LOGICAL TECHNIQUES

OF

MODEL 102-A

This data, as compiled and edited by the Applied Electronics Training Department, represents the contributions of a number of division engineers over a several year period.

The National Cash Register Company
Electronics Division
3348 W. El Segundo Blvd.
Hawthorne, California

LOGICAL TECHNIQUES OF CRC 102-A

PHYSICAL DESCRIPTION OF CRC 102-A

THE GENERAL PURPOSE DIGITAL COMPUTER

Briefly, a general purpose digital computer performs numerical mathematical operations according to a series of commands (program). The computer can modify its own commands during a program, either in a preordained manner, or conditionally, according to the outcome of certain tests on intermediate results of computation.

The general purpose computer basically consists of input-output devices, control elements, storage (memory), and arithmetic elements.

THE CRC 102-A GENERAL PURPOSE COMPUTER

The mode of operation of the CRC 102-A is serial -- all information in the computer is stored on single channels, and access to the information is on a time-sequential basis. All information is synchronized by a single timing signal (clock) generated in the computer.

Input and output information may be conveyed to and from the computer proper by electric typewriter (Flexowriter), punched paper tape (Flexowriter tape), magnetic tape (CRC 126 tape drive mechanism), or punched cards (IBM).

MACHINE LANGUAGE

Number System. Internally, the computer is all binary (base 2). Input-output may be octal (base 8) or decimal (base 10), at the will of the programmer. (Computation using decimal input-output requires the use of conversion routines in the computer, which, once introduced, can be fully automatic).

In general, a number is represented in a positional notation by coefficients (digits) written from left to right and associated conventionally with decreasing powers of the base of the number system. A whole number of n digits, base b, can be expressed in the form

$$a_{n-1} \quad a_{n-2} \quad . \quad . \quad . \quad a_1 \quad a_0.$$

where the a's may have any value $0 \leq a_i \leq b-1$. The magnitude represented by this notation is

$$a_{n-1}\, b^{n-1} + a_{n-2}\, b^{n-2} + \ldots + a_1\, b + a_0\, b^0$$

The period (decimal point, octal point, etc.) is placed to the right of the coefficient of the zero power of the base. Coefficients to the right of the point are associated with negative powers of the base in decreasing order.

$$.a_{-1} \quad a_{-2} \quad a_{-3} \quad \ldots \quad a_{-n} = a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + a_{-3} \times b^{-3} + \ldots + a_{-n} \times b^{-n}$$

In the binary system, the base is two, and the digits are 0 and 1. This system is eminently suited to electrical or electronic representation, as any device with two stable states can be used to store a digit (on-off switch, vacuum tube saturated or cut-off, "flip-flop" circuit, magnetic surface magnetized to saturation in either direction, etc.). Also, binary arithmetic, although it follows the same rules as arithmetic to any base, may be represented in very simple tabular form.

Binary Addition

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10$

Binary Multiplication

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

Decimal Equivalents

$000_b = 0_d$

$001_b = 1_d$

$010_b = 2_d$

$011_b = 3_d$

$100_b = 4_d$

$101_b = 5_d$

$110_b = 6_d$

$111_b = 7_d$

$1000_b = 8_d$ etc.

In the octal system the base is 8 and the digits are 0, 1, 2, 3, 4, 5, 6, 7.

Octal Addition Table

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

$0 + n = n$

Octal Multiplication Table

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

$0 \times n = 0$

Decimal Equivalents

0 thru 7_o = 0 thru 7_d

$10_o = 8_d$

$11_o = 9_d$

$17_o = 15_d$

$20_o = 16_d$

$77_o = 63_d$

$100_o = 64_d$ etc.

Octal Addition (Decimal Check)

$$\begin{aligned} 367 &= 3.8^2 + 6.8 + 7.8^0 &&= 247_d \\ \underline{733} &= 7.8^2 + 3.8 + 3.8^0 &&= \underline{475_d} \\ 1322 &= 1.8^3 + 3.8^2 + 2.8 + 2.8^0 &&= 722 \end{aligned}$$

Octal Multiplication (Decimal Check)

$$\begin{aligned} 54 &= 44_d \\ \underline{26} &= \underline{22_d} \\ 410 & \quad 88 \\ \underline{130} & \quad \underline{88} \\ 1710 &= 968_d \end{aligned}$$

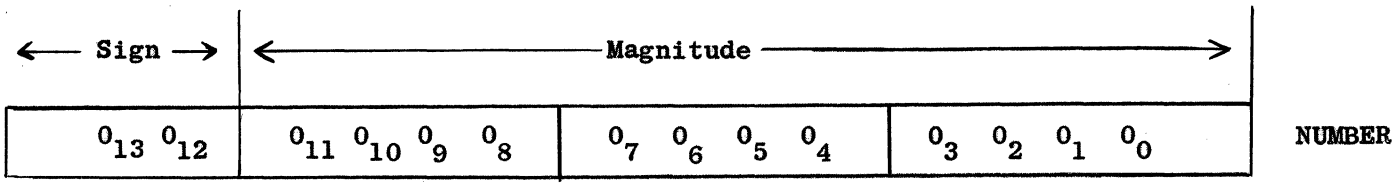
Conversion from the binary representation in the computer to the octal notation used in read-in, read-out, and programming merely requires setting off the binary number in groups of three binary digits, starting from the binary point, and interpreting each group of three binary digits as its octal equivalent. The inverse of this process is used to convert octal to binary.

Binary	Octal		Example	
000	0		101'110'100'111.101'000'101'101'	Binary
001	1			
010	2	=	5 6 4 7 . 5 0 5 5	Octal
011	3			
100	4			
101	5		03.1777 Octal	
110	6			
111	7	=	000011.001111111111 Binary	

Hereafter, binary digits will be referred to as "bits" and octal digits as "digits".

Words. In the CRC 102-A computer, a word consists of 14 octal digits (0_0 thru 0_{13}). Each digit contains three bits (P_0, P_1, P_2). Thus a word contains 42 bits, which are identified as $P_0 0_0$ thru $P_2 0_{13}$.

A word may be either a number or a command. If the word is a number, it consists of two sign digits and 12 magnitude digits.



The digits 0_0 thru 0_{11} represent the magnitude of the number. In this presentation, the octal point will be assumed to lie to the left of 0_{11}. Thus all numbers will be represented as octal fractions $-1 < n < 1$. The sign convention is as follows:

00 Positive

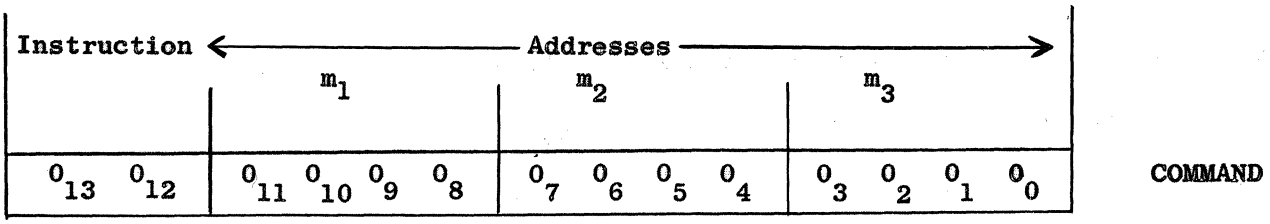
02 Negative

01 Positive with overflow

03 Negative with overflow

As all numbers are here represented smaller in magnitude than 1, any arithmetic operation giving a result ≥ 1 will cause an overflow indication. Unless certain precautions are observed, an overflow will automatically stop computation so that the operator can determine the cause of the overflow.

Commands in the CRC 102-A contain a two-digit number (the instruction) signifying the mathematical, logical, or input-output operation to be performed, and three four-digit numbers (m_1, m_2, m_3) which represent addresses (locations of information in the computer storage).



An example of a command is 35 0123 0126 0277 which would mean "look up the number in memory location 0123, look up the number in memory location 0126, add (35 is the "add" instruction) the latter to the former, and put the result away in memory location 0277." Parentheses used around an address as (m_1) are used to represent "contents of address m_1". It is important to make the distinction between an address m_1, and the information which is stored at that address, (m_1).

THE MEMORY

Magnetic Drum. Information is stored in the CRC 102-A as magnetic impulses on the surface of a rotating drum. The drum is 12 inches in diameter, 6 inches high, and rotates at a speed of 40 revolutions per second. The surface of the drum is coated with magnetic iron oxide, and information is recorded and played back with magnetic heads similar to those used in conventional tape recording. A pulse of current in one direction through the coil of a recording head magnetizes a small area of the drum surface to saturation in a corresponding direction, arbitrarily representing a binary "1". A pulse of current in the opposite direction through the recording head saturates the surface in the opposite direction, representing a binary "0".

Recording on the magnetic surface is of the "non-return-to-zero" form; that is,

the surface is magnetized to saturation in one direction or the other in every accessible bit-space on the drum at all times.

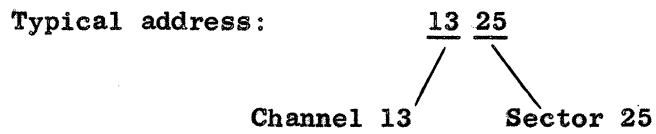
Once a series of bits has been recorded on the drum, it can be played back indefinitely without affecting the content of the number. However, recording new information in a space on the drum simultaneously removes the previous content of that space.

The operation of recording is also called "writing"; playback is referred to as "reading".

The drum is divided into a number of circular tracks or channels, each equipped with one or more reading, writing, or combination read-write heads.

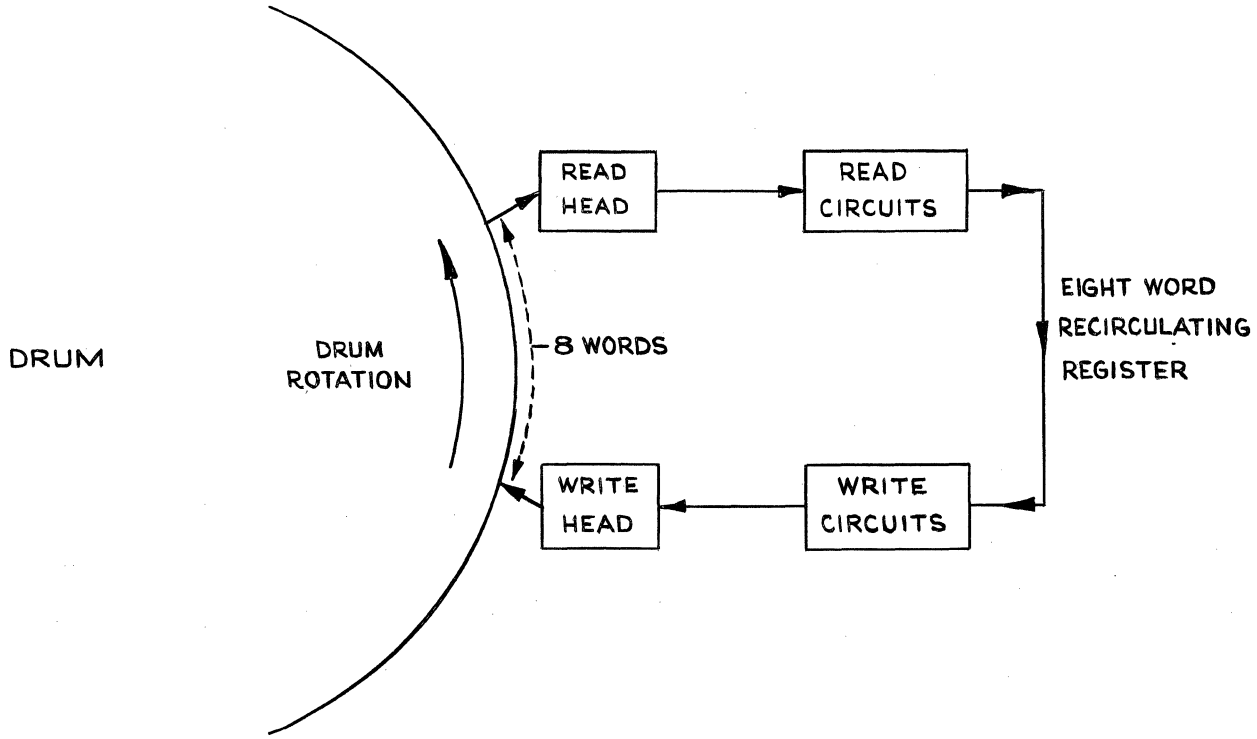
Main Memory-Addresses. Sixteen of these channels comprise the main memory, M. Each main memory channel contains 64 words of 42 bits each, thus the total main memory capacity is 64 X 16 = 1024 words. Each main memory channel has one combination read-write head. The spacing of channels is about $\frac{1}{4}$ inch on centers; channel width is about .2 inch. There are about 71 bits/inch on a channel.

The main memory channels are numbered octally from 00 to 17 for addressing purposes. Circumferential divisions of the drum are called sectors and are one word in length. The sectors are numbered octally from 00 to 77. The four octal digits of an address then identify a particular sector of a particular channel. The location on the drum corresponding to an address is called a "cell".



Addresses in the main memory run from 0000 (channel 00 sector 00) to 1777 (channel 17 sector 77) octally, representing 1024 (decimal) addresses in all.

Buffer Register. In addition to the main memory, there is an eight word recirculating line (J register), called the buffer. The buffer has a read head and a write head spaced eight words apart on the drum circumference. The read and write heads are placed so that information read off the drum by the read head is re-recorded on the drum by the write head eight word spaces away from the read head in the direction opposite to the direction of drum rotation. (see figure)



Thus information once recorded in the buffer continues to recirculate around the drum read-head write-head path until the line is broken for the insertion of new information. It is to be noted that the content of the J register (buffer) can be read out for computation purposes without affecting the recirculation. Addresses in the J register are numbered octally from 2000 to 2007.

One-Word Recirculating Registers. On another channel of the drum are located four pairs of read-write heads, forming four recirculating lines similar to the J register. Each of these registers is one word in length. They are the E, F, G, and H registers. The E and F registers are used for arithmetic purposes, to store operands and results of operations. The H register is called the control register. It is used to contain the instruction number, the addresses of operands, and the address of the next command. The G register is used sometimes for control purposes and sometimes for storing results of computation of operands.

During computation, operation of the E, F, G, and H registers is automatic. They are not available to the programmer for storage purposes.

Storage in the E, F, G, H, J recirculating registers is volatile. If the computer power is turned off, their content will be lost. Storage in the main memory is permanent; once recorded, information will remain on the drum until intentionally changed, even when the computer is turned off.

Timing Channels - Word Channel. The remaining three channels on the drum are permanently recorded, and are used for timing and sector identification:

The clock channel, C, has exactly 64 X 42 = 2688 clock pulses recorded in a closed loop around the drum. Each clock pulse corresponds to one bit-space in all channels of the drum. Output of the clock playback circuit is a square wave of 64 X 42 X 40 (rps) = 107,520 CPS frequency. All logical operations in the computer are synchronized to this clock wave form, thus minor variations in drum rotation speed ($\pm$ 5%) do not affect the process of computation.

The location of the sectors of the main memory and buffer channels is fixed by the word channel, M_w, which contains 64 words of the form

02 0000 0000 0000 to 02 0077 0077 0077

The first two digits of an address are used to select the correct channel, then the last two digits of the address are compared to the output of the word channel read circuits. When the word channel digits coincide with the sector address digits, the next word appearing in the selected channel is read out for computation purposes, or the result of a computation is read into the memory channel selected.

If a buffer cell address is called for, only the least significant digit of one of the word channel addresses is compared with the least significant digit of the buffer address.

Numbering of the word channel may be in any order around the drum, as long as each consecutive group of eight word-channel words contains the digits 0 through 7 in the same order in the least significant position of each of the three sector addresses, to permit proper access to the buffer when required.

The pulse appearing in $P_1\ 0_{12}$ of each sector is required to synchronize the pulse and digit counter when the clear proposition (Z) is true. During this time the counter counts when either $P_1\ 0_{12}'$ or M_w is true. Thus, during sector 02 0000 0000 0000 the

counter will reach the configuration $P_2\ 0_{12}$ and stop counting until the pulse in M_w appears. From then on each time the configuration $P_1\ 0_{12}$ is true, ($P_1\ 0_{12}'$ false) M_w will be true to keep the counter counting.

One additional pulse is generated for test purposes. This is derived from a bit in $P_1\ 0_{13}$ of one of the sectors. The product of M_w and $P_1\ 0_{13}$ gives a timing proposition appearing once per drum revolution.

In applications where computing time must be minimized, it is occasionally necessary to re-number the address channel for minimum-access coding of the problem in hand. The re-numbering can be accomplished using the Flexowriter (or Flexowriter tape) and a special circuit built into the computer. Provision is also made in the computer for the re-recording of the start pulse channel.

Access to the Memory. In the process of carrying out an arithmetic command, two operands must be "looked up", the operation performed, and the result "put away" in the memory. Consider the example given.

35 0123 0126 0277

"Add contents of 0126 to contents of 0123, put away sum in 0277".

In the execution of all arithmetic commands, the command (instruction and three addresses) is read into the H register. Then the contents of m_1 of H is read into the E register, the contents of m_2 of H is read into the F register, the operation (add, divide, etc.) is carried out with result appearing in the E register, replacing the operand, which is no longer needed and the new contents of the E register are read into the memory cell called out in m_3 of H.

In the example, m_1 of H is 0123. Channel 01 is selected electronically, and the output of M_w (word channel) is compared with the digits 23. When coincidence occurs, the channel 01 read circuit is connected to the input of the E line for one word time only, causing the content of cell 0123 to be read into the E line, where it will re-circulate (the information also remains in cell 0123). Then, as m_2 of H is 0126,

channel 01 would be selected again, and 26 is compared to the word channel output. At coincidence, the next word from channel 01 is read into the F register, where it will recirculate.

The addition is performed, with result being inserted into the E line, replacing the augend m_1.

Then the E line is connected to the channel 02 record circuit for one word time after coincidence between the last digits of m_3 of H (77) and the word channel, accomplishing the desired putaway of the sum into cell 0277.

Note that the actual location of a sector on the drum is physically displaced one word space from the word channel number associated with it. This allows time for the sector look-up circuits to operate in the manner described.

Special Addresses. 2000 Series (buffer) - Whenever the first two digits of an address are 20, the buffer is selected instead of a main memory channel. Comparison is made between only the last digit of the word channel and the last digit of the desired address. At coincidence, the proper connection is made between the E or F register and the buffer output or input circuits.

"Cell 2100" - When the digits 2100 are called out as an address in the m_1 or m_2 (lookup) positions in H, (not in m_3), the E or F register will be filled automatically with zeros. This cell does not exist physically, as filling with zeros is accomplished electronically by breaking the E or F line for one-word time. Therefore there can be no putaway to "Cell 2100". If 2100 is put into m_3 of H, the 1 in the second place of the address will be ignored and the information will be put away in cell 2000 (cell 00 of buffer).

Cell 3000 - If 3000 is called for in m_1 of H, the contents of the G register are transferred to the E register. If 3000 is called for in m_2 of H, the contents of the G register are transferred to the F register. 3000 in m_3 of H will result in buffer cell 00 putaway as for "Cell 2100".

Access Time. The drum rotates at 40 RPS, requiring 25 milliseconds for one revolution. If a desired word has just passed a main memory channel read-write head when sector lookup commences, a full revolution will be required before transfer of information will take place. Thus for the main memory, maximum access time is 25 milliseconds. Mean random access time is about half this figure or 12.5 milliseconds. For the buffer, maximum access time is 25/8=3.125 milliseconds; mean random access time is about 1.56 milliseconds.

FILLING THE MEMORY - STARTING COMPUTATION

Filling E Register From Flexowriter. Between the write head and the read head of the E register, there is a chain of five flip-flop delay stages (E_1, E_2, E_3, E_4, E_5). During normal recirculation of the contents of E, the read circuit is connected to E_1, E_1 to E_2, E_2 to E_3, E_3 to E_4, E_4 to E_5, and E_5 to the record circuit or E line input, E_0. The spacing between the write head and the read head is adjusted such that the combination of drum and flip-flops always contains exactly 42 bits. The flip-flops are driven by clock pulses in synchronism with the drum so that each bit progresses down the flip-flop chain at the same rate as on the surface of the drum.

In order to change the information in the E register, the E_5-E_0 connection is broken and the new information is fed synchronously into E_0, low order digit first, for the required number of pulse (bit) or digit times. This action will shift bits

into the low order end of the E line, and an equal number of bits will be lost at the high order end of the E line.

The actual shifting in of information is accomplished by inserting additional flip-flops (one or more of A_1, A_2, A_3, A_4) into the E line between E_5 and E_0, for one or more word times. If the A flip-flops are preset to a particular state, the number (binary, octal, or coded-decimal) represented by that state will be shifted into the right hand (low order) end of the E word.

For example, suppose the E line is cleared to all zeros, then flip-flops A_1, A_2, and A_3 are all set to the "1" position and inserted in the E line for one word time.

