A Functional Description of the Lincoln TX-2 Computer*

J. M. FRANKOVICH† AND H. P. PETERSON†

INTRODUCTION

THE TX-2 is a large scale digital computer designed and built at the Massachusetts Institute of Technology 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. Furthermore, the system organization facilitates the execution of several operations simultaneously, thereby increasing the effective speed of the computer.

The principal 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 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.

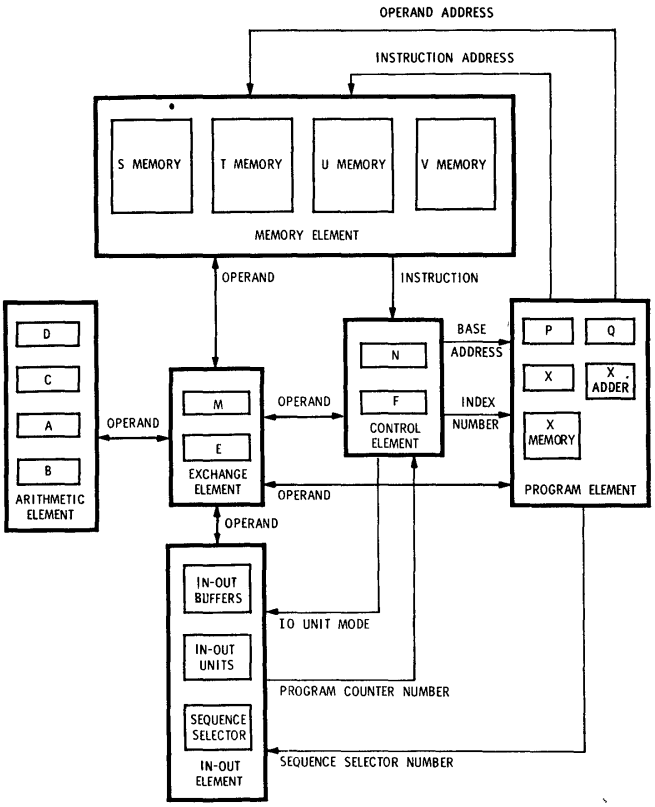

Fig. 1—TX-2 system schematic, showing the principal registers and transfer paths.

The computer communicates with the outside world through units in the In-Out Element, 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]

* This research was supported jointly by the Army, Navy, and Air Force under contract with the Mass. Inst. Tech., Cambridge, Mass.

† M.I.T. Lincoln Lab., Lexington, Mass.

[1] J. W. Forgie, "The Lincoln TX-2 in-out system," this issue· p. 156.

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.

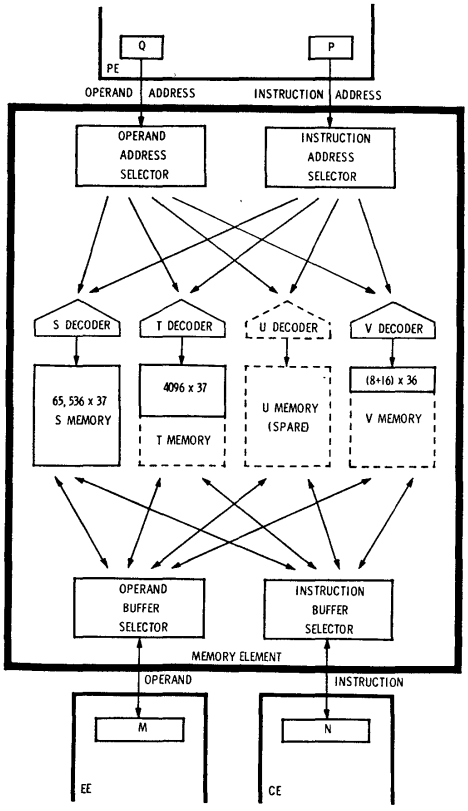

Fig. 2—TX-2 Memory Element. Two address and two buffer registers are used to permit simultaneous operation of any two of the four memories.

