The System Design of the IBM Type 701 Computer*

WERNER BUCHHOLZ†, MEMBER, IRE

Summary—**In designing any new piece of equipment a choice has to be made from a number of alternatives. Rather than just enumerating the features incorporated in the IBM Type 701 Computer, an attempt is made to record the reasons for their choice. Emphasis is given to the features which are believed to be new. These include improved arithmetic and logical facilities, as well as the methods developed for controlling the extensive input and output equipment directly from the stored program.**

Introduction

A LARGE, GENERAL-purpose, electronic digital computer is probably the most complex equipment made which has to function as one coordinated, centrally controlled machine. Not only the equipment, but also the logic behind it is complex. For such a complex system to be successful, it is important that the proposed design first be thoroughly evaluated while it is only on paper and construction has not been started. A realistic appraisal of the system design can be obtained at that stage by writing actual programs for proposed applications. Weaknesses thus discovered may be removed and any unnecessary features eliminated. It is entirely uneconomical to make these improvements by building, testing, and scrapping several experimental models of such a large machine, and it has been found that small pilot models are not very useful because they do not exhibit many of the important characteristics of the full-scale machine. Computers with a stored program lend themselves readily to experimentation on paper since the program as written on paper completely determines the action of the computer. Hence this method of approach was used in arriving at the best system for the IBM Electronic Data Processing Machines Type 701 and Associated Equipment, the complete name given to the large-scale, high-speed electronic computer installations produced by the International Business Machines Corporation.

The development of the basic logic of a system as large as the Type 701 computer becomes a specialized effort which is distinct from, but must be closely integrated with, the design of the electronic and electromechanical equipment.[1–4] Engineers familiar with the

* Decimal classification: 621.375.2. Original manuscript received by the Institute, June 25, 1953.

† International Business Machines Corp., Engineering Laboratory, Poughkeepsie, N. Y

[1] L. D. Stevens, "Engineering organization of input and output for the IBM 701 electronic data-processing machine," *Proc. Joint AIEE-IRE-ACM Computer Conference;* March, 1953.

[2] J. Logue, A. Brennemann, and A. Koelsch, "Engineering experience in the design and operation of a large-scale electrostatic memory," *IRE Convention Record;* March, 1953.

[3] C. E. Frizzell, "Engineering description of the IBM type 701 computer," PROC. I.R.E., pp. 1275–1287; this issue.

[4] H. D. Ross, Jr., "The arithmetic element of the IBM type 701 computer," PROC. I.R.E., pp. 1287–1294; this issue.

components and mathematicians having experience in using computers, both contributed at an early stage to the evaluation of the proposed system. There were many alternatives to be considered, and an attempt is made in this paper to set down the reasons for choosing the features incorporated in the final design. Many of these choices were interrelated, so that designing the system took on some of the aspects of solving a giant jigsaw puzzle. The complex nature of the puzzle may be obscured by the effort made to keep the logic of the computer simple. Rather than single out those features which are believed to be new, it is intended here to present them in the context of the over-all organization of the 701 system.

The 701 is a parallel binary computer designed primarily to solve large problems in scientific and engineering computation. It is the first such system to be manufactured in quantity. The computer is controlled by a stored program of single-address instructions. It has a large internal high-speed memory, large even by present-day standards as set by its immediate predecessors. This storage is further supplemented by the lower-speed, but higher-capacity, magnetic drum storage and by magnetic tape. There is a complete set of input-output components in the system, including punched cards and a line-at-a-time printer, as well as the magnetic tapes, all of which are under the direct control of the computer program. The number of such input-output devices can be varied to suit the requirements of individual installations.

The arithmetic speed of the 701 compares well with that of other fast computers. In the 701, however, high speed is combined with high storage capacity and fast input and output. On large problems, therefore, where the internal capacity must be supplemented by external devices, the 701 can produce results faster than any other computer in existence. The 701 was specifically designed to tackle problems which extend beyond the range of existing computers. But the versatility of the input-output also permits the 701 to be used efficiently on small problems when they occur in large numbers in an organization with central computing facilities.