E line before: 000 000 000 000 000 000 000 000 000 000 000 000 000 000

E line after: 000 000 000 000 000 000 000 000 000 000 000 000 000 111

In the octal notation, this would be written

Before 00.0000 0000 0000

After 00.0000 0000 0007

This process is used to fill information from the Flexowriter into the E register. With the control console set for octal fill (fill switch on, "Base 8" light on) every time a numeral key from 0 through 7 is struck (lower case L = 1), flip-flops A_1, A_2, and A_3 are set to the correct three-bit state representing the octal digit and are then connected into the E line for one word time. If the keys 7, 3, 4, 2 are struck in order, the E register would progress as follows:

00.0000 0000 0000
00.0000 0000 0007
00.0000 0000 0073
00.0000 0000 0734
00.0000 0000 7342

Note that, in order to write the number 00.7342, it would be necessary to strike the 7, 3, 4, and 2 keys in order, then strike the zero key eight times. As a convenience to the operator, the "f" key on the Flexowriter may be used to fill four zeros at one time in such situations. Thus 00.7342 could be filled by striking 7342 ff. (Always assuming the E register was cleared originally).

Hereafter the octal notation will ordinarily be used, but it should be remembered that an octal number such as 00.7342 would actually appear in the computer as 000 000.111 011 100 010 000 000 000 000 000 000 000 000. In preparation for filling octal digits, if the "Base 8" light is out ("Base 10" light on) striking the letter o on the Flexowriter or momentarily depressing the base 8 button on the control panel will set up the machine for octal filling.

Selecting an Address for Putaway. Before filling the first number or command into the computer, a memory location for that word must be selected. The computer is set up so that depression of the "Tab" (tabulate) key on the Flexowriter transfers the contents of the E register to the memory cell called out in the m_3 position of the H register.

The "hyphen" (distinguished from "minus" sign) key on the Flexowriter transfers the contents of the E register to the H register. To select a momory cell for putaway, type the four octal digits of the address, then strike the hyphen key. The address will transfer in to m_3 of H and the E register will be cleared.

Example - selecting cell 1324 for next putaway:

E 00 0000 0000 0000

H 00 0000 0000 0000

Type 1324

E 00 0000 0000 1324

H 00 0000 0000 0000

Type "hyphen"

E 00 0000 0000 0000

H 00 0000 0000 1324

If the number to be put away into the memory is 03.7654 4412 0000, -- type 03 7654 4412 f (f = Fill four zeros)

E 03 7654 4412 0000

H 00 0000 0000 1324

Type "tab"

E	00 0000 0000 0000
Sector 24 channel 13 (cell 1324)	03 7654 4412 0000
H	00 0000 0000 1325

The first word has now been put away in cell 1324 and the E register is cleared for the next word. Note that 1 was automatically added to the m_3 portion of H. This facilitates filling a long series of words into consecutive memory locations without the necessity of writing the next address each time.

Decimal Fill. Striking the "d" key on the Flexowriter or momentarily pressing the "Base 10" button on the control panel permits filling binary-coded decimal information into the memory, according to the 8-4-2-1 system.

Decimal	Binary
0	0000
1	0001

Decimal	Binary
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Four binary digits are required for each decimal digit filled, therefore, the word capacity is now 9 decimal digits and sign.

As each number key is struck, flip-flops A_1, A_2, A_3, A_4 are set up in the correct configuration corresponding to each binary-coded decimal digit, then inserted into the E line for one word time, shifting digits in from the right as before.

The computer must be returned to the "octal fill" condition before any commands are inserted, as all operation numbers and addresses are in the octal system.

N O T E

The computer is not capable of operating directly on binary-coded decimal numbers; a suitable decimal to octal conversion routine must be included in all programs in which computation using decimal information is contemplated.

Starting Computation. After the commands and numbers constituting a program are filled into the computer, the address of the first command must be written in the m_2 position of the H register before computation is started. This is done by typing the four octal digits of the address of the first command, then typing f (fill four zeros) then striking the hyphen key (transfers E to H).

The H register will then contain the address of the first command in the m_2 position.

Striking the s key will then start computation. The first command will be looked up and executed, and the program will be carried out in the desired order.

COMMANDS IN THE CRC 102-A

The Instruction Digits. Octal digits 0_{13} and 0_{12} of a command are the instruction digits. Any octal combination from 00 to 37 may be transferred from the memory to the command register H, during computation.

The significance of each configuration of $0_{13} - 0_{12}$ is indicated in the following table.

Octal Code (0_{13} on left)	Abbreviation	Meaning
00-03		Numbers - cause alarm*
04	b0	Buffer Out
05	bl	Buffer Load
06	cr	Read Card
07		Extra - causes alarm*
10		Extra - causes alarm*
11	fl	Fill (from paper tape)
12	pd	Punch Decimal
13	po	Punch Octal
14	bs	Block Search
15	wt	Write (magnetic) Tape
16	rt	Read (magnetic) Tape
17	ts	Test Switch -- Test Search
20		Extra - causes alarm*
21	pr	Print (on Flexowriter)
22	ht	Halt (computation)
23	dr	Divide and Round-Off
24	dd	Divide and Save Remainder
25	mr	Multiply and Round-Off

* When used as an instruction.

Octal Code	Abbreviation	Meaning
26	md	Multiply Double Length
27	sl	Shift Logically
30	sm	Shift Magnitude
31	sf	Scale Factor
32	ex	Extract
33	ta	Test Algebraically
34	tm	Test Magnitude
35	ad	Add
36	su	Subtract
37	to	Test for Overflow Marker

For detailed explanation of all instructions, see NCR Electronics Division publication "Programming Manual for Model 102-A".

Execution of Commands. In the normal course of computation, the command is first read into the H register. Then the contents of the m_1 address is read into the E register, and the contents of the m_2 address is read into the F register. In arithmetic operations the result appears in the E register, from which it is transferred to the memory location called for in m_3 of H.

All putaways in the memory come from the E register. In the case of double-length results (multiply double length, divide with remainder) the first half of the result is transferred from E to the memory, as the second half is transferred from G to E, then the second half of the result is transferred from E to the next physically adjacent memory cell on the drum.

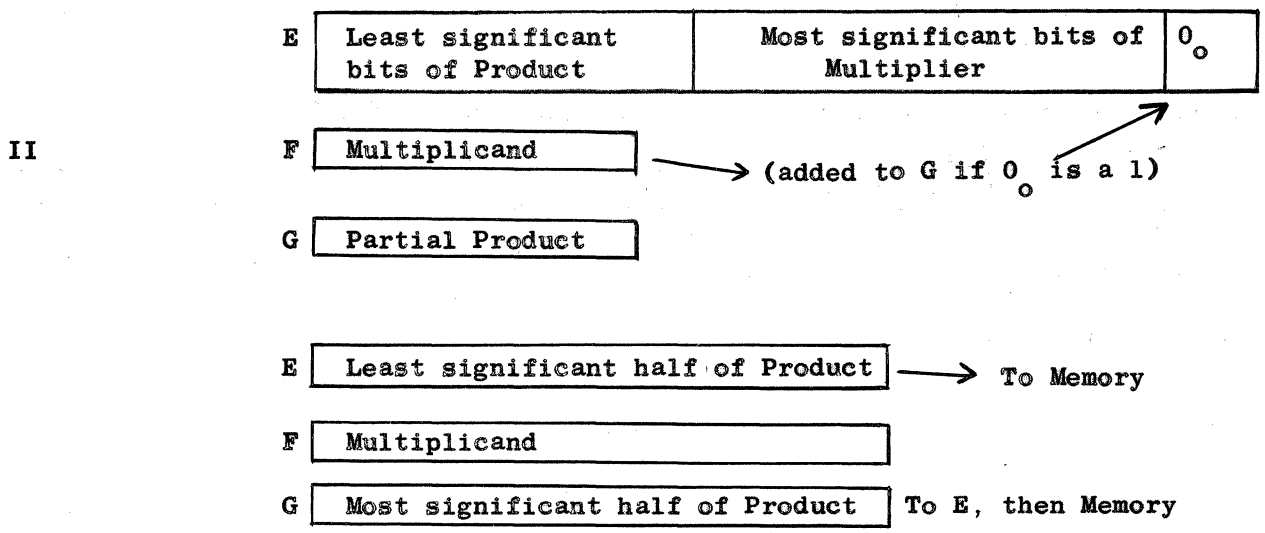
Example - double length multiplication (Instruction No. 26)

If the command to be executed reads 26 0102 0103 0244 and the content of cell 0102 is 00 0000 0000 0006 and the content of cell 0103 is 00 0000 0000 0004 the operation proceeds as follows:

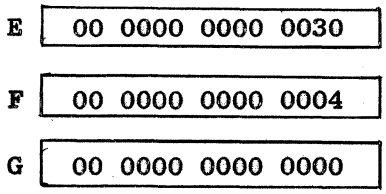
	E	00 0000 0000 0000
Start	F	00 0000 0000 0000
	H	26 0102 0103 0244
	E	00 0000 0000 0006
Look up m_1	F	00 0000 0000 0000
	E	00 0000 0000 0006
Look up m_2	F	00 0000 0000 0004

During the multiplication, the content of the F register is added to the G register if the right-hand bit of the E register is a 1. Then the E and G registers are shifted right one bit, with the right-hand bit from the G register shifting into the left end of the E register. If the right hand bit in the E register is a 0, no addition takes place, and the E and G registers are shifted right one bit, as shown in the following steps:

	E	Multiplier
I	F	Multiplicand
	G	Partial Product



The numerical result, with the example given, would be



The multiplication takes place only over the magnitude section of the two numbers. The proper sign is automatically inserted in the sign position of both halves of the product.

B. Example - division with remainder

In division with remainder, the dividend is in E, the divisor is in F at the start of the operation. As the division progresses, F is compared with E and if smaller, F is subtracted from E, a 1 is recorded in G, and E and G are shifted left. At the end of the division, G will contain the quotient and E the remainder. Content of E is put away in cell called for in m_3 of H, contents of G going to F. Next contents of E (quotient) is put away in the next physically adjacent cell on the drum.

Other commands are carried out in similar fashion.

Modifying a command. One of the most useful features of the CRC 102-A is its ability to modify commands during computation. If the address of a command is called out in the m_1 position of an add or subtract command, the address portion of the command can be changed by adding or subtracting any number. The operation portion of the command will be unaffected by the addition or subtraction. The address portion of the command being altered will always be treated as a positive number.

GENERAL DESCRIPTION OF CRC 102-A OPERATION

Computer Logic. The CRC 102-A computing circuits have been designed by means of two-valued symbolic logic, and all signals in the logical networks of the computer are constrained to one of two voltage levels, +100 volts or +125 volts. The output of any circuit which is restricted to these levels is called a proposition, analogous to the semantic application of symbolic logic. A proposition is "true" or in the "one" state if the voltage is +125 or "high"; a proposition is "false" or in the "zero" state if the voltage is +100 or "low". The flip-flops used as part of the recirculating registers, and additional flip-flops used for logical, control, and other purposes, operate (either directly or through driver amplifiers) between the two logical voltage levels. Mixing and gating of signals between flip-flops is accomplished by germanium diodes arranged to carry out the functions of logical sum (mixing) and logical product (gating or coincidence).

ORGANIZATION OF THE COMPUTER

The Program Counter - The Flow Diagram. The number of flip-flops in the computer has been limited to the maximum number required to carry out a single operation. The

machine carries out operations in a time-sequence with the interconnections between flip-flops correctly arranged in the diode network for each step in the sequence according to the state of a nine-stage binary register called the program counter.

The program counter can represent, binary fashion, any octal number from 000 to 777. The program counter controls the state of the diode network and thus the interconnection between memory, recirculating lines, input-output devices, and logical flip-flops.

Operation of the program counter is automatic. It is not accessible to the programmer.

The time-sequential nature of the logical circuits does not readily permit separation into a conventional block diagram, but the sequence of events for any particular operation-cycle can be shown in a flow diagram (See CRC Drawing No. 20243003).

Each block in the flow diagram corresponds to a configuration of the program counter as shown by the octal number in the upper left-hand corner of the block.

The program counter may change its state in one of two ways - it may count progressively, 000,001, 002, etc., or it may skip from one state to another as required by the outcome of a particular operation.

A horizontal line leaving a block indicates counting in the program counter; a vertical line leaving a block indicates skipping. In several cases (see Blocks 0, 7, 173) the program counter may skip back into the configuration just left. This operation is referred to as sticking. In leaving any block there are only two alternatives for the program counter: count or skip (where sticking is included as a special case of skipping).

Whether the program counter is to count or skip from a block is determined by its previous state and the result of the operation carried out in that block.

Except when sticking, the program counter stays in one configuration (one flow diagram block) for one-word time (14 octal digits, or 42 bits, approximately 420 u sec). At the end of each word-time, (actually during $P_2\ 0_{13}$) the state of the decision flip-flop (K) determines whether the program counter is to count or skip. If K is false, the program counter counts into the next block; if K is true, the program counter skips (or sticks) as shown in the flow diagram. The state of K at the end of a word-time is a result of the logical equations describing the operation during that word time and the actual propositions being handled during the word time.

Example - Fill 7 into E

As an example of a flow diagram sequence, consider the operation of filling an octal 7 into the E register from the Flexowriter.

When the computer has been turned on, and the warm-up period is over, the program counter returns to 0 (rest - upper left hand corner of flow diagram). Assuming the "Base 8" button has been pressed, and the fill switch turned to "fill" position, the 7 key on the Flexowriter is struck, setting ones into logical flip-flops A_1, A_2, and A_3. Striking any significant Flexowriter character also sets K false and the program counter counts to block 1 (001), where the signal sent from the Flexowriter is examined to determine whether it was a character (0-7) or a control symbol (d, o, s, f, hyphen, tab). Identification of the 7 as a character sets K true and the program counter skips to block 41 where A_1, A_2, and A_3 are inserted into the E line for one word time, shifting E left one octal digit and inserting a 7 in the least significant

position. (A_4 is inserted into the E line only when decimal information is filled.

K is always true coming out of block 41, so the program counter skips to block 40, where flip-flops A_1 - A_5 are reset to 0. The program counter skips to block 0, K remains true, and the program counter sticks in 0 repeatedly until another significant Flexowriter key is struck.

Function of Flip-Flops.

Arithmetic flip-flops A_1 through A_{12} are set up through the diode network to perform the function required in each program counter block.

Decision and carry flip-flop K is set up at the end of each word-time, in order to determine whether the program counter is to count or skip. In some program counter blocks, K is used during the remainder of the word-time as a carry flip-flop in addition, and for other arithmetic purposes.

Program counter flip-flops N_1 through N_9 control the interconnection of all logical elements (flip-flops, recirculating line and memory inputs and outputs, etc.). The logic for the program counter flip-flops is such that they operate as a conventional 9-digit binary (3-digit octal) counter if K is false, advancing one step at the end of each word time. If K is true at the end of a word time, the N flip-flops are forced into a non-consecutive configuration determined by the previous program counter number. It is frequently arranged that the state of only one N flip-flop need be changed to accomplish the skip. For example from flow diagram block 24 (000 010 100), K true at end of word causes skip to block 64 (000 110 100), requiring only that N_6 be set true.

Octal digit counter flip-flops D_1 through D_4 function as a binary counter which

re-cycles after counting to 13. The flip-flop outputs drive an array of logical product circuits (matrix) which produces a true output proposition on one of thirteen output lines corresponding to the location in time of each of the octal digits 0_0 through 0_{13}. These octal digit propositions are used to control the logic of operations which are to take place during only part of a word (sign changes in 0_{12}, channel and sector selection in 0_0 - 0_3, etc.).

Pulse counter flip-flops B_1 - B_2 function as a binary counter which re-cycles after a count of 2 and is driven by the clock pulses from the drum. The pulse counter outputs are combined in a matrix as above to provide output propositions P_0, P_1 and P_2 which are true during one bit-time of each octal digit-time. The P_2 proposition drives the octal digit counter. Thus an operation which is to occur during the least significant bit of each octal digit would be controlled by proposition P_0; an operation which is to occur during the most significant bit of the m_1 address portion of a word would be controlled by P_2 and 0_{11} (from the octal digit counter).

Record flip-flops R_1 and R_2 are used to control all recording of information in the memory. R_1 is set true for transferring the contents of the E register to the main memory (M), R_2 is set true for transferring the contents of the buffer (j) to the main memory.

Channel selector flip-flops L_1 through L_5 are set up to correspond to the first two digits of an address for lookup or putaway of information. Some examples of channel selection are shown below.

Octal Channel No. (First two digits of address)	Channel Selector Flip-Flops L_5	L_4	L_3	L_2	L_1	Location Selected
00	0	0	0	0	0	Channel 00 Main Memory
05	0	0	1	0	1	" 05 " "
17	0	1	1	1	1	" 17 " "
20	1	0	0	0	0	Buffer
21	1	0	0	0	1	(In lookup-"Cell 2100" (In putaway - Buffer
30	1	1	0	0	0	Cell 3000

Function of Recirculating Registers During Computation

First Command Lookup. After a program has been filled into the memory, including a sequence of commands and the necessary numbers (constants, etc.), pressing the "start" button or striking "s" on the Flexowriter will start computation.

The number in the m_2 position of the H register (the control number) is the address of the first command which will be carried out. If the H register was cleared before the start signal, the computer will look in cell 0000 for the first command. If the first command is located in (say) cell 0136, the operator must type 0136 f - (hyphen) before starting computation. This will put address 0136 in the m_2 position of H.

Sending a start signal causes the program counter to count to block 6 (see flow diagram). In blocks 6 and 7 the address in m_2 of H (first command) is looked up, and in block 10, the first command is read into the H register. At the same time, the previous contents of H are transferred to G with one being added to the control number, m_2, in the process.

Example - Add Command

A. Command lookup. Assuming all registers (except H) cleared at start, and the first command (in cell 0136) was 35 0123 0124 0263, the register contents would change as follows:

I	E 00 0000 0000 0000 F 00 0000 0000 0000 G 00 0000 0000 0000 H 00 0000 0136 0000	Going into block 6, flow diagram
II	E 00 0000 0000 0000 F 00 0000 0000 0000 G 00 0000 0137 0000 H 35 0123 0124 0263	At end of block 10, flow diagram

B. E and F lookup. In blocks 11 and 12 the address of the first operand (0123) is looked up, and in block 13 the contents of m_1 of H are read into E. In blocks 14 and 15 the address of the second operand (0124) is looked up and in block 16 the contents of m_2 of H are read into F.

If cell 0123 contained 00 1111 1111 1111
and cell 0124 contained 00 2222 2222 2222

the register content at the end of block 16 would be:

I	E 00 1111 1111 1111 F 00 2222 2222 2222 G 00 0000 0137 0000 H 35 0123 0124 0263	At end of block 16, flow diagram

As the information in m_2 of H is no longer needed, the new control number (address of next command) is transferred from G back to H in block 17, leaving G available for arithmetic use if required.

	E	00 1111 1111 1111	
	F	00 2222 2222 2222	
II			At end of block 17, flow diagram
	G	00 0000 0137 0000	
	H	35 0123 0137 0263	

The address 0127 also remains in G, but filling in another word in G will shift out this superfluous information.

C. Test for Instruction. In blocks 20 through 37, 1 is added to the instruction portion of H (35 in example) in each block starting with 20. The decision flip-flop, K, remains false until the addition results in an overflow into the $P_2\ 0_{13}$ position, at which time K is set true and the program counter skips as indicated. In the example the instruction number (I) is 35 (add). In block 20, I is increased to 36; in block 22, I becomes 40 (100 000); K is set true and the program counter skips to block 122 - start of add routine.

D. Putaway of Results. At the conclusion of the add routine (block 124, 125, or 127 depending on the nature of the operands) the program counter skips to block 134 for the putaway routine.

At the start of block 134 the registers appear as follows:

E	00 3333 3333 3333	(sum)
F	00 0000 0000 0000	(cleared)
G	00 0000 0137 0000	(G was not used in the add routine)
H	40 0123 0137 0263	

In blocks 134 and 135 the channel and sector of the putaway address (0263) are located and in block 136 the contents of E are transferred to the appropriate memory cell. Block 137 is used for putting away the second half of double-length results, when required.

After putaway, the program counter skips to block 6; the next command is looked up (in cell 0137 in example) and a new command cycle commences.

ALARMS

Non-Existent Instruction or Number for Command. If, during the examination of the instruction portion of H for the type of instruction (blocks 20-36, 400-403, 410-417, flow diagram), a configuration representing a number (00, 01, 02, 03) or a non-existent instruction (07, 10, or 20) is found, the program counter will skip to block 434, initiating a routine for printout of the content of the G register. The information printed out will indicate which step in the program caused the alarm. (For details of G register printout, see CRC 102-A Specifications).

After the printout, the program counter will skip to 0 and the computer will be in rest.

Automatic Overflow Test. If an addition, subtraction, or division operation gives a result $\geq$ 1 in magnitude, and the "automatic overflow test" switch is in the "in" position, the contents of the G register will be printed out as above, and the computer will halt, unless the next instruction is "Test for Overflow" (to) or "Shift Logical" (sl). In these latter cases it is assumed that the operator has anticipated the overflow and computation will continue according to the program.

PROGRAMMING

Programs. In order to carry out a computation or series of computations, the CRC 102-A must be programmed by the operator.

