sion, it is necessary that the starting point counter of the *00* accumulator remain unchanged by the monitor. This requires that the monitor have no "shorten," "lengthen," "round," "multiply," or "divide" instructions. One further restriction on the writer: The "sign" and "overflow" check indicators may never be turned on by the monitor, for that would interfere with the use of those indicators by the code being tested.

The instructions needed to produce the desired output and still satisfy the foregoing rules are, of course, more numerous than those needed if complete freedom in the use of the *705* had been possible. Between 4,000 and 5,000 memory locations are used by the monitor to store its 675 instructions and 400 characters of constants, and to provide 650 positions of temporary storage. The constants include a table of 192 characters which relates the operation codes to their corresponding mnemonic symbols. The large amount of temporary storage is necessary for concurrently saving the contents of the accumulator, determining the length of an operand or result, and compiling the output record. The monitor code is a relative code and may be assembled to operate in any part of the memory, provided the 192 table character location addresses all have the same thousands digit.

Table I lists each operation, the type of information that will appear in the history, and the approximate central processing unit time required to collect that information and write a record.

---

# The Logical Design of a Digital Computer for a Large-Scale Real-Time Application

M. M. ASTRAHAN B. HOUSMAN J. F. JACOBS
R. P. MAYER W. H. THOMAS

THE Lincoln Laboratory and International Business Machines Corporation (IBM) have, over the past 3 years, worked out the design for a new digital computer which is the central component of a large-scale real-time system. In this system, data from a large number of sources are fed automatically into the computer where they are processed under programmed control. A complete compilation of the real-time situation is compiled by the computer and presented to operators by means of a special display system. The computer automatically generates control commands for the external environment in response to corrections and command information fed into it by the operators.

The purpose of this paper is to describe the performance criteria of the computer in general terms, and to present some of the outstanding features of the design. These features are necessary in the computer because of its particular real-time application which requires a greater emphasis to be placed on reliability, speed, capacity, and flexibility than is usual in scientific or commercial applications.

M. M. ASTRAHAN, B. HOUSMAN, and W. H. THOMAS are with the International Business Machines Corporation, Poughkeepsie, N. Y.; J. F. JACOBS and R. P MAYER are with the Lincoln Laboratory of the Massachusetts Institute of Technology, Lexington, Mass.

Disregarding its special features for the moment, the computer is a large-scale general-purpose single-address parallel digital computer with a 32-bit word length. A high-speed magnetic core memory is provided which contains 270,336 bits of storage arranged in two banks of 33 planes, each plane consisting of a 64 by 64 core matrix. The 33d plane is used for parity check bits. In addition to its buffer drums for communicating with the external environment, an auxiliary drum storage system is provided. This system contains 3,244,032 bits of storage divided among eight cylinders each consisting of six fields of 33-bit words. Each field has 2,048 words distributed around the circumference of a cylinder. Five magnetic tape units of the IBM *728* design and a large cathode-ray tube display system are also provided. In order to obtain a reliability capable of providing continuous 24-hour operation, the whole machine, with the exception of some of the input-output equipment, is duplicated. This guards against over-all system catastrophies caused by sudden machine failures and allows planned partial shutdowns for maintenance purposes.

One major consideration in the design was the availablity of components, particularly in the case of the high-speed memory. The original thinking on this computer resulted in a need for a memory faster and larger than in any existing computers. This pointed toward the magnetic core memory development which had been under way for some time at the Massachusetts Institute of Technology (MIT). The logical design of the computer was centered around the engineering judgment that the minimum memory cycle time would turn out to be in the order of 6 microseconds and that the largest matrix of cores should be 64 by 64.

