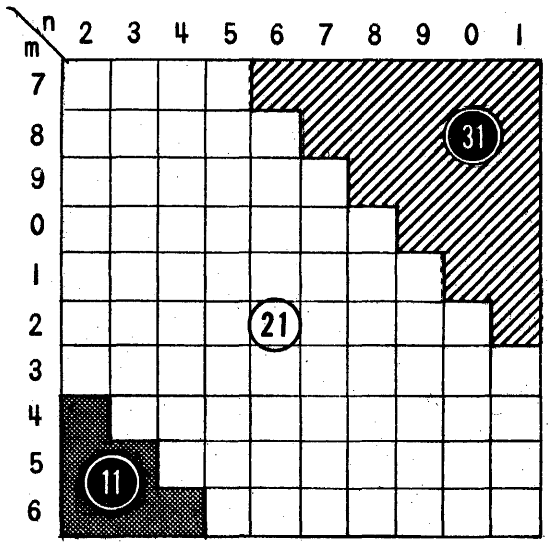
Line L = [W L + n Y]



$\boxed{\mathbf{WY}}$

Av = 27

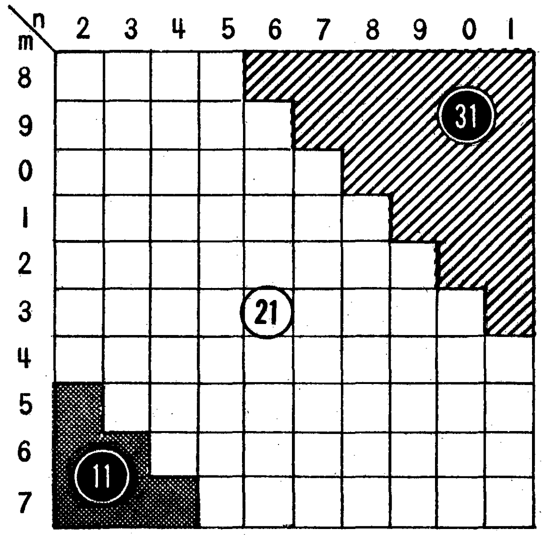
Line L = [B L + n W L + m]



BW

Av = 22.5

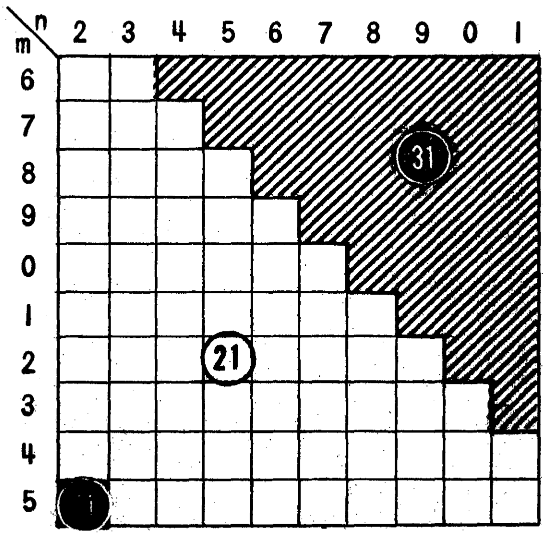
Line L = [W L + n B L + m]



WB

Av = 22.5

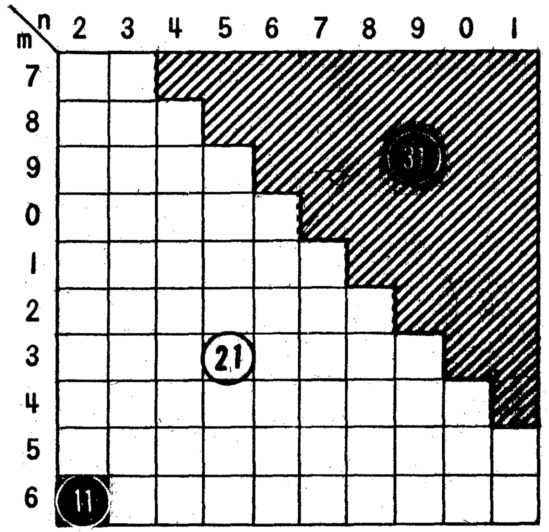
Line L = [W L + n A L + m]



WA

Av = 24.5

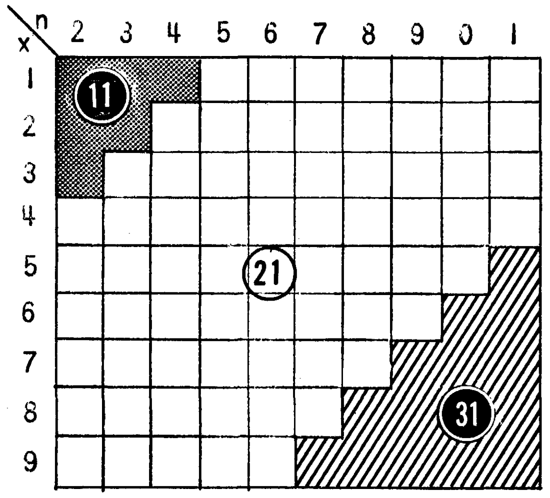
Line L = [A L + n W L + m]



AW

Av = 24.5

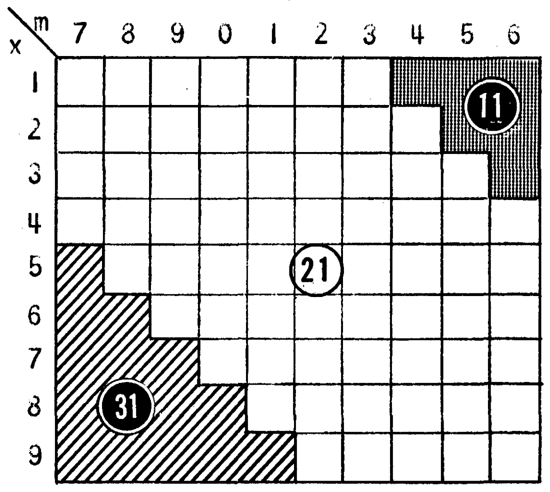
Line L = [W L + n 0 x]



WO

Av = 17 + x

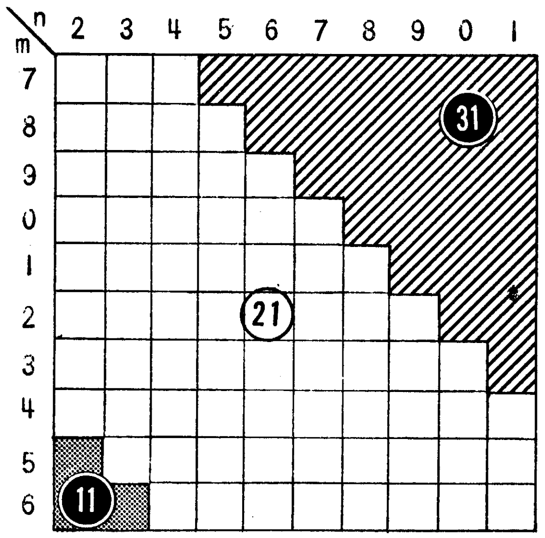
Line L = [0 x W L + m]



OW

Av = 17 + x

Line L = [W L + n W L + m]



WW

Av = 23.5

a) Transfer

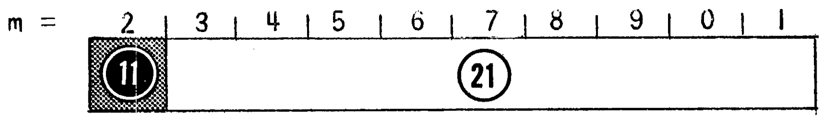
Line L = [W L + n T L + m]

m \ n	2	3	4	5	6	7	8	9	0	1
1	11	21	21	21	21	21	21	21	21	21
2	12	12	22	22	22	22	22	22	22	22
3	13	13	13	23	23	23	23	23	23	23
4	14	14	14	14	24	24	24	24	24	24
5	15	15	15	15	15	25	25	25	25	25
6	16	16	16	16	16	16	26	26	26	26
7	17	17	17	17	17	17	17	27	27	27
8	18	18	18	18	18	18	18	18	28	28
9	19	19	19	19	19	19	19	19	19	29
0	20	20	20	20	20	20	20	20	20	20

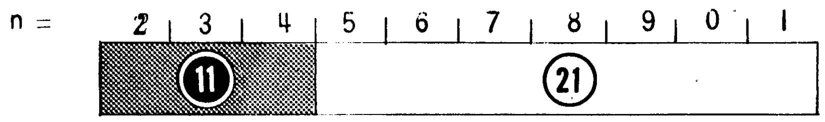
WT

Av = 20

b) NO TRANSFER



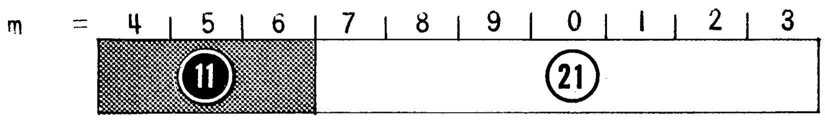
Line L [W L + n K]



WK

Av = 18

Line L = [K W L + m]



KW

Av = 18

Line L = [W L + n U L + m]

m \ n	2	3	4	5	6	7	8	9	0	1
9	9	19	19	19	19	19	19	19	19	19
0	10	10	20	20	20	20	20	20	20	20
1	11	11	11	21	21	21	21	21	21	21
2	12	12	12	12	22	22	22	22	22	22
3	13	13	13	13	13	23	23	23	23	23
4	14	14	14	14	14	14	24	24	24	24
5	15	15	15	15	15	15	15	25	25	25
6	16	16	16	16	16	16	16	16	26	26
7	17	17	17	17	17	17	17	17	17	27
8	18	18	18	18	18	18	18	18	18	18

WU

Av WU = 18

5. Examples of Minimum Latency Coding

It is not generally possible or desirable to program each line of a routine in minimum time; therefore, the following order of precedence should be alloted to the sections of a data processing run.

1. Instructions which process each individual item.

2. Instructions which concern the transfer of blocks into and out of the computer.

Consider the following routine which is designed to place consecutive two-word items into temporary storage.

```
                                               MAXIMUM

100  B   110  A*  109                             31
101  C   110  U   110                             19
102                                              ───
103      tape read                                50 MC item
.        routine
.
.
109  001002  000000
110  V70300  W  200
111      processing
112      routine                                121 seconds/tape
.
.


                                               MINIMUM

100  B   113  A* 105                              11
101  C   113  U  113                              12
102      tape read                               ───
.        routine                                  23 MC item
.
105  001002  000000
.
.
.                                                 56 seconds/tape
113  V70300  W  200
114      processing
115      routine
```

Thus, the total savings obtained by changing two lines of instructions is of the order of 65 seconds per tape processed.

Practically every routine on UNIVAC makes use of generalized overflow for control purposes. It is, therefore, useful to have this frequently-used routine in minimum form. Compare the following:

```
                                         AVERAGE

000     R   052   U   050                  10
 .
 .
050     B   052   A   053                  21
051     C   052   U   052                  21
052    [  -          - ]                   52 MC
053     000000   000001

                                         MINIMUM

000     R   012   U   008                   8
 .
 .
 .
008     B   010   A   012                  11
009     C   011   U   011                  12
010     000000   000001                    31 MC
011    [  -          - ]
012    [  -          - ]
```

6. <u>Multiplication Tables</u>

Additional tables have been prepared for instruction pairs involving multiplication by a known multiplier. Because these tables are quite extensive, only the most common instruction pairs, that is, BM and MB, are shown. However, as explained below, instruction times for KM and MK prototypes can also be looked up.

The tables have been prepared in a fashion similar to that used for the other arithmetic instructions. However, multiplication is not a fixed length operation; the number of stages required depends on the number of additions performed during the multiplication, which in turn, depends on the multiplier digits.

Figure 2A shows the stages required for multiplication involving no additions, i.e., multiplication by zero. A sample problem is solved for the instruction pair

B 2 M 6

in Figure 2B. This instruction pair requires 31 minor cycles, as shown. (The multiplication stages are starred.)

If additions are involved, one minor cycle is added for each addition. Figure 2C shows a sample problem

B 2 M 6

where the multiplier is a single digit "2", requiring two additions. Actually, the additions occur somewhere between M5 and M15, but they are attached to the end for descriptive convenience.

If an instruction of the type K is involved, instead of type B, these tables may still be used. If K occurs as a LHI, use the appropriate table for BM with n = 2. If K occurs as a RHI, use the proper table MB with m = 7.

The following chart shows the number of additions required for various multiplier digits.

Multiplier Digit	Number of Additions
0	0
1,3	1
2,4,6	2
5,7,9	3
8	4

If the multiplier contains more than one digit, the total number of additions is equal to the sum of the additions required for all the digits. For instance,

Multiplier	Number of Additions
25	5
75	6
33333333333	11

A		B		C
Stage	TSD	Stage	TSD	Stage
TO	0	βTS	0	βTS
M1 (Multiplier→rX)	1	γTO	1	γTO
TO	2	γTS	2	γTS
M2	3	*δTO	3	*δTO
TO	4		4	
M3	5		5	
TO	6	* M1 (Multiplier→rX)	6	* M1
M4	7	* TO	7	* TO
TO	8	* M2	8	* M2
M5	9	* TO	9	* TO
M6	0	* M3	0	* M3
M7	1	* TO	1	* TO
M8	2	* M4	2	* M4
M9	3	* TO	3	* TO
M10	4	* M5	4	* M5
M11	5	* M6	5	* M6
M12	6	* M7	6	* M7
M13	7	* M8	7	* M8
M14	8	* M9	8	* M9
TO	9	* M10	9	* M10
M15	0	* M11	0	* M11
	1	* M12	1	* M12
	2	* M13	2	* M13
	3	* M14	3	* M14
	4	* TO	4	* TO
	5	* M15	5	* M15
	6	αTO	6	* A1
	7	αTon	7	* A2
	8	βTO	8	αTO
	9		9	α
	0		0	βTO
	1	βTS	1	βTS

Figure 2

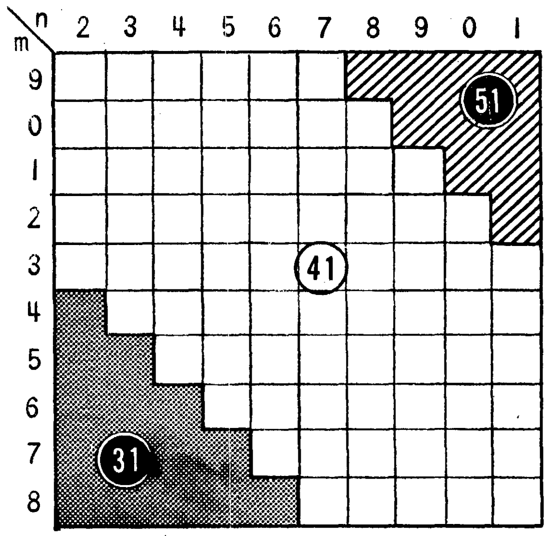
PART II

MINIMUM LATENCY

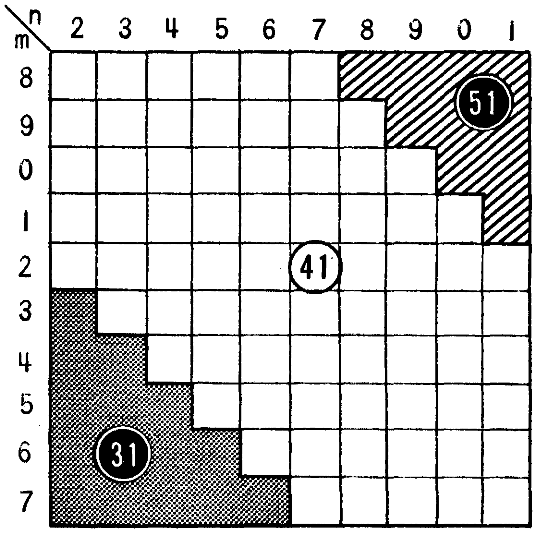
TABLES

NO ADDITIONS

Line L = [B L + n M L + m]

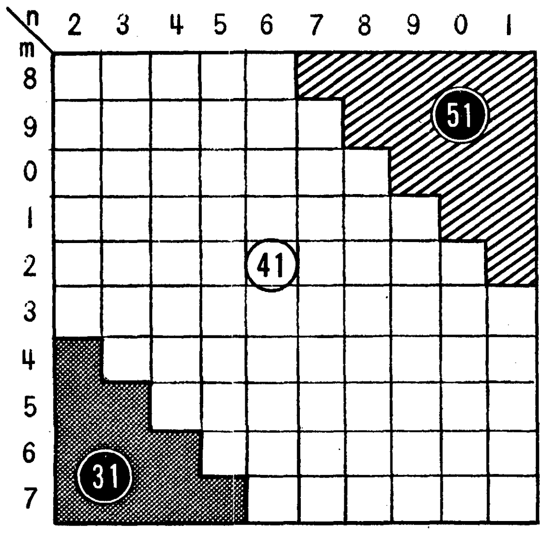


Line L = [M L + n B L + m]

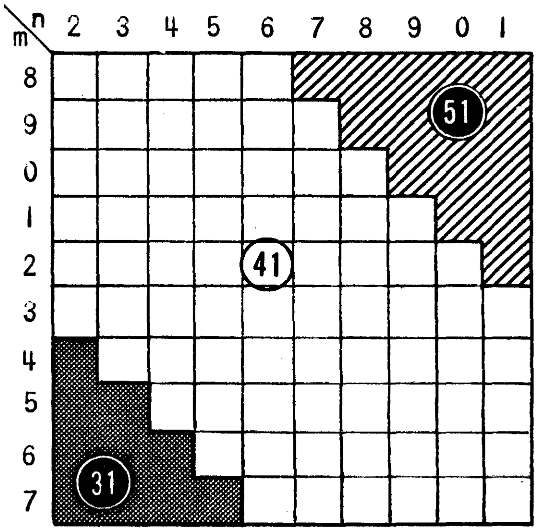


ONE ADDITION

Line L = [B L + n M L + m]

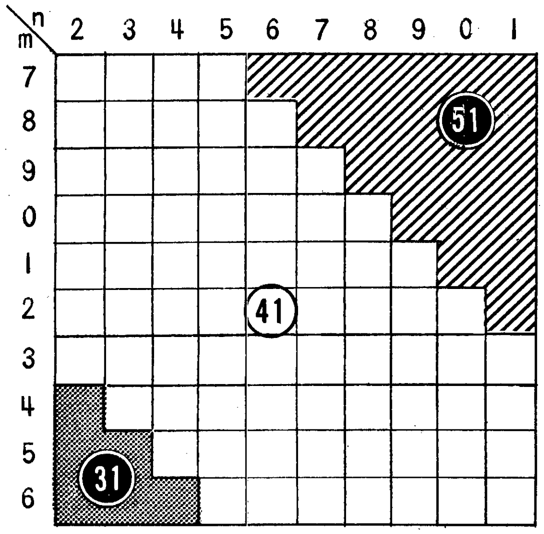


Line L = [M L + n B L + m]

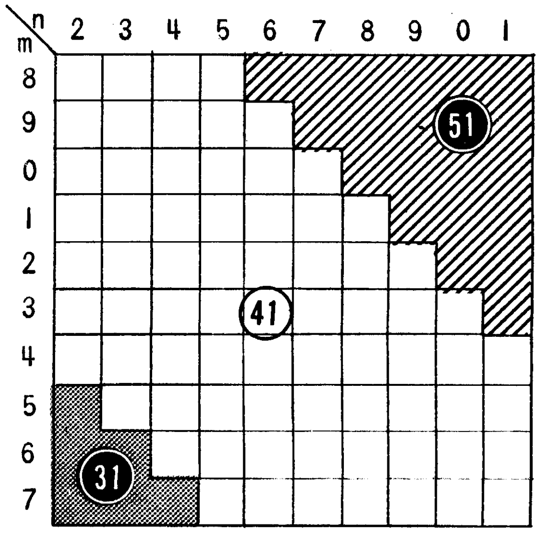


TWO ADDITIONS

Line L = [B L + n M L + m]

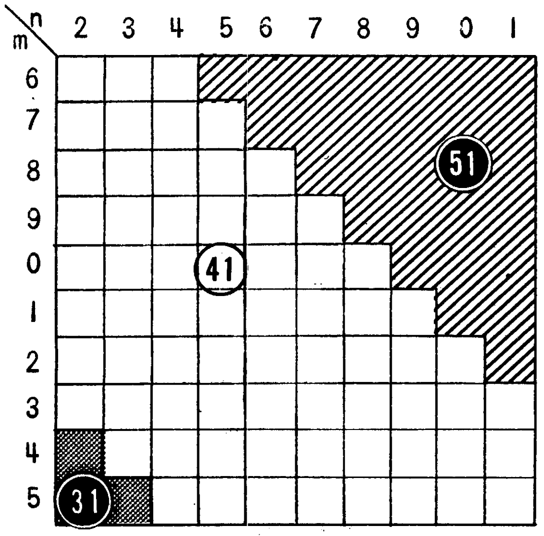


Line L = [M L + n B L + m]

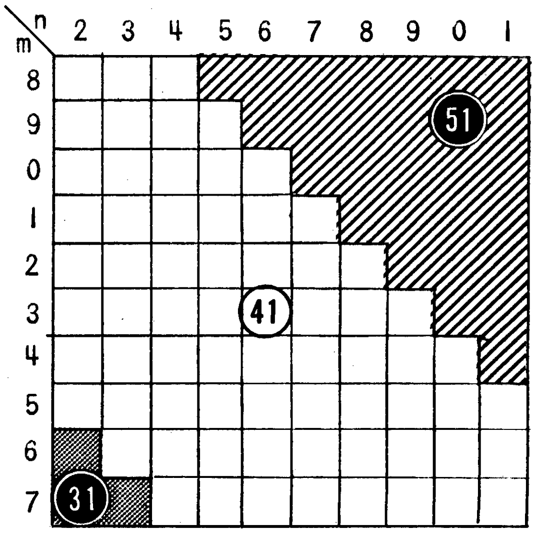


THREE ADDITIONS

Line L = [B L + n M L + m]

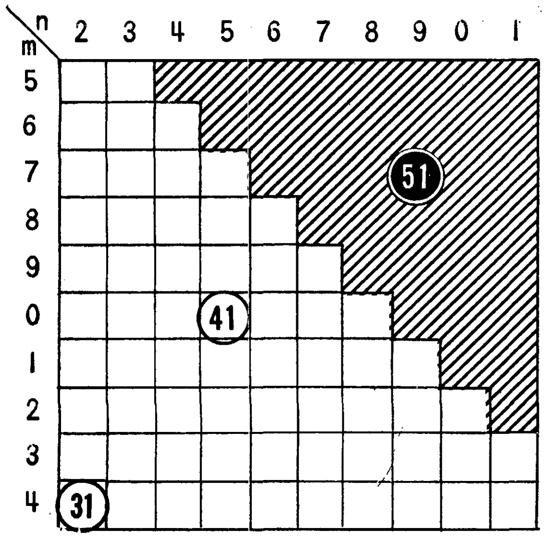


Line L = [M L + n B L + m]

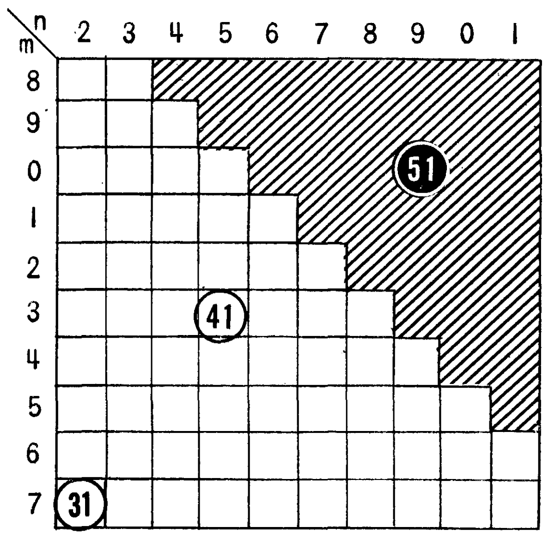


FOUR ADDITIONS

Line L = [B L + n M L + m]

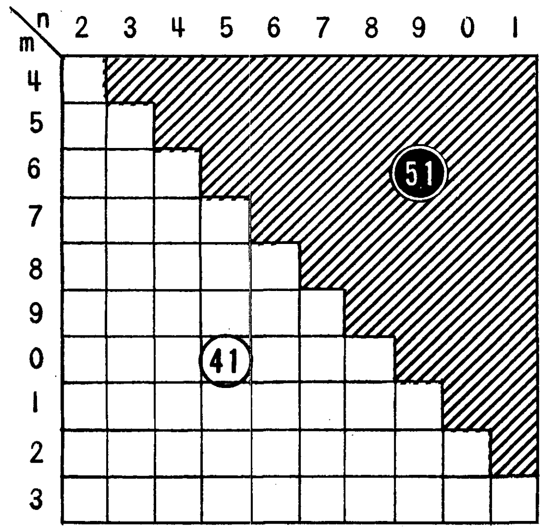


Line L = [M L + n B L + m]