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 central 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 also 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-counts-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 *d* digits is not discussed here.[2] The *cf* digits will be discussed later.

The operation code for the instruction is specified by

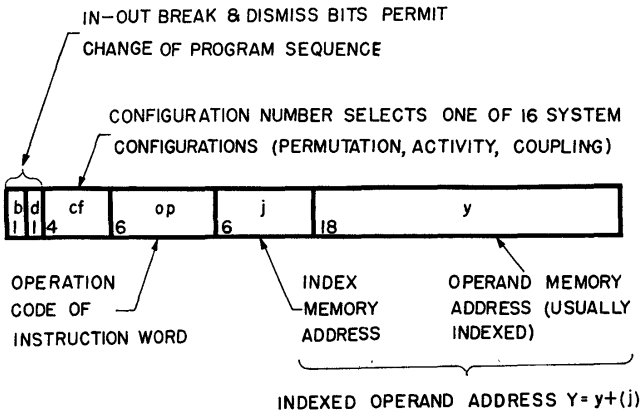

Fig. 3—TX-2 instruction word layout.

[2] *Ibid.*

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 over-all 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.

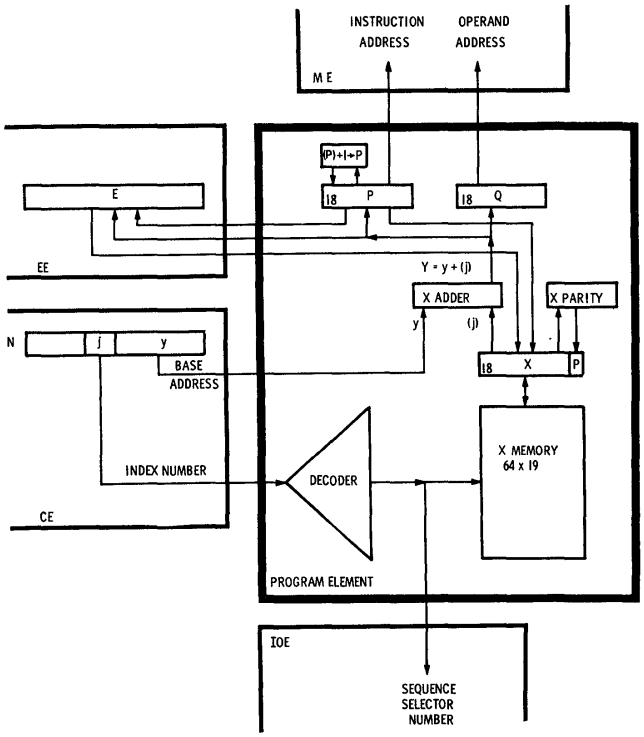

Fig. 4—TX-2 Program Element, determining the instruction and operand memory addresses, performing X memory operations.

Arithmetic Element

The registers and sufficient basic operations in the Arithmetic Element (AE) 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 AE.

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 on the next page.

