Chapter 34

The engineering design of the Stretch computer[1]

Erich Bloch

Summary The Stretch computer is an advanced scientific computer with variable facilities for floating-point, fixed-point, and variable-field-length arithmetic and data-handling facilities.

The performance goal of 100 × 704 speed is achieved by high-speed circuits, multiplexing, and simultaneous-operation technique of instruction and data-fetching, as well as overlap within the execution units. This massive overlap and multiplexing results in complicated recovery routines between the look-ahead and instruction units. These units are described in detail, as are the arithmetic units and significant algorithms used in the floating-point arithmetic.

A flexible set of circuits using a current-switching technique with overriding-level facility is described, as well as the packaging of circuits on printed cards. The frame and gate concept is also shown. Performance figures and hardware count illustrate the size, complexity, and performance of the system.

Introduction

The Stretch computer [Dunwell, 1956] project was started in order to achieve two orders of magnitude of improvement in performance over the then existing 704. Although this computer, like the 704, is aimed at scientific problems such as reactor design, hydrodynamics problems, partial differential equation etc., its instruction set and organization are such that it can handle with ease data-processing problems normally associated with commercial applications, such as processing of alphanumeric fields, sorting, and decimal arithmetic.

In order to achieve the stated goal of performance, all factors that go into the computer design must contribute towards the performance goal; this includes the instruction set [Buchholz, 1958], the internal system organization, the data and instruction word length, and auxiliary features such as status-monitoring devices, the circuits, packaging, and component technology. No one of them by itself can give this hundred-fold increase in speed; only by the combining and interacting of these contributing factors can this performance be obtained.

[1] *Proc. EJCC*, pp. 48–59, 1959.

This paper reviews the engineering design of the Stretch System with primary concentration on the central computer as the main contributor to performance. In it, these new techniques, devices, and instructions have been pushed to the limit set by the present technology and, therefore, its analysis will convey best the problems encountered and the solutions employed.

The Stretch system

Early in the system design, it appeared evident that a six-fold improvement in memory performance and a ten-fold improvement in basic circuit speed over the 704 was the best one could achieve. To meet the proposed performance criteria, the system had to be organized in such a way that it took advantage of every possible overlap of systems function, multiplexing of the major portion of the system, processing of operations simultaneously, and anticipation of occurrences, wherever possible. The system had to be capable of making assumptions based on the probability that certain events might occur, and means had to be provided to retrace the steps when the assumption proved to be wrong.

This simultaneity and multiplexing of operations reflects itself in the Stretch System at all levels, from overall systems organization to the cycle of specific instructions. In the following description, this will be discussed in more detail.

If one considers the Stretch System (Fig. 1) from an overall point of view it becomes apparent that the major parts of the system can operate simultaneously:

a The 2-μsec, 16,384-word core memories are self-contained, with their own clocks, addressing circuits, data registers and checking circuits. The memories themselves are interleaved so that the first two memories have their addresses distributed *modulo* 2 and the other four are interleaved *modulo* 4. The *modulo*-2-interleaved memories are used primarily for instruction storage; since, for high-performance instructions, halfword formats are used, the average rate of obtaining instructions is one per ½ μsec. Similarly, a 0.5-μsec

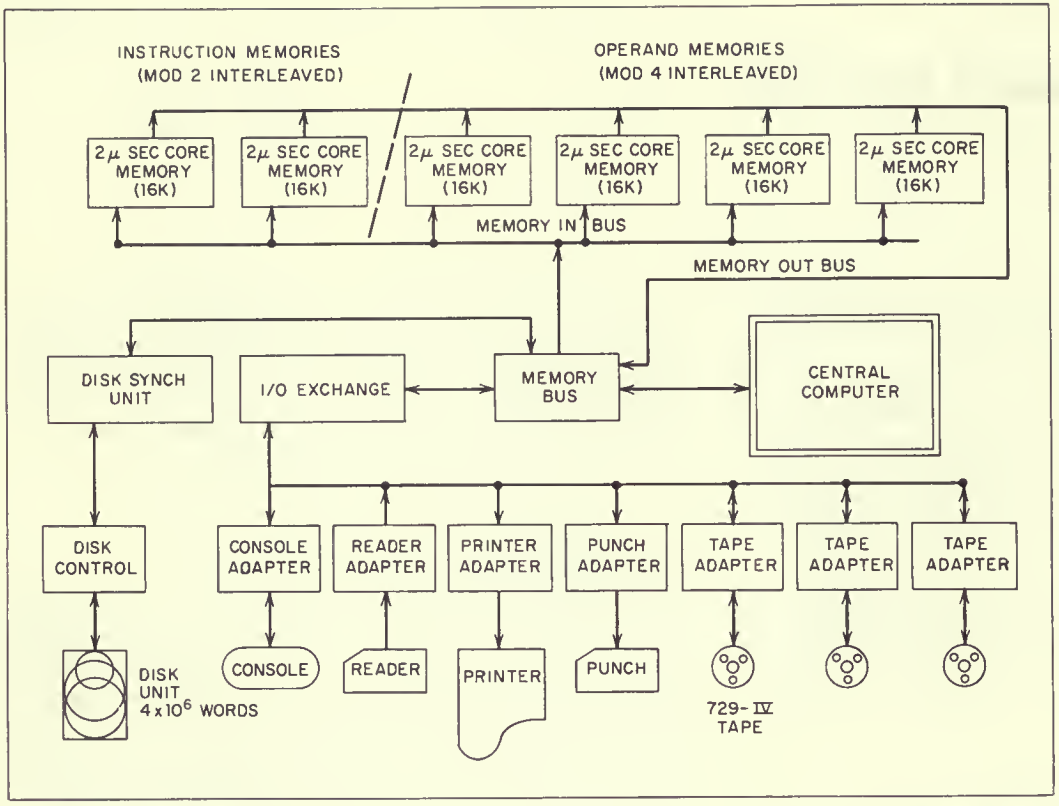

Fig. 1. The Stretch system.

data-word rate is achieved by the use of four *modulo*-4 organized memories. The addressing of the memories and the transfer of information from and to the memories by a memory bus permits new addresses, information, or both to pass through the bus every 200 mμsec.

b The simultaneously-operating Input/Output units are linked with the memories and the computer through the Exchange, which, after initial instruction by the computer, coordinates the starting of the I/O equipment, the checking and error-correction of the information, the arrangement of the information into memory words, and the fetching and storing of the information from and to memory. All these functions are executed without the use of the computer, so it can in the meantime continue its data processing and computation.

c The central computer processes and executes the stored program. Here, now, the simultaneity and multiplexing of functions has reached its ultimate.

Before discussing the computer organization, a few general features must be mentioned for completeness:

a Word length: 64 bits plus eight bits for parity checks and error-correction codes.

b Memory capacity and addressing: A possible 256,000 words can be randomly addressed. These storage positions are all in external memory, except for the 32 first addresses. These positions consist of the internal registers (accumulators, time clocks, index registers).

c The instructions are single-address instructions with the exception of a number of special codes that imply the second address explicitly.

The instruction set (Fig. 2) is generalized and contains a full set for single- and double-precision floating-point arithmetic, and a full set for variable-field-length integer arithmetic (binary and decimal). It also has a generalized set for index modification and a branching set, as well as a set of

I/O instructions. All told, 765 different types of instructions are used in the system.

d The instruction format (Fig. 3) makes use of both half and full words; half words accommodate indexing and floating-point instructions (for optimum performance these two sets of instructions use a rigid format), and full-word formats are used by the variable-field-length instructions. Notice that the latter specifies the operand field by the address of its left-most bit, the length of the field, and the byte[1] size, as well as the starting point (offset) of the implied operand (accumulator). Both halves of the word are independently indexable.

e A general monitoring device used for important status triggers is called the Interrupt [Brooks, 1957] System. This system monitors the flip-flops which reflect internal malfunctions, result significance (exponent range, mantissa zero, overflow, underflow), program errors (illegal instruction, protected memory area), and input/output conditions (unit not ready, etc.). The status of these flip-flops can cause a break in the normal progression of the stored program for fix-up purposes. Their status is automatically interrogated at all times.

[1] *Byte:* a generic term to denote the number of bits to be operated on as a unit by a variable-field-length instruction.

COMPUTER VOCABULARY

INSTRUCTION CATEGORY	CLASS	MODIFIER	EXAMPLES	NUMBER OF INSTR
VARIABLE FIELD LENGTH ARITHMETIC	BINARY DECIMAL	SIGNED UNSIGNED SAME SIGN NEGATIVE SIGN	ADD (TO MEMORY) LOAD/STORE MPY DIVIDE CUMULATIVE MPY	280
RADIX CONVERSION	BIN/DEC			32
LOGIC CONNECTS			16 LOGIC STATEMENT	48
FLOATING POINT ARITHMETIC	NORMALIZED UNNORMALIZED	SAME SIGN OPPOSITE SIGN NEGATIVE SIGN NOISY MODE	ADD (SINGLE & DOUBLE) LOAD/STORE MPY/(SINGLE & DOUBLE) DIV (WITH REMAINDER) INTERCHANGE DIVIDE CUMULATIVE MPY SQUARE ROOT	240
INDEXING ARITHMETIC	DIRECT IMMEDIATE PROGRESSIVE			43
BRANCHES STORE INST CTR	UNCONDITIONAL INDEXING INDICATOR BIT }	IF { 1 { 0 SET 0 LEAVE BIT INVERT BIT		68
TRANSMIT/SWAP I/O INSTRUCTION				24
			TOTAL	735

Fig. 2. The instruction set.

