order paint
order doors

THE LINCOLN TX-2 COMPUTER

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF
TECHNOLOGY

Program register
multiplier r
Accumulator reg
reg
mem
CAR
Instruction reg

PRESENTED AT:
THE WESTERN JOINT COMPUTER CONFERENCE
LOS ANGELES FEBRUARY 1957

THE RESEARCH REPORTED IN THIS DOCUMENT WAS SUPPORTED JOINTLY BY THE ARMY, NAVY, AND AIR FORCE UNDER CONTRACT WITH THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY

THE LINCOLN TX-2 COMPUTER DEVELOPMENT*

Wesley A. Clark**
MIT Lincoln Laboratory
Lexington 73, Massachusetts

Introduction

The TX-2 is the newest member of a growing family of experimental computers designed and constructed at the Lincoln Laboratory of MIT as part of the Lincoln program for the study and devel opment of large-scale digital computer systems suitable for control in real time. Although in general characteristics and design philosophy it owes a great deal to its predecessors, Whirlwind I and the Memory Test Computer, the Lincoln TX-2 incorporates several new developments in components and circuits, memories, and logical organization. It is the purpose of this paper to summarize these new features and to give some idea of the historical development and general design objectives of the TX-2 program.

History

With the development by Lincoln and IBM engineers of the SAGE computer for air defense, real-time control computer systems had reached an impressive level of size, sophistication, and complexity. The highly successful 64 x 64 coincident-current magnetic core memory array was in operation in the Memory Test Computer which had given up its earlier 32 x 32 array to Whirlwind. Vacuum tubes abounded in all directions. It was apparent that

*The research reported in this document was supported jointly by the Army, Navy, and Air Force under contract with the Massachusetts Institute of Technology.

**Staff Member, Lincoln Laboratory, Massachusetts Institute of Technology.

the further advances in system design which could be made by increasing memory size, eliminating vacuum tubes wherever possible, and organizing input-output buffering, control, and communications into more efficient forms, would be well worthwhile.

The development of a 256 x 256 switch-driven magnetic core array was begun and the Philco surface-barrier transistor made its appearance. After some very promising bench experiments with flip-flops and logic circuits, it became apparent that this transistor was potentially well-suited to use in large-scale systems and warranted further study. Accordingly, plans were laid for a succession of experimental digital systems of increasing size and complexity which would make possible the development and evaluation of circuits using the surface-barrier transistors, and which would lead to a computer of advanced design capable of making efficient use of the 256 x 256 memory

A double-rank shift register of 8 stages containing about 100 transistors was constructed and put on life-test in April 1955. It has since been circulating a fixed pattern almost continuously with no known errors and no natural transistor failures.

As the next step, it was decided to build a small high-speed error-detecting multiplier and incorporate marginal checking and other system features. The value of a multiplier as a preliminary model had been well demonstrated by the 5-digit system built during Whirlwind's early development. The shift, carry, count, and complement operations, under closely controlled timing conditions, were felt to be representative of all of the operations in the manipulative elements of the type of computer planned. Accordingly, an 8-bit system using 600 transistors was designed and completed in August 1955 and has been in nearly continuous operation since that time. Operating margins are periodically checked, and in steady state operation, the multiplier's error-rate has been about one every two months or one error per 5×10^{11} multiplications at 10^5 multiplications per second. Most of these errors appear to have been caused by cracks in the printed wiring which open intermittently.

During this period, a better idea of the general characteristics of the projected computer began to develop and the engineers designing the 256 x 256 memory were encouraged to think in terms of a 36-bit word length. The notion of a logically separate input-output processor was examined and rejected in favor of a minimum buffering scheme in which data is transferred directly to and from the central memory of the computer. The possibility of programming these transfers by means of additional program sequences and associated program counters, thus taking advantage of the extensive facilities of the central machine itself for processing input-output data, was recognized.

It was realized that another development step was desirable before attempting such an elaborate 36-bit system. The 8-bit multiplier had produced a certain measure of confidence and familiarity with circuits, packaging, and techniques of logical design, but there remained the problems associated with communicating with memory units and input-output equipment operating at vacuum-tube levels over relatively large distances from a central machine which operated at transistor levels. It appeared that the memory development, which had now entered the construction phase, would also benefit by a preliminary evaluation of the 256 x 256 array and its switching, timing, and noise problems in an operating computer of some kind, possibly with a reduced word length. It was, therefore, decided to design and built next a simple machine - in fact, the simplest reasonable machine - in order to bring about an early intermediate closure of the various efforts within the program.

After some thought about the various possible minimal machines, a design was completed in which the word length would be 18 bits - a graceful half of the projected final form. We began to refer to this computer as the TX-0 and to the projected machine as the TX-2. Because the 256 x 256 memory array required 16 bits for complete addressing, the single-address instruction word of the TX-0 was left with 2 bits in which to encode instructions. The particular set of instructions chosen included three which required a memory address (add, store, and conditional jump) and one which did not. In this last instruction, the remaining 16 bits were used to control certain necessary and useful primitive operations such as clearing and complementing the accumulator, transferring words between registers, and turning on and off input-output equipment.

The TX-0, equipped with a Flexowriter, a paper-tape reader, and a cathode-ray tube display system was completed, except for the memory, in April 1956. Twenty planes of the 256 x 256 memory array were installed the following August and the TX-0, now containing about 3600 transistors and 400 vacuum tubes, began to function as a complete computer. Since that time, it has been used to run a variety of testing and demonstration programs, and a symbolic address compiler and other utility programs have been constructed and are currently in use.

Not only has the tx-0 served the evaluational purposes for which it was built, but it has also demonstrated an effectiveness as a usable computer that is somewhat surprising in view of its simplicity. Its relatively high speed of about 80,000 instructions per second and its 65,536-word memory compensate in large measure for the limitations of its instruction code and logical structure.

With the successful completion of the TX-0, the final steps in the development were undertaken in packaging, circuit refinement, and logical design of the TX-2. A great deal had been learned about the performance of the transistors and memory, the types of logical circuits which are practical, techniques of marginal checking, and the lesser system problems such as color scheme selection and the proper location of pencil sharpeners. As design work progressed, the TX-2 took form as a system of about 22,000 transistors and 600 vacuum tubes. It is an interesting fact that at each step of the development since the shift register, the number of transistors involved was about 6 times the number in the preceding step. This is graphically shown in the accompanying figure. At the time of writing, approximately 16 million transistor-hours have accumulated in the shift register, multiplier, and TX-0. There have been two natural deaths and a dozen or so violent ones, primarily due to contact shorting with clipleads and probes.

Design Objectives

In describing design objectives, it should be pointed out that speed of operation was <u>not</u> the primary consideration to which all other attributes were sacrificed. It would have been possible, at the expense of a few more logic circuits, to increase the speed of multiplication, division, and shift-type operations. Similarly, the operation of the index register system could have been made more efficient at the cost of an additional small, fast memory. The principal objective was rather that of achieving a balance between the factors of speed, reliability, simplicity, flexibility and general virtue.

A key aspect is that of expandability which, in an experimental computer in an active environment, certainly ranks with the foregoing qualities in importance. The address structure in the TX-2 permits an expansion of the memory by about a factor of 4, partly to allow for new memory developments such as the transistor-driven 64 x 64 array which was begun following the completion of TX-0. New instructions and pieces of terminal equipment will certainly be added during the course of future operation. Extra space and spare plugs have been artfully distributed about in constructing the computer frame. Finally, modular construction will permit a fairly easy physical expansion when required.

The result of all this activity has been a computer of relatively large capability. In addition to incorporating high-speed transistor circuits and a large magnetic-core array, the Lincoln TX-2 has two major and distinguishing design characteristics:

1. The structure of the arithmetic element can be altered under program control. Each instruction specifies a particular form of machine in which to operate, ranging from a full 36-bit computer to four 9-bit computers with many variations. Not only is such a scheme able to make more efficient use of the memory in storing data of various word lengths, but it also can be expected to result in greater over-all machine speed because of the increased

parallelism of operation.

Peak operating rates must then be referred to particular configurations. For addition and multiplication, these peak rates are given in the following table:

PEAK OPERATING SPEEDS OF TX-2

Word Lengths (in bits)	Additions per second	Multiplications per second
36	150,000	80,000
18	300,000	240,000
9	600,000	600,000

2. Instead of one instruction counter, the TX-2 has 32 such counters which are assigned separately to different users of the computer, who then compete for operating time from instruction to instruction. A special part of the machine selects a particular user based partly on a predetermined priority schedule and partly on the current needs of that user. This multiple-sequence operation, in which many essentially independent instruction sequences interrupt and interleave one another, is an extension of the breakpoint operation found in DYSEAC of the National Bureau of Standards.

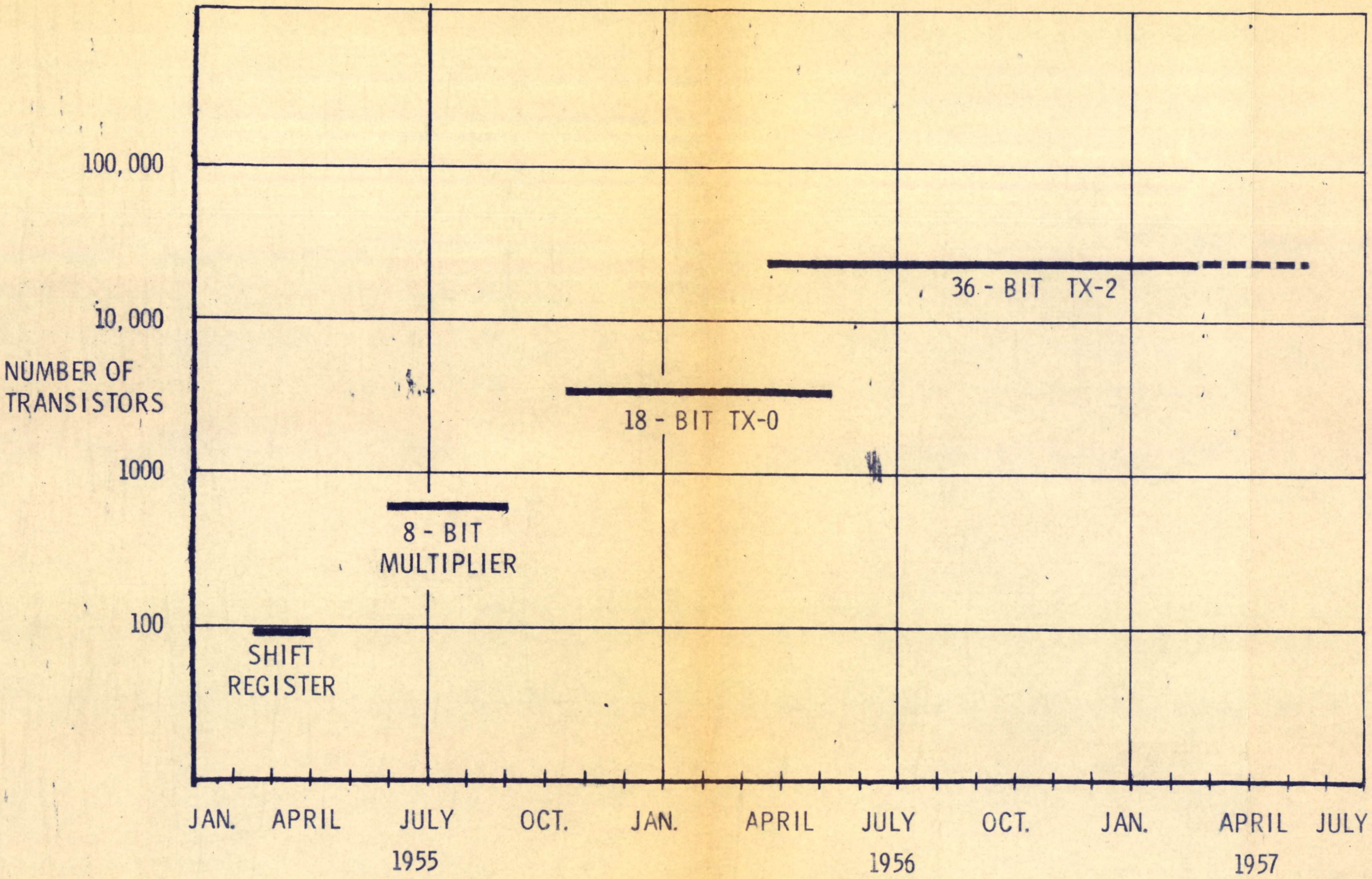
The value of these features will have to be assessed during the course of future machine operation. The features themselves are discussed in more detail in papers which follow.

LIST OF FIGURE CAPTIONS

Figure 1 Steps in the Development of the Lincoln TX-2

Figure 2 The Lincoln TX-0 and TX-2 Computers - Foreground, TX-0 console; middle center, TX-0 central computer frame; right rear, partially completed TX-2 frame showing plug-in unit construction; left rear, the 256 x 256 memory

B-69366
S-1621
C60-378



STEPS IN THE LINCOLN TX-2 DEVELOPMENT PROGRAM

Fig. 1

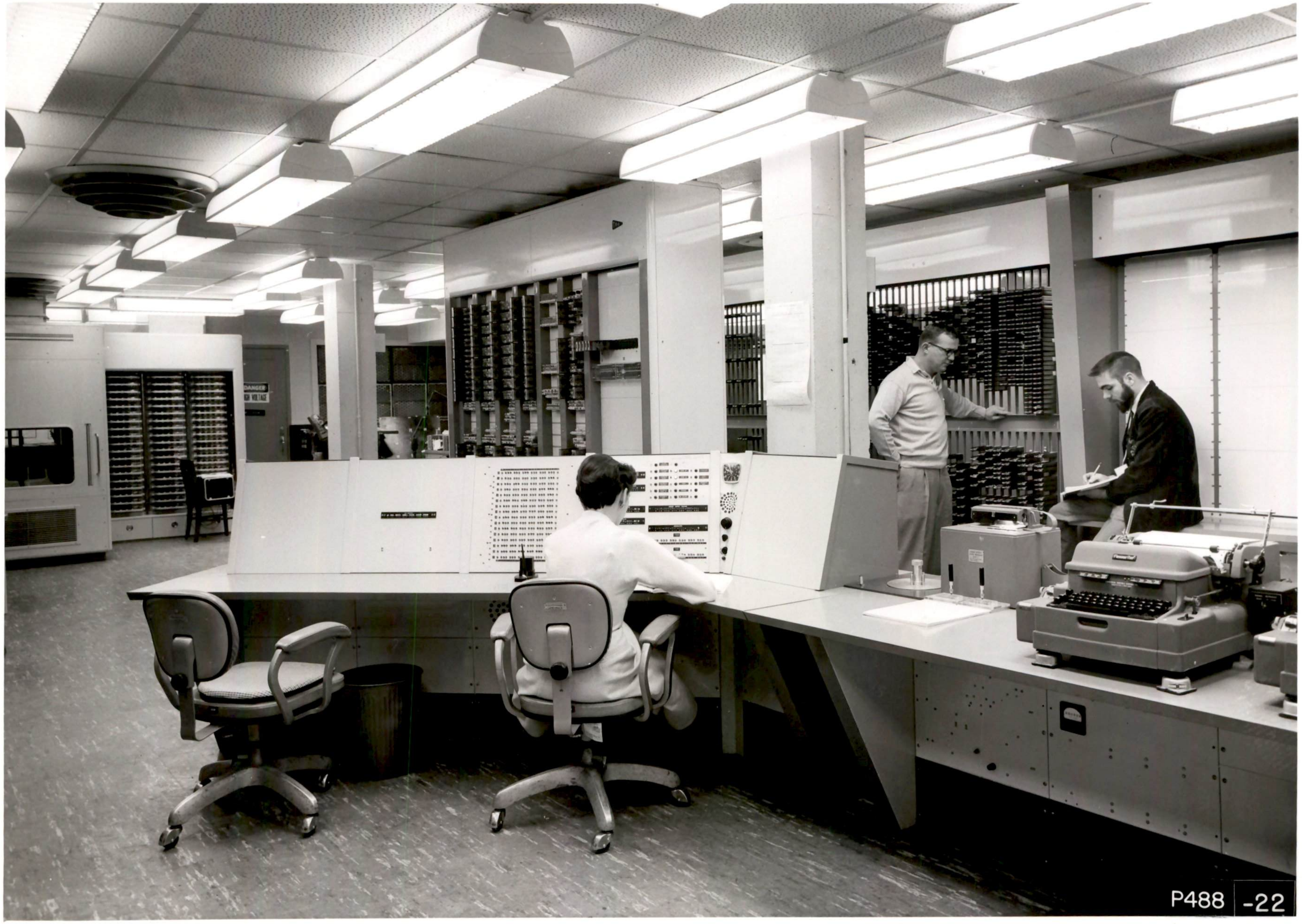


fig 2

A FUNCTIONAL DESCRIPTION OF THE LINCOLN TX-2 COMPUTER *

John M. Frankovich and H. Philip Peterson **
MIT Lincoln Laboratory
Lexington 73, Massachusetts

Introduction

TX-2 is a large scale digital computer designed and built at the MIT Lincoln Laboratory utilizing new memory and circuit components and some new logical design concepts. The computer will be applied as a research tool in scientific computations, and in data-handling and real-time problems. The design of the computer reflects not only the characteristics of the components available, but also the nature of the intended applications. This paper explains the functional and organizational aspects of the computer which are important from the user's point of view.

General Structure of TX-2

TX-2 is a parallel binary computer with a 36-digit word length. The internal memory is all random-access and will initially consist of 69,632 registers of parity checked magnetic-core memory and about 24 additional toggle switch and flip-flop registers. About 150,000 instructions can be executed per second. Instructions are of the indexed single-address type and a fixed point, signed fraction, one's complement number system is used.

Several unusual ideas incorporated in the system organization reduce the amount of information unnecessarily manipulated during program sequences.

* The research in this document was supported jointly by the Army, Navy, and Air Force under contract with the Massachusetts Institute of Technology.

** Staff Members, Lincoln Laboratory, Massachusetts Institute of Technology

Furthermore, the system organization facilitates the execution of several operations simultaneously, thereby increasing the effective speed of the computer.

The principle registers and information paths in the computer are illustrated schematically in Fig. 1. A,B,C,D,E,F,M and N are the 36-bit flip-flop registers in the machine. M and N are memory buffer registers, each of which has a parity flip-flop and associated circuitry which is used to check the parity of memory words. P,Q, and X are 18-digit registers; X also has a parity digit which is used to check the parity of words in the X memory. Control flip-flops are not shown in Fig. 1.

Instructions are full memory words and are placed in the Control Element during the instruction memory cycle. During the operand memory cycle, an operand is usually transmitted between the Memory Element and some other element--always through the Exchange Element. The 36-digit configuration of the memory is not, however, maintained throughout the computer during operation timing. A programmer can, in effect, control several independent, shorter operand word length computers simultaneously during the execution of each instruction. This flexibility is realized by specifying a particular system configuration with each instruction.

The computer communicates with the outside world through units in the In-Out Elements, several of which can be simultaneously operated. Whenever an input or output information transfer can occur, signals to the

Program Element from the In-Out Element automatically call into operation the associated instruction sequence. This multiple-sequencing aspect of the computer will not be described in this paper.[1]

Memory Element

The availability of a large, fast, core memory for TX-2 permitted an emphasis on the design of a machine with an all random-access memory which could be as large as 262,144 words. The homogeneous aspect of so large a memory system simplifies the programmer's coding problems and permits continued high-speed operation regardless of the program location in the internal memory.

The TX-2 Memory Element (see Fig. 2) is divided into four independently operating memories, each containing up to 65,536 36-digit words. The operating speed of TX-2 is determined by the cycle time for the memories: the 65,536 word S Memory is expected to have a cycle time of between six and seven microseconds; and the 4096 word T Memory, a cycle time between five and six microseconds. Both memories are parity checked.

Although the U Memory currently is not specified, it may contain a 4096 word core memory in the initial system. The V Memory consists of 8 flip-flop registers in the cental machine and 16 toggle switch registers which contain the program sequence executed whenever the START button on the operator's console is pushed. The contents of the toggle switch registers can be used as instructions or operands, but naturally cannot be altered by a program. The six 36-bit registers A,B,C,D,E and F are part of the V Memory but their contents can be used only as operands during the execution of an instruction. The programmer has, in a limited sense, a two address instruction machine when he refers to these registers in load and store type instructions. The other two flip-flop registers in the V Memory are a 60 cycle per second clock and a random number register.

When an instruction calls for the storing of an operand in memory, the operand memory cycle can be extended up to two microseconds. The extension occurs between the time that the memory register is read and the time that it is rewritten. During this extension time the memory register transfers in the central computer take place, the parity of the word read from memory is checked, and the parity of the new memory word computed. Because the extended cycle is less than the two complete cycles traditionally used for word-modifying instructions, an increase in computing efficiency is realized.

The P Register in the Program Element specifies the location of an instruction in memory and the N Register in the Control Element holds the instruction after it has been read from memory. The two leftmost digits of P select the memory system from which the instruction word is to be obtained; the right 16 digits address the word within the memory. Similarly, the Q Register locates the operand in one of the memory systems, the operand being placed in M.

Control and Indexing

An instruction word read into N has the structure shown in Fig. 3. The first two digits of the word specify information to the In-Out Element, and the four cf digits specify the computer configuration. The interpretation of the b and n digits is not discussed here.[2] The cf digits will be discussed later.

The operation code for the instruction is specified by the six op digits. On simple load and store type instructions these six digits are further subdivided into two groups of three. The first group determines the operation and the second specifies the register in the central computer which is being loaded or whose contents are being stored.

The base address for the operand, formed by the 18 y digits, is usually modified by the contents of the index register selected by the six j digits. The index registers form a unique 64 register, parity checked core memory which has a 1 microsecond access time. The contents of the specified index register is read into the X register of the Program Element via the paths indicated in Fig. 4. The base address and the index are fed into a full adder circuit which produces the sum, $Y = y + (j)$, in about 1 microsecond. The overall complexity of the Program Element was reduced by having the adder produce both the sum, Y, and the unmodified base address, y; either of these quantities can be directed to the operand memory address register Q. Whenever the zeroth index register is chosen, the adder produces only the unmodified base address. The effect is the same as having the index register contain zero, so the programmer can avoid index modification altogether.

The instruction memory address register P normally is indexed by one as each instruction is executed, but jump instructions may cause the output of the index adder to be directed to P. The adder also provides a communication path for index jump instructions from the X Memory to the Memory Element by way of the Exchange Element.

Arithmetic Element

The registers and sufficient basic operations in the Arithmetic Element to implement addition, multiplication, division, shift and various logical operations are shown in Fig. 5. Operation timing for most of the TX-2 instructions is also performed in the Arithmetic Element.

The design of the AE reflects the desire to attain high speed operation for TX-2 even when long-time instructions are being performed in the AE. The only instructions which require more than a memory cycle time for execution are those which involve shifting. These are, for example, multiply, divide, shift and normalize. For this reason the AE contains a sufficient number of storage registers to permit these instructions to be carried out in the AE while the remainder of TX-2 is freed to perform other instructions.

The four registers in the AE can each communicate with the E Register in the Exchange Element and thus with the Memory Element. As mentioned earlier, these registers are addressable as part of the V Memory System. Therefore, programmers have access to the results in any register of an AE computation.

The AE registers, designated by A, B, C, and D, are described below:
The A Register accumulates the results of all the arithmetic operations except division for which it holds the remainder. It holds one of the operands and accumulates the results of the three logical operations (AND, INCLUSIVE OR, EXCLUSIVE OR) which, it should be noted, are bit-wise operations. The information in the A Register can also be shifted (i.e. multiplied by some positive or negative power of two) or cycled (i.e. shifted, without preserving the special significance of the sign bit, as in a closed ring).

The B Register serves as an extension of A during multiplication, certain shifts and cycles, and, in a sense, during division when the least significant digits of the double-length divident are stored in B. The resulting quotient then appears in B. Moreover, the information in B can be shifted or cycled independently of A. In multiplication, the multiplier originally in A is transferred via parallel paths directly into B (where the least signigicant digit then controls the operation).

The C Register stores the partial carries during arithmetic operations, most important during multiplication as described later. Since these partial carries are actually bit-wise logical products AND, C is also used to accumulate logical products.

The D Register holds the multiplicands, divisors, addends and one of the operands for the logical operations. It also holds the numbers which control the shifting and cycling of A and B, namely the number of places, up to 62, and the direction, right or left. The facility of D to count is used also in accumulating the results of the normalizing of A and counting ONES in A.

Besides the above mentioned facilities, each of the AE registers can be complemented, which allows subtractions to be done.

AE Circuits

There are four Add One circuits on D, so that different parts of A and B can be controlled separately and simultaneously. For simplicity, just one Add One circuit is shown in Fig. 5. These Add One circuits use the simultaneous carry principle, permitting one count to occur every 0.4 microseconds. Each can count up to 127.

The logical Product circuit of A and D into C and the Sum Modulo 2 ("exclusive or") circuit of A and D into A when used at the same time are called a Partial Add. When the Complete Carry circuit is activated after a Partial Add, the result is a full addition of D and A into A. The Complete Carry circuit uses the high-speed carry principle and takes

.7 36
7
252

about 1.5 microseconds for 36 bits.

The Partial Carry and Shift Right circuit is also known as "multiply step" and was, we believe, first used on Whirlwind I. As used in multiplication, this circuit obviates the need for a full addition for each "one" in the multiplier. Carries are propagated only one stage during each step except the last when a complete carry is executed. This iterative process takes about 16 microseconds in the worst case for a full 36-digit multiplication. The iterative process for division, on the other hand, requires a complete addition at each step and consequently takes about 72 microseconds in the worst case.

Two features of the AE control ought to be mentioned here. A 7-bit step counter, like the Add One circuit on D, is used to control multiplication and division and to limit the shifting in normalizing and the cycling in counting "ones". A flip-flop signifying overflow during addition and division is also used to remember the sign of the product during multiplication and the sign of the quotient during division. If a division overflow occurs the sign is replaced by the overflow state and the quotient is lost.

Control of the Arithmetic Element is independent of the rest of the machine, thus providing the time saving device of continuing to execute non AE instructions while AE is performing one of the longer shift operations or a division.

System Timing

In part the high speed of TX-2 is attained by overlapping the operation of as many components as is logically possible without incorporating large amounts of circuitry. The time consuming cyclic operations in an indexed single-address computer are the instruction memory cycle, the index memory cycle, the index addition time, the operand memory cycle, and the operation timing. These cycles occur in the mentioned sequence during the execution of ordinary instructions. Several asynchronous "clocks" which use a 5 megacycle pulse source control the various cycles. The instruction and operand memory cycles can be overlapped if they take place in different memory systems.

The overlap of these cycle times for a sequence of load type instructions is illustrated in Fig. 6 (a). Here different instruction and operand memories with roughly equal cycle times is assumed. If a sequency of store type instruction are executed which require extended memory cycles for the operand, then the situation shown in Fig. 6 (b) results. Fig. 6 (c) shows the time used when both the instruction and the operand are in the same memory.

"Peak" operating speed for the computer is attained only in Fig. 6 (a); additional circuitry could improve Fig. 6 (b) and Fig. 6(c), but only at considerable cost. It is interesting to note that if the computer is to run at peak speeds, the address of the operand used by the current instruction must be available before the earliest moment at which the next instruction memory cycle could begin. If the total accumulated time from the beginning of an instruction memory cycle till the time that the address of the operand is known is greater than the instruction memory cycle time, then the computer can not run in the ideal manner shown in Fig. 6 (a). This means that the access time of memories and the index add time must be kept as short as possible.

Fig. 6 (d) depicts the timing of events when the In-Out Element causes a change in program sequency by changing the contents of the P register. The additional X Memory cycle which must be performed in doing this produces a timing situation similar to that of the X Memory load and store type instructions.

The operation timing for an instruction is executed when the operand is available from memory. Only the Arithmetic Element step counter instructions, multiply, divide, shift, etc., require an operating timing cycle longer than a memory cycle. Since only the Arithmetic Element is tied up when these instructions occur, the Control Element permits any non-Arithmetic Element instruction to be executed while the AE is busy. Division takes up to 75 microseconds, so the programmer can write as many as 14 non-AE instructions following a divide, all of which can be executed before the division is completed.

Configuration

The design of a general purpose computer must necessarily reflect the contradictory demands for both short and long word lengths, floating and fixed point arithmetic operations, and a multitude of logical and decision instructions. The computer should be able to process information at an optimum rate in a variety of problems without the need for intricately coded programs. This ability should be achieved without excessively complex and costly circuitry.

The full 36-digit word in TX-2 represents a reasonable length for operands in some numerical computations, notably scientific and engineering computations. Though floating point arithmetic operations are not included in the instruction code, both they and multiple-precision operations can be easily synthesized by means of the existing instructions. The logical instructions in the code facilitate operations on individual digits, but also, a configuration which the programmer specifies anew with each instruction permits him to perform arithmetic operations on operands which are less than 36 digits long. When such is the case, several shorter operands can be manipulated simultaneously.

The four cf digits in an instruction word (see Figure 3) are decoded as shown schematically in Figure 7. The contents of the selected one of 16 9-digit configuration words is placed in a flip-flop register whose output levels determine a static configuration for the entire computer during the execution of the instruction. The contents of the first twelve registers are specified by a notation whose meaning will be clarified in the following discussion.

The full 36-digit word length is always maintained for instruction words, but during operation timing, every 36-digit register in the Memory, Exchange and Arithmetic Elements is considered broken into four 9-digit quarters (numbered from 1 to 4, from right to left as in Fig. 8 (a). While the instruction is being executed, these quarters are recombined on the basis of the configuration.