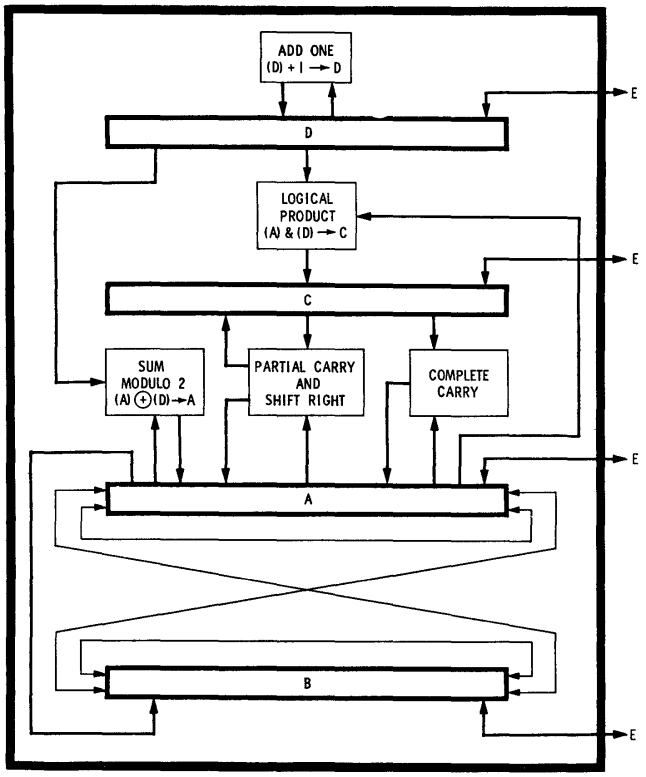

Fig. 5—TX-2 Arithmetic Element, showing the circuits and transfer paths for AE operations.

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 dividend 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 significant 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 72, 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 microsecond. 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 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-mega-cycle 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 are assumed. If a sequence of store type instructions is executed which requires 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 until the time that the address of the operand is known is greater than the instruction memory cycle time, then the computer

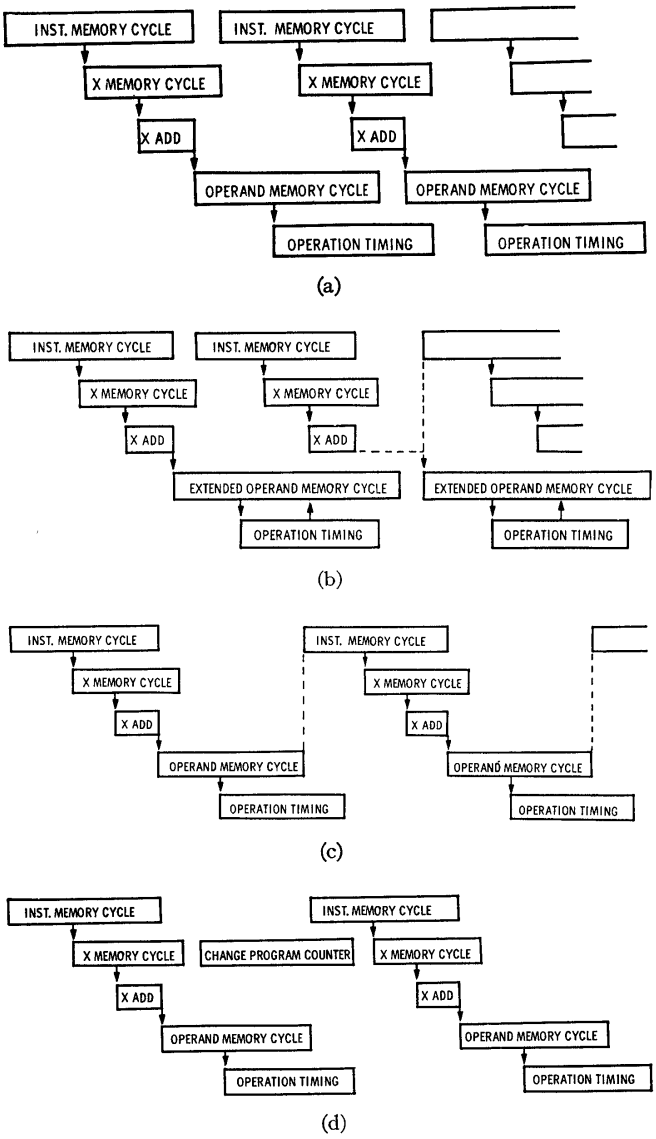

Fig. 6—TX-2 timing schematic, showing overlapped execution of memory and operation cycles. (a) Consecutive load type instructions—instructions and operands in different memories. (b) Consecutive store type instructions. (c) Instruction and operand in same memory. (d) Change sequence.