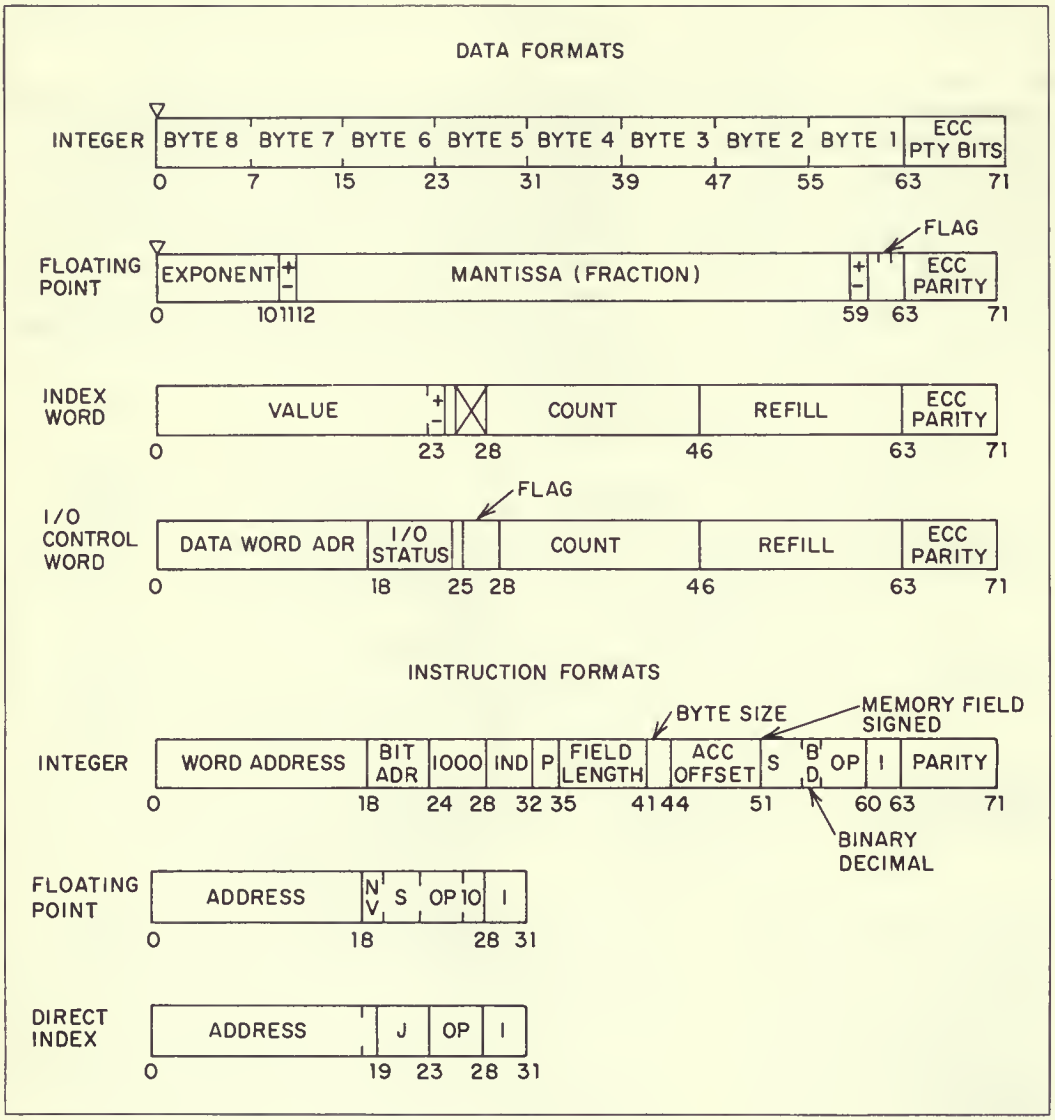

Fig. 3. Data word—and instruction word formats.

The Stretch computer

If one considers the internal organization of the majority of computers that have been produced during the last eight years (and the 704 is a case in point), the organization looks as shown in Fig. 4*a*. There is a sequential flow of instructions into the computer, and after due processing and execution, the next instruction is called from memory. Compare this with Fig. 4*b*, showing the organization of Stretch, where two instruction words and four operands can be fetched simultaneously. In addition, the execution of the instruction is done in parallel and simultaneously with the described fetching functions.

All the units of the computer are loosely coupled together, each one controlled by its own clock system, which in turn is synchronized by a master oscillator. This multiplexing of the units of the computer results in a large number of registers and adders, since

time-sharing of the major computer organs is no longer possible. All in all, the computer has 3,000 register positions and about 450 adder positions.

Despite the multiplexing and simultaneous operation of successive instructions, the result appears as if sequential step-by-step internal operation were utilized. This has made the design of the interlocks quite complex.

Data flow

The data flow through the computer is shown in Fig. 5 and is comparable to a pipeline which in a steady state (namely, once filled) has a large output rate no matter what its length. The same is true here; after start-up the execution of the instructions is fast and bears no relation at all to the stages it must progress through.

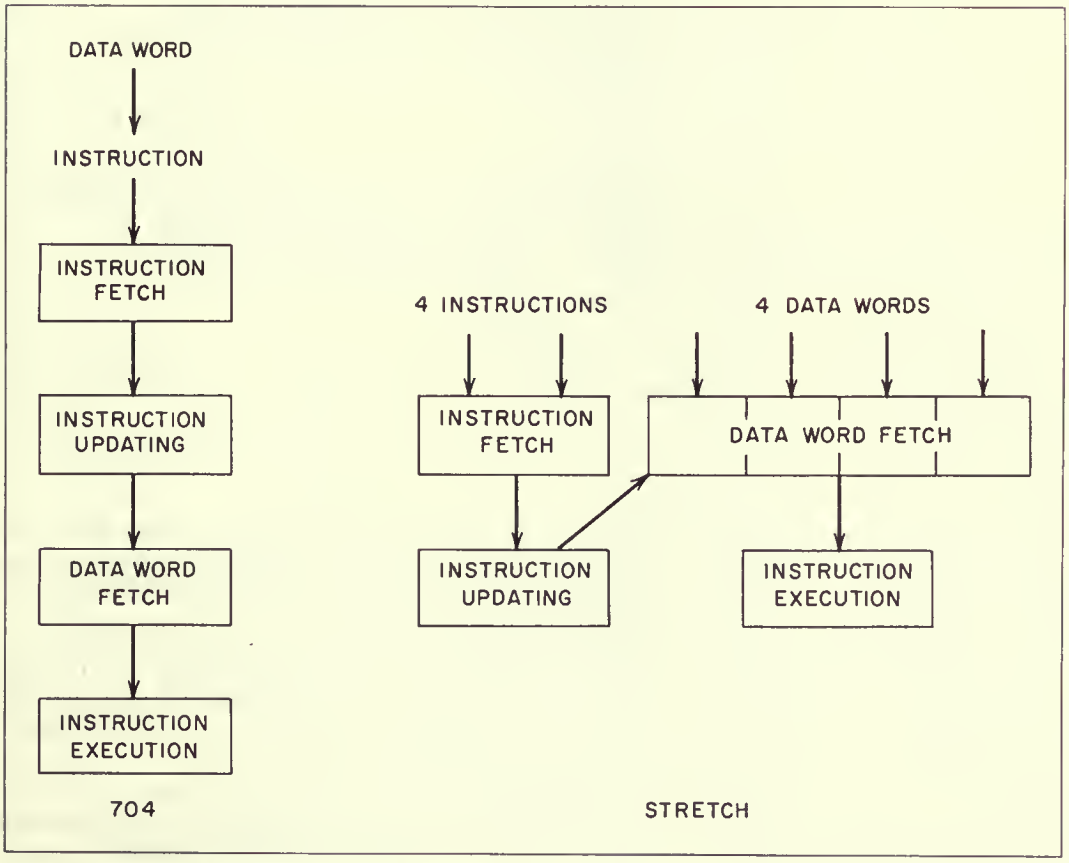

Fig. 4. Comparison of Stretch and 704 organization.