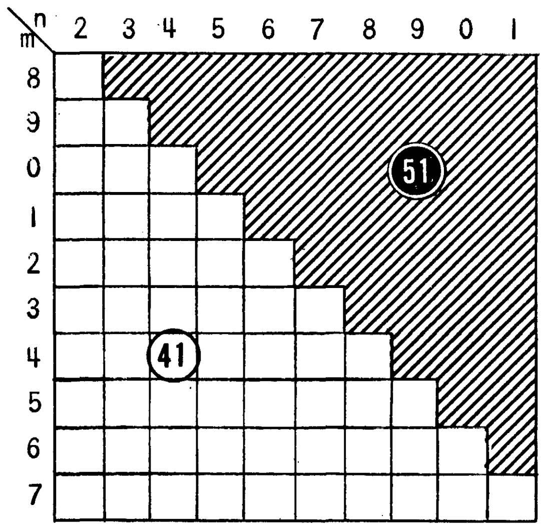


FIVE ADDITIONS

Line L = [B L + n M L + m]

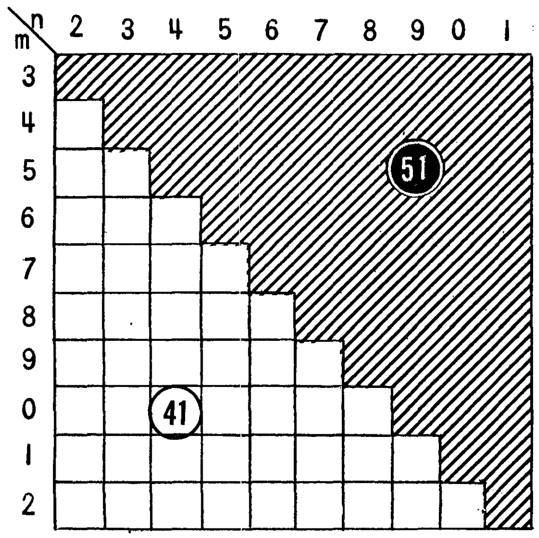


Line L = [M L + n B L + m]

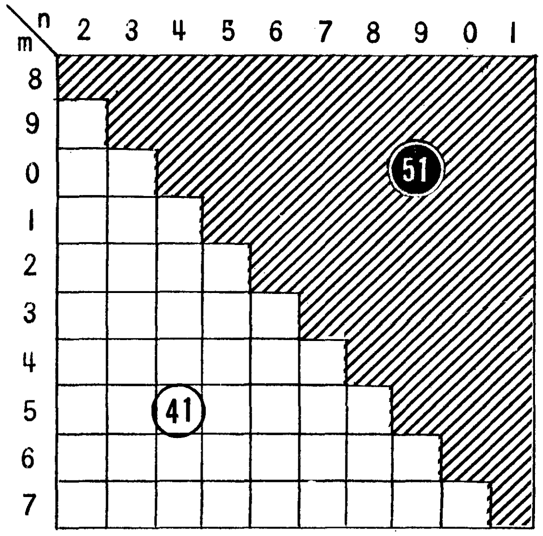


SIX ADDITIONS

Line L = [B L + n M L + m]

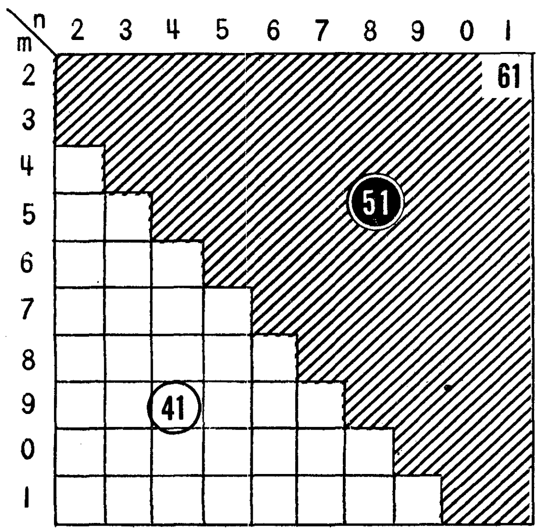


Line L = [M L + n B L + m]

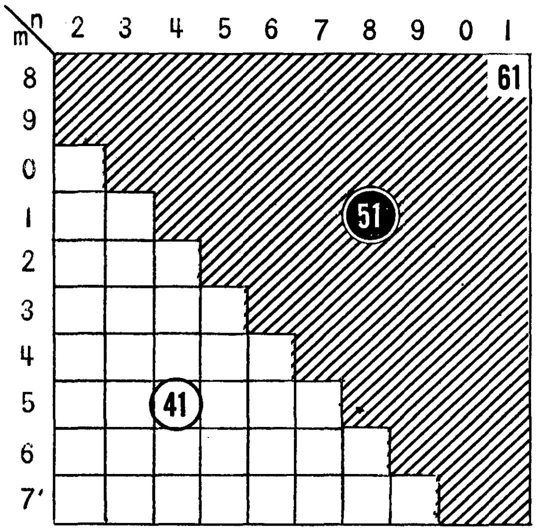


SEVEN ADDITIONS

Line L = [B L + n M L + m]

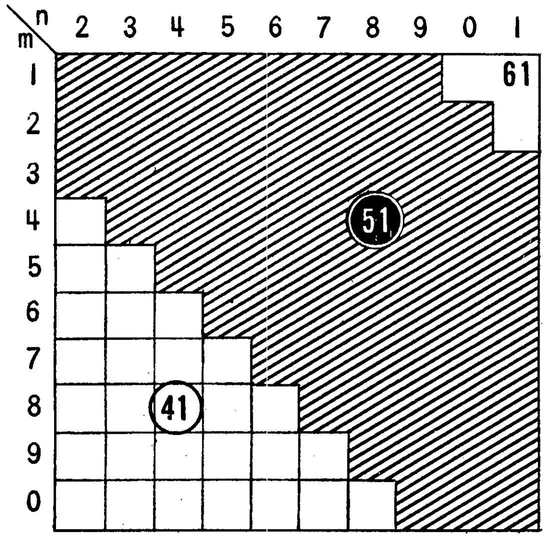


Line L = [M L + n B L + m]

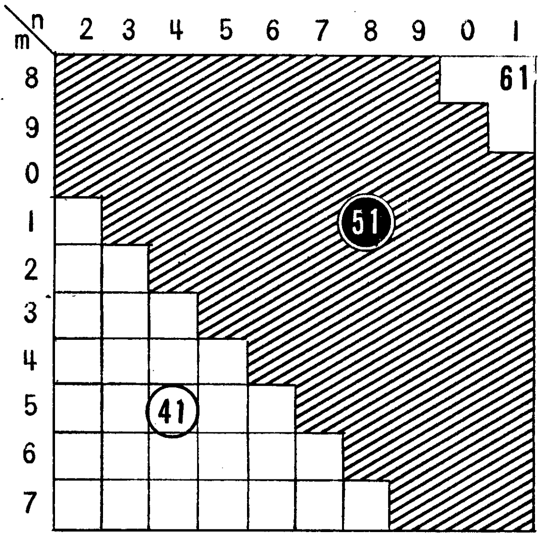


EIGHT ADDITIONS

Line L = [B L + n M L + m]

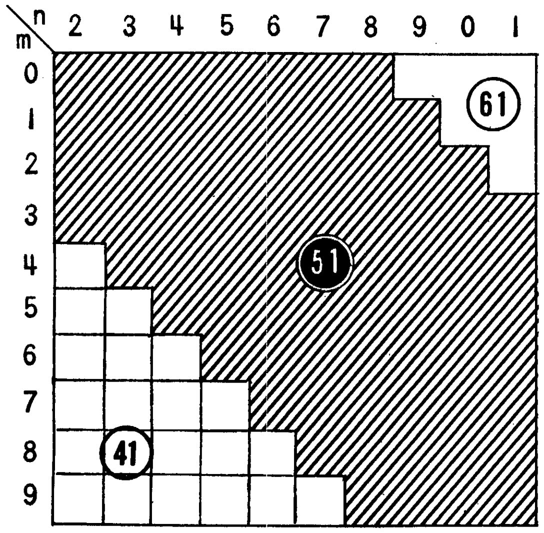


Line L = [M L + n B L + m]

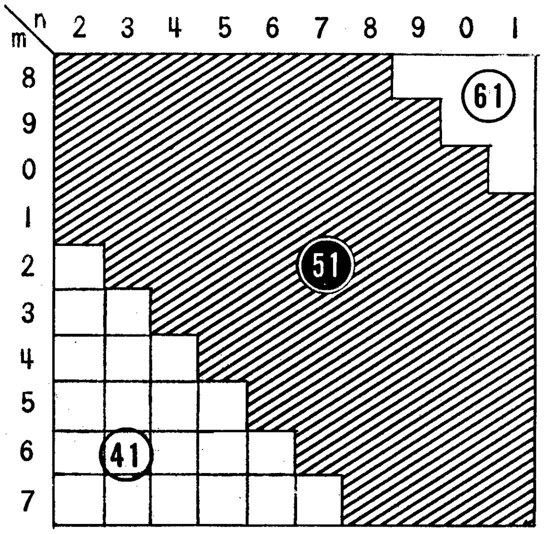


NINE ADDITIONS

Line L = [B L + n M L + m]



Line L = [M L + n B L + m]



7. <u>Efficient Coding Example</u>

The following problem will further illustrate the use of linear and minimum latency coding. It is designed as a subroutine of a payroll computation. A net pay is supplied to the routine in rA. Compute the number of bills and coins of the various denominations required to make up a pay envelope as follows:

0	0000000xxx‸xx	total
1	00000000000x	number of $100
2	00000000000x	" " $20
3	00000000000x	" " $10
4	00000000000x	" " $ 5
5	00000000000x	" " $ 1
6	00000000000x	" " 25¢
7	00000000000x	" " 10¢
8	00000000000x	" " 5¢
9	00000000000x	" " 1¢

```
103   F    116
                   H    200          Amount to 200                                11
104   .4   000
                   G    201          # $100 bills to 201                          11
105   E    200
                   L    192                                                       11
106   .3   000
                   X    000                                                       11
107   C    142                       10's digit + 143 - R. H. address 142
                   F    184                                                       11
108   E    200
                   .2   000                                                       11
109   A    111
                   C    115          Unit's digit +130 - R. H. address 115        11
110   F    112
                   U    118                                                        8
111     -
                   U    143
112   000 000
                   000 011
113      -
                   000 002
114      -
                   000 001
115  [   -                                                                      8/17
                      -    ]
116   000 000
                   001 000
117      -
                      -
118   E    200
                   K    000                                                       21
119   P    191
                      -                                                           31
120     -
                   C    206          *I.P. ¢ x 4 = # of quarters                  11
121   X
                   .1                                                             11
122   K
                   P    193          **F.P. (F.P. x 10/4 =# of dimes)             41
123   C    207
                   X                                                              11
124   K
                   P    189          I.P. (LSD x 20/10/4)=# of nickels            31
125   C    208
                   X                                                              11
126   K
                   P    190          I.P. (LSD x 100/20/10/4) = # of pennies      31
```

*I.P. = Integral Part

**F.P. = Fractional Part

```
127   C    209
                 U    115 .                                                  8
128     -
                      -
129     -
                      -
130   V    202
                 U    140                                                   10
131   V    173
                 U    140                                                    9
132   V    154
                 U    140                                                   18
133   V    175
                 U    140                                                   17
134   V    170              Pick correct comb. of 10 & 20 bills
                 U    140                                                   16
135   V    172
                 U    140                                                   15
136   V    178
                 U    140                                                   14
137   V    180
                 U    140                                                   13
138   V    182
                 U    140                                                   12
139   V    181
                 U    140                                                   11
140   W    202
                 Y    200                                                   21
141   EXIT ΔΔ
                 SUB. ΔΔ
142  [W    204
                 U    130 ]                                               9/18
143   V    155
                 U    142                                                    9
144   V    156
                 U    142                                                   18
145   V    167
                 U    142                                                   17
146   V    158
                 U    142                                                   16
147   V    169              Pick up correct comb. of 5 & 1 bills
                 U    142                                                   15
148   V    162
                 U    142                                                   14
149   V    161
                 U    142                                                   13
```

```
150   V    113
                  U    142                                          12
151   V    153
                  U    142                                          11
152   V    164
                  U    142                                          10
153     -
                  000  003
154     -
                  000  001
155     -
                     -
156     -
                     -
157     -
                  000  001
158     -
                     -
159     -
                  000  003
160     -
                     -
161     -
                  000  001
162     -
                  000  001
163     -
                     -
164     -
                  000  001
165     -
                  000  004
166     -
                     -
167     -
                  000  002
168     -
                     -
169     -
                  000  004
170   000  000
                  000  002
171   000  000
                  000  000
172   000  000
                  000  002
173   000  000
                  000  001
174   000  000
                  000  000
```

```
175   000 000
                000 001
176   000 000
                000 001
177      -
                   -
178   000 000
                000 003
179   000 000
                000 000
180   000 000
                000 003
181   000 000
                000 001
182   000 000
                000 004
183   000 000
                000 000
184   000 000
                000 100
185      -
                   -
186      -
                   -
187      -
                   -
188      -
                   -
189   000 000
                000 002
190   000 000
                000 005
191   004 000
                000 000
192   W   204
                U   130
193   000 000
                000 025
194      -
                   -
195      -
                   -
196      -
                   -
197      -
                   -
198      -
                   -
199      -
                   -
200      -
                   -
201      -
                   -
202   000 000
                000 000
203   000 000
                000 000
```

STUDENT EXERCISES

1. Code and time an efficient routine which will place into rA the smallest of the 4 quantities in memory location 100-103.

2. Code and time an efficient routine which will add the 300 words in the memory starting at location 100.

3. Code and time an efficient routine which will check a tape of one-word items for ascending sequence. Assume the standard sentinel convention is followed.

COLLATION METHOD OF
SORTING

1. Flowchart Description of Collation

This section assumes the reader is familiar with the general principles of collation as described in the paper "A Brief Description of Sorting Methods for the Univac System." The following discussion is intended as a further examination into the manner by which collation is achieved on the Univac. As the intent is instructional, the methods described are fundamental ones basic to all collation runs, but, of course, this section is not intended as a primer of all the shop tricks employed in speeding up collation routines. For simplicity in the presentation of these principles, two-way collation will be described.

Plates 1-5 describe a two-way collation for ten-word items. The following symbols used in these flow charts are defined below.

T_a	Tape mounted on Uniservo a
$T_a \rightarrow A$	The next sequentially available block from T_a is brought into the computer in a forward direction.
$A \leftarrow T_a$	Same as above, except the tape moves in a backward direction
A	The block from T_a (consists of 6 items)
A_i	The ith item of A (i = 1, 2,..., 6). The items in a block are numbered from top to bottom.
A_i^o	The keyword of A_i
La	The label in the identification block of T_a
C_k	The kth C item (k = 1, 2,..., 6)
C	The collation of 6 C items
$C \rightarrow T_c$	The C block is written on T_c
Z,S,$\not{I}$,=	Sentinels; The following collation sequence applies $\not{I} < A_i^o < Z < =$ for all A_i^o

The flow charts assume that the input tape to be sorted is in the standard data tape format described in the section on identification blocks and reruns. This input tape is assumed to be mounted on T_5 but any other servo will do (except T_1 or T_2 unless these servos are also changed).

The instructions are assumed to be in the memory.

At the start of the problem certain counters and variable connectors are set which will be described as they are encountered. Next, an "S" sentinel is placed in the key word position of a block, $\rightarrow\boxed{S \rightarrow A_1^o}\!-$, and this block is written twice on the first output tapes, T_1 and T_2. This is necessary since these tapes will later be read backwards and the sentinel blocks are therefore needed to indicate the front end of the tapes.

At $\rightarrow\boxed{\text{"Tape label"} \rightarrow \text{SCP}}$, the routine requests the operator to type in the desired input tape label, La. The first block on T_5 is then read into the computer and the lable recorded on tape is compared with the label typed in by the operator. If the labels do not agree, T_5 is rewound with interlock, and the tape label and typed label are printed on SCP. If the operator finds that he has typed incorrectly, he may force "no transfer" at breakpoint 1. The routine will then return to ㉕ and allow him to type the correct label. If the input tape was incorrectly labeled, he may mount the correct tape, force transfer on breakpoint 1 and the routine will return to ㉖. When ① is reached we are assured that the input tape to be sorted is the appropriate one. The tape label, La, suitably modified to indicate that the items are sorted, will be placed on the final output tape.

The logical operations between ① and ⑩ are commonly referred to as the "internal sort". The purpose of the internal sort is to place the items of each block in order and then (in two-way collation) to split this input into two nearly equal piles.

At ① the first data block from T_5 is read into the computer. This block is called "A": its individual items are $A_1, A_2, \ldots, A_6$, any one of which is identified by A_i. Next, the key word of the last A item of the block is compared with a Z sentinel for the end of tape test. If this is not the last block on the input tape, the items of A are arranged in ascending sequence by one of the methods to be described later. This ordered A block is written on T_m ($m = 1$, initially); one is added to Δ, which is the input block counter; and since the variable connector α_1, is set, the output tape designation is changed from servo 1 to 2. Control is returned to ① whence it follows that each block from T_5 has its individual items ordered and is then written on either T_1 or T_2. When the first sentinel block on T_5 is read, the position of the last valid (nonsentinel) item is determined at ③. Then at ④ the remaining item positions are filled with Z sentinels, variable connector α_2 is set, and control is transferred to ② where the block is sorted and written on the current output tape.

At ⑤ the number of blocks to be sorted, Δ, is examined. The purpose of this examination is to determine whether the first "external sort" or merging will be in ascending or descending order. Since the final output should be in ascending sequence, the first merge must be an ascending one if the number of data passes required is odd and it must be a descending one if the data passes required are even. This fact is clarified in the following table which shows that each external pass over the data changes its sequence. The number of data passes can be obtained from the block counter Δ. If the block counter is such that control is sent to ⑦ the external sort is set to do an ascending merge; but if control is transferred to ⑧, a descending merge is set up.

Cumulative Data Passes	Consecutive Input Blocks in Sequence	Input Sequence Required Where Total Data Passes		Consecutive Output Blocks in Sequence	Type of Output Sequence for Total Data Passes	
		Odd	Even		Odd	Even
1	1	* A	* A	2	A	D
2	2	A	D	4	D	A
3	4	D	A	8	A	D
4	8	A	D	16	D	A
5	16	D	A	32	A	D
6	32	A	D	64	D	A
7	64	D	A	128	A	D
8	128	A	D	256	D	A
9	256	D	A	512	A	D
10	512	A	D	1024	D	A
11	1024	D		2048	A	

* Fixed because of ascending internal sort

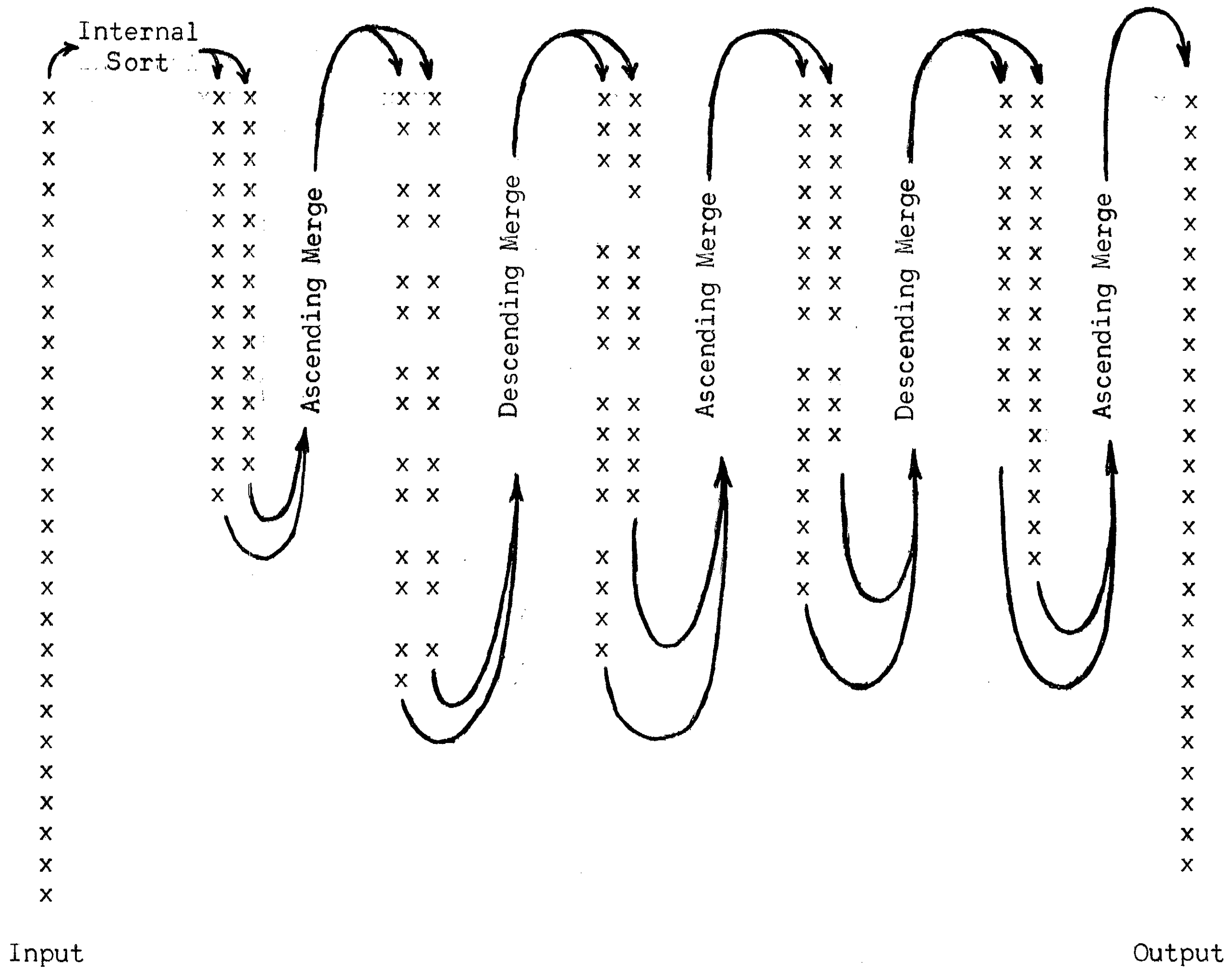
When the items are arranged in ascending fashion from the top of the block to the bottom, they are in descending sequence from botton to top. It is therefore possible to perform a descending merge first when the internal sort produced an ascending sequence for the items in a block. To avoid confusion the item positions in the block are always numbered from top to bottom (A_1, A_2,..., A_6 for ten-word items). To pick up the items from the bottom, the external pass counters, i and j, are set to 6; the additive constant, ℓ, necessary to alter i and j for the next item is set to -1; the constant, L, to which i and j are compared to determine when the last item of a block has been used is set to 1; and n, the reset for i and j, is set to 6. Note in this routine the above is the normal setup of the counters. Only if the first external merge is an ascending one, and then only on the first pass, are the counters set to the values indicated at $\textcircled{7}$. In either case, we first go to subroutine 1 where sentinels are placed on the two output tapes and, after setting the counters appropriately, transfer control to $\textcircled{10}$ where the external sort proper begins.

As indicated at $\textcircled{5}$, an input tape containing only one or two blocks to be sorted must be given special treatment. Consider the case for only one block of data. (This must be a partial block since the first block containing sentinels on T_5 was counted in Δ). First, we note that the block is already sorted, but without the ID block. At $\textcircled{24}$ this single data block is read back into the computer; control is sent to subroutine 4 where the ID block is written on $T_{\odot}$. Control is returned to the current loop; the data block is written on $T_{\odot}$ along with a full sentinel block; the tapes are rewound and then the routine is stopped.

If there were only two blocks of data, control would be transferred to subroutine 4 where the label is placed on the output tape and the variable connector $\textcircled{\phi 2}$ is set since only one merge is necessary. After returning to $\textcircled{7}$, an ascending merge is performed. It should be noted that subroutine 1 is not entered; hence, no sentinels are placed at the top of the output tape.

The following is a description of the merge:

At $\textcircled{10}$ the external collation begins. External collation is merely a succession of two-way "tape" merges which are actually successive merges for many strings of blocks. The merges alternate with each complete pass between ascending and descending sequence. The prime problem in external collation is the determination of the end of a data string. In the desirable case, the strings all consist of the same number of blocks. This number is called the string length. By counting each block merged and comparing this count with the string length counter, the end of the string is easily determined. For all strings to be of equal length, the number of data blocks to be sorted must be a power of two. Usually, this does not occur and therefore strings of partial length will be encountered. As shown in the following example, these partial strings are always at the beginning or the end of a tape.



x Represents a data block
All merging is done by the items within each string
Strings of blocks are spaced to differentiate between strings

The end of partial strings at the front of a tape can be detected by the reading-in of a sentinel block recorded at the front of the tape. The length (number of blocks) of the strings at the back end of the tape may be obtained by counting the number of blocks written. Since only the count for the last string is necessary, this is a feasible procedure.

Initially, the block counters for input, Δa and Δb, and the output block counter Δc are set to zero. The input string length, R, is set to one (one block strings). The input tapes, T_a and T_b, are set to read T_1 and T_2, respectively, and the output tapes T_c and T_d are set to T_3 and T_4.

At $\textcircled{10}$ the two-standby block procedure for reading information from tape is set up. The first data block from T_a is read into A ($\rightarrow\boxed{A \leftarrow T_a}-$ signifies a backward read). The second block from T_a is read into the A standby block, $\bar{A}$. The first block from T_b is read into B, the second T_b block is read into rI. Y, which indicates the location to which the block in rI will be transferred, is set to $\bar{B}$, the B standby block.