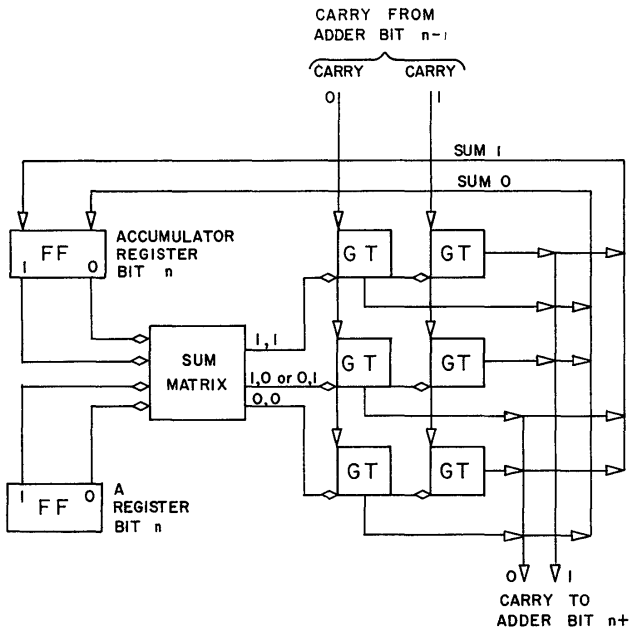
The availability of components also affected the choice of the high-capacity storage medium for which magnetic drums were chosen. The drum which seemed the most attractive and the one which was chosen was that used on the IBM *650*. The vacuum tubes and circuitry in the computer were developed from designs previously made by the Digital Computer Laboratory at MIT and the Electronic Data Processing Machine (EDPM) development laboratories at IBM. The redesign of these basic circuits was dictated by the increased reliability requirements. In many cases this necessitated the use of more vacuum tubes and other components than are required in scientific or commercial applications.

The features of the computer which are the primary subject of this paper are in the logical design areas. The majority of these features were dictated by the need to obtain the most efficient use of the drum capacity and core-memory speed with the minimum number of circuits. As a consequence, special attention was given to overlapping operations within the machine and to the balance between the available components and the logical arrangements which would maximize their usefulness.

The first of these features which will be discussed is the arithmetic element. Many of the data which are processed by the machine are in Cartesian co-ordinates. A large number of computations which are involved in the application perform the same operation on $x$ as on $y$. A saving in time is effected by the use of a dual arithmetic element which treats these quantities separately and simultaneously. The second feature is a high-speed multiplication technique which dictated the design of the adder in the arithmetic element. It was found that all of the proposed arithmetic operations, with the exception of multiplication and division, would be performed during the cycle time of high-speed memory. A special adder was developed so that the multiplication process could be made more compatible with the memory cycle.

The third feature is an indexing system which automatically takes care of the large class of bookkeeping operations (i.e., address modification) without taking extra operation time. The fourth of these features is an input-output control system which was designed to allow the computer to utilize the time which might otherwise be wasted during periods when data are being transferred between it and the input-output system. This is done by allowing the computer to continue doing work while waiting for the input-output equipment to present, or receive, its information. The fifth and final feature is a buffer drum system which matches the computer speed to the incoming data rate.

**Fig. 1. Adder circuit**




## Arithmetic Element

In the design of a digital computer, much consideration must be given to the characteristic known as "word length." For many real-time applications, much of the input data are generated by analogue devices such as thermometers, and much of the output data are sent to analogue devices such as valves or indicators. A precision of $10^3$ is often sufficient for such input and output data, with a precision of about $10^5$ being used to reduce the effects of roundoff and truncation errors during intermediate calculations. Thus a word length of from 10 to 16 bits is implied.

In a stored-program computer, consideration must also be given to the number of bits required to specify an instruction. A basic operation code of 64 instructions requires six bits. For memory sizes ranging from 2,048 to 16,384 registers, the address part of an instruction requires from 11 to 14 bits. Special control bits, such as those used for automatic indexing, add four or five more bits to the instruction word. Thus, the word length for a full instruction (including the basic operation code, control bits, and addresses) might be between 21 and 25 bits.