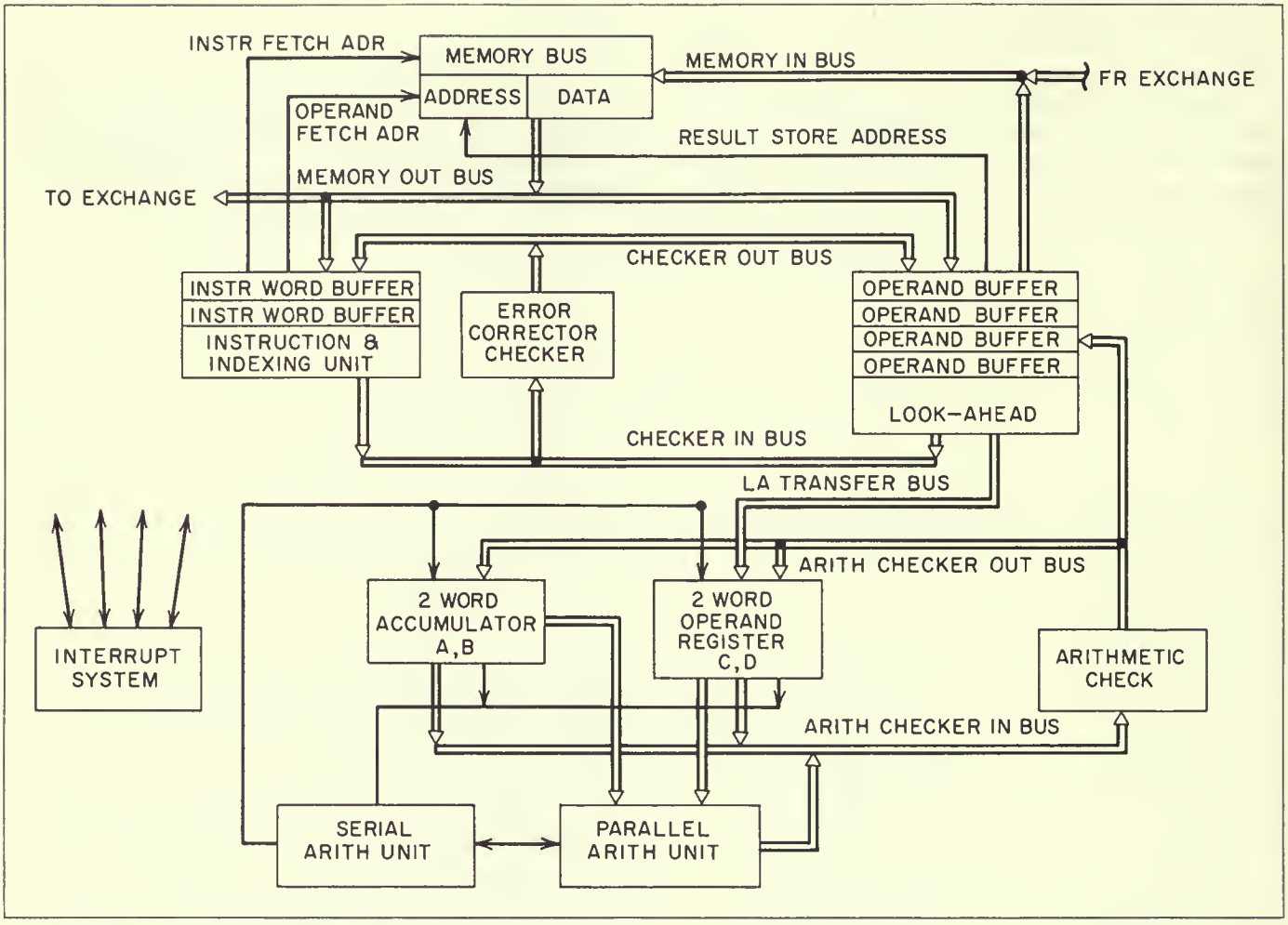

Fig. 5. Stretch computer—units and data flow.

The *Memory Bus* is the communication link between the memories on one side and the exchanges and the computer on the other. It monitors the requests for storage to, or fetches from, memory, and sets up a priority scheme. Since I/O units cannot hold up their requests, the exchange will get highest priority, followed by the computer. In the computer the instruction-fetch mechanism has priority over the operand-fetch mechanism. All told, the memory bus gets requests from and assigns priority to eight different channels.

Since memory can be accessed from multiple sources, and once accessed it is on its own to complete its cycle, a busy condition can exist. Here again, the memory bus tests for busy conditions and delays the requesting unit until memory is ready to be interrogated on data fetches. The return address is remembered and the requesting unit receives the information when it becomes available. To accomplish this, from the time information is requested the receiving data register is in a reserved status.

Requests for stores and fetches can be processed at a 200 mμsec rate and the time, if no busy or priority conditions exist, to return the word to the requesting unit is 1.6 μsec, a direct function of the memory read-out time.

The *Instruction Unit* [Blaauw, 1959] is a computer of its own. It has its own instruction set, its own small memory for index word storage, and its own arithmetic unit. During its operation as many as six instructions can be at various stages of execution.

The Instruction Unit fetches the instruction words from mem-

ory, it steps the instruction counter, and performs the indexing of instructions and the initiation of data fetches. After a preliminary decoding of the class of instruction, it recognizes its own instructions and executes indexing instructions. On branches, conditional or unconditional, the instruction unit executes these. In the case of conditional branches, it makes the assumption that the branch will not be successful.

This assumption and the availability of two full-word buffer registers keep the flow of instruction to the computer continuous. Therefore, the rate of instructions entering the instruction unit is for all practical purposes independent of the memory cycle.

Since, for high speed instructions, half-word formats are used, four of these at any one time can be in buffer storage. As soon as the instruction unit starts processing an instruction, it is removed from the buffer, thus making room for the next memory-word access (Fig. 6). Incidentally, half-word instructions and full-word instructions can be intermixed within the same word, and therefore the latter can cross a word boundary. This permits maximum packing of instructions in memory and also serves as a facility for automatic program assemblers and compilers.

The adder path, index registers, and transfer bus to look-ahead complete the instruction unit system (Fig. 6). It should be noted that the index registers are part of the instruction-unit data path, therefore permitting fast access (no long transmission lines) to an index word. There are 16 index words available to the programmer. The index registers, consisting of multi-aperture cores, are oper-

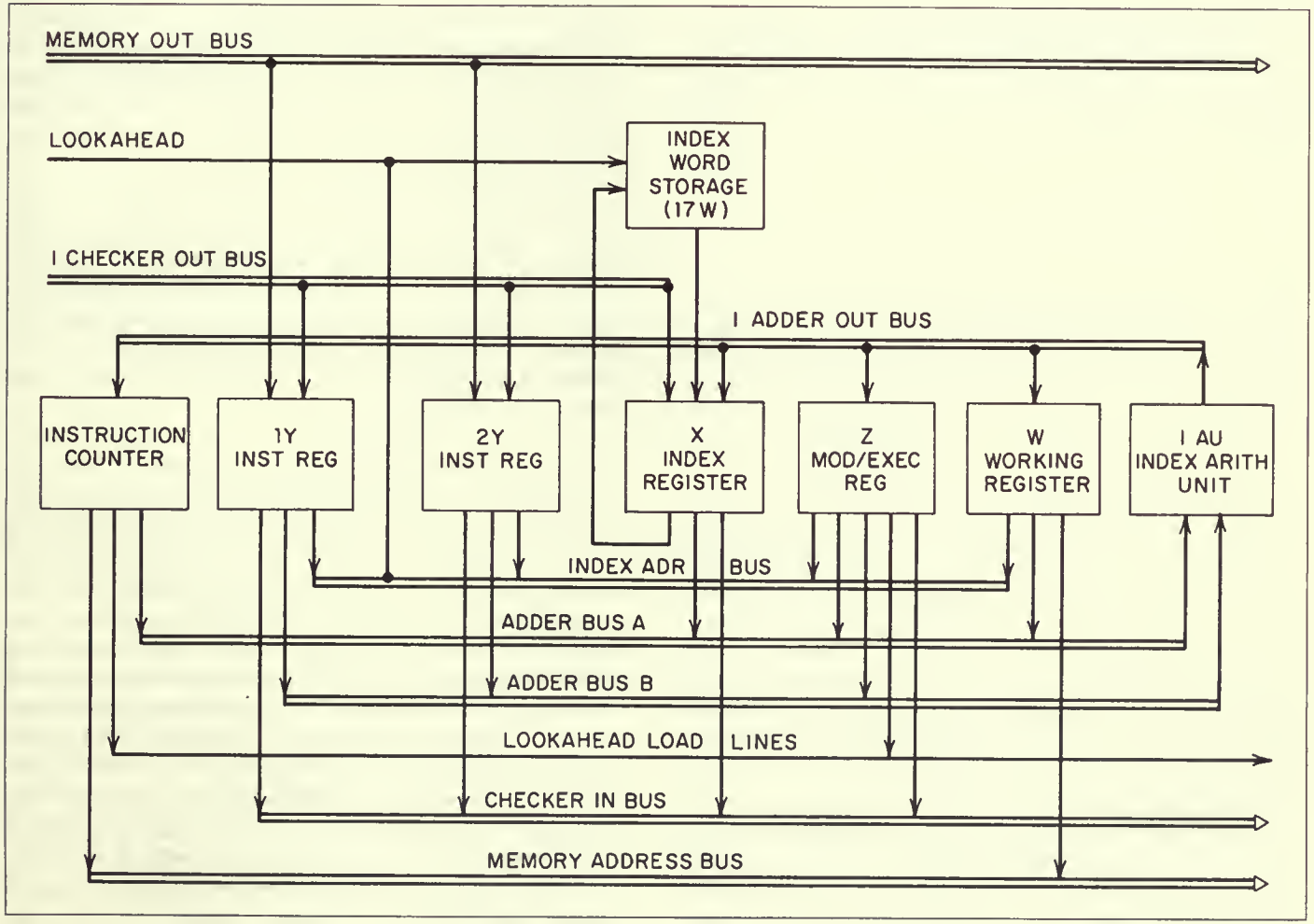

Fig. 6. Instruction unit.

ated in a non-destructive fashion, since in a representative program, the index word is used nine out of ten times without modifying it. This permits fast operation under these conditions, and additional time is only applied where modification is involved.