Providing such extensive input-output facilities necessitated the development of effective techniques for their control which are considered to be new. The internal logic of the computer has also been refined, as compared with computers preceding the 701, and a number of features have been incorporated which greatly simplify its programming.

General Design Philosophy

Throughout the design of the system the philosophy has been to keep the equipment to a minimum, to make that equipment fit a simple logical pattern, and to avoid special-purpose devices. An effort has been made to keep the instructions as simple as possible and to avoid obscure restrictions and overlapping between the functions of different instructions. While the equipment was being designed there existed a strict regime of discarding any frill which would not be of benefit in more than one type of application.

Avoiding special-purpose equipment not only decreased the cost but also increases the reliability. For the same reason, standard, well-tried components are used as much as possible, especially for input and output. By providing high-speed card readers and punches, existing punched-card equipment becomes available as an auxiliary to the computing installation. Standard key-punches, for instance, can be used to punch and print on cards for the purpose of introducing new programs and original data in a convenient, accurate, and readable form.

The Internal System

Number System

The internal operation of the 701 is entirely in the binary system, but the input and output equipment will handle decimal and alphabetic, as well as binary, data. Thus the input-output advantages of a decimal machine are combined with the simplicity and low cost inherent in a binary computer. The simplicity of binary operation is particularly evident in multiplication and division. A further advantage of the binary system is that any other code system can readily be re-coded in terms of binary numbers; for instance, punched cards can be read and analyzed in the 701 regardless of whether they are punched in decimal, alphabetic, or any other code including any form of control punching.[5] A decimal-binary translation is, of course, required at the input and output, but the translation program can overlap with the operation of the card reader, punch, or printer, so that the time taken for translation does not normally slow down the input and output.

In order to achieve the high internal speed needed for scientific computing, the bits (binary digits) of a number or word are handled in parallel. "Word" is the term usually used to include numbers as well as instructions and other non-numeric information; the maximum number of bits in a word, including the sign bit if any, is called the word length. The proper choice of word length, and hence the number of parallel channels required in the computer, presents a question which cannot be answered lightly. If the word length is too short there will be many applications for which a single word is not enough to carry all the bits needed to represent a number. Because of the accumulation of unavoidable rounding and truncation errors, it may become necessary to carry more bits during a calculation than are retained for the answers. Hence too short a word length leads to the necessity of using two or more words for each number, and simple additions or multiplications must be replaced by very much more complicated,

[5] M. M. Astrahan and N. Rochester, "The logical organization of the new IBM scientific calculator," *Proc. Assn. for Computing Machinery;* May, 1952.

space- and time-consuming, "multiple-precision" operations. As a result much of the high speed and storage capacity of an otherwise good computer is nullified.

Too great a word length is also bad. The time for carrying and shifting increases, and thus the speed goes down. The cost of the arithmetic circuits and registers, and the cost for a given number of words of storage go up. Only a few applications will need the full word length; hence there will be a strong tendency to pack several numbers into one word so that the cost may be reduced by keeping the amount of storage to a minimum. Unpacking these numbers requires instructions and time, and the effective speed and storage capacity would be cut still further.

Thus with a given set of applications there will be an optimum word length for which the speed is at a maximum or the cost is at a minimum. The relation between the variables is, of course, far from simple, and a compromise is needed in any case. For the 701 a survey of proposed applications was made, and it was found that the range of word lengths between 10 and 12 decimal digits would be satisfactory. There exists a broad peak of efficiency in this range. Below 10 digits there are too many problems requiring double-precision arithmetic, and there are not enough problems which could make effective use of more than 12 digits. Numbers of 10 or 12 decimal digits with sign can be represented in binary form by a minimum of 35 or 41 bits, respectively, including sign. Hence the survey showed that the word length of the 701 should be in the range of 35 to 41 bits.

Several other factors then quickly narrowed down the choice of word length. The single-address type of instruction, with the number of operations and the memory size being considered, required a minimum length of 18 bits, so that for the most efficient packing of instructions in memory the word length should be a multiple of 18. For engineering reasons the magnetic tape was to be provided with up to six parallel information channels; hence words could be recorded on tape most efficiently if the word length was to be a multiple of 6. The desire to reduce the bulk and the cost of the computer further prompted a choice near the bottom of the range of 35 to 41 bits. The final choice of 36 bits fills all these specifications.