The conflict between the 16-bit word length for data and the 25-bit word length for instructions was resolved by the choice of a 32-bit word length. This allows two 16-bit numbers to be placed in any register and allows additional control bits to be included in instruction words. These additional control bits are used for increasing the potential speed of the machine. Since the memory must be capable of delivering the full register simultaneously for instructions, it can also deliver both numbers simultaneously. A separate arithmetic unit was provided for each of these numbers so that both could be manipulated simultaneously, thus doubling the speed of the computation. This arithmetic unit was termed the "dual arithmetic" unit.

Special operations and control bits have been added to the basic operation code to increase the utility of the dual arithmetic unit. A conditional transfer of control, or branch, can be made to take place if a specified arithmetic unit is negative, or only if both are negative. It is possible to store the full 32-bit word, or either half independently. (The address part of an instruction is placed in the right-half word; hence a "right store" instruction can also be used for modifying addresses.) Shift and cycle instructions are provided for shifting either arithmetic unit itself, or both together, or for cycling data between the right and left arithmetic units. A "twin" feature, available on some instructions, allows the left-half word from memory to be sent to both arithmetic units simultaneously. Thus, a vector in the dual arithmetic unit may be multiplied by a scalar from a left-half word of memory by a single "twin and multiply" instruction.

Each instruction which can cause an overflow contains a pair of control bits which allow suppression of an overflow alarm from either or both arithmetic units. Of course it is not always possible to make efficient use of the dual arithmetic element. Single 16-bit words can be handled just as easily by using the afore-mentioned controls, with no loss in speed except for an occasional cycle to exchange left for right.

Regardless of the word length used, there is usually a sizable body of data consisting of items only a few bits long. Storage space can be saved by packing a number of such items into a register of storage. Packing and unpacking such items can be very time consuming unless special instructions are provided. In this computer, an "extract" instruction allows a given item to be obtained from a memory register without obtaining other items from the same register. A "deposit" instruction allows the item to be replaced in storage without disturbing other items in the same register.

## High-Speed Multiply

The memory unit of a general-purpose computer must be used one or more times for every instruction. In a real-time computer the memory should be made as fast as possible and should never have to wait

for other processes. In this computer, with the design goal of 6 microseconds per memory cycle, it would have been desirable to design a 6-microsecond multiplier because of the frequent use of the "multiply" instruction. On the other hand, a basic flip-flop speed of 0.5 microsecond and a word length of 16 bits allowed a reasonably simple 8-microsecond multiplier, using 0.5 microsecond for each addition and shift.

Each step of a multiplication should allow for the possibility of a carry propagated a full register length, or 16 bits. Although such a carry condition cannot occur mathematically, it can be used to simplify the design procedure because it is a reasonable approximation to mathematically possible carries. If the basic gate-tube propagation time is 0.04 microsecond, then a 16-bit carry will require 0.64 microsecond. To reach the goal of 0.5 microsecond per multiplication step, the carry is accomplished concurrently with the 0.5-microsecond flip-flop resolution time. This is done by making sure that all pulses which arrive at a given flip-flop are spaced 0.5 microsecond apart even though they are carry pulses which have arrived as much as 0.64 microsecond after the initiation of the carry.

As shown in Fig. 1, each bit position makes use of a diode matrix to form the sum of two bits: one from the augend in the accumulator, and one from the addend in the *A* register. The matrix controls two sets of gate tubes: one set to be sensed by an incoming "0 carry" pulse, and the other by a "1 carry" pulse. When a carry pulse arrives, only one gate tube passes it and indicates the sum of the two bits plus the carry. This pulse is then sent out as either a "0 carry" or a "1 carry" pulse to the next highest bit, and as either a "zero" or a "one" sum pulse to the accumulator flip-flop, depending on the value of the sum. Thus, the carry takes only the gate-tube propagation time to pass each bit. Carry pulses supplied to the right (the least significant) end at 2 megacycles will cause every bit to receive carry pulses (of either "0" or "1" value) at a 2-megacycle rate, although the pulses at each bit will be displaced in time, depending on the bit's distance from the right end of the register. This principle forms the basis of the so-called "asynchronous adder."