After processing through the instruction unit, the updated (indexed) instruction enters a level of the *Look-ahead* (Fig. 5). Besides the instruction, all necessary information, its associated instruction counter value, and certain tag information are also stored in the same level. The operand, already requested by the instruction unit, will enter this level directly and will be checked and error-corrected while awaiting transfer to the arithmetic units for execution.

An interlocked counter mechanism in the look-ahead keeps its four levels in step, preventing out-of-sequence execution of instructions, even if all information for a succeeding one is available, before the previous instruction has been started.

The pre-accessing of operands by the look-ahead and of instructions by the instruction unit leads sometimes to embarrassing positions, for which a fix-up routine must be provided. Consider the program

(n)	STORE Accumulator m
$(n + 1)$	LOAD R
$(n + 2)$	ADD m

and assume instruction (n) is in look-ahead, waiting for execution. If $(n + 2)$ now enters the look-ahead, a reference to m cannot be made, since the data stored in that position is subject to change by the STORE instruction. The look-ahead must recognize this and "forward" the result of instruction (n), when received, to the level where $(n + 2)$ is stored.

Another example is the case where the instruction unit assumed that a conditional branch would not be executed. This instruction is stored in look-ahead and, when it is recognized that the branch was successful, all modifications of addressable registers made by the instruction unit in the meantime must be restored. Look-ahead in this case acts as a recovery memory for this information. A similar condition exists when interrupts occur due to arithmetic results. The look-ahead here again has the data stored pertaining to registers which were modified erroneously in the meantime. The restoring and recovery routines described break into the instruction unit processing, interrupting temporarily the flow of instruction and their indexing.

The arithmetic units described later are slaves to the look-ahead, receiving not only operands and instruction codes but also the start-execution signal. Conversely, the arithmetic units signal to the look-ahead the termination of an operation and, in the case of "To Memory" operations, place into the look-ahead the result word for transfer to the proper memory position.

Arithmetic units

The design of the arithmetic units was established along lines similar to the design of look-ahead and the instruction unit. Every attempt was made to speed up the execution of arithmetic operations by multiplexing techniques and overlapping of the algorithm, where mathematically permissible.

The arithmetic units, consisting of the Serial Unit and the Parallel Unit, use the same arithmetic registers, namely a double-length accumulator (A,B) consisting of 128 bits and a double-length operand register (C,D) consisting of 128 bits. The reason for the use of the same arithmetic registers is the fact that at any time, a shift from floating-point to variable-field-length operation (or *vice versa*) can be made by the program. Therefore, the result obtained by a floating-point operation can serve as the starting operand for a variable-field-length operation. The chief reason for the double-length registers is the definition of maximum field length to be 64 bits. The field can start with any bit position, and therefore can cross the word boundary.

The executions of floating-point mantissa operations and variable-field-length binary multiply and divide operations are performed by the parallel unit, whereas the floating-point exponent operation and the variable-field-length binary and decimal add-type operations are executed by the serial unit. The square-root operation and the binary-to-decimal conversion algorithm are executed in unison by both units. Salient features of the two units will now be described.

The serial arithmetic unit [Brooks et al., 1959] (Fig. 7). The serial arithmetic consists of a switch matrix which can extract 16 consecutive bits from A,B and C,D. These 16 bits then can be aligned in such a way that the low-order bit of a field as specified by the instruction is at the right end of the field. This wrap-around circuit then feeds into a carry-propagate adder or, in case of logical-connect instructions, into the logic unit. At the adder output, a true complement unit and a binary-to-decimal correction unit are used for subtract and decimal operations. The inverse process of extracting is used to insert the processed byte back into the register without disturbing any neighboring positions. Notice that in one clock cycle, the information is extracted, the arithmetic is performed and the result inserted back into the registers. In addition, the arithmetic information is checked by parity checks on the switch matrices and by duplication and comparison of the arithmetic procedure in a duplicate unit.

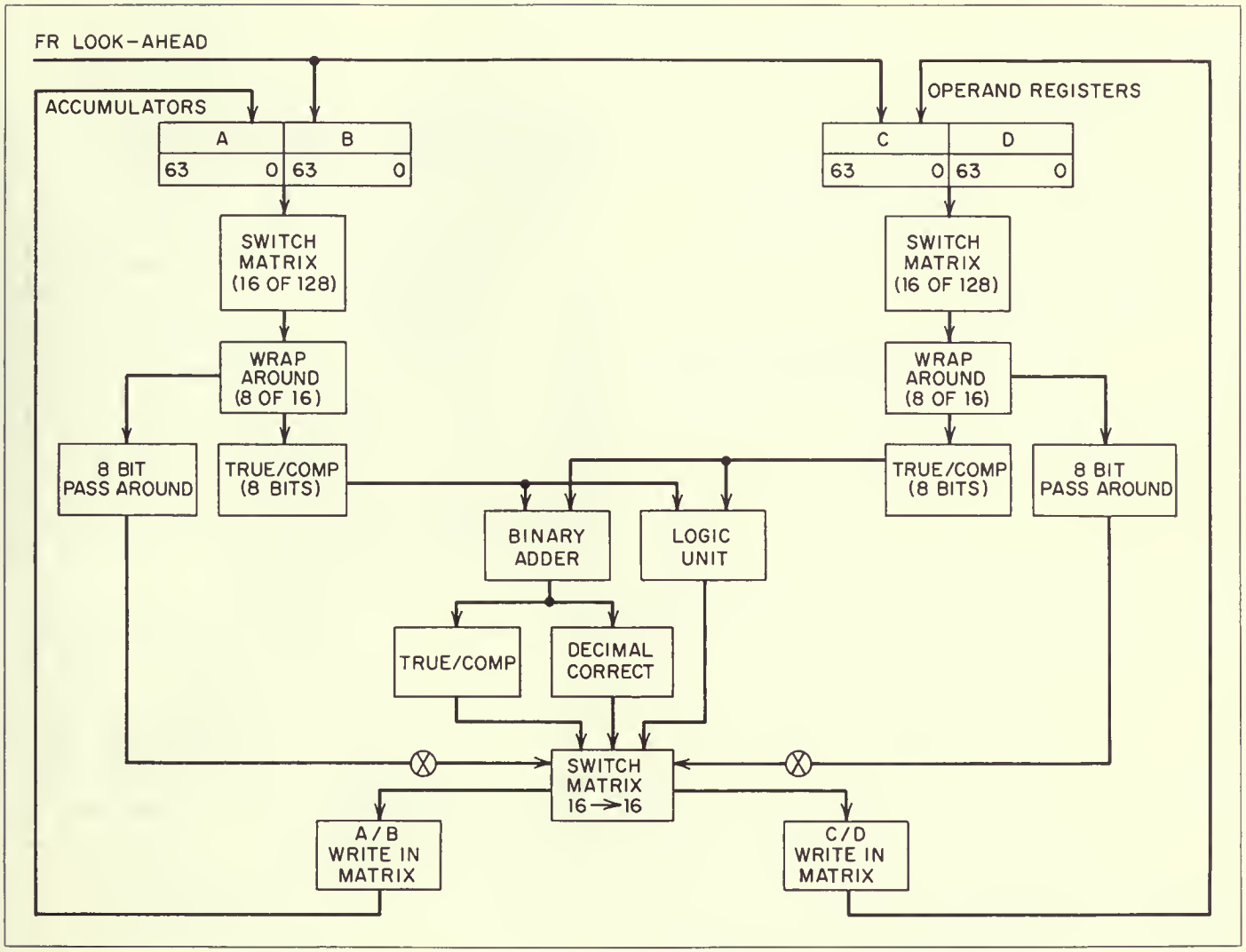

Fig. 7. Serial arithmetic unit.

Parallel arithmetic unit. The parallel arithmetic unit (Fig. 8) is designed to execute floating-point operations with a maximum of efficiency. Since both single- and double-precision arithmetic is performed, the shifter and adder exist in a double-length format of 96 bits. This insures almost the same performance for single- and double-precision arithmetic. The adder is of a carry-propagation type with look-ahead over 4 bits at a time to reduce the delay that normally results in a ripple-carry adder. This carry look-ahead results in a delay time of 150 mμsec for 96-bit binary-number additions. All additions and subtractions are made in one's complement form with automatic end-around carry.

The shifter is capable of shifting up to 4 positions to the right and up to 6 positions to the left. This shifter arrangement takes care of the majority of shifting operations encountered under normal operation. Where higher-order shifts are required, a successive operation is set up between the parallel unit register and the shifter.