Types of Storage

The 701 has three types of high-speed storage: electrostatic, magnetic drum, and magnetic tape. (Cards are also a form of storage, but they do not share the high speed and the possibility of rereading, or erasing and rewriting, previously recorded information without manual intervention.) The three storage media complement each other in their properties. Electrostatic storage is the fastest and has the shortest time of access to form of storage when considering the volume and cost per bit stored. Magnetic tape, though slower, is quite a fast medium when information can be organized on tape in the order in which it is to be used. But the time to gain access to any one item stored on a tape reel at *random* is counted in minutes instead of microseconds. On the other hand, the volume and the cost of tape storage are extremely low. Tape and electrostatic storage together thus take care of the two extreme requirements of high storage capacity and low access time. Drums were provided as a third medium to bridge the rather wide gap between the other two. Drums are useful for auxiliary programs or tabular data which may be needed in random order but not so often that a delay of a few tens of milliseconds would seriously affect the over-all computing time.

The electrostatic type of working memory was chosen because it makes practical a combination of high operating speed and adequate storage capacity. Although it shares these properties with memory devices of the recirculating type which have been used in many computers, electrostatic memory has yet a further advantage. Its short access time permits instructions to be stored at any location and in any order without affecting speed or performance. Thus there is no need for elaborate efforts to attain speed and efficiency by optimum programming. The random-access property also makes it easier to operate input, output, and external storage devices out of synchronism with the central computer. Memory can receive or supply information on only a few microseconds' notice; yet it can wait indefinitely if an external unit is not ready. Hence just a minimum of buffer storage is needed to tie the various units together. Most of the external units use a single one-word buffer register, namely, the multiplier-quotient register which is already there for arithmetic purposes. Tape requires an extra 6-bit register, the only special-purpose buffer storage in the machine.

Electrostatic memory, as used in the 701,[2,3] requires regeneration at regular intervals since it is a volatile type of storage. It needs adequate regeneration just as it needs an adequate supply of power in order to retain information. This could be done by interrupting operations and regenerating all of memory in one lump, whenever necessary. Instead it was decided to regenerate memory whenever it is not used for other purposes and to satisfy the need for further regeneration by attaching a number of extra regeneration cycles to each instruction, the number depending on how hard that instruction uses memory. In this way it becomes possible (see Table I, right) to determine exactly how much time each instruction takes, a necessary step when one wants to save time by overlapping internal and external operations, as will be discussed later on. Memory was designed according to the specification that regeneration should be adequate to avoid any restrictions on how