In general, the operator will fill a group of commands into the memory, fill the numerical information into the memory, fill the address of the first command into m_2 of H and start the computer. Ordinarily the program will include one or more output commands (print, punch IBM cards, etc.) and a command to halt at the end of the program.

The computer ordinarily carries out a program according to the sequence of commands in the program, obtaining a command from the memory, executing that command, storing the results (if any), obtaining the next command from the memory, executing the command, etc.

Only one arithmetic operation can be carried out for each command (some commands require two or more logical steps in the computer, such as "multiply and round off", but such a group of steps will be referred to as one operation, as it is called for by one instruction, in this case "mr"). For instance, "add two numbers together and divide the sum by two", would require two commands in the computer, one for the addition and one for the division.

Filling Methods. The information may be filled directly into the computer using the Flexowriter, or a Flexowriter tape can be punched and the taped information filled into the computer at some later time by running the tape through the tape reader attached to the Flexowriter on the computer control console. In either case, a printed record of the program will be typed by the Flexowriter as the program is fed into the computer.

Review of Flexowriter symbols for filling computer:

Character	Flexowriter Key	Set-up of A flip-Flops in Computer A_5	A_4	A_3	A_2	A_1	Notes
C H A R A C T E R	0	0	0	0	0	0	All characters: contents of A_1 - A_4 are filled into E register if filling in decimal mode
	1 (Lower Case L)	0	0	0	0	1	Contents of A_1 - A_3 are filled into E register if filling in octal mode
	2	0	0	0	1	0	
	3	0	0	0	1	1	
	4	0	0	1	0	0	
	5	0	0	1	0	1	
	6	0	0	1	1	0	
	7	0	0	1	1	1	
	8	0	1	0	0	0	Will fill correctly only in decimal mode. If filling octally 8 fills as 0, 9 fills as 1
	9	0	1	0	0	1	
	+	0	0	0	0	0	Same as 0
	- Minus	0	0	0	1	0	Same as 2 (distinguished from hyphen key)

	Flexowriter Key	Set-up of A Flip-flops in Computer A_5	A_4	A_3	A_2	A_1	Notes
	Space	0	1	1	0	0	Special characters for decimal filling. If filling octally space fills as 4, period fills as 7
	Period	0	1	1	1	1	
C	d	1	1	0	0	1	Sets A_6 true for filling four bits from A_1 -A_4 into E register
O							
N	o	1	1	0	0	0	Sets A_6 false for filling three bits from A_1 -A_3 into E register
T							
R							
O	Hyphen	1	1	1	0	0	(Distinguished from minus key) Transfers content of E register to H register
L							
	Tab	1	1	1	1	0	Records contents of E register in memory. (In main memory, P_2 0_{13} always records as 0)
S							
Y							
M							
B	s	1	1	1	1	1	Starts computation
O							
L	f	1	0	0	0	0	Fills 4 zeros in either decimal or octal mode, during initial filling of cleared registers
S							
	Note:	A_5 identifies control symbols					

A Sample Program

A. Problem and Approach. As an example, assume it is desired to compute the square root of an octal number, X between 0 and 0.7777 7777 7777.

One method of computing the square root in such a case is as follows:

1. Assume initially that the square root of X is the largest possible value, 0.7777 7777 7777. Call this the first trial value T_1
2. Square T_1
3. Compare T_1^2 with X
4. If T_1^2 is within 0.0000 0000 0002 of X, assume that the computation is finished and that $T_1 = \sqrt{X}$ (as nearly as this method permits)
5. If T_1^2 exceeds X by more than .0000 0000 0002, obtain a second trial value T_2, according to the formula $T_2 = \frac{T_1}{2} + \frac{X}{2T_1}$. (It can be shown that a sequence of such trial values will converge to the square root of X).
6. Repeat steps 2, 3, and 4 using T_2.
7. If computation is not finished, obtain T_3 by the method of step 5 ($T_3 = \frac{T_2}{2} + \frac{X}{2T_2}$).
8. Repeat steps 2, 3, and 4 using T_3.
9. Continue the process until step 4 determines that computation is finished.
10. Print the result on the Flexowriter.
11. Cause the computer to halt.

B. Computer Steps. The above sequence of events, broken into steps for computer operations, can be tabulated to indicate the sequence of commands required.

1. Set up T = .7777 7777 7777
2. Multiply T by T
3. Subtract X from T^2
4. Compare result with .0000 0000 0002
5. If T^2 - X exceeds .0000 0000 0002 divide X by T (If not, 9. Print T)
6. Add result to T (10. Halt)
7. Divide sum by two, giving $\frac{T}{2} + \frac{X}{2T}$. This is new trial value.
8. Return to step 2 using new trial value

C. Allocating Memory Space. Ordinarily a set of consecutively numbered memory cells is allocated for the commands in a program, and another set of cells is set aside for numbers and storage of results.

From the above diagram it can be seen that approximately ten commands will be required for the program. Let us reserve cells 0100 thru 0117 in the main memory for commands and put the numbers in cells starting with number 0120.

D. Making Out The Program. If cell 0120 is used for the location of the trial value of the square root the first step in the program requires 00.7777 7777 7777 to be placed in cell 0120. This could be accomplished with an add command:

Commands		Numbers	
Address	Command	Address	Number
0100	ad 0121 2100 0120	0120	Trial value of sq. root
	(contents of "cell 2100" = 0)	0121	00.7777 7777 7777

The next step calls for the squaring of the trial value:

0101	mr 0120 0120 0122	0122	Temporary storage of intermediate results

Then X must be subtracted from T^2

0102	su 0122 0123 0122	
	(the previous information in 0122 is no longer needed so this result can now be stored there).	

E. Complete Program. After proceeding similarly through the entire tabulation of operations the complete program appears as follows:

Commands		Notes
Address	Command	
0100	ad 0121 2100 0120	Adds .7777 7777 7777 to 0 and puts result in cell 0120. Effectively copies contents of 0121 into 0120 ("contents" of 2100 are always 0)
0101	mr 0120 0120 0122	T^2 in 0122

Commands			Notes
Address	Command		
0102	su 0122 0123 0122		T^2 -X in 0122
0103	tm 0122 0124 0106		Test for end of computation
0104	pr 0120 2100 0001		If end reached, print T, otherwise jump to 0106 for next command. (2100 in m_2 causes printout in octal mode, full length. 0001 in m_3 means print one word only.
0105	ht 2100 2100 0000		Halt. 2100 2100 0000 provides minimum access time. Any addresses may be used.
0106	dr 0123 0120 0122		If end not reached, divide X by T X/T in 0122
0107	ad 0122 0120 0122		$\frac{X}{T}$ + T in 0122
0110	sl 0122 0125 0120		Shift content of 0122 one binary digit right. Divides by two; also removes overflow if any. T/2 + X/2T in 0120 ready for next trial.
0111	tm 3000 2100 0101		Unconditional jump to 0101. Content of cell 3000 is always 0 in this case, since the control number >0.

Numbers		Notes
Address	Number	
0120	Latest trial value of square root	Replaced by new value at sl command
0121	00.7777 7777 7777	First trial value
0122	Temporary storage of intermediate results T^2, T^2 -X, X/T, T + X/T	No two of these results are needed simultaneously.
0123	X	Number for which square root required
0124	00.0000 0000 0002	Used in tm command to determine whether to print or continue trials
0125	02.0000 0000 0001	02 indicates right shift. 1 X 2^{-12} indicates one bit shift

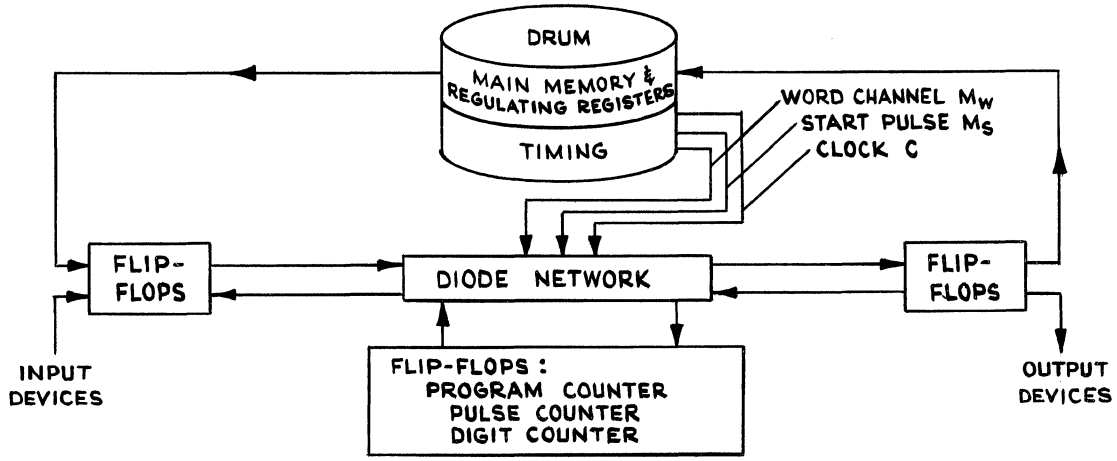
F. Filling the Program into the Computer. Assuming the computer and Flexowriter to be turned on, the following sequence of steps would fill the sample program and start computation:

Step	Notes
Turn on "fill" switch	Connects Flexowriter to Computer
Press "clear" button	Clears and synchronizes computer circuits
Press "base 8" button or "o" key on Flexowriter	Prepares computer for filling octal mode.

Step	Notes
Type 0100	Fills "0100" into $0_3 0_2 0_1 0_0$ of E register
Strike "hyphen" key	Transfers 0100 (address of first putaway) to m_3 of H
Type 35 0121 2100 0120	Puts first command in E register (35 = add)
Strike "tab" key	Transfers 35 0121 2100 0120 to cell 0100. Increases m_3 of H to 0101 (address of next putaway)
Type 25 0120 0120 0122	Puts second command in E register (25 = mr)
Strike "tab" key	25 0120 0120 0122 to cell 0101. Increases m_3 of H to 0102
Continue to putaway of last command in cell 0111, then	
Type 0121	Fills 0121 into $0_3 0_2 0_1 0_0$ of E register
Strike "hyphen" key	Transfers 0121 (address of next putaway) to m_3 of H. Contents of cell 0120 makes no difference: first step in program puts desired information into cell 0120
Type 00.7777 7777 7777	Fills number into E register
Strike "tab" key	00.7777 7777 7777 to cell 0121. m_3 of H increases to 0122
Strike "tab" key	Skips cell 0122, m_3 of H increases to 0123
Type sign (00) and magnitude X	Fills X into E register
Strike "tab" key	Puts X into cell 0123; increases m_3 of H to 0124

Logical Structure of the CRC 102-A

COMPUTER BLOCK DIAGRAM AND DESCRIPTION



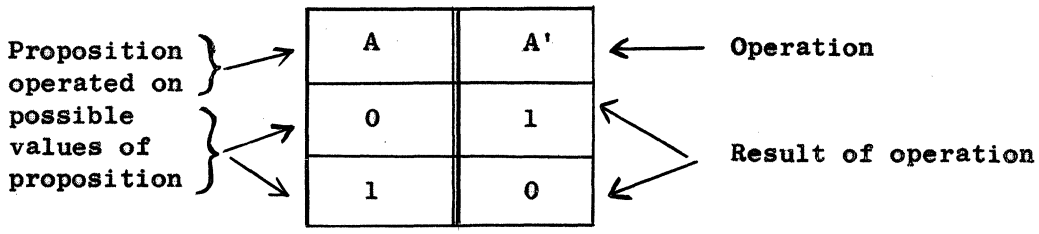
The clock provides the basic repetition rate for the computer.

Clock Waveform	+125 v
(C)	+100 v

The computer circuits are designed through the use of symbolic logic. The logic is handled in mathematical form according to a set of rules called Boolean Algebra. A proposition in the computer is analogous to a proposition in semantics - it can be in one of two states, true or false, corresponding to one of two voltages appearing at the point under consideration in a network or circuit. "True" is indicated by a high (+125 v) voltage. "False" corresponds to a low voltage (+100 v). True and false are also represented by the binary digits 1 and 0 respectively.

By defining certain logical operations, we can combine propositions in various ways to achieve desired electronic results in the computer. The operations used are negation ("Prime", "not", or "inversion"), "and" (logical product, gating, or coincidence), and "or" (logical sum or mixing).

The simplest logical operation is negation. If a proposition, A, is true, then its negation, A', is false. This relation can be shown in a truth table as follows:



The "and" (logical product) operation combines two or more propositions in such a way that both must be true for the result to be true; the symbol for logical product is the same as the algebraic symbol for multiplication, a dot between the two letters representing the propositions, or simply juxtaposition of the two letters: A.B = AB = A and B = (logical) product of A and B.

Truth Table for Logical Product

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

AB = BA

ABC = A(BC) = (AB)C

A.1 = A

A.0 = 0

The "or" operation (logical sum) combines two (or more) propositions in such a way that the sum is true if at least one of the propositions is true. The symbol for logical sum is +.

Truth Table for Logical Sum

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

$A + B = B + A$

$A+B+C = A+(B+C) = (A+B)+C$

$A + 1 = 1$

$A + 0 = A$

Some additional rules for logical algebra.

$AA' = 0$ always false

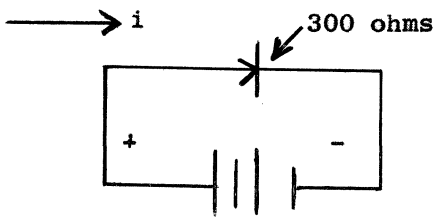
$A+A' = 1$ always true

$A(B+C) = AB+AC$

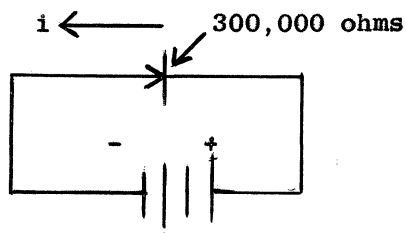
$(AB)' = A' + B'$

$(A+B)' = A'B'$

Propositions to the left of the double line in the charts above can be considered as imputs to a logical circuit, and propositions to the right as outputs. Germanium diodes are used in the CRC 102-A computer to perform the logical product and sum operations. The diode is a non-linear electronic component which exhibits a low resistance (say 300 ohms) to current flow in one direction and a high resistance (say 300,000 ohms) to current flow in the opposite direction.

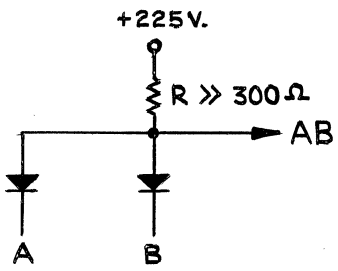


Diode Conducting

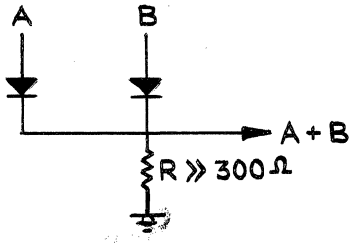


Diode cut-off

The logical operations of sum and product are performed with diodes as follows:

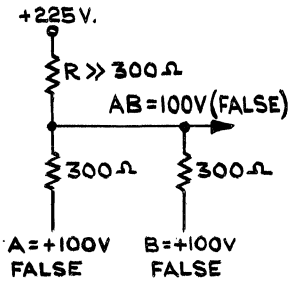


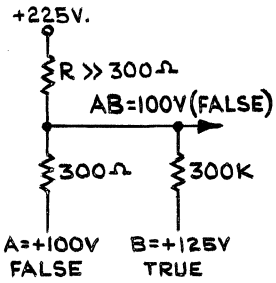
LOGICAL PRODUCT

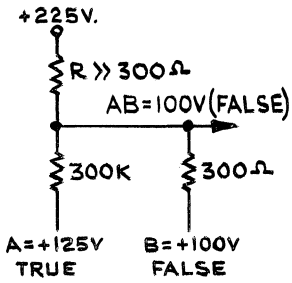


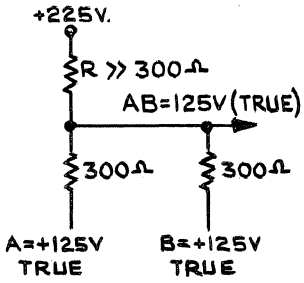
LOGICAL SUM

Equivalent circuits for four cases of logical product:





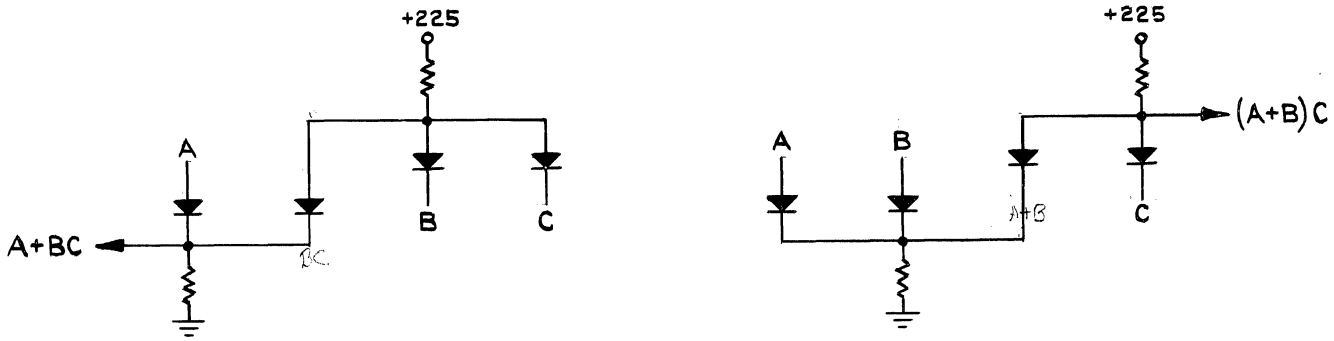




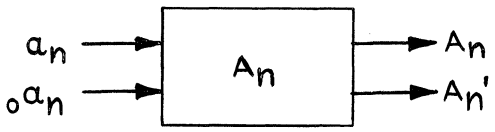
Reference to the truth table for logical product shows that the above circuit satisfies the four requirements. Similarly, equivalent circuits for the four cases of logical sum verify the operation of the sum circuit.

Note that the output of a logical circuit is of the proper form and voltage to be the input to another circuit.

TYPICAL COMBINATIONS OF LOGICAL CIRCUITS

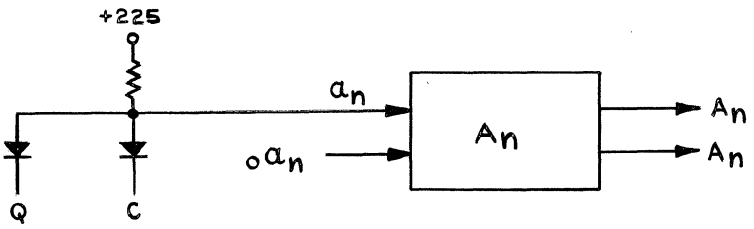


Obtaining the Primes. The negations (primes) are obtained from bistable (two-stable-state) electrical circuits (flip-flops) or from drivers (inverters). The flip-flop has two inputs and two outputs. One output is the negation of the other.



Flip-flop Representation

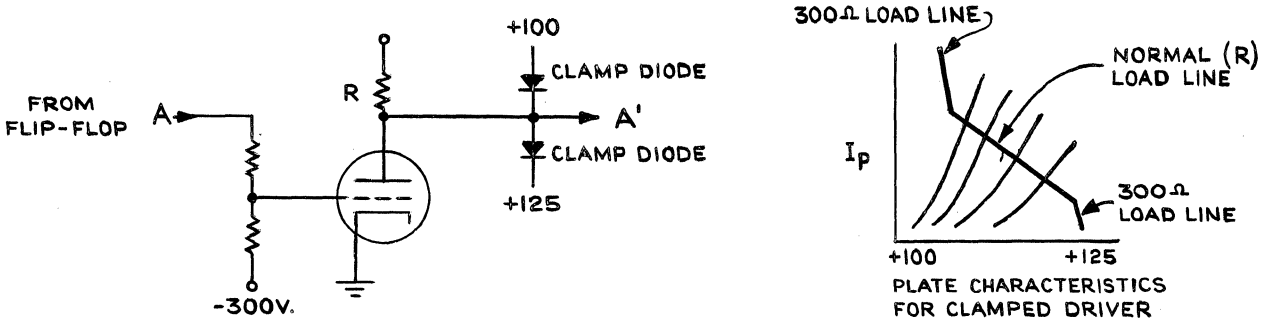
A negative pulse on input a_n sets output A_n true (+125 v.) and output A_n' false (+100 v.). A negative pulse on input $_0a_n$ sets output A_n false and output A_n' true. Pulsing a_n is also called "setting 1" or "setting A_n true". Pulses for setting flip-flops are obtained by coincidence (logical product) with the fall of the clock waveform. (C).



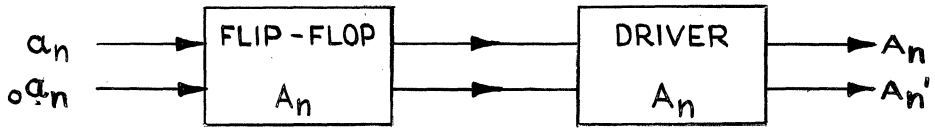
If Q is true, flip-flop A_n will be set true when C drops from + 125 to +100 v, if $_1$ was originally in the false state. If A_n was originally in the true state, it will remain so until a negative pulse is applied to $_0a_n$. Clock products used to drive flip-flops in this fashion are called grid propositions.