At $\textcircled{11}$, the variable connector $\textcircled{\beta}$ is encountered. The connector was set to β_1 if the first merge is to be a descending one and to β_2 if it is to be ascending. At $\textcircled{\beta_1}$ the item with the greatest keyword is selected and sent to working storage. At $\textcircled{\beta_2}$ the item with the smallest keyword is selected. In either case, we go to subroutine 2, where the last item selected is sent to the current output position, C_k. Note that the output block, C, is always filled from the top. Also note, that when each C block is written on the current output tape, $T_{\circledcirc}$, a count is kept by adding one to Δc. After returning from subroutine 2, the next input item is obtained to replace the one just put in the output block, C. Whenever an input block is exhausted, control is transferred to either $\textcircled{14}$ or $\textcircled{15}$. These routines set up the two-standby block read routine so as to make available the next A or B block, as required. In subroutine 3, we find the read routine itself.

Upon returning from subroutine 3, the appropriate input block counter, Δa or Δb, is increased by one, and the new A or B block is obtained from the standby position. As mentioned before, the end of an input string is indicated by either the input block counter becoming equal to the string length counter, R, or the presence of an S sentinel in the block just transferred into A or B from the standby block position.

Since at least one block of data must be present on both T_a and T_b for the first external merge, it is necessary to consider the end of an input string of exactly R blocks. When an input block from one input tape is processed completely, say the A input. Δa is increased by one for the block just processed and the next A block is brought from the standby block, $\bar{A}$, into position A. Next, the input block just obtained is examined for sentinels. If none are present, it means the end of T_a has not yet been reached. Assuming this is the case, the block counter Δa is compared with the string length R. If we have reached the end of a string, control is transferred to ⑯. Since ε_1 is set initially, we will set variable connectors $\textcircled{\zeta}$ and $\textcircled{\gamma}$ to ζ_2 and γ_2. Control is returned to ⑫ to send out the remaining B items of the string. When the B string is completely processed, Δb = R and control is transferred to ⑳, the end of string operations. Here the input and output block counters are reset; the end of string connectors $\textcircled{\gamma}$ $\textcircled{\delta}$ $\textcircled{\varepsilon}$ $\textcircled{\zeta}$ are reset; and the output tape switched. The routine then returns to ⑪ to merge the next strings.

At some time the end-of-tape condition will be obtained. This condition signals end of input string as well. Consider the case where the end of T_a is reached first. Control is sent to ⑱ where the contents of E are placed in the keyword position of A_6 (A_1 if this is the first external merge being done in ascending sequence). If an ascending merge is being done E= "=", otherwise E = $\not<$. θ_2 is then set and control transferred to ⑯. The $\not<$ or = in A_j and the new variable connector settings force all the remaining B items of the current string onto the output tape. Two conditions may then arise: 1) This is the last string on T_b; or 2) There is just one more string left.

In the first case, an S sentinel block will be read on T_b and control will be transferred to ⑲. Since θ_2 is set, control is shunted to ㉑, the end of tape operations. If $\textcircled{\phi_2}$ is set this was the last data pass and the data is completely sorted. Therefore Z sentinels are written on $T_\odot$ and all tapes are rewound. If $\textcircled{\phi_2}$ is still set, there is at least one more data pass. In this case, the input item counters are reset. $\rightarrow\boxed{0 : C}\overset{=}{\underset{\neq}{-}}$ determines which is the current output tape. If this were T_c, for instance, Δa is set to the difference between the length of a complete output string (2R) and the actual length of the last output string. Since it is impossible to have a partial string on back of both output tapes, Δb is reset to zero. The end of tape and end of string connectors are reset, and then the servos are interchanged so that the output tapes for this merge are the input tapes for the next. At ㉒ the contents of E and F are interchanged, since the type of merge (ascending or descending) will be changed. The variable connector $\textcircled{\lambda}$ serves to alternate the setting of $\textcircled{\beta}$. At ㉓ the input string length counter, R, is doubled since the output strings of the just completed merge are now the input strings. If the new input string length is greater than or equal to half the number of input blocks, the next merge is the last, in which case the tape label is recorded on the next output tape and $\textcircled{\theta_2}$ set to write Z sentinels on $T_\odot$ at the end of the next pass.

If this is not the last merge pass, then the new output tapes must be positioned past the two sentinel blocks. Since the two-standby block reading scheme is employed, the reading heads must be at the front of T_a and T_b.

But, because of the slightly different rates of acceleration and deceleration for reading backward and writing forward, it is necessary to read one block forward if writing is to be done following a backward read. This is the purpose of rewinding $T_{\circledcirc}$ before recording the ID block. Thus, instead of re-recording sentinel blocks on the new output tapes, the tapes are repositioned with their read-write heads forward of the sentinels.

The variable connector $\textcircled{\mu}$ is employed only in the exceptional case when the first merge is an ascending one and we were consequently selecting items from the top to the bottom of the input blocks. If this were the case, $\textcircled{\mu 2}$ was set, and control is transferred to $\textcircled{9}$ to reset the counters to their normal form. In either case, we eventually return to $\textcircled{10}$ where the next pass is started.

This explains the case where the last strings on T_a and T_b were reached at the same time. In the case where the end of T_a, and the end of the current, but not the last, B string is reached, there is one remaining B string to merge with no A string. Reaching the end of T_a causes $\textcircled{\theta_2}$, $\textcircled{\zeta 2}$, $\textcircled{\gamma 2}$ to be set as in the first condition described. After the last block of the B string is processed, a sentinel block will not be read into position B. Since $\Delta b = R$, control is transferred $\textcircled{17}$ where ζ_2, previously set, sends us to $\textcircled{20}$, the end of string operations. From there control is transferred to $\textcircled{11}$ to begin merging the new B string with the supposed A string. The $\not>$ or the $=$ in $A \gtrless$ forces all the B items onto the output tape until the T_b sentinel block is obtained. Then control is transferred to $\textcircled{19}$, and since θ_2 is still set and from there to $\textcircled{21}$, the end of tape operations.

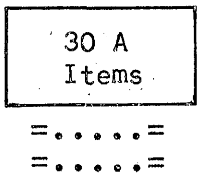
2. Two Methods for Performing Internal Collation

Two methods in common use will be described as means for sorting internally a block of items into an ascending sequence. The first of these methods is a general one and is nearly identical with the external collation technique. The flow chart attached displays this method as adapted for two-word items.

The essence of the method consists of dividing the input data exactly in half and then considering it as composed of one-item strings and finally merging the one-item strings of each half together to form two-item strings. These two-item strings are then merged to form four-item strings and the process continues in like manner until all items are in single sorted string.

The problem of partial strings which is encountered in external collation is easily avoided here since there are 30 two-word items per block and, as $32 = 2^5$, we can simply attach two = sentinel items to the end of our block when it is in the memory and sort 32 items. The sentinels will end up at the end of the sorted items and may then be omitted when the block is written on the output tapes. This is a feasible scheme in internal collation, since the amount of time required in handling the two dummy items is relatively small. It should also be noted that it is not necessary to change the merging from ascending to descending at the end of each pass since the front of the merged items is just as readily accessible as the end of the items. This, of course, is not the case in external collation where the first items merged are separated 2 minutes (tape rewind time for $\frac{1}{2}$ tape) in time from the last items merged. Since the last items merged in external sorting are immediately available to the computer, efficiency dictates that the tapes be read backwards.

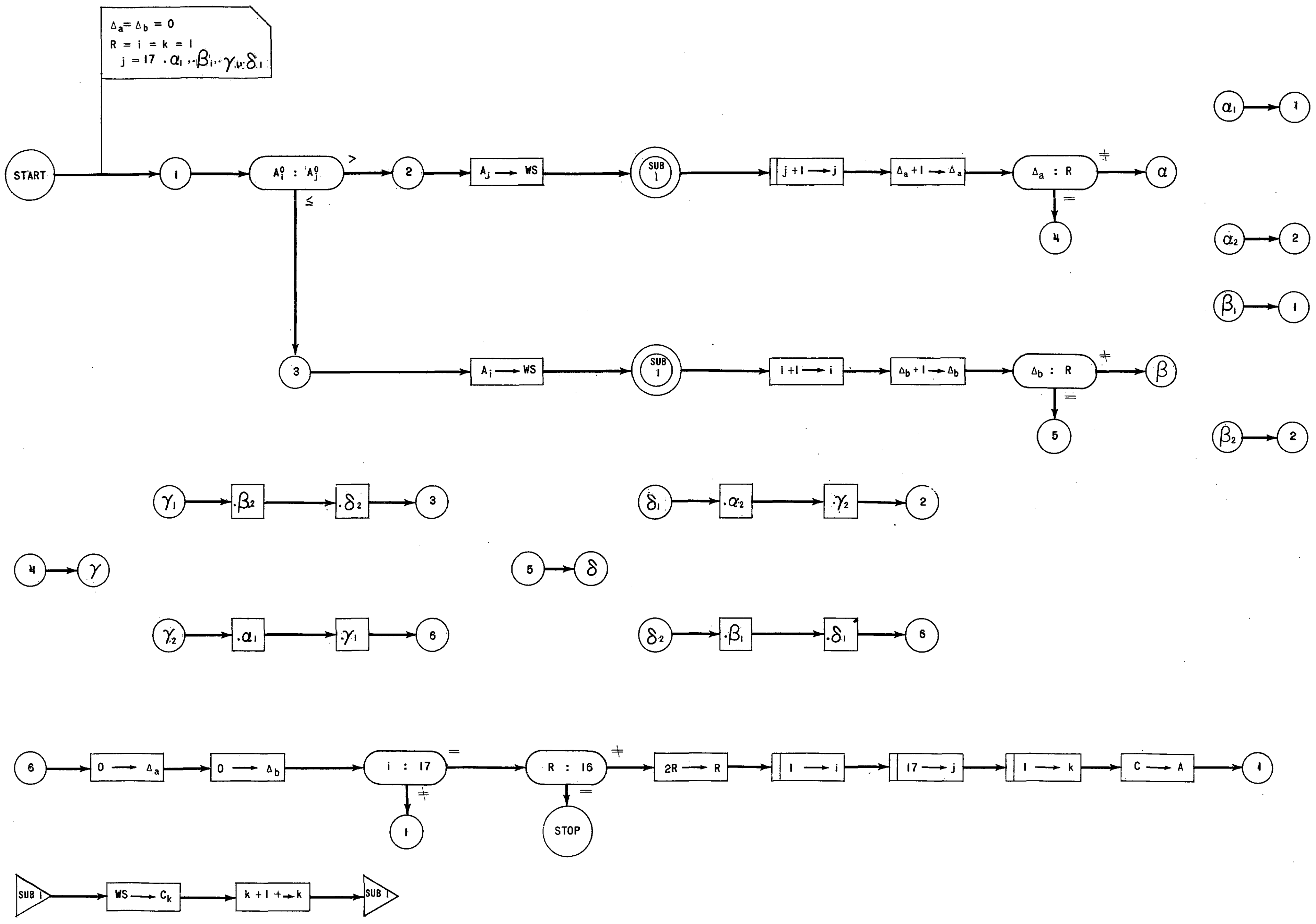
The block of 30 A items to be sorted are read into the memory and a pair of = sentinels are positioned below the last item, making a total of 32 items.



Any one of the first 16 items is labeled A_i where $i = 1, 2, \ldots, 16$. The last 16 items are also labeled A, but any one of these is indicated by A_j where $j = 17, 18, \ldots, 32$. The input string counters are Δa and Δb, while the string length is R. At $\textcircled{1}$, A_1^0 (the superscript 0 designates the key word of the item) is compared with A_{17}^0 the smallest being stored as the output item C_1. The string counter Δa or Δb is increased (which one depends on which item was the smallest) and tested against the string length R. When an input string is exhausted, variable connectors are set to force into C the remaining items of the other string. At the end of both strings control is transferred to $\textcircled{6}$ where i is tested against 17. If the test indicates inequality, there are more strings to merge and control is returned to $\textcircled{1}$. If $i = 17$, then the last string in each set of 16 items has merged. The input string length, R, is examined. If R = 16, then a 32-item string has just been produced and C contains the sorted A block. If this is not the case, R is doubled and i, j, k are reset and the 32 C items become a new set of 32 A items. A_i and A_j are defined as above and the merging is repeated.

It is apparent that the method just described is a perfectly general one and is adaptable to any item size. It is, however, a lengthy one, and the following method is generally used for larger item sizes. The method is illustrated by coding it for ten-word items. The input block is assumed to be in 700-759 and the sorted items (ascending sequence)will be stored in cells 640-699.

The basis of the method is quite simple. The first item of the block is selected and compared with each of the remaining items of the block. The routine assumes that the input block is already in ascending sequence and so assumes A_1 will be placed in 640, A_2 in 650, ..., A_6 in 690, unless



$\Delta_a = \Delta_b = 0$
R = i = k = 1
j = 17 · α_1, · β_1, · γ_1, · δ_1
START
1
$A_i^0 : A_j^0$
>
≤
2
$A_j \rightarrow$ WS
SUB 1
$j+1 \rightarrow j$
$\Delta_a + 1 \rightarrow \Delta_a$
Δ_a : R
≠
=
α
4
3
$A_i \rightarrow$ WS
SUB 1
$i+1 \rightarrow i$
$\Delta_b + 1 \rightarrow \Delta_b$
Δ_b : R
≠
=
β
5
$\alpha_1 \rightarrow 1$
$\alpha_2 \rightarrow 2$
$\beta_1 \rightarrow 1$
$\beta_2 \rightarrow 2$
$\gamma_1 \rightarrow \cdot\beta_2 \rightarrow \cdot\delta_2 \rightarrow 3$
$\delta_1 \rightarrow \cdot\alpha_2 \rightarrow \cdot\gamma_2 \rightarrow 2$
$4 \rightarrow \gamma$
$5 \rightarrow \delta$
$\gamma_2 \rightarrow \cdot\alpha_1 \rightarrow \cdot\gamma_1 \rightarrow 6$
$\delta_2 \rightarrow \cdot\beta_1 \rightarrow \cdot\delta_1 \rightarrow 6$
6
$0 \rightarrow \Delta_a$
$0 \rightarrow \Delta_b$
i : 17
=
≠
1
R : 16
≠
=
STOP
$2R \rightarrow R$
$1 \rightarrow i$
$17 \rightarrow j$
$1 \rightarrow k$
$C \rightarrow A$
1
SUB 1
$WS \rightarrow C_k$
$k+1 \rightarrow k$
SUB 1

otherwise instructed. The first item, A_1, is selected and compared to all the remaining items, A_i. The routine then adds ten to the assumed output destination of A_1 for every A_i which is found to be less than or equal to A_1. At the same time ten is subtracted from the assumed destination of each such A_i. After testing all A_i against A_1, the address of the output item, in which A_1 is to be stored, is known. For example, if the sequence of the input item codes is

Item	Code
A_1	176
A_2	21
A_3	87
A_4	9
A_5	930
A_6	21

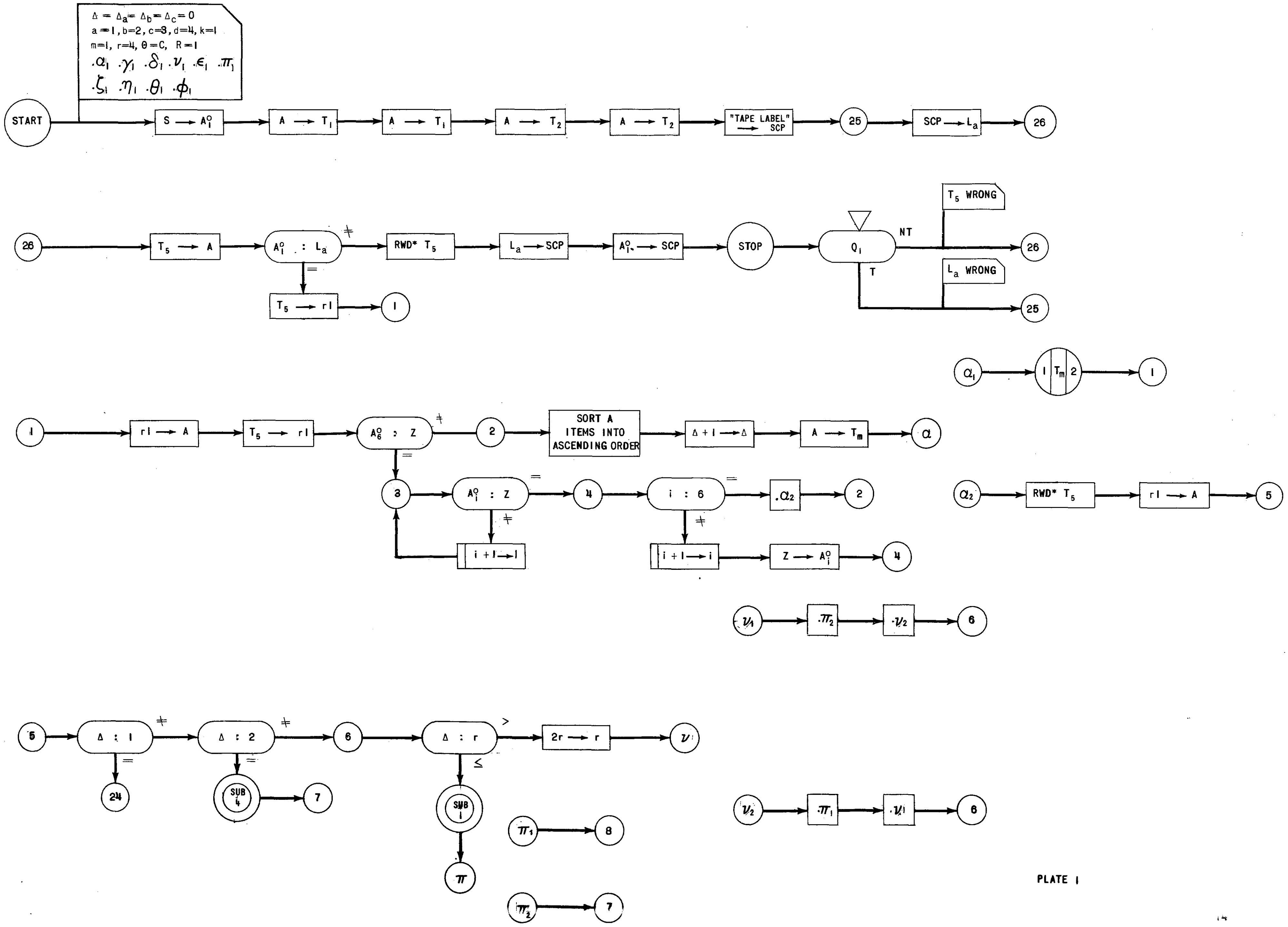
A_1 will be stored in cell 680. A_2 is then compared with the remaining A_i (i = 3, 4, 5, 6) in a similar manner. Thus, A_2 is stored in 660.

The following coding example is the heart of the Internal Sort for ten-word items:

```
                 Input Block   700,...,759
                 Output Block  640,...,699

020   L  700
                ------
021   B  710
               T  025
--------------------
022   B  070
               S  061
023   C  070
               B  041
024   A  061
               C  041
025   B  720
               T  029
--------------------
026   B  071
               S  061
027   C  071
               B  041
```

```
028   A   061
                 C   041
029   B   730
                 T   033
_________________________

030   B   072
                 S   061
031   C   072
                 B   041
032   A   061
                 C   041
033   B   740
                 T   037
_________________________

034   B   073
                 S   061
035   C   073
                 B   041
036   A   061
                 C   041
037   B   750
                 T   041
_________________________

038   B   074
                 S   061
039   C   074
                 B   041
040   A   061
                 C   041
041  [Y   700
                 Z   640]
042  [B40070
                 L   710]
043   C   041
                 B   046
044   A   062
                 C   046
045   B   042
                 A*  064
046   C   042
                 U   021
.
.
.
061   000000
                 000010
062   000000
                 000004
```

```
063

064   010001
                   000010
 .
 .
 .
070  [Y   710
                   Z   650 ]
071  |Y   720              |
                   Z   660 |
072  |Y   730              |
                   Z   670 |
073  |Y   740              |
                   Z   680 |
074  [Y   750              |
                   Z   690 ]
```



Δ = Δa = Δb = Δc = 0
a = 1, b = 2, c = 3, d = 4, k = 1
m = 1, r = 4, θ = C, R = 1
α1 γ1 δ1 ν1 ε1 π1
ζ1 η1 θ1 φ1
START
S → A°1
A → T1
A → T1
A → T2
A → T2
"TAPE LABEL" → SCP
25
SCP → La
26
26
T5 → A
A°1 : La
≠
=
T5 → r1
1
RWD* T5
La → SCP
A°1 → SCP
STOP
Q1
NT
T
T5 WRONG
La WRONG
26
25
α1
1 | Tm | 2
1
1
r1 → A
T5 → r1
A°6 : Z
≠
=
2
SORT A ITEMS INTO ASCENDING ORDER
Δ + 1 → Δ
A → Tm
α
3
A°i : Z
≠
i + 1 → i
4
i : 6
=
≠
α2
2
i + 1 → i
Z → A°i
4
α2
RWD* T5
r1 → A
5
ν1
π2
ν2
6
5
Δ : 1
≠
=
24
Δ : 2
≠
=
SUB 4
7
6
Δ : r
>
≤
2r → r
ν
SUB 1
π
π1
8
π2
7
ν2
π1
ν1
6

PLATE I

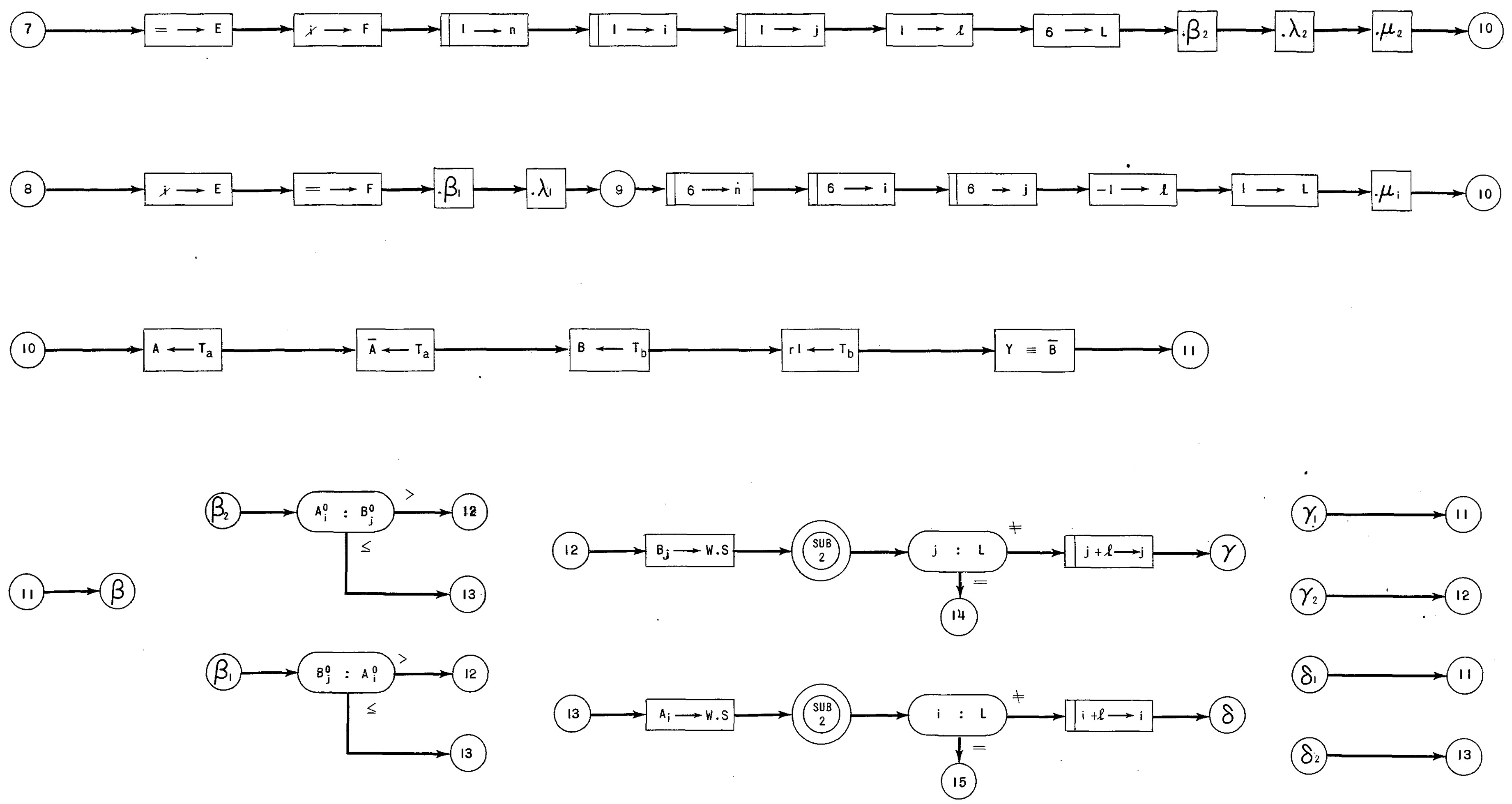
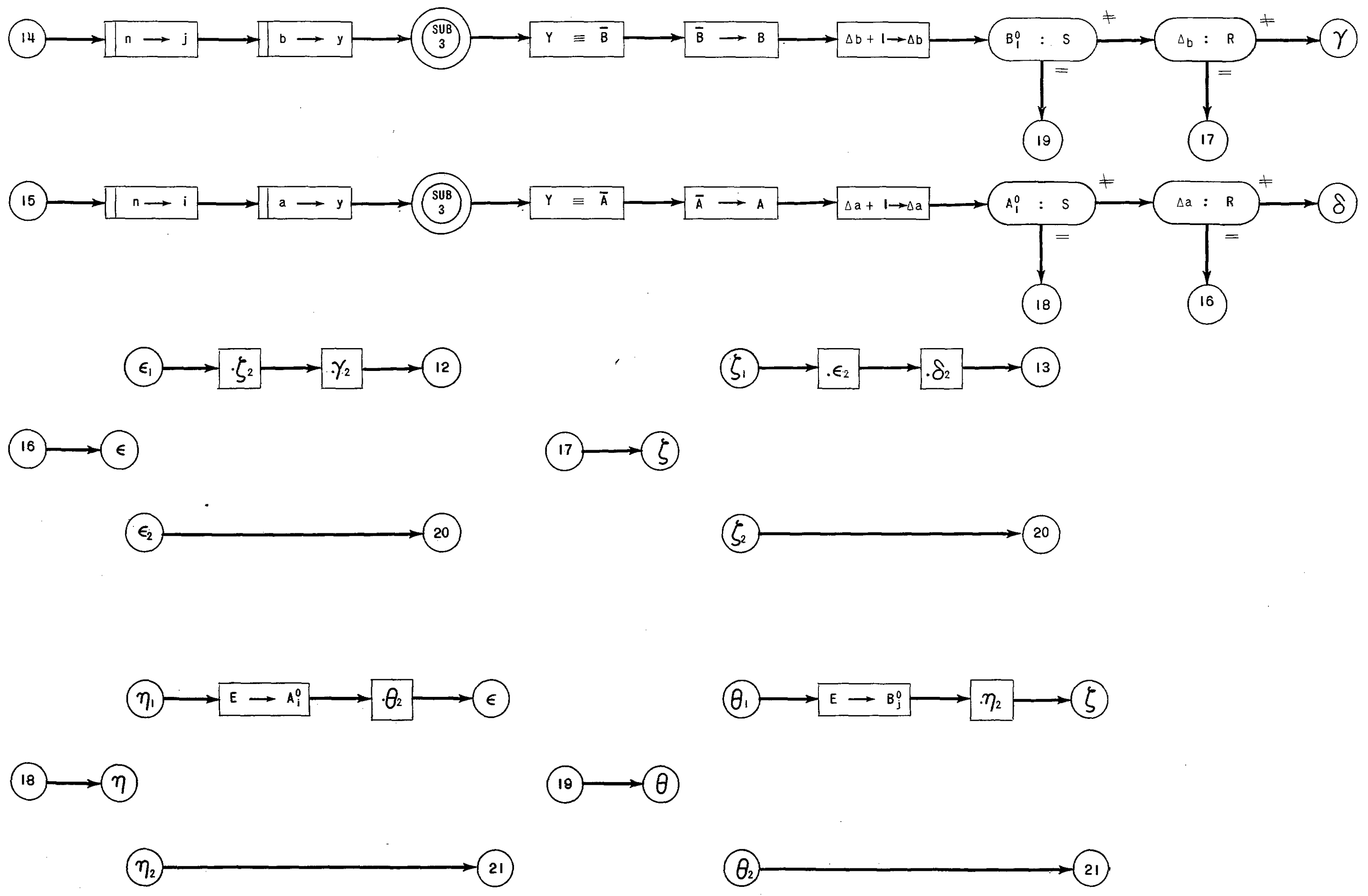