TABLE I

LIST OF 701 INSTRUCTIONS WITH BRIEF EXPLANATIONS

Code	Short Name	Time[6]	Full Name and Explanation[7]
00	*Stop*	—	*Stop and Transfer* Stop, and prepare to transfer to x (see *Tr*) when the computer is started again.
01	*Tr*	48 (24)	*Transfer* Take the next instruction from half-word address x.
02	*Tr Ov*	48 (24)	*Transfer on Overflow* Transfer to x only if the overflow indicator is on; then reset the overflow indicator.
03	*Tr+*	48 (24)	*Transfer on Plus* Transfer to x only if the accumulator sign is plus; the other accumulator positions are ignored.
04	*Tr 0*	48 (24)	*Transfer on Zero* Transfer to x only if the accumulator contains zero; the accumulator sign is ignored.
05	*Sub*	60 (36)	*Subtract* Subtract the contents of x from the accumulator.
06	*R Sub*	60 (36)	*Reset and Subtract* Reset the accumulator to zero before subtracting.
07	*Sub Ab*	60 (36)	*Subtract Absolute Value* Subtract the absolute value of the contents of x from the accumulator.
08	*No Op*	48 (24)	*No Operation* Do nothing.
09	*Add*	60 (36)	*Add* Add the contents of x to the accumulator.
10	*R Add*	60 (36)	*Reset and Add* Reset the accumulator to zero before adding.
11	*Add Ab*	60 (36)	*Add Absolute Value* Add the absolute value of the contents of x to the accumulator.
12	*Store*	60 (24)	*Store* Store the accumulator contents (except the two overflow positions) at x, replacing the previous contents; the accumulator is left unchanged.
+13	*Store A*	60 (24)	*Store Address* Store the contents of bit positions 6 through 17 of the accumulator in place of the rightmost 12 bits at half-word address x. Note: Instruction must have + sign part.
−13	*Extr*	60 (24)	*Extract* Wherever an accumulator bit position contains a zero (or +), store a zero (or +) in the corresponding position at memory address x; leave all other bits in memory unchanged. Note: Instruction must have − sign part.
14	*Store MQ*	60 (24)	*Store Number from MQ Register* Store the contents of the *MQ* register at x.
15	*Load MQ*	60 (24)	*Load MQ Register* Load the contents of x into the *MQ* register.
16	*Mpy*	456	*Multiply* Multiply the contents of x by the contents of the *MQ* register. The most significant 35 bits of product are left in accumulator and the others in *MQ* register, in place of their previous contents.
17	*Mpy R*	456	*Multiply and Round* The same as *Mpy* followed by *Round*, giving a rounded 35-bit product.
18	*Div*	456	*Divide* Divide contents of the accumulator and *MQ* register, taken as a unit, by the contents of x. Contents of x must exceed in magnitude the accumulator contents. The resultant quotient appears in the *MQ* register, and the remainder is left in the accumulator.
19	*Round*	48 (24)	*Round* Increase the magnitude of the accumulator contents by one in position 35, if there is a one in position 1 of the *MQ* register. The *MQ* register is not changed.
20	*L Left*	48 (24)[8]	*Long Left Shift* Shift both the accumulator and *MQ* register contents as a unit to the left by x places (x not greater than 255). The accumulator sign is set to that of the *MQ* register.
21	*L Right*	48 (24)[8]	*Long Right Shift* Shift both accumulator and *MQ* register contents as a unit to right by x places. *MQ* register sign is set to that of the accumulator.
22	*A Left*	48 (24)[8]	*Accumulator Left Shift* Shift accumulator contents to left by x places. Sign is not changed.
23	*A Right*	48 (24)[8]	*Accumulator Right Shift* Shift the accumulator contents to the right by x places. The sign is not changed.
24	*Read*	48 (24)[9]	*Prepare to Read* Prepare to read one unit record from the input component specified by x. Start mechanical movement forward if necessary.
25	*Read B*	48 (24)[9]	*Prepare to Read Backward* (For tape only.) Same as *Read* except tape is moved backward.
26	*Write*	48 (24)	*Prepare to Write* Prepare to write one unit record on the output component specified by x. Start mechanical movement forward, if necessary.
27	*Write EF*	48 (24)[9]	*Write End of File* (For tape only.) Write an end-of-file gap on the tape unit specified by x.
28	*Rewind*	48 (24)	*Rewind Tape* (For tape only.) Rewind tape unit specified by x to starting point.
29	*Set Dr*	48 (24)	*Set Drum Address* (For drum only.) Set up x as the drum address of the first of a sequence of words to be read or written on the drum specified by the last preceding *Read* or *Write* instruction.
30	*Sense*	48 (24)	*Sense and Skip, or Control* If x specifies an input, sense input line for a signal; if a signal is present, skip the next instruction. If x specifies an output, send out a control signal; do not skip.
31	*Copy*	60 (24)[9]	*Copy and Skip* 1. If reading, store one word arriving from the input at memory address x. At the end of a unit record, skip *two* instructions. At the end of a file (i.e. after the last record is read), skip *one* instruction. Otherwise do not skip. 2. If writing, send one word from memory, address x to the output; do not skip.

[6] The normal time is given first. The time in parentheses applies only if this is one of 12 instructions immediately following *Mpy*, *Mpy R*, or *Div*.

[7] x denotes the address part of an instruction.

[8] These are minimum times. Time to shift $=12(1+K)$ microseconds where K is smallest integer $\geq x/8$, provided this exceeds the minimum of 48 (24) microseconds.