cannot run in the ideal manner shown in Fig. 6(a). This means that the access time of all 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 sequence by changing the contents of the *P* register. The additional *X* Memory cycles which must be performed in carrying this out produce 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 Fig. 3) are decoded as shown schematically in Fig. 7. The contents of the selected one of 16 9-digit configuration words are 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 P0, P1, P2, or P3 as defined in Fig. 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 word 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

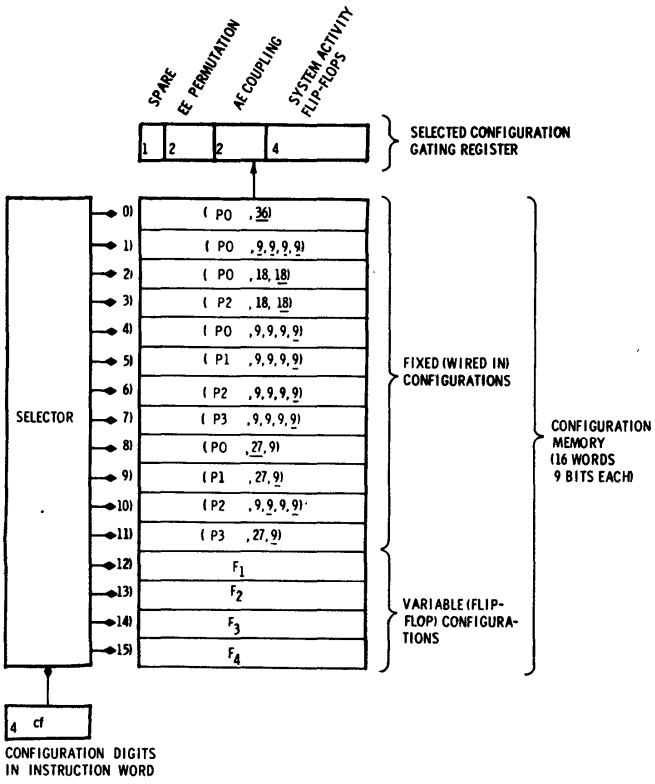

Fig. 7—TX-2 configuration selection. The *cf* digits select a configuration for the computer for use during the execution of the instruction.

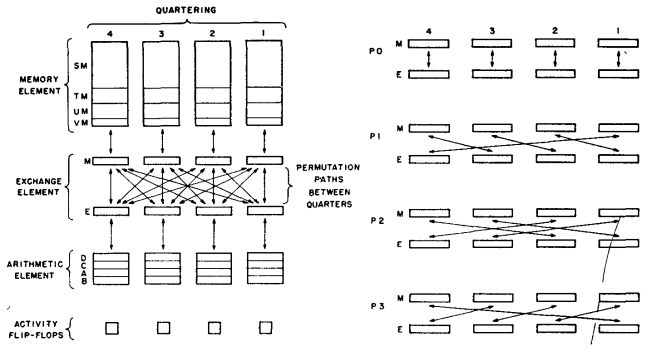

Fig. 8—TX-2 configuration. (a) Quartering, permutation paths, and activity flip-flops. (b) The four sets of permutation paths available, one of which is used during the execution of an instruction.

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 of that quarter. If the activity flip-flop of a given quarter holds a "one," as specified by the configuration, then the operation timing events of the instruction occur in that quarter. If the activity flip-flop holds 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.

In Fig. 9(a) (next page) 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 among 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 Fig. 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 (P0, $\underline{36}$) 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 (P0, 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 (P2, 9, $\underline{9}$, 9, $\underline{9}$) configuration has the same arithmetic elements but with the associated memories interchanged. The (P2, 18, $\underline{18}$) configuration illustrates an 18-digit arithmetic element which uses the "other" half of memory.