To expedite the execution of the multiply instruction, 12 bits

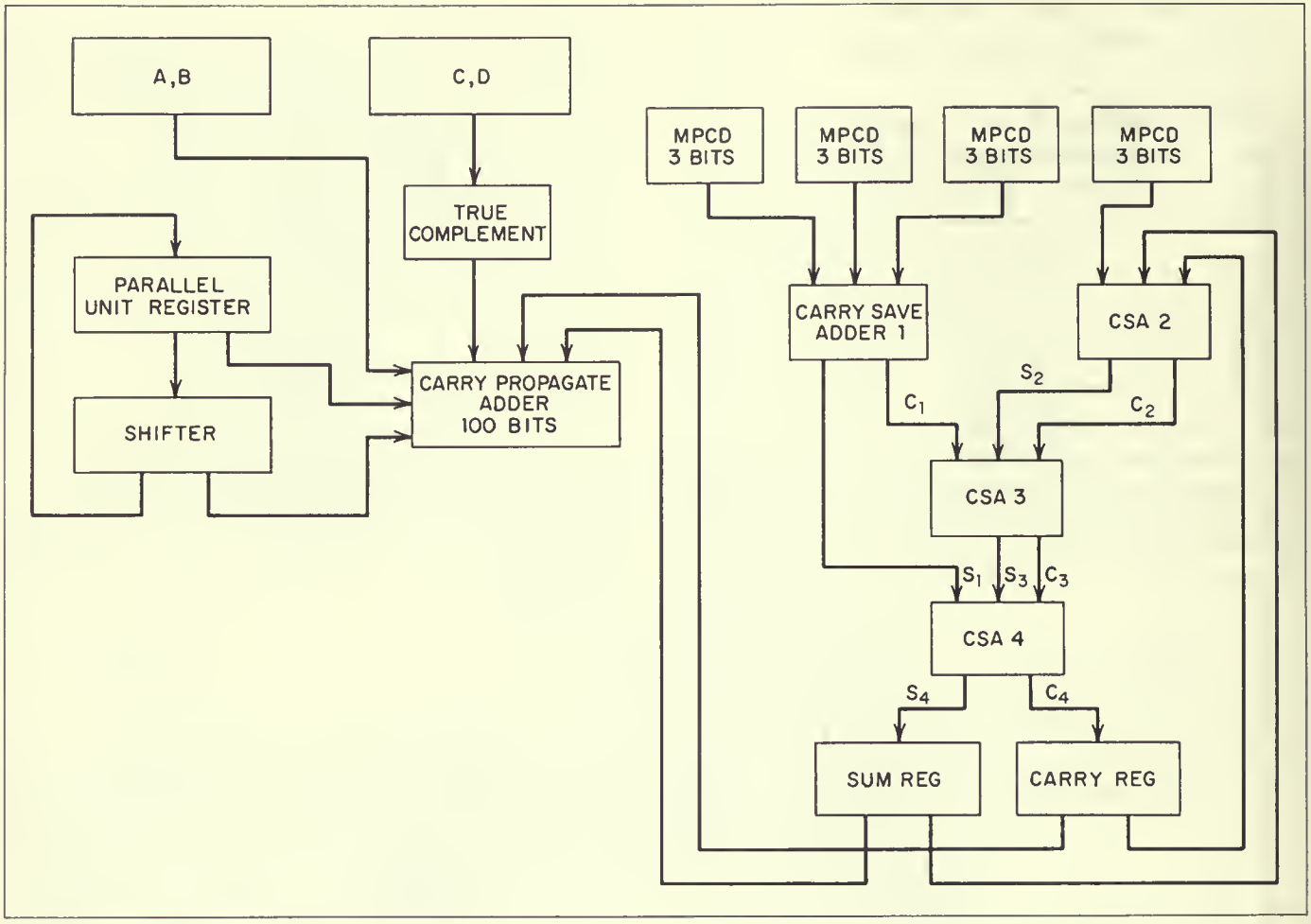

Fig. 8. Floating-point arithmetic unit.

of the multiplier are handled within one cycle. This is accomplished by breaking the 12 bits into groups of three bits each. The action is from right to left and consists of decoding each group of three bits. By observing the lowest-order bit of the next higher group, a decision is made as to what multiple of the multiplicand one must add to the partial product. Since only even multiples of the multiplicand are available, subtraction and addition of the multiples can result. The following example will elaborate this point: (*MCD* means multiplicand)

		Groups		
$n+4$	$n+3$	$n+2$	$n+1$	n
		Multiplier, 12 bit group		
xx0	011	110	101	010
		Octal value		
3		6	5	2
		If two additions of multiples were permitted		
$4 \times MCD$		$6 \times MCD$	$6 \times MCD$	$2 \times MCD$
$-1 \times MCD$			$-1 \times MCD$	
		Instead of subtracting $1 \times MCD$ in $n+1$, subtract $8 \times MCD$ in n.		
$4 \times MCD$		$6 \times MCD$	$6 \times MCD$	$2 \times MCD$
		$-8 \times MCD$		$-8 \times MCD$
		Resulting decoding		
$4 \times MCD$		$-2 \times MCD$	$6 \times MCD$	$-6 \times MCD$

The four multiple multiplicand groups and the partial product of the previous cycle are now fed into carry-save adders of the form,

Sum $S = A \veebar B \veebar C$
Carry $C' = AB + AC + BC$

There are four of these adders, two in parallel followed by two more in series (Fig. 8). The output of Carry-Save Adder 4 then results in a double-rank partial product, the product sum and the product carry. For each cycle this is fed into Carry-Save Adder 2, and, during the last cycle, into the carry-propagate adder, for accumulation of the carries. Since no propagation of carries is required in the four cycles, where multiple multiplicands are added, this operation is fast and is the main contributor to the fast multiply-time of Stretch.

The divide scheme [Robertson, 1958] has a similarity to the multiply scheme. Multiples of the divisor are used, namely, 3/2 × divisor, 3/4 × divisor and 1 × divisor. This, plus shifting over strings of ones and zeros, results in the generation of the required 48 quotient bits within thirteen machine cycles. Most machines using a nonrestoring divide method require 48 cycles for 48 quotient bits. The following example explains this technique. This scheme depends on the use of normalized divisors:

DIVIDEND (DD) = 101000000000000
DIVISOR (DR) = 1100011
2's COMP DR ($\overline{DR}$) = 0011101
3/4 DR = 100101001

(*a*) *Using skip over 1/0 only:*

	101000000000000	DIVIDEND
Step 1:	0011101	ADD $\overline{DR}$
	1101101	

Remainder negative, 1st quotient bit = 0; shift one position. Leading 1 indicates that next quotient bit must be 1; $Q_1Q_2 = 01$

	011010000	REMAINDER
Step 2:	1100011	ADD DR
	10010111	

Overflow: Remainder positive and $Q_3 = 1$, leading zero indicates $Q_4 = 0$

	1011100	REMAINDER
Step 3:	0011101	ADD DR
	1111001	

Negative remainder; $Q_5 = 0$; leading 1's indicate $Q_6Q_7Q_8 = 111$

Number of quotient bits per cycle:

Cycle 1: 01 = 2
Cycle 2: 10 = 2
Cycle 3: 0111 = 4

(*b*) *The same problem with both skip over 1/0* and *3/4 – 3/2 complement:*

	101000000000000
Step 1:	0011101
	11011010000

Same as before, $Q_1Q_2 = 01$

Step 2:	100101001	Add 3/4 DR
	111111001	

This (by table look-up) indicates $Q_3Q_4Q_5Q_6Q_7Q_8 = 100111$

Quotient bits generated per cycle:

Cycle 1: 01 = 2
Cycle 2: 100111 = 6

In general, this method results in the generation of 3.7 quotient bits per subtraction. While the mantissa operations of multiply and divide are performed by the parallel unit, the serial arithmetic unit executes the exponent arithmetic. Here again is a case where overlap and simultaneity of operation is used to special advantage.

Checking. The operation of the computer is checked in its entirety and correction codes are employed where data transfers from memory and input-output units are involved. In particular, all information sent to memory has a correction code associated with it, which is checked for accuracy on its way from memory. If a single error is indicated, then correction is made and the error is recorded via a maintenance output device. Within the machine, all arithmetic operations are checked, either by parity, duplication, or a "casting out three" process. These checks are overlapped with the execution of the next instruction.

Hardware count. Figure 9 shows the percentage of transistors used in the various sections of the machine. It becomes obvious that the parallel unit and the instruction unit use the highest percentage of transistors. In case of the parallel unit this is due to the extensive circuits for multiply and to the additional hardware to achieve speed up of the divide scheme. In the instruction unit, the controls consume the majority of the transistors, because of the high multiplexed operation encountered.

Performance. The performance comparisons in Fig. 10 show the increase in speed achieved, especially in floating-point operations,

UNIT	NO. OF TRANSISTORS	% OF TOTAL	NO. OF FRAMES
MEMORY CONTROLS	10,500	6.0	2
INSTRUCTION UNIT			
DATA PATH	17,700	22.0	2
CONTROLS	19,500		3-1/2
LOOK-AHEAD			
DATA PATH	17,900	15.6	1
CONTROLS	8,600		1-1/2
ARITH REGISTERS	10,000	5.9	1
SERIAL ARITH UNIT			
DATA PATH	10,000	10.5	1-1/2
CONTROLS	8,700		1
FLOATING PT UNIT			
DATA PATH	32,700	21.0	2-1/2
CONTROLS	3,000		1/2
CHECKING	24,500	14.5	1
INTERRUPT SYSTEM	6,000	3.5	1/2
TOTAL	169,100	100.0	18

DOUBLE CARDS	4,025
SINGLE CARDS	18,747
POWER	21 KW

Fig. 9. Component count.

over the 704. It should be noted that for a large number of problems this particular increase in all arithmetic speeds is almost proportional to the performance increase of the problem as a whole, since the instruction execution-times are overlapped to a great extent with the preparation and fetching of instructions. Simulation of Stretch programs on the 704 proved a performance of 100 × 704 speed in mesh-type calculations. Higher performance figures are achieved where double- or triple-precision calculations are required.