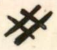
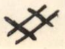
Parallel register transfers are the usual means for moving information about in the machine. The EE permutation digits select one of the four permutations $P_{||||}$, $P_{\cancel{|||}}$, $P_{\cancel{\cancel{|||}}}$, $P_{\cancel{///}}$, shown in Figure 8 (b). The chosen permutation effects the corresponding cross-communication paths between the quarters of the E and M registers of the Exchange Element. As operands are transmitted through the EE, the quarters of the work follow the set of paths determined by the selected permutation. The result is that the operand is shifted 9n places to the left as it moves from M to E or 9n places to the right as it moves from E to M, n = 0,1, 2 or 3. Thus the programmer can have any quarter of the AE communicate with any quarter of the ME.

This communication ability is focused more sharply by having the configuration specify a system activity. All operation timing events in a given quarter of the AE and EE and the quarter of the ME connected via the selected permutation path in the EE are controlled by the activity flip-flop on that quarter. If the activity flip-flop of a given quarter is a one, as specified by the configuration, then the operation timing events of the instruction occur in that quarter. If the activity flip-flop is a zero then nothing happens.

During the execution of arithmetic operations, the AE coupling bits further specify the connections of the lateral information paths between quarters in the AE. Information flows laterally only through the shift and the carry circuits, and the connection of these circuits alone determines the word length of the numerical quantities manipulated in the AE.

As shown in Fig. 9 (a), every quarter of the AE has coupling units at each end which receive the shift and carry information entering the quarter. The general type of connections between several quarters is shown in Fig. 9 (b). The digit length of operands during add and shift operations is determined by the number of quarters coupled together. In TX-2 from one to four quarters can be coupled together to permit arithmetic operations on 9, 18, 27, or 36 digit operands. The various combinations of coupling unit connections actually chosen by the AE coupling are symbolized in Fig. 9 (c). Since A-register, B-register and AB-register shifts are permitted in the Arithmetic Element, the programmer can obtain 18, 36, 54 or 72 digit shifts. All the possible shift (and cycle) configurations are shown in Fig. 9 (d).

Only those inputs to the coupling units which would yield useful arithmetic element structures are realized by the AE coupling. It should be emphasized that the programmer can realize several arithmetic elements simultaneously. The coupling (36) gives only one 36-bit AE, but the coupling (18,18) gives two complete, independent 18-bit arithmetic elements which are separately, but simultaneously controlled by the instruction being executed. Two arithmetic elements are again available with the coupling (27,9) one 27 bits and the other 9 bits long, and the (9,9,9,9) case gives four 9-bit arithmetic elements. The permutation paths in the Exchange Element permit each arithmetic element to communicate with any quarter of a memory word and the activity flip-flops can specify just which of the realized arithmetic elements will actually be active and in active communication with the connected part of memory.

In Figure 10, several examples are given of the different configurations which can be realized in TX-2. The most straightforward configuration has one 36-digit arithmetic element and communicates directly with memory. The notation, (||||,$\underline{36}$) signifies the permutation (no shift) and the form of the arithmetic element (one 36-digit). The underlining indicates that the whole system is active. Slightly more varied is the (||||, 9,$\underline{9}$,9,$\underline{9}$) configuration which specifies four 9-digit arithmetic elements communicating directly with memory, but with only two of them active. The ( ,9,$\underline{9}$,9,$\underline{9}$) configuration has the same arithmetic elements but with the associated memories interchanged. The ( ,18,$\underline{18}$) configuration illustrates an 18-digit arithmetic element which uses the "other" half of memory.

Ond of the 9 configurations digits is at the moment unused, but will probably be used to control the extension of the sign of numbers as they pass through the EE on the way from the ME to the AE. The scheme presently under consideration would permit programmers to add, for example, a 9-digit memory operand to an 18-digit arithmetic element. This scheme would permit closer packing of operands in memory and significantly increase the speed of solving some real-time problems, where short data words need to be extended so that higher precision can be maintained during computations. The working details of the scheme have yet to be fixed.

The configuration memory from which the programmer chooses a configuration for use with each instruction was shown in Fig. 4. Twelve of the configuration memory registers are fixed circuitry whose contents cannot be changed without changing the wiring of the computer. These configurations are assumed to be ones which will be useful to most programmers. The last four registers in the memory consist of the 36 digits of the F register. As will be seen the programmer can quite simply alter the contents of this register and thereby obtain any of the (less than 2^9) possible configurations.

Instruction Code

Of the 64 possible operation codes, only 51 are currently decoded to define instructions. In Table I the effect of each instruction is described. If several computers are defined by the configuration, then the effect occurs in all of them simultaneously and independently. The notation used in the definition of the operation is described in Table II.

The instructions are grouped according to type. Load and store type instructions simply effect an operand transfer between the selected register and memory. The load complement instructions are variants which load the ones complement into the specified registers. Exchange simply interchanges the contents of A and the indicated memory register. The insert instruction allows any set of bits in A, as specified by the bits in B to be stored in memory. In the index memory load and store instructions, the j bits select the index register involved so the operand address is not modified.

All of the add and step counter instructions can also be classed as load type instructions in so far as the operand memory cycle is concerned. The multiply instruction forms the full product in the A and B registers. Division is the inverse of multiplication, the double length dividend in A and B being divided by the memory operand. The remainder is left in A and the quotient in B. Normalize shifts the contents of A and B left till the magnitude of the number in A is between one-half and one. The number ofshifts to do this, the normalizing factor, is subtracted from the memory operand in D. The shift and cycle instructions use the memory operand, rather than the address section of the instruction, to specify the number of places to shift. This is necessary since more than 18 bits are required to specify all the possible shifts for the $(\underline{9},\underline{9},\underline{9},\underline{9})$ configuration. The count ones instruction adds the number of bits in A which are ones to the memory operand in D. This provides a simple means for determining bit density in areas of storage, since the one's count for several words can be accumulated in D.

The two replace add instructions, using the index memory, facilitate instruction and index modification. Both require two memory cycle times for execution.

Thw two in-out read instruction transmit information between the memory and the selected in-out unit. The details of these and the in-out select instruction are given in another paper.

Single bits in memory can be manipulated with the three bit-setting instructions. The bit-sensing instruction facilitates the use of and the variety of jump instructions available simplifies the coding of logical-decision functions. The two index jump instructions permit indexed program loops to refer successively in either the forwards or backwards direction to operands in a data block. The unconditional jump instruction uses the <u>cf</u> digits to specify whether the selected index register will be used to remember the previous contents of P. These contents are always transmitted to the E register whenever a jump occurs.

Arithmetic overflows can be caused by addition, subtraction and division instructions. Such overflows as do occur are remembered in overflow flip-flops in the arithmetic element. The overflow condition can be detected by a jump instruction, or by the in-out element in a manner described in another paper. If an overflow is anticipated, however, it can be shifted into the A register by executing a normalize instruction.

A normalize usually shifts AB left, but if an overflow exists AB is shifted right one place, and the overflow placed in the most significant digit position of A to the right of the sign digit. The memory operand is increased by one in the D register, when this occurs, rather than decreased. This interpretation of an overflow permits floating point operations to be programmed quite simply in the arithmetic element. The in-out select and operate instructions differ from all the others in the sense that the y digits are used to specify different operations. In-out select chooses the mode in which an in-out unit will run. The operate instruction will control individual useful commands, as for example, round-off.

Instruction Times

The average execution time for instructions depends upon whether one memory or two different overlapped memories are used for instructions and operands. In the latter case the average time is the longer of the instruction memory and operand memory cycle times, and in the first cause the sum of the two cycle times. It should be remembered that any instruction which involves storing an operand in memory has the normal operand memory cycle time extended by from one to two microseconds. Instructions which alter or transfer the contents of index memory registers, require approximately two normal memory cycles even when instruction and operand memory cycles are overlapped.

Successive step counter instructions require a time which depends upon the length of the longest active arithmetic element. In the case of multiply, divide and count ones this time is a function of the operand word length only, but the shift, cycle and normalize times depend upon the number of places actually shifted. Divide requires about 2 microseconds per digit and all other step counter instructions o.4 microseconds per digit. These shift times become significant only when they exceed the one or two memory cycles already required. In the worst, 36-digit case about 75 microseconds is required for division and 19 microseconds for multiplication. A 72 place shift would take 32 microseconds. These are the times required for these instructions when they are written in sequence. If the operand word length is shorter, then these times become proportionally less, down to the minimum memory times required.

Summary

The organization of TX-2 permits a programmer to pay considerable attention to coding details and receive a worthwhile reward in the form of increased efficiency of operation. The operating speed can be doubled when instruction and operands are stored in different memories. Further increases result by the sequencing of instructions so that non-arithmetic-element instructions are executed concurrently with AE step-counter instructions. And the ability to choose a configuration with each instruction means not only that some instructions take less time, but also that many of them can be eliminated from a program altogether.

However, this versatility and efficiency is not accompanied by a disastrous loss in simplicity. The system organization is such that details can be easily ignored by the naive programmer, without the details having even subtly obtrusive effects. If all the digits in an instruction word are zero except for the operation code and the base address, then TX-2 appears as a simple single address 36-bit operand word computer with a single, uniformly addressed 70,000 word memory.

If the j-bits are used, then the machine is enlarged to become an indexed single-address 36-bit operand word computer for which the entire instruction code is meaningful. When the b and d bits are used, then the programmer can control the manner in which several in-out units running concurrently can cause program sequence changes. And by selecting various configurations the programmer can perform more operations simultaneously with each instruction.

The different facilities for indexing, memory overlap, instruction overlap, multiple-sequencing and configuration can be ignored or used as the programmer desires. Ignoring them would seem to permit straightforward coding; using them actually permits much shorter and faster codes for a given function. Each facility is easily represented by a clear conceptual picture of what the facility permits, the only real difficulty being the greater number of simultaneous actions possible with each instruction. However, higher speeds and greater system capacity are obtained by shorter cycle times, increased bit storage and greater simultaneity of events. In TX-2 all three aspects are emphasized.

TABLE I

Type	Mnemonic Code	Operation	Name
	lda	$(Y) \rightarrow \left\{\begin{matrix} A \\ B \\ C \\ D \\ E \\ E \\ F \end{matrix}\right.$	Load into A
	ldb		Load into B
	ldc		Load into C
	ldd		Load into D
Load	lde		Load into E
	lde		Load into E
	lde		Load into F
	lca	$\overline{(Y)} \rightarrow \left\{\begin{matrix} A \\ B \end{matrix}\right.$	Load complement into A
	lcb		Load complement into B
	ldx	$(y) \rightarrow j$	Load into index
	sta	$\left.\begin{matrix} (A) \\ (B) \\ (C) \\ (D) \\ (F) \end{matrix}\right\} \rightarrow Y$	Store A
	stb		Store B
	stc		Store C
	std		Store D
	stf		Store F
Store	exa	$\left\{\begin{matrix} (Y) \rightarrow A \\ (A) \rightarrow Y \end{matrix}\right.$	Exchange A
	ins	$(B) \& (A) \vee (B) \& (Y) \rightarrow Y$	Insert digits of A
	stx	$(j) \rightarrow y$	Store index
	add	$(A) + (Y) \rightarrow A$	Add
	sub	$(A) + \overline{(Y)} \rightarrow A$	Subtract
	dma	$\lvert(A)\rvert + \overline{\lvert(Y)\rvert} \rightarrow A$	Difference of magnitude
	and	$(A) \& (Y) \rightarrow A$	Logical and
	ori	$(A) \vee (Y) \rightarrow A$	Logical or - inclusive
Add	ore	$\left\{\begin{matrix} (A) \oplus (Y) \rightarrow A \\ (A) \& (Y) \vee (C) \rightarrow C \end{matrix}\right\}$	Logical or - exclusive (and accumulate product)
	axm	$(j) \quad (y) \rightarrow y$	Add index to memory
	amx	$(j) \quad (y) \rightarrow j$	Add memory to index
	sbo	$1 \rightarrow Y_j$	Set j-th bit one
Set bit	sbz	$0 \rightarrow Y_j$	Set j-th bit zero
	sbc	$(Y_j) \rightarrow Y_j$	Set j-th bit complement

TABLE I (Continued)

Type	Mnemonic Code	Operation	Name
	mul	$(A) \times (Y) \rightarrow AB$	Multiply
	div	$(AB) \div (Y) \rightarrow \left\{\begin{matrix} A \text{ (remainder)} \\ B \text{ (quotient)} \end{matrix}\right.$	Divide
	sha sab shb	$\left.\begin{matrix}(A)\\(AB)\\(B)\end{matrix}\right\} \times 2^{(Y)} \rightarrow \left\{\begin{matrix}A\\AB\\B\end{matrix}\right.$	Shift A Shift AB together Shift B
Step-Count	cya cab cyb	$\begin{matrix}(A)\\(AB)\\(B)\end{matrix} \text{ cyc } (Y) \rightarrow \left\{\begin{matrix}A\\AB\\B\end{matrix}\right.$	Cycle A Cycle B together Cycle B
	nab	$\left\{\begin{matrix}(AB) \times 2^{nf} \rightarrow AB \\ (Y) - nf \rightarrow D\end{matrix}\right.$	Normalize AB
	coa	$(Y) + no \rightarrow D$	Count ones in A
	rds	$\left\{\begin{matrix}(Y) \rightarrow 10\\(10) \rightarrow Y\end{matrix}\right\}$	Read and shift
In-out			
	rdn	$\left\{\begin{matrix}(Y) \rightarrow 10\\(10) \rightarrow Y\end{matrix}\right\}$	Read without shift
	jpe	If $(E_j) = 1$, then $y \rightarrow P$	Jump if j-th bit of E is a one
	jpp	If any $(A) > 0$ then $Y \rightarrow P$	Jump if the contents of any A is positive
	jpn	If any $(A) \leq 0$	Jump if the contents of any A is ngeative
	jpz	If any $(A) = 0$	Jump if the contents of any A is zero
Jump	jpo	If any (A) overflowed	Jump if the contents of any A has overflowed
	jxp	If $(j) \geqslant 0$, then $(j) - cf \rightarrow j$, $y \rightarrow P$	Jump if index positive and decrease index
	jxn	If $(j) < 0$, then $(j) + cf \rightarrow j$, $y \rightarrow P$	Jump if index negative and increase index
	jpu	If $cf = 1, 3$, then $(P) + 1 \rightarrow j$ If $cf = 0, 1$, then $y \rightarrow P$ If $cf = 2, 3$, then $Y \rightarrow P$	Jump unconditionally
Misc.	ios opr		In-out select Operate

TABLE II

Notation	Meaning
$\longrightarrow$	goes into
(x)	contents of x
$Y = y + (j)$	indexed memory address
$\|(x)\|$	magnitude of (x)
$\overline{(x)}$	one's complement of (x)
$\&$	logical and operation
v	inclusive or operation
$\oplus$	exclusive or operation
$+$	one's complement addition
nf	number of shits to normalize
no	number of ones
Y_j	j-th digit of register Y

REFERENCES

1. Forgie, J.W., "The Lincoln TX-2 In-Out System," Proc. Western Joint Computer Conference (Feb. 1957).

2. Op. Cit.

LIST OF FIGURE CAPTIONS

For paper entitled: A FUNCTIONAL DESCRIPTION OF THE LINCOLN TX-2 COMPUTER

Figure 1 TX-2 System Schematic - Showing the principal registers and transfer paths.

Figure 2 TX-2 Memory Element - Two address and two buffer registers are used to permit simultaneous operation of any two of the four memories.

Figure 3 TX-2 Instruction Word Layout

Figure 4 TX-2 Program Element - Determining the instruction and operand memory addresses, performing X Memory operations.

Figure 5 TX-2 Arithmetic Element - Showing the circuits and transfer paths for AE operations.

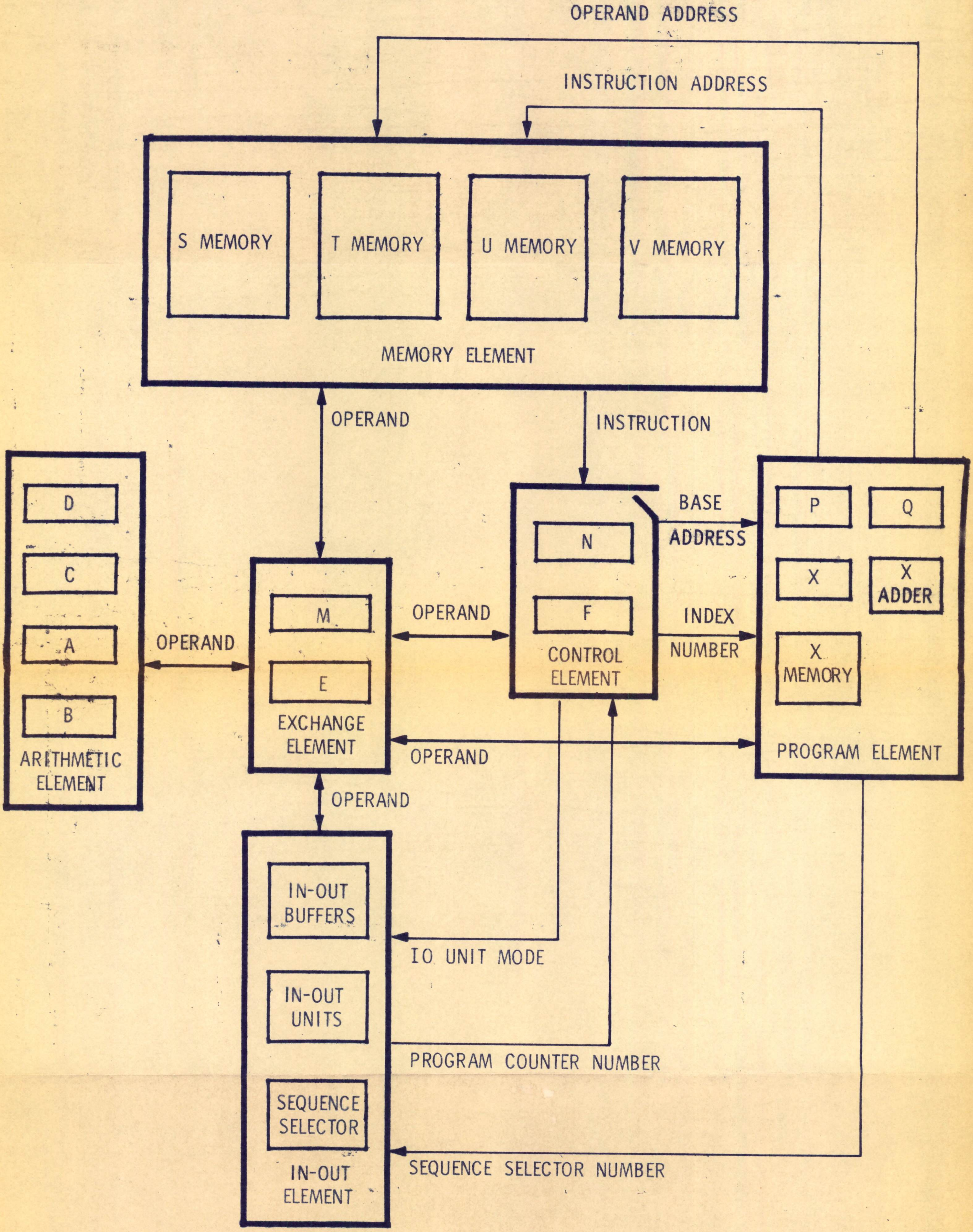
Figure 6 TX-2 Timing Schematic - Showing overlapped execution of memory and operation cycles.

Figure 7 TX-2 Configuration Selection - The cf digits select a configuration for the computer for use during the execution of the instruction.

Figure 8 TX-2 Configuration - (a) Quartering, permutation paths, and activity flip-flops. (b) The four sets of permutation paths available, one of which is used during the execution of an instruction.

Figure 9 TX-2 Arithmetic Element Coupling Units - (a) the coupling units which receive information moving laterally into the i-th quarter of the AE, i=1,2,3,4. (b) Coupling unit connections between a contiguous group of quarters which realize a 9-bit (j=0), 18-bit (j=1), 27-bit (j=2), or 36-bit (j=3) "arithmetic element." (c) The four forms the arithmetic element can assume with associated operand word structure. (d) The possible shift path arrangements realized with the configurations.

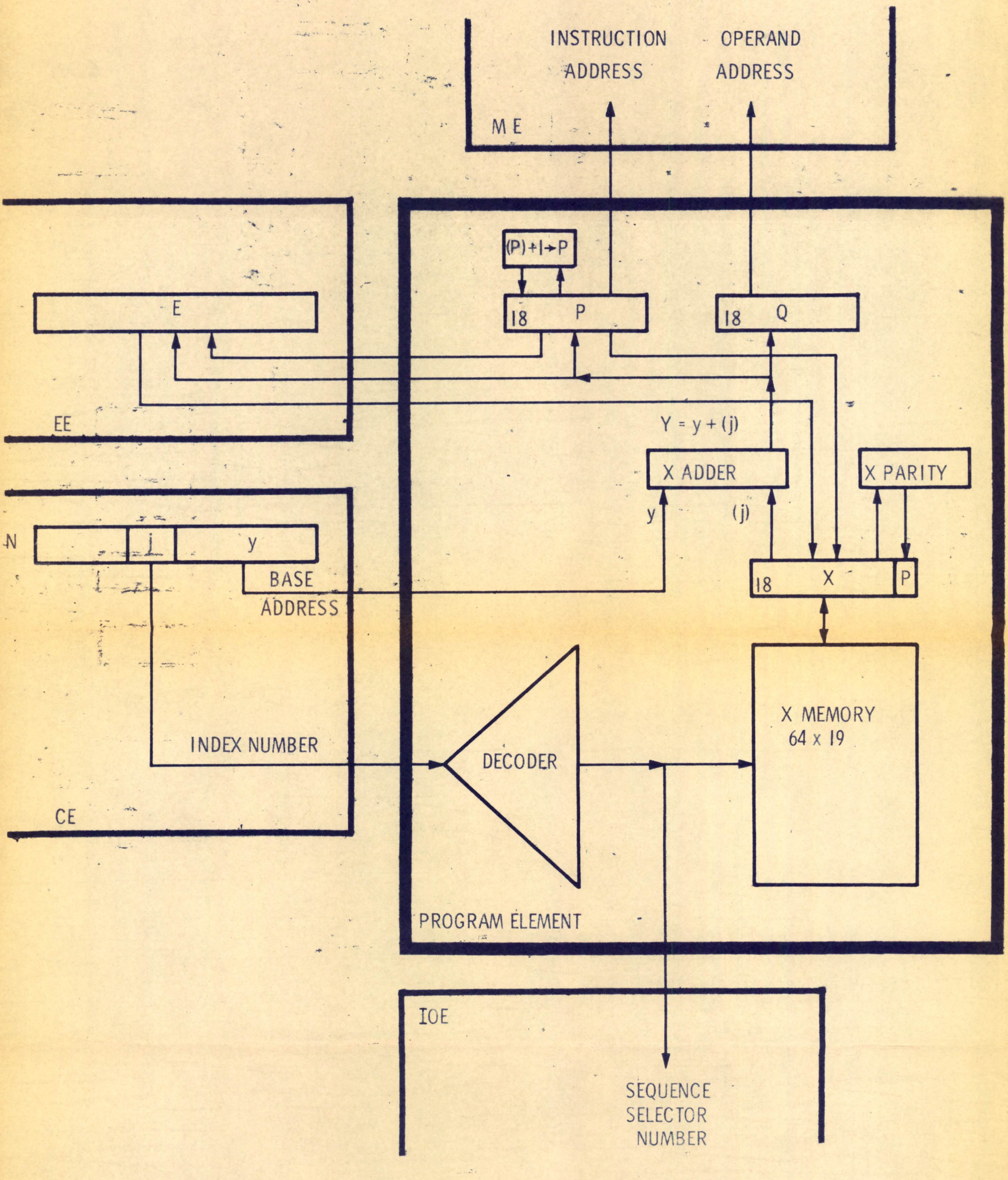
Figure 10 Illustrative Examples of Different Configurations - Areas of activity during execution of instruction are shown shaded. Effects of AE coupling are shown by juxtaposition.



TX-2 SYSTEM SCHEMATIC

B-69408
S-1630
C60-377

Fig. 1

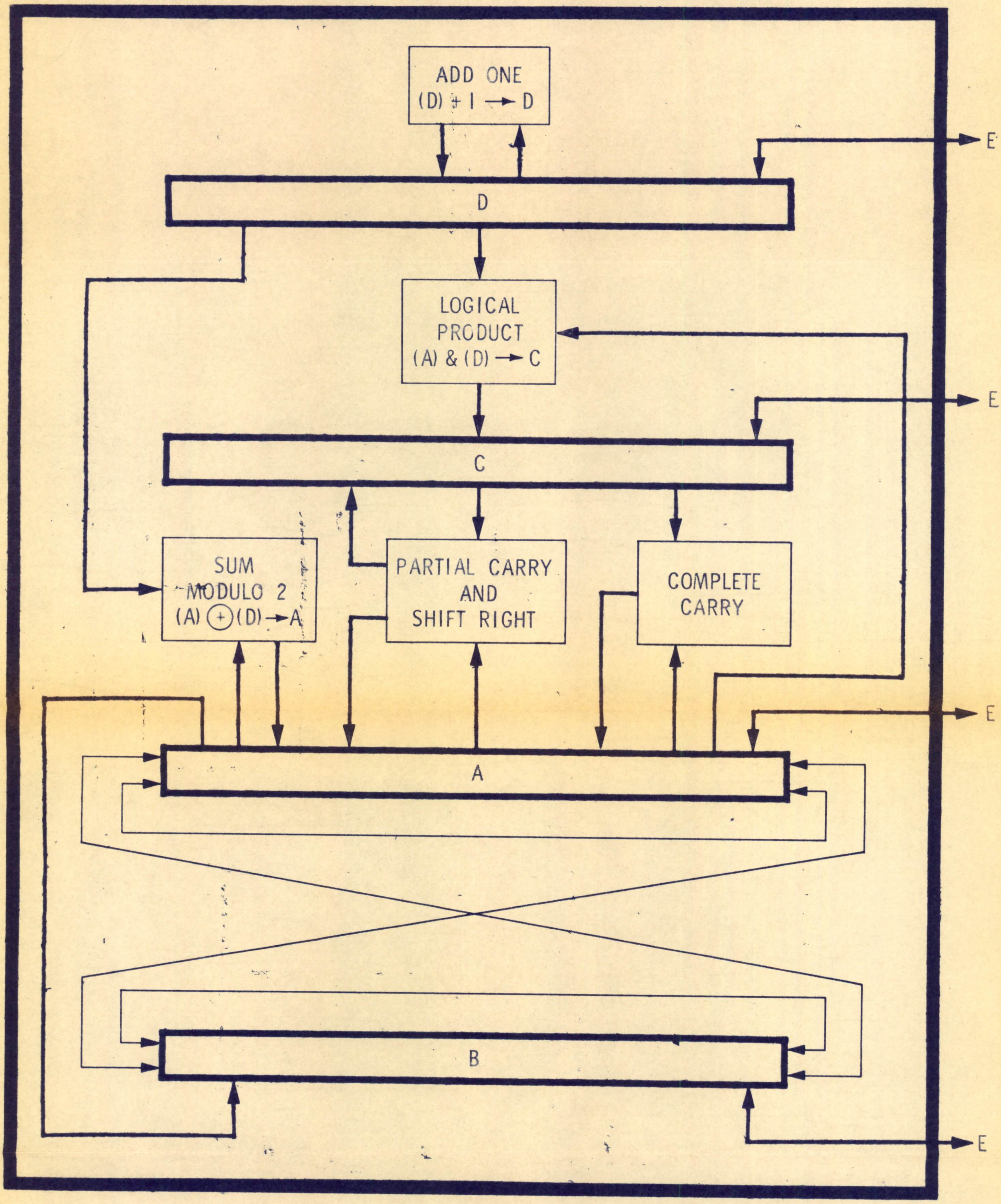


TX-2 PROGRAM ELEMENT

Showing paths enabling index adding and storing and loading program counter from the index memory and the exchange element

B-69393
C 60-345
S-1673

fig. 4

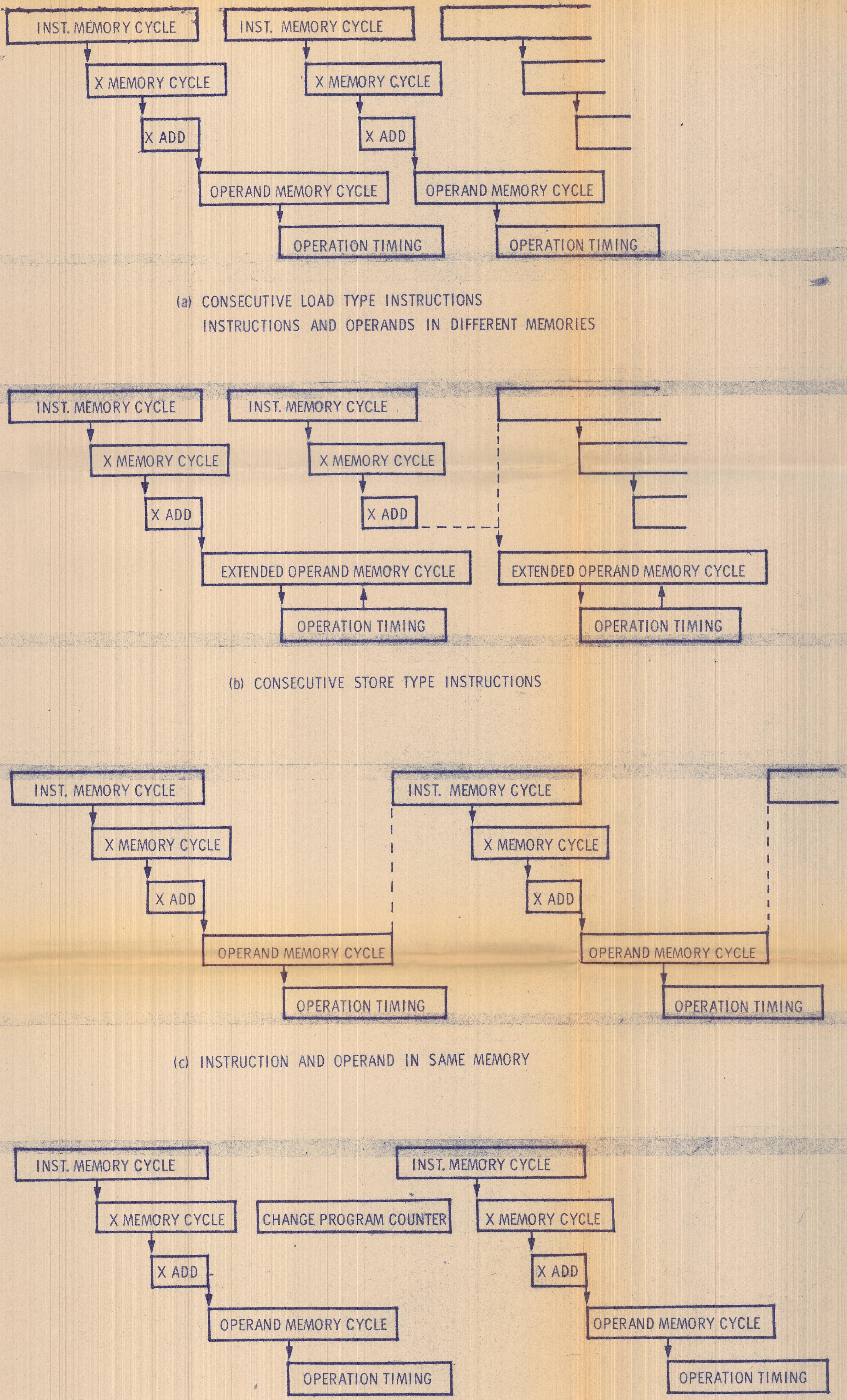


CIRCUITS AND TRANSFER PATHS (GENERAL) OF ANY TX-2 ARITHMETIC ELEMENT FORMS

B-69416
C60-344
S-1675

Fig. 5.