[9] These times may increase if input-output synchronization requires a delay.

While the power is on, regeneration is the normal state of rest.

Storage Capacity

A high-speed, stored-program computer cannot be used efficiently unless the number of words available in the working memory is large enough. The size of the 701 memory was determined from a study of several proposed applications. The memory had to be able to accommodate the data and instructions for those sections of a problem which consume most of the calculating time.

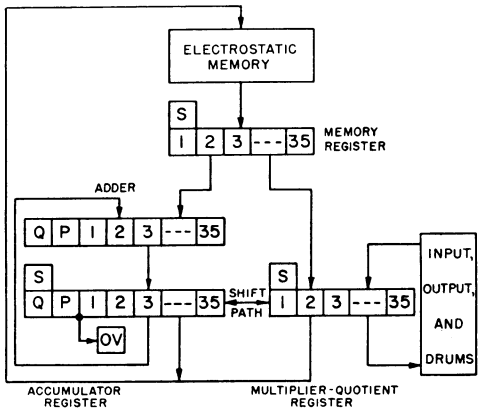
Memory sizes of 2^{10}, 2^{11}, and 2^{12} words were considered, these numbers being convenient choices for binary addressing systems. On the basis of experience obtained prior to 1951, the time of the application study, it was concluded that 2^{10} words would not be enough, but that 2^{11} or 2048 words of 36 bits would be an adequate memory size for the sample problems. One reason for this rather large size, by present standards, was the policy of replacing control circuits by subprograms, as will be discussed in later sections. Sufficient memory space had to be allowed for these subprograms, but the memory assignment is quite arbitrary and the extra memory adds considerable flexibility at relatively low cost when compared with a design of less memory and more special-purpose control circuits.

The 1951 study also indicated, however, that there would be little advantage in making the working memory much larger than indicated above. Once a certain size is reached, additional high-speed memory does not contribute much. Big problems which do not fit can then be split efficiently into smaller sections which do fit. The sections not in current use are stored on drum or tape, depending on the frequency with which they must be called into electrostatic memory. Drums and tape are successively larger storage reservoirs which back up the working memory; tape has the further property that it is an output medium which can be removed for safekeeping and used as an input at any later time to reintroduce data or instructions into the computer. Because of the long initial access times of tape and drum storage, information should be moved into and out of memory in large blocks. No attempt has been made to provide convenient access to individual words of data or instructions in these auxiliary storages. Before instructions stored on tape or drums can be executed, an entire program or block of instructions must first be placed into electrostatic memory. The high speed inherent in the 701 would not be effective otherwise.

Because of the choice of 2048 words as the electrostatic memory capacity, drum storage has also been split into logical blocks of 2048 words. Thus, if desired, the entire memory could be unloaded into one block of drum storage. For structural reasons, four such logical blocks were placed into one physical package of drum storage. Hence drum storage comes in increments of 8192 words. Tape provides an even larger jump in capacity. About 115 blocks of the size of memory can be stored on one reel of tape. Tape drives are built in pairs, again for engineering reasons. Four tape drives are considered a standard complement. Four are sufficient for many applications, and this number of tapes also permits sorting of data by repeated merging. The amount of tape and drum storage provided can be changed, however, depending on individual requirements.

Arithmetic and Program Control

The arithmetic elements of the 701 are described elsewhere,[4] and the methods used internally for adding, subtracting, multiplying, and storing the results will



LEGEND:

1,2,....,35	NUMERIC BIT POSITIONS
S	SIGN POSITION
P,Q	OVERFLOW POSITIONS
OV	OVERFLOW INDICATOR

Fig. 1—Block diagram of information paths.

not be discussed here. The external characteristics of these operations are apparent from the brief list of instructions in Table I. For ease of reference, Fig. 1 shows the information paths of the 701 in schematic form. The arithmetic system is noteworthy for its simplicity. No more than three registers are needed: the memory register, the accumulator register, and the multiplier-quotient register (*MQ* register). Of these three, the memory register serves purely internal functions, such as holding the multiplicand during successive cycles of multiplication so as to avoid repeated access to memory. The accumulator and *MQ* registers are the only ones which need to concern the programmer since they participate actively in arithmetic and other operations. The *MQ* register also doubles as a buffer register for external units, as has been mentioned. The *MQ* register thus is the only link between the internal information paths and the outside world.