For multiplication, each "add" step must be accompanied by a shift-right of one place, and some steps (for a zero multiplier bit) will require a shift-right without any addition. The shift on "add" is easily accomplished by sending the "sum" pulse from the adder to the accumulator flip-flop to the right instead of to the flip-flop which generated the sum. The shift-right without addition must propagate down the accumulator at the same rate as the carry in order to avoid conflict with a previously initiated carry. This so-called "ripple shift" is accomplished simply by letting the shift pulse from one bit initiate the shift command for the next bit.

Special controls could have been included to allow the sum to be displaced to the left instead of to the right during a divide process, and to be undisplaced during a plain addition. However, the expense of such controls was not justified. A corrective shift-left for "add" instructions (including a second corrective shift-left after an end-around carry) can be accomplished without delaying the memory cycle. Unlike "multiply," which is used extensively in most programs, the extra time required for "divide" does not appreciably increase the average instruction execution time because of the infrequent use of the "divide" instruction in most programs.

## Indexing System

Most digital computer programs involve sequences of instructions which are repeated many times on different data. It is very wasteful of memory space to include separately stored instructions to process each piece of data since the instructions will vary only in the address to which they refer. On the other hand, time is required to modify instruction addresses for each execution. An indexing feature has been included in this machine to minimize this time.

The indexing feature consists of a set of four index registers, an index adder, and associated control circuitry. Each instruction which refers to a memory address may have control bits inserted into it to specify an index register. The contents of the specified index register are added to the address part of the instruction prior to its execution; however, the instruction as stored in memory remains unchanged. This indexing addition does not require any extra time since it is executed while waiting for the memory cycle to be completed. Thus, if an "add" instruction with an address part containing 1,000 is executed, and if its control bits specify an index register which contains 24, the instruction will be executed as if it had an address part of 1,024 but will remain stored in memory as "add 1,000."

In order to utilize an index register in a cyclic program consisting of a "loop" of instructions, it must be possible to set up the index register to an initial value. Two instructions, "reset index register" and "reset index register from right accumulator," have been provided in this computer. They load the specified index register with the address part of the "reset index register" instruction or with the contents of the right accumulator.

There are three other functions that the indexing system must provide: (1) the modification of the index register each time the loop is executed, (2) the testing to determine if the loop has been executed the desired number of times, and (3) the branching of control back to the beginning of the loop unless it has been executed the desired number of times. All of these functions are handled by one powerful instruction, "branch and index." Control bits included in the "branch and index" instruction specify an index register and a decrement. Each time it is executed, the specified index register is first inspected to determine if it contains a negative number. If the number is positive, the contents of the register are reduced by the specified decrement and by the instruction branches to the address specified in its address part, usually to the beginning of the loop. Thus, execution of the loop will continue until the index register contents have been reduced to a negative number, at which time the branch is not executed and the program continues in sequence.

In addition to the four index registers, the right accumulator may also be used as an index register. This feature was included to facilitate table look-up programs. Assume that a program has been executed so that the table argument has been computed in the right accumulator. If the next instruction executed is a "clear and add" instruction which specifies the right accumulator as an index register and has the first address of the table as its address, the first address and the argument will be added before the instruction is executed. Thus, with only one instruction, the desired table contents are obtained.

When an indexed loop is used in searching for a desired value in a table, it is often necessary to determine the contents of the index register when the desired value is found. An instruction called "add index" has been provided to permit this important operation.

## Input-Output Control

The system application requires that large quantites of data be entered into the core memory, processed, and delivered

out again. The data sources and destinations have varying and unpredictable word rates and access times. Since computing time is at a premium, the input-output control design goal was that an arbitrarily sized block of words be transferred with a minimum of time devoted to the transfer. This minimum consists of the time needed to execute the program steps which set up the transfer, plus the one memory cycle required to transfer each word into or out of core memory.