Fig. 6



(a) CONSECUTIVE LOAD TYPE INSTRUCTIONS
INSTRUCTIONS AND OPERANDS IN DIFFERENT MEMORIES

(b) CONSECUTIVE STORE TYPE INSTRUCTIONS

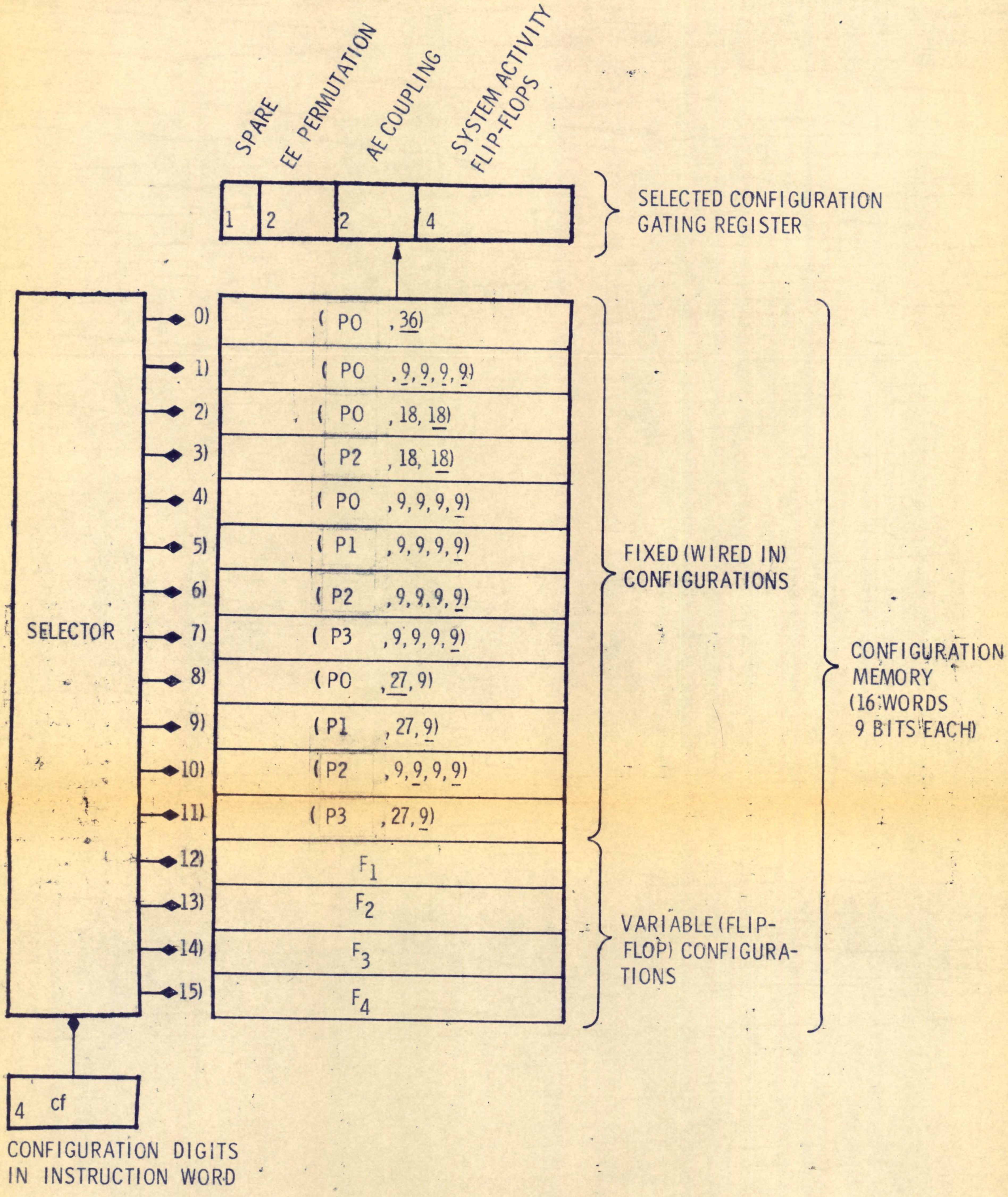
(c) INSTRUCTION AND OPERAND IN SAME MEMORY

(d) CHANGE SEQUENCE

-69361
S-1626 (a) C60-382
S-1627 (b) C60-383
S-1628 (c) C60-384
S-1629 (d) C60-385

TX-2 TIMING SCHEMATIC

Showing overlapped execution of memory and operation cycles



TX-2 CONFIGURATION SELECTION

B-69404-1
C60-346
SN-1674
SN-1681
C60-421

fig 7

C-69367-1
S-1625
C60-379
SN-1678
C60-422

fig. 8

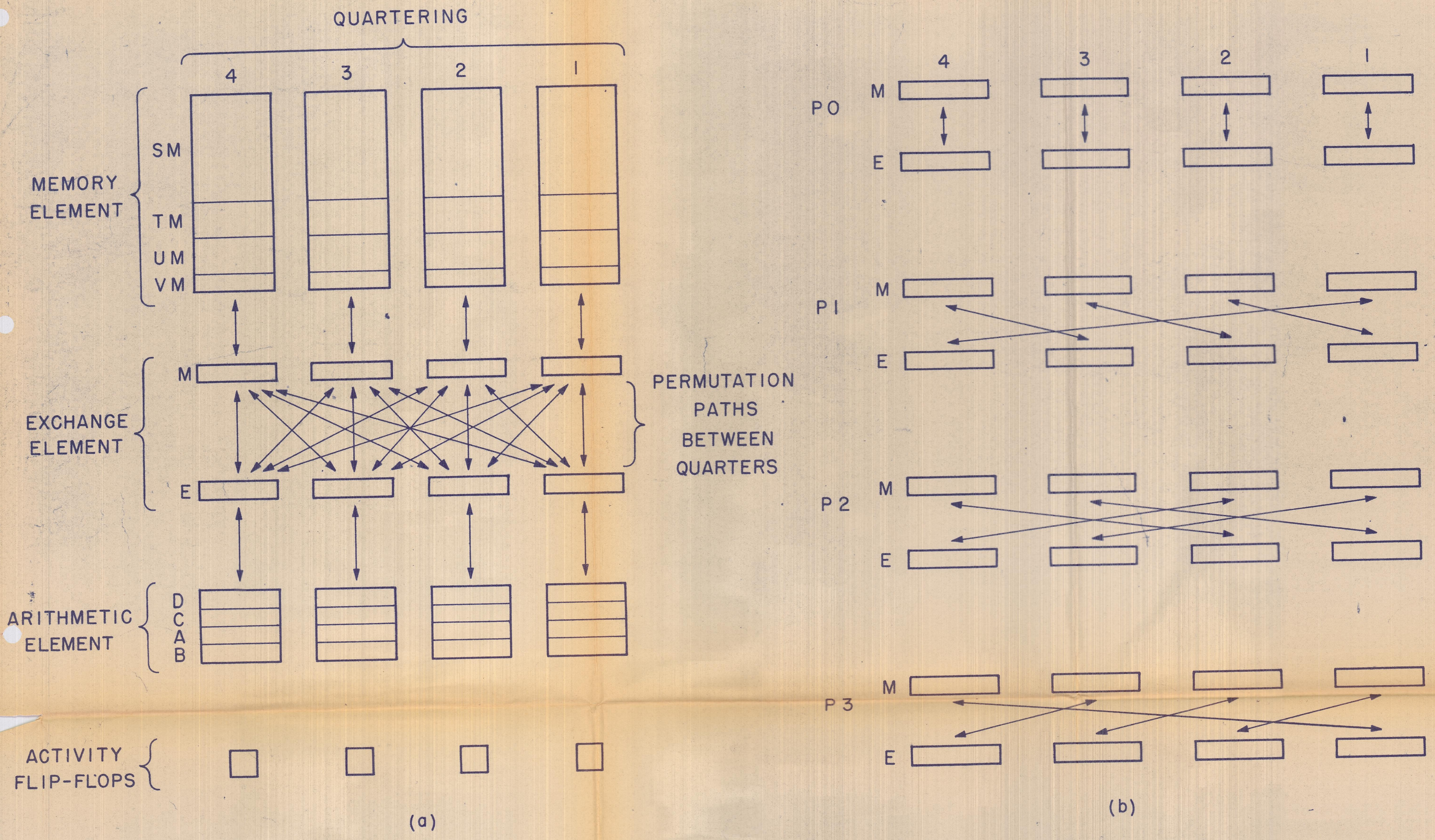


FIG. 4

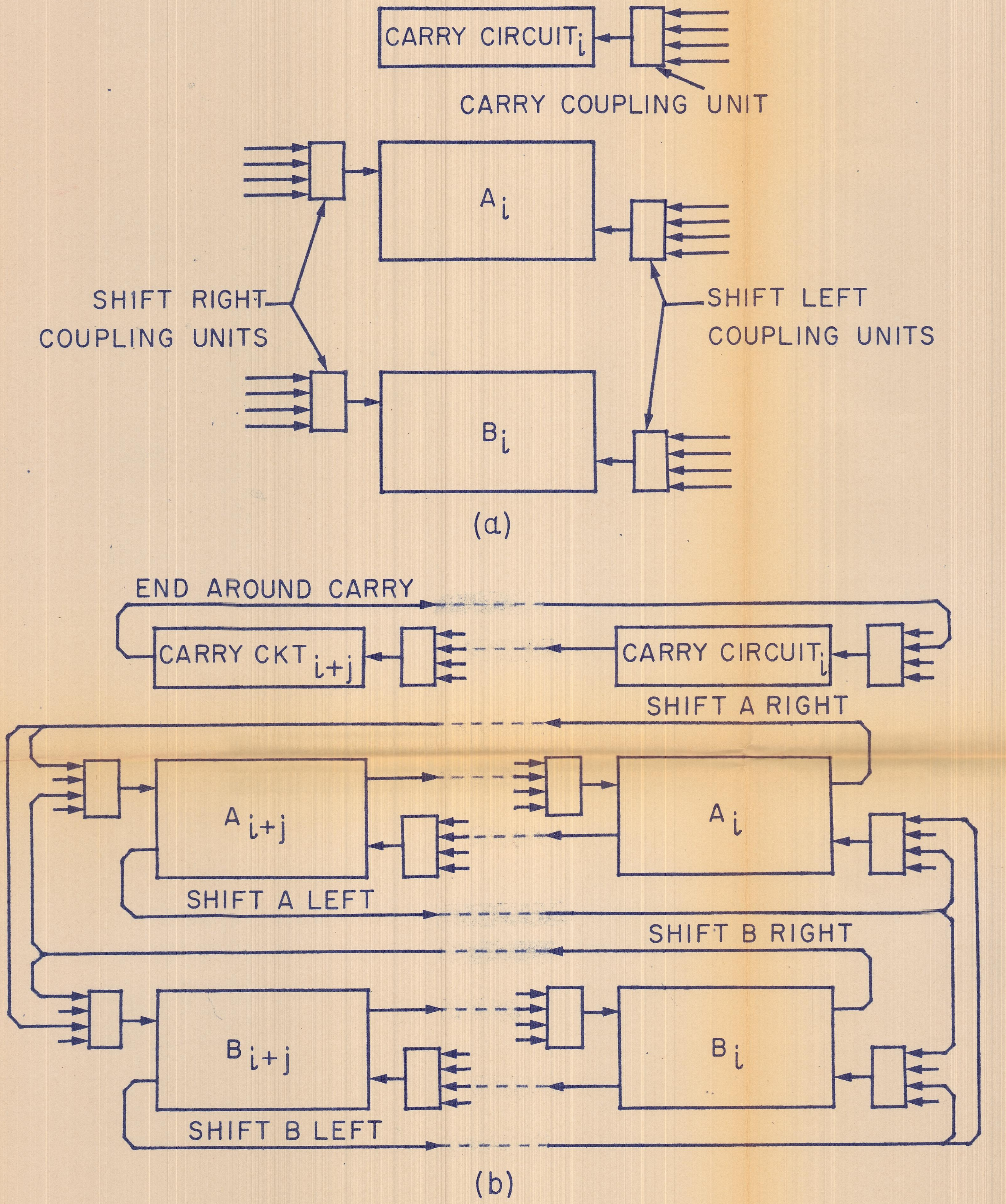
TX-2 CONFIGURATION

(a) QUARTERING PERMUTATION PATHS AND ACTIVITY FLIP-FLOPS SHOWN

(b) PATHS IN EXCHANGE ELEMENT

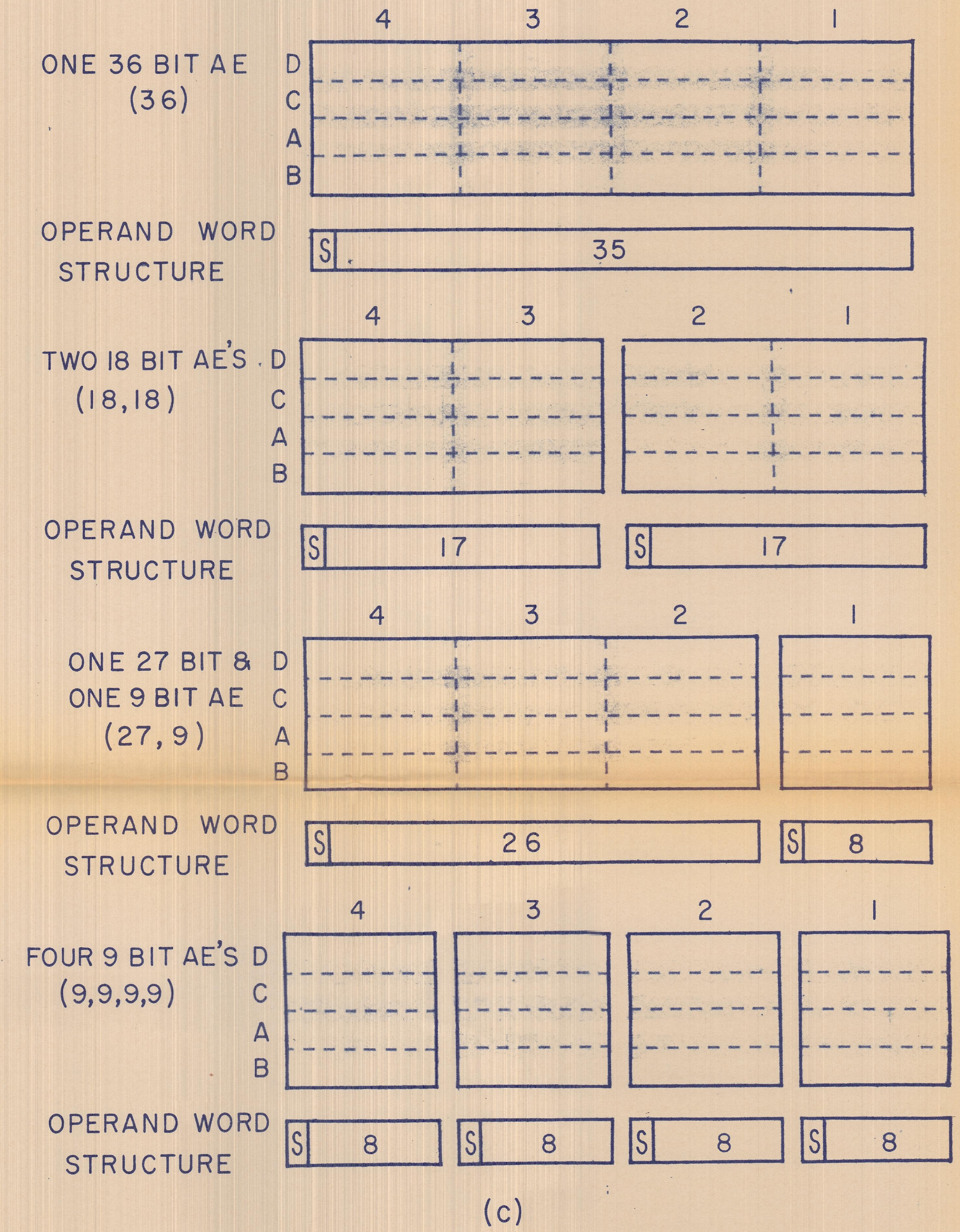
C-69380-1
C 60-374
SN-1676
C60-419

Fig 9abc



(a) ith QUARTER COUPLING UNITS

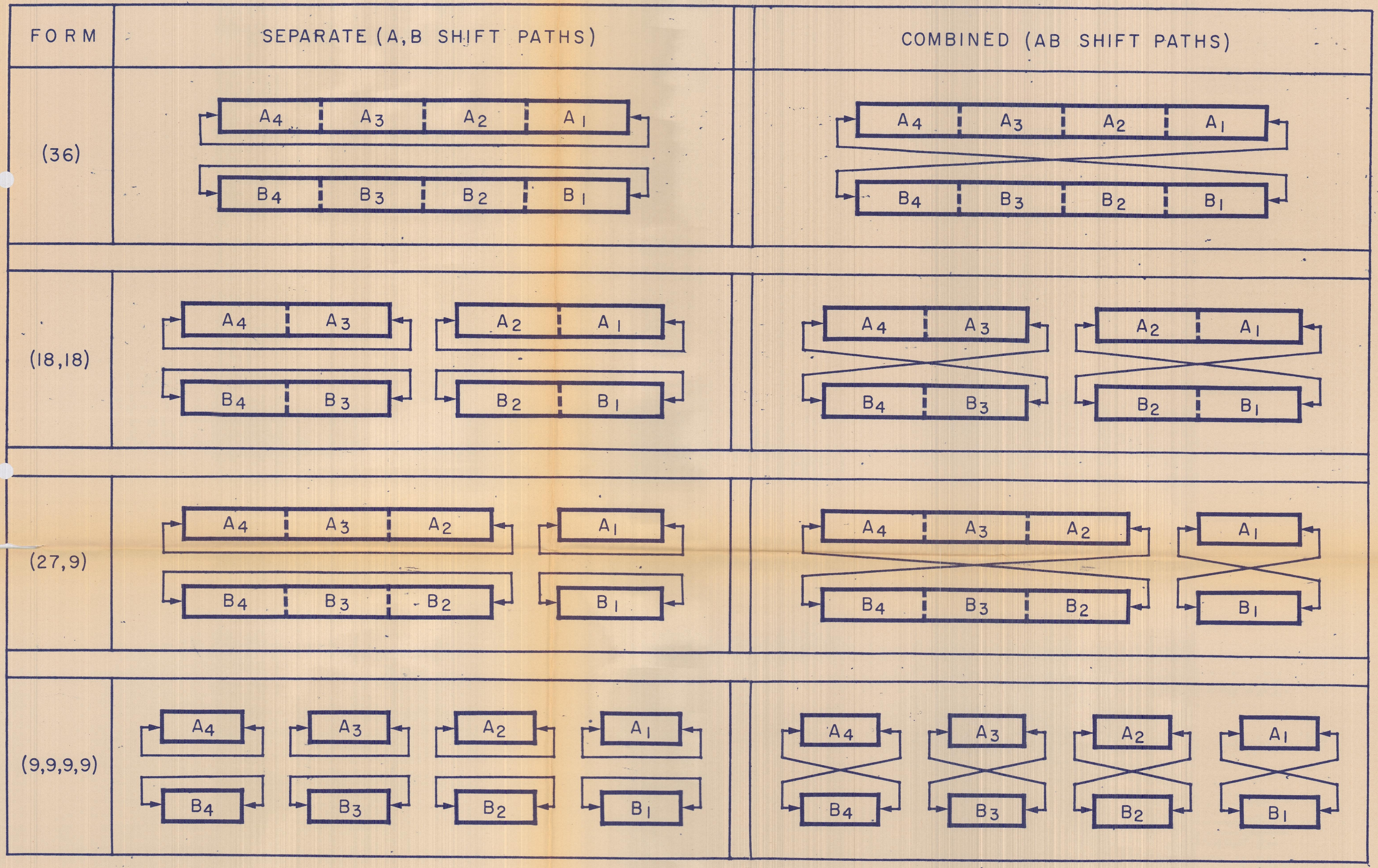
(b) COUPLING UNIT CONNECTIONS



(c) TX-2 CONFIGURATION—ARITHMETIC ELEMENTS AND OPERAND WORD STRUCTURES

C-69411
C60-372
S-1677

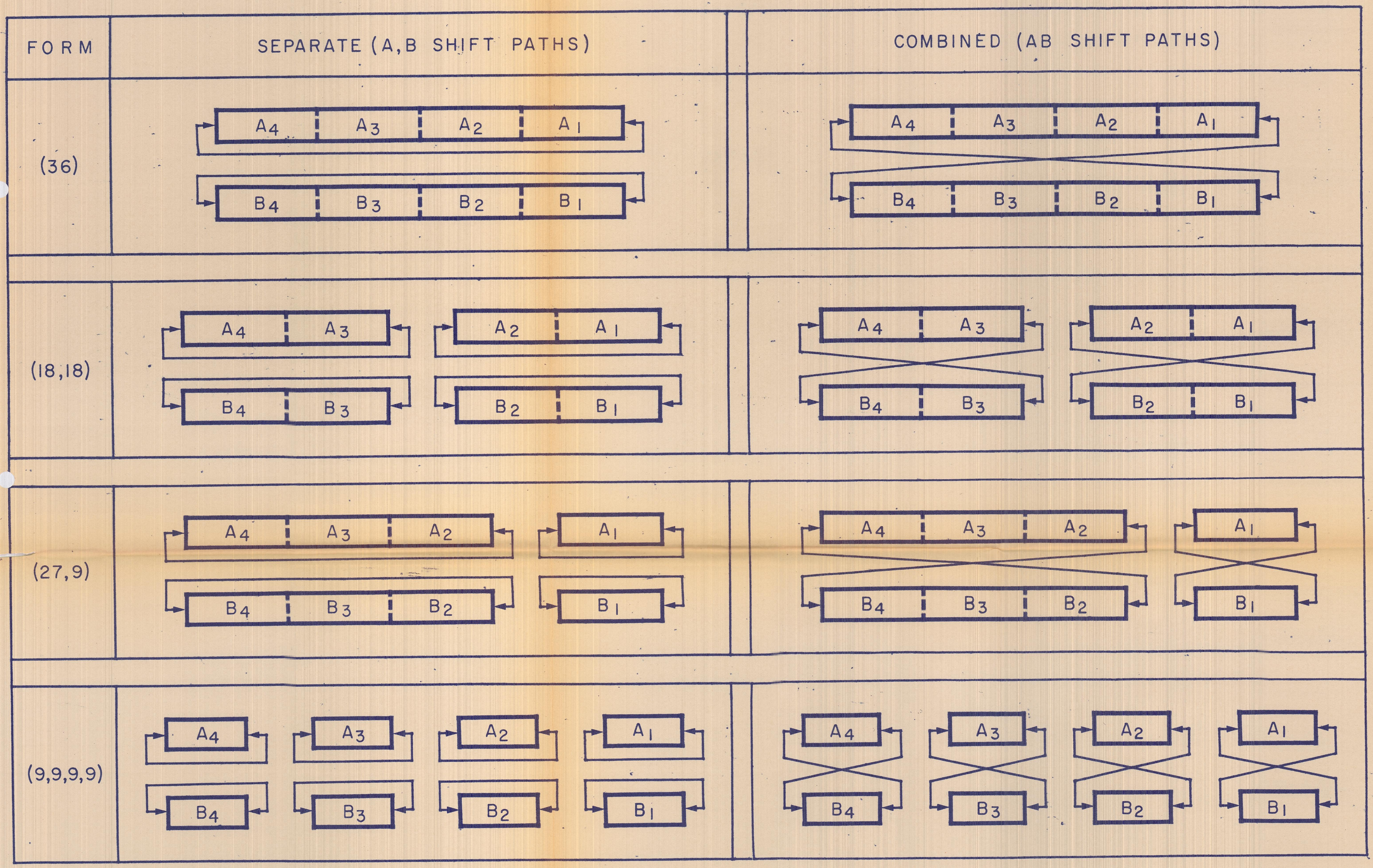
Fig. 9d



TX-2 SHIFT PATH ARRANGEMENTS

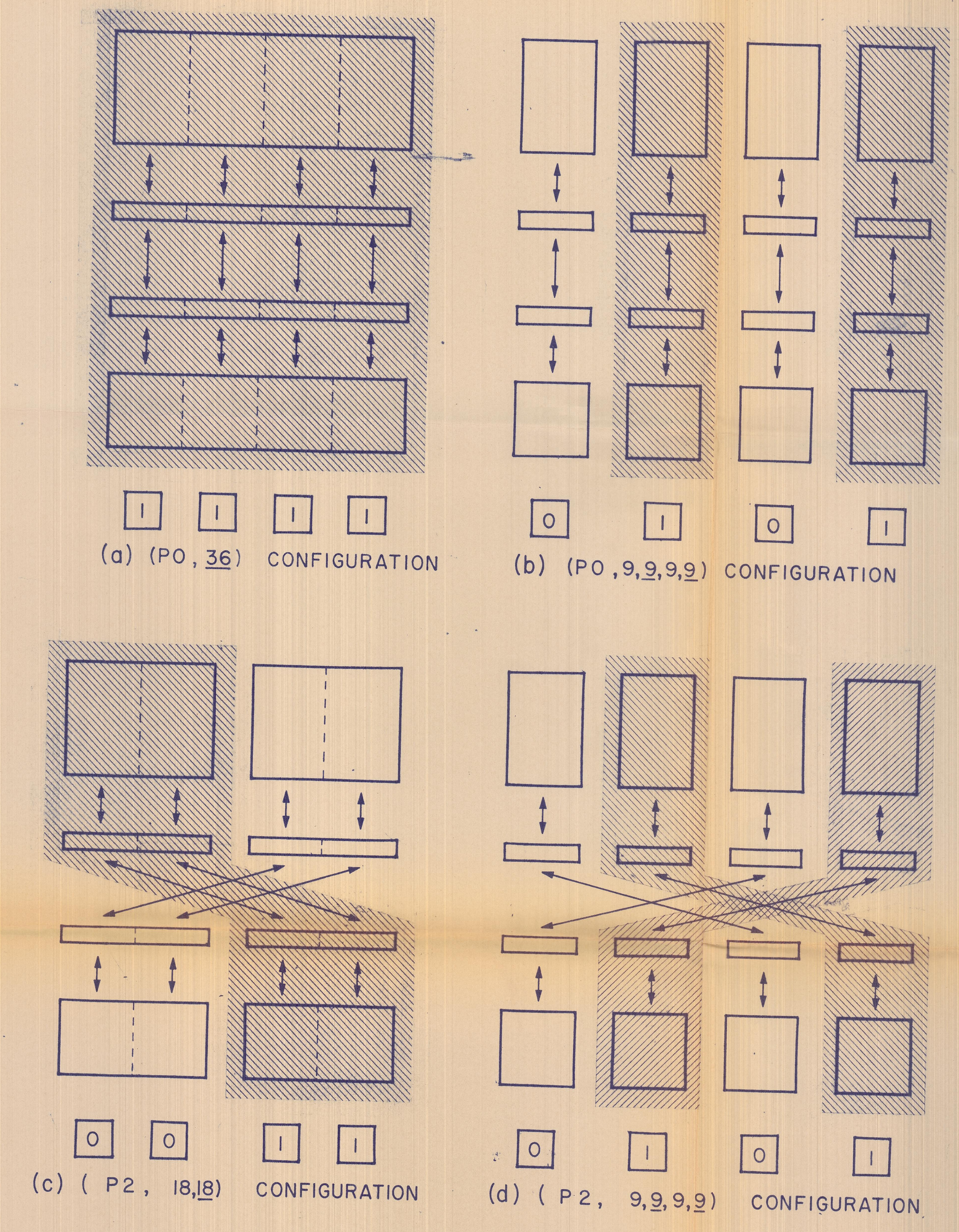
C-694П
C60-372
S-1677

Fig. 9d



TX-2 SHIFT PATH ARRANGEMENTS

Fig. 10



(a) (PO, $\underline{36}$) CONFIGURATION

(b) (PO, 9, $\underline{9}$, 9, $\underline{9}$) CONFIGURATION

(c) (P2, 18, $\underline{18}$) CONFIGURATION

(d) (P2, 9, $\underline{9}$, 9, $\underline{9}$) CONFIGURATION

TX-2 CONFIGURATIONS

AREAS OF ACTIVITY DURING EXECUTION OF INSTRUCTION SHOWN SHADED. EFFECT OF AE COUPLINGS ILLUSTRATED BY JUXTAPOSITION OF QUARTERS.