The block diagram of Fig. 2 shows the registers required to execute a program stored in memory. The *instruction counter* keeps track of the location of the next instruction to be executed. Normally it advances by one step for each instruction. Instructions called

from memory first go to the memory register already mentioned. From there the various components of an instruction—the sign, the operation part, and the address part—are switched to the *sign register*, the *operation register*, and the *address register*. These registers are used to set up the memory deflection and the appropriate switching paths required for the execution of the instructions.[3]

Single-Address Instructions

The 701 employs a single-address system of instructions. Only one address in memory or one external unit may be specified with one instruction, in addition to

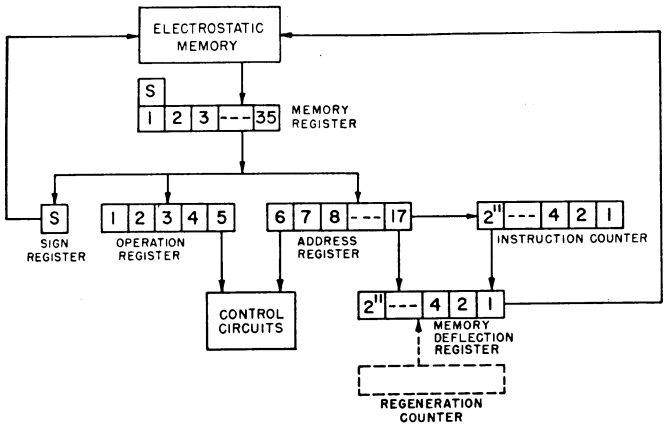


Fig. 2—Block diagram of program control paths.

the operation which is to be carried out. The single-address system is the simplest of the various systems possible, some of which may be more economical in storage space and computer time. These savings were not considered worth the extra equipment needed with multiple-address systems.

One instruction is represented by 18 bits which are made up of the sign bit, 5 bits for the operation part, and 12 bits for the address part. The 5 bits of the operation part provide for 2^5 or 32 different operations. A 33rd operation, *Extract*, was added by making use of the sign bit. The 33 operations are listed in TABLE I together with short explanations. It will be seen from this table that the abbreviated names, which are recommended for writing out a program, are not just arbitrary codes surrounded by an aura of mystery. They are derived from, and immediately suggest, the full names of the operations which, in turn, were carefully chosen to be meaningful English phrases. These abbreviations make it much easier for a person to read a program which he did not write himself. Attention to such detail has been found most valuable in introducing newcomers to the computing field.

Full-Word and Half-Word Addressing

Since one instruction is made up of 18 bits, a 36-bit word can hold two instructions. Thus an important property of the 701 is that any full word of 36 bits (or 35 numeric bits and sign) can be split into two half words of 18 bits (or 17 numeric bits and sign), each half word being separately addressable. This greatly simplifies programs which must modify instructions and doubles the memory capacity for storing data which do not require more than 18 bits, the equivalent of about 5 decimal digits and a sign. The capacity of electrostatic memory is thus 2048 full words or 4096 half words, or an intermediate number if they are mixed.

Full and half words are addressable independently in a manner which puts the assignment of specific areas in memory to either size of word entirely under the control of the programmer. Addresses are designated by 12-bit binary numbers and a sign. This allows for 2^{12} or 4096 different addresses, and each of the 4096 half words has its own address. Full words are designated by even addresses only. For those instructions which can refer to either full or half words, the sign part is used to indicate

TABLE II

SUMMARY OF 701 SYSTEM CHARACTERISTICS

General:
- Parallel operation.
- Binary notation internally.
- Control by stored program.

Word Size:
- Either 36 bits or 18 bits, including sign; approximately equivalent to 10 or 5 decimal digits and sign.
- Accumulator has two extra bits for overflow.

Instructions:
- Single-address system.
- 33 distinct operations.
- Instructions are 18-bit words.