Circuits

Having reviewed the systems organization of Stretch, it is now of interest to discuss briefly the components, circuits, and packaging techniques used to implement the design.

The basic component used in Stretch is the high-speed drift transistor which exists in both an NPN and a PNP version. This transistor has a frequency cut-off of approximately 100 mc and for high-speed operation must be kept out of saturation at all times. This then explains why both the PNP and NPN version are used: mainly to avoid the problem of level translation, which would be required due to the potential difference of the base and the collector. This difference is 6 volts, an optimum point for this device.

Figure 11 shows the basic circuit configuration. It consists of a current source, represented by the −30 volt supply and resistor *R*. The functional operation of the circuits consists of two possible

	OPERATION	IBM 704	IBM 705	STRETCH
	1. FLOATING POINT			
		±128		±2048
	EXPONENT RANGE	±2		±2
	MANTISSA BITS	27		48
	FLOATING ADD	84 μSEC		1.0 μSEC
	FLOATING MPY	204 μSEC		1.8 μSEC
	FLOATING DIV	216 μSEC		7.0 μSEC
	LOAD/STORE	24 μSEC		0.6 μSEC
	2. BINARY VARIABLE			
	FIELD LENGTH ARITH			
	BIT RANGE			1 TO 64
16 BIT FIELD	ADD/LOAD/STORE			2.0 μSEC
16 BIT FIELD	MPY			10.0 μSEC
16 BIT FIELD	DIVIDE			15.0 μSEC
	3. DECIMAL			
	ARITHMETIC			
	DIGIT RANGE		1 → MEM CAPACITY	1 TO 21
FOR 5 DIGITS	ADD		119 μSEC	3.5 μSEC
FOR 5 DIGITS	MPY		799 μSEC	40.0 μSEC
FOR 5 DIGITS	DIVIDE		4828 μSEC	65.0 μSEC
FOR 5 DIGITS	LOAD/STORE		204 μSEC	3.2 μSEC
	4. MISCELLANEOUS			
	ERROR CORRECTION	NO	NO	YES
	CHECKING	NO	YES	YES
	WORD SIZE	36 BITS		64 BITS

Fig. 10. Comparison of Stretch and 705/704 operation times.

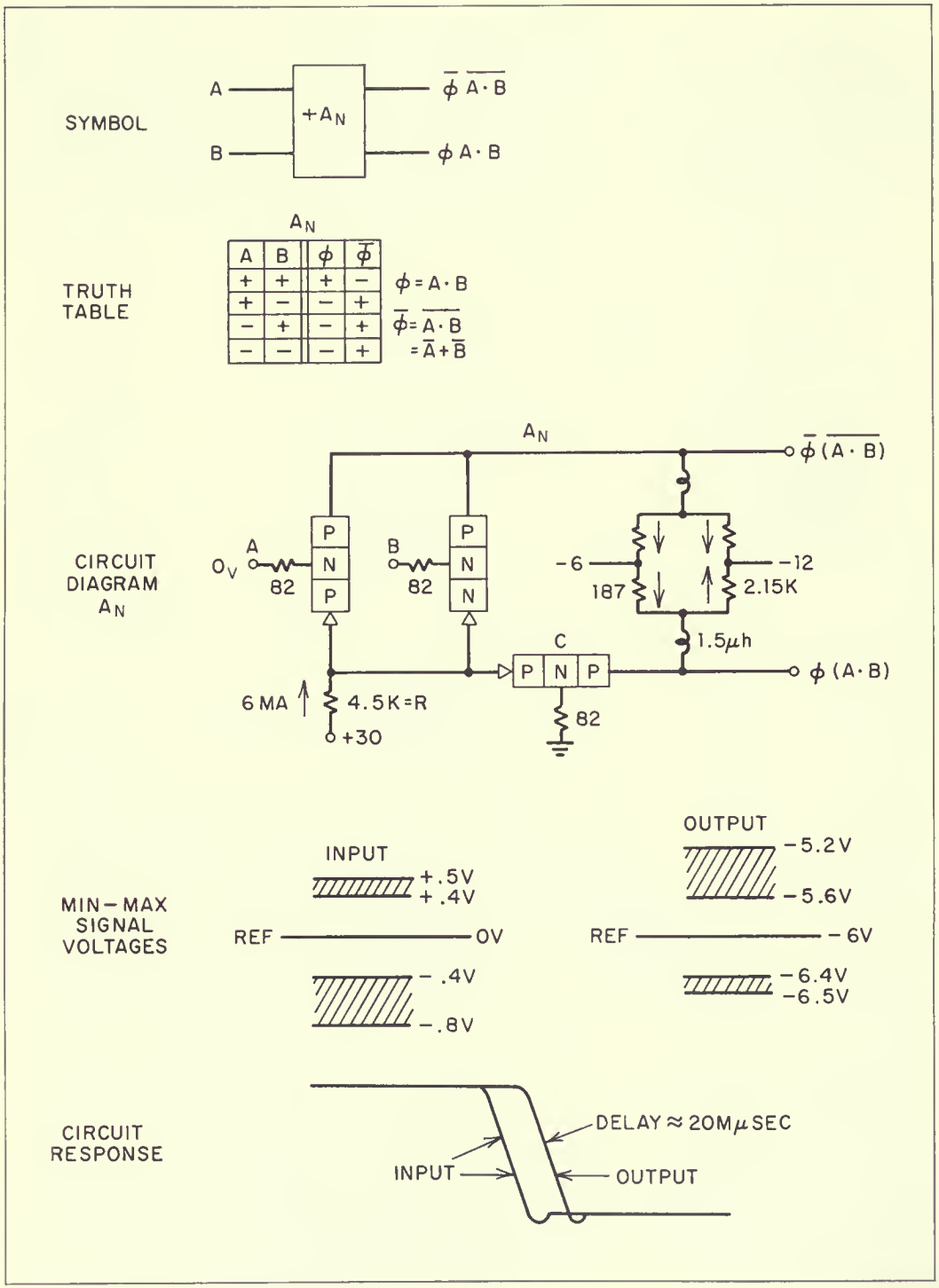

Fig. 11. Current switching circuits (+AND).

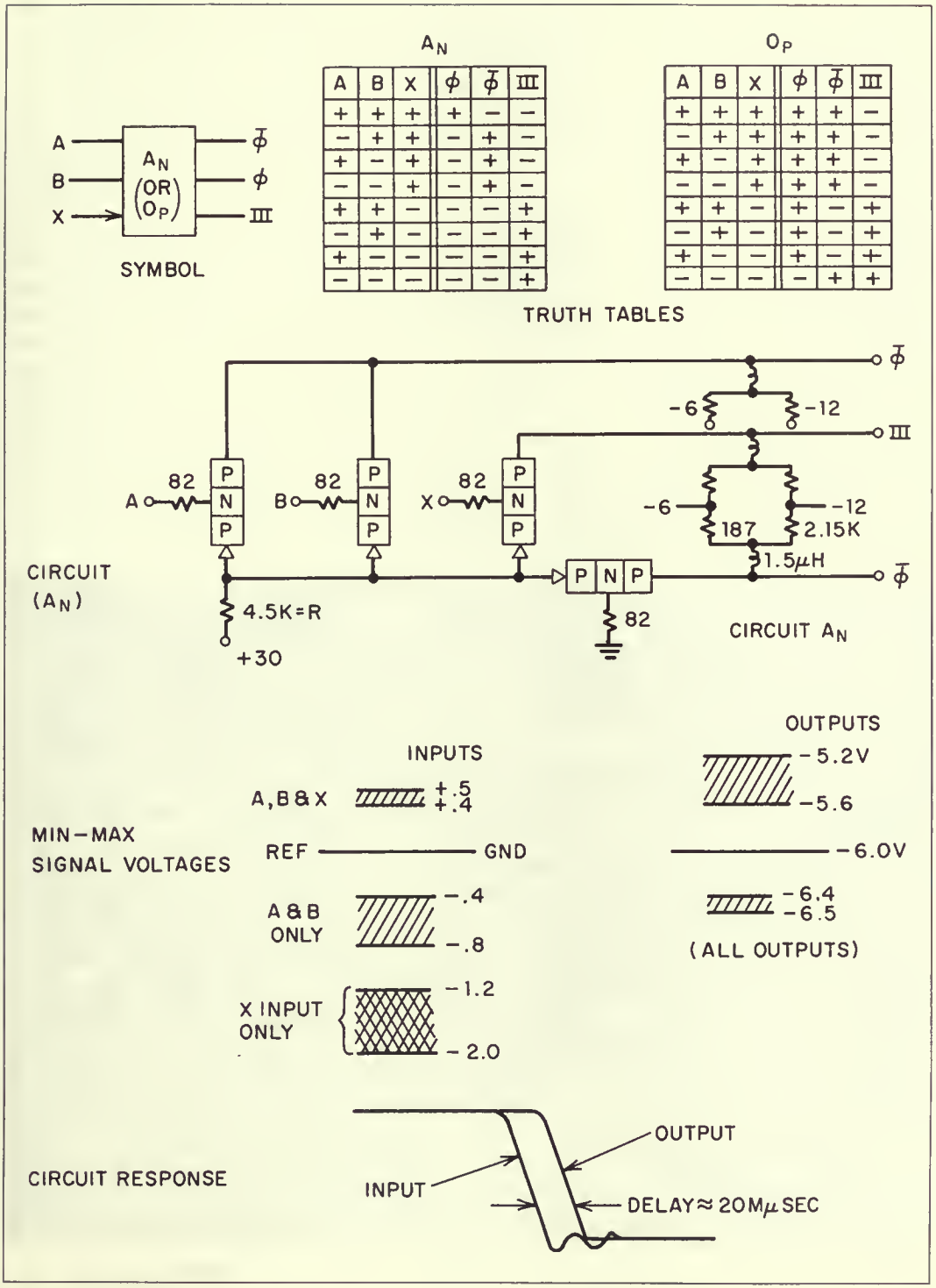

Fig. 12. Third-level circuit.