C-69385-1
S-1622 (a)(b) CG0-380
S-1623 (c)(d) CG0-381
S-1679 (a)(b) CG0-423
S-1680 (c)(d) CG0-424

THE LINCOLN TX-2 INPUT-OUTPUT SYSTEM[+]

James W. Forgie[++]
Lincoln Laboratory
Lexington 73, Massachusetts

Introduction

The Lincoln TX-2 Computer input-output system contains a variety of input-output devices suitable for general research and control applications. The design of the system has aimed at connecting these devices in such a way that a number of them may be operated concurrently. Since the computer is experimental in nature, and changes in the complement of input-output devices are anticipated, a modular scheme has been utilized to facilitate expansion and modification. The experimental nature of the computer also requires that the input-output system provide a maximum of flexibility in operating and programming for its input-output devices.

The input-output devices currently scheduled for connection to TX-2 include magnetic tape units for auxiliary storage; photoelectric paper tape readers for program input; a high-speed printer, cathode-ray tube displays, and Flexowriters for direct output; analog-to-digital conversion equipment;

[+] The research in this document was supported jointly by the Army, Navy and Air Force under contract with the Massachusetts Institute of Technology.

[++] Staff Member, Lincoln Laboratory, Massachusetts Institute of Technology.

data links with other computers; and miscellaneous special-purpose equipment. This paper will not be concerned with the details of these devices but will limit itself to a discussion of the logical incorporation of them into the system.

This paper, in describing the TX-2 input-output system, occasionally makes reference to certain aspects of the design of the remainder of the TX-2 computer. The author assumes that the reader is familiar with the general characteristics of the TX-2 computer as set forth in the paper entitled A Functional Description of the Lincoln TX-2 Computer by John M. Frankovich and H. Philip Peterson appearing in these proceedings.

The Multiple Sequence Program Technique

Of the various organizational schemes which permit the simultaneous operation of many devices, we have chosen the "multiple sequence program technique" for incorporation in TX-2. A multiple sequence computer is one having several program (instruction) counters. By arranging to have the program sequences associated with these program counters time-share the hardware of the central computer, it is possible to obtain a machine which behaves as if it were a number of logically separate computers. We call these logical computers sequences and therefore refer to TX-2 as a multiple-sequence computer. By associating each input-output device with such a sequence, we effectively obtain an input-output computer for each device.

Since the single physical computer in which these sequences operate is capable of performing only one instruction at a time, it is necessary to interleave the sequences if they are to operate concurrently. This interleaving process can take place aperiodically to suit the needs and under the control of the individual input-output devices which happen to be operating. Of course, the number of sequences which can operate concurrently and the complexity of the individual sequences is limited by the peak and average data handling rate of the central computer hardware.

In a multiple-sequence computer the main body of the computation can be carried out in any sequence, but if maximum efficiency of input-output operation is to be achieved, it is necessary to confine the bulk of arithmetic operation to a few special sequences called main sequences which have no associated input-output devices. The input-output sequences may then be kept short with the result that a large number can be operated concurrently.

Multiple Sequence Operation in TX-2

In TX-2 one-half of the index register memory has been made available for storing program counters. Thus, a total of 32 sequences may be operated in the machine. (Actually an additional sequence of special characteristics is obtained by using index register number 0 as a program counter. This special sequence will be discussed later). Some of these sequences are associated with input-output devices. Others perform functions, such as interpreting arithmetic overflows, which are called into action by conditions arising within the central computer. Finally, there are the main sequences which are intended to carry out the bulk of the arithmetic computations performed by the machine.

A priority scheme is used to determine which sequence will control the computer at a given time. If more than one sequence requires attention at a particular instant, control of the machine will go to the one having the highest priority with the highest at the top. Asterisks mark sequences which are not associated with any particular in-out device. At the top of the list is a special sequence (number 0) which will be used to start any of the other sequences at arbitrary addresses. The next two sequences interpret alarms (under program control, of course). These three sequences have the highest priorities since they must be capable of interrupting the activities of other sequences. The input-output devices follow, with high-speed, free-running units at the tops, etc. The main sequences (we anticipate three) are at the botton of the list. The priority of any sequence may be easily changed, but such changes are not under program control. Priorities are intended to remain fixed under normal operating conditions. The list totals about 25 sequences leaving eight spaces for future expansion.

Switching between sequences is under the control of both the input-output devices (generalized to include alarms, etc.) and the programmed instructions within the sequences.

Once a sequence is selected and its instructions are controlling the computer, further switching is under control of the programmed instructions. Program control of sequence switching is maintained through two bits called the _break_ and _dismiss_ bits in each instruction. The _break_ bit governs changes to higher priority sequences. When the _break_ bit permits a change, and some higher-priority sequence requests attention, a change will be made. The _dismiss_ bit indicates that the sequence has completed its operation (for the moment, at least) and that lower-priority sequences may therefore receive attention. The interpretation of the _break_ and _dismiss_ bits will be discussed in more detail in a later section.

The TX-2 Input-Output Element

The TX-2 Input-Output Element is shown schematically in Fig. 1. It consists of a number of input-output devices, associated buffers, and a Sequence Selector. Each device has enough control circuitry to permit it to operate in some selected mode once it has been placed in that mode by signals from the central computer. Associated with each device is a buffer storage of appropriate size. This buffer may be large or small to suit individual data rate requirements, but in general, the buffers to be used in TX-2 will be the smallest possible. For the most part, buffering for only one line of data from the device (e.g., 6 bits for a paper tape reader) will be provided. Each input-output device also has associated with it one stage of the Sequence Selector. The Sequence Selector provides the control information necessary for proper interleaving of the program sequences. When it is desired to add a new input-output device to the computer, the three logical packages: in-out unit, buffer, and Sequence Selector stage, must be provided.

As shown in Fig. 1, data transfers between the input-output element and the central computer go by way of the Exchange Element. Fig. 1 indicates two-way paths between the E Register and all in-out buffers. Actually, most devices are either readers or recorders, but not both, and therefore require one-way paths only. Naturally, only the necessary paths are provided. The drawing simply shows the most general case.

Signals from the Sequence Selector connect the appropriate buffer register to E for data transfer purposes. When a sequence is selected (i.e., its program counter is supplying instruction locations), the associated buffer is connected to the E Register, and all other buffers are disconnected. The occurrence of a read instruction will then effect a transfer of information between the buffer and E. A particular buffer is thus accessible only to read instructions occurring in the sequence associated with the buffer's in-out unit.

Fig. 1 shows paths from the Sequence Selector to a coder which proves an output called the program counter number. These paths are used in the process of changing sequences to be described in a later section.

Fig. 1 also shows paths for mode selection in the In-Out Element. The utilization of these paths is described in the next section under the operation of the ios instruction.

Input-Output Instructions

In addition to the break and dismiss bits on all instructions, the programmer has three computer instructions for operating the input-output system. There are two read instructions rdn and rds which effect data transfers between the in-out devices and the central computer memory. The third instruction, ios, is used to select the mode of operation of the in-out devices.

Both read instructions cause a word to be obtained from memory. If the in-out device associated with the sequence in which the read instruction occurs is in a reading (input) mode, appropriate bits of the memory word are altered, and the modified word is replaced in memory. If the in-out device is in a recording (output) mode, appropriate bits of the memory word are fed to the selected in-out buffer, and the word is replaced in memory. Thus, the same read instruction suffices for both input and output operations. The distinction between rdn and rds lies in the process of assembling full memory words from short buffer words. An rdn instruction will place the 6 bits from a tape reader in the right 6 bits of a 36-bit memory word. The remaining 30 bits will be left unchanged. An rds instruction for the same tape reader will place the 6 bits in a splayed pattern (every sixth bit across the memory word) and will shift the entire word one place to the left before replacing it in memory. Except for the shift, the other 30 bits remain unchanged. A sequence of 6 rds instructions, one for each of 6 tape lines and all referring to the same memory address, will suffice to assemble a full 36 bit word.

The distinction between rdn and rds could be obtained from mode information in the in-out device, but the inclusion of both instructions in the order code allows the programmer to interchange the two types freely to suit his needs. The rdn instruction makes use of the permutation aspect of TX-2 configuration control and is therefore particularly convenient for dealing with alphanumeric Flexowriter characters. Configuration is not applicable to the rds instruction.

The ios instruction serves to put a particular in-out device into a desired mode of operation. The j-bits of the instruction word, normally the index register number, in this case specify the unit number of the in-out device. This number is the same as the program counter number for the associated sequence although the correspondence is not one of necessity. The y-bits of the instruction word specify the mode of operation in which the unit is to be placed. Two of the y-bits are sent directly to the jth Sequence Selector stage and serve to control the sequence independent of the mode of its associated in-out device. These two bits allow ios instructions to arbitrarily dismiss or request attention for any sequence in the machine. By means of these instructions one sequence can start or stop all others in the machine. A third y-bit determines whether the mode of the in-out device is to change as a result of the instruction. If it is to change, the remaining 15 bits specify the new mode. An ios instruction occurring in any sequence can thus start or stop any sequence and/or change the mode of its in-out device.

A further property of the ios instruction is that it leaves in the E Register a map of the state of the specified in-out control prior to any changes resulting from the instruction itself. ios instructions may therefore be used to sense the state of the in-out system without altering it in any way.

The Process of Sequence Changing and Sequence Selector Operation.

At some point just before the completion of the instruction memory cycle in TX-2, the Control must decide whether the next instruction would be taken from the current sequence or from some new sequence. The information on which this decision must be based comes from the break and dismiss bits of the current instruction word and from the Sequence Selector. Fig. 2 shows a detailed drawing of one stage of the Sequence Selector. All stages except the highest-priority one are identical. The lowest-priority stage returns the final three control signals to the Control Element.

Each Sequence Selector stage retains two pieces of information concerning its associated sequence. One flip-flop (SS j.1) remembers whether or not the sequence is selected (i.e., whether or not it is receiving attention). The priority signal (labelled no higher priority sequence requests attention) passes from higher to lower priority stages until it encounters a stage which requests but is not receiving attention. Such a stage is said to have priority at the moment, and its output to the program-counter-number coder prepares the number of the new program counter in anticipation of a sequence change.

The process of changing sequences involves storing the program counter for the old sequence and obtaining the counter for the new. Actually, to speed up the overall process, the new program counter is obtained first, so that it may be used while the old is being stored. Using the paths shown in Fig. 1, the new program counter number is placed in the j-bits of the N Register. The new program counter is then obtained from the X Memory and interchanged with the old program counter contents which have been in the P Register.[1] The K Register, which has been holding the old program counter number since the last sequence change, is now interchanged with the j-bits, and the old counter is stored at the proper X Memory location. The state of the Sequence Selector is changed to conform to the change of sequence by sending a Select New Sequence command from Control. This command clears the ss j.2 flip-flop in the old-sequence stage and sets the ss j.2 flip-flop to a one in the new-sequence stage.[2]

Interpretation of the Break Bit

The break bit of an instruction word is utilized by the programmer to indicate whether or not a change to a higher priority sequence may occur at the completion of the instruction. The fact that a programmer permits a break does not mean that the sequence has completed its current task but merely that no harm will be done if a change to some higher-priority sequence is made. Breaks should be permitted at every opportunity if a number of in-out devices are operating. The sort of situation in which a break cannot be permitted occurs when the E Register is left containing information which the program requires at a later step. If a change occurred in this case, the E Register contents would be destroyed and consequently lost to the program.

When a break is permitted by the current instruction, a sequence change will actually take place only if some higher-priority sequence requests attention. A signal from the Sequence Selector to Control provides this information (Fig. 2). When a break type of sequence change is made, the SS j.1 flip-flop in the Sequence Selector remains unchanged.

As a result the sequence which was abandoned in favor of some higher-priority one continues to request attention.

Interpretation of the Dismiss Bit

The dismiss bit is used by the programmer to indicate that the current sequence has completed its task. To provide synchronization in the in-out system, dismiss bits must be programmed between attention requests from the in-out devices. In this case the dismiss in-operation guarantees that the computer will wait for the next signal from the in-out device before proceeding with the associated program sequence.

The dismiss bit is also used to accomplish the halt function in TX-2. The halted state of a multiple-sequence computer results when all sequences have been dismissed and all in-out units turned off. The priority signal from the Sequence Selector to Control provides the information as to whether or not any sequence in the machine requests attention. When none request attention, the Control stops all activity in the machine as soon as a dismiss bit appears on an instruction in the current sequence. Activity is resumed in the machine as soon as some in-out device or push button requests attention.

The sequence change which results from a dismiss bit is identical with that resulting from a break except that a dismiss current sequence command accompanies the select new sequence command from Control to the Sequence Selector (Fig. 2).

Starting a Multiple-Sequence Computer

In a single sequence computer the starting process involves resetting the program counter to some arbitrary value and starting the control. In a multiple-sequence computer the program counter for a particular sequence must be reset and the sequence started. In TX-2 a special sequence (number 0) having the highest priority is used to facilitate starting. This sequence has the special feature that its program counter always starts at an initial memory location specified by a set of toggle switches. Attention for the sequence is requested by pushing a button on the console. By executing a short program stored in the toggle-switch registers of the V Memory, this sequence can start (or stop) any other sequence in the machine. The starting process for an arbitrary sequence involves resetting its program counter by means of an ldx (load index register) instruction, and starting its sequence with an ios instruction.

The Use of the Arithmetic Element in Multiple-Sequence Operation

While efficient operation requires that the bulk of arithmetic operations be carried out in a main sequence, the arithmetic element in TX-2 is available to all sequences. Since once a change has been made to a higher-priority sequence, control cannot return to a lower-priority sequence, until the higher-priority one has been dismissed, a simple rule allows the arithmetic element to be used in any sequence without confusion. If whenever a higher-priority sequence requires the arithmetic element it stores the contents of any registers it will need (A, B, C, D, or F) and reloads them before dismissing, all lower priority sequences will find the registers as they left them. This storing and loading operation requires time and therefore lowers the total handling capacity, but the flexibility obtained may well be worth the loss in capacity.

A special problem results from the step-counter class of arithmetic element instructions. These instructions can require many microseconds to complete, and while TX-2 is designed to allow in-out and program element instructions to take place while the arithmetic element is busy, the case can arise in which an arithmetic element instruction (load, store, etc.) appears before the AE is finished with a step-counter class instruction. In this case the machine normally waits in an inactive state until the operation is complete, but since there is a chance that some higher-priority sequence may request attention in the interim and have instructions which can be carried out, provision is made to keep trying changes to higher-priority sequences as they request attention. The machine thus waits in an inactive state only when no higher-priority sequences have instructions which can be performed. This provision allows the programmer to ignore the arithmetic element in considerations of peak and average rate calculations when he desires to operate a maximum number of in-out devices.

Conclusions

Multiple sequence operation of input-output devices as realized in TX-2 has a number of significant characteristics. Among them are:

1. A number of in-out devices may be operated concurrently with a minimum of buffering storage.
2. Machine time is efficiently utilized since no time need be lost waiting for input-output devices to complete their operation. Other machine activity may proceed meanwhile.
3. Each input-output device may be treated separately for programming purposes. Efficiency of operation is obtained automatically when several separately-programmed devices are operated concurrently. Average and peak rate limitations must, of course, be considered.

4. A maximum of flexibility in programming for input-output devices is obtained. The full power of the central machine may be used by each input-output sequence if desired. Routines for each device may be as long or as short as the particular situation requires.

5. The modular organization of the input-output equipment permits simple additions and modifications to the complement of in-out devices.

6. The organization of buffering storage allows the amount and kind of such storage to be tailored to the needs of the individual devices and the data-handling requirements to be met by the system.

7. The multiple-sequence program technique appears to be particularly well suited to the operation of a large number of relatively slow input-output devices of varying characteristics as opposed to a smaller number of high-speed devices.

TABLE I

TX-2 Sequence Assignments in the Order of Their Priority

+ Start Over (special index register number 0 sequence)

+ In-out alarms

+ Arithmetic alarms (overflows, etc.)

Magnetic Tape units (several sequences)

High-speed printer

Analog-to-digital converter

Photoelectric paper tape readers (several sequences)

Light Pen (photoelectric pick-up device)

Display (several sequences)

MTC (Memory Test Computer)

TX-O

Digital-to-analog converter

Paper tape punch

Flexowriters (several sequences)

+ Main sequences (three)

+ Indicates that the sequence has no input-output device.

FOOTNOTES

1. The P Register is shown in Fig. 4 of the previously mentioned paper by Frankovitch and Peterson, page of these proceedings.

2. The relative timing of the central computer actions during the change process is shown in Fig. 6D of the paper by Frankovitch and Peterson, page of these proceedings.

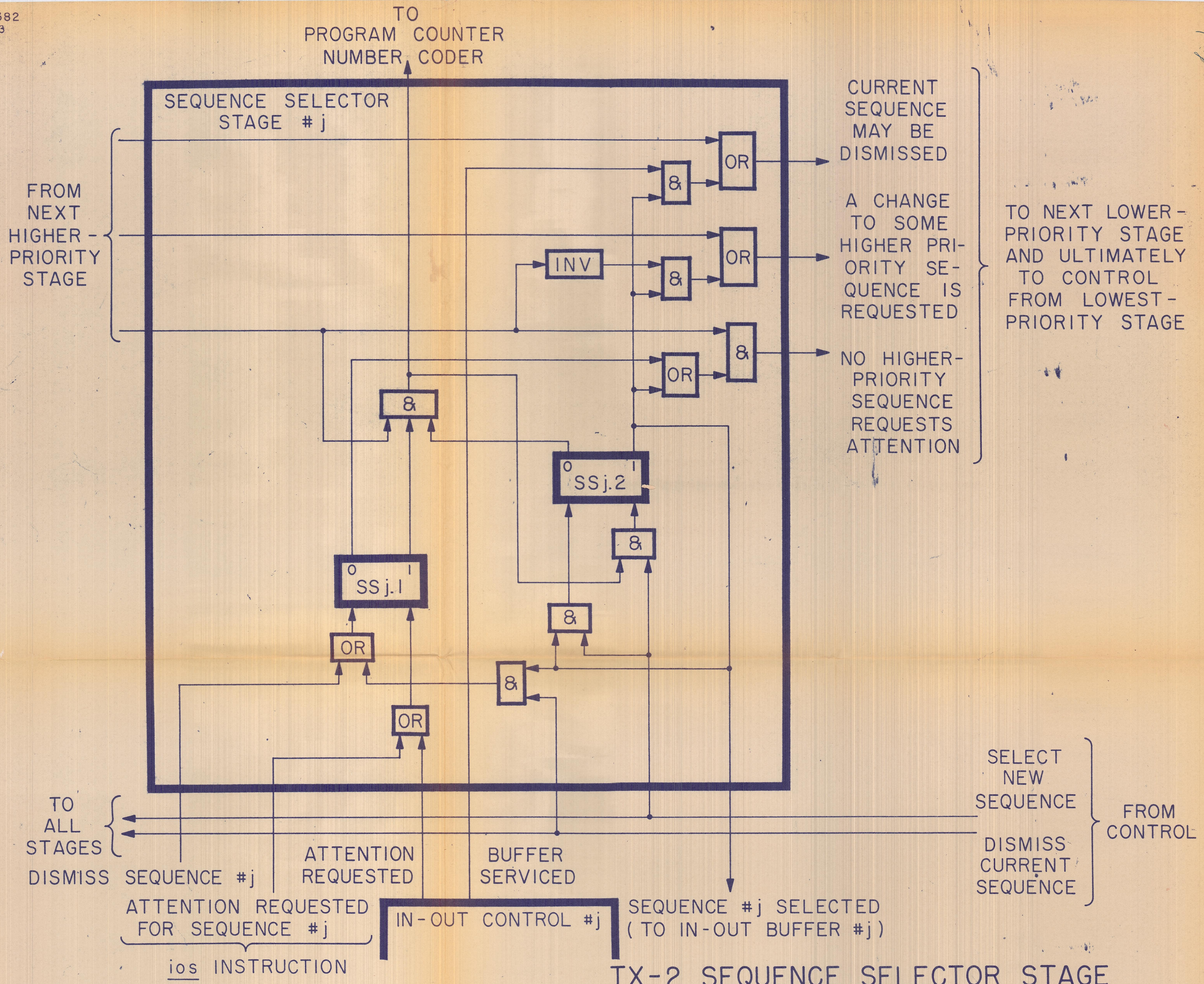
CAPTIONS FOR FIGURES

Fig. 1 - Block Diagram of TX-2 In-Out Element

Fig. 2 - Block Diagram of TX-2 Sequence Selector Stage

C-69382
C60-373
S-1717

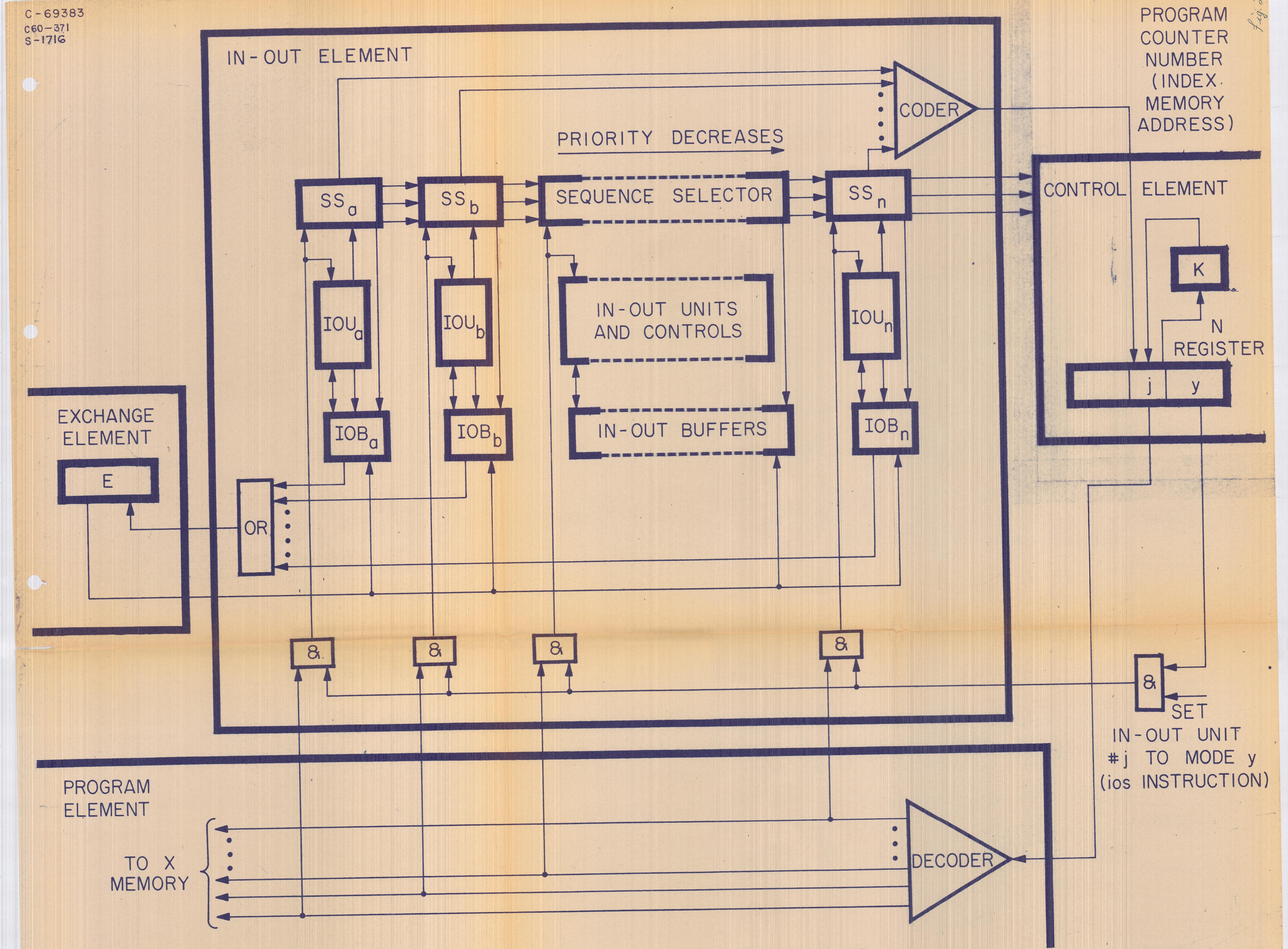
fig. 1



TX-2 SEQUENCE SELECTOR STAGE

C-69383
C60-371
S-1716

fig 2



TX-2 IN-OUT ELEMENT

A SHORT ACCESS TIME MEMORY USING TWO CORES PER BIT*

Richard L. Best**

A single computer can do the work of two or more separate computers if it can operate simultaneously several programs and pieces of input-output equipment.[1,2] The single computer needs only enough index registers and program counters to schedule and control its operations. Live flip-flop registers could be used, but their expense would limit the number of such registers.

A ferrite-core memory with a capacity of sixty-four 19-bit words, an access time of 0.8 μsec, and a cycle time of 4 μsec has been developed at Lincoln Laboratory. The ferrite cores are 47 mils O.D., 27 mils I.D., and 12 mils thick.

Access time is the minimum delay between setting the address register and strobing, while cycle time is the time between successive strobes with a repetitive READ-WRITE cycle. The short access time allows frequent reference to the memory without reducing machine speed. In the Lincoln TX-2 computer for which this memory is designed, the speed is mainly limited by the 6.5-μsec cycle time of the 65,536-word memory that stores the bulk of the machine information.[3]

OPERATING PRINCIPLE

The operating path on the hysteresis loop of the ferrite core material used is shown in Fig. 1. The READ time is much shorter than the WRITE time because of the much greater current used for the READ operation.

*The research reported in this document was supported jointly by the Army, Navy, and Air Force under contract with the Massachusetts Institute of Technology.

**Staff Member, Lincoln Laboratory, Massachusetts Institute of Technology.

The winding configuration of the single plane unit is shown in Fig. 2. The vertical lines are the "word" lines and the others are "digit" lines. A word is selected externally by a switch which connects the upper end of the selected word line (e.g., Y) to the -3 volt supply and leaves the others (W, X, Z) floating. The READ driver applies a negative current pulse of amplitude 4 1/3 I to the common word line junction at the bottom. This current, represented by the waveforms of Fig. 3 flows through all the cores on the selected word line (Y). For example, cores A and B are the cores for a given word bit. Normally, only one of these cores could be left in the SET condition by the previous WRITE operation in that register so that only one of themis switched to the CLEAR state and thus generates a voltage pulse that appears at the terminals of the digit line. The digit line, however, links one of the cores (A or B) in the same direction as the word line, and the other core in the opposite direction. Thus, the polarity of the pulse appearing at the terminals of the digit line indicates which core was switched, and whether a ONE or a ZERO was read.

A direct current or amplitude 1/3 I always flows in the digit line, as shown in Fig. 3. It is small enough so that it has negligible effect during READ time; one core sees 4I and the other 4-2/3I. The polarity of a given digit current is controlled by the buffer register flip-flop associated with that digit. During WRITE, a current of plus 2/3I is sent down the selected word line. The digit current of 1/3I adds to the WRITE current in one core and subtracts from it in the other, so that one core has a current of I and the other a current of 1/3I. Thus, the current ratio used during WRITE is 3:1 with a disturb current (current in unselected cores) of no more than 1/3I.

The WRITE current is 108 ma-turns and the switching time is 2 μsecs; the READ current (depending on the polarity of the digit current) is either 4 or 4-2/3 times this with a switching time of 0.3 μsecs. Since each of the two windings makes four turns on each core through which it passes, the digit current is only 8 ma, the write driver output current 18 ma, and the read driver output current 117 ma. Fig. 4 shows the complete memory plane (4-1/4" x 6-1/4") and Fig. 5 shows a portion of it enlarged.

MODES OF OPERATION

There are three modes of operation of the X memory: (1) READ-WRITE, (2) READ, and (3) CLEAR-WRITE. READ-WRITE has been described above. The READ operation, used when the contents of two registers are needed quickly, performs the necessary function of clearing both cores in each bit before writing. When the computer returns to WRITE in registers that have had a READ cycle only, the CLEAR-WRITE cycle is used. CLEAR-WRITE is the same as READ-WRITE except that the strobe pulse is eliminated. Actually, a WRITE cycle alone would be sufficient, but the CLEAR-WRITE cycle was added as an aid to program trouble-shooting, since if a WRITE operation should follow a previous WRITE operation on the same register, some bits would have both cores set. A subsequent READ would clear both cores, their outputs would subtract in the digit winding, and the response of the sense amplifier would be unpredictable.

The block diagram is shown in Fig. 6. The particular method of word selection used is determined partially by the computer's use of the outputs of the decoder in the exchange and in-out elements.[1,2] Two write drivers are used and the output of the first level selection determines which one is used.