Reading into core memory will be assumed in the following: Writing out of memory is analogous. The design goal required the use of an independently operating input-output control which could remember the input-output unit selected, the number of words to be transferred, and the location in core memory for the block of words. A "break" system was needed which could interrupt the program operation for one memory cycle whenever the input-output unit had provided a word. The instructions in the computer are designed in such a way that a "break" memory cycle can be initiated at the end of any memory cycle with no effect on the instructions other than to delay their execution.

The system chosen operates as follows: Special instructions in the computer connect the proper input-output unit to the information transfer paths, load counters which keep a record of the location in memory in which the data are to be stored and keep a count of the words transferred, and start the flow of data. The program operation then continues normally, except for interruptions of one memory cycle per word caused by the break system. Each time a word is transferred, the counters are stepped accordingly in preparation for the next word.

The operation is terminated when the counters signal that the requested number of words has been received. It can also be terminated earlier by a disconnect signal from the input-output unit. This occurs when the program has requested more words than the input-output unit has to send and when the input-output unit has run out of words. Facilities have been included to allow the computer to examine the counters in order to determine how many of the words requested were transferred.

Interlocks hold up the program if an input-output operation is called for before the preceding one is finished. Therefore, to use the system efficiently, the programmer must provide enough work for the computer in order to consume all the time taken by the input-output operation before another input-output operation is started. To help accomplish this function, a "conditional branch" instruction is provided which can detect whether an input-output operation is still in process. This instruction also provides the programmer with a means of determining whether an input transfer has been completed before attempting to use the data.

## Drum Buffer System

The real-time application for which this computer was designed required that the computer receive its input data from many independent, asynchronous sources. The exact quantity of data to be received from any one source could not be determined; however, it was possible to estimate the average and maximum amounts with a reasonable degree of accuracy. The application is such that the total amount of input data received from all the sources combined is less than the sum of the individual maximum amounts.

Another characteristic of the data sources is that they operate at a much lower speed than that of the computer. It is not operationally feasible to interrupt the computer operations to accept each piece of data as it arrives. A buffering mechanism is necessary to gather the data at the slow incoming rate and then pass on large blocks of the data to the computer at the computer's speed. Magnetic drums were chosen for this buffer.

The desired characteristics of the drum buffer system follow: The input data should be written on the drum as soon as possible after they are received and definitely before another piece of data is received from the same source. The writing of input data on the drum should not interfere with the reading of data by the computer. The computer should be able to read selectively from the drum; that is, to read data from only one source at a time. In order to minimize the number of drum storage registers required, the buffering system must be able to combine the data received from many sources.

To provide the "no inference" characteristic, a dual-access drum is used. This drum has two sets of drum heads. One set, called the outside of drum system (OD), is used to communicate only with the outside world. The other set, called the computer-side of drum system (CD), communicates only with the computer.

To provide the "minimum storage" and "write as soon as possible" characteristics, the "random storage" drum was developed. Each drum register has a status bit associated with it. This bit is used to indicate the "full" or "empty" status of the register. As a register passes under the OD drum heads, the status of the register is sensed. If the register is empty, a "drum demand" pulse is generated and sent to interrogate the data input equipment. If there are data from one of the sources, a "data available" pulse is sent back to the drum status circuits and the data are sent to the drum write register. As the information is written on the drum, the status marker is changed to indicate that the register is now full.

A single drum channel is not used for the status indication. There is not enough time to read the status bit, decide whether it indicates "full" or "empty," generate the "drum demand" pulse, interrogate the data sources, and write the full indication with the same drum head. If a second head were used on the same drum channel, for physical reasons, it would have to be located many registers away, thus necessitating the use of a shift register or delay-line device between the two heads. Therefore, two drum channels are used for the status indication, and the indicator bit is shifted back and forth between them. The two channels are called the OD and CD status channel since they are read by the OD and CD side of the drum, respectively.