The driver stage is an amplifier with output clamped by diodes to limit the output voltage to the range +125 volts. Drivers are usually placed between flip-flops and large diode networks to supply the large currents required without undue loading of the flip-flop.

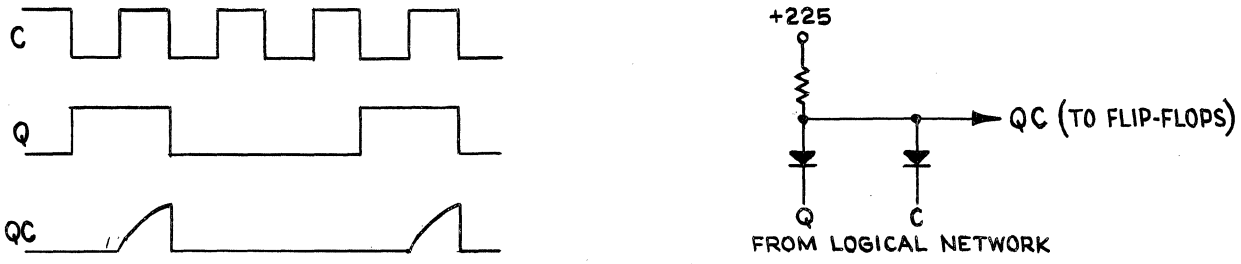
SIMPLIFIED DRIVER CIRCUIT WITH OUTPUT CLAMPING



Dual triode tubes are often used so that both outputs of a flip-flop may be amplified in one unit.



In order to synchronize the operation of various flip-flops in the computer, a product is formed between logical network output propositions and the clock signal.

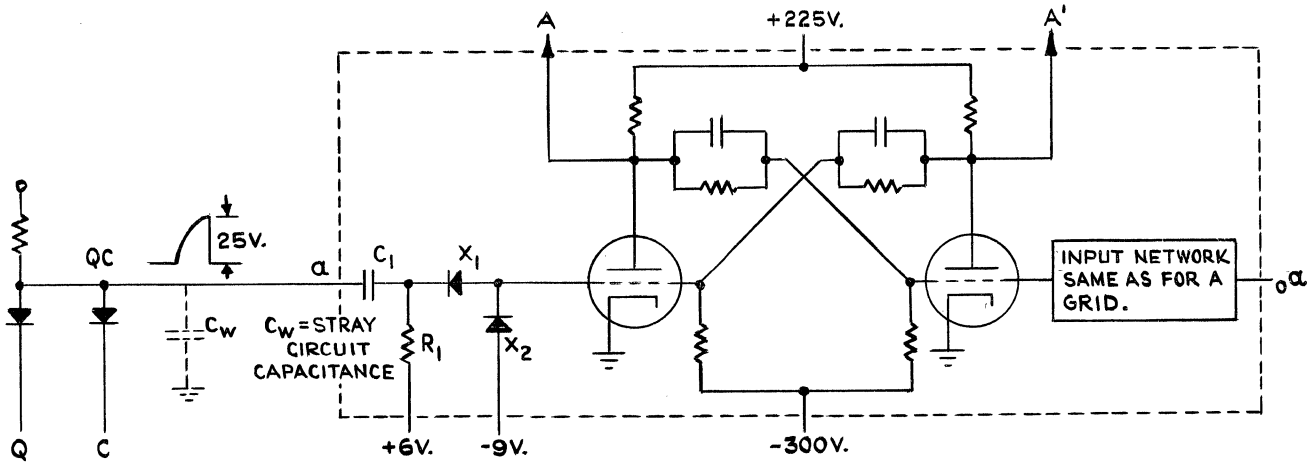


The exponential rise of the product depends on the time constant of the product resistor and the stray circuit capacitance. The resistor is selected as large as possible in order to conserve network current, yet not so large as to prevent the clock product from rising to full 125 v. amplitude in 5 microseconds. The fall time of the clock pro-

duct is short, as the drivers for the product diodes represent low impedance sources. It is the fall of the product proposition which triggers the flip-flops. The flip-flop output cannot change until one clock pulse time after the input proposition (Q) goes true. This is an important consideration in the design of the computer circuits.

The flip-flop input circuit is designed to respond only to negative-going pulses over a certain minimum amplitude.

FLIP-FLOP INPUT CIRCUIT
(In plug-in Unit)



If the flip-flop is in condition to be triggered by a negative pulse, the left grid will be at about 0 volts, diode X_1 will be cut off until the input signal drops at least 6 volts, and diode X_2 will be cut off until the grid drops to at least -9 volts. Thus, if a negative stray or noise pulse of 5 volts or less is applied to the flip-flop input, no signal will reach the grid.

However, when the desired trigger signal, QC, is applied to the flip-flop input, it is differentiated by the input capacitor, C, the resulting negative spike is well over 6 volts in amplitude, and the flip-flop reverses its state, the left hand grid dropping to -9 volts, where it is clamped by diode X_2, and the left-hand plate rising to about +180 v. No further change will take place until a negative pulse is applied to the opposite flip-flop input.

When the left-hand grid is at -9 volts, the flip-flop could be triggered by a stray positive pulse at the grid, but as diode X_1 is cut off, application of positive pulses to the flip-flop input produces no effect.

LOGICAL DESIGN

A logical equation for a network is derived from the information contained in a truth table by equating the proposition at the head of an "output" column of the table to the (logical) sum of all combinations of the input propositions which correspond to a "1" in that output column. Each combination of input propositions is expressed as a (logical) product.

Where the output is a grid proposition, C (clock proposition) must be included as a factor in each product.

Example: Writing logical equation from truth table

Truth Table

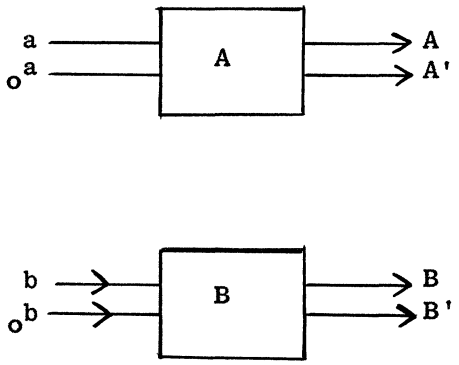
A	B	D	$_{o}a$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Resulting Equation:

$$_{o}a = (A'BD + AB'D' + ABD')\,C$$

As an example of logical formulation, consider the use of two flip-flops as a scale-of-3 counter. A table of the desired successive states of the flip-flops can be drawn as follows:

Time	A	B
t-1	1	0
t 0	0	0
t_1	0	1
t_2	1	0
t_3	0	0



Then the desired grid signals for each next successive state of the flip-flops can be filled in:

One Cycle

A	B	a	$_o$a	b	$_o$b
0	0	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
0	0	0	0	1	0
		etc.			

As the operation is cyclic, only the first three tabular entries need be considered. The logical equations for the network can now be written, where C is the clock proposition needed for timing the counter.

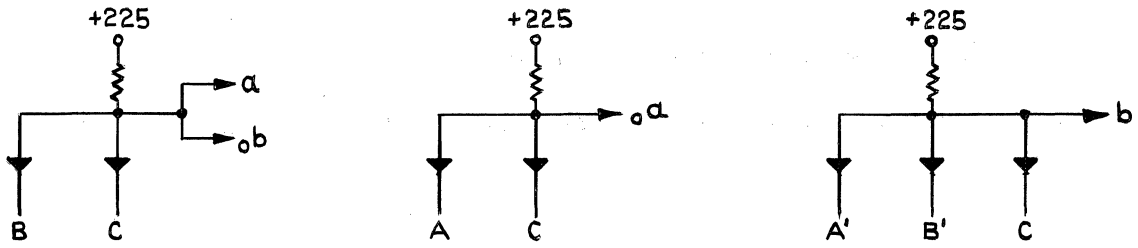
$$a = BC$$

$$_oa = AC$$

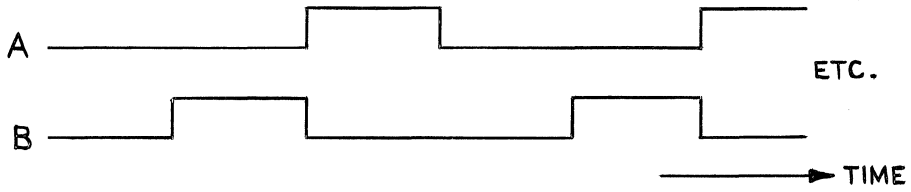
$$b = A'B'C$$

$$_ob = BC(= a)$$

Then the corresponding diode circuit can be drawn.

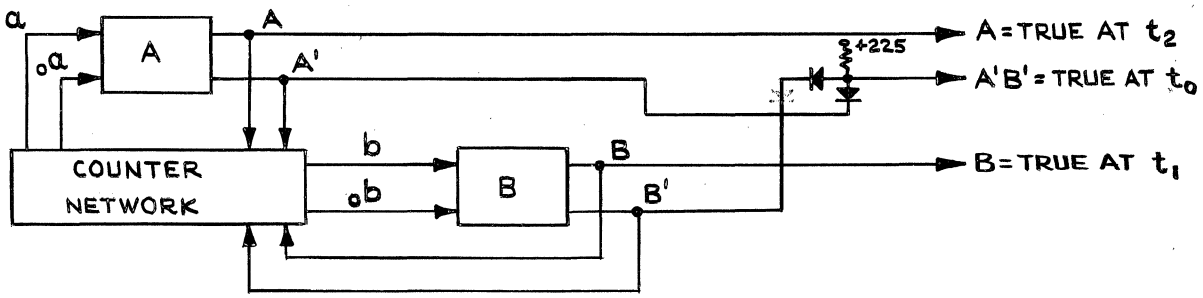


The flip-flop output propositions A and B would then appear as follows. (Assume at start the flip-flops are in the A'B' state)



Thus the two flip-flops and the diode network form a two-digit binary counter which resets to 0 after a count of two. A similar counter (pulse counter) is used in the computer to locate the positions in time of bits P_o, P_1, P_2, of each octal digit.

If a unique output is required for each configuration of such a counter, a logical matrix is used which produces a true output on a separate line for each state of the counter. For the above scale-of-three counter a very simple matrix results.



In the computer, flip-flops B_1 and B_2 constitute the pulse counter, and the product $B_1' B_2'$ in the output matrix is also used as part of the counter network.

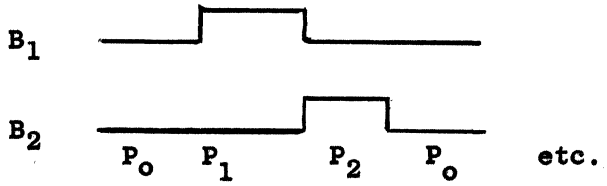
Simplified logic is as follows:

$$b_1 = P_o C \qquad {}_ob_1 = B_1 C \qquad b_2 = B_1 C \qquad {}_ob_2 = B_2 C$$

where

$$P_1 = B_1 \qquad P_2 = B_2 \qquad P_0 = B'_1 B'_2$$

FLIP-FLOP OUTPUT WAVEFORMS IN PULSE COUNTER



The complete logic includes the reset proposition Z which is true when the reset button is pressed, and the start pulse M_s which occurs at $P_0{}^0{}_0$ time.

Expressed in words, the pulse counter should count as above except when the reset proposition is true, when it should set to the P_1 state ($B_1B'_2$) at the time the start pulse occurs.

COMPLETE LOGIC FOR PULSE COUNTER

$$b_1 = (P_0 + Z M_s) C$$

$${}_ob_1 = B_1 (Z' + M'_s) C$$

$$b_2 = B_1 (Z' + M'_s) C$$

$${}_ob_2 = (B_2 + Z M_s) C$$

From the above it can be seen that whenever ZM_s is true, the parenthetical portions of the first and fourth equations will be true and the counter will set to the P_1 state. Whenever Z is false, the logic reduces to the simplified case.

A more complicated case of a counter and matrix is the digit counter, which locates the position in time of each of the octal digits 0_o through 0_{13}. It is a scale-of-14 counter, using flip-flops D_1 through D_4 and the logic is as follows:

$$d_1 = D'_1 \ (Z' + M_s') \ P_2C$$

$$_od_1 = (D_1P_2 + ZM_s) \ C$$

$$d_2 = D_1D'_2 \ (D_3' + D_4') \ (Z' + M_s') \ P_2C$$

$$_od_2 = (D_1D_2P_2 + ZM_s) \ C$$

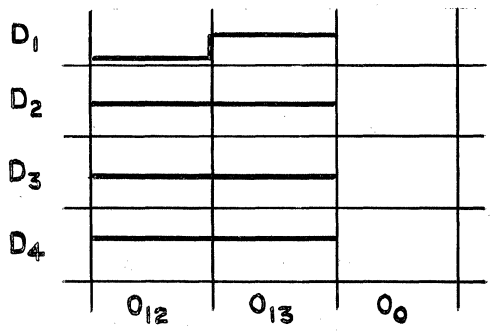
$$d_3 = D_1D_2D_3' \ (Z' + M_s') \ P_2C$$

$$_od_3 = (D_1D_2D_3P_2 + P_2O_{13} + ZM_s) \ C$$

$$d_4 = D_1D_2D_3D_4' \ (Z' + M_s') \ P_2C$$

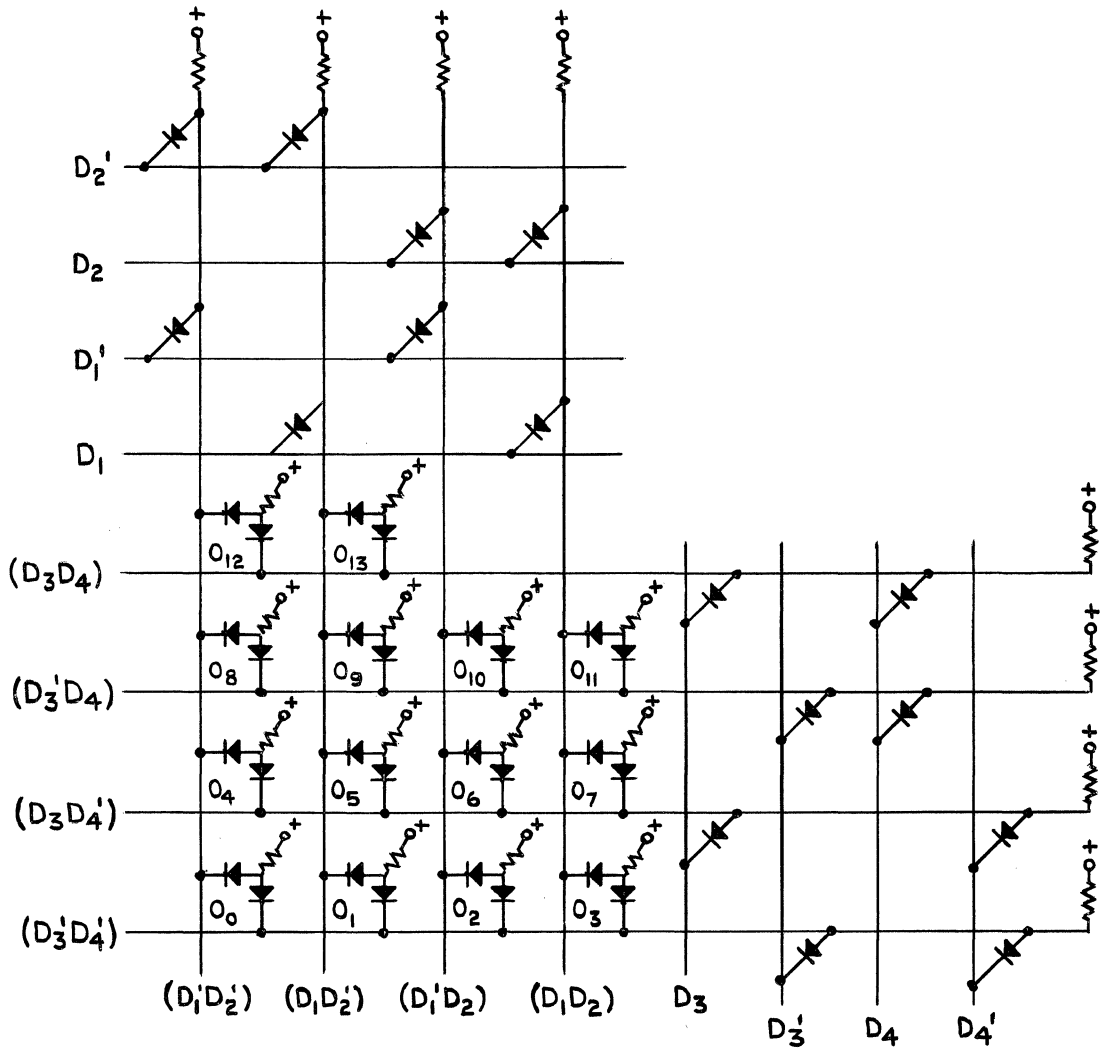
$$_od_4 = (P_2O_{13} + ZM_s) \ C$$

Assuming Z is false which removes all terms of the form ZM_s and all factors of the form $(Z' + M_s')$ and noting that factors P_2C mean that the change to the next state occurs every third bit (every octal digit), it can be shown that the counter progresses in ordinary four-digit binary fashion from 0_0 to 0_{13} at which time the flip-flop output propositions are as follows:



At time $P_2 0_{13}$, D_1 is set 0 by $D_1 P_2 C$, d_2 and $_o d_2$ are both false, so D_2 remains in the 0 state, and D_3 and D_4 are set 0 by $P_2 0_{13} C$, thus the counter is reset for the next cycle.

A matrix for producing a unique output corresponding in time to each octal digit could be constructed as follows:



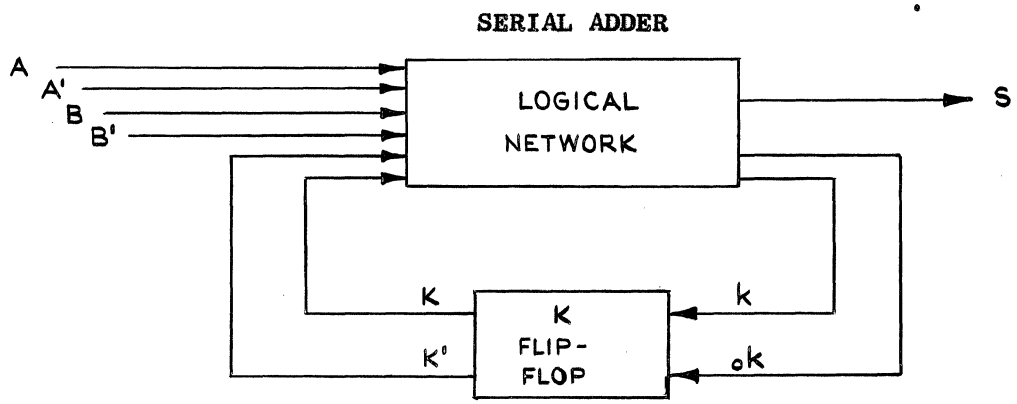
Writing $0_0 = D_1' D_2' D_3' D_4'$

$0_1 = D_1 D'_2 D'_3 D'_4$

$0_2 = D'_1 D_2 D'_3 D'_4$

etc. to $0_{13} = D_1 D'_2 D_3 D_4$

Examination of the completed matrix shows the correct combination of the partial products (shown in parentheses) to produce the desired output propositions as indicated. As another example of a logical network, consider a full serial adder for binary numbers. The adder will have three pairs of inputs, A and A' (Addend), B and B' (Augend), K and K' (carry from previous pulse time). The outputs are S(sum) and the grid propositions for the carry for the next digit k and $_{o}k$.



Verbally, the rules for the adder network are as follows: If there are no 1's in the (unprimed) inputs, the sum output should be 0 and the carry flip-flop (k) should be set to 0. If there is one 1 among the unprimed inputs the sum output should be 1 and the carry flip-flop should be set 0. If there are two 1's among the unprimed inputs, the sum output should be 0 and the carry flip-flop should be set 1. If the unprimed inputs are all 1's, the sum output should be 1 and the carry flip-flop should be set 1.