7
= → E
ɣ → F
1 → n
1 → i
1 → j
1 → ℓ
6 → L
.β2
.λ2
.μ2
10
8
ɣ → E
= → F
.β1
.λ1
9
6 → n
6 → i
6 → j
−1 → ℓ
1 → L
.μ1
10
10
A ← Ta
Ā ← Ta
B ← Tb
r1 ← Tb
Y ≡ B̄
11
11
β
β2
A0i : B0j
>
≤
12
13
β1
B0j : A0i
>
≤
12
13
12
Bj → W.S
SUB 2
j : L
≠
=
14
j + ℓ → j
γ
13
Ai → W.S
SUB 2
i : L
≠
=
15
i + ℓ → i
δ
γ1
11
γ2
12
δ1
11
δ2
13

PLATE 2



14
n → j
b → y
SUB 3
Y ≡ B̄
B̄ → B
Δb + 1 → Δb
B_1^0 : S
≠
=
19
Δb : R
17
γ
15
n → i
a → y
Y ≡ Ā
Ā → A
Δa + 1 → Δa
A_1^0 : S
18
Δa : R
16
δ
ε₁
·ζ₂
·γ₂
12
ζ₁
.ε₂
.δ₂
13
16
ε
17
ζ
ε₂
20
ζ₂
η₁
E → A_i^0
·θ₂
θ₁
E → B_j^0
·η₂
18
η
19
θ
η₂
21
θ₂

PLATE 3

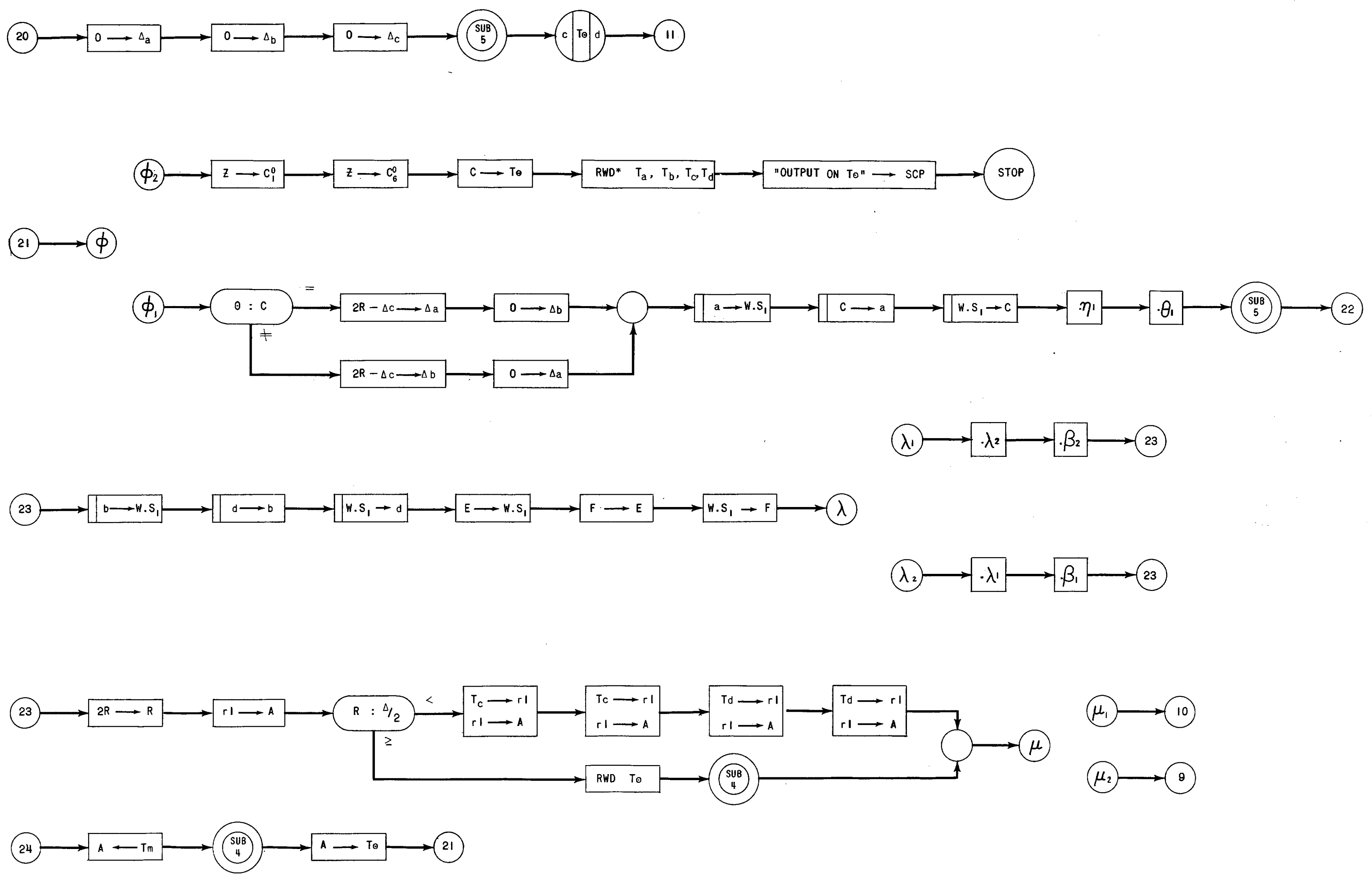
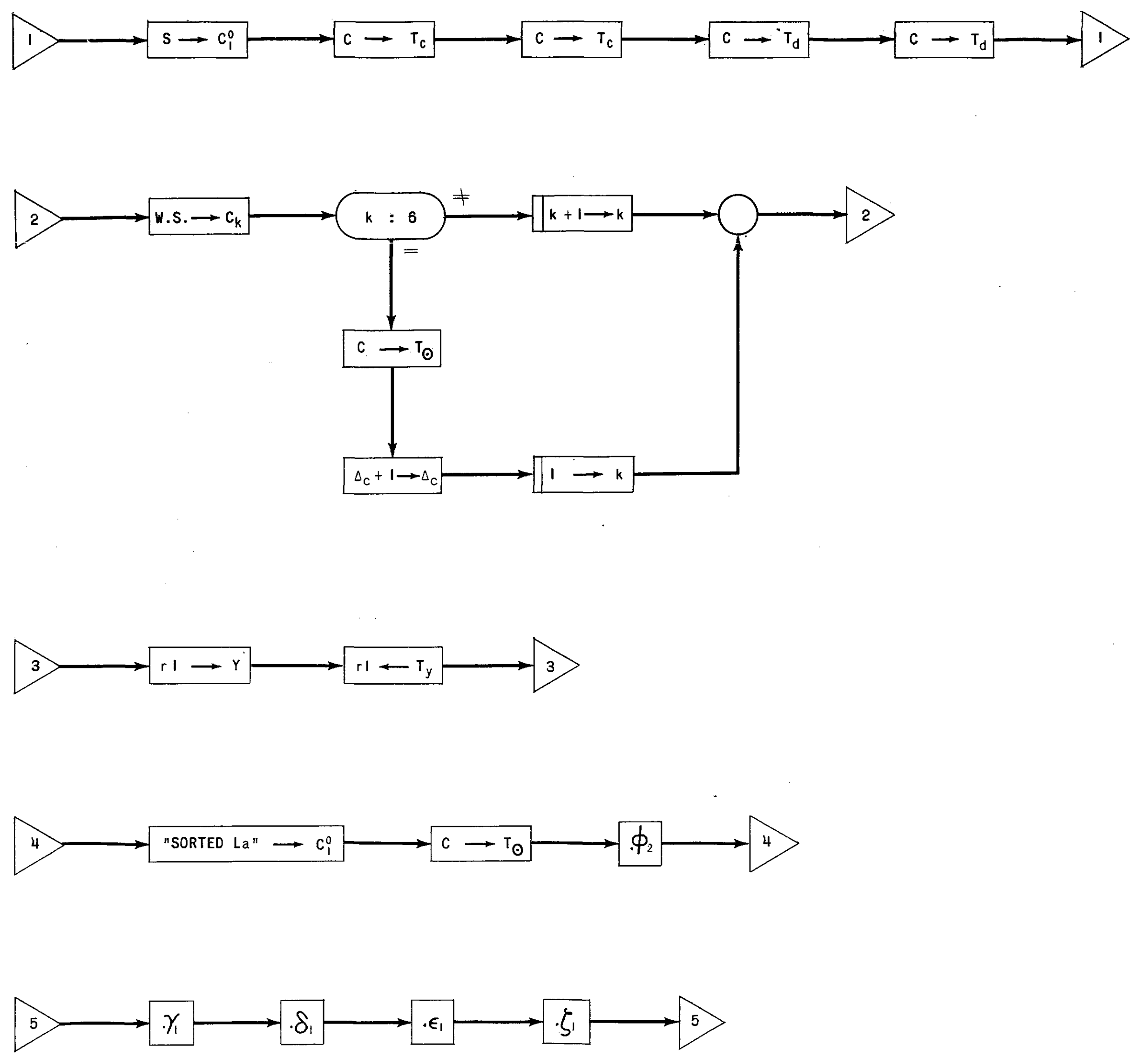

20
0 → Δa
0 → Δb
0 → Δc
SUB 5
c Tθ d
11
φ2
Z → C⁰1
Z → C⁰6
C → Tθ
RWD* Ta, Tb, Tc, Td
"OUTPUT ON Tθ" → SCP
STOP
21
φ
φ1
0 : C
=
≠
2R − Δc → Δa
0 → Δb
2R − Δc → Δb
0 → Δa
a → W.S1
C → a
W.S1 → C
·η1
·θ1
SUB 5
22
λ1
·λ2
·β2
23
23
b → W.S1
d → b
W.S1 → d
E → W.S1
F → E
W.S1 → F
λ
λ2
·λ1
·β1
23
23
2R → R
r I → A
R : Δ/2
<
≥
Tc → r I
r I → A
Tc → r I
r I → A
Td → r I
r I → A
Td → r I
r I → A
RWD Tθ
SUB 4
μ
μ1
10
μ2
9
24
A ← Tm
SUB 4
A → Tθ
21

PLATE 4



1
$S \rightarrow C_1^0$
$C \rightarrow T_c$
$C \rightarrow T_c$
$C \rightarrow T_d$
$C \rightarrow T_d$
1
2
$W.S. \rightarrow C_k$
$k : 6$
$\neq$
$=$
$k+1 \rightarrow k$
$C \rightarrow T_{\odot}$
$\Delta_c + 1 \rightarrow \Delta_c$
$1 \rightarrow k$
2
3
$r1 \rightarrow Y$
$r1 \leftarrow T_y$
3
4
"SORTED La" $\rightarrow C_1^0$
$C \rightarrow T_{\odot}$
$\cdot\phi_2$
4
5
$\cdot\gamma_1$
$\cdot\delta_1$
$\cdot\epsilon_1$
$\cdot\zeta_1$
5

PLATE 5

SERVICE ROUTINES

1. Introduction

The service routines are designed to be a practical aid to the programmer and to improve his efficiency in the use of the computer. The routines of interest here fall into two catagories, correctors and diagnostic routines.

The corrector routines are Mark VIII, which is a manual corrector operating from the Supervisory Control, and AC-2, AC-3, and AC-4, which are automatic correctors proceeding from pseudo-instructions on tape.

In the class of diagnostic routines we have the Codedit, Codecheck, and the Analyser which prepare, in edited form, the input coding and also to some extent an analysis of the coding. In addition, there exist a series of routines known as the Automonitor Routines which give a detailed account of the actual running of a program.

These routines, with the addition of certain more specialized service routines and some of the engineering test routines make up the service routines tape which is generally available at every computer. Proper use of the service routines will materially decrease the amount of programmer and computer time spent on program checks and revisions.

2. Locator

The Locator is always the first routine on the service routines tape. Its function is to read into the computer any of the routines which follow it.

After an initial read is performed, the Locator prints out WHAT BLOCK and sets up a 10 instruction. Type in a word in the form 000 000 00B where BBBB is the block number of the routine desired. The Locator will in effect perform an initial read on the routine desired and stop prior to transferring control to line 000 of that routine. To proceed with the desired routine, merely hit the start bar.

If the block number of the desired routine is not known, depress breakpoint 0 at the start of the routine. Force transfer when Q0 sets up in the static register. The SCP (set at normal) will print out the block number of each service routine on the tape.

The service routines tape is normally mounted on servo one. If it is used on any other servo, force transfer on breakpoint one after the initial read. The SCP will print out 9 SERVO U00000. Type in the servo number of the servo used in the form SSSSSSSSSSSS and the Locator will make the necessary modifications in itself and in the particular service routine desired.

3. MARK VIII

It is sometimes necessary for a programmer to assemble a tape from a group of tapes or make corrections on an existing tape. These operations occur most frequently during the "debugging" stage of programming. In order to avoid a great deal of duplicated effort and to simplify these operations, we make use of the MARK VIII service routine.

The MARK VIII routine operates from control words typed in from the Supervisory Control. Its basic operation is to copy a tape or a portion of a tape onto another tape.*

Two types of manual control are exercised by the programmer. These are Breakpoint Options (BKPT), and type-ins as set up by the routine.

Breakpoint Options

To use these options depress the appropriate breakpoint selector switch and force transfer when the computer stops on that breakpoint number.

After the reading of the MARK VIII instructions or at any time after a Clear C operation, there are two breakpoint options available:

BKPT 1 - Change of Servos: MARK VIII (except when the special methods described below are used) normally copies from servo 2 to servo 3. To use any other servos, force transfer on BKPT 1. The SCP will print 9servoU00044 and set up a 10 m instruction. Type in a word of the form IIIIII000000 where I is the input servo number and O is the output servo number.

BKPT 7 - Change of Write Density: To change the write density from 100/inch to 20/inch or from 20/inch to 100/inch, force transfer on BKPT 7. The write density is not changed by the Clear C operation. After the routine changes the write density, it prints out the new density as follows:

```
730100A00036    20/inch
            or
530100A00036    100/inch
```

*NOTE: The procedure for correcting a tape on Univac requires that the entire tape be recopied inasmuch as a new block inserted in the midst of other information will render the following block unreadable due to the position of the erase head and the differences in the servo's read and write acceleration times.

Control Type-Ins

After breakpoint options, if any, have been exercised, the SCP prints out .6blockΔlmit. At this point there are the following operations which may be executed at the programmer's discretion:

a. Skip through a tape

b. Copy a tape

c. Correct a tape

d. Merge tapes

Skipping Through a Tape: There are two methods available, special and regular.

Special Method. After the SCP types out .6blockΔlmit, type in a word as YS000000BBBB, where S is the servo number and BBBB is the number of blocks to be skipped.
Regular Method. After the SCP types out .6blockΔlmit, type in a word as 00000000BBBB, the SCP will then print out 83next9order. Type in SSSSSSSSSSSS and the operation will be executed. When either of these methods is employed the first word of every skipped block is printed out on the SCP. (A switch on the Supervisory Control may be set to skip these printouts.) In either case the control is transferred to the beginning of MARK VIII again.

Copying a Tape: There are two methods available, special and regular.

Special Method. After .6blockΔlmit appears on the SCP, type in ZXY00000BBBB, where X is the input servo, Y is the output servo, and BBBB is the number of blocks to be copied.
Regular Method. After .6blockΔlmit, type in 00000000BBBB, where BBBB is the number of blocks on the tape to be copied. The SCP will then print out 83next9order, and set up a type in. Type in ZZZZZZZZZZZZ.

Correcting a Tape.

After .6blockΔlmit is printed, type in 00000000BBBB, where BBBB is the number of blocks on the tape to be copied. The SCP will then print out 83next9order. Type in the block and word number of the first word to be corrected as follows: 00BBBB0000WW, where BBBB is the block number and WW is the word number. (The first block is 0001, but the first word in the block is 00.) The SCP will print out the old word and set up a type in.

Type in the new word. The SCP will then again type out 83next9order. Type in the next block and word number and proceed as above until all the corrections have been entered. After the last correction, type in ZZZZZZZZZZZZ instead of a block and word number, and the routine will then complete the copying operation and proceed to the ending routine. The corrections must be in sequence by block number, but not necessarily by word number.

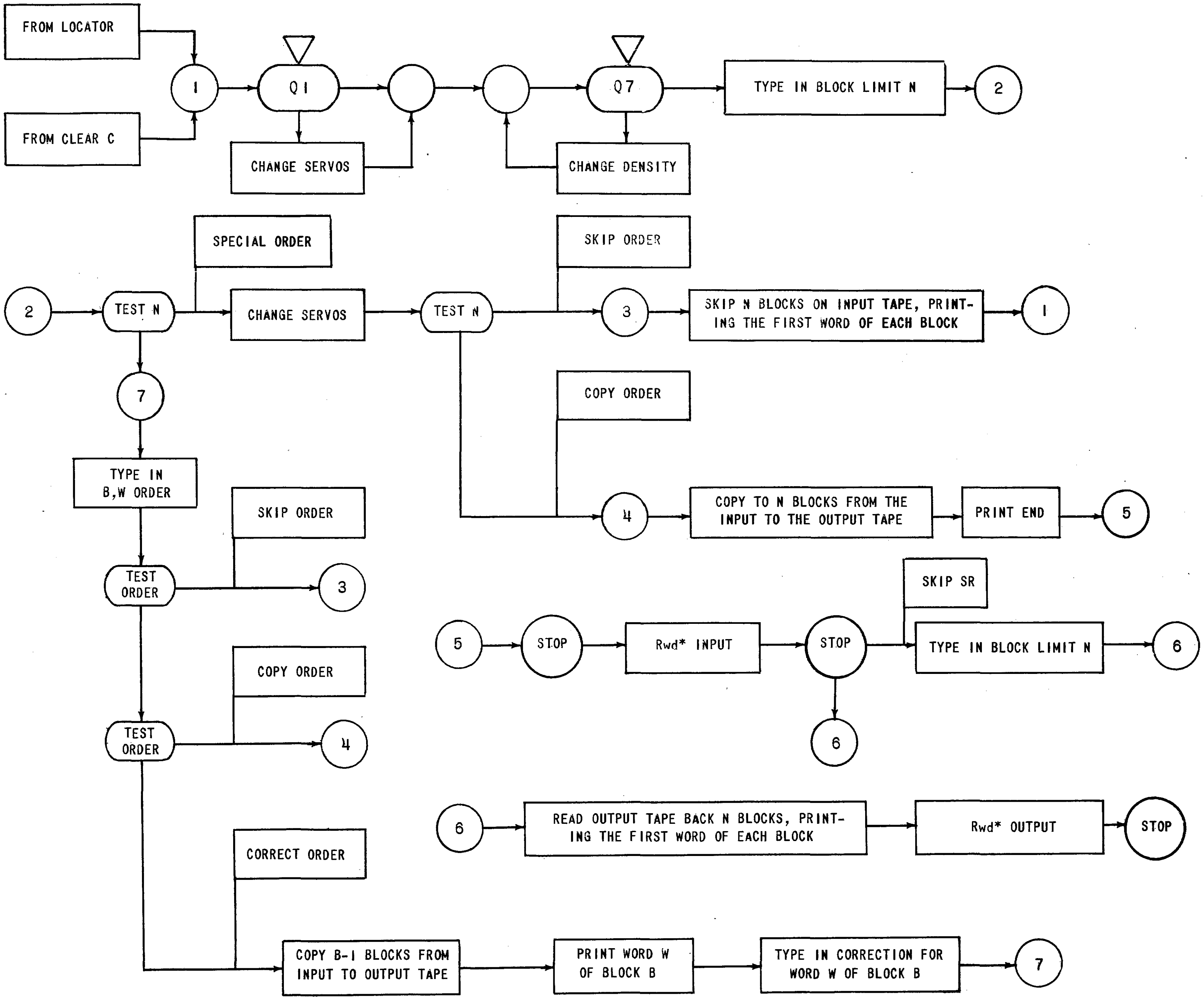
Merging Tapes

Tape merging is executed by a series of the above operations, taking advantage of the fact that it is not necessary to rewind the tapes.

Ending Routine

Upon completion of the copy or correct operations, the computer prints 500038900end. To rewind the input tape, hit the start bar once. (To avoid rewinding the input set up a skip instruction in the static register when the rewind instruction appears on the next δ time.) The computer will stop with a U 044 set up in the static register. Hit the start bar once to read the output tape backward a number of blocks equal to the block limit <u>last</u> typed in. To change this block limit, skip the U 044 and type in, when called for, the number of blocks desired to be read back as 00000000BBBB. The SCP will print the first word of every block read backwards. Upon completion of the backward reads, the computer will rewind the output tape.

When using the MARK VIII routine, the programmer should note that at any time a Clear C operation starts the routine anew, except for changes in write density and servo. Should manual intervention be necessary, it is useful to know that the input block in the memory is at 100 - 159.