SELECTION CIRCUITS

One channel of the selection circuit is shown in Fig. 7. The decoder uses 5-way emitter follower AND gates (QI-5) which drive parallel inverters (Q6 and Q7). The collector load of these transistors is such as to provide an overdrive of base current into Q8 or Q9 during both selection and deselection. When neither read nor write driver is active, the word lines are free to float between 0 and -10 volts. Only one of the first level selection transistors (Q10 or Q11) will be saturated, so base current flows only into either Q8 or Q9. The read driver generates a negative pulse so that the large read current (117 ma) flows in the normal direction in the 2N123's. The write current flows in the reverse direction, but it is only 18 ma, and doesn't require a very high reverse β. It would have been more economical of transistors to decode in two steps, but access time is at a premium here, so that the faster circuit was used.

READ-WRITE DRIVERS

The read driver shown in Fig. 8 consists of three SBT transistors in series (because of the voltage needed) driving a 6197 to saturation. The back voltage presented by the cores to this driver is constant because, as mentioned before, it always switches one of the two cores in each pair. The 5:1 transformer holds the tube load to a low value.

The write driver (Fig. 9) is very simple - the current in the 1640-ohm resistor is switched into the memory load during WRITE by saturating Q2 which cuts off Q1. Since the selection circuits are returned to -3 volts, the output terminal of this circuit is always below ground.

DIGIT CIRCUITS

The digit driver (Fig. 10) is connected directly to the corresponding flip-flop in the buffer register. One of the two transistors is always saturated so that current flows in the digit winding and in a direction determined by the flip-flop. The terminals of the digit winding are connected to input stage (Q1 and Q2) of the sense amplifier shown in Fig. 11 which responds to the voltage difference between the inputs. The open circuit READ signal on the digit winding is a 1/4 microsecond positive or negative 1/2 volt pulse. The sense amplifier loads the winding to reduce the pulse to about half this amplitude. A saturation signal is fed to the gates Q3 and Q5 so that the strobe pulse forces the flip-flop to the correct position. If the signal on the free end of the digit winding is positive the flip-flop is left in the same state; if it is negative the flip-flop is complemented.

HARDWARE

Figs. 4 and 5 show the complete memory plane (4 1/4 x 6 1/4 inches) and an enlarged portion of it. The cores are mounted in slots on a lucite plate; the wires pass through openings made by the intersection of milled slots on one side of the plate with similar slots on the other side milled at right angles to the first. Each winding makes 4 turns per core, and is sewed through the cores in pairs.

Fig. 12 shows the memory plane mounted in position among the plug-in circuits. The wires fanning out from the left side of the plane go to the word selection circuits. The address register isn't shown. The twisted pairs fanning out from the top and bottom of the plane go to plug-in units each of which contains a digit driver and sense amplifier. The buffer register is to the right of the plane, and the two right-most columns contain logic circuitry necessary when this memory is incorporated in the TX-2 computer.

CONCLUSION

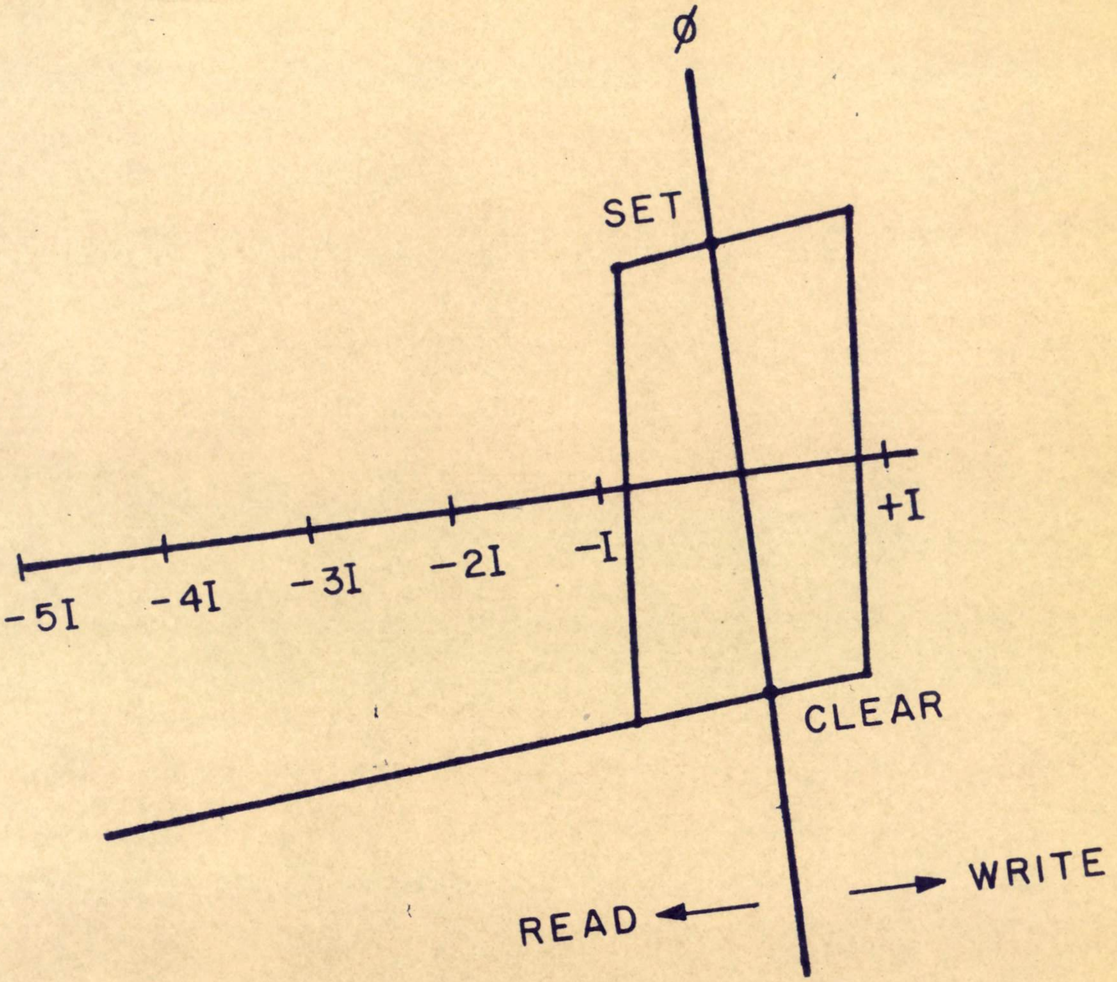
A short access time ferrite-core memory will be used to store index registers and program counters so that the Lincoln TX-2 computer will be able to operate simultaneously several program sequences and input-putput devices. Two cores per bit and external word selection allow a 3:1 current ratio during WRITE and a large overdrive during READ to greatly reduce READ time.

REFERENCES

1. "Functional Description of the Lincoln TX-2 Computer", J. M. Frankovich, Proceedings of the Western Joint Computer Conference, February 1957.

2. "The Lincoln TX-2 Input-Output System", J. W. Forgie, Proceedings of the Western Joint Computer Conference, February 1957.

3. "Part I, the TX-0 Memory", J. L. Mitchell, Proceedings of the Eastern Joint Computer Conference, December 1956.

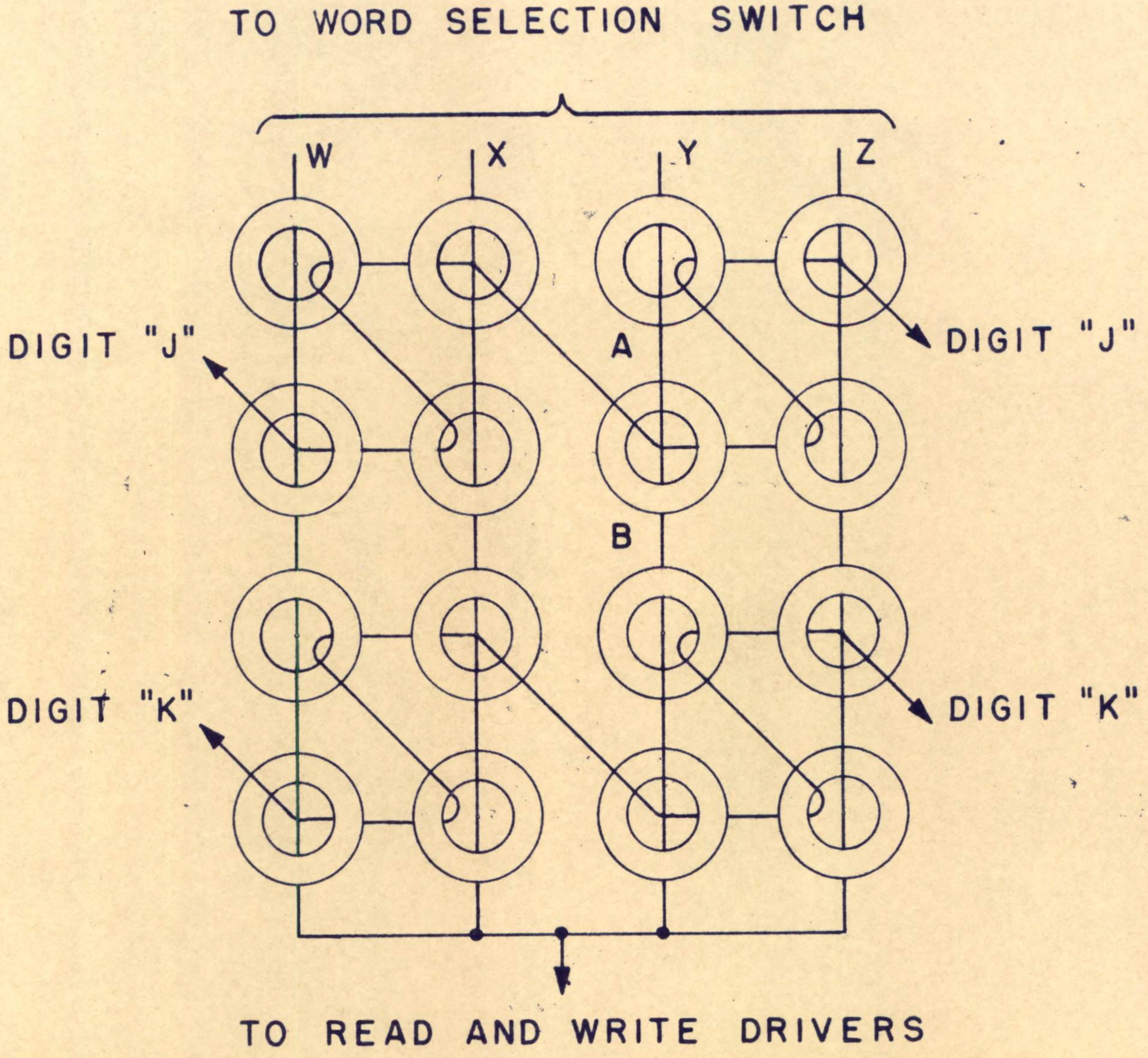


HYSTERESIS LOOP SHOWING OPERATING PATH

A-69625
C60-399

Fig. 1

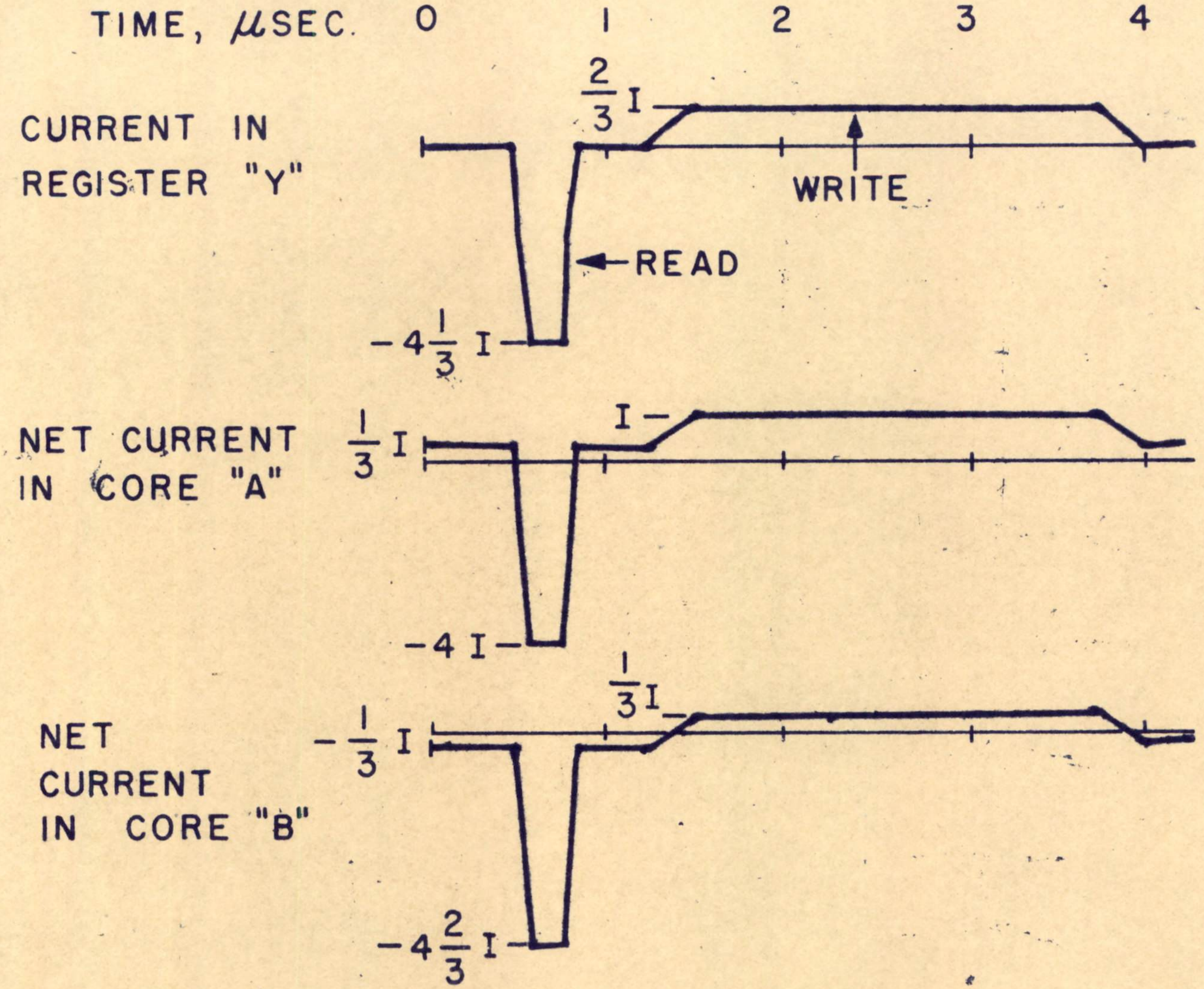
A-69363-1
C60-347
SN-1682
C60-420



WINDING CONFIGURATION, INDEX MEMORY

Fig. 2

A-69364
C60-348
S -1692



TIMING DIAGRAM, INDEX MEMORY

Fig. 3

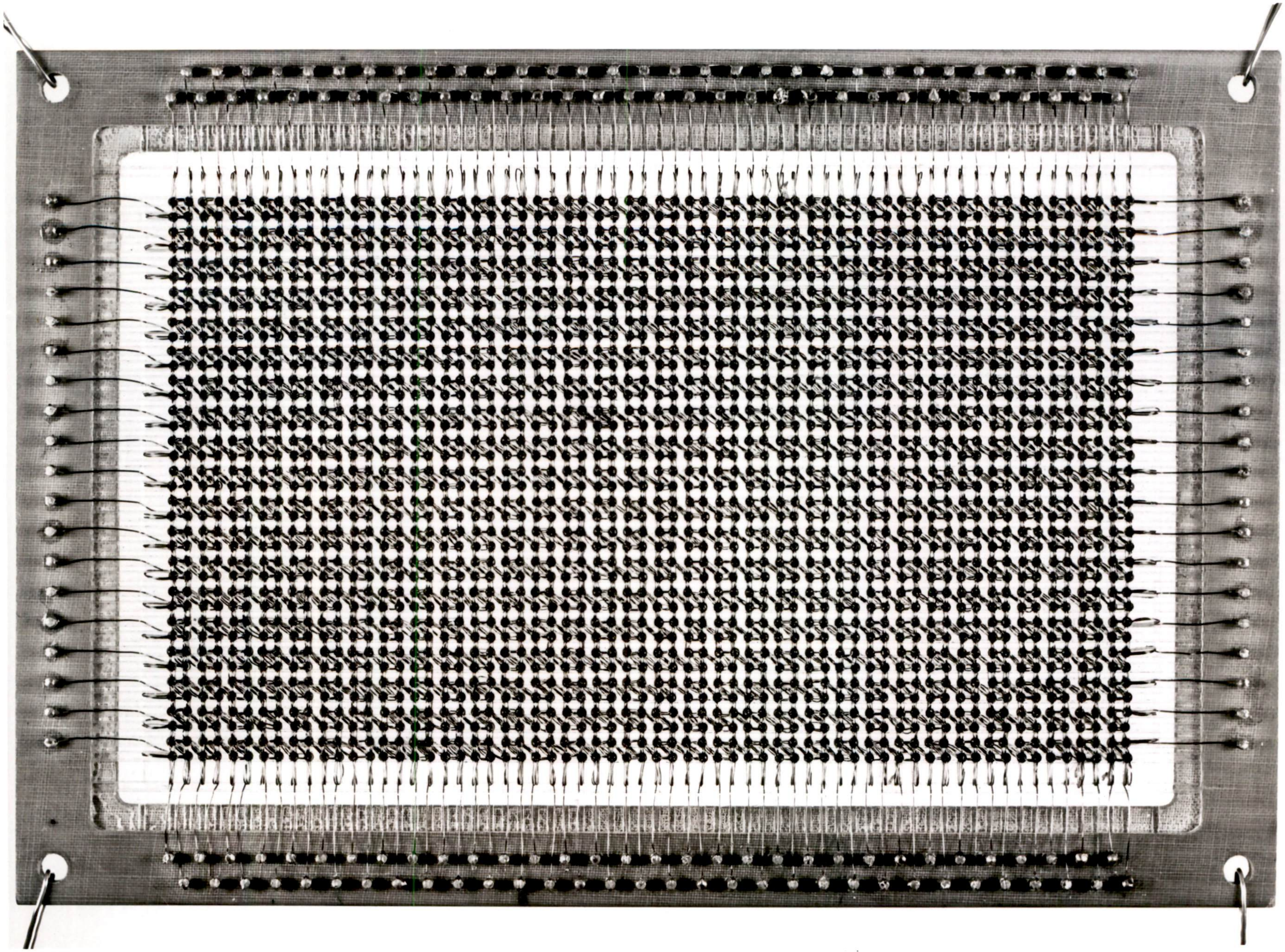


Fig. 4

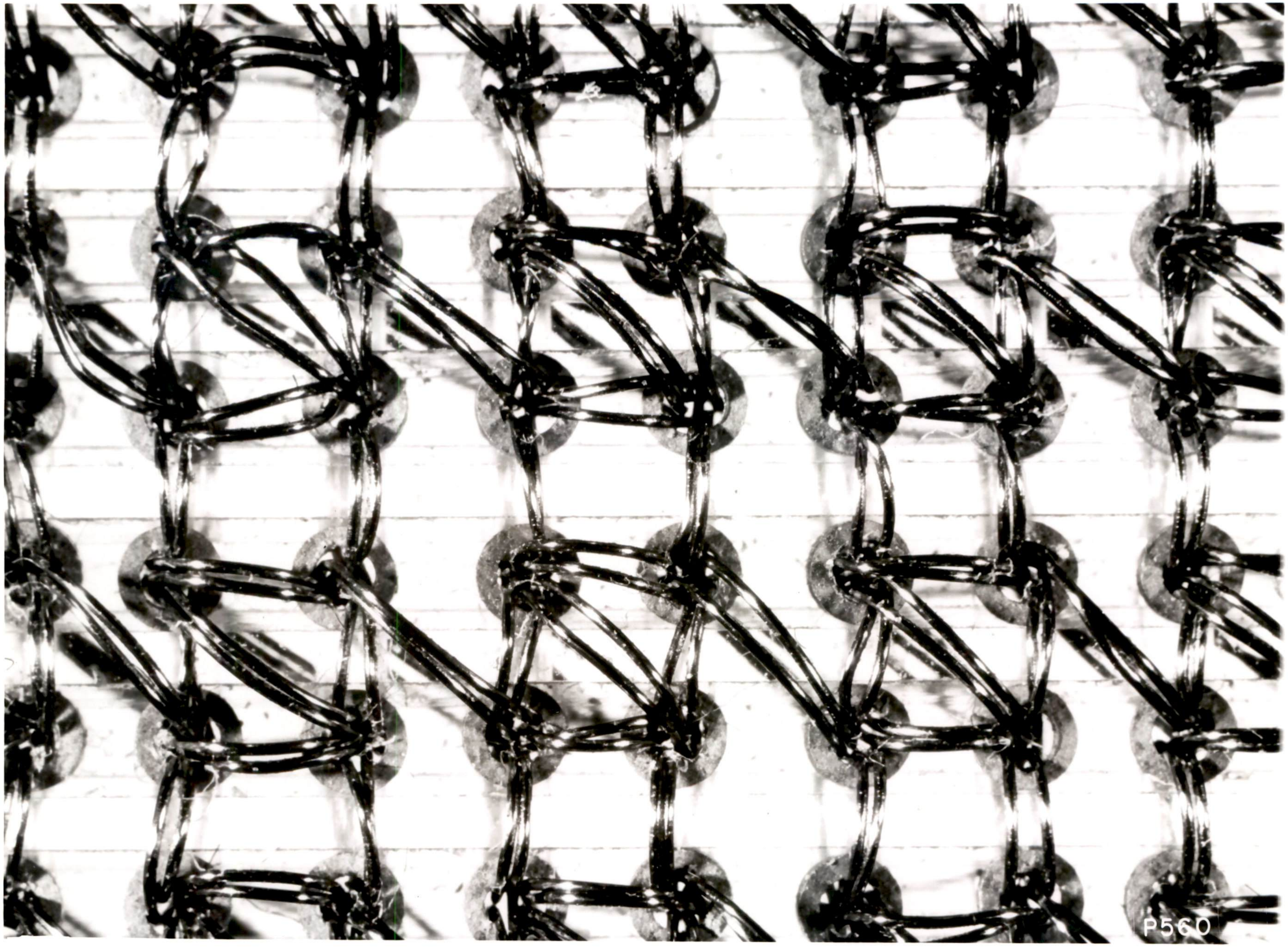
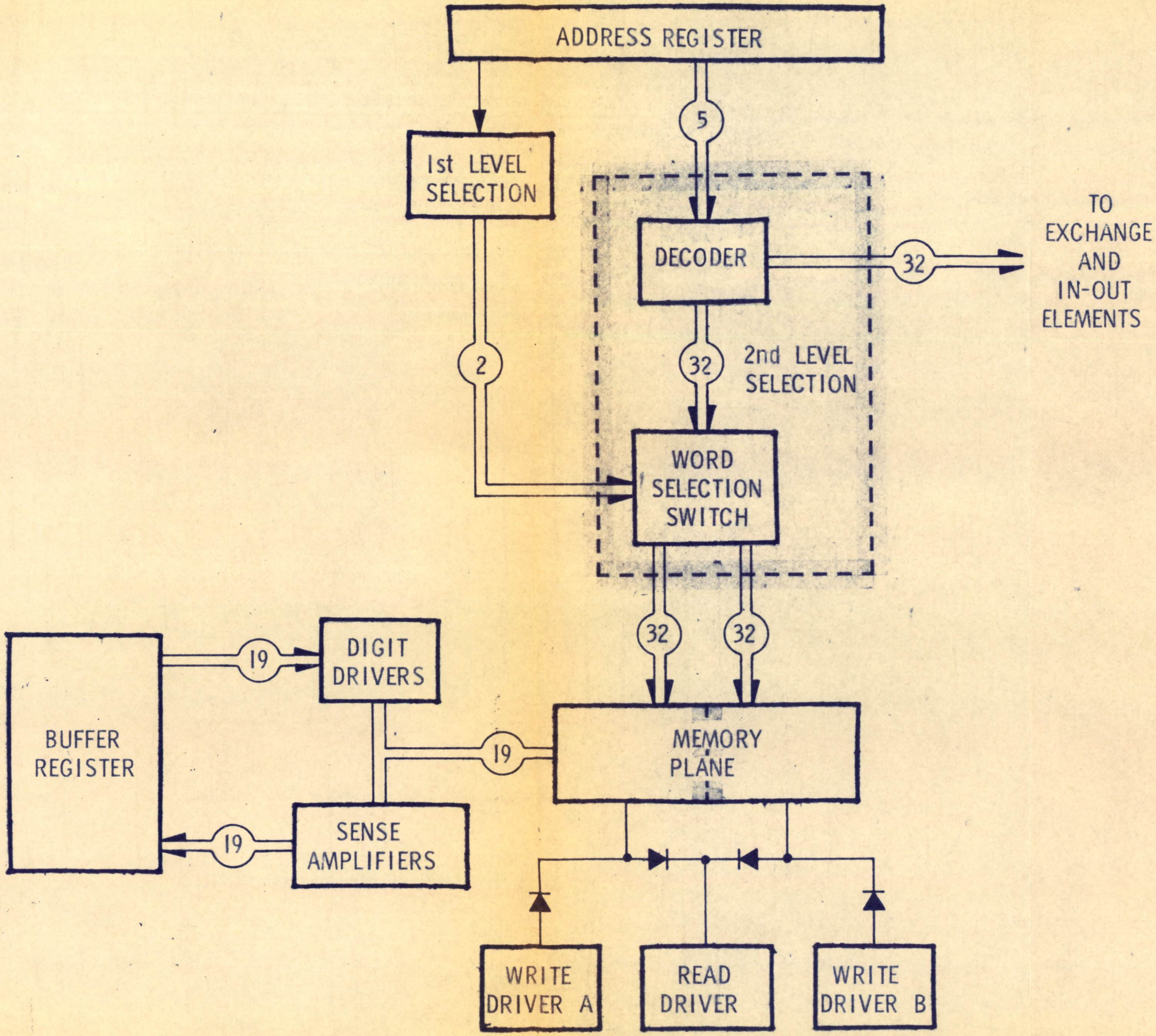