TRUTH TABLE FOR SERIAL ADDER

A	B	K	S	k	$_ok$
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	1	0
1	1	1	1	0	0

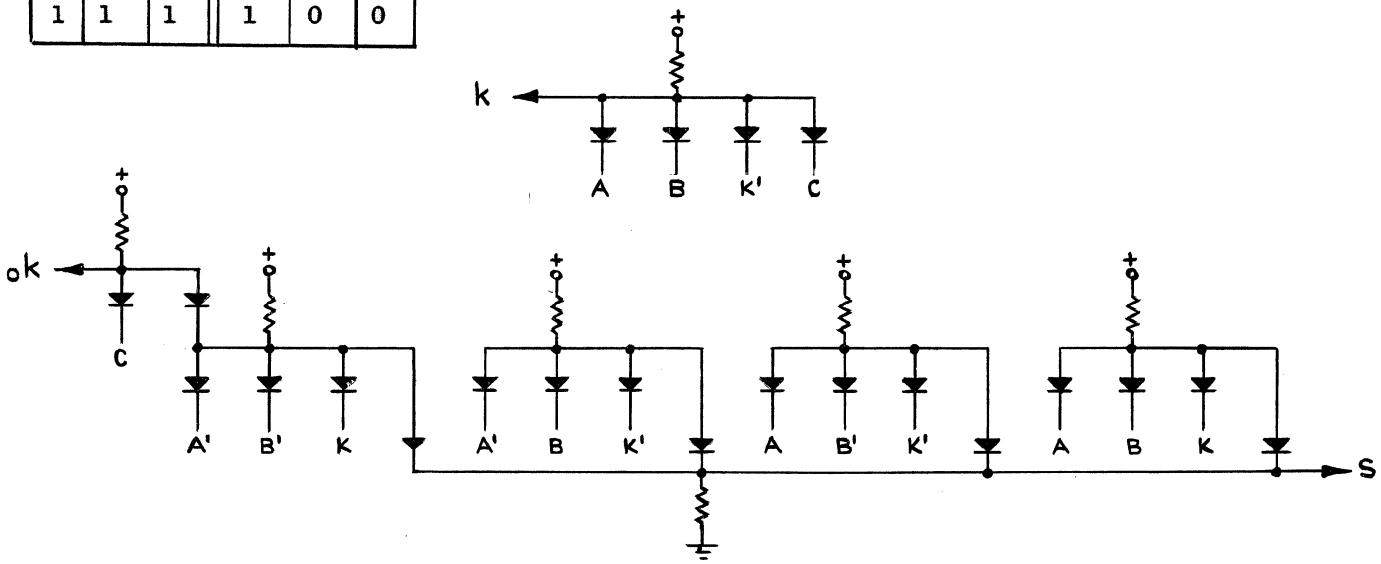
Logic for serial adder

C = clock proposition

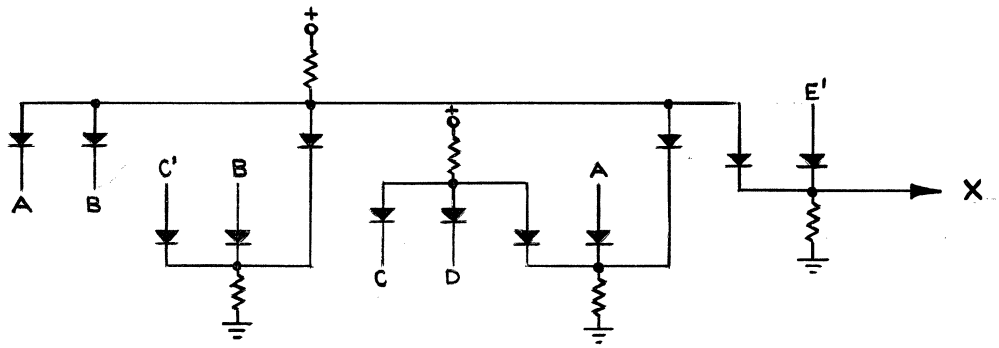
$_ok$ = A'B'KC, k = ABK'C

S = A'B'K + A'BK' + AB'K' + ABK

NETWORK FOR SERIAL ADDER



Example of circuit simplication by logical algebra: consider the proposition X = AB(C' + B) (CD + A) + E'. The network for the proposition as it stands would require 12 diodes.



Multiplying out the expression and discarding factors which are always true and terms which are always false:

$X = (ABC' + AB)(CD + A) + E'$

$X = AB(C' + 1)(CD + A) + E'$

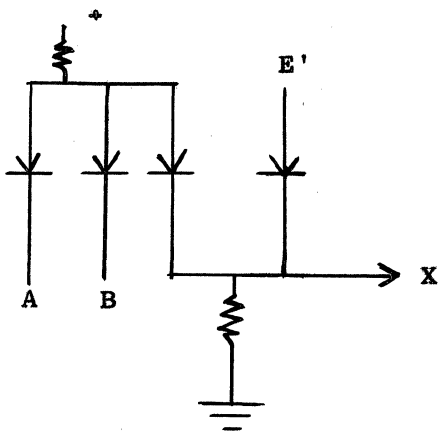
$X = AB(CD + A) + E'$

$X = ABCD + AB + E'$

$X = AB(CD + 1) + E'$

$X = AB + E'$

Only four diodes in resulting circuit:



Example - logic for the "Extract" command.

E_0 = E register input proposition

E_5 = E register output proposition

F_2 = F register output proposition

G_5 = G register output proposition

Part of the extract command (block 100, Flow Diagram) says "Extract the digits of the number in the E register into the number in the G register according to the number in the F register and put the result back into the E register". This is written $E_0 = E_5 F_2 + G_5 F'_2$. Thus, at any particular pulse time during extraction, if F_2 is true, the digit in E_5 will go back into the E register; if F_2 is false, the digit in G_5 will go into the E register.

Example - Test for sign in the ta command. As part of the "Test Algebraically" command (Block 64, Flow Diagram), the number in the E register is compared to the number in the F register. If the number in the E register is larger Algebraically (more positive) than the number in the F register, flip-flop A_1 is to be set true.

At the time the thirteenth octal digit (0_{12}) of the E register is in E_5 and the thirteenth octal digit of the F register is in F_2, flip-flops A_1 and A_2 are in these

states:

If the two numbers are equal in magnitude, A_2 is false; if they are unequal in magnitude A_2 is true.

If the magnitude of the number in E is greater than the magnitude of the number in F, A_1 is true; otherwise A_1 is false.

The second bit of the thirteenth octal digit ($P_1 0_{12}$) indicates the sign of the number, 0 for positive, 1 for negative. The logical formulas for this time reduce to:

$$a_1 = A'_1 F_2 (E'_5 + A_2)\, C$$

$$_o a_1 = A_1 E_5\, C$$

If the two numbers are equal in magnitude, A_2 is false, and the equations reduce to $a_1 = A'_1 F_2 E'_5 C$, $_o a_1 = A_1 E_5 C$.

The results for this case can be shown in a table. (In a previous step, A_1 was set false). In this example, the table on the right expresses in one-zero notation, the information contained in the table on the left.

sign of (E)	sign of (F)	is E>F?
+	+	no
+	-	yes
-	+	no
-	-	no

E_5	F_2	a_1
0	0	0
0	1	1
1	0	0
1	1	0

Thus, A_1 is set true only for the case E> F. If the two numbers are unequal in magnitude, A_2 is true, and $a_1 = A'_1 F_2 C$

$$_o a_1 = A_1 E_5 C$$

With A_2 true,

A_1 means /E/ > /F/

A'_1 means /F/ > /E/

Is /E/ > /F/?	Sign of (E)	Sign of (F)	Is E > F?	A_1	E_5	F_2	a_1	$_o a_1$	Resulting state of A_1
yes	+	+	yes	1	0	0	0	0	1
yes	+	-	yes	1	0	1	0	0	1
yes	-	+	no	1	1	0	0	1	0
yes	-	-	no	1	1	1	0	1	0
no	+	+	no	0	0	0	0	0	0
no	+	-	yes	0	0	1	1	0	1
no	-	+	no	0	1	0	0	0	0
no	-	-	yes	0	1	1	1	0	1

Thus, A_1 becomes (or remains) true only if E > F, and the conditions of the algebraic test are satisfied.

APPENDIX

NUMBER SYSTEMS

Since computers work with numbers, it is necessary to get some idea of exactly how a number system functions. For example, 1954 can be expressed as $1 \times 10^3 + 9 \times 10^2 + 5 \times 10^1 + 4 \times 10^0$. Similarly, a decimal fraction like .30103 can be expressed as $3 \times 10^{-1} + 0 \times 10^{-2} + 1 \times 10^{-3} + 0 \times 10^{-4} + 3 \times 10^{-5}$. From this illustration, it can be seen that any number can be expressed as a combination of powers of ten.

It is possible, however, to use powers of some other number than 10 and equally well represent a number. To see how this may be done, consider using the powers of 5:

Powers of 5	Powers of 10
$(1244)_5$	$(199)_{10}$
$(1 \times 5^3 + 2 \times 5^2 + 4 \times 5^1 + 4 \times 5^0)_{10}$	$1 \times 10^2 + 9 \times 10^1 + 9 \times 10^0$
$(125 + 50 + 20 + 4)_{10}$	
$(199)_{10}$	

Thus, the number $(1244)_5$ is the same as the number $(199)_{10}$.

The number which is raised to the different powers in representing quantities is called the base of the number system. We will make it a practice to indicate the base of the number by a subscript.

It is sometimes useful to represent numbers in terms of powers of 2. This is of value because the base of the number system is always 1 larger than the value of any digit in a number. This means that the only values that the individual digits can take are 1 or 0. These two combinations can be represented by two conditions of a circuit (conducting or non-conducting), (high or low voltage). In order to put this system to use, it becomes necessary to change from the decimal base 10 to the base 2 (i.e. use powers of 2 instead of 10).

Since 2 and 10 are not related by any whole number power and since the number 8 is the third power of 2, in practice it is convenient to first change from the base 10 to the base 8 and from the base 8 to the base 2 according to the following table:

Base 8	Base 2	
0	000	(0)
1	001	(0 + 0 + 1)
2	010	(0 + 2 + 0)
3	011	(0 + 2 + 1)
4	100	(4 + 0 + 0)
5	101	(4 + 0 + 1)
6	110	(4 + 2 + 0)
7	111	(4 + 2 + 1)

There are several schemes to change from the base 8 to the base 10 or vice versa. For example, to change from the base 10 to the base 8, one could write the number in terms of powers of 10, change each number to its equivalent octal (Base 8) value and perform the indicated operations using the octal multiplication and addition tables. However, for one accustomed to decimal arithmetic, the following system is easier:

1954/8 = 244 + 2/8 2 is last digit

244/8 = 30 + 4/8 4 is third digit

30/8 = 3 + 6/8 6 is second digit

3/8 = 0 + 3/8 3 is first digit

Or: $(1954)_{10} = (3642)_8 = (011\ 110\ 100\ 010)_2$, the last number from the table above.
3 6 4 2

To get from octal (base 8) to decimal, the following system is probably the easiest:

$(3642)_8 = 3 \times 8^3 + 6 \times 8^2 + 4 \times 8^1 + 2 \times 8^0 = 3 \times 512 + 6 \times 64 + 4 \times 8 + 2 \times 1 = 1954.$

It should be emphasized that either system may be used for either change, but care must be taken to use the right arithmetic system in each case.

To convert decimal fractions to octal ones, the following system may be used:

.30103	
8	
2.40824	First digit is 2
8	
3.26592	Second digit is 3
8	
2.12736	Third digit is 2
8	
1.01888	Fourth digit is 1
8	
0.15104	Fifth digit is 0

Thus, $(0.30103)_{10} = (0.23210)_8$

The change from octal back to decimal fractions may be made as follows:

$(.23210)_8 = 2 \times 1/8 + 3 \times 1/8^2 + 2 \times 1/8^3 + 0 \times 1/8^4$ .30078125

The difference between these two represents a rounding error. Again it must be emphasized that either of these processes may be used for either conversion, if the proper arithmetic system is used.

Since the 102A uses binary (Base 2) arithmetic, it is convenient for the octal code to be used; it converts easily to binary, and it takes up little more space than decimal, (binary code would take up 3.6 times as many characters).

As a sample of non-decimal arithmetic, let us multiply together two numbers in octal notation. To do this we shall need octal multiplication and addition tables:

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

Addition

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

Multiplication

```
  2357
  1774
 11674
21211
21211
2357
4724104
```

This is done in the same way as decimal arithmetic except that the octal multiplication and addition tables were used. Persons using the CRC 102-A should become familiar with octal arithmetic because it is used by the machine for calculations.

The complement of a number is that number which when added to the given number will clear the storage register of a machine. This number is useful in subtracting in a machine not designed with a special mechanism for subtraction. To see how this might be used, assume that the register of the machine will hold numbers from 0 to 99 with a place for an overflow marker to note when the machine has exceeded 99. Let us subtract 77 from 99:

$$\begin{aligned} 99 - 77 &= 99 + 100 - 77 - 100 \\ &= 99 + 23 - 100 \\ &= 22 + (\text{overflow} - 100) \end{aligned}$$

Since we recognize that the answer is 22, and that the overflow represents the fact that the register has recognized that over 100 counts have been put into it, we may form a rule that if the register shows an overflow, we will throw it away and the

remaining part of the answer is the correct solution. Let us now interchange the numbers so that we may see what we would do it we got a negative answer.

$$
\begin{aligned}
77 - 99 &= 77 + 100 - 99 - 100 \\
&= 77 + 1 \quad - 100 \\
&= 78 - 100 \quad \text{(no overflow)} \\
&= -(100 - 78) \\
&= -(\text{complement of } 78) \\
&= -22
\end{aligned}
$$

Thus, if no overflow is formed in this operation, the answer is minus the complement of the sum. This seemingly involved method is used because the operation of taking the complement is easier, in practice, than a special subtract operation. The method can be used, of course, in any number system.

BOOLEAN ALGEBRA

Perhaps the most important single invention in the field of logic is the invention by Boole of a symbolic representation of propositions in such a fashion that they can be easily manipulated without reference to the contents of the proposition. The key idea is probably contained in the equation $A + A' = 1$. (Read A plus not A is everything). A' or "not A" is thus a symbol for "everything not in A". The other things we are interested in can be represented by $A.B$ (everything in both A and B) and $A + B$ (everything in A or B, or both).

The sort of relations among propositions which interest us are those which are true regardless of the truth of the propositions themselves. These universally true relations are called tautologies. They can be verified by what is called a truth table. In a truth table, one assigns values (true or false) to each proposition and observes the truth of the relation. For example, if 1 represents true, and 0, false, the elementary propositions given above will be seen to be:

A	A'	B	B'	A + B	AB
1	0	1	0	1	1
1	0	0	1	1	0
0	1	1	0	1	0
0	1	0	1	0	0

This table can be explained as follows:

If A is true, not A is false; if A is not true, not A is true

If B is true, not B is false; if B is not true, not B is true

If either A or B or both are true, A + B is true

If and only if both A and B are true, AB is true

This table can be thought of as defining these operations. Note that the arithmetic signs have been chosen so that performing the indicated operation on the truth value of the proposition automatically gives the truth value of the relation.

We can use these relations to verify tautologies. To see how this is done, let us verify the relation A + A' = 1.

A	A'	A + A'
0	1	1
1	0	1

1 (The relation is true because
1 A A and 1
1 agree for all values of the proposition

Again, AA' = 0

A	A'	AA'
1	0	0
0	1	0

Let us verify the following list of useful propositions:

A + A=A

A	A + A
1	1
0	0

BA = AB

A	B	AB	BA
0	1	0	0
0	0	0	0
1	0	0	0
1	1	1	1

A + B = B + A

A	B	A + B	B + A
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

A(B C) (AB)C

A	B	C	(BC)	(AB)	A(BC)	(AB)C
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1

In a like manner, the following propositions may be verified:

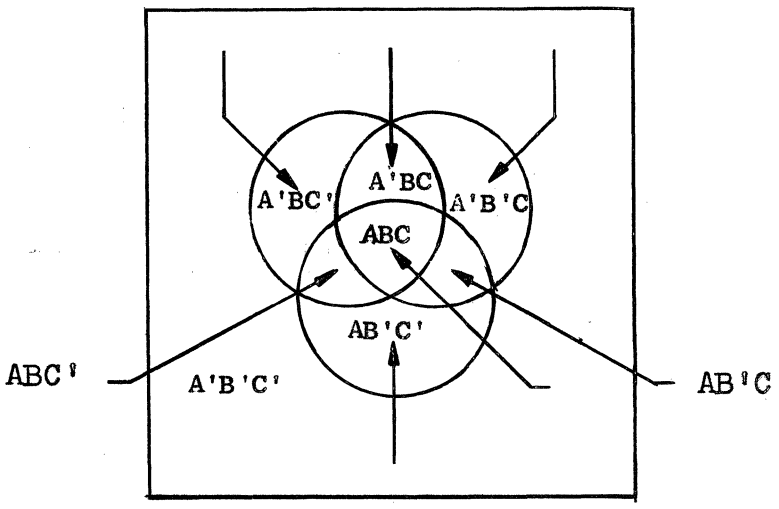
$$A + (B + C) = (A + B) + C$$
$$A(B + C) = AB + AC$$
$$A + BC = (A + B)(A + C)$$
$$A0 = 0$$
$$A1 = A$$
$$A + 0 = A$$
$$A + 1 = 1$$
$$AA' = 0$$
$$(A')' = A$$
$$(AB)' = A' + B'$$
$$(A + B) = (A'B')'$$
$$A + AB = A$$
$$A + A'B = A + B$$

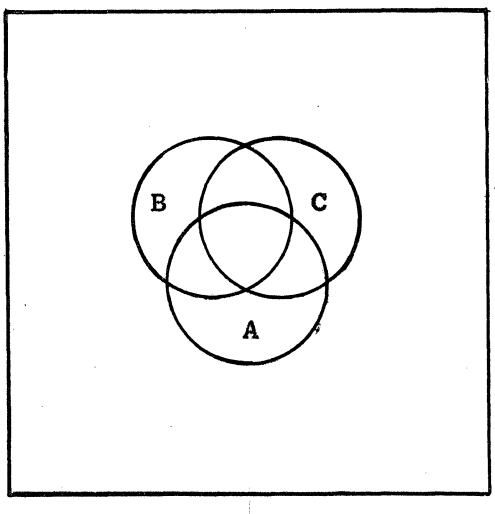
WITH THESE SIMPLE PROPOSITIONS AND THE FURTHER RULE THAT A PROPOSITION MAY BE SUBSTITUTED FOR ITS EQUAL IN ANY EXPRESSION, ANY TAUTOLOGY MAY BE VERIFIED, AND NEW ONES MAY BE DEVELOPED.

Example:

$$ABC + A'BC + AB'C + ABC' + A'B' \cdot C + A'BC' + AB'C' + A'B'C'$$
$$= A(BC + B'C + BC' + B'C') + A'(BC + B'C + BC' + B'C')$$
$$= (A + A')(BC + B'C + BC' + B'C')$$
$$= (A + A')(B(C + C') + B'(C + C'))$$
$$= (A + A')(E + B')(C + C')$$
$$= (I)(I)(I)$$
$$= I$$

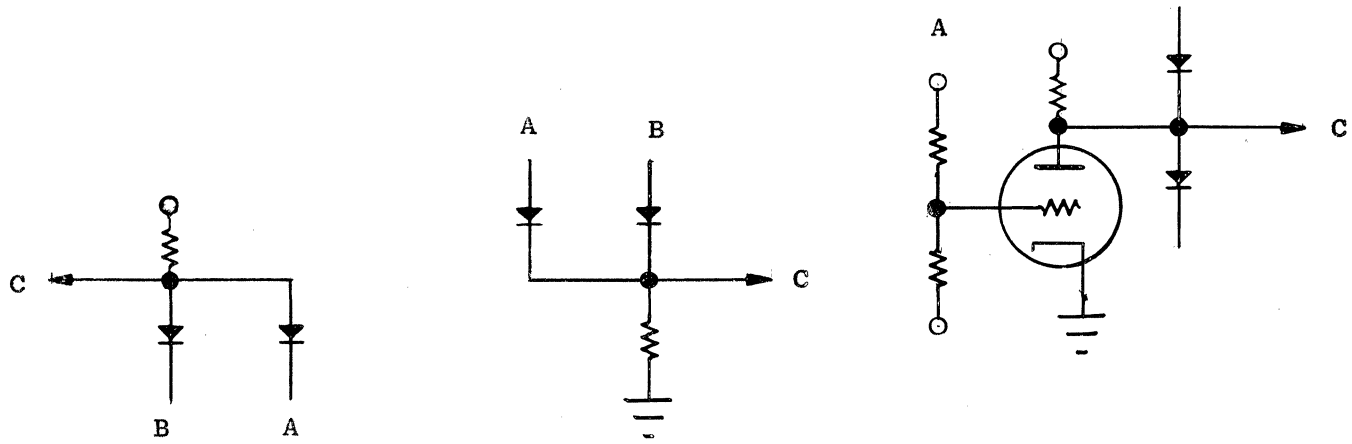
An alternative method of establishing the fundamental relations in Boolean Algebra is by use of a Venn's diagram. In this diagram, I is the area of the rectangle and A, B, and C are the points in each of three circles. A' is then the points in I but outside A, A B is the set of points in A or B or both, and AB is the set common to both circles. This diagram is labeled to show a solution of the example above.