FLOW CHART MK VIII

CODING FOR MARK VIII

```
000    [61 000                 Rewind Service Routine Tape
               R   000]        000000 U  001 ⟶ 000
001     L  003
               B   001
002     R  060
               Q1 051          Force Transfer for Servo Change
---------------------------------------------------------------------
003     Yii030                }Transfer Read and Write Orders to
               Z  090         }Working Storage
004     F  055
               Q7 048          Force Transfer to Change Write Density
---------------------------------------------------------------------
005     50 012                 .6BLOC KΔLMIT ⟶ SCP
               10 080          000000 00XXXX  ⟶ 080 (Block Limit)
006     B  080
               T  025          Transfer Control if Special Order
---------------------------------------------------------------------
007     R  095
               U  094          Transfer Control to Read First Block
---------------------------------------------------------------------
008     R  086
               F  055
009     50 041                 83NEXT 9ORDER ⟶ SCP
               10 082          Next Order   ⟶ 082
010     B  082
               T  032          Transfer Control if this is a Copy Order
---------------------------------------------------------------------
011     L  042
               T  021          Transfer Control if this is a Skip Order
---------------------------------------------------------------------
012     .6BLOC
               KΔLMIT
---------------------------------------------------------------------
013     B  096
               Q  017          Transfer Control if this is the Incorrect
                               Block
---------------------------------------------------------------------
014     R  098
               U  097          Transfer Control to Write Current Block
---------------------------------------------------------------------
```

```
015    R   095
                  U   094     Transfer Control to Read Next Block
--------------------------------------------------------------------
016    H   096                Increase Block Counter
                  U   013
--------------------------------------------------------------------
017    L   003
                  B   082
018    T   000                Transfer Control if this is the Last Block
                  ;6  037
--------------------------------------------------------------------
019    E   082
                  A   035
020    C   085                Fabricated Corrector Instructions ⟶ 085
                  U   085
--------------------------------------------------------------------
021    B   036
                  L   080
022    50 100                 First Word of Input Block ⟶ SCP
                  Q   000     Transfer Control if Last Block
--------------------------------------------------------------------
023    R   095
                  U   094     Transfer Control to Read Next Block
--------------------------------------------------------------------
024    A   036                Increase Block Counter
                  U   022
--------------------------------------------------------------------
025    H   082
                  .5  000
026    A   080
                  C   084     Prepare Servo Change Control
027    E   080
                  H   080     Prepare Block Limit
028    R   060
                  U   052     Transfer Control to Servo Change Routine
--------------------------------------------------------------------
029    YZZ030
                  Z   090     Transfer Read, Write, and Block Control
                              to WS
030    R   095
                  U   094     Transfer Control to Read First Block
--------------------------------------------------------------------
```

```
031     L  029
                  U  010        Transfer Control to Copy Test

032     L  080
                  U  013        Transfer Control to Finish Copying

033    [73 100
                  A  036]
034    [12 000
                  30 100]
035     50 100
                  10 100
036     000000
                  000001

037    [53 100                  Write Last Block
                  A  036]
038     50 038                  500038 900END ──→ SCP
                  900END        Stop

039    [82 000                  Rewind Input Tape
                  U  042]       Transfer Control to Test Output Tape Option

040     101111
                  111111
041     83NEXT
                  9ORDER

042     9SERVO
                  U  044

043     50 012                  .6BLOC KΔ LMIT ──→ SCP
                  10 080        000000 00XXXX ──→ 080 (Block Limit)
044     B  080
                  L  036
045     23 000
                  30 100
046     50 100                  First Word of Input Block──→ SCP
                  Q  041        Transfer Control if Last Block

047     S  036                  Decrease Block Counter
                  U  045
```

```
048    B  037
                 F  033
049    G  037
                 C  033
050    50 037               Present Write Density ⟶ SCP
                 U  003
```

```
051    50 042               9SERVO U00044 ⟶ SCP
                 10 084     IIIII 000000 ⟶ 084 (Input-Output Servos)
052    F  040
                 B  084
053    E  034
                 H  034
054    E  039
                 H  039
055    B  084
                 ;61111
056    E  037
                 H  037
057    E  033
                 H  033
058    E  041
                 H  041
059    E  045
                 H  045
```

EXAMPLE PROBLEMS FOR MARK VIII

Problem 1:

Blks 1 through 15 of tape A $\longrightarrow$ tape D

Blks 13 through 17 of tape B $\longrightarrow$ tape D

Blks 1 through 5 of tape C $\longrightarrow$ tape D, correcting

Blk 1 word 07

Blk 3 word 05

Skip through D to test its readability

Solution

Assume tapes A, B, C, D are mounted on servos 2, 3, 4, and 5 and that the MARK VIII is in the memory. The operator's actions are:

1. Type in Z25 000 000 015
2. Hit start bar to rewind tape A
3. Clear C and hit start bar
4. Type in Y30 000 000 012
5. Type in Z35 000 000 005
6. Hit start bar to rewind tape B
7. Set brkp't 1, clear C, and hit start bar
8. Force transfer, release brkp't 1, and hit start bar
9. Type in 444 444 555 555
10. Type in 000 000 000 005
11. Type in 000 001 000 007
12. Type in correction for blk 1 word 07 of tape C
13. Type in 000 003 000 005

14. Type in correction for blk 3 word 05 of tape C

15. Type in ZZZ ZZZ ZZZ ZZZ

16. Hit start bar to rewind tape C

17. Clear SR to zeros and hit start bar

18. Type in 000 000 000 025

Problem 2:

Blks 1 through 2 of tape A $\longrightarrow$ C

Blks 1 through 5 of tape B $\longrightarrow$ C

Blk 3 of tape A $\longrightarrow$ C

Blks 8 through 19 of tape B $\longrightarrow$ C

Skip through C to test its readability and copy at 20/inch for printing.

Solution

Assume A, B, and two blanks are on servos 2, 3, 4, and 5. The operator's actions are:

1. Type in Z24 000 000 002

2. Clear C and hit start bar

3. Type in Z34 000 000 005

4. Clear C and hit start bar

5. Type in Z24 000 000 001

6. Hit start bar to rewind tape A

7. Clear C and hit start bar

8. Type in Y30 000 000 002

9. Type in Z34 000 000 012

10. Hit start bar to rewind tape B

11. Clear SR to zeros and hit start bar

12. Type in 000 000 000 020

13. Clear C, set brkp't 7, drop interlock on T_4, hit start bar

14. Force transfer, release brkp't 7, and hit start bar

15. Type in Z45 000 000 020

16. Hit start bar

17. Hit start bar

4. AC-3

Autocorrector 3 is a single purpose routine designed to perform tape correction only. AC-3 has several features which serve in reducing the possibility of clerical errors as well as programmer and computer time.

Unlike MARK VIII, which operates from Supervisory Control type-ins, the AC-3 is tape controlled. Thus, when only tape correction is to be done and when a large number of corrections are to be made, the AC-3 is more conservative of Univac time than the MARK VIII. In addition to the value of tape controlled operation, the AC-3 has several other features designed as a programmer convenience. They are:

a. The corrections need not be recorded on the control tape in block number sequence as AC-3 sorts the corrections before they are applied.

b. Before each correction is actually applied, it can either be printed on the SCP or written on a tape, providing a permanent record of the corrections is made.

c. In conjunction with item b, options are available for making minor corrections to the control tape information.

The control tape which is mounted on Uniservo 4 consists of a series of two-word items in the following format:

```
OOB BBB OOO OWW
CCC CCC CCC CCC
```

Where BBBB is the number of the block to be corrected ($1 \leq$ BBBB ≤ 2000), WW is the word within that block to be corrected ($00 \leq$ WW ≤ 59), and CCC CCC CCC CCC is the correction for that word. For example, if word 51 of block 27 is to be changed from BOO 301 COO 397 to BOO 351 HOO 397 the correction item would be

```
000 027 000 051
BOO 351 HOO 397
```

A maximum of 329 correction items can be handled at one time with AC-3. To mark the end of the correction list on tape 4, the sentinel ZZZ ZZZ ZZZ ZZZ must follow the last correction item. The remaining information following the sentinel in the block is ignored by AC-3.

The operating instructions for AC-3 are listed below.

1. The tapes to be mounted are as follows

Uniservo	Tape
1	Service routine containing AC-3
2	Tape to be corrected (see 3)
3	Blank tape for corrected output (see 3)
4	AC-3 control tape
5	Blank tape for the correction listing if option under step 3 is to be exercised

2. Initial read Ac-3 through use of Service Routine Locator. Computer will stop after first block of AC-3 is in the memory.

3. Several conditional transfer breakpoint options are now available:

   a. To print the AC-3 operating instructions depress breakpoint zero, set the SCP selector switch to normal, and operate the start bar.

      The computer will stop on a Q0. Force transfer and operate the start bar. The AC-3 operating instructions will print on SCP and the computer will stop again on Q0. At this point Q0 should be released and other options, if they are to be exercised, should have their appropriate breakpoint buttons depressed. Operate start bar to continue.

   b. To modify the Uniservos assigned to the tape to be corrected and the blank to contain the corrected version depress breakpoint one and operate the start bar.

      The computer will stop on a Q1. Release breakpoint one, force transfer and operate the start bar. Computer will print

      ```
                   In    Out   Servo
      ```

      and stall on an input ready. Type in the new Uniservo designations in the following form

      ```
                          XXX XXX YYY YYY
      ```

      where X is the input servo and Y is the output servo. The computer will automatically modify the appropriate instructions and continue.

   c. To have the correction list written on tape 5 depress breakpoint 3 and operate the start bar. This option will be exercised when step 5 is reached.

4. The computer will then print out

   ```
                             block limit
   ```

   and stall on an input ready. Type in the number of blocks on the "tape to be corrected" in this format

   000 000 00X XXX   ($1 \leq \text{XXXX} \leq 2000$)

5. If breakpoint 3 has been depressed, computer will stop on Q3. Release breakpoint 3 and force transfer. If the correction list is known to be correct (e.g. no errors in unityping), the SCP output selector switch should be placed on skip to conserve computer time. Operate the start bar to continue with AC-3.

   If an SCP output is desired, the print selector switch should be set to check and the margins set for 39 digits.

6. When the corrections have been applied, the computer will rewind all tapes and stop.

7. If the option for writing the corrections on tape 5 has been exercised, the uniprinter settings should be

    a. Print selector switch set to check

    b. Margins set for 36 digits

Occasionally it may be desirable to modify the correction information on the control tape without the necessity of retyping it or doing a MARK VIII correction. This, of course, could arise when an error in listing the corrections has been made or in unityping. To expediate this correction of the control tape, a set of additional breakpoint options have been provided. These breakpoints, if option is to be exercised, should be set on step 3 above.

To modify a correction, depress the appropriate breakpoint listed below.

a. To modify the block and word number of a correction depress breakpoint four.

b. To modify the correction itself depress breakpoint five.

c. To skip this correction entirely depress breakpoint six.

When the computer is about to apply a correction, it will print the old word, the block and word number, and the new word, and will then stop on the appropriate breakpoint. Operate the start bar each time the computer stops on the breakpoint until the correction to be modified is printed out. Then for modifications a and b above, force transfer, release the breakpoint and operate the start bar. The computer will stall on an input ready. Then

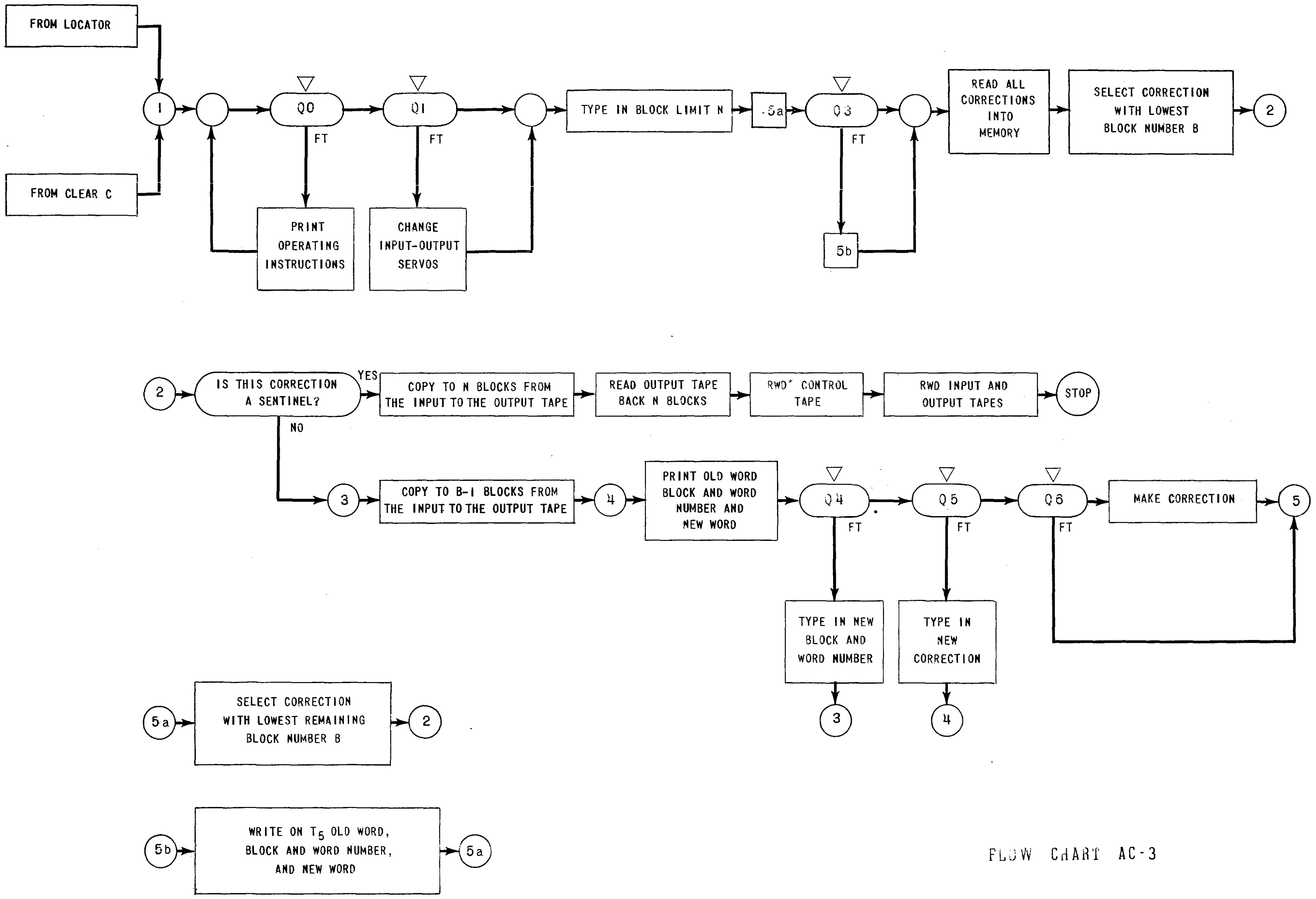
a. To modify the block and word number, type in the new block and word number in the same format as on the control tape. This new block number must not be less than the replaced block number nor larger than the block number of the correction item to follow (this may require scanning the list of corrections manually).

b. To change the correction itself, type in the new correction. If both a block and word number and its correction are to be changed, both breakpoints four and five must be used.

For skipping a correction simply force transfer, release the breakpoint and operate the start bar.

AC-3 may be rerun at any time by operating the clear C switch. This causes the input, output, and control tapes to be rewound. Tape 5 will not be rewound.

A schematic flow chart of AC-3 is appended.



FROM LOCATOR
FROM CLEAR C
1
Q0
FT
PRINT OPERATING INSTRUCTIONS
Q1
FT
CHANGE INPUT-OUTPUT SERVOS
TYPE IN BLOCK LIMIT N
5a
Q3
FT
5b
READ ALL CORRECTIONS INTO MEMORY
SELECT CORRECTION WITH LOWEST BLOCK NUMBER B
2
2
IS THIS CORRECTION A SENTINEL?
YES
COPY TO N BLOCKS FROM THE INPUT TO THE OUTPUT TAPE
READ OUTPUT TAPE BACK N BLOCKS
RWD CONTROL TAPE
RWD INPUT AND OUTPUT TAPES
STOP
NO
3
COPY TO B-1 BLOCKS FROM THE INPUT TO THE OUTPUT TAPE
4
PRINT OLD WORD BLOCK AND WORD NUMBER AND NEW WORD
Q4
FT
TYPE IN NEW BLOCK AND WORD NUMBER
3
Q5
FT
TYPE IN NEW CORRECTION
4
Q6
FT
MAKE CORRECTION
5
5a
SELECT CORRECTION WITH LOWEST REMAINING BLOCK NUMBER B
2
5b
WRITE ON T5 OLD WORD, BLOCK AND WORD NUMBER, AND NEW WORD
5a

FLOW CHART AC-3

5. AC-4

Like AC-3, the Autocorrector 4 is a single purpose routine designed to perform tape correction only. The distinguishing feature of AC-4 over the Mark VIII and AC-3 is that it provides a check on the corrections to be effected. As in AC-3, the correction items are recorded on a control tape and thus computer time is conserved.

The control tape consists of a series of three-word items having the following format:

```
OOB BBB OOO OWW

XXX XXX XXX XXX

CCC CCC CCC CCC
```

Where BBBB is the number of the block to be corrected ($1 \leq$ BBBB ≤ 2000), WW is the word within the block to be corrected ($00 \leq$ WW ≤ 59), XXX XXX XXX XXX is the incorrect word and CCC CCC CCC CCC is the correct word. For example, if word 51 of block 27 is to be corrected from B00 301 C00 397 to B00 351 H00 397 the correction item would be:

```
000 027 000 051

B00 301 C00 397

B00 351 H00 397
```

Following the last correction item will be the sentinel word ZZZ ZZZ ZZZ ZZZ. The remaining information in the correction data block will be ingnored. The correction items must be recorded on the control tape in ascending order by block number.

Briefly, the mode of operation of AC-4 is this: the tape to be corrected is copied onto a blank output tape until an incorrect block is located. The word to be corrected is then compared with the word the programmer expected to be there, as indicated by the second word of the correction item. If they agree, the correction is made by replacing that word of the input tape by the third word of the correction item and the process is repeated for the next correction item. If the comparison shows a disagreement, it may indicate either an error in listing the block and word number or in the unityping of where X is the input servo and Y is the output servo. The computer will automatically modify the appropriate instructions and continue.

4. The computer will then print

Block Limit

and stall on an input ready. Type in the number of blocks on the "tape to be corrected" in this format:

000 000 00X XXX ($1 \leq$ XXXX ≤ 2000)

5. The computer will then proceed with corrections, comparing word two of the correction item with the word in the designated location before the correction is made. Step 8, below, will cover the operation when discrepancies are detected.

6. When all corrections have been applied, the input tapes will be rewound. The output tape will be read backwards to check its legibility and then rewound.

7. AC-4 may be restarted at any time by clearing C.

8. If, during the application of the correction list, a discrepancy is found between the second word of the correction item and the word in the designated location, the following information will print on SCP and the computer will stop:

Block and word number
The word occupying that location
The word expected to be in that location
The word to be put in that location

[If the AC-4 operating instructions are desired, depress Q0 and operate the Start Bar. The computer will stop on a Q0. Force transfer, release the breakpoint and operate the Start Bar. The operating instructions will be printed on SCP and the computer will again stop.]

Several options are now available:

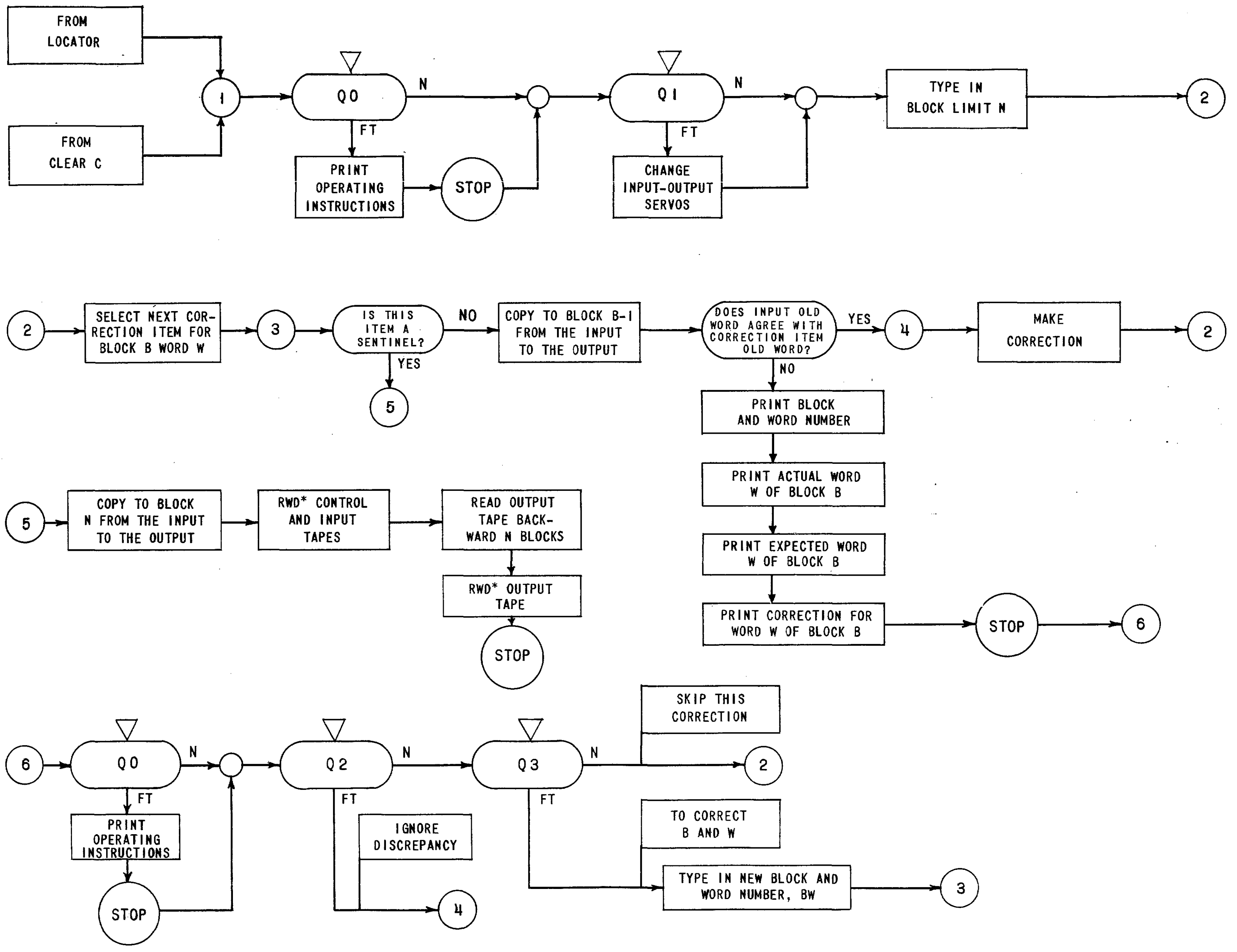
A. To disregard the discrepancy and proceed with the correction, depress breakpoint two, and operate the Start Bar. The computer will stop on a Q2. Force transfer, release the breakpoint and operate the Start Bar.

B. To change the block or word number, depress breakpoint three and operate the Start Bar. The computer will stop on a Q3. Force transfer, release the breakpoint, and operate the Start Bar. The computer will print on SCP

New Blk, Word

and stall on an input ready. Type in the new block and word number. This new block number must not be less than the old one nor greater than the block number for the next correction item. The computer will apply this new correction and continue in normal operation.

C. To skip this correction, simply operate the Start Bar.


FROM LOCATOR
FROM CLEAR C
1
Q0
N
FT
PRINT OPERATING INSTRUCTIONS
STOP
Q1
N
FT
CHANGE INPUT-OUTPUT SERVOS
TYPE IN BLOCK LIMIT N
2
2
SELECT NEXT CORRECTION ITEM FOR BLOCK B WORD W
3
IS THIS ITEM A SENTINEL?
NO
YES
5
COPY TO BLOCK B-1 FROM THE INPUT TO THE OUTPUT
DOES INPUT OLD WORD AGREE WITH CORRECTION ITEM OLD WORD?
YES
4
MAKE CORRECTION
2
NO
PRINT BLOCK AND WORD NUMBER
PRINT ACTUAL WORD W OF BLOCK B
PRINT EXPECTED WORD W OF BLOCK B
PRINT CORRECTION FOR WORD W OF BLOCK B
STOP
6
5
COPY TO BLOCK N FROM THE INPUT TO THE OUTPUT
RWD* CONTROL AND INPUT TAPES
READ OUTPUT TAPE BACKWARD N BLOCKS
RWD* OUTPUT TAPE
STOP
6
Q0
N
FT
PRINT OPERATING INSTRUCTIONS
STOP
Q2
N
FT
IGNORE DISCREPANCY
4
Q3
N
SKIP THIS CORRECTION
2
FT
TO CORRECT B AND W
TYPE IN NEW BLOCK AND WORD NUMBER, BW
3

FLOW CHART
FOR AC-4

6. <u>MA-2</u>

Occasionally, it is desirable to merge the data from two tapes on a word-by-word basis as contrasted to the block merging accomplished by the Mark VIII. Examples of this might be:

a. copying a subroutine into a master routine where the subroutine does not take up an intergral number of blocks.
b. correcting tapes on which words have been left out or extra words added.
c. merging data from two tapes where block-by-block merge is not feasible.

The Automatic Line Merge, MA-2, is a service routine designed for this purpose. Certain other features have been included in MA-2 to increase its flexibility and usefulness.

The basic operation of MA-2 is to transfer words from one or two input tapes designated A and B to an output tape C. The uniservos for A, B, and C are specified by an initial control word. This control word and the control words following it are normally unityped onto a control tape which is mounted on uniservo 4 although a breakpoint option may be exercised to use another uniservo to have the control words typed in manually through SCK. Other breakpoint options are available for minor corrections of the control words which are printed on the SCP before they are executed.

There are three different types of control words which MA-2 recognizes. The first of these must be the first control word received by MA-2. It is called an initial control word and has the following format

ABC 000 000 000

where A and B are the two input uniservo numbers (use zero for B if there is only one input) and C the output uniservo. An initial instruction causes the MA-2 tape instructions to be appropriately adjusted.