Fig. 5

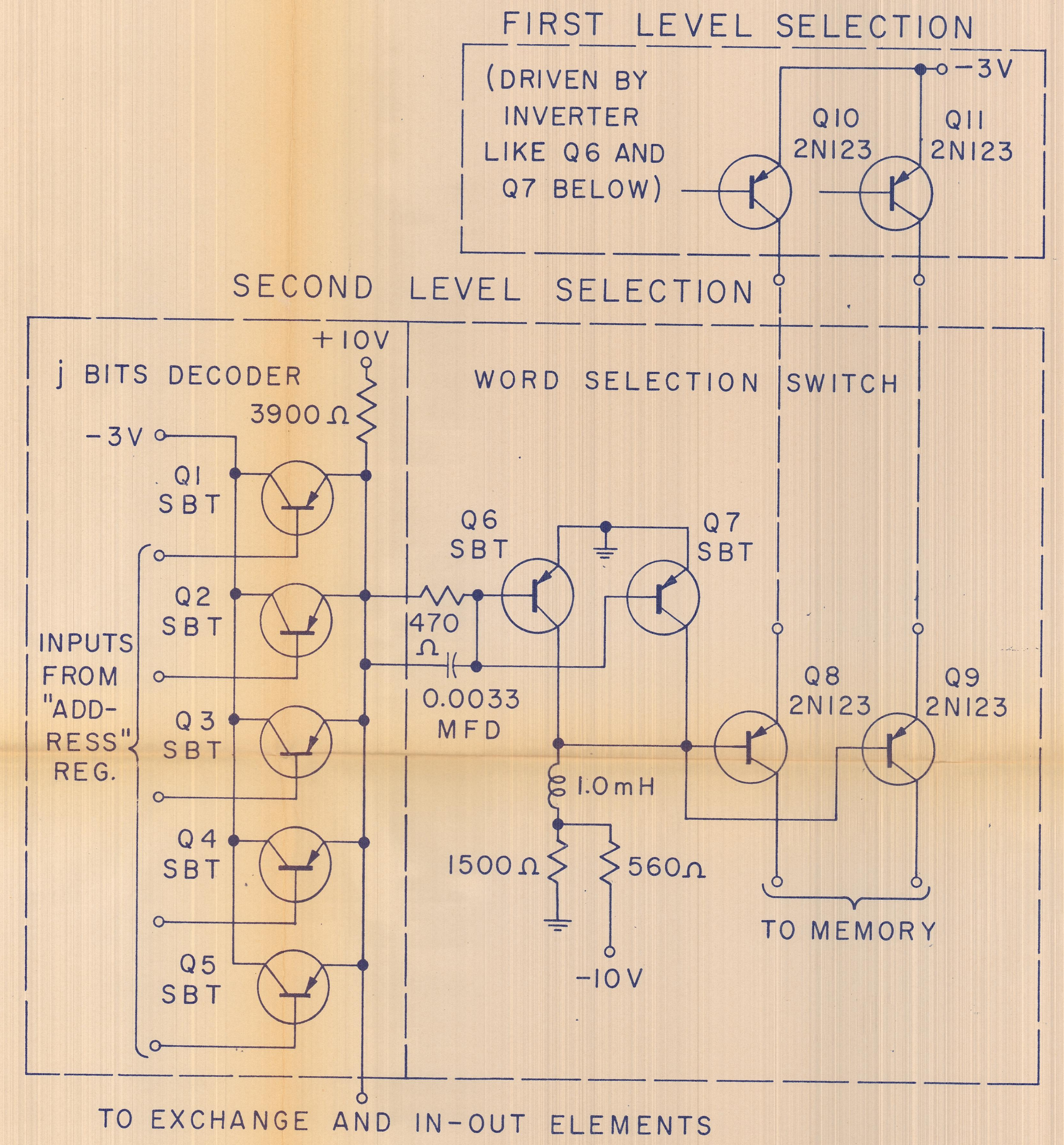
B-47247-1
C60-363
C60-395
S-1693



BLOCK DIAGRAM, X MEMORY

Fig. 6

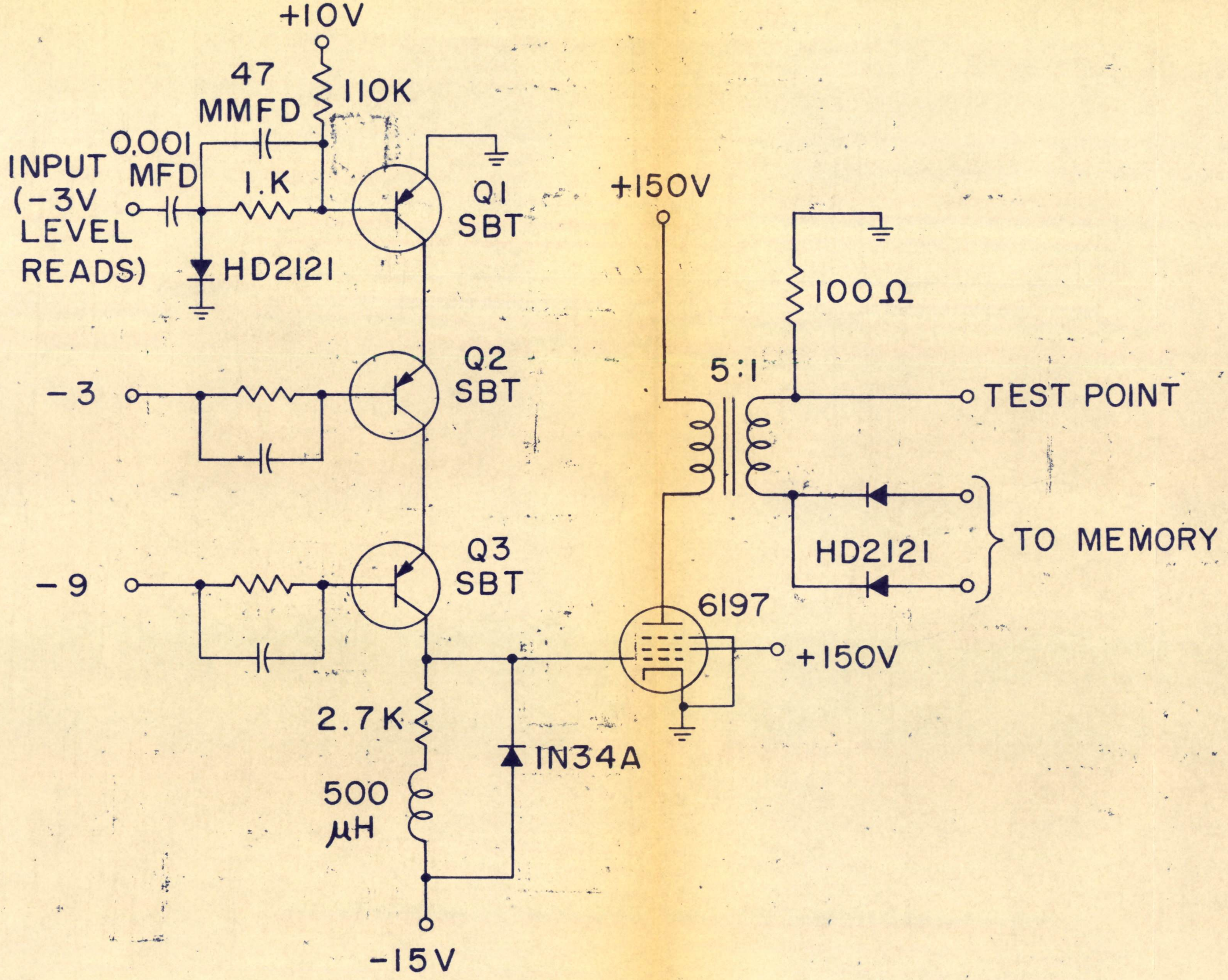
C-69406-1
C60-369
C60-398
S-1685



REGISTER SELECTION CIRCUIT, X MEMORY

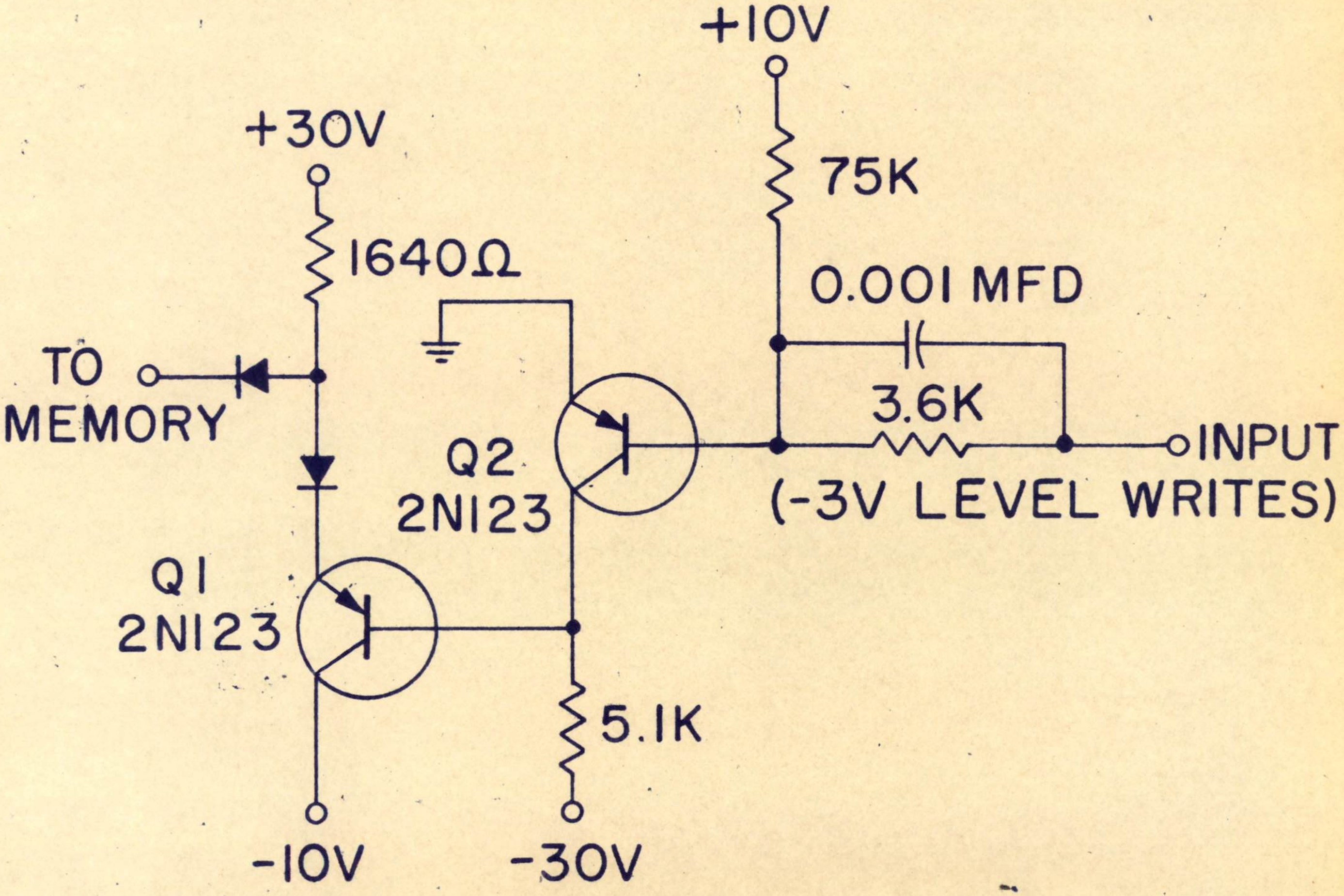
Fig. 7

B-69407-1
C60-366
S-1696



READ DRIVER, X MEMORY

Fig. 8

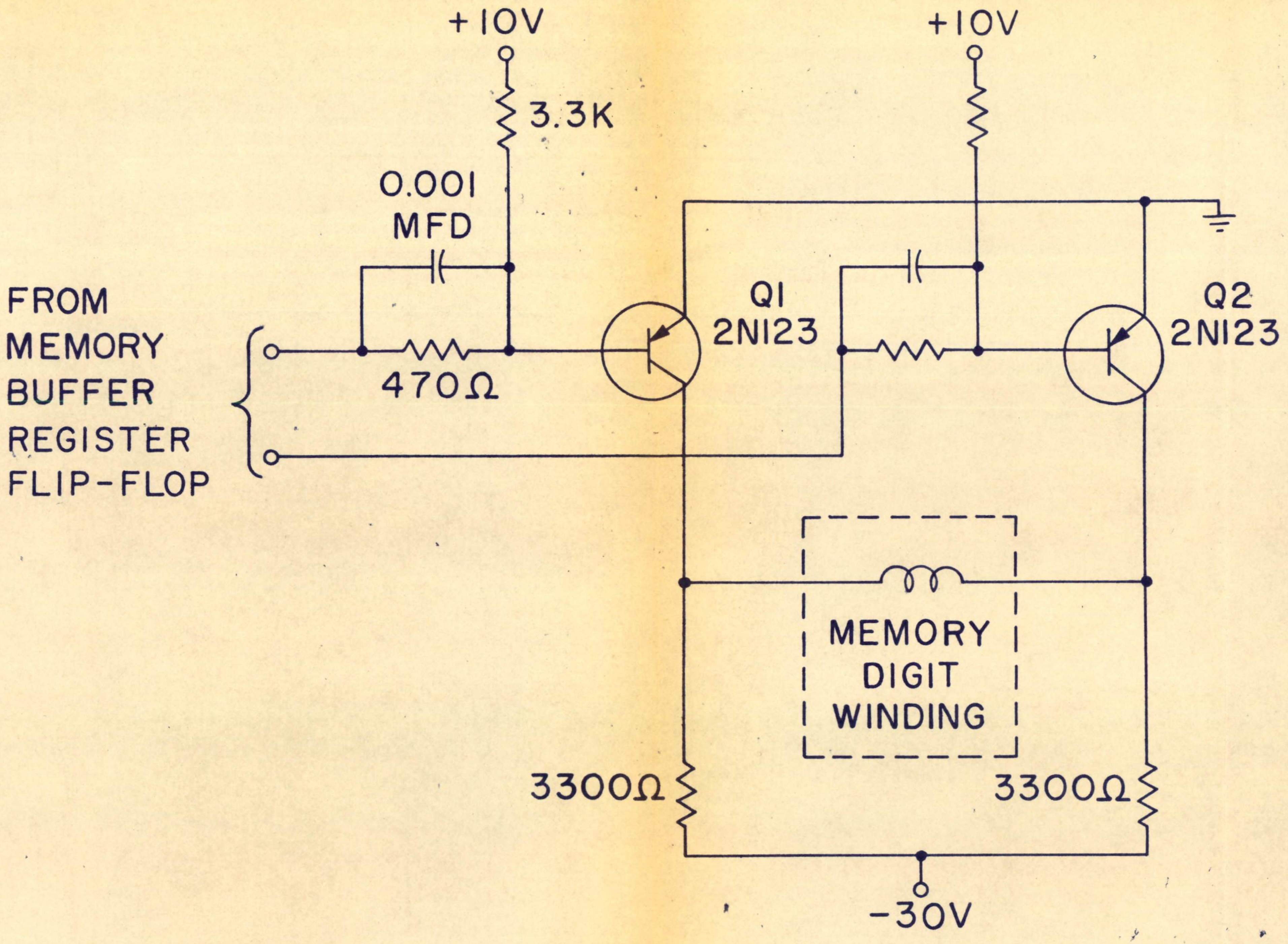


WRITE DRIVER, X MEMORY

Fig. 9

A-69384
C60-355
S-1689

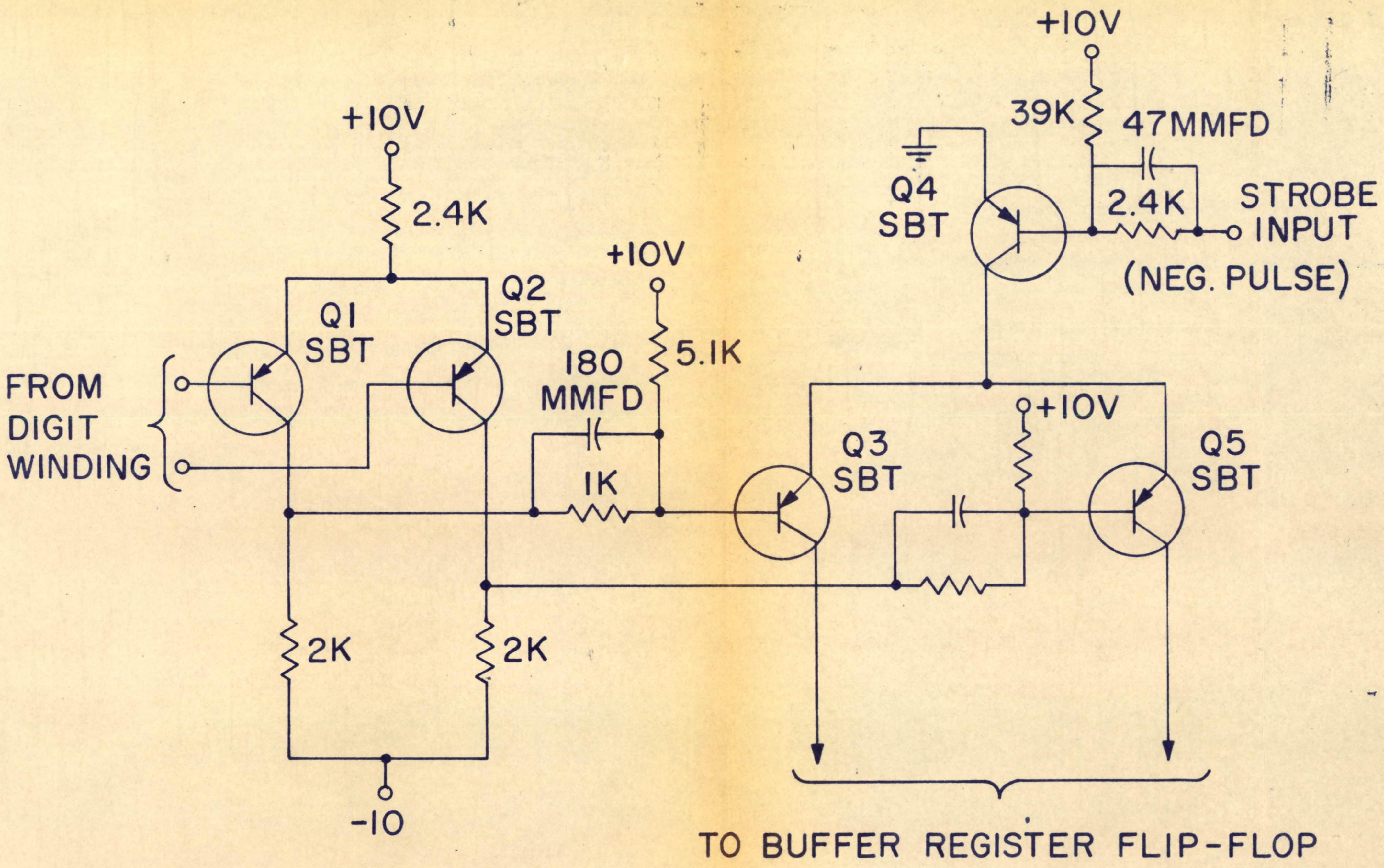
B-69405-1
C60-365
C60-397
S-1697



DIGIT DRIVER, X MEMORY

Fig. 10

B-69394-1
C60-364
C60-396
S-1698



SENSE AMPLIFIER, X MEMORY

Fig. 11

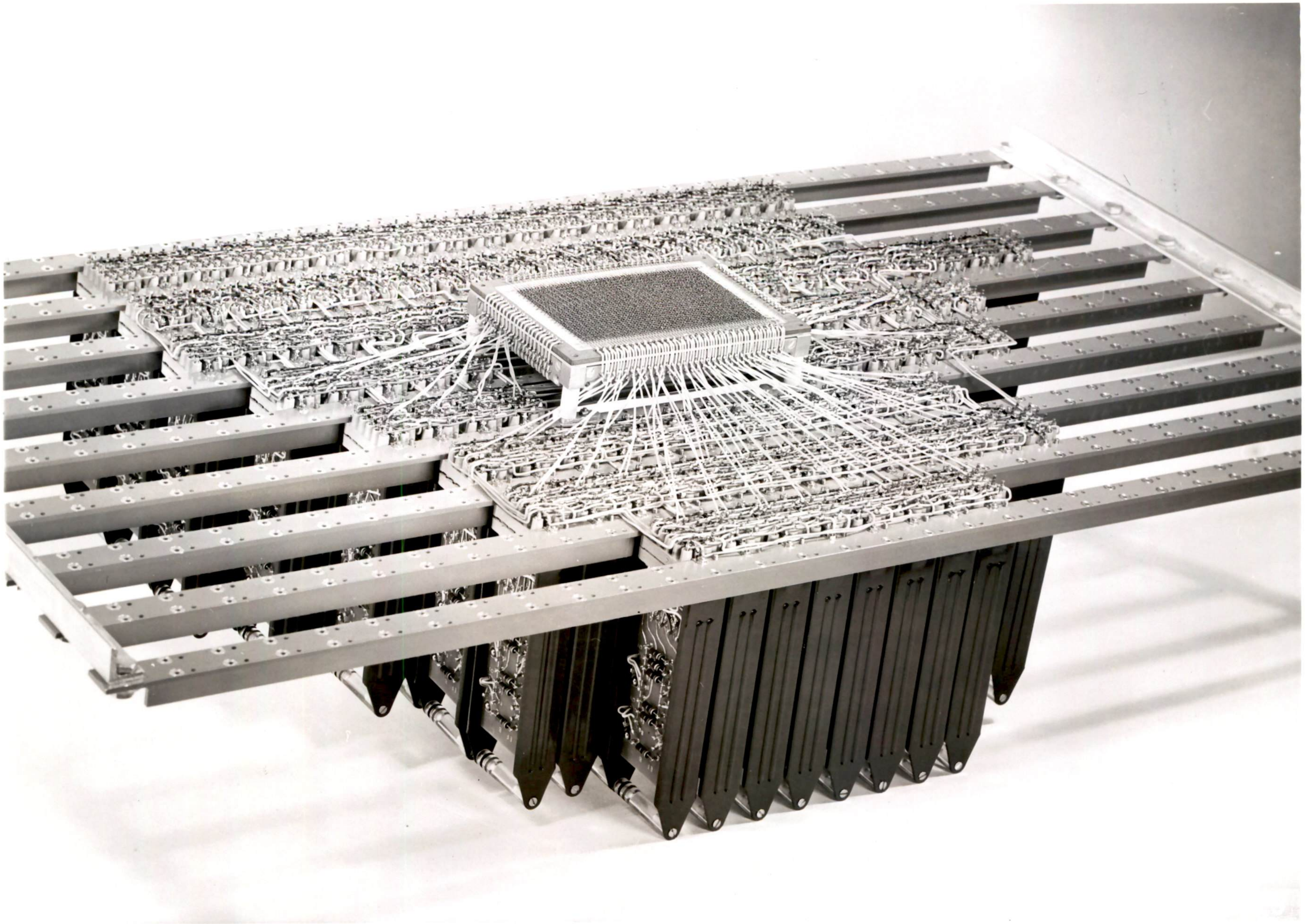


Fig. 12

TX-2 CIRCUITRY[+]

Kenneth H. Olsen[++]
Lincoln Laboratory
Lexington 73, Massachusetts

Circuit Configurations

Only two basic circuits are needed to perform most of the logical operations in the TX-2 computer; a saturated transistor inverter and a saturated emitter follower. To the logical designer who works with them, these circuits can be considered as simple switches which are either open or closed.

The schematic diagram of an emitter follower and the symbol used by the logical designers is shown in Figure 1. With a negative input, the output is "shorted" to the -3 volt supply as through a switch. When several of these emitter followers are combined in parallel, as in Figure 2, any one of them will clamp the output to -3V. We have then an OR circuit for negative signals and an AND circuit for positive signals. The transistor inverter is shown in Figure 3 with its logic symbol. Basic AND, OR circuits result from the connection of these simple switches in series or parallel,

[+] The research in this document was supported jointly by the Army, Navy, and Air Force under contract with the Massachusetts Institute of Technology.

[++] Staff Member, Lincoln Laboratory, Massachusetts Institute of Technology as in Figures 4 and 5. More complex networks like the TX-2 carry circuit use these elements arranged in series-parallel as shown in Figure 6.

In Figure 3 the resistor R_1 is chosen so that under the worst combinations of stated component and power supply variations, the drop across the transistor will be less than 200 millivolts during the "on condition". R_2 biases the transistor base positive during the off condition to provide greater tolerance to noise, I_{co}, and signal variations. Capacitance C was selected to remove all of the minority carriers from the base when the transistor is being turned off. The effect of C on a test circuit driven by a fast step is shown in Figure 7. Note that the delay due to hole storage is only a few millimicroseconds.

We run the circuits under saturated conditions to achieve stability and a wide tolerance to parameters without the needs for clamp diodes. Unlike vacuum tubes which always need an appreciable voltage across them for operation, a transistor requires practically no voltage across it. In spite of the delay in turning off saturated transistors, these circuits are faster than most vacuum tube circuits. Faster circuit speed is not due to the fact that the transistors are faster than vacuum tubes, but because they operate at much lower voltage levels. A vacuum tube takes a signal of several volts to turn it from fully "on" to fully "off"; a transistor takes less than one volt.

Flip-Flop

On the basis of previous experience, we decided that the advantages of having one standard flip-flop were worth some complication in TX-2 circuitry. The circuit diagram of the flip-flop package in Figure 8 is basically an Eccles-Jordan trigger circuit with a three-transistor amplifier on each output. The input amplifiers isolate the pulse input circuits and give high input impedance. The amplifiers give enough delay to allow the flip-flop to be set at the same time that it is being sensed. Figure 9 shows the waveforms of this flip-flop package when complemented at a 10 megapulse rate. The rise and fall times, about 25 millimicroseconds, are faster than one normally sees in a single inverter that pulls to ground and an emitter follower that pulls to -3 volts. Figure 10 is a plot of the pulse amplitude necessary to complement the flip-flop at various frequencies. Note the independence of trigger sensitivity to pulse repetition rate. This circuit will operate at a 10 megapulse rate, twice the maximum rate at which it will be used in TX-2.

The TX-2 circuits reproduced most often were designed with a minimum number of components to achieve economies in manufacture and maintenance. The design of less frequently reproduced circuits made liberal use of components - even redundancy to achieve long life and broad tolerance to component variations. The goal was system simplicity and high performance with a lower total number of components than might otherwise be possible. For example, the number of flip-flops in the TX-2 is small compared to the gates which transfer information from one group of flip-flops to another; so the flip-flops were allowed to be relatively complicated but the TX-2 transfer gates were made very simple. A transfer gate is in fact only a single inverter. The emitter is connected to the output of the flip-flop being read and the collector is connected to the input of the flip-flop being set. The output impedance of the flip-flop is so low that, when the output is at the ground level, a pulse on the base of the transfer gate shorts the input of the other flip-flop to ground and sets its condition.

Marginal Checking

We planned, of course, to incorporate marginal checking in the design of these circuits so that, under a process of regularly scheduled maintenance, deteriorating components could be located before they caused failure in the system. We also found it practical to use the technique during the design of the circuits to locate the design center of the various parameters and to indicate the tolerance of circuit performance of these parameters. A further application of marginal checking has been found in other systems during shakedown and initial operation to pin point noise and other system faults not serious enough to cause failure and therefore very difficult to isolate by other means.

The operating condition of the inverters is indicated by varying the +10 Volt bias. In the flip-flop schematic in Figure 8, the inverters were divided into two groups for marginal checking, and the two leads labelled MCA and MCB were varied one at a time for most critical checking of the circuit. The following curves show the locus of failure points for various parameters as a function of the marginal checking voltage. Figure 11 shows the tolerance to tau, a measure of hole storage and Figure 12 shows the tolerance to P, the current gain. Operating margins for supply voltages, temperature, and pulse amplitude are shown in Figures 13 through 16.

Packaging

The number of types of plug-in units was kept small for ease of production and to keep the number of spares to a minimum. The circuits are built on dip soldered etched boards and the components are hand soldered to solid turret lugs. The boards are mounted in steel shells shown in Figure 17 to keep the boards from flexing. The male and female contacts are machined and gold plated. The sockets are hand wired and soldered in panels as in Figure 18。

Conclusion

The result of these design considerations is a 5 megapulse control and arithmetic element which will take less than 40 square feet of space and dissipate less than 800 watts of power。 The simplicity of the circuits has encouraged a degree of logical sophistication which would not have been chanced before。