COMPUTER ELECTRONICS

The operations of Boolean Algebra can easily be represented in terms of voltages. In the CRC 102-A the voltages chosen are 125 V for true and 100 V for false. The drivers of the logical circuits are clamped at these values, and these are the reference voltages used in the "not" circuit shown below:



A	B	C
100	100	100
100	125	100
125	100	100
125	125	125

or

0	0	0
0	1	0
1	0	0
1	1	1

C = AB

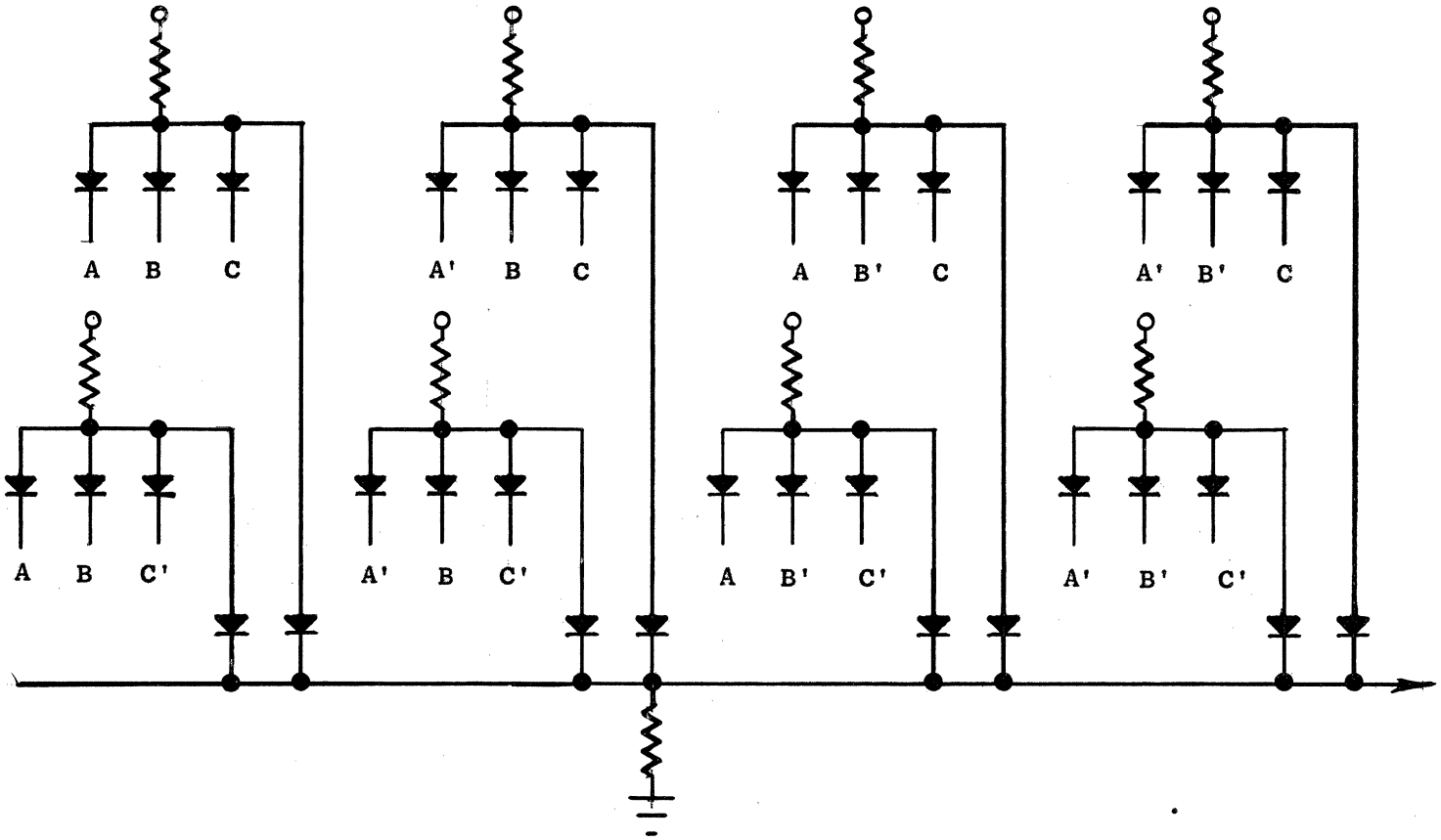
A	B	C
100	100	100
100	125	125
125	100	125
125	125	125

C = (A + B)

A	C
100	125
125	100

C = A'

Thus, we can use a combination of diodes to represent a proposition in Boolean Algebra. If we set up a matrix to give the sum illustrated in the section on Boolean Algebra,


A B C
A' B C
A B' C
A' B' C
A B C'
A' B C'
A B' C'
A' B' C'

The truth table for this net work is:

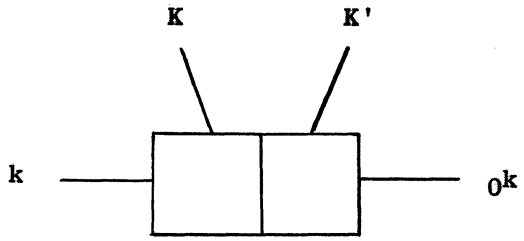
A	B	C	ABC	A'BC	AB'C	ABC'	AB'C'	A'BC'	A'B'C	A'B'C'	SUM
0	0	0	0	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	0	0	1
0	1	1	0	1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	0	0	0	1
1	0	1	0	0	1	0	0	0	0	0	1
1	1	0	0	0	0	1	0	0	0	0	1
1	1	1	1	0	0	0	0	0	0	0	1

This table reveals that each set of the type ABC is true for only one condition of the values of A, B, C, so that it might be considered as a counter with 8 stable states. Thus, by choosing the proper combination of terms any one of 2^8 possible functions might be chosen in the logical networks. Of course, some of these could be simplified in some fashion, but the number of possibilities is still quite large.

As an exercise the basic tautologies in the section on Boolean Algebra may be set up and truth tables made to verify them, using the 1 to signify 125 V condition and 0 for the 100 V condition.

It is important in a computer that all stages of the counter change states at the same time, rather than in serial fashion, as in ordinary scalers. A "clock" circuit synchronizes the machine, delivering one pulse for each binary digit. These pulses serve as gate pulses in the diode networks.

The computer uses flip-flops or Eckles-Jordan circuits to store information and drive the logical circuits. These will be represented symbolically as follows:



Notice that the grids of these flip-flops are not driven by the same signal, so that the proper grid must be triggered to change their states. Clipper diodes are placed in the grid circuits to prevent positive signals from affecting the circuits, and the resultant circuit will respond only to negative pulses. The input circuits are further arranged to differentiate the rectangular clock pulses to provide a sharp triggering waveform.

A scale of 16 counter may be set up using four flip-flops. These counters take on the following states:

A_4	A_3	A_2	A_1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

After the 16th pulse, the counter resets itself to 0000.

Each counter may be considered as a proposition in Boolean Algebra. When A is not conducting, and A' is, the voltage at the plate of A is more positive than that of the conducting A'. These plates are clamped at the logical voltages, 100 V and 125 V. Thus, we may think of A being true when it is not conducting and false when it is conducting, according to the convention we have developed. On inspection of the table for this circuit, it will be observed that when A_1' is true we must change the flip-flop to A_1 is true. This can be done by supplying a clock pulse to A_1. When A_1 is true and the clock is true, the flip-flop should change to A_1' is true. Putting these sentences into Boolean Algebra, we find that

$$a_1 \quad A_1' \, c$$

$$o^{a_1} \quad A_1 c$$

In a like manner, the flip-flop A_2 should be triggered so that A_2 will become true, if the counter now has A_1 true and A_2 false; and A_2 should be triggered so that A_2 will become false if A_2 is true and A_1 is true:

$$a_2 = A_1 A_2' \, c$$

$$o^{a_2} = A_1 A_2' \, c$$

A_3 should be triggered when A_1 and A_2 are true:

$$a_3 = A_1 A_2 A_3' \, c$$

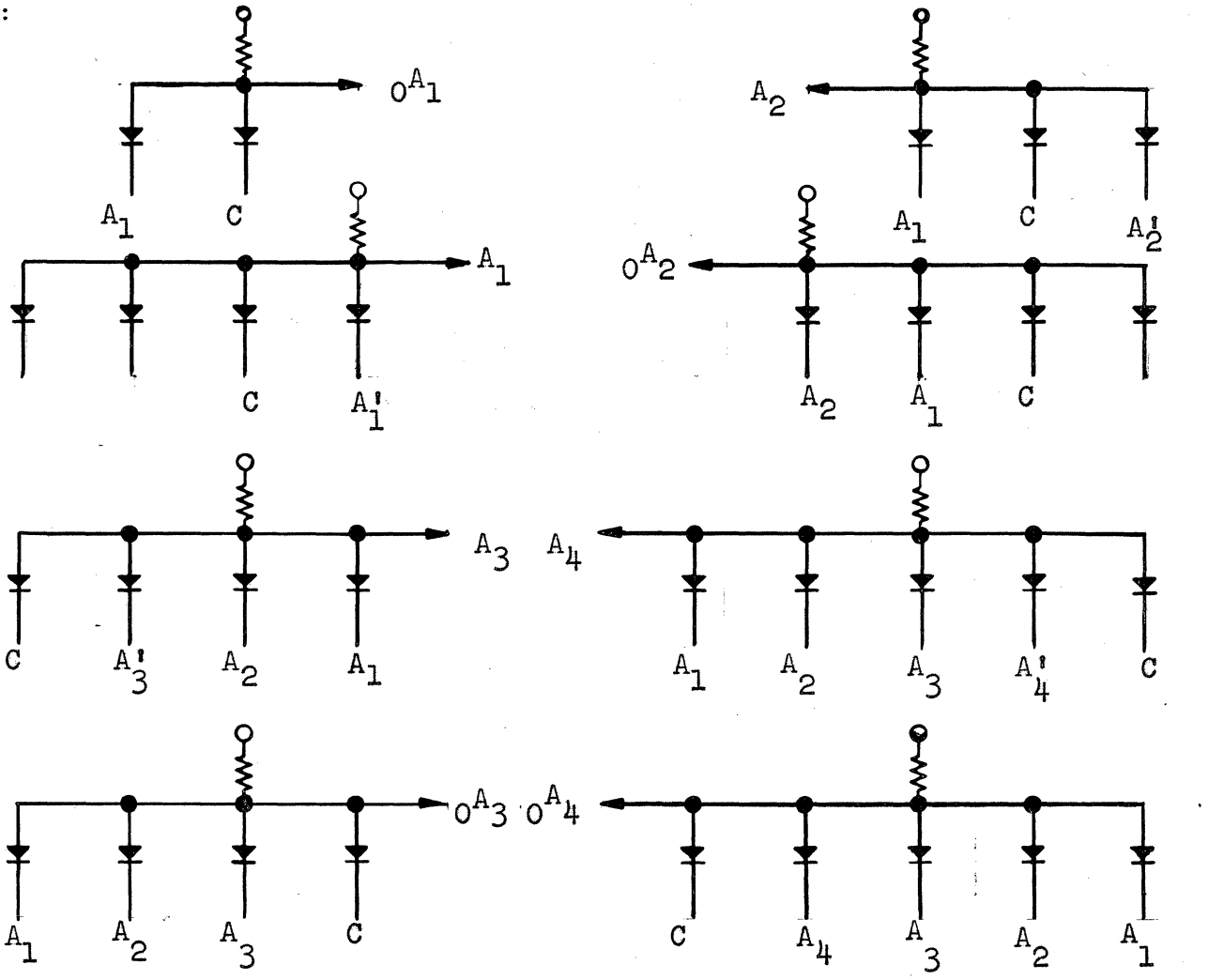
$$o^{a_3} = A_1 A_2 A_3 \, c$$

Similarly,

$$a_4 = A_1 A_2 A_3 A_4' c$$

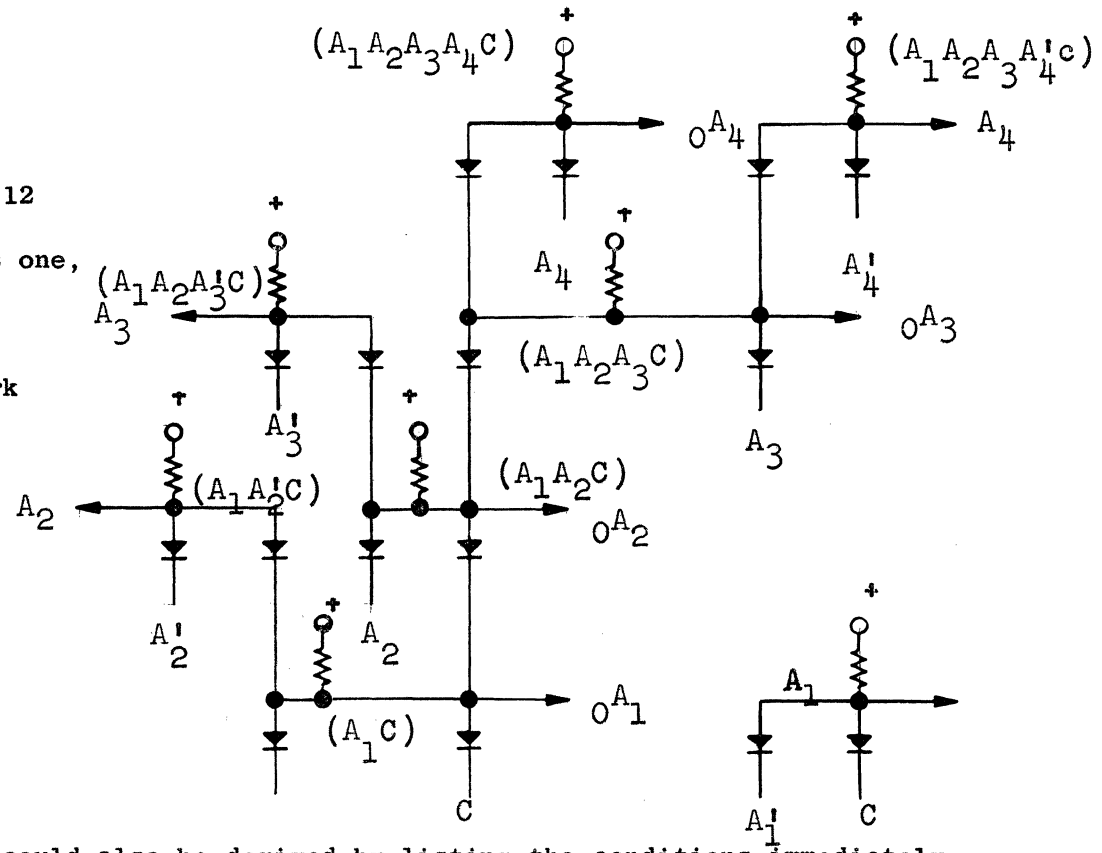
$$0^{a_4} = A_1 A_2 A_3 A_4 c$$

These equations can be translated into diode networks in the fashion shown below:



In practice, we combine these circuits as much as possible in order to save diodes.

This network saves 12 diodes over the previous one, but the total number of resistors in each network is the same: one for each output desired.



The grid equations could also be derived by listing the conditions immediately before a given change of state. For example:

$$a_1 = A_1A_2'A_3'A_4' + A_1A_2'A_3A_4' + A_1A_2'A_3'A_4 + A_1A_2'A_3A_4$$

$$= A_1A_2'(A_3'A_4' + A_3A_4' + A_3'A_4 + A_3A_4)$$

$$= A_1A_2'(A_3'(A_4' + A_4)\ A_3(A_4' + A_4))$$

$$= A_1A_2'(A_3' + A_3)(A_4' + A_4)$$

$$= A_1A_2'$$

As a second example of counter design, let us consider a scaler like the preceding one, except that it shall be made to reset to zero after 13 counts.

A_1	A_2	A_3	A_4
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
0	0	0	0

The following observations may be made:

If A_1' is false, it will change to true at the next clock pulse;

If A_1 is false, it will change to true at the next clock pulse;

If A_2' is false and A_1 is true, A' will change to true on the next clock pulse;

If A_2 is false and A_1 is true and $A_1A_2'A_3A_4$ is <u>not</u> true, A_2 will change to true on the next clock pulse.

If A_3 is false and A_1 and A_2 are true, A_3 will change to true on the next clock pulse.

If A_3' is false and A_1 and A_2 are true of if $A_1A_2'A_3A_4$ is true, A_3' will become true on the next clock pulse.

If A_4 is false and A_1 and A_2 and A_3 are true, then A_4 will change to true on the next clock pulse.

If $A_1A_2'A_3A_4$ is true, A_4 will change to false on the next clock pulse.

If these statements are reduced to Boolean Algebra, they become:

$$0^{a_1} = A_1 c$$

$$a_1 = A_1' c$$

$$0^{a_2} = A_1 A_2 c$$

$$a_2 = A_1 A_2' (A_1 A_2' A_3 A_4)' c$$

$$a_2 = A_1 A_2' (A_3' + A_4') c$$

$$a_3 = A_1 A_2 A_3' c$$

$$0^{a_3} = (A_1 A_2 A_3 + A_1 A_2' A_3 A_4) C$$

$$0^{a_3} = A_1 A_3 (A_2 + A_4) c$$

$$0^{a_4} = A_1 A_2 A_3 A_4' c$$

$$a = A_1 A_2' A_3 A_4 c$$

Consider next the biquinary system as might be used in a counter:

	A_4	A_3	A_2	A_1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

If $A_1 = 0$, it will change to 1 if A_3 is not 1 next clock pulse.

If $A_1 = 1$, it will change to 0 next clock pulse.

If $A_2 = 0$ and $A_1 = 1$, A_2 will change to 1 next clock pulse.

If $A_2 = 1$ and $A_1 = 1$, A_2 will change to 0 next clock pulse.

If $A_3 = 0$ and $A_1 = A_2 = 1$, A_3 will change to 1 next clock pulse.

If $A_3 = 1$, it will change to 0 next clock pulse.

If $A_4 = 0$ and $A_3 = 1$, A_4 will change to 1 next clock pulse.

If $A_4 = 1$ and $A_3 = 1$, A_4 will change to 0 on next clock pulse.

In Boolean terms:

$$A_1 = A_1' A_3' C$$

$$0^{A_1} = A_1 C$$

$$A_2 = A_1 A_2' C$$

$$0^{A_2} = A_1 A_2 C$$

$$A_3 = A_1 A_2 A_3' C$$

$$0^{A_3} = A_3 C$$

$$A_4 = A_3 A_4' C$$

$$0^{A_4} = A_3 A_4 C$$

The method of selecting those terms for which a change will occur would not give these results after simplification unless a further rule is used; this specifies that a pulse may be added corresponding to a condition that the counter does not reach in normal operation, or suppress a pulse the counter does not reach in normal operation.

Since this counter has "forbidden states", it is necessary to check to be sure that if it takes on such a configuration when the machine starts, it will get back to its normal counting routine. This may be checked by using the counter equations to determine the next states of the counter for such conditions. For example, if this one started in

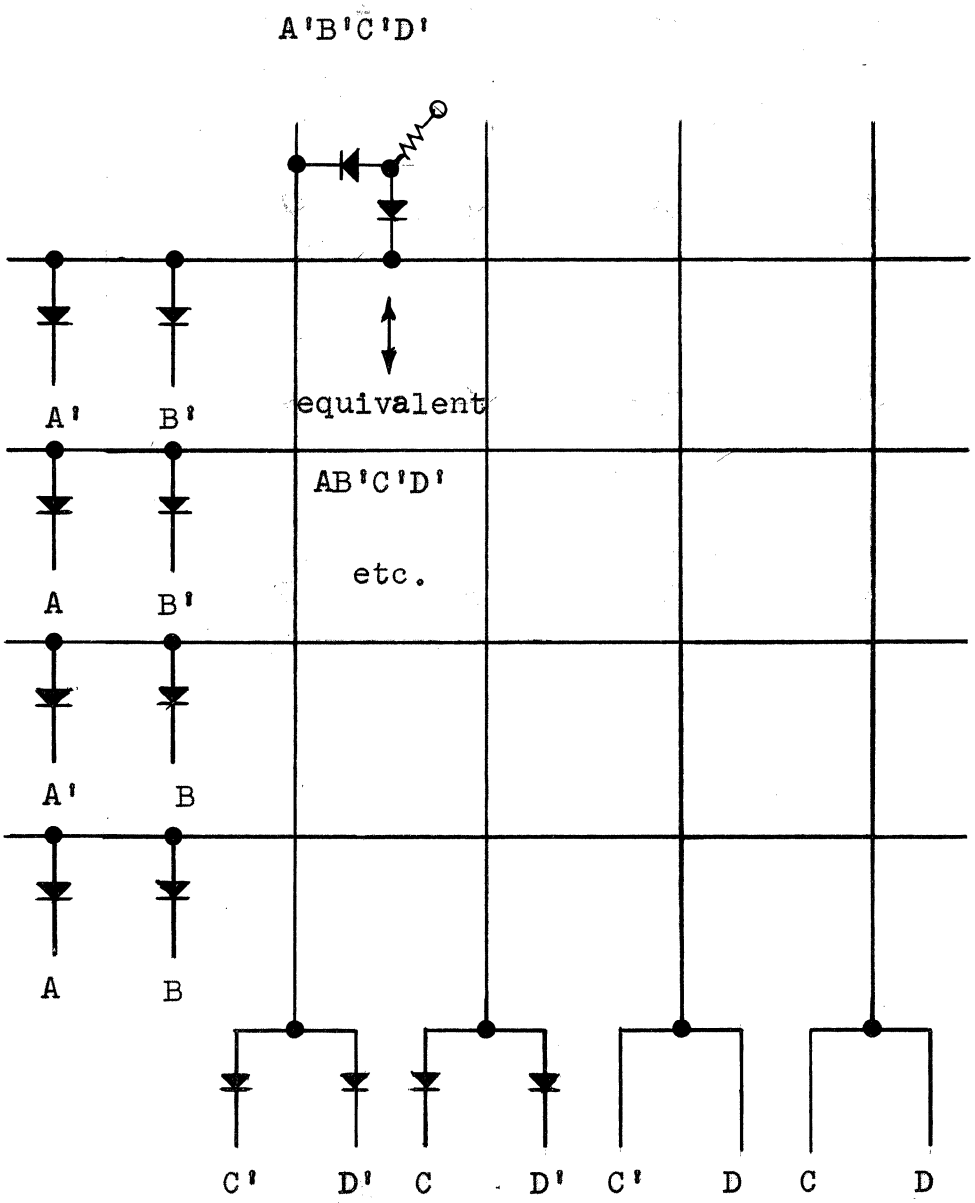
1110

the next states would be

0010
0011
0100
1000
1001
1010
1011
1100
0000
0001
0010
etc.