The second type of control word is a pseudo instruction instructing MA-2 to perform one of the five following operations:

1. <u>Copy</u> N words from tape A or B to C.
2. <u>Skip</u> N words on tape A or B.
3. <u>Fill</u> tape C with N "special" words.
4. <u>Add</u> the next N words from the control tape (or SCK) to tape C.
5. Read tape C backwards to test its readability, also, print the total number of blocks on the tape, the first word of each block, and then rewind the tape with interlock.

These pseudo instructions are in the following format

CS_iS_o TLL 00B BBB

C is a control digit specifying one of the five operations noted above. In particular if

C = Z	A copy operation is to be performed
C = Y	A skip operation is to be performed
C = F	A fill operation is to be performed
C = A	An add operation is to be performed
C = B	A read back operation is to be performed

S_i is the input tape and it will be either the letter A or B for those operations requiring an input tape.

S_o is the output tape. It is always the letter C for those operations requiring an output tape.

T is of significance only when a fill operation is to be done. In these cases it will be a digit specifying the kind of "special" word to be used in the fill operation. If

T = 0	The fill word is 000 000 000 000
T = 1	The fill word is i̸i̸i̸ i̸i̸i̸ i̸i̸i̸ i̸i̸i̸
T = 2	The fill word is ZZZ ZZZ ZZZ ZZZ

as many as seven other fill words may be specified by inserting the additional fill words in words 13 to 19 of block 3 of MA-2. If this is done, T has the range $0 \leq T \leq 9$. Neither S_i nor S_o need be specified on a fill operation.

$\left.\begin{array}{l}\text{LL}\\\text{BBBB}\end{array}\right\}$ Together specify the number of words affected by the pseudo instruction. If the number of words to be affected are less than 60, BBBB = 0 and $00 \leq LL \leq 59$, while if the number of words are 60 or more BBBB = number of 60 word multiples and LL = remaining words. For example, if the number of words, N, are

N = 15	Then LL = 15 and BBBB = 0000
N = 75	Then LL = 15 and BBBB = 0001
N = 120	Then LL = 00 and BBBB = 0002

The third type of control word is the normal Univac instruction. That is, if a control word (exclusive of the initial control word which is the servo designator) does not have as its left-most digit the letter Z, Y, F, A, or B it is treated as a Univac instruction and executed. In particular a control word

620 000 900 000

will cause tape 2 to be rewound and Univac stopped. Do not expect the contents of the working registers to remain unaltered between the execution of successive control words. This means a control word sequence

```
L00 500 B00 900
000 000 Q00 051
```

will not produce the expected result of comparing (500) and (900) for equality.

The operating instructions for MA-2 are:

1. Tapes to be mounted are as follows

Uniservo	Tape
1	Service routine containing MA-2
4	Control tape (if options under steps 3b and c are not exercised)
A	Input tape A
B	Input tape B (if a second input is used)
C	Blank tape for output

2. Initial read MA-2 through use of Service Routine Locator. Computer will stop after the first block of MA-2 is in the memory. Depress breakpoint seven.

3. Several conditional transfer breakpoint options are now available:

    a. To print the MA-2 operating instructions depress breakpoint zero, set the SCP selector switch to normal, and operate the Start Bar.

       The computer will stop on a Q0. Force transfer and operate the Start Bar. The MA-2 operating instructions will print on SCP and the computer will again stop on Q0. At this point Q0 should be released and other options, if they are to be exercised, should have their appropriate breakpoint buttons depressed. Operate the Start Bar to continue.

    b. To modify the uniservo assigned to the control tape, which is normally #4, depress breakpoint one and operate the Start Bar.

       The computer will stop on a Q1. Force transfer, release breakpoint one, and operate the Start Bar. The computer will print on SCP

TTTTTTTTTTTT

and stall on an input ready. Type in the new uniservo designation for the control tape in this form:

XXXXXXXXXXXX

where X is the uniservo number. The computer will automatically modify the appropriate instructions and continue.

c. To modify MA-2 for manual operation through the SCK, depress breakpoint three and operate the Start Bar.

The computer will stop on a Q3. Force transfer, release breakpoint three and operate the Start Bar. The manual operation of MA-2 is described in step 4b.

4. The computer will then begin the word-by-word merging in the following manner, depending on whether or not the manual option of 3c was exercised:

   a. If the control tape is used, the merging proceeds automatically with MA-2 selecting its instructions from the control tape. Just before each order is executed, it is printed on the SCP thus providing a permanent record of the corrections is made. The merging is stopped only by a 90 000 instruction on the control tape.

   b. If the manual option has been exercised, the computer will print on the SCP

   ABC

   and stall on an input ready. Type in the initial control word (servo specifications). The computer will modify the appropriate tape instructions and stall again on an input ready. Type in the next instruction. The computer will continue calling for instructions until a 90 000 instruction is supplied.

5. The routine may be rerun at any time by clearing C. Occasionally, it may be desirable to modify the instructions

on the control tape without the necessity of retyping it or doing a Mark VIII correction. This could, of course, arise when an error in listing the instructions or in unityping them has been made. A set of additional breakpoint options have been incorporated in MA-2 to facilitate the modification of the control tape. These breakpoints, if their options are to be exercised, should be set on step 3 above:

a. To insert an instruction, depress breakpoint four. Each time the computer stops on a Q4 operate the Start Bar until the instruction just printed on SCP is the one <u>in front of which</u> a new instruction is to be inserted. When this occurs, force transfer, release Q4 (unless further insertions are to be made), and operate the Start Bar.

   The computer will print

   INSERT

   and stall on an input ready. Type in the additional order.

b. To change an instruction, depress breakpoint five. Operate the Start Bar each time the computer stops on a Q5 until the instruction to be changed has just been printed on SCP. then force transfer, release Q5 (unless further instruction changes are to be made), and operate the Start Bar.

   The computer will print

   CHANGE

   and stall on an input ready. Type in the corrected instruction.

c. To delete an instruction, depress breakpoint six. Operate the Start Bar each time the computer stops on a Q6 until the instruction to be deleted has just been printed on the SCP. When this occurs, force transfer, release Q6 (unless further deletions are to be made), and operate the Start Bar.

   The computer will print

   DELETE

   and pass on to the next instruction.

Certain instruction errors are automatically detected by MA-2:

a. If a partially filled output block is still in the computer at the time a B instruction (read output tape backwards) is received, the computer prints

```
TC BKS  xxxx
OUTPUT BLOCK
NOT FULL
Hyy 5zz U00107
```

where xxxx is the total number of blocks on C and zz + 1 is the number of words in the partially filled output block. The computer will stop on a Q7. If the remainder of the block is to be filled with zeros, simply operate the Start Bar. If other corrective action is necessary, force transfer, set one of the breakpoints listed in the error conditions noted above, and operate the Start Bar. The computer will stop on the selected breakpoint and the operations discussed in the previous paragraph should be followed.

b. If a copy or skip order is supplied to MA-2 without listing the input servo, the computer will print

```
ERROR SET BP
```

and stop. Set one of the breakpoints 4, 5, or 6 and operate the Start Bar. The computer will stop on the selected breakpoint and the operations discussed for that breakpoint should be followed.

The following examples will illustrate the use of MA-2:

Example 1

Prepare an output tape, C, containing the following parts of input tapes A and B:

BLK 1 W 00	to BLK 2 W 16	from A
BLK 1 W 10	to BLK 1 W 19	from B
BLK 2 W 17	to BLK 2 W 59	from A
BLK 2 W 00	to BLK 2 W 49	from B

Let tapes A, B, and C (blank) be mounted on servos 2, 3, and 5, respectively. The control tape (or manual type-ins) would then contain

Instructions	Explanation
235 000 000 000	A = T_2, B = T_3, C = T_5
ZAC 017 000 001	Copy 77 words from A to C
YBO 010 000 000	Skip 10 words on B
ZBC 010 000 000	Copy 10 words from B to C
ZAC 043 000 000	Copy 43 words from A to C
YBO 040 000 000	Skip 40 words from B to C
ZBC 050 000 000	Copy 50 words from B to C
620 000 630 000	Rewind A and B
B00 000 000 000	Read C back and rewind
900 000 000 000	Stop

Example 2

Tape A contains 4 blocks. The first 6 words are heading, the remainder is data. Transform the heading into an identification block filling the remainder of the block with zeros. Follow this ID block with the data placing Z sentinels after the data and add two sentinel blocks.

Let tape A be mounted on servo 2 and a blank, C, be mounted on servo 5. The control tape would then contain

Instructions	Explanation
205 000 000 000	A = T_2, C = T_5
ZAC 006 000 000	Copy 6 words from A to C
F00 054 000 000	Fill C with 54 words of zeros
ZAC 054 000 003	Copy 234 words from A to C
F00 206 000 002	Fill C with 126 sentinel words
B00 000 000 000	Read C back and rewind
620 000 900 000	Rewind A and stop

Example 3

Tape A is a unityped program tape, 2 blocks in length. In proofreading the unityping it was found that words 33 and 34 of block 2 were left out, and everything beyond these words, therefore, has been shifted, the last two words being zero. These words should be B00 220 L00 135 and A00 235 T00 120 respectively. Tape B is a 19 block program tape. Word 12 of block 10 of tape B is to be corrected to read B00 110 H00 132. Tape A is to replace that part of tape B from word 20 of block 15 to word 19 of block 17. In addition, a block of ignores is to be attached to the completed program tape.

Let tapes A and B be mounted on servos 2 and 3 and a blank tape, C, be mounted on 3. The control tape contents are:

Instructions	Explanation
235 000 000 000	$A = T_2$, $B = T_3$, $C = T_5$
ZBC 012 000 010	Copy 612 words from B to C
YB0 001 000 000	Skip the word to be corrected
A00 001 000 000	Add 1 word to C
B00 110 H00 132	Word to be added
ZBC 007 000 005	Copy 307 words from B to C
YB0 000 000 002	Skip 120 words on B
ZAC 033 000 001	Copy 93 words from A to C
A00 002 000 000	Add 2 words to C
B00 220 L00 135	Word to be added
A00 235 T00 120	Word to be added
ZAC 025 000 000	Copy 25 words from A to C
ZBC 040 000 002	Copy 160 words from B to C
F00 100 000 001	Fill C with 60 ignore words
620 000 630 000	Rewind A and B
B00 000 000 000	Read C back and rewind
900 000 000 000	Stop

7. <u>HERB I</u>

HERB I is a single purpose routine designed to automatically compare for identity the contents of two tapes. Whenever the routine finds a discrepancy, the two words and their block and word number is printed on SCP. An option is provided for producing an output tape that is a synthesis of the input tapes.

There are a number of computer operations that may warrant the use of an auto-comparator such as HERB I. Some of these are listed below:

a. as a check on the manual operations of MK VIII such as copying a tape or correcting a tape.
b. as a means of spotlighting the differences between two supposed identical routines.
c. as a means of verifying the accuracy of a duplicated unityping job and at the same time providing a corrected output.

The operating instructions for HERB I are listed below

1. The service routine tape is mounted on Uniservo 1. The two tapes to be compared are mounted on servos A and B (any servos may be used). If an output tape is desired, mount a blank tape on servo C.

2. Set the SCP Selector Switch on computer digit. Set the margins for
   a) No output tape: 39 digits
   b) Output tape: 52 digits

3. Initial read Herb I through use of the Service Routine Locator. Computer will stop with the first block of HERB I in the memory.

4. If the output tape feature is desired, set breakpoints 4 and 5.

5. Operate Start Bar.

6. Computer will stall on an input ready. Type in a control word in the following format:

   AAA BBB CCC xxx

   where A and B are the uniservo numbers of the two tapes to be compared, C is the uniservo number of the desired tape (C = 0 if no output tape option is being used), and xxx is the number of blocks on the input tapes to be compared.

7. The computer will begin comparing A and B word-by-word for identity. If C $\neq$ 0, the routine will copy each A word passing the identity test onto the output tape. The operation of the routine when a discrepancy between A and B is detected is covered in step 10.

8. When the two input tapes have been compared the desired number of blocks, the input and output (if used) tapes are rewound and the computer stopped.

9. To rerun at any time clear C. This will cause A, B and C (if used) to be rewound and the routine will pick up at step 6.

10. In case a discrepancy between A and B is encountered by the routine, three words will be printed on SCP in the following sequence:

1. word from tape A
2. block and word number of the discrepancy
3. word from tape B

if the output tape option is not being used, the routine returns to the comparison of the remaining words on the input tapes.

If an output tape option is used, the computer will stop on breakpoint 4 after the printout. The following options are now available:

a. To allow the A word to appear on the output tape, operate the Start Bar. The computer will stop on breakpoint 5. Operate the Start Bar. The A word will again be printed on SCP.

b. To allow the B word to appear on the output tape, operate the Start Bar. The computer will stop on breakpoint 5. Force transfer and operate the Start Bar. The B word will again be printed on SCP.

c. To allow a new word to appear on the output tape, force transfer and operate the Start Bar. The computer will stall on a type in. Type in the desired word.

After one of the above options have been exercised, the computer will continue with the comparisons.

PROGRAMMING MANUAL FOR THE HIGH-SPEED PRINTER

1. General Characteristics

The Remington Rand High-Speed Printer converts information stored in the form of magnetic pulses (the Univac XS-3 code) into a visible printed record. The speed of conversion is much greater than anything heretofore commercially available. Some general characteristic of the printer are listed below:

Characters printed per line	130
Lines printed per minute, optional at	200, 400, or 600
Horizontal character spacing	10 per inch
Vertical character spacing, optional at	6, 3, or 2 per inch
Number of different printable characters	51
Printing format control	plug board and paper loop

In brief, the High-Speed Printer operates in the following fashion. Coded data is read from the magnetic tape in groups of 120 digits called blockettes. Each blockette is stored in a memory with the data being in the same coded form as on the tape. A continuously revolving shaft, on which are mounted the 130 typewheels (actually 65 double wheels), also carries a commutator which causes to be generated sequentially the code for each of the characters on the typewheels. This coded representation is sent to a 120 place comparator which also receives in the same code the contents of the 120 place memory. Whenever agreement occurs, which may, of course, be in many places of the comparator at once, signals leave those places on separate wires.

Opposite each typewheel there is a hammer which can be driven against it. The paper, with an inked ribbon or special carbon paper in front of it, passes between the typewheels and hammers, and it is by making a hammer drive the local portion of the paper against a typewheel that printing is achieved.

The comparator output signals mentioned above ultimately release the energy which drives the hammers. Before doing so however, they pass through a plugboard by means of which they can be routed to any desired hammer. Thus any memory position can be made to print out in any print position.

Time is allowed for one complete typewheel revolution so that every character in the memory has a chance to print. Then the paper is spaced, a new blockette of information is read in and the cycle repeated.

Physically, the High-Speed Printer is composed of four units: the Uniservo, the Printer Unit, the Memory Unit, and the Power Supply Unit. These units are interconnected by means of cables. Figure 1 is a picture of the printer.

The above description of the printer implied that the magnetic tape input must be in blockette form. On the middle right quarter of the Univac Supervisory Control Panel is a series of ten Block Sub-Divider Buttons. Each button corresponds to a Uniservo. When one or more buttons are depressed, each block written on the corresponding Uniservos will be in blockette form. That is, the computer will automatically interrupt the write instructions for those Uniservos for a short period after each multiple of 120 digits has been written. Thus each block of 720 characters will be composed of 6 sets of 120 character blockettes, a small blank space appearing between each blockette.

Since the Tape-To-Card Converter also requires magnetic tape recorded in blockette form, but at a somewhat different spacing for its efficient operation, the Block Sub-Divider Buttons for Uniservos 1, 2, 3, 4, 5, 6, 7 contain the appropriate delays for High-Speed Printer tapes while the buttons for Uniservos 8, 9, - have delays appropriate for the Tape-To-Card Converter.

To prepare High-Speed Printer tapes on the Univac, the programmer need merely depress the Block Sub-Divider Button corresponding to the Uniservo he has selected from among those listed above to receive this output. All write orders for these Uniservos should be 5n m's. Because of the extra space needed between blockettes, a 1500' reel of tape can contain up to 7500 blockettes (= 1250 blocks). In making time estimates, the programmer should note that because of the reduced number of blocks there are fewer starts and stops, and thus a full tape recorded in High-Speed Printer blockettes can be made by the computer in 3.5 minutes.

2. Printing Format Control Features

In addition to the ability to print the contents of a memory location anywhere on the page, the High-Speed Printer also possesses several other highly useful editing facilities.

Zero Suppression. It is frequently desirable to suppress the printing of zeros to the left of the first significant (non-zero) digit in a result. It may be required to do this in several areas along a line of printing. By means of the plug board on the printer, this may automatically be done in as many as 18 independent arbitrary areas(or fields).

Fast Feed. On many forms there will frequently be areas where no printing is to occur. Rather than step through these areas a line at a time without printing anything, which would not only be slow but would also require putting "blank" blockettes of information on the tape, thus wasting tape and computer time as well, the paper can be continuously moved at high speed from one printing area to the next. This operation is known as "Fast Feeding". It is controlled by means of a punched paper loop which is placed on the printer and moves in synchronism with the paper. The Fast Feed can be started by means of either a hole punched in the paper loop (in a certain channel) or by any of several special coded combinations written on the magnetic tape. It is stopped by sensing the appropriate punched holes on the paper loop.

Multiline. Normally the contents of the memory (one blockette or 120 digits) is printed out on a single line. By means of the "Multiline" symbol, placed at the beginning of a blockette of information, and in conjunction with the plugboard, a blockette can be broken up and printed out in as many as six consecutive lines. This feature may permit great savings of tape. An obvious use is in name-and-address printing.

Multiple Printing. By means of the plugboard, the contents of a memory location can be printed in not just one but in two or three places across the line. In addition, if the High-Speed Printer is operating in Multiline, the additional printings do not even have to be on the same line (subject to certain restrictions to be discussed later). Of course, no more than 130 characters can be printed on any one line.

3. Wiring For Single Line Printing

For printing of this nature; no Multiline, Multiple Printing, or Zero Suppression; only the top panel of the plugboard is involved. The complete plugboard is shown in Figure 3. The first 5 columns of holes on the left of the top panel of the plugboard are numbered 1-24, 25-48, . . . , 97—120 . These holes are connected to the comparator output lines and are in one-to-one correspondence with the 120 positions of the memory. In the next section of the plugboard to the right, there are 130 pairs of holes, each pair corresponding to a printing position. The two holes of a pair are vertically one above the other and are internally connected together. (The reason for using two holes is to have a free one into which a second jumper can be put so that the comparator signal can be sent to more than one point. This is necessary in Multiple and Multiline Printing.)

It is only necessary now to connect by means of jumpers the set of 120 holes, corresponding to the memory locations, to the set of 130 pairs of holes (that is, to one hole in each pair) corresponding to the hammers or printing positions. This can be done in one-to-one or any other desired pattern. If it is not required to print certain memory locations, or if it is known that certain locations will always contain non-printing characters, these may be left unplugged.

For most purposes, 6, 8, or 10 inch jumpers will be found best. Short jumpers make for a tight, neat-looking board but have the disadvantage that it becomes difficult to enter the heart of a heavily plugged-up area to change or add a jumper.

For electrical reasons only, it is necessary to place a few jumpers in the Multiline section of the plugboard even though no Multiline Printing is to be done. Connect (upper panel):

32A to 33F
33A to 34A
33B to 34B
33C to 34C
33D to 34D
33E to 34E

There is one more thing that must be done before the board can be used. The bottom four holes of the 16th column of the board determine the line spacing, i.e., whether single space, double space or triple space. A jumper is plugged from the bottom hole (marked "out") to any one of the three above (marked "1", "2", "3" and corresponding to single space, double space and triple space, respectively).

4. Modes of Operation

Under normal operation, the High-Speed Printer will stop printing (and reading tape) whenever a blockette is read which contains the printer stop symbol Σ (1 11 0000). The blockette containing the stop symbol is completely read into the memory but is not printed. If a blockette is read into the memory which contains a printer breakpoint symbol β (0 11 0001) and the breakpoint switch is in the breakpoint position, the printer interprets the β as a Σ. The blockette containing the β or Σ can be printed if desired.

When the "Print, No Read - Read, No Print" switch is placed in the "Read, No Print" position, the printer will read a tape without printing until a stop or breakpoint symbol is encountered. It is also possible to skip down a tape in either the forward or backward direction and stop when a Fast Feed I symbol (0 01 1111), printer stop, or breakpoint symbol is encountered.

There are 63 possible UNIVAC code combinations, only 51 of which appear as characters on the typewheels of the High-Speed Printer. The other 12 characters are normally not printed and are either completely ignored or serve as editing symbols, such as the stop and breakpoint mentioned above.

At certain times, however, it may be desirable to have a visual record of <u>all</u> the characters on the tape including the normally non-printing ones. This can be done by operating the Computer Digit switch. The effect of this is to cause each blockette to print out in two lines.

On the first line only the normal printing characters appear. On the second line, any normally non-printing symbols which may be present in the blockette will print, and only such symbols. They will print as regular characters and can be identified by the following table.

Pulse Code	Name	Computer Digit Print-Out
1 00 0000	Ignore	5
0 00 0001	Space	6
0 01 0000	Multiline	E
1 01 1110	¢	C
0 01 1111	Fast Feed I	D
0 10 0000	Tab	N
1 10 0001	Tab	O
1 10 0010	Fast Feed II	P
0 10 1111	Fast Feed III	M
1 11 0000	Stop	V
0 11 0001	Breakpoint	W
0 11 1110	Fast Feed IV	T

Note that in each pair of lines comprising a blockette, one and only one character should be printed in every column.

5. Zero Suppression

Zero suppression is controlled by the lower panel of the plugboard. As mentioned previously, up to 18 fields of arbitrary length may be employed.

It should be remembered that the zeros are suppressed on read-in to the memory, not on print-out, so that the way in which the memory is plugged to the printing positions must be borne in mind.

Single Line Printing Only. The holes in columns 26 through 31 on the lower panel of the plugboard (PB II) are numbered 1 to 120 and correspond to the 120 positions of the memory, or the 120 digits coming from the tape. To start a zero suppression field in "normal" or Single Line Printing, plug from the hole corresponding to the address where the first zero is to be deleted (if present) to the first of the 18 numbered holes in column 25 (hole 25A). Then plug from hole 23A (labeled "Start (SL)"), to hole 24A. If there is to be more than one field to undergo zero suppression, plug the starting positions of the successive fields to the 2nd, 3rd, 4th, etc. holes in column 25 and then plug 23B to 24B, 23C to 24C, etc.

To end each field, plug from the address corresponding to the last zero to be deleted (if everything has been zero up to that point, of course) to successive holes in column 32. Then plug 33A to 34A, and if several fields are to undergo zero suppression, plug 33B to 34B, 33C to 34C, etc.

The fact that the High-Speed Printer can be run in either "Multiline" or "Single Line" (Normal) fashion has already been discussed. Since these two modes of operation, both of which may occur in a run, in general produce different formats, it may be desirable to observe one set of zero fields on single line printing and a different set when on Multiline. There are three possible cases to consider.

Fields for Single Line and Multiline Printing different. To do this, plug the fields for Single Line Printing as described above. They will not be observed when the printer is on Multiline. For the Multiline fields, plug the starting positions successively to holes 25R, 25Q, 25P, etc. (working upwards). Then plug 23S to 24S (Labeled "Start (ML)"), 23R to 24R, etc., terminating one hole below the topmost hole plugged in column 25 (in the Multiline group).

Plug the ending positions for the Multiline fields into holes 32R, 32Q, 32P, etc., again working upward, and then plug 33S to 34S, 33R to 34R, etc., terminating, as before, one hole below the top hole plugged in column 32. The Multiline fields will not be observed when the printer is on Single Line.

Fields for Single Line and Multiline identical. If the fields to undergo zero suppression are exactly the same, that is, occupy the same positions in the blockette, for Single Line and Multiline, the plugging is simplified. Plug the fields as for Single Line. Then run a jumper from the first unused hole in column 23 to the hole marked "Start (ML)" which is 24S. Likewise, run a jumper from the first unused hole in column 33 to "End (ML", hole 34S. All the fields will now be observed both on Normal and Multiline printing.

Some fields common to Single Line and Multiline. When only some of the zero suppression fields are common to both Multiline and Single Line Printing while others are distinct to one or the other, the situation is somewhat more complex. Here "Y" jumpers must be used for the common fields. The stem of the "Y" is plugged into the address location and then one fork is plugged in with the upper (Single Line) group in column 25 (or 32) and the other fork is plugged to the lower (Multiline) section of column 25 (or 32). The other (non-common) fields are plugged as before. The jumpers between columns 23 and 24 and between columns 33 and 34 are run as previously.

Two more points with respect to Zero Suppression should be noted:

1. If one Zero Suppression field follows another immediately (e.g., 22-37, 38-51), the ending point of the first field need not be plugged.

2. If a single digit field is to undergo zero suppression, a "Y" jumper must be used. The stem of the "Y" is plugged into the address location and one fork goes to column 25 (start) while the other goes to 32 (end). If the single digit field is followed immediately by another field (as in algebraic sign followed by number) then, as stated above, the ending of the single digit field need not be plugged and a straight jumper can be used (to plug the start).

6. Fast Feed

There are four magnetic tape symbols which may be used to initiate a Fast Feed. They are:

Pulse Code	Name	Modified Unityper I and Unityper II Symbol
0 01 1111	FF I	@
1 10 0001	FF II	|
0 10 1111	FF III	?
0 11 1110	FF IV	=

The Fast Feed symbol used is not printed and it will be observed as a Fast Feed instruction only if it is placed in the <u>first</u> digit position of a blockette. The fast-feeding is accomplished <u>before</u> the blockette containing the symbol is printed.