One of the 9 configuration digits is at the moment unused, but will probably be used to control the ex-

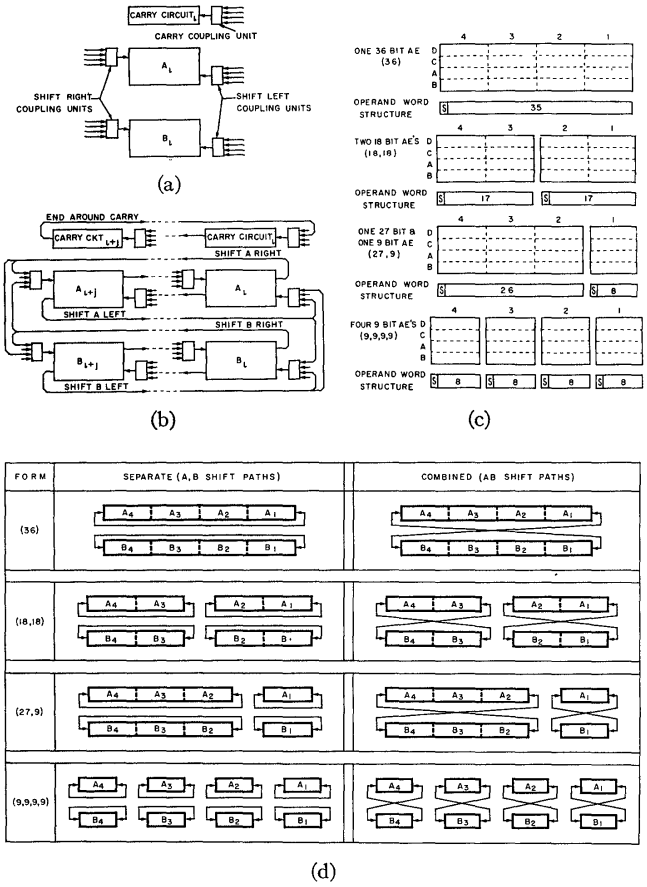

Fig. 9—TX-2 arithmetic element coupling units. (a) i-th quarter coupling units. The coupling units receive information moving laterally into the i-th quarter of the AE, $i=1, 2, 3, 4$. (b) Coupling unit connections between a contiguous group of quarters which realize a 9-bit ($j=0$), 18-bit ($j=1$),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