Thus, the counter would not recirculate between forbidden states if started in this position. In a like manner it can be shown that starting from any other "forbidden state" will soon give rise to normal operation.

It is possible to construct the network to evaluate
ABCK + ABC'D + AB'CD + A'BCD + A'B'CD + A'BC'D + AB'C'D + A'B'C'D + ABCD' + ABC'D' + ABC'D' + AB'CD' + A'BCD' + A'BCD' + A'B'CD' + A'BC'D + AB'C'D' + A'B'C'D by use of what is called a matrix. This is done as follows:



No resistors are necessary on A'B', etc. unless it is desired to use this proposition by itself.

Suppose that two numbers are added, one digit at a time. To do this with usual arithmetic operations, one adds the least significant digits, records the least significant digit of the sum, records most significant digit of the sum to be added to the sum of the next pair of digits, and continues. If this most significant digit be

denoted by K, the two numbers by E and F, and the sum by S, these could be written as follows:

K	1	0	0	0	0	1	0
E		3	1	4	1	5	9
F	7	.	3	0	1	0	3
S	0	3.	4	4	2	6	2

This example was done in decimal arithmetic. Note that the least significant carry (K) is always zero.

Let us do a similar problem in binary arithmetic:

K	1	1	1	0	0	0	0	0	1	0
E	X	1	0	1	0	1	1	0	0	1
F		1	1	1	1	0	0	0	0	1
S	1	1	0	0	1	1	1	0	1	0

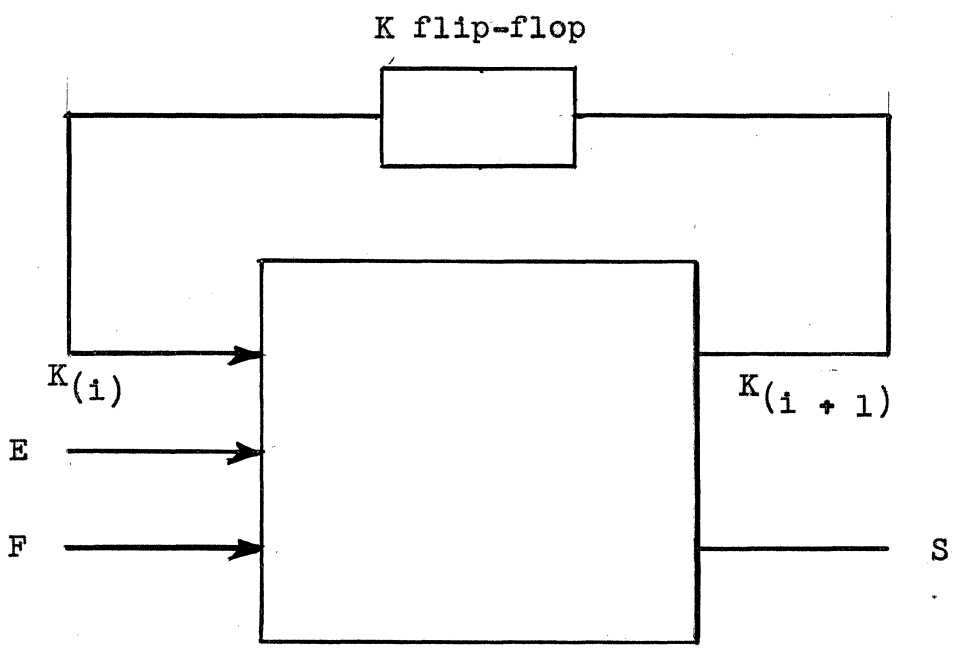
Since the order of digits in the machine is the least significant digit first, followed by successively more significant digits, we can probably do our operations on each set of digits as they pass through our machine.

If we make a table of all possible sets of digits, and determine the sums, and new carry K, we can find equations for both S and the K_1

E	F	K	S	K_1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Since we will want the carry K in the next step, it can be put into a flip-flop, and its output read, by use of a clock pulse, at the next digit time. We will need then

to know when the sum is different from zero, and how to set the K flip-flop.



From the table,

$S = E'F'K + E'FK' + EF'K' + E\,FK$

$k = E'FK + EF'K + EFK' + EFK$

$_0k = E'F'K' + E'FK' + E'F'K + EF'K'$

$S = K\ (EF + E'F') + K'\ (E'F + EF')$

$k = K\ (E'F + EF') + EF\ (K + K') = EF + K\ (E'F + EF')$

$_0k = E'F'K' + E'F'K + E'FK' + EF'K' = E'F'\ (K + K') + K'\ (E'F + EF')$

$= E'F' + K'\ (E'F + EF')$

Since the carry flip-flop will stay in one condition until changed, the second term of the simplified expression for k may be omitted because it says K is already set true. Thus $k = EF$; similarly $_0k = E'F'$

Suppose now that we want to add one to some number. If we let K be originally equal to one, we may substitute 0 for one of the addens. Suppose this one be F

Then, $S = KE' + K'E$

$k = 0$

$0^k = E'$

This is used in the CRC 102-A to increase the addresses in the H line by 1.

Consider the following problem:

Set a flip-flop A initially 0.

Change it to 1 if $E \neq F$ and leave it there.

Solution: If $E \neq F$ for at least one digit, for this digit either $E'F$ or EF' is true.

Then, the equations become:

$k = 0$

$0^k = (E'F + EF')$

Problem: to set a flip-flop 1 if $E > F$: If $E \neq F$, then the most significant digit where $E \neq F$ determines which is larger. If we set the flip-flop each time $E \neq F$, the final state of the flip-flop shows which number is larger, if it is set one way when E is larger and the other for F larger. Then,

$0^k = E'F$ $\quad (E = 0, F = 1$ i.e. $E < F)$

$1^k = EF'$ $\quad (E = 1 \quad F = 0 \quad E > F)$

However, if $E = F$, the condition of the flip-flop is not changed, so if $K = 0$ before the start of the word, the condition of K will be

$K = 1, \quad E > F$

$K = 0 \quad E < F$

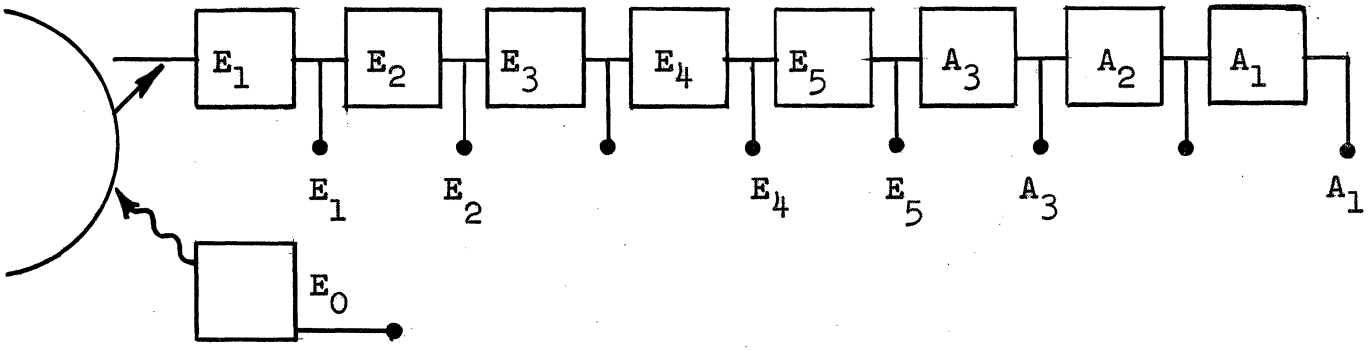
STORAGE SYSTEMS

In the CRC 102-A memory there are three types of storage. One system, permanent storage, occupies most of the magnetic drum. Information stored in this part of the machine will stay in it until it is replaced.

The other types of storage are "volatile". One, the buffer register, holds 8 words and recirculates these so that one of the 8 is always available.

Skillful use of this register will shorten considerably the look up time, which consumes a major part of the computer working time in any problem. The second volatile system is a set of 4 one word cells, E, F, G and H. E is used in input, output, and arithmetic operations to hold the numbers as they are being used. F is used in arithmetic operations. G is shared between control functions and arithmetic operations and control functions, and is "3000" for read out only. H is the control cell, and, at various times, contains a command, the address of the next command, or the address to which a word in the E register will go in the permanent storage or buffer

The "E line" may be represented as follows:

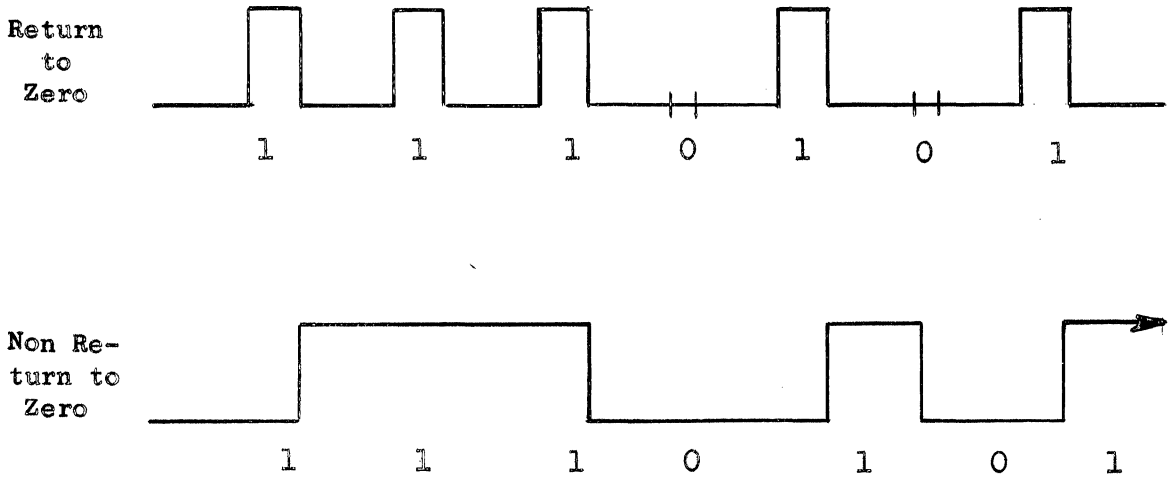


Each E except E_0 is a flip-flop driven from the previous flip-flop or from the shaping circuits from the read out head. The grid equations for E_2 -E_5 are in the form:

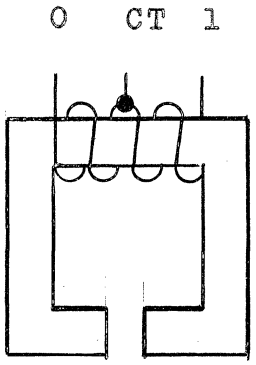
$$e_i = E_{i-1}C$$

$$_oe_i = E'_{i-1}C$$

To see what happens at the read head, we must consider the recording system in the CRC 102-A, the so-called non-return-to-zero system. In this system, the write head does not return to zero polarity after each bit is recorded, but only when the next digit is different from the preceding one:

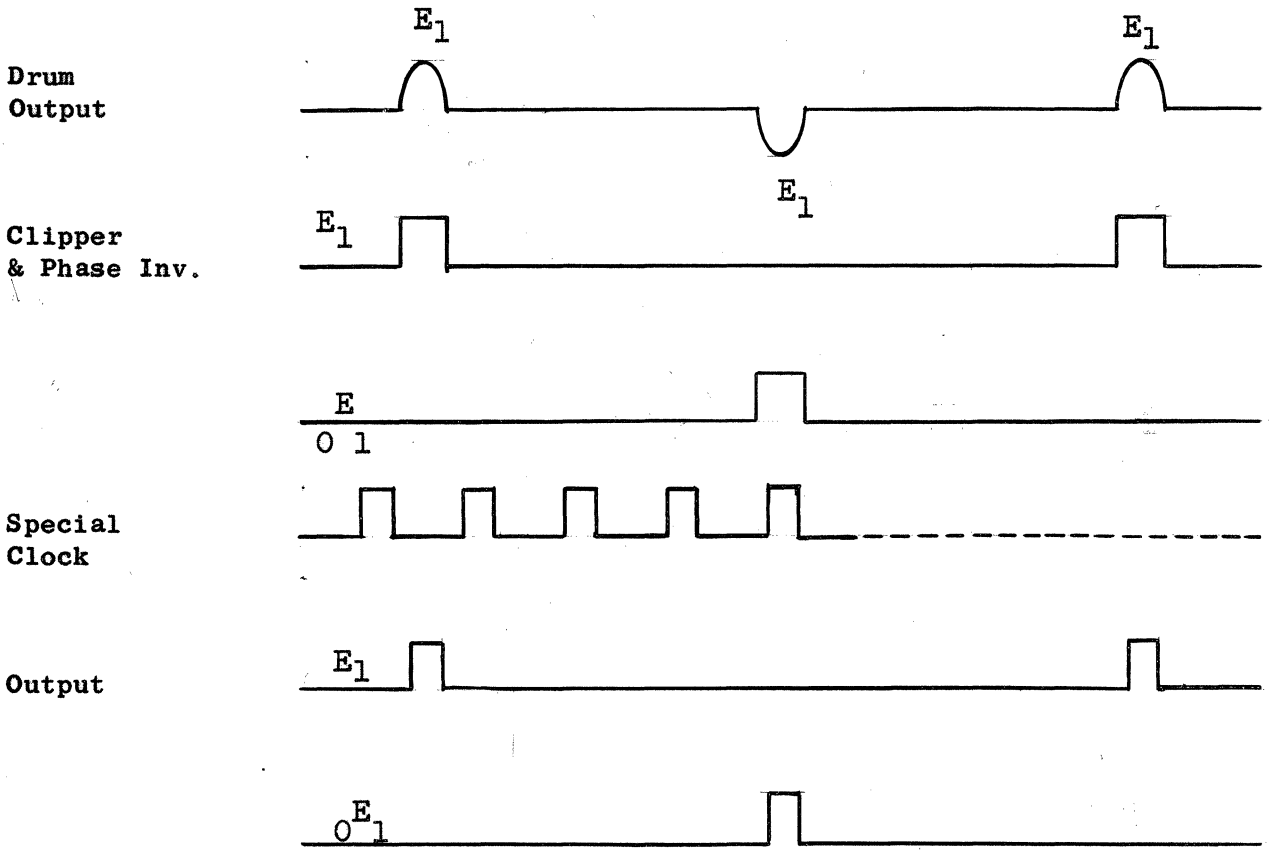


Since the above waveforms represent idealized writing currents, the waveforms also represent the direction of magnetization on the drum. This may be seen from the diagram of the write head:



Energizing the "1", half of the winding will orient the magnetic N-S poles in the drum one way, energizing the 0, half of the winding will orient the magnetic N-S poles the other way. The read head will have 0 output except where the flux changes, and there

it will give a pulse of one polarity if the change is plus and a pulse of the opposite polarity if the change is minus. This is used to gate a very sharp clock pulse, to change the state of E_1 as required.



In practice, the spacing between the read and write heads is adjusted for the coincidence of the output signal with the special clock pulse. In the recirculating lines, an erase head is used before each write head, which sets the magnetic material to "zero" polarity, making the 0 winding on the write head unnecessary ("single ended writing"). There are no erase heads in the main storage, so the double ended system is necessary there to record 0 where there was a 1.

Returning to the discussion of the E line, it will be noticed that several flip-flops are associated with it. At least one must be used to drive the record head, but the others are used in switching the line to a longer or shorter length, which will cause shifts when properly used.

In the machine the magnetic line and associated circuits hold 37 bits, and E_1 -E_5 hold the remaining 5 bits. For normal recirculation, the circuits are set up so that $E_5 = E_0$. If one of the A flip-flops is connected to E_5 and E_0 is read off of this flip-flop, i.e. $E_0 = A_3$, the word is then 43 bits long, but only 42 clock pulses occur in any word time, so each bit advances 42/43 of a complete circuit, which is equivalent to 1 binary shift toward the most significant end. Notice that what was in A_3 now is in E_5, and what was the most significant bit is now A_3. If A_3 is set to zero after each word time, we would have, in effect, shift logical in a positive direction.

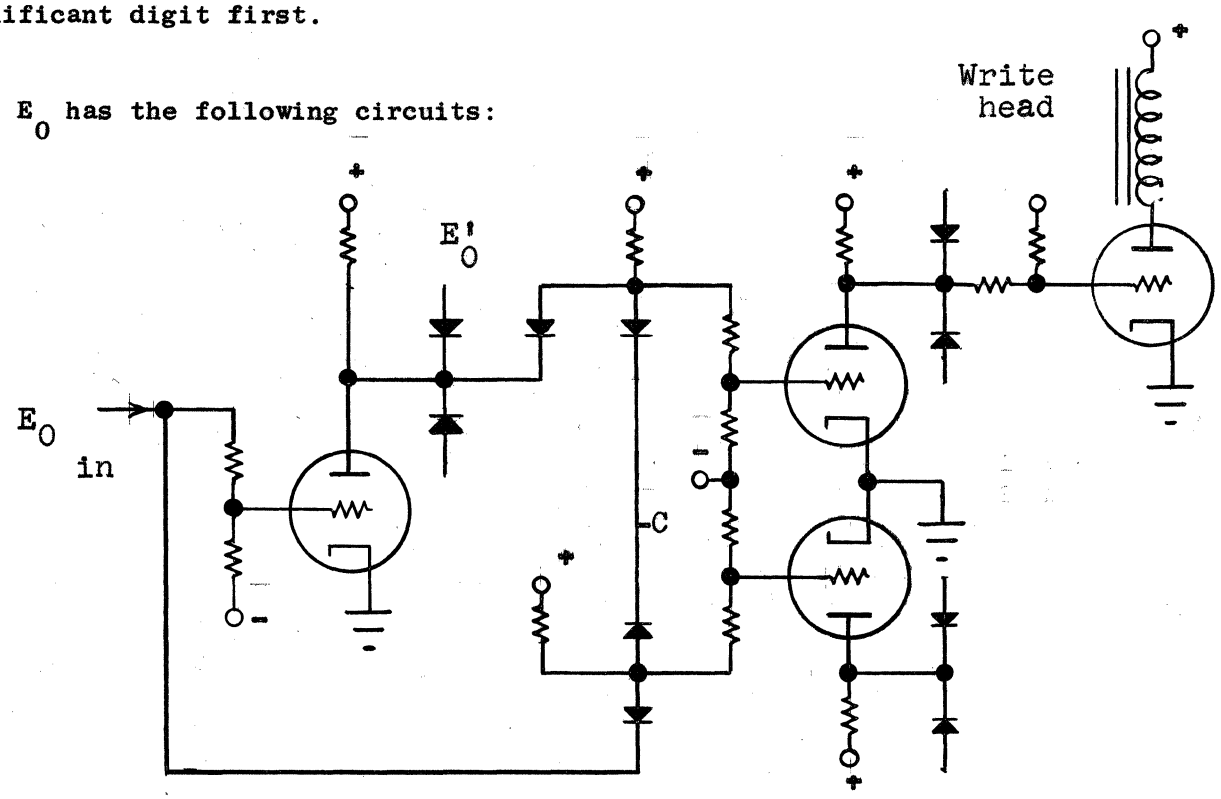
If on the other hand we make $E_4 = E_0$, the line is now 41 bits long, and will advance 42/41 binary places in one word time. Since the output is no longer taken from E_5, the information in it will be read out and lost unless some precaution is taken to retain it elsewhere in the machine.

Now, since each bit advances one bit further than the word length in one word time, this corresponds to a shift toward the least significant end of one binary digit. However, that information which was in E_4 will, after one word time, be the most significant digit as well as the contents of E_5, unless precautions are taken to prevent this from happening by forcing $E_0 = 0$ for this time. If this is done, we have a shift logical in the negative direction.