As mentioned in Section 4, it is desirable that the first blockette of a new form (or form group) should employ an FF I: otherwise there is no restriction on which Fast Feed symbols are used.

When started, a Fast Feed will continue feeding paper without printing or reading until a hole is encountered in that channel of the Paper Control Loop corresponding to the Fast Feed symbol (e.g., a Fast Feed started by a FF II symbol is stopped only by encountering a hole in channel 2 of the Paper Control Loop). A Fast Feed can also be started by punching a hole in channel 5 of the Paper Control Loop. This type of Fast Feed is stopped only by a hole sensed in channel 6. This "Loop Controlled" Fast Feed is useful in taking care of "overflow" from one form to the next. A Loop Controlled Fast Feed is never started until the first digit of the blockette currently being read is sensed. If this should be a Fast Feed symbol, it takes precedence over the channel 5 hole; i.e., the magnetic tape controlled Fast Feed is obeyed, the Loop Controlled Fast Feed is ignored.

A special punch with sprocket hole aligning keys is used in preparing the Paper Loop Control tape. To punch the start of a Fast Feed, punch a hole in channel 5 on the line corresponding to the first line to be skipped. To stop the Fast Feed, punch a hole on the line where the first line of printing is to occur after the Fast Feed. Channel 6 is used for stopping if the Fast Feed was started with a channel 5 hole. Channel 1, 2, 3, or 4 is used for a tape-started Fast Feed.

The longest loop the High-Speed Printer can handle is 22 inches; the shortest is 11 inches. Of course, the punching for one form can be repeated several times, if desired, (and if the form is eleven inches long or less). Indeed, this is preferable since greater loop life will result.

There are two restrictions that must be observed in the use of Fast Feed:

1. A Fast Feed, whether loop or tape started, must always cause at least one line to be skipped.

2. A channel 5 hole should never be punched on the same line with any other hole.

During a Fast Feed operation, the paper is moved at a rate of 20"/second, or, in other words, 7200 lines per minute.

7. Multiline Printing

The Multiline symbol Ꞧ (0 01 0000) is used to put the High-Speed Printer into the Multiline mode of operation. It should be placed in the first digit position of the blockette to be multilined, unless there is a Fast Feed symbol required too, in which case the Fast Feed symbol is placed first, the Multiline symbol second.

The printer returns to normal operation at the end of each multilined blockette so that even if consecutive blockettes are to be done in Multiline, a new Multiline symbol must be placed in each. The Multiline symbol will not be printed.

Multiline operation is entirely a function of the plugboard and the plugging operations may be broken into three groups:

1. Selecting the number of lines in which a blockette is to be printed.

2. Selecting the memory positions which are to be printed on each line.

3. Selecting the printing location on each line for each character.

<u>Selection of the number of lines.</u> This plugging consists of four steps and only the upper right hand section of the plugboard is under concern.

1. Run a jumper from the hole marked "Home" (33F) to the hole in column 32, counting down from the top, the number of rows corresponding to the number of lines into which the blockettes are to be divided in Multiline.

2. For rows above the hole selected in (1), run jumpers between corresponding holes in columns 32 and 33.

3. For rows including and below the hole selected in (1), run jumpers between corresponding holes in columns 33 and 34, down to and including row E.

4. If the number of lines selected is 2 to 6, connect holes 32S and 32T (marked "One Line") with a jumper; if the number of lines selected is 1, omit the jumper between these two holes.

The following examples will illustrate the required plugging:

Example 1: Four line printing
1. Connect 33F to 32D
2. Connect 32A to 33A; 32B to 33B;
   32C to 33C
3. Connect 33D to 34D; 33E to 34E
4. Connect 32S to 32T

Example 2: One line printing
1. Connect 33F to 32A
2. No plugging required
3. Connect 33A to 34A; 33B to 34B;
   33C to 34C; 33D to 34D
   33E to 34E
4. No plugging required

Example 3: Six line printing

1. Connect 33F to 32F
2. Connect 32A to 33A; 32B to 33B
   32C to 33C; 32D to 33D
   32E to 33E
3. No plugging required
4. Connect 32S to 32T

<u>Selection of the memory locations to be printed on each line.</u> In Multiline work there is a set of 15 relays brought into play, different ones of which can be energized on different lines under the control of the plugboard. Further, by plugging the memory locations to the contacts of these relays, the characters to be printed out on each line can be selected simply by the relays which have been chosen to be energized for each line.

Each relay has 12 contacts and thus can handle 12 digits. The first step, then, is to decide how many relays are required to print each line and to plug them accordingly. To avoid confusion, relays should be chosen sequentially. Thus, if 40 digits are to be printed on the first line, 48 digits on the second line and 6 digits on the third line, relays 1, 2, 3, and 4 (room for a total of 48 digits) should be plugged to operate on the first line; 5, 6, 7, and 8 should be plugged for the second line, and relay 9 for the third line.

This is accomplished as follows (see upper right hand portion of the plugboard): Holes 32G, H, I, J, K, and L (labeled "ML Counter, 1, 2, 3, 4, 5, 6", respectively) give output signals in turn when the printer is on line 1, 2, 3, etc., of a Multiline operation. Holes 33G and 34F (internally connected together), 33H and 34G, 33I and 34H, . . . , 33U and 34T are pairs of holes going to the 15 relays coils in order. They are labeled 1 to 15 on the plugboard. To operate relay 1 on line 1, run a jumper between holes 32G and 34F. Then to also pick up relays 2, 3, and 4 on line 1, as required in the example, run jumpers between 33G and 34G, 33H and 34H, and 33I and 34I. This, of course, could be extended if more relays were required to be picked up on line 1. (As many as 10 relays may be used on any given line. This gives sufficient contacts--120, to handle the entire memory.)

For line 2, again assuming four relays are to be picked up, plug 32H (line 2 output) to 34J (relay 5 coil). Then plug 33K to 34K, 33L to 34L and 33M to 34M.

On line 3, only one relay is assumed required. Pick this up by connecting 32I to 34N.

One important rule should now be noted: If, on any given line in a Multiline operation, zero, one, or two relays are used, then a resistor must be plugged in on that line. This is easily done by running a jumper--

a. From the unused hole in the pair corresponding to the second relay in the chain (if 2 relays are used), or
b. From the unused hole in the pair corresponding to the single relay (if only one relay is used), or
c. From the line output hole itself (holes 32G to L) if no relays are used (i.e., nothing is to print on that particular line) to one of the six holes in the section marked "Resistors" (holes 32M to R).

Thus, in the example above, since only one relay is used on line 3, a jumper should be run from hole 330 to 32M. Note carefully that this plugging of resistors applies only to the lines within the group size selected for the Multiline operation. Thus, in the above example, where a 3 line Multiline was assumed, nothing at all need be plugged to the last three output lines (32J, K, and L) since they will never be excited. This should not be confused with the situation where, say, in a 4 line Multiline it is not desired to print anything on line 2. Here, no relay need be connected to line 2, but a resistor should be.

Having wired up the relay coils, the next step is to send the memory locations (actually, comparator outputs) to the proper relay contacts. The 180 relay contacts (12X15) go to 180 pairs of holes which are located in columns 17 through 31 in the upper plugboard panel. One column represents the contacts for one relay.

Now, unless the run is exclusively Multiline, the plugboard will have already been wired for Single Line Printing, as described in Section 3. That is, the holes in columns 1 through 5 (labeled "From Comparator") will be wired to the paired holes in columns 6 through 16 (labeled "To Single Lines Relays"). Thus, each memory location can, in general, be picked up from the unused hole in the pair here. If certain memory locations do not print in Single Line work (and are to print in Multiline work), or if there is to be no Single Line Printing at all, then the memory locations must be picked up from the first five columns directly. It is then simply a question of taking those memory locations which are to print on the first line and plugging them sequentially to the relays which have been assigned to the first line, and similarly for the other lines. Remember that the holes corresponding to the relay contacts are paired.

As an example, suppose the first 40 memory locations are to print in Single Line Printing in positions 1-40. Then hole 1A would have been plugged to 6A, 1B to 6C, 1C to 6E, . . ., 2P to 9G. Now suppose these same 40 memory locations are also to print on line 1 of a Multiline operation. The first 4 relays (columns 17, 18, 19, and 20) would have been allocated for this purpose. Then the plugging would be: 6B to 17A, 6D to 17C, 6F to 17E, . . ., 9H to 20G. Notice that many contacts on the fourth relay are unused. Information to print on line 2 would be plugged starting with the fifth relay (column 21).

Selection of the printing location for each character. The other sides of the Multiline relay contacts go to a set of 180 pairs of holes on the lower plugboard panel (columns 7 through 21). Again, each column corresponds to a relay. Thus, column 17 on the upper panel is the "input" side of relay 1, while column 7 on the lower panel is the "output" side of the same relay. The individual holes in the columns are likewise in one-to-one correspondence. Thus, a comparator signal entering the first (or second) hole in column 17 of the top panel will appear at the first (and second) hole in column 7 of the bottom panel when relay 1 is energized (i.e., on line one).

The first five columns and the first ten holes of the sixth column on the lower panel (making 130 holes in all) are connected one-to-one to the 130 thyratrons which drive the printing hammers. The procedure now is to connect the line 1 relay outputs (however many relays may be used for line 1) to the thyratron holes in the manner desired. Now on line 2, some of the characters may need to be printed in, as yet, unused positions. These are plugged directly to the corresponding thyratrons. Other characters (or more properly, memory locations) may need to be printed in positions already printed on line 1. To do this, plug from these particular holes in the line 2 relay output set to those unused holes of the pairs of the line 1 relay output, where the other hole of the pair is already plugged to the desired thyratron.

A similar procedure is followed for line 3 and the following lines. There will always be a free hole in a relay output pair into which a jumper from an output on a later line can be plugged to achieve printing in the same position as the earlier line.

As an example of the above plugging, consider the following exercise:

> Print the first three memory locations in the first three print positions on line 1; the next four memory locations in the first four positions on line 2; and the next 5 memory locations in the first five positions of line 3.

It is assumed that the first Multiline relay has been assigned for line 1, the second for line 2, and the third for line 3. Further, that the comparator outputs have been picked up and are plugged into, in order, 17A, 17C, 17E, 18A, 18C, 18E, 18G, 19A, 19C, 19E, 19G, and 19I (all on the lower panel). The required plugging then is (all on the lower panel):

```
7A to 1A, 7C to 1B, 7E to 1C                      (line 1)
8A to 7B, 8C to 7D, 8E to 7F, 8G to 1D            (line 2)
9A to 8B, 9C to 8D, 9E to 8F, 9G to 8H, 9I to 1E  (line 3)
```

8. Multiple Printing

Plugging procedure for Single Line operation. By means of the paired holes in the section of the upper panel marked "To Single Line Relays", a memory location can be readily printed out as many as three times on a line.

to do this, simply plug from the unused hole in the first position chosen to one of the paired holes (preferably the top) in the second place where printing is desired. If triplicating is required, run a jumper from the unused hole of the second position to the third position.

Example: Print the contents of the first memory location in positions 1, 61, and 121.

```
Plug  1A to  6A
      6B to 11A
     11B to 16A
```

If some of the memory positions for Single Line Printing are common to Multiline Printing also, it is desirable to do the Multiple plugging for Single Line first, then the last unused hole in a pair may be connected to the Multiline relays.

Plugging procedure for Multiline operation. In Multiline work, a character cannot only be printed out 3 times on a given line, but then can be printed out up to 3 times again on any other line in the group. This is done by "chain plugging", as noted above, in the "To Multiline Relays" section of the plugboard. By connecting the output to contacts on relays (or a single relay), which close on the same line, the digit may be printed up to 3 times on that same line. Then by extending the chain of plugging to relays which close on other lines, the same character may be repeated on those lines. On the lower panel, the positioning of the characters on the line is taken care of as in regular Multiline work (discussed in Section 7).

Example: Print out memory location 1 in positions 1 and 61 on Single Line operation. On Multiline operation print out memory location 1 in positions 1, 25, and 49 on line 1, and in positions 25 and 49 on line 2.

Assume relay 1 is energized on line 1, and relay 2 on line 2. Then the plugging is, by panel:

On the upper panel, plug	1A	to	6A
	6B	to	11A
	11B	to	17A
	17B	to	17C
	17D	to	17E
	17F	to	18A
	18B	to	18C
On the lower panel, plug	7A	to	1A
	7C	to	2A
	7E	to	3A
	8A	to	7D
	8C	to	7F

Note that by this method it is possible to have, at most, 180 print-outs from one blockette in Multiline, since there are only 180 relay contacts. However, there is complete flexibility in the positioning of the information on each line and, of course, complete independence between Single Line and Multiline Printing.

There is another system for doing multiple printing on Multiline which allows the entire memory to be triplicated, if desired (i. e., produce as many as 360 print-outs from one blockette). However, in using this method there must be no Single Line work at all or, if Single Line operation does occur, one must accept the same duplicating or triplicating pattern as is plugged for the Multiline work.

This system is particularly useful where each line of the Multiline group is printed directly under the line above and the group as a whole is duplicated or triplicated across the page. This system is used as follows:

1. Plug for straight Multiline work as in Section 7, ignoring the Multiple Printing problem (i.e., simply plugging for one printing).

2. On the lower panel, in the "From Multiline Relays" section, outputs going to the same print position on different lines will have been chain plugged together and run to one set of thyratron holes. However, each of the outputs for the last line will have a free hole. If only duplicating of the group is required, run straight jumpers from these free holes to the appropriate thyratron holes. If

triplicating is required, "Y" jumpers are used to connect each free output hole to two thyratrons.

Example: Print the first memory location on line 1, the second memory location on line 2 and the third on line 3. On all three lines the printing is to be in triplicate, printing in columns 1, 49, and 97. (Use the first Multiline relay on line 1, the second on line 2 and the third on line 3.)

On the upper panel, plug	1A	to	17A
	1B	to	18A
	1C	to	19A
On the lower panel, plug	7A	to	1A
	8A	to	7B
	9A	to	8B
	9B	to	3A and 5A ("Y" jumper)

Note that if the printer should go into Single Line Printing and <u>if</u> something were plugged to print (on Single Line) in <u>either</u> positions 1, 49, or 97, it would print in <u>all</u> <u>three</u> positions. This might be all right but, if independently some other memory location should be plugged to another of these three points, there would be trouble. (The printer would stop with a Print Check Error.) Thus, great care must be exercised in using this method of Multiple Printing if Single Line operation can also occur in the run.

9. <u>Checking Features</u>

The error detection circuits on the High-Speed Printer may be considered in the three sections through which information recorded on tape is printed: 1- reading of data from tape, 2- storage of data in the memory, and 3- printing of the memory.

<u>Error detection in reading tape</u>. Each digit read from tape is given a binary bit count. If the number of binary ones present is odd the character has been read from tape correctly. If an even count is detected, the remainder of the blockette is read, but the Odd-Even Check Error is set and the printer stops. Nothing is printed.

Further, each blockette read from tape must contain exactly 120 digits. If a blockette is longer or shorter than 120 digits, the printer will set the 120 Check Error and stop. Nothing will be printed.

<u>Error detection in storage</u>. Each of the digits coming from the tape is placed in the proper position in the memory by virtue of the set of 120 address lines, which are sequentially excited by the function table which decodes the Main (or Address) Counter. If an address line should fail to rise to the signal level when it is supposed to, the corresponding digit coming from tape could not enter the memory. The Address Check circuits look for this and, if an address line should fail to be excited at the proper time, the Address Check Error is set and the printer stopped.

<u>Error detection in printing</u>. The fundamental checking of the printing is done by means of the group of 130 check thyratrons working in conjunction with the All-Out Detector. At the end of each print cycle, all 130 check thyratrons should be extinguished and the All-Out Detector looks for this. If some column fails to print when it should or prints an incorrect character, its check thyratron will be on when probed. This causes the machine to stop with the Print Check Error set.

The determination of whether all check thyratrons have been extinguished or not is made by a circuit called the All-Out Detector. If this circuit should fail in a certain way, a steady All-Out signal would be given and printing error would fail to stop the machine. To guard against this, the All-Out Detector is itself checked every line and, if not functioning properly, the printer will stop with the All-Out Detector Error set.

In addition to the major checking circuits described above, there are other checks applied to the control circuits and paper feed mechanism to ensure complete accuracy of printing. The description of these checks requires a detailed understanding of the printer logic.

10. Example Problem

As an example of simple plugging for the High-Speed Printer, suppose a series of paychecks are to be printed. The desired check format is shown in Figure 4. The blockette layout is shown in Figure 5. The simple nature of this example permits us to do Multiline Printing only, with one blockette per check. An explanation of the net pay field in the blockette is called for. It is obviously desirable that the amount of the check be printed in as unalterable a manner as possible. This is easily done if, say, twenty dollars and fifty cents is printed as $20.50 and not as (assuming pay up to $999.99 is permitted) $ 20.50 which would be the result from simple zero suppression. Thus, the net pay field is assumed to be pre-edited by the UNIVAC when it is inserted in the blockette. Thus, the above example would appear as _$2050 in the blockette.

A. Selection of the Number of Lines (6line Multiline):

Plug (Upper Panel)		
32F	to	33F
32A	to	33A
32B	to	33B
32C	to	33C
32D	to	33D
32E	to	33E
32S	to	32T

B. Selection of the Multiline Relays:

Line	Relays	Plug (Upper Panel)		
1	1,2,3	32G	to	34F
		33G	to	34G
		33H	to	34H
2	none	32H	to	32M
3	4,5	32I	to	34I
		33J	to	34J
		33K	to	32N

4	6,7,8,9	32J	to	34K
		33L	to	34L
		33M	to	34M
		33N	to	34N
5	10,11	32K	to	34O
		33P	to	34P
		33Q	to	32O
6	12,13,14	32L	to	34Q
		33R	to	34R
		33S	to	34S

C. Comparator Outputs (Memory Locations) to Relay Contacts:

Line	Field	Plug (Upper Panel)				
1	check number	1B	to	17A		
				17B	to	18S
		1C	to	17C		
				17D	to	18U
		1D	to	17E		
				17F	to	18W
		1E	to	17G		
				17H	to	19A
		1F	to	17I		
				17J	to	19C
		1G	to	17K		
				17L	to	19E
	net pay	2B	to	17M		
		2C	to	17O		
		2D	to	17Q		
		2E	to	17S		
		4D	to	17U (decimal pt)		
		2F	to	17W		
		2G	to	18A		
	date	1N	to	18C		
				18D	to	19G
		1O	to	18E		
				18F	to	19I
		3B	to	18G (dash)		
				18H	to	19K
		1P	to	18I		
				18J	to	19M
		1Q	to	18K		
				18L	to	19O

		4B	to	18M	(dash)	
				18N	to	19Q
		1R	to	18O		
				18P	to	19S
		1S	to	18Q		
				18R	to	19U
3	regular pay	2O	to	20A		
		2P	to	20C		
		2Q	to	20E		
		17V	to	20G	(decimal pt)	
		2R	to	20I		
		2S	to	20K		
	withholding tax	3H	to	20M		
		3I	to	20O		
		3J	to	20Q		
		20H	to	20S	(decimal pt)	
		3K	to	20U		
		3L	to	20W		
	bond deduction	3U	to	21A		
		3V	to	21C		
		20T	to	21E	(decimal pt)	
		3W	to	21G		
		3X	to	21I		
	overtime pay	2T	to	22A		
		2U	to	22C		
		2V	to	22E		
		21F	to	22G	(decimal pt)	
		2W	to	22I		
		2X	to	22K		
	FICA tax	3M	to	22M		
		3N	to	22O		
		22H	to	22Q	(decimal pt)	
		3O	to	22S		
		3P	to	22U		
	insurance	4I	to	22W		
		4J	to	23A		
		22R	to	23C	(decimal pt)	
		4K	to	23E		
		4L	to	23G		
	name	5A	to	23I		
		5B	to	23K		
		5C	to	23M		
		5D	to	23O		
		5E	to	23Q		
		5F	to	23S		
		5G	to	23U		

		5H	to	23W	
		5I	to	24A	
		5J	to	24C	
		5K	to	24E	
		5L	to	24G	
		5M	to	24I	
		5N	to	24K	
		5O	to	24M	
		5P	to	24O	
		5Q	to	24Q	
		5R	to	24S	
		5S	to	24U	
		5T	to	24W	
		5U	to	25A	
		5V	to	25C	
		5W	to	25E	
		5X	to	25G	
	net pay	17N	to	25I	
		17P	to	25K	
		17R	to	25M	
		17T	to	25O	
		4C	to	25Q	(decimal pt)
		17X	to	25S	
		18B	to	25U	
5	medical pay	3C	to	26A	
		3D	to	26C	
		3E	to	26E	
		25R	to	26G	(decimal pt)
		3F	to	26I	
		3G	to	26K	
	union dues	3E	to	26M	
		3F	to	26O	
		26H	to	26Q	(decimal pt)
		3G	to	26S	
		3H	to	26U	
	other ded.	3Q	to	26W	
		3R	to	27A	
		26R	to	27C	(decimal pt)
		3S	to	27E	
		3T	to	27G	
6	first adjustment	1H	to	28A	
		1I	to	28C	
		1J	to	28E	
		27D	to	28G	(decimal pt)
		1K	to	28I	
		1L	to	28K	
	second adjustment	1T	to	28M	

	1U	to	28O	
	1V	to	28Q	
	28H	to	28S	(decimal pt)
	1W	to	28U	
	1X	to	28W	
third adjustment	2H	to	29A	
	2I	to	29C	
	2J	to	29E	
	28T	to	29G	
	2K	to	29I	
	2L	to	29K	
badge number	4M	to	29M	
	4N	to	29O	
	4O	to	29Q	
	4P	to	29S	
	4Q	to	29U	
	4R	to	29W	
	4S	to	30A	
	4T	to	30C	

4. From Relay Contacts to Print Positions

Line	Field	Plug (Lower Panel)					
1	check number	7A	to	1K,	8S	to	3S
		7C	to	1L,	8U	to	3T
		7E	to	1M,	8W	to	3U
		7G	to	1N,	9A	to	3V
		7I	to	1O,	9C	to	3W
		7K	to	1P,	9E	to	3X
	net pay	7M	to	2D			
		7O	to	2E			
		7Q	to	2F			
		7S	to	2G			
		7U	to	2H			
		7W	to	2I			
		8A	to	2J			
	date	8C	to	2S,	9G	to	5X
		8E	to	2T,	9I	to	6A
		8G	to	2U,	9K	to	6B
		8I	to	2V,	9M	to	6C
		8K	to	2W,	9O	to	6D
		8M	to	2X,	9Q	to	6E
		8O	to	3A,	9S	to	6F
		8Q	to	3B,	9U	to	6G
3	regular pay	10A	to	7B			
		10C	to	7D			

		10E	to	7F
		10G	to	7H
		10I	to	7J
		10K	to	7L
	withholding tax	10M	to	7X
		10O	to	8B
		10Q	to	2K
		10S	to	2L
		10U	to	2M
		10W	to	2N
	bond deduction	11A	to	8N
		11C	to	8P
		11E	to	8R
		11G	to	3C
		11I	to	3D
4	overtime pay	12A	to	10B
		12C	to	10D
		12E	to	10F
		12G	to	10H
		12I	to	10J
		12K	to	10L
	FICA tax	12M	to	10P
		12O	to	10R
		12Q	to	10T
		12S	to	10V
		12U	to	10X
	insurance	12W	to	11B
		13A	to	11D
		13C	to	11F
		13E	to	11H
		13G	to	11J
	name	13I	to	4E
		13K	to	4F
		13M	to	4G
		13O	to	4H
		13Q	to	4I
		13S	to	4J
		13U	to	4K
		13W	to	4L
		14A	to	4M
		14C	to	4N
		14E	to	4O
		14G	to	4P
		14I	to	4Q
		14K	to	4R
		14M	to	4S
		14O	to	4T

		14Q	to	4U
		14S	to	4V
		14U	to	4W
		14W	to	4X
		15A	to	5A
		15C	to	5B
		15E	to	5C
		15G	to	5D
	net pay	15I	to	5W
		15K	to	9H
		15M	to	9J
		15O	to	9L
		15Q	to	9N
		15S	to	9P
		15U	to	9R
5	medical pay	16A	to	12B
		16C	to	12D
		16E	to	12F
		16G	to	12H
		16I	to	12J
		16K	to	12L
	union dues	16M	to	12N
		16O	to	12P
		16Q	to	12R
		16S	to	12T
		16U	to	12V
	other deductions	16W	to	12X
		17A	to	13B
		17C	to	13D
		17E	to	13F
		17G	to	13H
6	first adjustment	18A	to	12K
		18C	to	1Q
		18E	to	1R
		18G	to	1S
		18I	to	1T
		18K	to	1U
	second adjustment	18M	to	2A
		18O	to	2C
		18Q	to	7N
		18S	to	7P
		18U	to	7R
		18W	to	7S
	third adjustment	19A	to	16P
		19C	to	16T

	19E	to	16V
	19G	to	20
	19I	to	2P
	19K	to	2Q
badge number	19M	to	8T
	190	to	8V
	19Q	to	8X
	19S	to	9B
	19U	to	9D
	19W	to	9F
	20A	to	4A
	20C	to	4B