Consider first the OD side of the drum system as shown in Fig. 2. When a piece of data is received by the input equipment associated with a data source, a "data received" pulse is generated and is used to set the corresponding flip-flop in the drum demand chain to the ONE condition.

As each drum register approaches the drum write heads, the status bit on the OD status channel associated with the register is read. The status read head is located slightly before the write heads. If the status bit is a ONE, indicating that the register is full, the only action that takes place is the writing of a ONE in the CD status channel by the status write head. If the status bit is a ZERO, indicating an empty register, a "drum demand" pulse is generated and sent to the drum demand chain. This pulse interrogates each of the flip-flops in order until it finds one which has been set to the ONE state or until it reaches the end of the chain. If the pulse encounters a flip-flop in the ONE state, it becomes a "data available" pulse and (1) resets the flip-flop, (2) causes the data from the source associated with the flip-flop to be transferred to the drum writing circuits, and (3) causes the status circuit to write a ONE in the CD status channel. If there are no data available, the ZERO indication is passed on to the CD status channel.

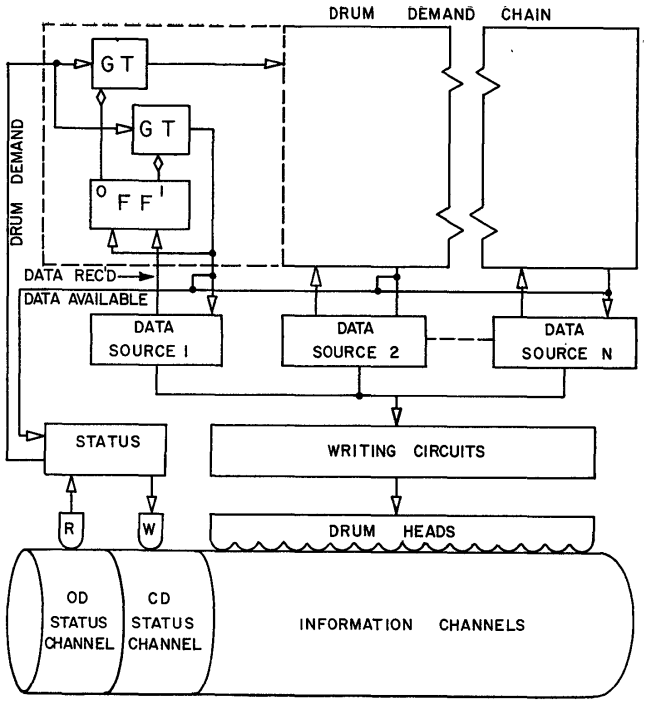



Fig. 2. OD side, drum system

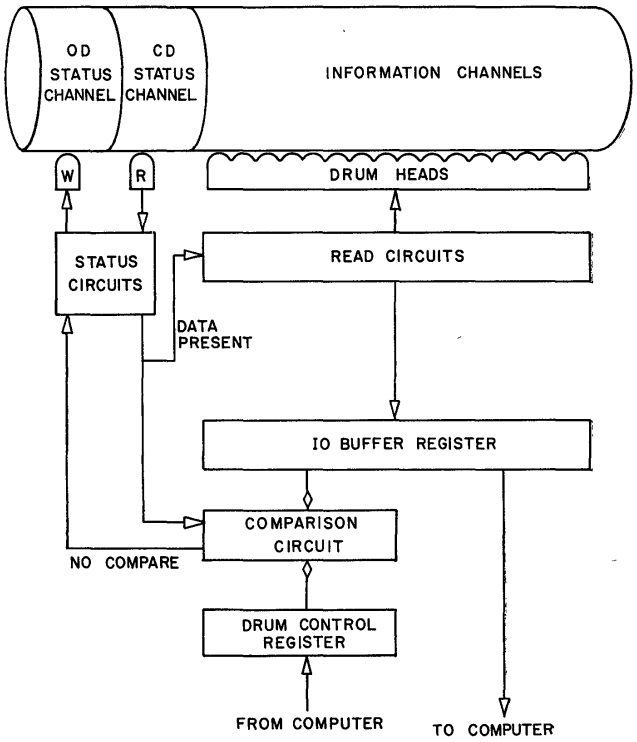



Fig. 3. CD side, drum system

Each piece of data written on the drum has some identity bits associated with it to identify the source of data.

There is a possibility that as the drum fills up, some of the data received will not be stored on the drum before the next piece of data is received from the same source. This is particularly true of the sources at the tail end of the demand chain. The probability of storage on the drum depends on the rate of input data and on the number of empty registers on the drum. Most of the data rates are such that if the drum is kept at least 50 per