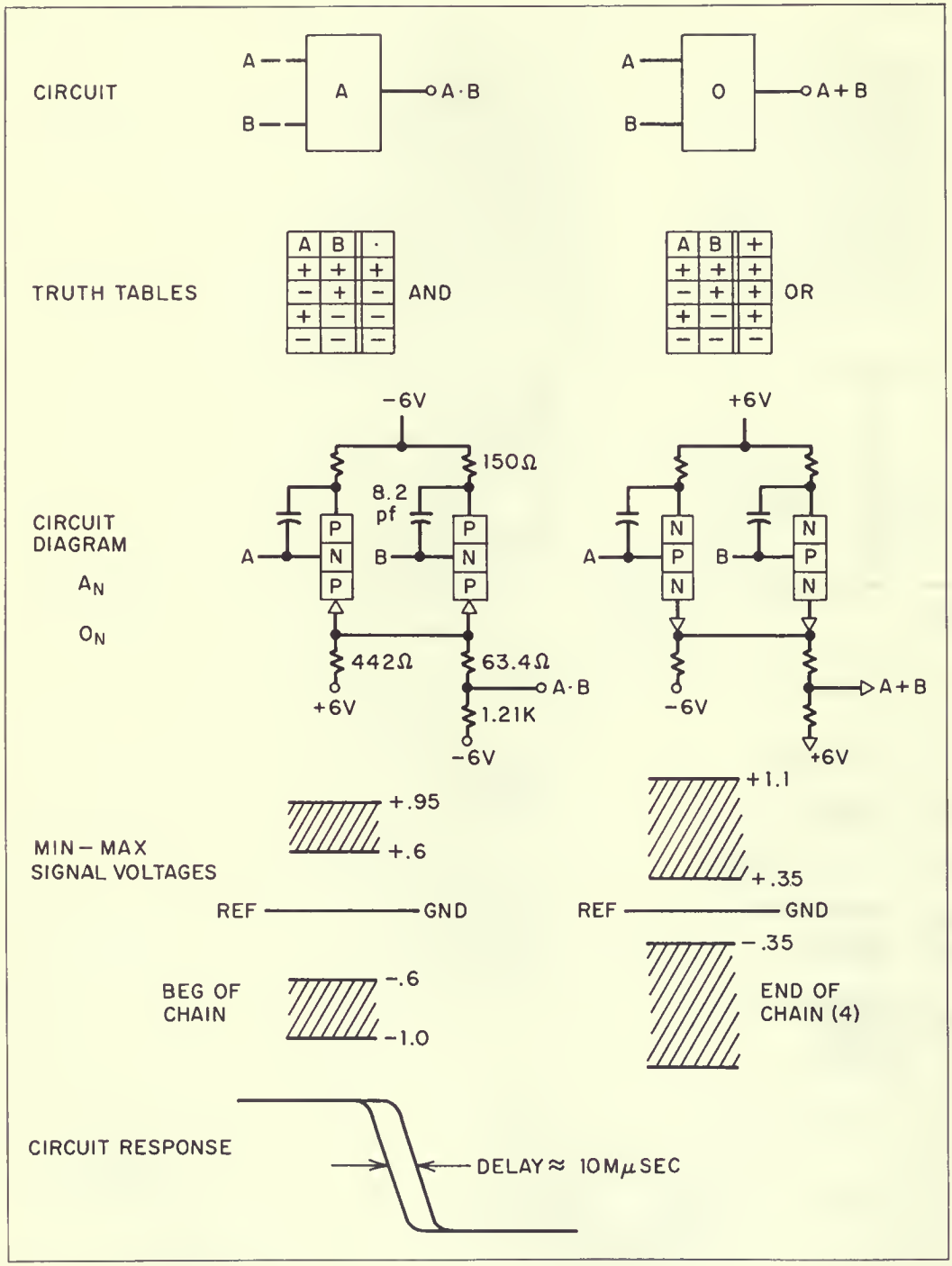

Fig. 13. Emitter-follower circuit.

paths represented by transistor A or C. Which path is chosen by the current depends on the condition existing on base A. If point A is positive with respect to ground by 0.4 volts, that particular transistor is cut off, making the emitter of transistor C positive with respect to the base and, therefore, making C conducting. The current supplied by the current source (6 ma) will then flow through transistor C to the load ϕ. Output ϕ, then, is positive by 0.4 volts with respect to the -6 volt reference. This indicates at ϕ the equivalent function impressed on A. At the same time, $\bar{\phi}$ is negative with respect to the -6 volt power supply by 0.4 volt, representing, therefore, the inverse of the function impressed on A. Conversely if A is negative with respect to the ground reference, transistor A is the conducting one, keeping emitter C negative with respect to its base. The current flows through transistor A, making $\bar{\phi}$ positive with respect to -6 and ϕ negative with respect to -6. Again, the output of ϕ reflects the function impressed on A, whereas $\bar{\phi}$ represents the inverse of the function.

If an additional transistor now is paralleled with A, it becomes obvious that only if both bases A and B are positive will output ϕ be positive and $\bar{\phi}$ negative. If any or none of the bases A and B are positive, then ϕ will be negative and $\bar{\phi}$ will be positive. In other words, an AND function is obtained on output ϕ.

This principle, which is reflected in all the circuits, is essentially the principle of current switching or current steering.

Logical functions for the PNP circuits are, therefore, a $+$AND or $-$OR. Two outputs from each circuit block are available: the AND function and the inverse of the AND function.

A dual circuit exists for NPN transistors with input levels at -6 volts and output levels at ground. This circuit will give the $+$OR or $-$AND function.

A thorough investigation of the systems design showed that the circuits described so far are versatile enough to be used throughout the system. However, there are enough special cases (resulting from the many data buses and registers throughout the machine) that could use a distributor function or an overriding function. This caused the design of a circuit which permitted great savings in space and transistors by adding a third voltage level. Figure 12 shows the PNP version of the third-level circuit.

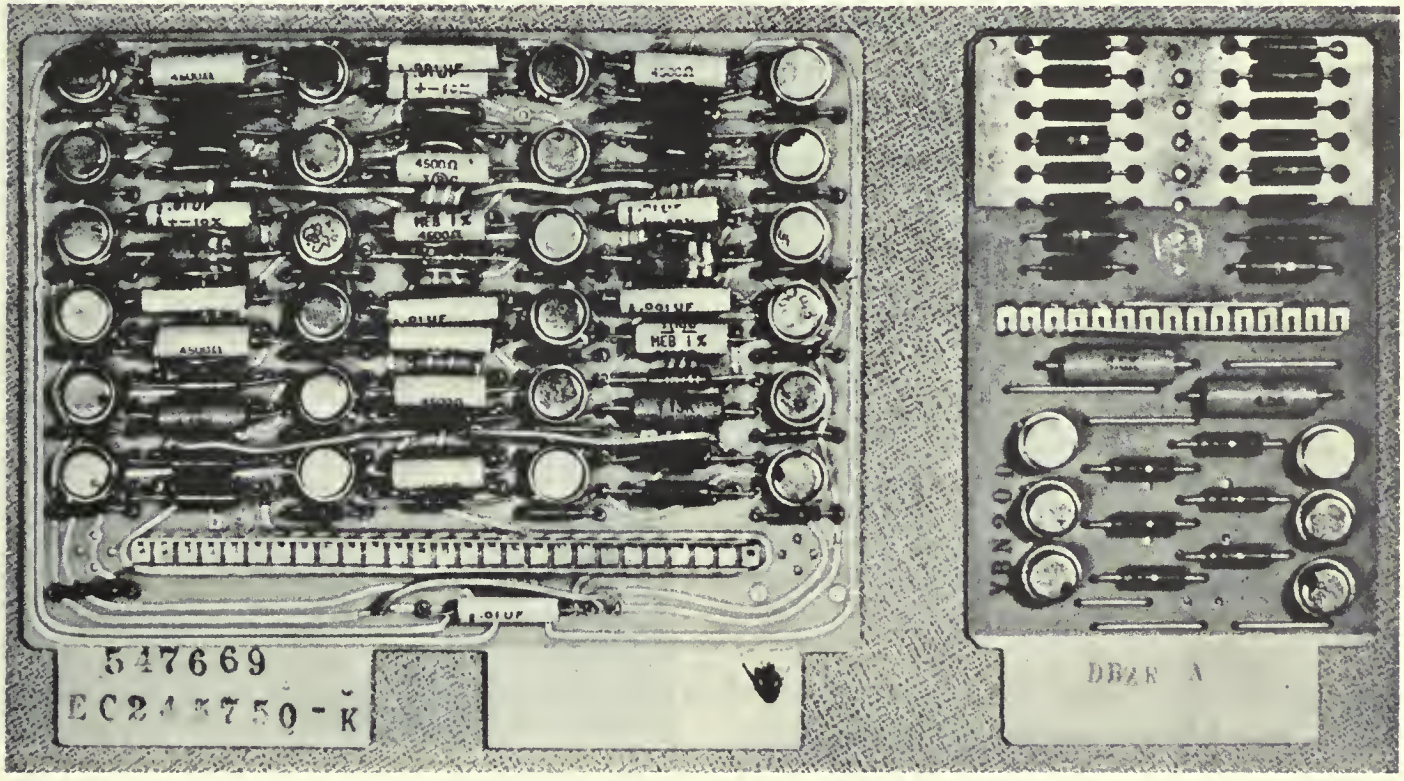

Fig. 14. The circuit package.