5. Zero Suppression

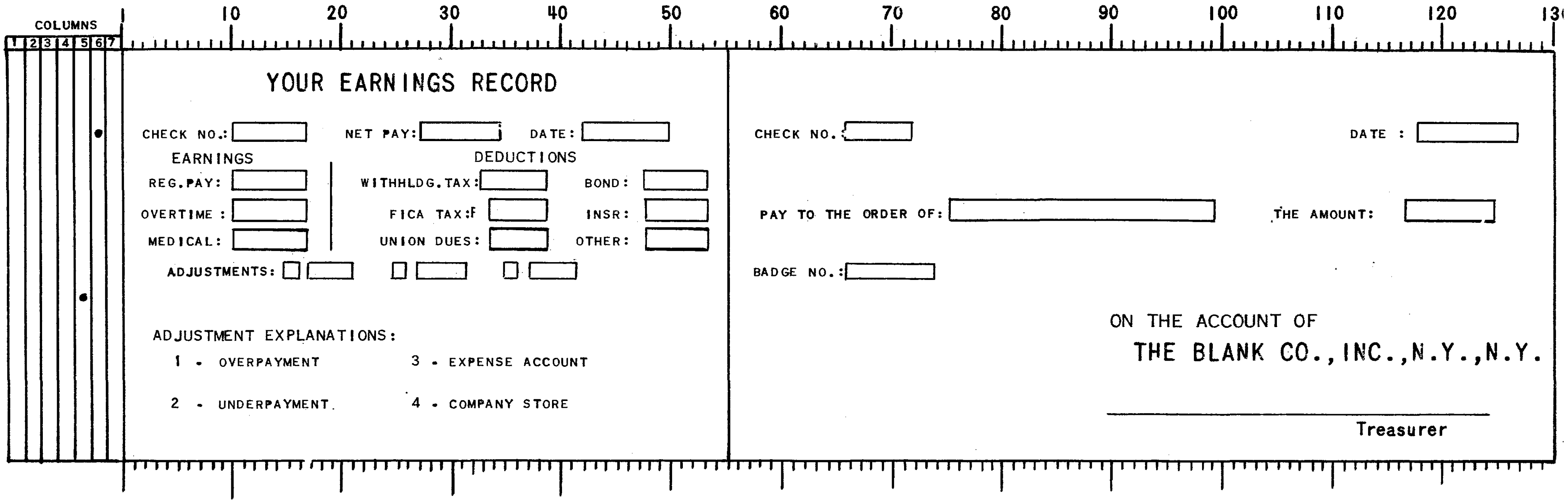
Plug (Lower Panel)

Start Zero Fields						End Zero Fields					
26I	to	25R,	23R	to	24R	26L	to	32R,	33R	to	34R
27A	to	25Q,	23Q	to	24Q	27D	to	32Q,	33Q	to	34Q
27M	to	25P,	23P	to	24P	27P	to	32P,	33P	to	34P
27S	to	250,	230	to	240	28C	to	320,	330	to	340
28D	to	25N,	23N	to	24N	28H	to	32N,	33N	to	34N
28K	to	25M,	23M	to	34M	280	to	32M,	33M	to	34M
28P	to	25L,	23L	to	34L	28T	to	32L,	33L	to	34L
29A	to	25K,	23K	to	34K	29D	to	32K,	33K	to	34K
29E	to	25J,	23J	to	34J	29H	to	32J,	33J	to	34J
29I	to	25I,	23I	to	34I	29L	to	32I,	33I	to	34I
29Q	to	25H,	23H	to	34H	29T	to	32H,	33H	to	34I
30A	to	25G,	23S	to	34S	30D	to	32G,	33S	to	34S

6. Line Spacing

Plug (Upper Panel)
16X to 16U

Figure 4



YOUR EARNINGS RECORD

CHECK NO.: NET PAY: DATE:

EARNINGS

REG.PAY:

OVERTIME :

MEDICAL:

DEDUCTIONS

WITHHLDG.TAX: BOND:

FICA TAX:F INSR:

UNION DUES: OTHER:

ADJUSTMENTS:

ADJUSTMENT EXPLANATIONS:

1 - OVERPAYMENT 3 - EXPENSE ACCOUNT

2 - UNDERPAYMENT 4 - COMPANY STORE

CHECK NO.: DATE :

PAY TO THE ORDER OF: THE AMOUNT:

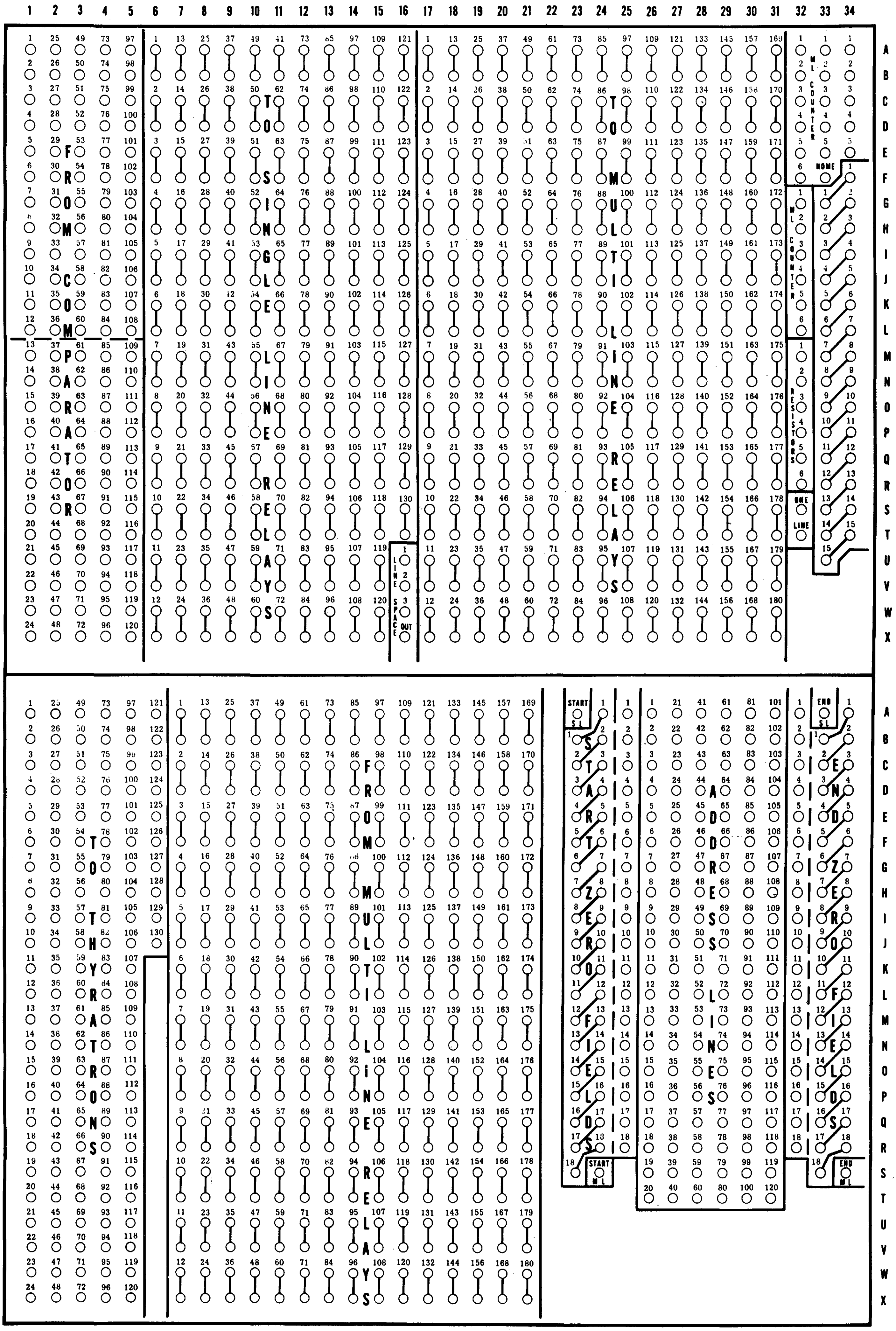
BADGE NO.:

ON THE ACCOUNT OF

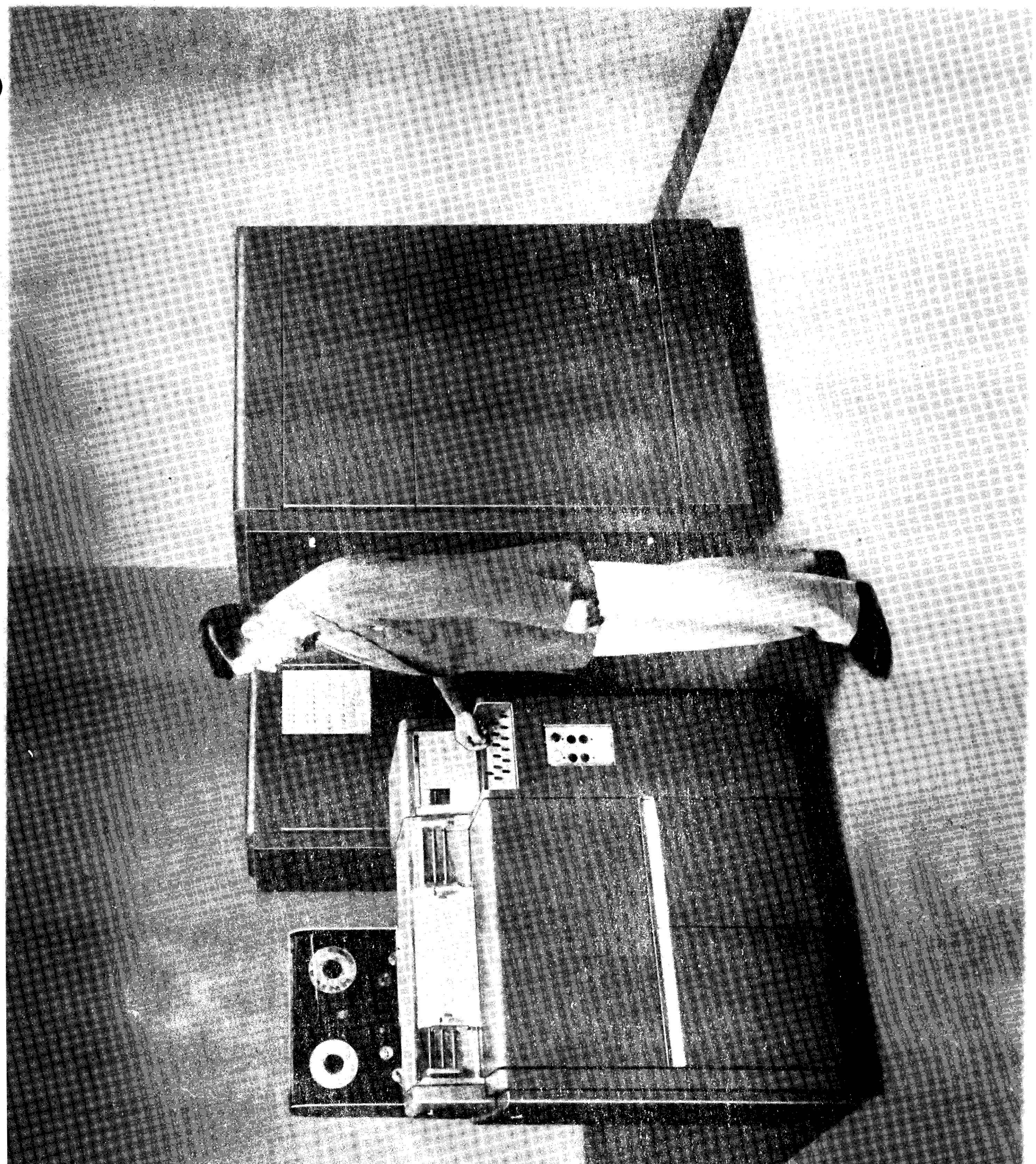
THE BLANK CO., INC., N.Y., N.Y.

Treasurer

PRINT WHEEL POSITIONS



FROM COMPARATORS
TO SINGLE LINE RELAYS
TO MULTI LINE RELAYS
ML COUNTER
HOME
ML COUNTER
RESISTORS
ONE LINE
LINE SPACE
OUT
TO THYRATRONS
FROM MULTI LINE RELAYS
START SL
START ZERO FIELDS
START ML
ADDRESS LINES
END SL
END ZERO FIELDS
END ML



	00	01	10	11
0000				
0001		,		
0010	-	.		:
0011	0	;	)	+
0100	1	A	J	/
0101	2	B	K	S
0110	3	C	L	T
0111	4	D	M	U
1000	5	E	N	V
1001	6	F	O	W
1010	7	G	P	X
1011	8	H	Q	Y
1100	9	I	R	Z
1101	'	#	$	%
1110	&		*	
1111	(			

PRINTING CHARACTERS AND THEIR PULSE CODES FOR THE HIGH-SPEED PRINTER

Figure 2

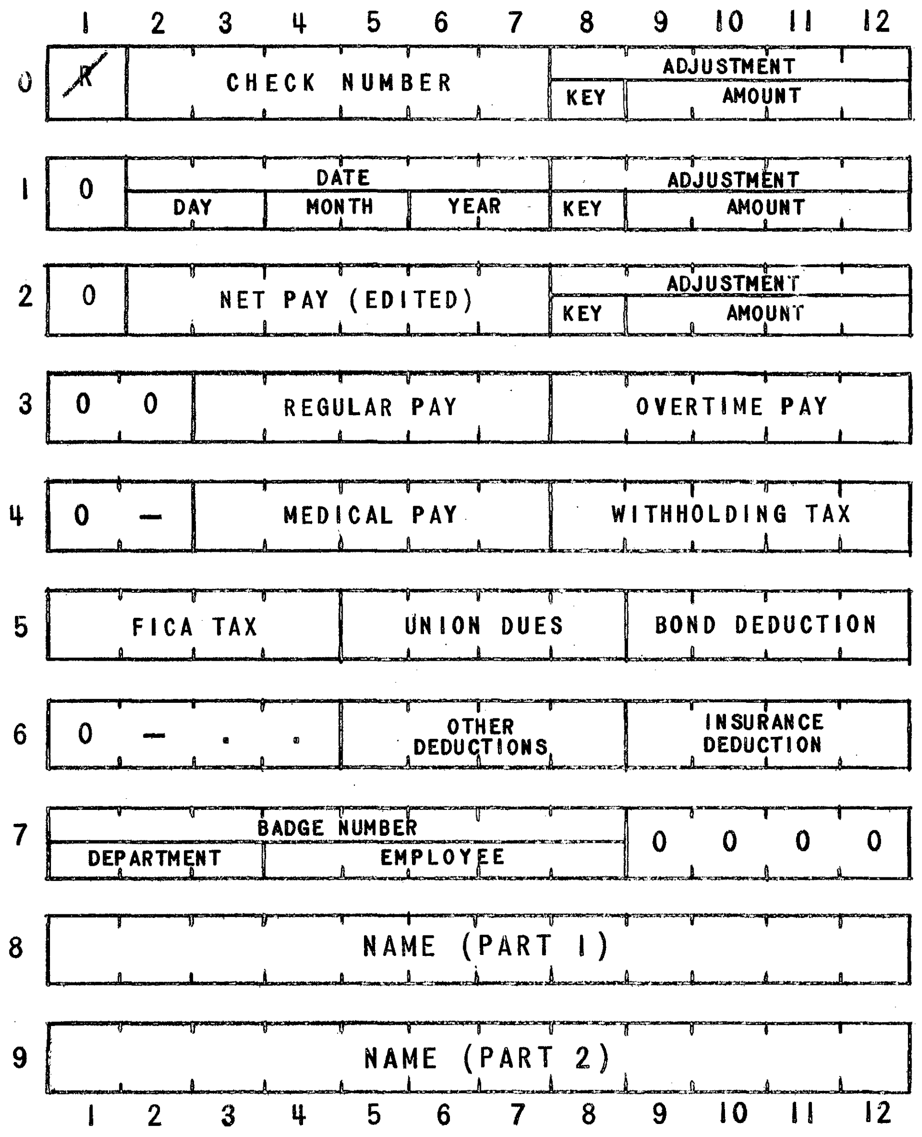

1 2 3 4 5 6 7 8 9 10 11 12
0
~~R~~
CHECK NUMBER
ADJUSTMENT
KEY
AMOUNT
1
0
DATE
DAY
MONTH
YEAR
ADJUSTMENT
KEY
AMOUNT
2
0
NET PAY (EDITED)
ADJUSTMENT
KEY
AMOUNT
3
0 0
REGULAR PAY
OVERTIME PAY
4
0 —
MEDICAL PAY
WITHHOLDING TAX
5
FICA TAX
UNION DUES
BOND DEDUCTION
6
0 — . .
OTHER DEDUCTIONS
INSURANCE DEDUCTION
7
BADGE NUMBER
DEPARTMENT
EMPLOYEE
0 0 0 0
8
NAME (PART 1)
9
NAME (PART 2)
1 2 3 4 5 6 7 8 9 10 11 12

FIGURE 5

CHANGES IN UNIPRINTER AND UNITYPER I TO FIT THE 63-CHARACTER CODE FOR PROCESSING OF DATA FOR HIGH SPEED PRINTER

1. Keyboard for Supervisory Control and Unityper I

The keyboard and the decoding unit have been changed to allow the typing onto magnetic tape of any of the 63 pulse code combinations. All indications of upper and lower case have been removed from the keyboard and also one of the duplicate numeric sections, namely that one over the alphabetic portion. More resistors have been added in the decoding unit to allow for the decoding of 63 characters where previously only 51 were decoded. No other changes in the Unityper have been made. The changes are illustrated by the accompanying figure and table.

2. Printer Dolly of Uniprinter

A new switch has been added on the inside which has two positions: "print" and "stall". This switch affects only the twelve new pulse code combinations.

In the "stall" position, when one of the new characters enters the printer, the printer stops. The operator determines the character by looking at the neon lights and prints manually any character he desires, after which the printer automatically proceeds.

If the switch is in the "print" position, then when one of the new characters is encountered, a ";" (semi-colon) is printed in lower case operation or a ":" (colon) in upper case operation.

One additional change is that the tab operation is indicated by printing a "v" in the computer digit mode of operation.

3. Typing-In of Information for the Uniprinter

The Uniprinter operates in the same fashion as before. The programmer and operator must remember that to make the printer "shift lock", he must depress the "$" key; to perform the "unshift" operation, he must depress the "?" key; and to perform a "single shift" operation, he must depress the "%" key.

<u>NEW UNIVAC PULSE CODE</u>

PULSE CODE	MODIFIED UNIPRINTER I AND S.C. KEYBOARD	MODIFIED UNIPRINTER			
		Normal		Comp. Digit	
		L Case	U Case		
1 00 0000	i	ignore		x	
0 00 0001	Δ	space		space	
0 00 0010	-	-	_	-	
1 00 0011	0	0	)	0	
0 00 0100	1	1	½	1	
1 00 0101	2	2	"	2	
1 00 0110	3	3	#	3	
0 00 0111	4	4	$	4	
0 00 1000	5	5	%	5	
1 00 1001	6	6	*	6	
1 00 1010	7	7	&	7	
0 00 1011	8	8	'	8	
1 00 1100	9	9	(	9	
0 00 1101	'	;	:	;	*
0 00 1110	&	;	:	;	*
1 00 1111	(	;	:	;	*

Table 1 - Part 1

PULSE CODE	MODIFIED UNITYPER I AND S.C. KEYBOARD	MODIFIED UNIPRINTER			
		Normal		Comp. Digit	
		L Case	U Case		
0 01 0000	r	Car.	Ret.	/	
1 01 0001	,	,	,	,	
1 01 0010	.	.	.	.	
0 01 0011	;	;	:	;	
1 01 0100	A	a	A	A	
0 01 0101	B	b	B	B	
0 01 0110	C	c	C	C	
1 01 0111	D	d	D	D	
1 01 1000	E	e	E	E	
0 01 1001	F	f	F	F	
0 01 1010	G	g	G	G	
1 01 1011	H	h	H	H	
0 01 1100	I	i	I	I	
1 01 1101	#	;	:	;	*
1 01 1110	¢	;	:	;	*
0 01 1111	@	;	:	;	*

Table 1 - Part 2

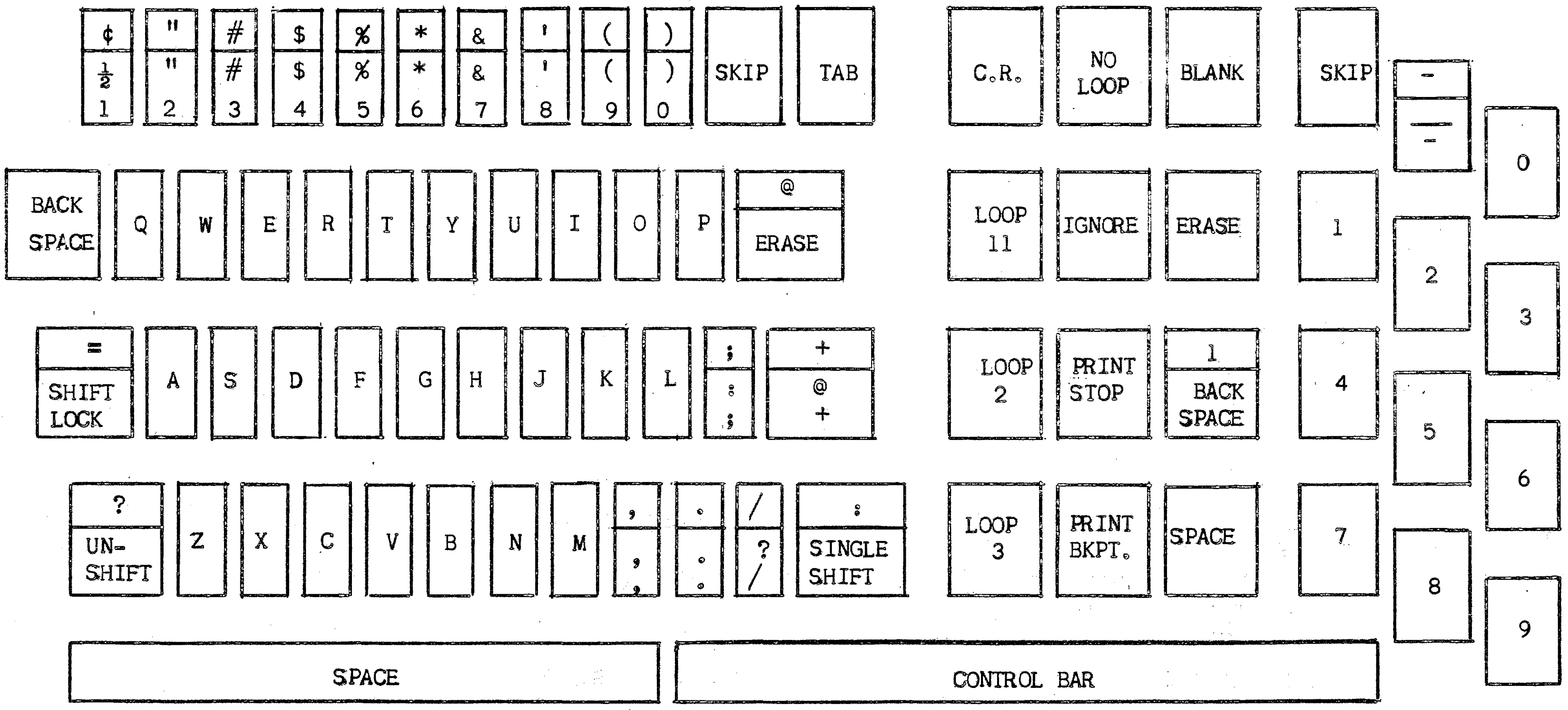
PULSE CODE	MODIFIED UNITYPER I AND S.C. KEYBOARD	MODIFIED UNIPRINTER			
		Normal		Comp. Digit	
		L Case	U Case		
0 10 0000	t	tab		V	
1 10 0001	"	;	:	;	*
1 10 0010	1	;	:	;	*
0 10 0011	)	;	:	;	*
1 10 0100	J	j	J	J	
0 10 0101	K	k	K	K	
0 10 0110	L	l	L	L	
1 10 0111	M	m	M	M	
1 10 1000	N	n	N	N	
0 10 1001	O	o	O	O	
0 10 1010	P	p	P	P	
1 10 1011	Q	q	Q	Q	
0 10 1100	R	r	R	R	
1 10 1101	$	shift lock		Z	
1 10 1110	*	;	:	;	*
0 10 1111	?	unshift		8	

Table 1 - Part 3

PULSE CODE	MODIFIED UNITYPER I AND S.C. KEYBOARD	MODIFIED UNIPRINTER			
		Normal		Comp. Digit	
		L Case	U Case		
1 11 0000	Σ	Prin.	Stop	Stop	
0 11 0001	β	Prin.	Bkpt.	Y	
0 11 0010	:	;	:	;	*
1 11 0011	+	+	@	+	
0 11 0100	/	/	?	/	
1 11 0101	S	s	S	S	
1 11 0110	T	t	T	T	
0 11 0111	U	u	U	U	
0 11 1000	V	v	V	V	
1 11 1001	W	w	W	W	
1 11 1010	X	x	X	X	
0 11 1011	Y	y	Y	Y	
1 11 1100	Z	z	Z	Z	
0 11 1101	%	single shift		-	
0 11 1110	=	;	:	;	*
1 11 1111					

Table 1 - Part 4

* With the switch on "print", these are the characters that will be printed; on "stall", the printer stops.



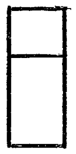


Fig. 1 - Changes in Keyboard of Unityper I and Supervisory Control