cent empty by the computer, the probability of storage is better than 0.99. The data received from this type of source are such that the occasional loss of a piece of data does not seriously affect the system. Fortunately, the rate from the critical sources is slow enough to allow a complete search of the drum for an empty register before a second message can be received from the same source.

Now consider the CD side of the drum as shown in Fig. 3. The status heads and circuits on the CD side are almost identical to those on the OD side. During periods when the computer is not reading the drum, the status information is automatically and continuously transferred from the CD status channel to the OD status channel.

The computer may read the drum in either of two modes, status or identity. When reading in the status mode, the contents of every full register passing under the heads is transferred to the core memory through the input-output buffer register and a ZERO is written in its associated status bit in the OD status channel, indicating that the register is now empty. When reading in the identity mode, the computer first places the identity code desired into the drum control register. Only those words with matching identity are transferred to the core-memory.

The transfer of words continues until a disconnect pulse is received from the computer, indicating that it has received the number of words it has asked for, or until the drum itself generates a disconnect pulse. With an unknown quantity of data received from each source, the computer may ask for more data than has been received. In fact, in order to insure

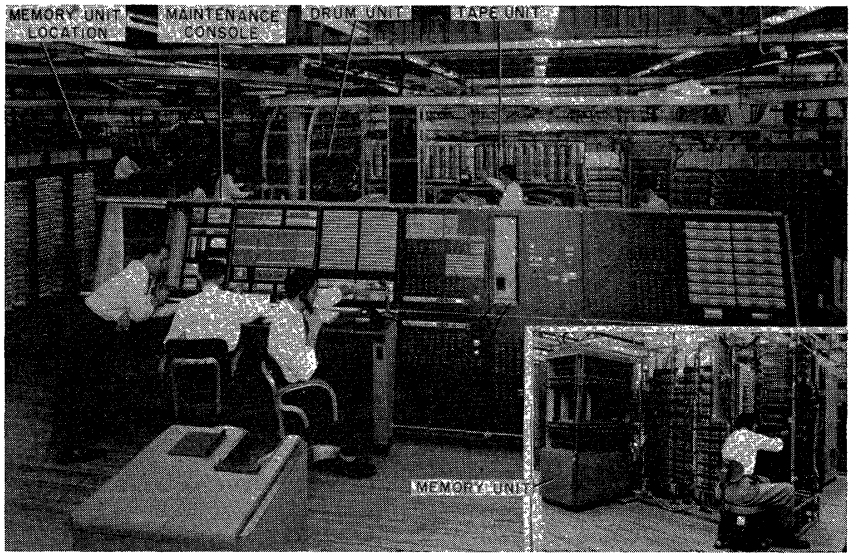



Fig. 4. Portions of computer in test cell

that the computer receives all the data that have collected on a drum field between readings, the normal procedure is to ask for more than expected. A disconnect pulse is automatically generated at the end of a complete drum revolution, and the number of words read is determined from the word counter contents.