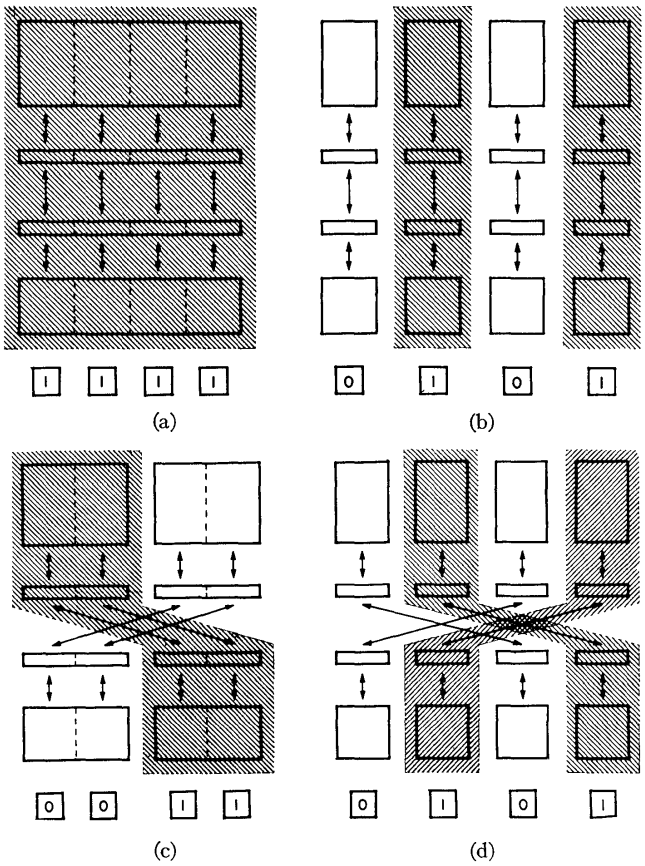

Fig. 10—Illustrative example of different configurations. Areas of activity during execution of instruction are shown shaded. Effects of AE coupling are shown by juxtaposition. (a) (P0, 36) configuration. (b) (P0, 9, 9, 9, 9) configuration. (c) (P2, 18, $\underline{18}$) configuration. (d) (P2. 9, $\underline{9}$, 9, $\underline{9}$) configuration.

tension 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 higher precision can be maintained during computations. 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. 7. 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 (opposite) 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 (p. 154).

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 one's 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

TABLE I

Type	Mnemonic Code	Operation	Name
Load	lda	$(\bar{Y})\rightarrow A$	Load into A
	ldb	$(\bar{Y})\rightarrow B$	Load into B
	ldc	$(\bar{Y})\rightarrow C$	Load into C
	ldd	$(\bar{Y})\rightarrow D$	Load into D
	lde	$(\bar{Y})\rightarrow E$	Load into E
	ldf	$(\bar{Y})\rightarrow F$	Load into F
	lca	$\overline{(Y)}\rightarrow A$	Load complement into A
	lcb	$\overline{(Y)}\rightarrow B$	Load complement into B
	ldx	$(y)\rightarrow j$	Load into index
Store	sta	$(A)\rightarrow Y$	Store A
	stb	$(B)\rightarrow Y$	Store B
	stc	$(C)\rightarrow Y$	Store C
	std	$(D)\rightarrow Y$	Store D
	ste	$(E)\rightarrow Y$	Store E
	stf	$(F)\rightarrow Y$	Store F
	exa	$(Y)\rightarrow A$; $(A)\rightarrow Y$	Exchange A
	ins	$(B)\ \&\ (A)\vee\overline{(B)}\ \&\ (Y)\rightarrow Y$	Insert digits of A
	stx	$(j)\rightarrow y$	Store index
Add	add	$(A)\ +\ (Y)\rightarrow A$	Add
	sub	$(A)\ +\ \overline{(Y)}\rightarrow A$	Subtract
	dma	$\lvert(A)\rvert+\lvert\overline{(Y)}\rvert\rightarrow A$	Difference of magnitude
	and	$(A)\ \&\ (Y)\rightarrow A$	Logical and
	ori	$(A)\ \vee\ (Y)\rightarrow A$	Logical or—inclusive
	ore	$(A)\oplus(Y)\rightarrow A$; $(A)\ \&\ (Y)\vee(C)\rightarrow C$	Logical or—exclusive (and accumulate product)
	axm	$(j)+(y)\rightarrow y$	Add index to memory
	amx	$(j)+(y)\rightarrow j$	Add memory to index
Set bit	sbo	$1\rightarrow Y_j$	Set j-th bit one
	sbz	$0\rightarrow Y_j$	Set j-th bit zero
	sbc	$\overline{(Y_j)}\rightarrow Y_j$	Set j-th bit complement
Step-Count	mul	$(A)\times(Y)\rightarrow AB$	Multiply
	div	$(AB)\div(Y)\rightarrow A$ (remainder), B (quotient)	Divide
	sha	$(A)\times 2^{(Y)}\rightarrow A$	Shift A
	sab	$(AB)\times 2^{(Y)}\rightarrow AB$	Shift AB together
	shb	$(B)\times 2^{(Y)}\rightarrow B$	Shift B
	cya	$(A)\ \text{cyc}(Y)\rightarrow A$	Cycle A
	cab	$(AB)\ \text{cyc}(Y)\rightarrow AB$	Cycle AB together
	cyb	$(B)\ \text{cyc}(Y)\rightarrow B$	Cycle B
	nab	$(AB)\times 2^{nf}\rightarrow AB$; $(Y)-nf\rightarrow D$	Normalize AB
	coa	$(Y)+no\rightarrow D$	Count ones in A
In-out	rds	$(Y)\rightarrow IO$; $(IO)\rightarrow Y$	Read and shift
	rdn	$(Y)\rightarrow IO$; $(IO)\rightarrow Y$	Read without shift
Jump	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 $(A)\leq 0$, then $Y\rightarrow P$	Jump if the contents of any A is negative
	jpz	If any $(A)=0$, then $Y\rightarrow P$	Jump if the contents of any A is zero
	jpo	If any (A) overflowed, then $Y\rightarrow P$	Jump if the contents of any A has overflowed
	jxp	If $(j)\geq 0$, 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		In-out select
	opr		Operate