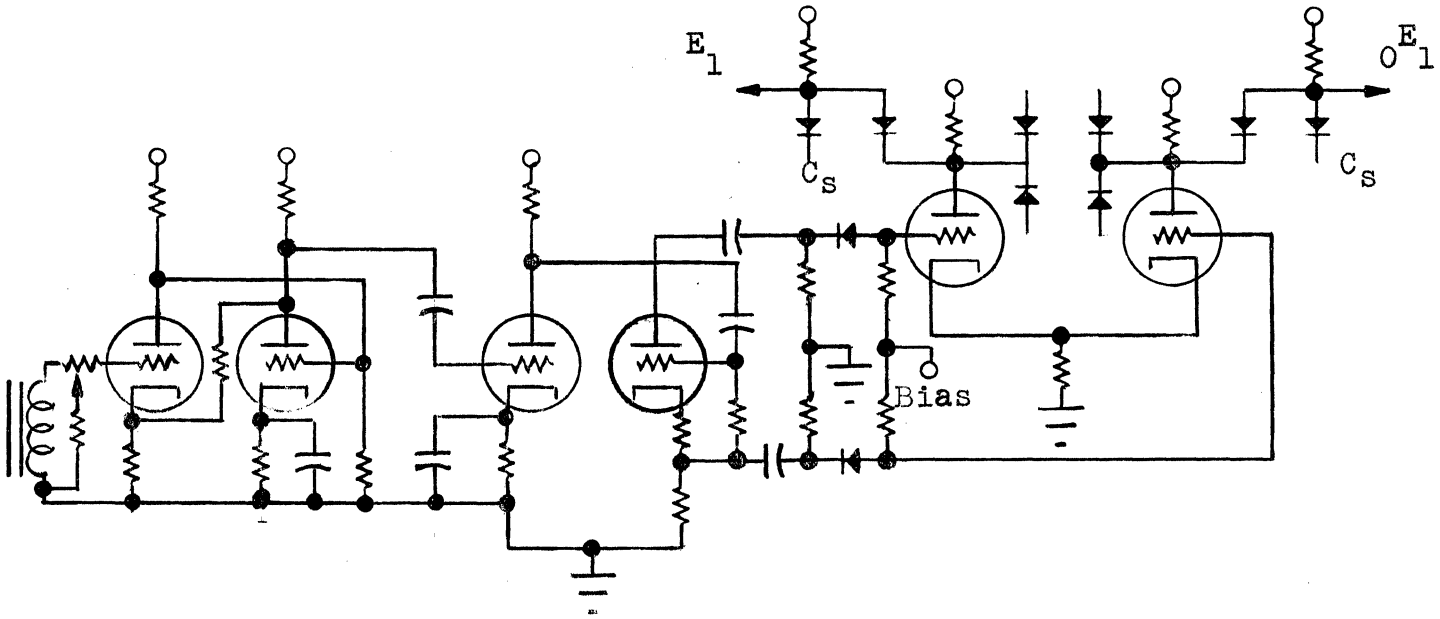
														E_1	E_2	E_3	E_4	E_5	A
1	longer	110	011	100	011	110	000	111	110	000	011	111	100	000		011			0
Original		011	001	110	001	111	000	011	111	000	001	111	110	000		001			1
1	shorter	001	100	111	000	111	100	001	111	100	000	111	111	000		000			

To interpret the above diagram, consider the line to be a chain of flip-flops, each of which contains one bit. If this line were stopped after the last clock pulse of a word, the contents of each flip-flop would be the digit corresponding to the significance of each flip-flop in the word. If the line is lengthened or shortened by more than one flip-flop, a corresponding number of shifts per word time of configuration will result. In filling the computer, 3(or 4) flip-flops (octal (or decimal)) are added to the E line for one word time. If those are preset to correspond to an octal (or decimal) digit, this will effectively store one digit in the least significant place in E. In a similar fashion one may read out of the E line, taking the most significant digit first.

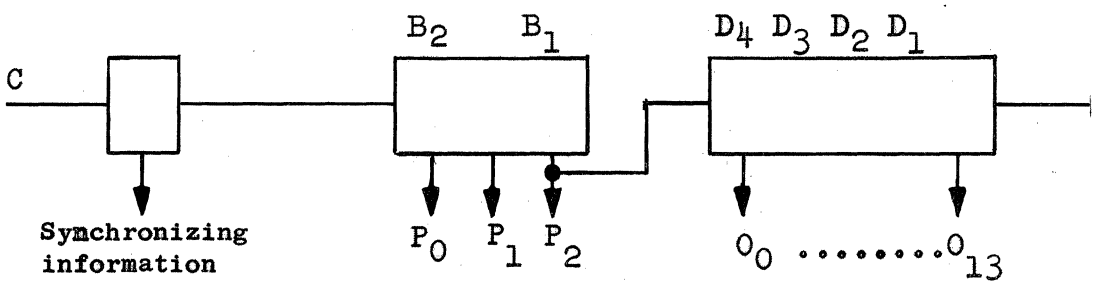
E_0 has the following circuits:



The read head has the following circuits associated with it:



It is desirable to have a counter synchronized with the word so that any part of the word may be examined separately, and we will know where to find it. To do this we will divide the 42 bits of a word into 14 octal digits, each containing 3 bits. We, thus, need a scale of 3 counters followed by a scale of 14:



the synchronizing information may be derived as follows: Each word in the word channel has the following structure:

00 00XX 00XX 00XX

If we form a function $Q = O_2 + O_3 + O_6 + O_7 + O_{10} + O_{11} + O_{12} + O_{13}$

Then $Qw_{ch} = 0$ when the counter is synchronized. Every time $Qw_{ch} = 1$ let us inhibit the clock signal to the word counter for one clock pulse. The counter will thus shift until it agrees with the word channel.

The logical network is controlled by a nine stage counter $N_1 \ldots\ldots N_9$. This counter may either count to the next higher number or skip to a completely different number. A special case of skipping, is to skip to the same number (i.e. no change). This counter is called the Program Counter. The Program Counter is controlled by the state of the K flip-flop at $P_2 O_{13}$.

$K P_2 O_{13} = 1$ skip

$K P_2 O_{13} = 0$ count

The count terms for the Program Counter flip-flop grids are standard binary counter terms. The skip terms are chosen especially for the occasion, one for each grid that must be changed on each skip.

Matrices are used on the Program Counter plates so that the Program Counter states may be derived with a minimum of diodes.

E and G have 5 external flip-flops.

F, J, H have 2 external flip-flops.

There are 12 general purpose A flip-flops, A_1..... A_{12}, a special purpose A_{13}, and K, which has a very large number of uses.

For recirculation:

$E_0 = E_5$

$G_0 = G_5$

$F_0 = F_2$

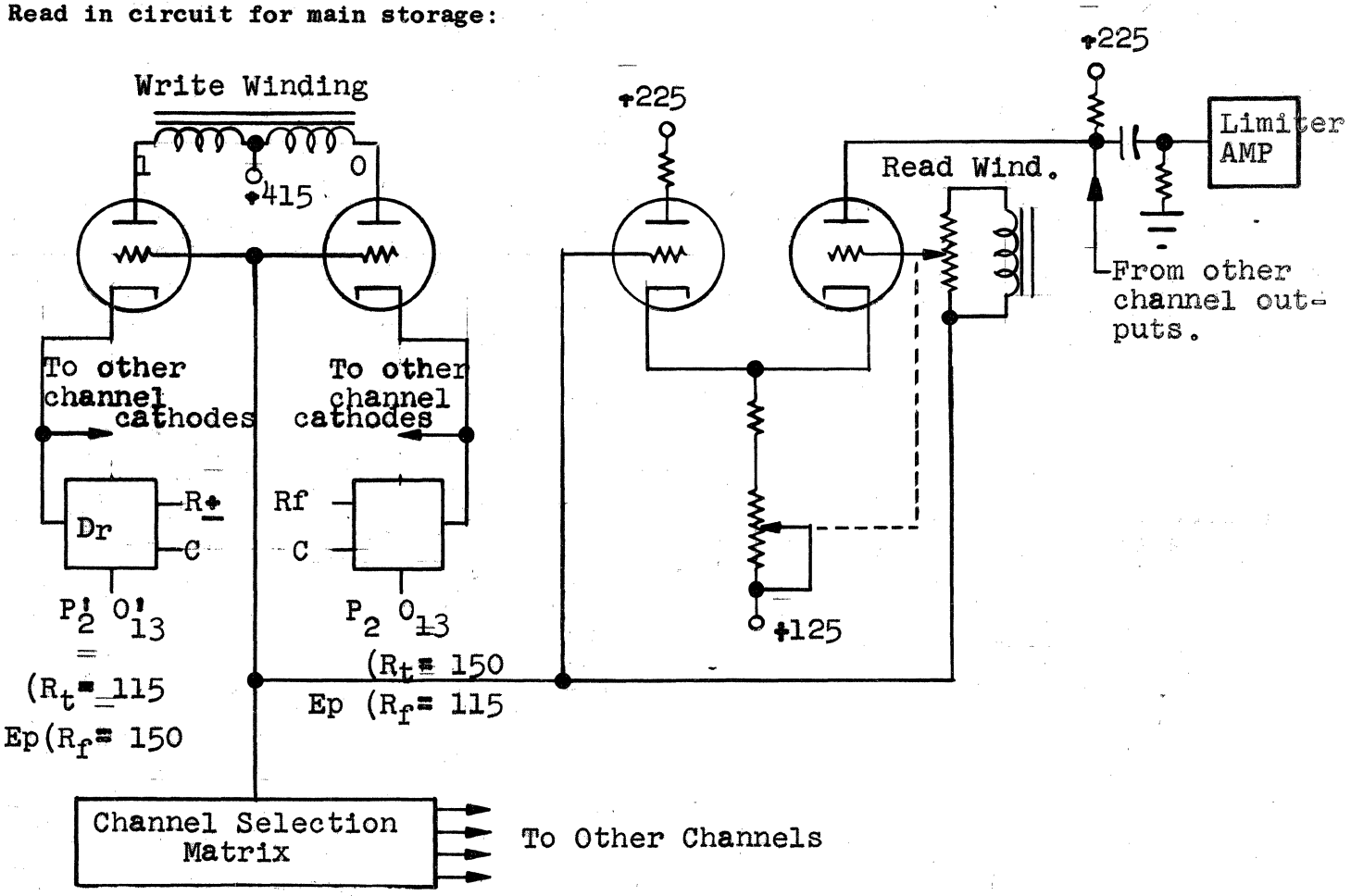
$J_0 = J_2$

$H_0 = H_2$

The following Flexowriter keys have these codes:

	A_5	A_4	A_3	A_2	A_1
0	0	0	0	0	0
1	0	0	0	0	1
.					
.					
.					
.					
9	0	1	0	0	1
.	0	1	1	1	1
Sp	0	1	1	0	0
f	1	0	0	0	0
d	1	1	0	0	1
0	1	1	0	0	0
-	1	1	1	0	0
Tab	1	1	1	1	0
s	1	1	1	1	1

Read in circuit for main storage:



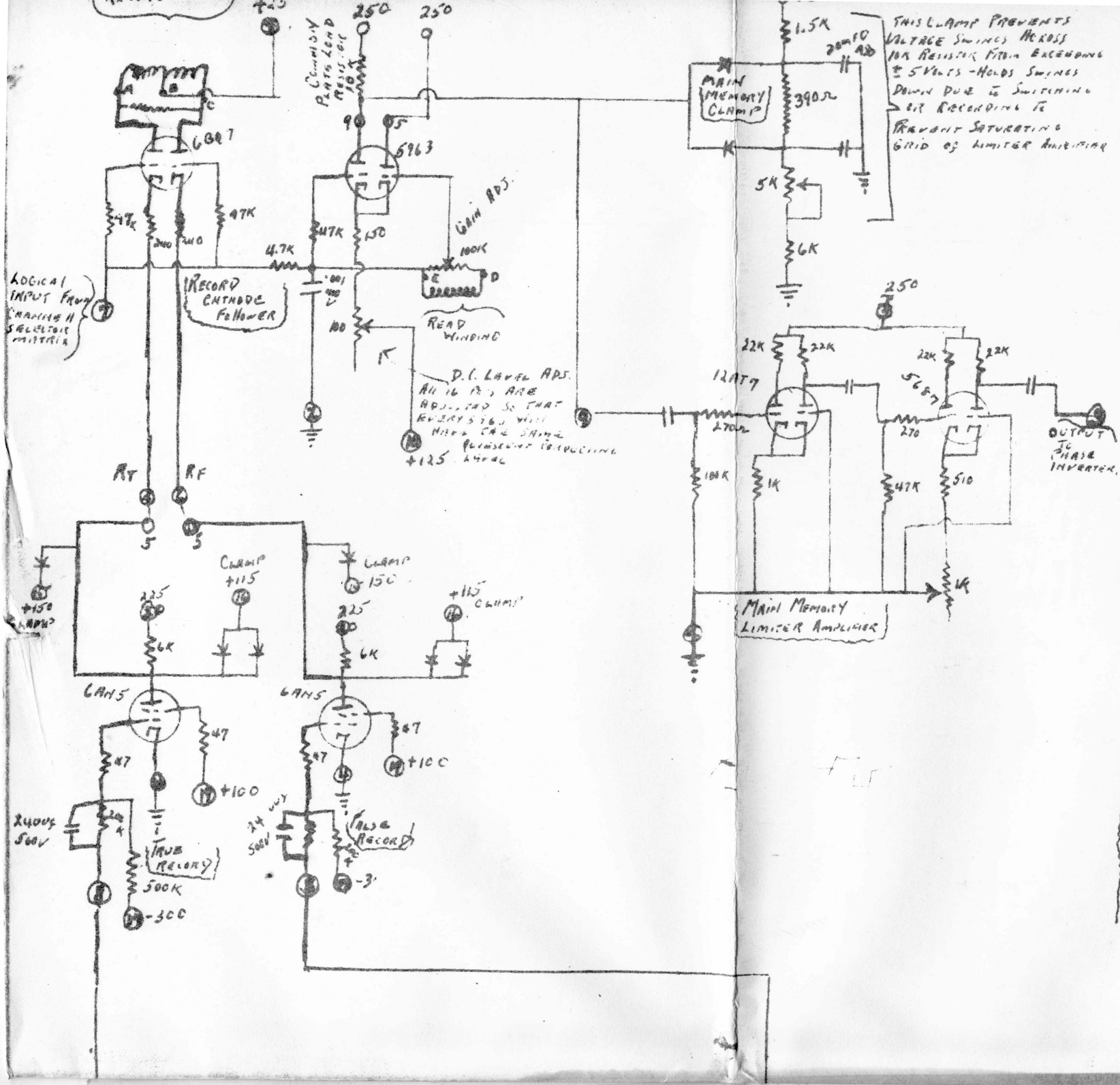
Record Equations:

$$R_T = E_5 R_1 + J_2 R_2 \qquad R_1 R_2 = 0$$

$$R_F = E'_5 R_1 + J'_2 R_2 \qquad \text{Read from } E_5$$

or

Buffer into Main Storage



Main Memory Recording

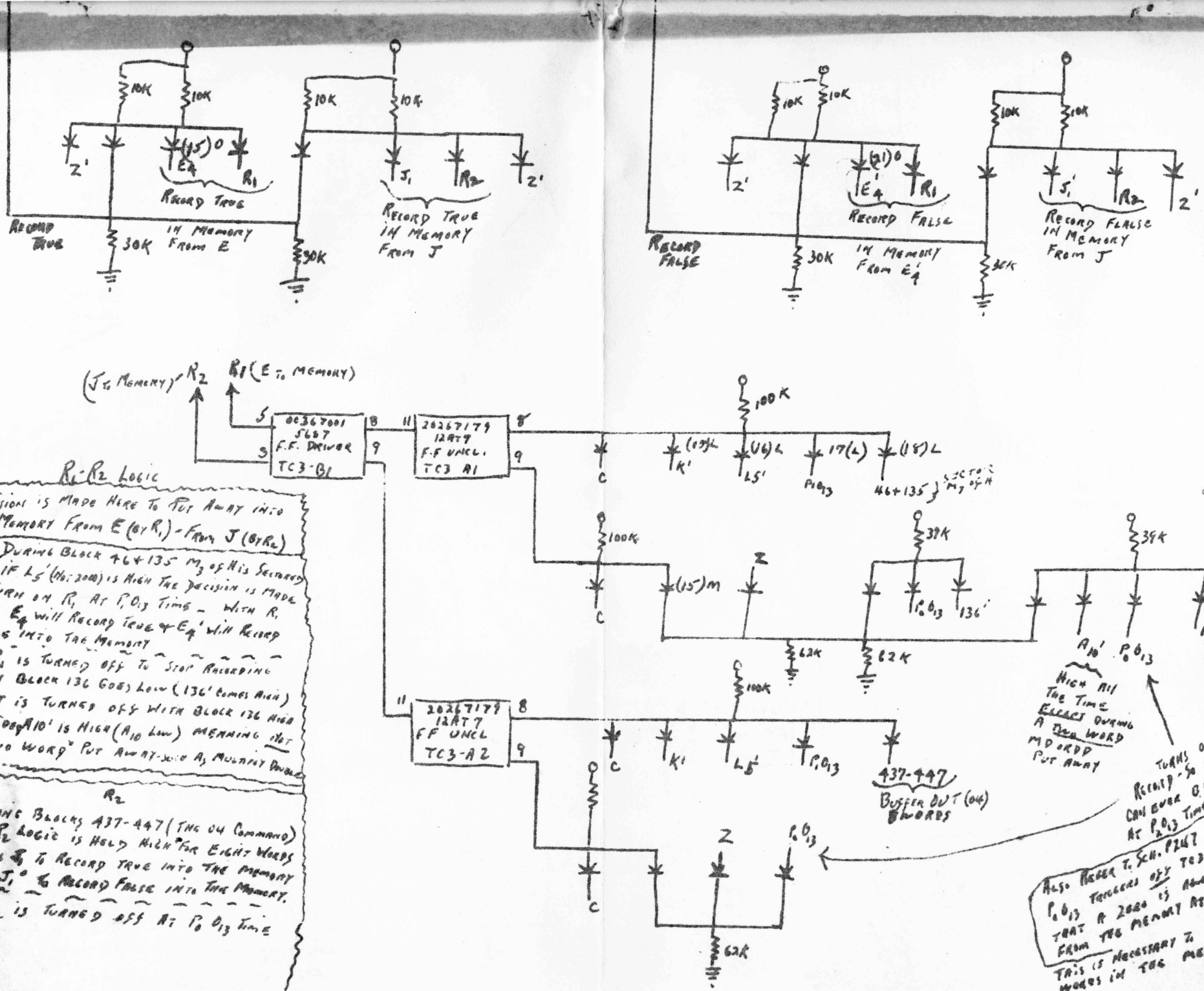
With N_2 Setting The L is - The L Matrix Selects A Channel To Bring Pin 7 (or 8) High On One Of The Record Cathode Followers - This Brings The Grids Of The 6BQ7 And 5963 To +125, The Cathodes Of The 6BQ7 Are Held At +150 Volts By The True And False Gain Record Clamps

In Record True Or Record False It Is Desired The Selected Gain Grids Is Brought To +125 - Causing This Tube To Conduct So That Its Plate Drops To The 115 Clamp Level - This Brings The Appropriate Cathode Of The 6BQ7 Down To 115 Permitting This Half To Conduct And Sending Current Thru One Half The Record Winding Recording A One Or A Zero.

Reading

Channel Selection Occurs As Above Grids Of 6BQ7 And 5963 Come To +125 - Neither True Or False Record Is Being Called For & Cathodes Of 6BQ7 Remain At +150 (Non Conducting)

The Cath. Of 5963 Is Biased At +125 - Its Grids Coming To +125 Partially Bring It Into The Conducting State - The Small Signal From The Read Winding Is Cathode Coupled To The Left Half Of The Tube - The Signal Is Amplified And Decoupled Across The Common 10K 12AT7 Load Resistor (A Clamp Holds It To ± 5) And Couples It To The Main Memory Limiter 12AT7 Grid - A Pos. Signal Here Drives The Right Half To Cut Off (Common Cathode Rises This) And A Neg. Signal On Left Half Cuts Off This Half. The Signal Is Amplified By The 5687 And Its Cathode Bias Is Adj. Until The Point Where No Limiting Action Is Observed. The Output Here Is Applied To The Phase Inverter Grid



R1-R2 Logic

DECISION IS MADE HERE TO PUT AWAY INTO THE MEMORY FROM E (BY R_1) - FROM J (BY R_2)

DURING BLOCK 46+135 M_3 OF H'S SECTORS AND IF L5' (NO. 2000) IS HIGH THE DECISION IS MADE TO TURN ON R_1 AT $P_1 0_{13}$ TIME - WITH R_1 HIGH E_4 WILL RECORD TRUE & E_4' WILL RECORD FALSE INTO THE MEMORY

R_1 IS TURNED OFF TO STOP RECORDING WHEN BLOCK 136 GOES LOW (136' COMES HIGH) OR IT IS TURNED OFF WITH BLOCK 136 HIGH PROVIDED A10' IS HIGH (A_{10} LOW) MEANING NOT A "TWO WORD" PUT AWAY-SUCH AS MULTIPLY PRODUCT

R_2

DURING BLOCKS 437-447 (THE 04 COMMAND) THE R_2 LOGIC IS HELD HIGH FOR EIGHT WORDS CAUSING J_1 TO RECORD TRUE INTO THE MEMORY AND J_1' TO RECORD FALSE INTO THE MEMORY.

R_2 IS TURNED OFF AT $P_0 0_{13}$ TIME