TX-2 CIRCUITRY

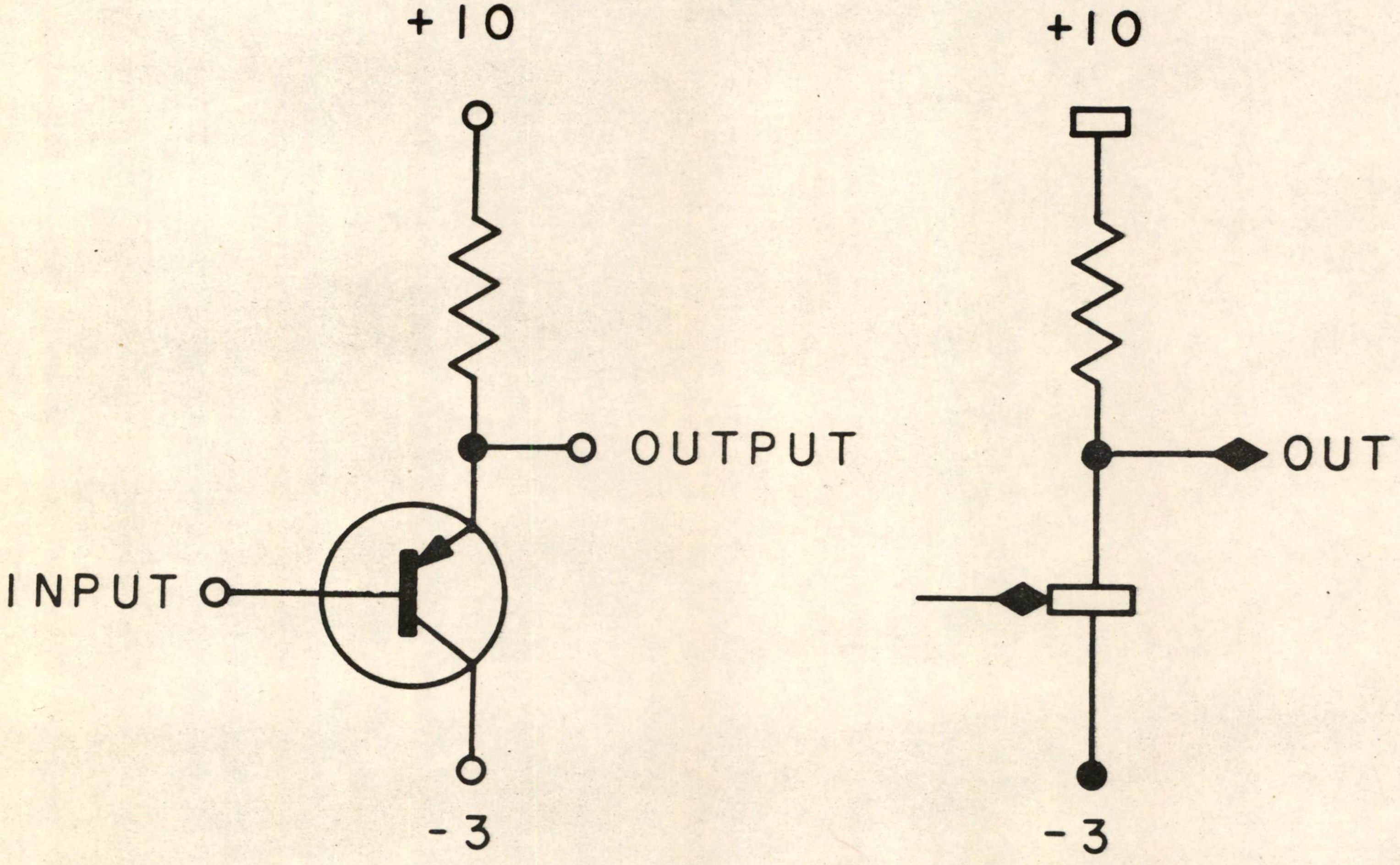
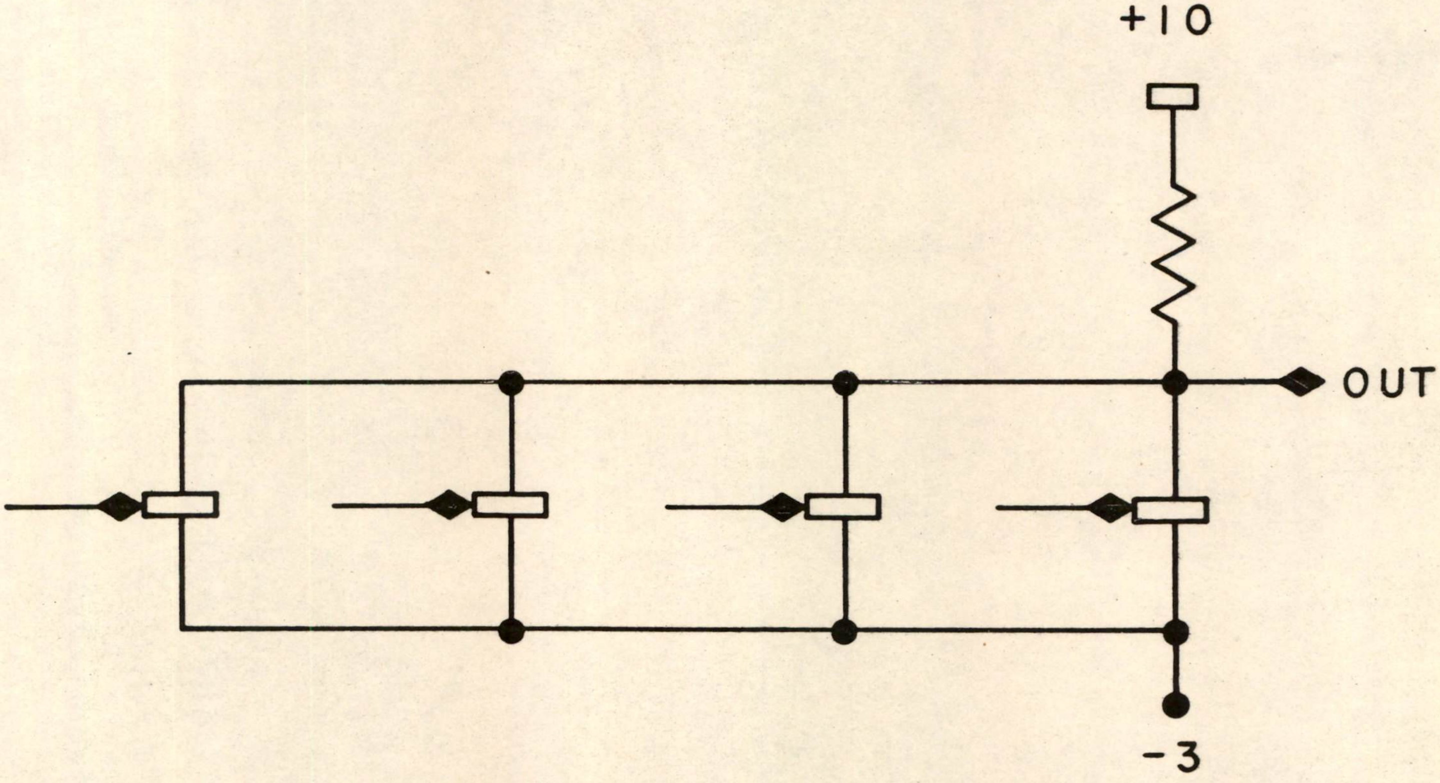
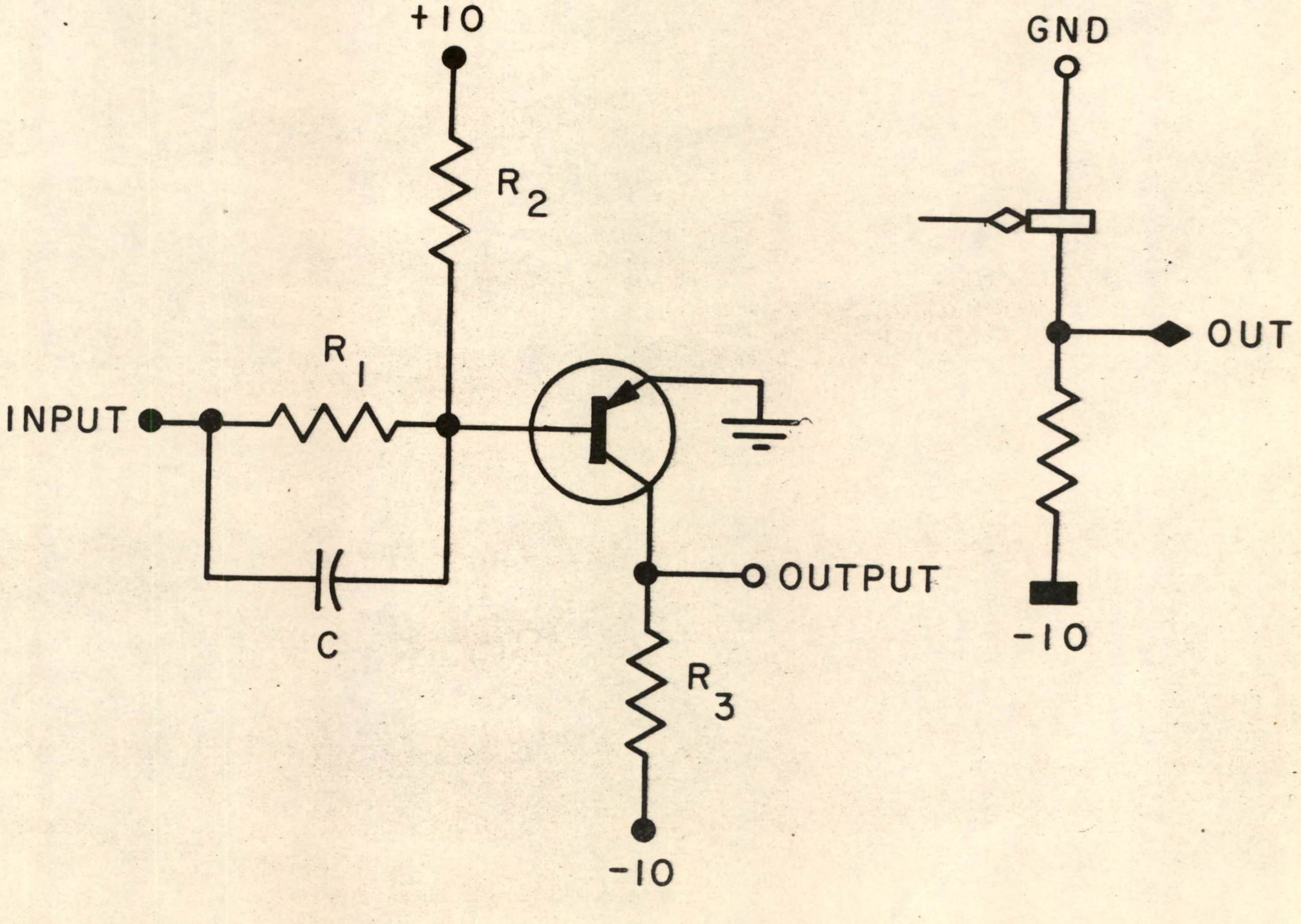
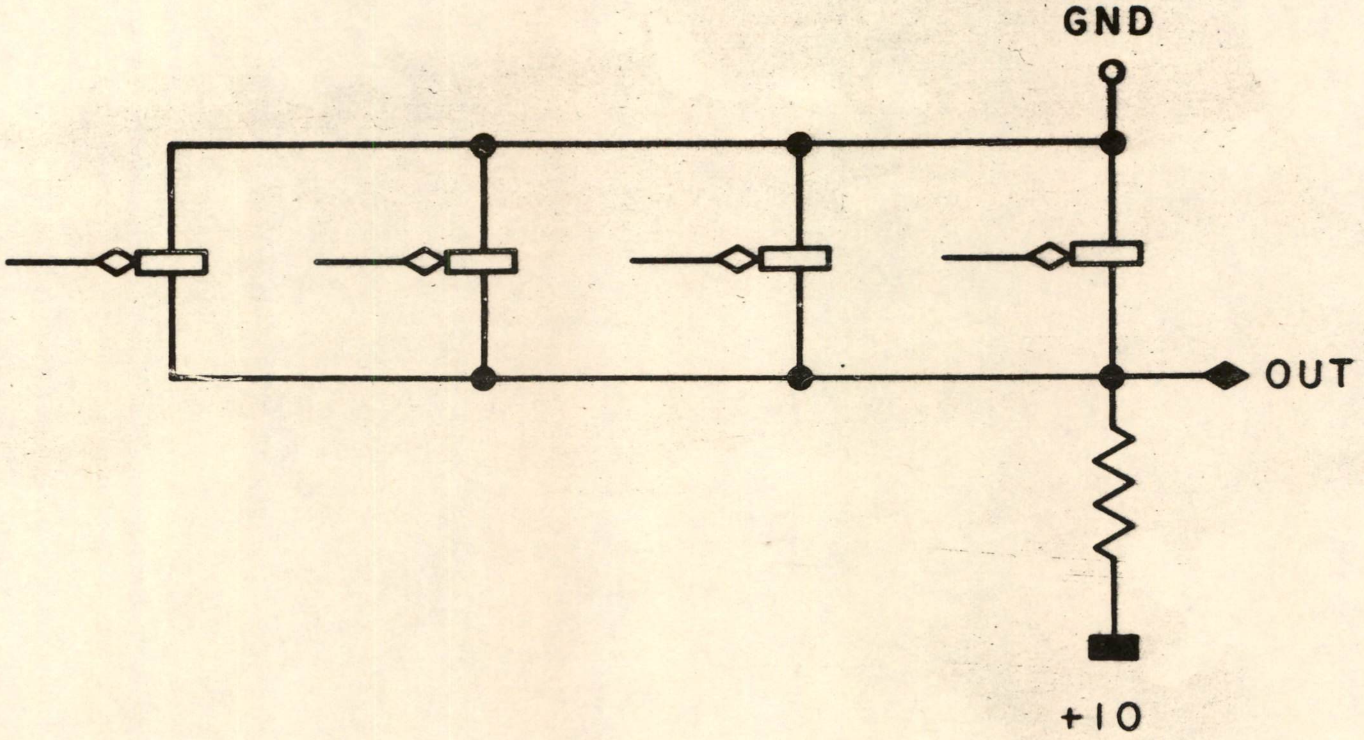


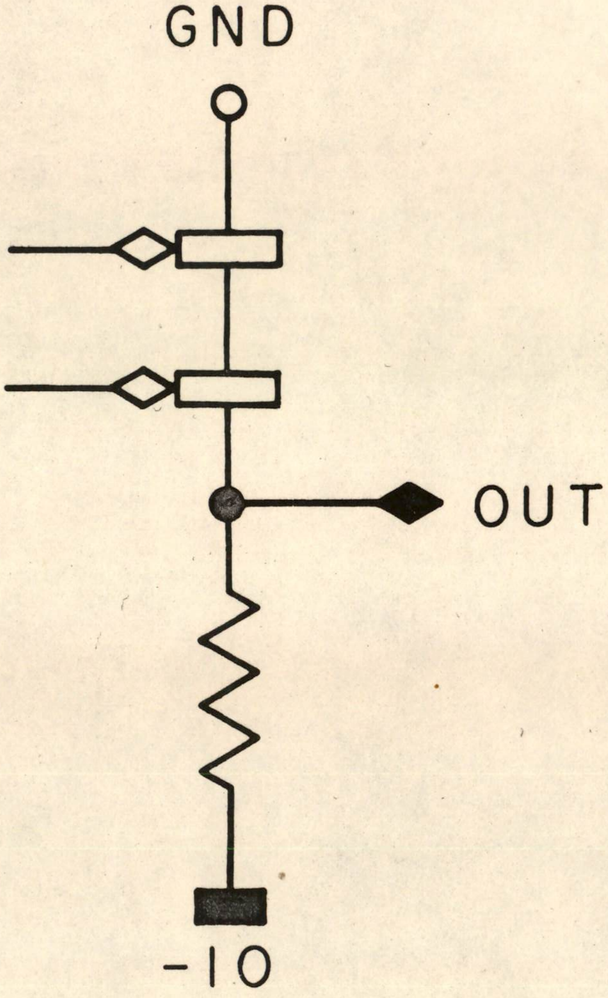
fig1

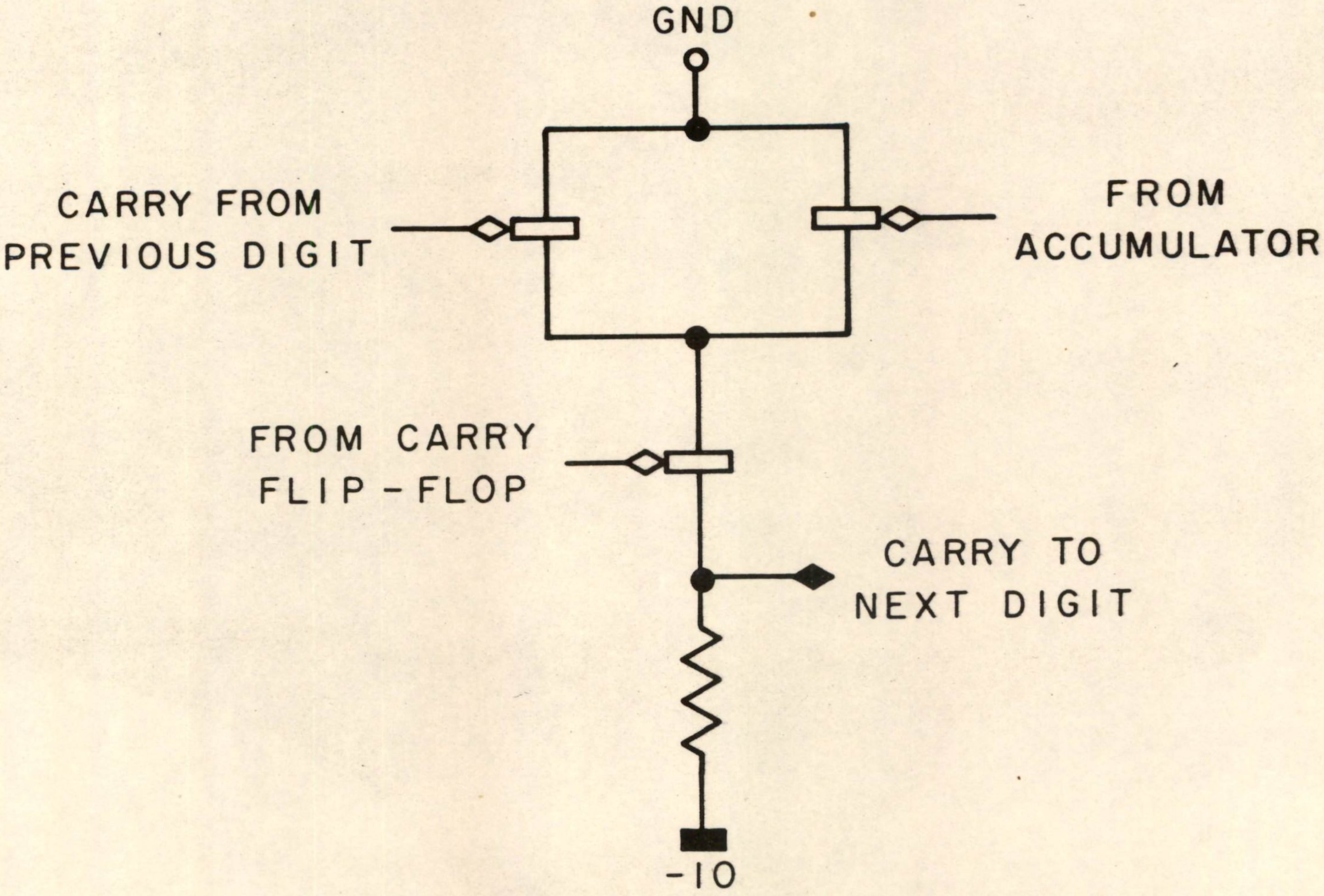

+10
OUT
-3

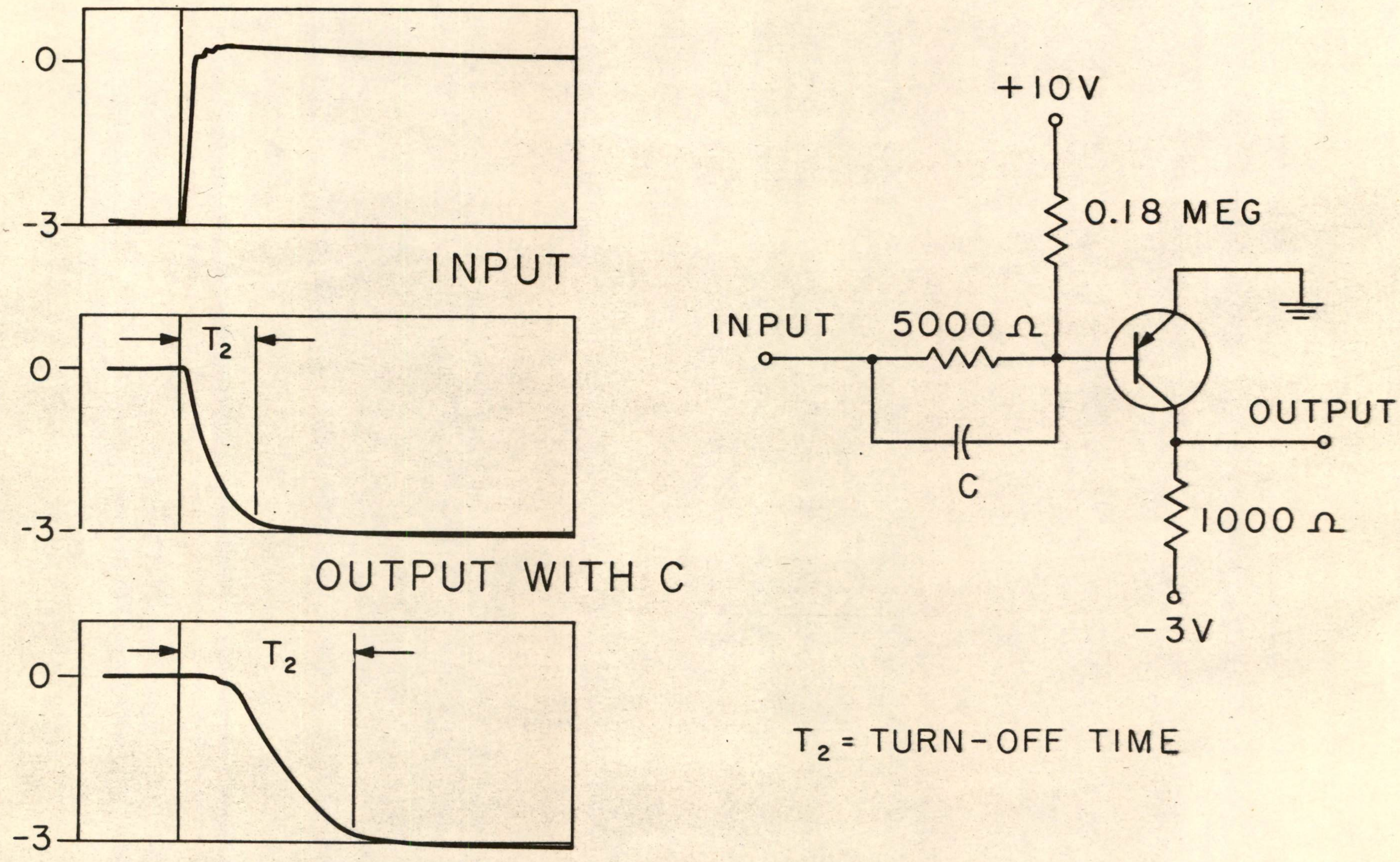
C60-361




GND
OUT
+10


GND
OUT
-10


GND
CARRY FROM
PREVIOUS DIGIT
FROM
ACCUMULATOR
FROM CARRY
FLIP-FLOP
CARRY TO
NEXT DIGIT
-10



INPUT

OUTPUT WITH C

OUTPUT WITHOUT C

T$_2$ = TURN-OFF TIME

TURN—OFF TIME

Fig 7

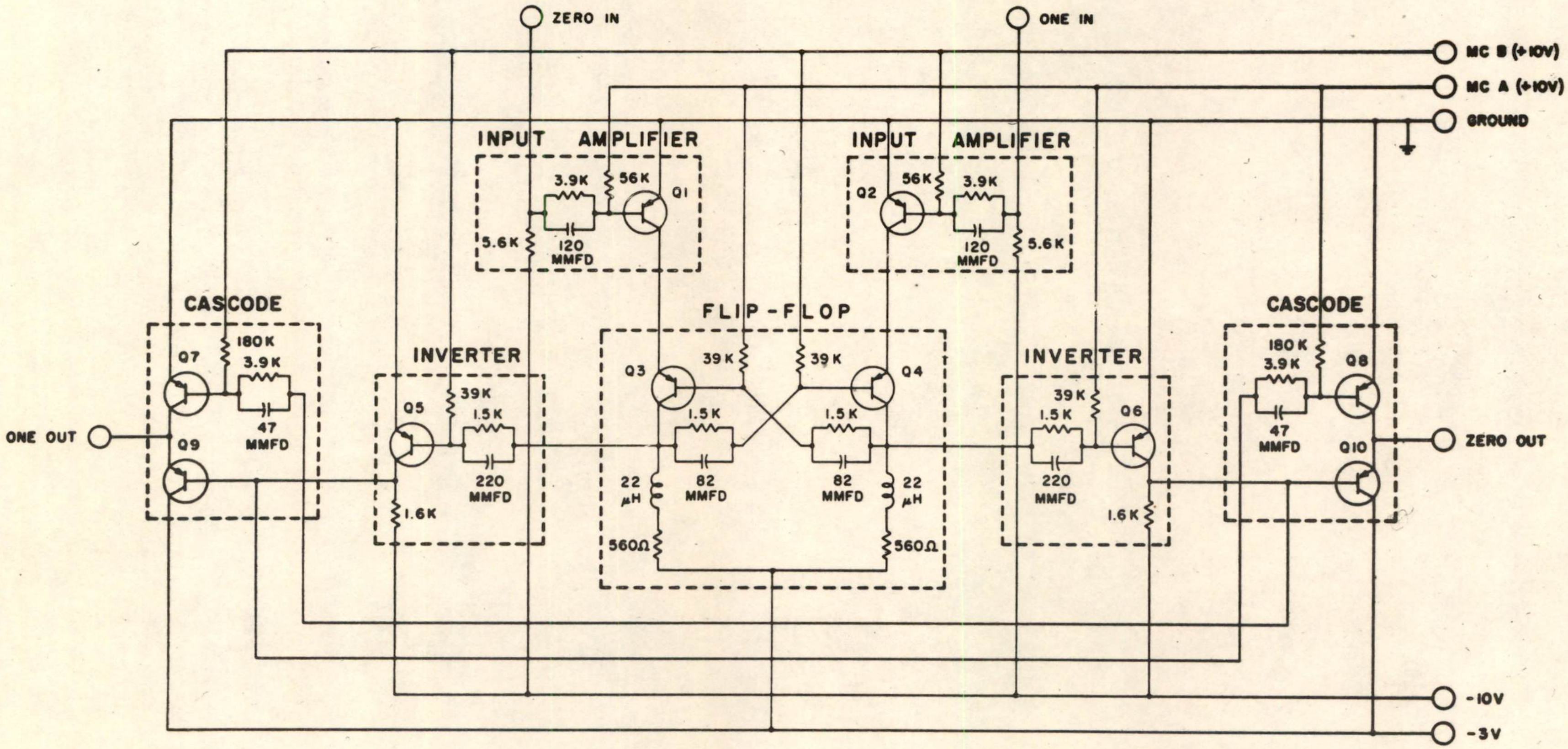
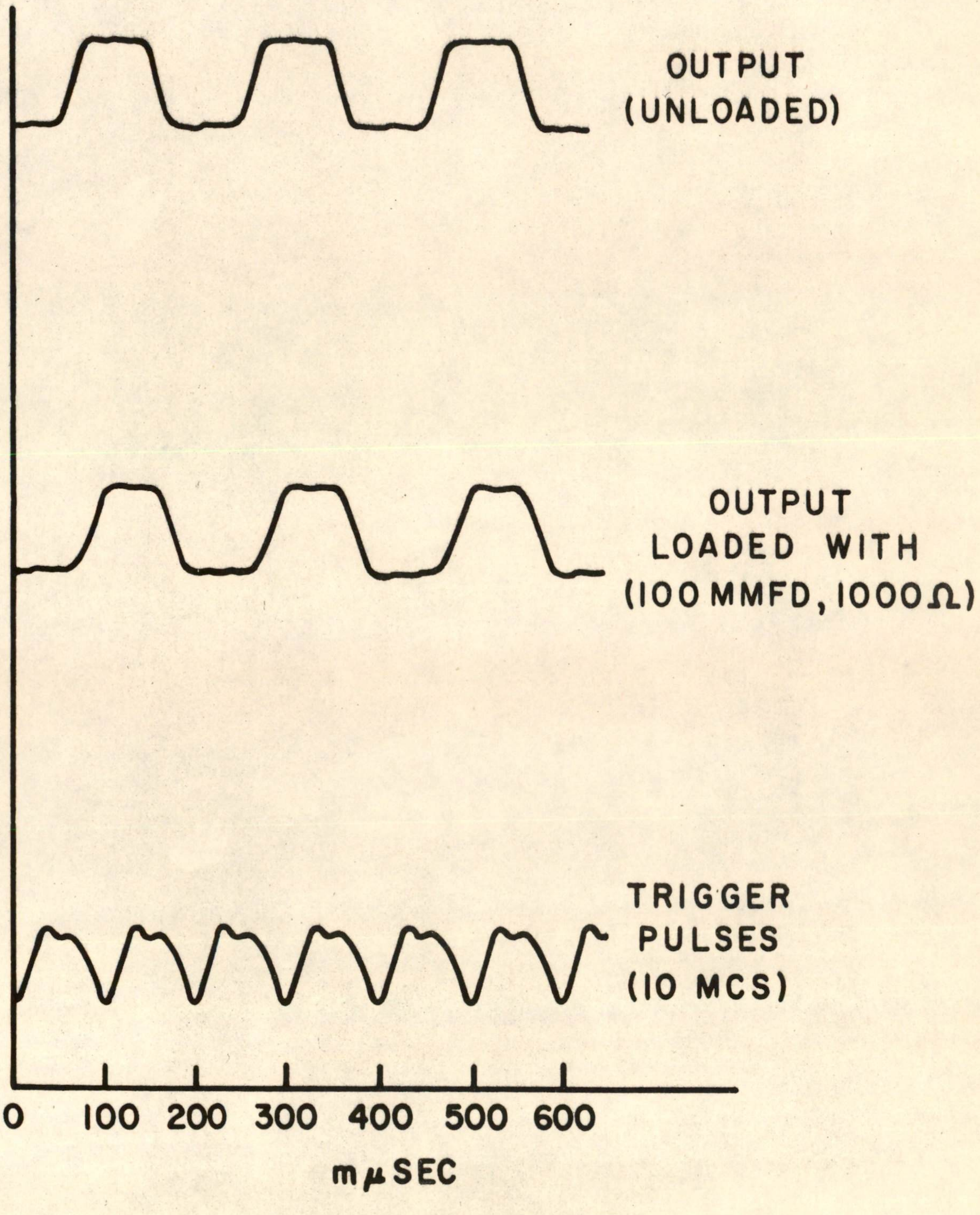
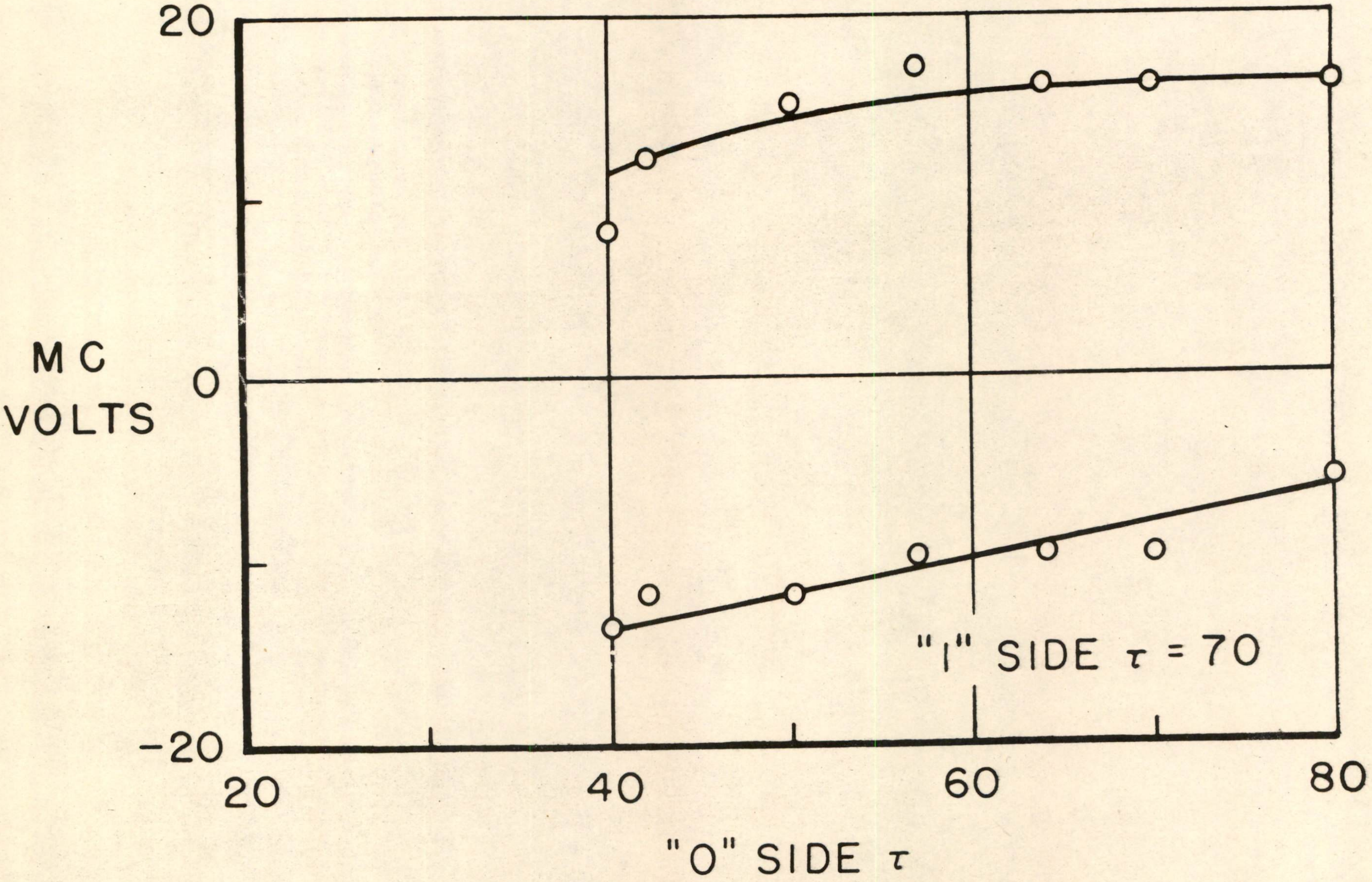
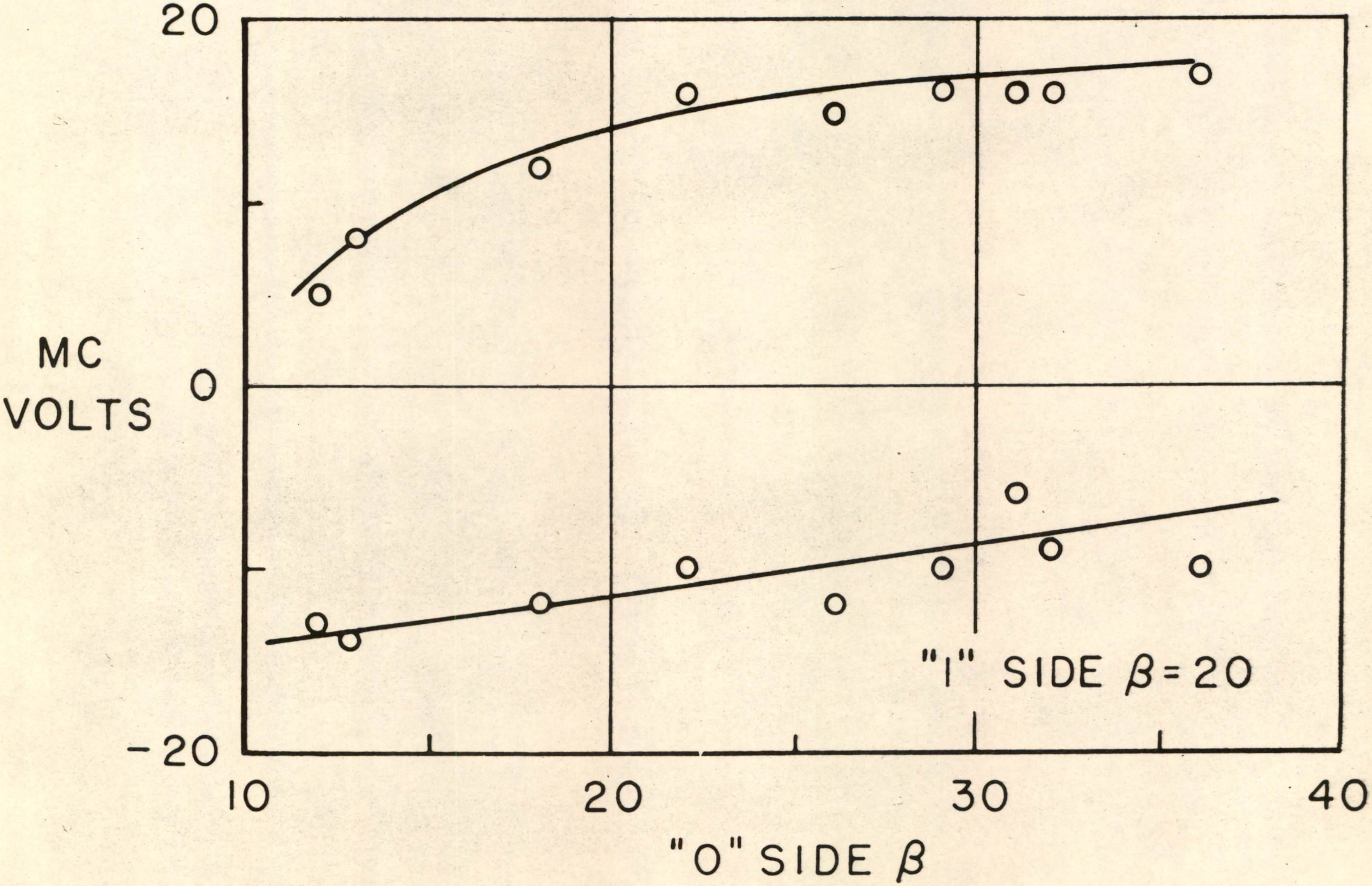
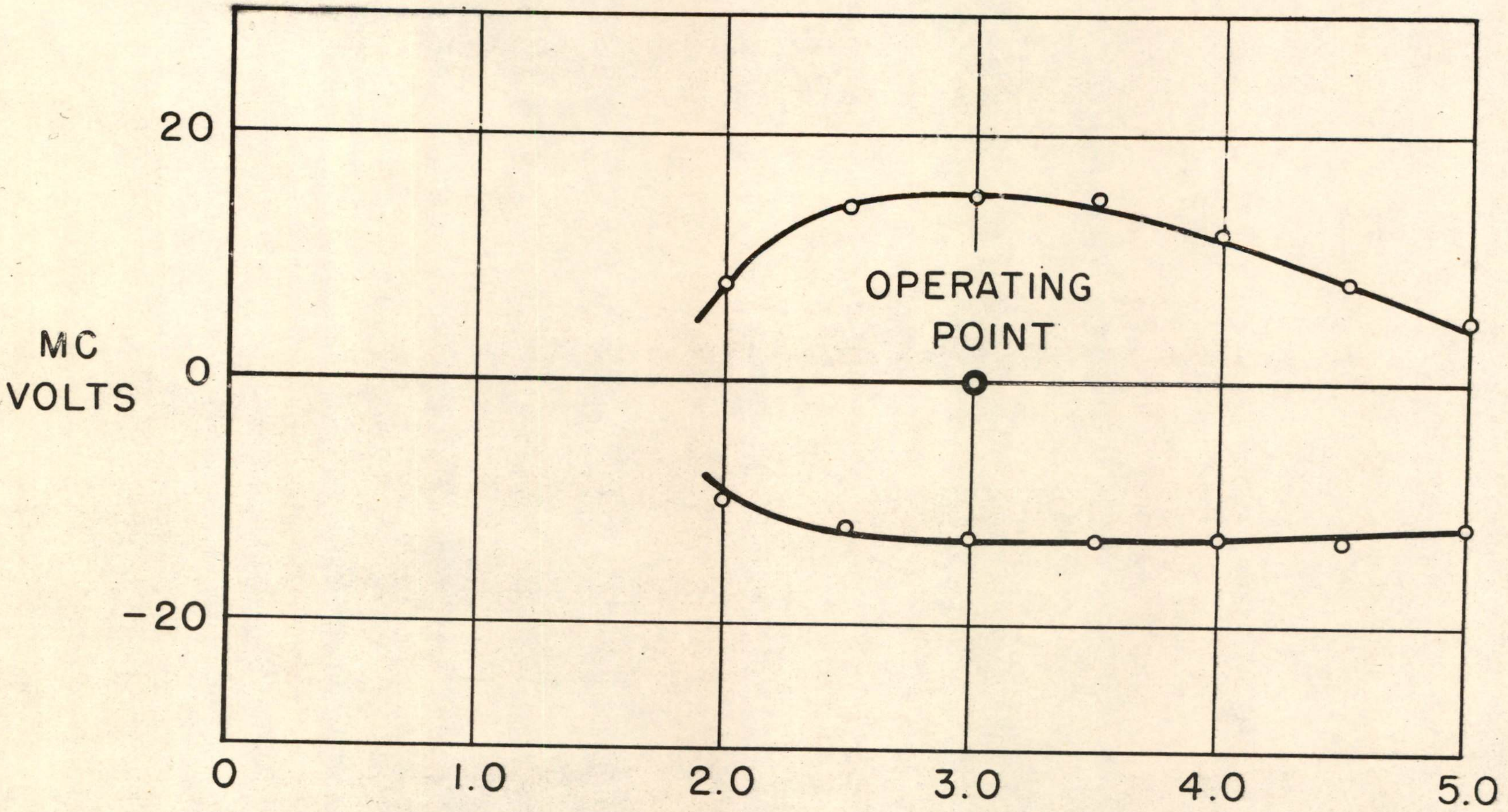