Some of the data sources have messages which contain more information than will fit into one drum register. To handle such information, another feature has been added to the drums associated with these sources: the drum has been divided into multiple word slots of adjacent registers. For this application the only meaningful status bit is the one associated with the first register of a slot, and the source identity is contained in the first register of a slot. The operations associated with those drums are almost identical to those of the single register drums.

In addition to the input buffer drums just described, there are output buffer drums which handle outgoing data. They operate in a similar, though inverse, manner to the input drums.

## Conclusion

Fig. 4 illustrates a portion of the computer in the test cell. Because of its size and layout, it was not possible to obtain a picture of the whole computer. The unit in the foreground is the operator's maintenance console. It contains switches for manual data or instruction entry and manual control, neon indicator lights for the major flip-flops and registers in the computer, visual and audible indicators for computer generated alarms, marginal checking controls and indicators, and power system and air-conditioning indicators. The cutout contains a view of one of the memory units.

The computer, including the directly connected input-output equipment, contains approximately 12,500 tubes. It has an execution time of 12 microseconds for arithmetic instructions, excluding "multiply" and "divide," which require 15.5 and 53 microseconds, respectively. The prototype model has been in satisfactory operation for more than one year.

# Computer Design to Facilitate Linear Programming

R. C. GUNDERSON

AT the risk of being redundant, this paper will begin by stressing the growing importance of linear programming in business, industry, and government, as it is this importance which is its motivation.

Primarily, it is the application of linear programming with which the paper will deal. Optimal planning of procedures has become a necessity, rather than a luxury, to present-day management. For example, one organization is presently saving an estimated $20,000 a day by optimal planning through linear programming procedures. Without too great a stretch of the imagination, the fascinating possibilities of linear programming linked with automation might be pictured. This could yield factories staffed with skilled technicians to feed data from changing markets into computers, which would then choose the optimal combinations of specifications and direct the machinery to produce under these new specifications. There are countless other such possibilities which could make effective use of this powerful tool.

R C. GUNDERSON is with Remington Rand Univac, St. Paul, Minn.

It is not intimated here that the use of linear approximations is a new addition to mathematics or economics. Rather, it is the wider acceptance of their usefulness which is new. This, coupled with the fact that much work has been done in the past decade to develop a general formulation of the computational procedures involved in linear programming, presents an exciting facet of computer application to users and manufacturers.

It is evident, then, that some consideration should be given to the requirements of this problem in the building of our future computers. Essentially, the actual needs are for the most part familiar to the computer industry. Moreover, the logical properties of computers which this problem requires are extremely compatible with those desired by logicians.

Let some of the qualities of this problem which make it especially well adapted to high-speed digital computation be examined for a moment. First, since input and output time still lags behind computation time on all present-day large-scale computers, the relatively small amount of input, simple but voluminous computations and logical operations, and the small amount of output required, lend themselves well to computers. Second, the iterative nature of the matrix manipulations is ideally suited to stored program computation. Finally, the ability to generalize the procedure, enabling the solution of a number of maximization or minimization problems containing dissimilar data, reduces the programming involved to mere data preparation.

In the following, some of the essential physical properties of a computer which make handling linear programming problems more efficient will be discussed. Operating on matrices by rows or columns necessitates much greater rapid access storage than an element by element operation would require. Present-day problems, which undoubtedly will soon be dwarfed, require a minimum of 4,000 words, and would run much more efficiently with 8,000 to 12,000 words of rapid access storage. The so-called "housekeeping" operations required by the limitations of present storage systems increase running time by approximately one fifth.

The Simplex method of solution, probably the most efficient and most used linear programming procedure, requires access to the stored matrix of coefficients at random, as dictated by the computation. Moreover, the generation of an additional vector during each major iteration necessitates a large-capacity secondary storage in the more sophisticated problems. There is, then the additional requirement of a large, random access, secondary storage media, probably 15,000 to 30,000 words in size,