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 until the magnitude of the number in A is between one-half and one. The number of shifts 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

TABLE II

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

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.

The two in-out read instructions 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 single bits in memory as operands.

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 cf 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 the operand memory cycle times, and in the first case 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 0.4 microsecond 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.

Conclusion

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 instructions 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.

Discussion

C. H. Richards (Convair-Astronautics): What is the accumulator length of TX-2, and where is the binary point located?

Mr. Frankovich: This is a 36-bit word accumulator, in a ones-complement machine. The binary point really exists only by virtue of what happens during multiplication or division type instructions. The left digit is the sign digit of whatever configuration you have, and the remaining digit is a numeric digit; and ordinarily during multiplication you can consider the binary point to be between the sign digit and the first significant digit on the remainder of the operand. During division, however, we have a different interpretation, so we cannot really say that this is a fractional machine. During addition it makes no difference where you put the binary point. During division the quotient is generated in a different register than the accumulator, so we cannot say that it is a fractional machine during that operation.

Chairman Pfister: When you multiply two 36-bit words, together you have a 72-bit product; where does the product go when you have an accumulator with only 36 bits?

Mr. Frankovich: There are 4 registers in the arithmetical element; and another register which acts as the right-hand extension of this accumulator—a B register. During multiplication the full 72-digit product is generated in the accumulator in the B register. The binary point is at the left-hand of the accumulator during the entire process. The other two registers are used, one to hold the partial carry during addition operations; another is used to carry out division, thereby enabling the arithmetical element to be completely selfcontained during such a long period of instruction.

D. L. Shell (General Electric): What happens on overflow?

Mr. Frankovich: We have four overflow indicators in the arithmetical element. If we have a full 36-bit operand for an instruction, then we use only the left-most overflow indicator, and associate one overflow indicator with each quarter; none of the other overflow indicators are affected at all. On the other hand, if we have four 9-bit operands, then we use all four overflow indicators to indicate overflow for any one of them.

I might also mention that during the jump on overflow instruction you can specify it by means of configuration control, in a very straightforward manner. There are further techniques for handling such situations which are devised to make programming easier.

Mr. Groelinger (Ramo-Wooldridge): Can the exchange element be used to store ac cumulator content in *several* places in memory?

G. G. Chapin (Remington Rand UNIVAC): Can you read from one memory element into more than one arithmetical element?

Mr. Frankovich: This can be done in two ways. As far as the one and one instruction in each transfer: if you want to store one-half of the arithmetical element in several places in the memory, be it in the left half, or the right half, wherever your location might be, then you give instructions to each transfer, unless the transfer were to be done in the same register. If you are loading the arithmetic element, you can load either half of the accumulator from a given memory register, but again this takes two instructions.

G. G. Chapin (Remington Rand UNIVAC): Can jump instructions be conditioned on more than one 9-bit section of the accumulator simultaneously?

Mr. Frankovich: Yes. The configuration control device is used universally and homogeneously upon all arithmetical instructions. If you have four 9-bit operands, and you want to jump on the basis of two of them, the jump instruction is interpreted to be "jump on either the first-quarter or the third-quarter."