If transistor X were eliminated, then transistors A and B in conjunction with the reference transistor C would work normally as a current switching circuit, in this case a +AND circuit. If transistor X is added with the stipulation that the down level of X is more negative than the lowest possible level of A or B, it becomes apparent that when X is negative, the current will flow through that branch of the circuit in preference to branch ϕ or $\bar{\phi}$, regardless of inputs A and B. Therefore, the output of ϕ and $\bar{\phi}$ will be negative, provided input X is negative. Output *III* is the inverse of input X. If, however, X is positive, then the status of A and B will determine the function ϕ and $\bar{\phi}$ implicitly. This demonstrates the overriding function of input X.

Similarly, the NPN version (not shown) results in the OR function of ϕ if input X is negative and in a positive output at ϕ and $\bar{\phi}$, regardless of status A and B, if X is positive. Again minimum and maximum signal swings are shown in Fig. 12.

The speed of the circuits described so far depends on the number of inputs and the number of circuits driven from each load. The response of the circuit is anywhere between 12 and 25 mμsec per logical step with 18 to 20 mμsec average. The number of inputs allowable per circuit is eight. The number of driven circuits is three. Additional circuits are needed to drive more than three bases and where current switching circuits communicate over long lines, termination networks must be added to avoid reflections.

To improve the performance of the computer in certain critical places, emitter-follower logic is used as shown in Fig. 13. These circuits, having a gain less than one, after a number of stages require the use of current switching circuits as level setters and gain devices. Both AND and OR circuits are available for both a ground-level and a −6-level input. Change from a −6-level circuit to a ground-level circuit is obtained by applying the appropriate power supply levels. Due to the variations in inputs and driven loads, the circuits must be designed so that the load can vary over a wide range. This resulted in instability which had to be offset by the feedback capacitor C shown in the circuit.

All functions needed in the computer can be implemented by the use of the aforementioned circuits, including flip-flop operation, which is obtained by tying a PNP current switch block and an NPN current switch block together with proper feedback.

Packaging

The circuits described in the last paragraph are packaged in two ways:

A circuit package using the smaller of the two printed circuit boards shown in Fig. 14, called a single card, contains AND or OR circuits. It should be mentioned that the printed wiring is one-sided and that besides the components and transistors, a rail is added which permits the shorting or addition of certain loads depending on the use of the circuits. This rail then has the effect of reducing the different types of circuit boards in the machine. Twenty-four different boards are used and of these, two types reflect approximately 70% of the total single card population.

Due to the large number of registers, adders, and shifters used in the computer, it seems reasonable that functional packages could be employed economically, because of wide usage. This results in the high-density package also shown in Fig. 14, called

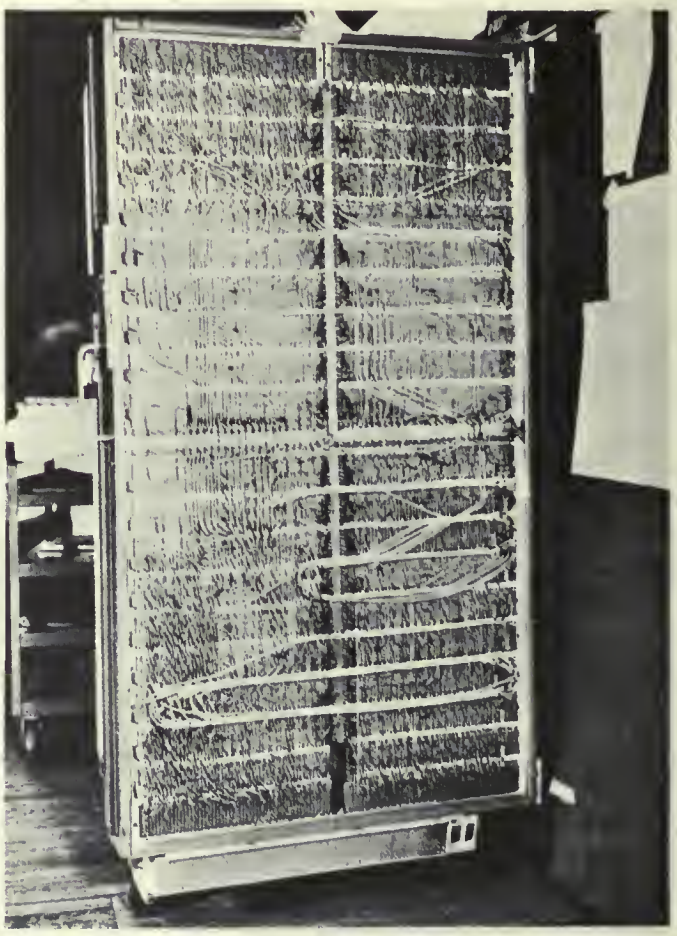

Fig. 15. The back panel.

a Double Card, which has 4 times the capacity of a single card and which has wiring on both sides of the board. Furthermore, components are double-stacked; and again, the rail is used to effect circuit variations due to different applications. Eighteen double card types are used in the system. Approximately 4,000 double cards are used, housing 60% of the transistors. The rest of the transistors are on approximately 18,000 single cards.

The cards, both single and double, are assembled in gates, and two gates are assembled into a frame. Figure 15 shows the gate back-panel wiring, using wire-wraps; and Figs. 16 and 17 the frame construction, both in a closed and open version.

To achieve high performance, special emphasis must be placed on keeping noise to a low level. This required the use of a plane

Fig. 16. The frame (closed).

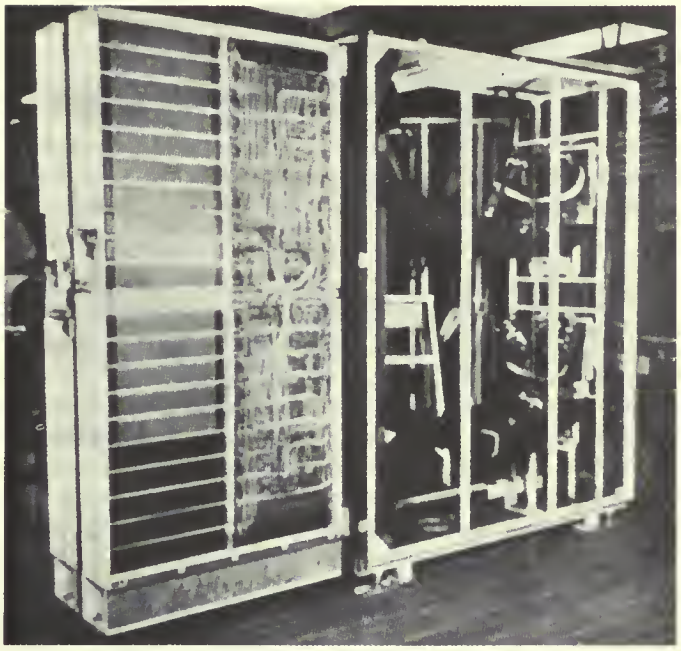

Fig. 17. The frame (extended).

which overlies the whole back panel, against which the intercircuit wiring is laid. In addition, the power-supply distribution system must be of such a low impedance that extraneous noise cannot induce circuit malfunction. For this reason, a bus system, consisting of laminated copper sheets, is used to distribute the power to each row of card sockets. The wiring rules are such that single-conductor wire is used up to a maximum of 24″, twisted pair to a maximum of 36″, unterminated coax to a maximum of 60″, and terminated coax to a maximum of 100 feet. The whole back-panel construction and the application of single wire, twisted pair, or coax are calculated by a computer program to minimize the noise on each circuit node.

The two gates of a frame are a sliding pair with the power supply mounted on the sliding portion. All connecting wires between frames are coax and arrayed in layers which are formed into a drape.

References

BlaaG59; BrooF57*a*, 59; BuchW58; DunwS56; RobeJ58; BlosR60; BuchW57, 62; BrooF60; CockJ59; CoddE59, 62.