ZERO IN
ONE IN
MC B (+10V)
MC A (+10V)
GROUND
INPUT AMPLIFIER
3.9K
56K
Q1
5.6K
120 MMFD
INPUT AMPLIFIER
Q2
56K
3.9K
120 MMFD
5.6K
CASCODE
Q7
180K
3.9K
47 MMFD
Q9
ONE OUT
INVERTER
Q5
39K
1.5K
220 MMFD
1.6K
FLIP-FLOP
Q3
39K
39K
Q4
1.5K
1.5K
82 MMFD
82 MMFD
22 µH
22 µH
560Ω
560Ω
INVERTER
39K
1.5K
Q6
220 MMFD
1.6K
CASCODE
180K
3.9K
Q8
47 MMFD
Q10
ZERO OUT
-10V
-3V

Fig 8

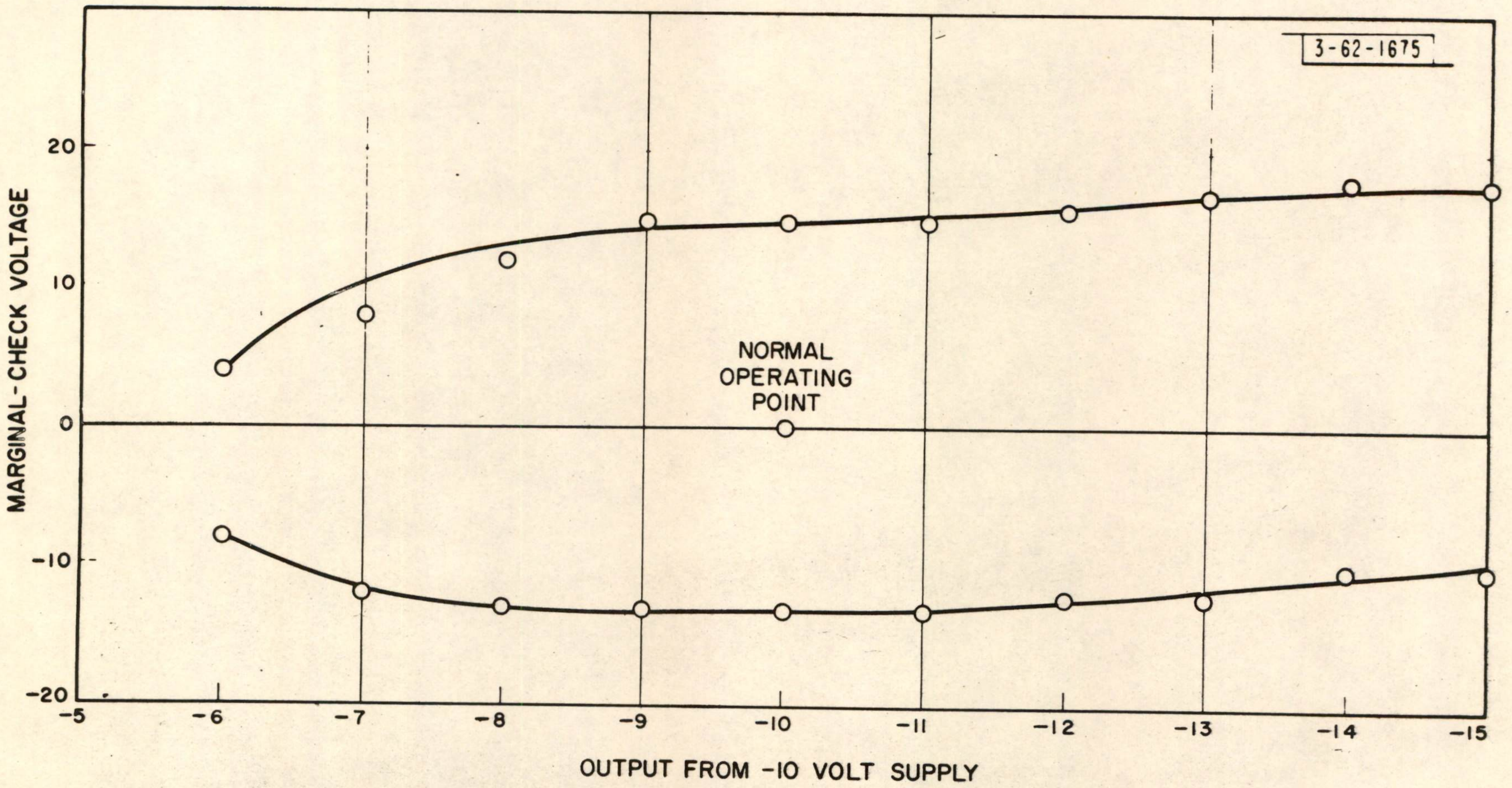

OUTPUT
(UNLOADED)
OUTPUT
LOADED WITH
(100 MMFD, 1000Ω)
TRIGGER
PULSES
(10 MCS)
0
100
200
300
400
500
600
mμSEC

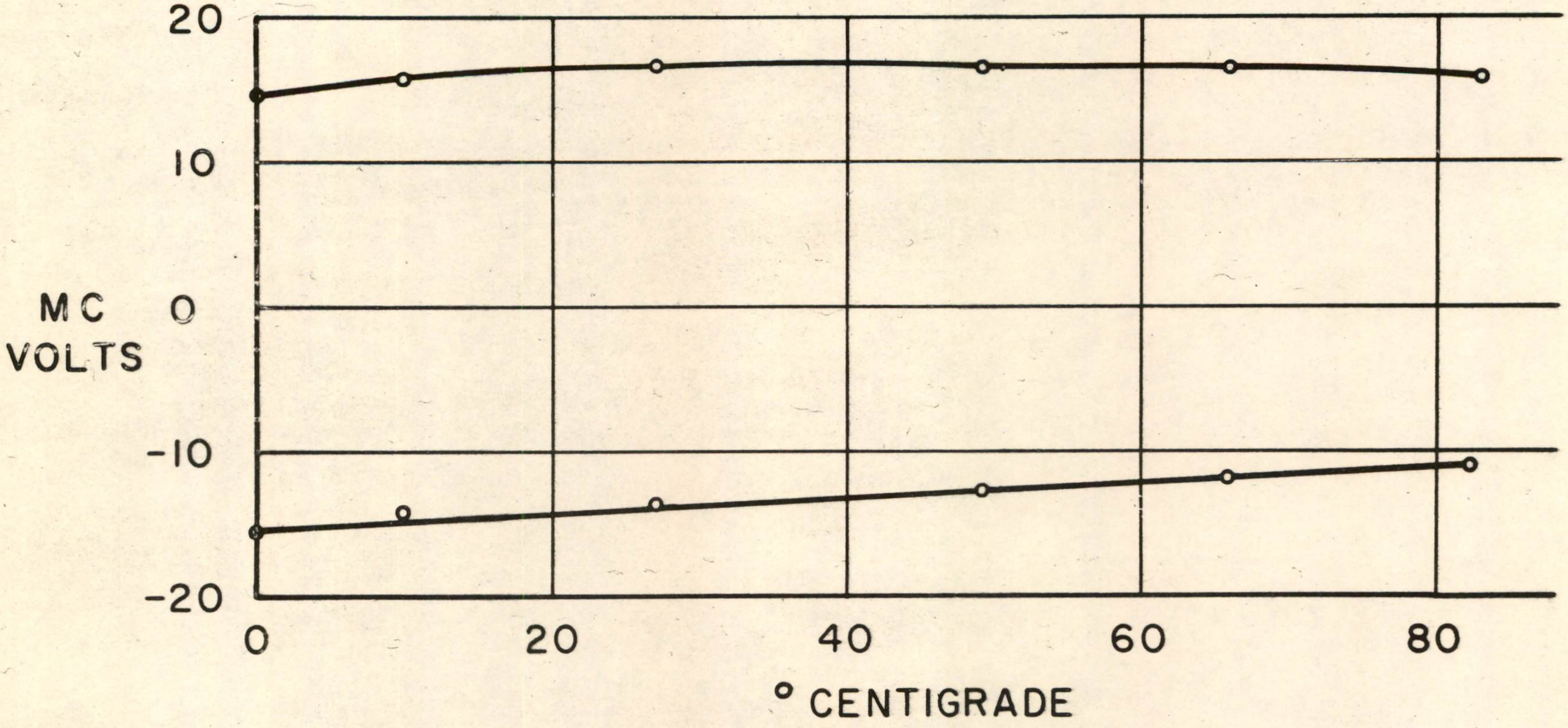

20
0
-20
MC
VOLTS
20
40
60
80
"O" SIDE τ
"I" SIDE τ = 70


20
0
-20
MC
VOLTS
10
20
30
40
"O" SIDE β
"I" SIDE β = 20

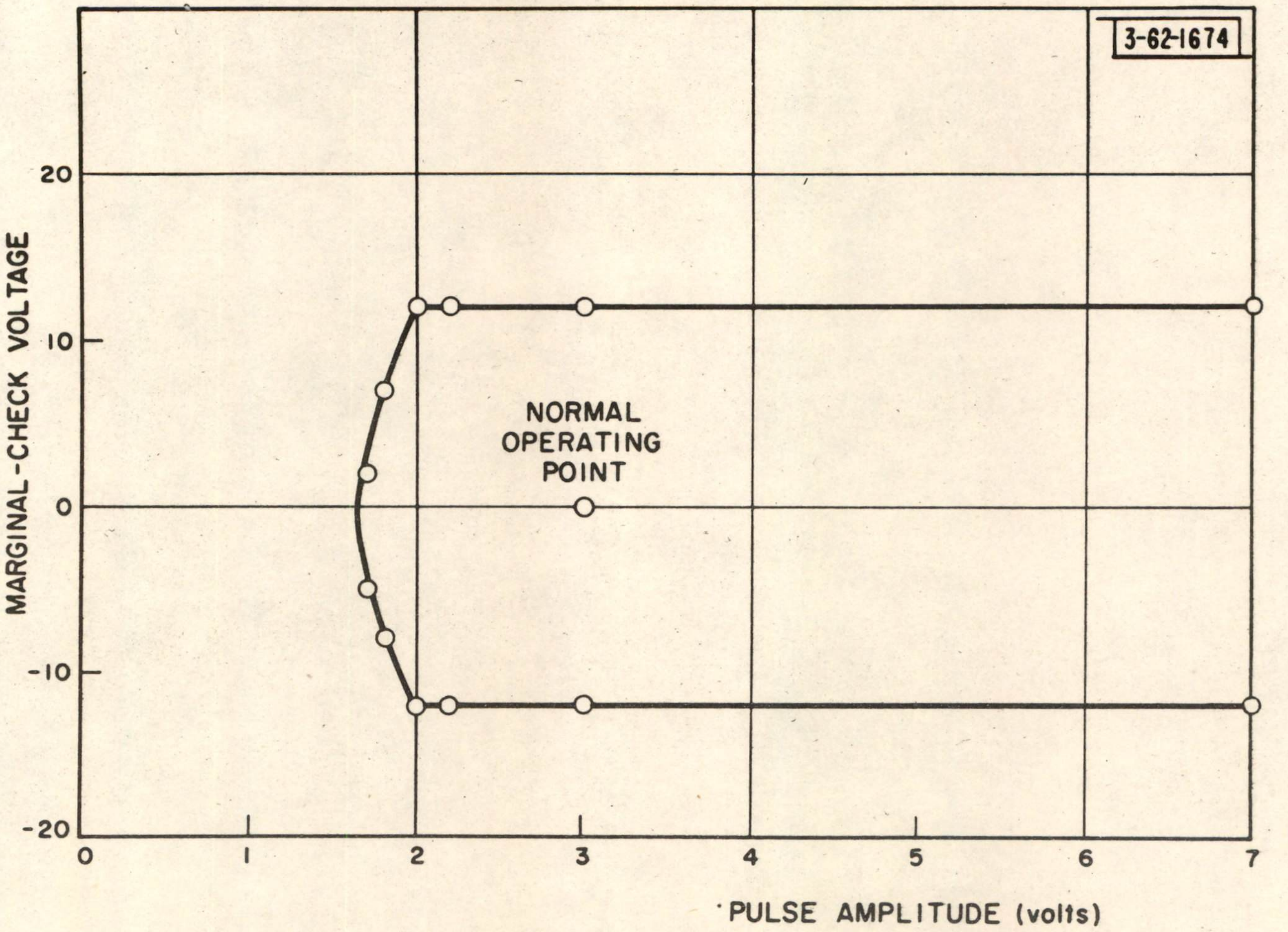


-3 VOLT SUPPLY MARGINS


3-62-1675
MARGINAL-CHECK VOLTAGE
20
10
0
-10
-20
NORMAL OPERATING POINT
-5
-6
-7
-8
-9
-10
-11
-12
-13
-14
-15
OUTPUT FROM -10 VOLT SUPPLY



TEMPERATURE MARGINS


3-62-1674
MARGINAL-CHECK VOLTAGE
20
10
0
-10
-20
0
1
2
3
4
5
6
7
PULSE AMPLITUDE (volts)
NORMAL
OPERATING
POINT

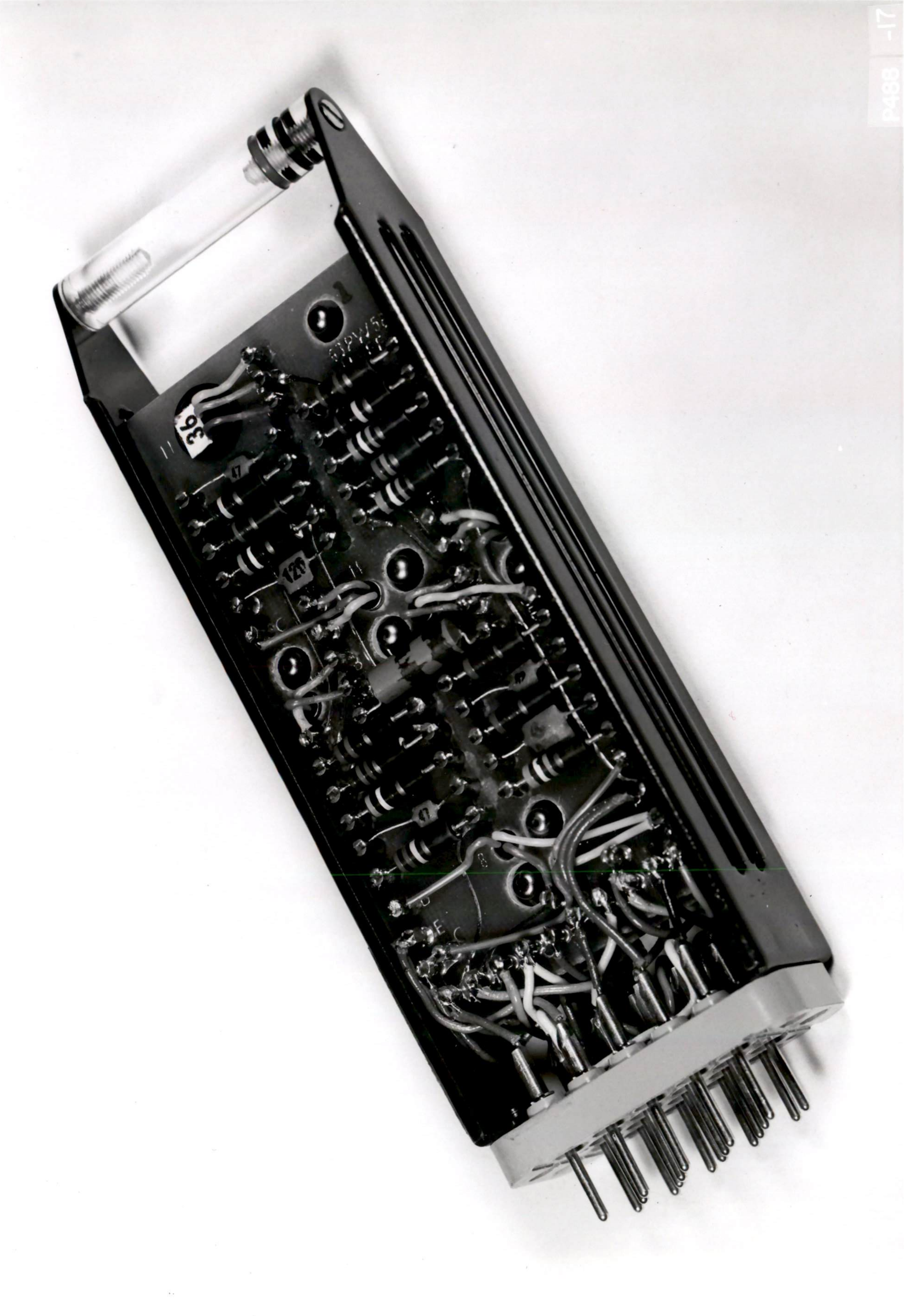


Fig. 17

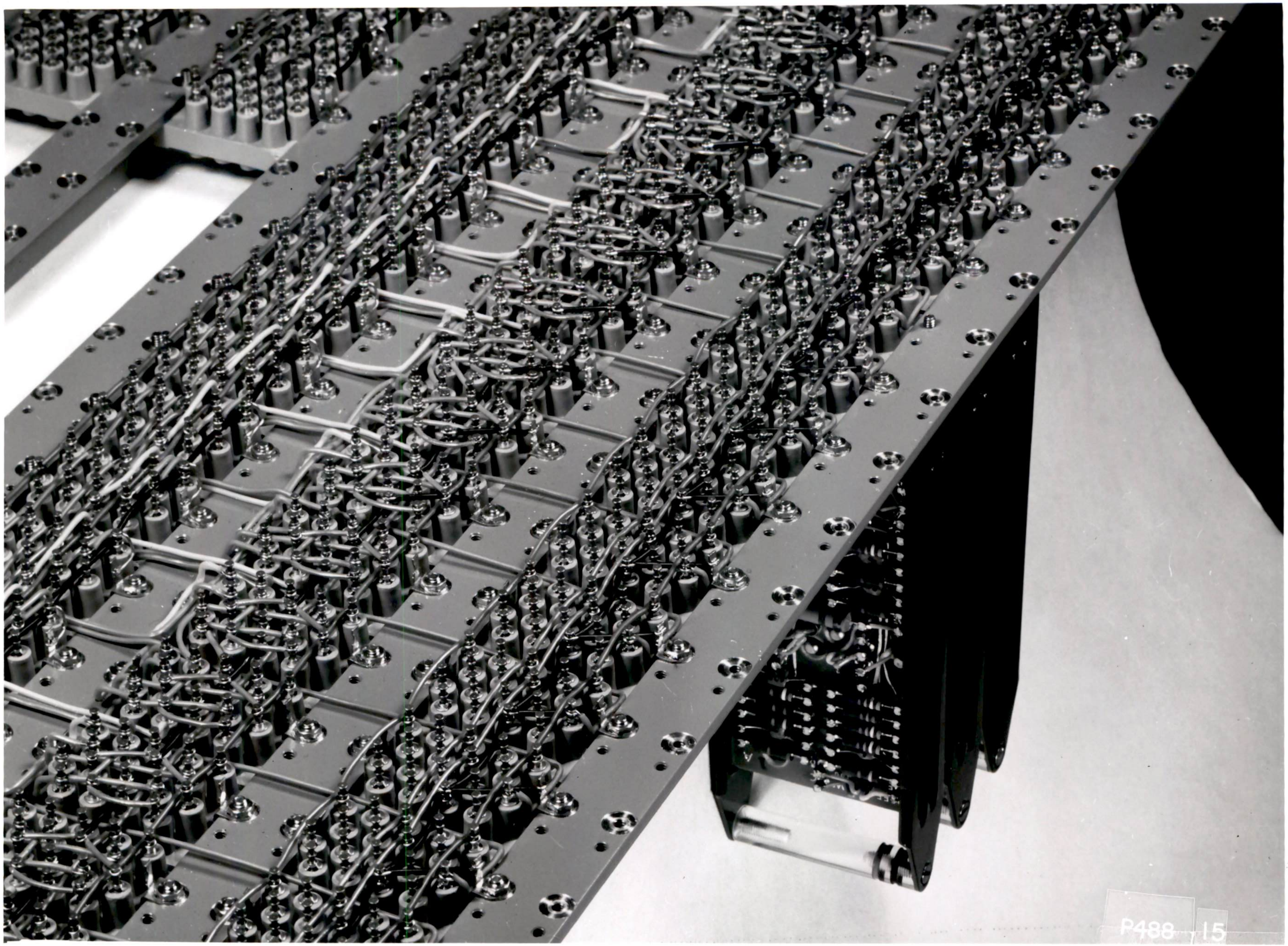


Fig. 18

ACKNOWLEDGMENT

We are indebted to Mr. Frank P. Hazel for his assistance in editing; to Mrs. Barbara Clark, Miss Carole Olson, Miss Elaine Tonra, Miss Ruth McDonald and Miss June Karlson for their typing of the manuscripts; and to Miss Alice Griffin and her associates for the preparation of the many figures.