Computing Speed:
- Multiplication or Division: 0.456 millisecond.
- Addition or Subtraction: 0.060 millisecond.

Electrostatic Storage:
- Capacity—2048 words of 36 bits each.
- Each full-word location may store a pair of 18-bit words.
- 72 tubes, 1024 bits per tube.

Magnetic Drums:
- Four drums, each with a capacity of 2048 words of 36 bits each, in basic system.
- Average access time to first word of block—40 millisceonds.
- Reading or writing rate—800 words per second.

Magnetic Tapes:
- Four magnetic tape units in basic system.
- Material: Oxide-coated plastic tape, ½ inch wide.
- Recording in 7 parallel channels, 6 for information and 1 for redundancy checking.
- Tape may be written forward, read forward, or read backward under program control.
- Density within a unit record: 200 words per foot.
- Distance between unit records—1 inch.
- Maximum tape length on reel—1400 ft.
- Access time to start of unit record—approx. 10 milliseconds.
- Reading or writing rate within unit record—1250 words per second.

Page Printer:
- Rate—150 lines per minute.
- Prints numeric, alphabetic, and special characters.
- Prints at full speed in decimal form using simultaneous conversion program.
- Prints 72 characters in any of 120 character positions on one line, more at reduced speed.
- Automatic line spacing or skipping, under control of stored program.

Card Reader and Card Punch:
- Reads or punches any 72 of the 80 card columns.
- Reading rate—150 cards per minute.
- Punching rate—100 cards per minute.
- Reads or punches cards in standard IBM decimal code at full speed using simultaneous conversion program.
- Reads or punches cards in binary form at full speed with 24 words of 36 bits each to a card.

whether the address part specifies a full word (minus) or a half word (plus). For example, a full word of 36 bits (S, 1, 2, 3, . . . 35) may have the address −1962. If the same 36 bits were used to store two half words, their addresses would be +1962 for the left 18 bits (S, 1, 2, . . . 17) and +1963 for the right 18 bits (18, 19, 20, . . . 35). Bit 18 of the full word becomes the sign bit (S) of half word +1963, bit 19 becomes bit 1, and so on.

With this system there is no doubt as to whether a given space in memory has already been assigned, because the numbers designating the addresses of full words and of the corresponding half words are alike, except for the sign and the even-odd part of the units digit. Another advantage of this addressing system is that the locations in memory of successive instructions, each of which occupies a half word, are numbered consecutively. It is also possible to transfer control directly to any instruction, regardless of which position in a full word it occupies.

The distinction between full and half words is made solely in electrostatic memory. Only full words, which may, of course, represent pairs of half words or instructions, are transmitted to and from input-output devices or stored on the drums. When a half word is entered from memory into the computer it is treated as a full word, of which the left 18 bits (including sign) are the specified half word and the right 18 bits are zeros. This is so regardless of whether the half-word address was even or odd. Similarly, on storing numbers from the accumulator or MQ register, the contents of the left half (bits S, 1, 2, . . . 17) can be stored at any half-word address in memory.

The rather arbitrary choice of moving the half word to the left end of the computer registers was intended to make the multiplication of full words and half words consistent with each other. The computer arithmetic acts as if all numbers were fractions with the binary point immediately to the left of the leftmost bit (bit 1). Because of the shifting facilities there is no difficulty, however, in handling numbers with the binary point in different positions.

Absolute-Value Representation

Positive or negative binary numbers are always stored as absolute values with a separate bit indicating the sign. Although the complement of a number may appear in the accumulator during a subtraction, the number will be re-complemented immediately to restore the answer to the absolute-value form at the end of the operation. The reason for insisting on an absolute-value representation of negative numbers in the 701 does not become evident until one goes beyond simple addition and subtraction to such processes as multiplication, division, shifting, and overflow control. It is then seen that permitting complements to appear in the accumulator at the end of an operation—or, worse yet, in memory—results in a long chain of complications whenever negative numbers are involved. Having negative numbers represented in absolute-value form is a feature well worth the extra re-complementing step which must be provided internally for addition and subtraction.

A property of complement systems is that two numbers having the same absolute value but opposite signs are represented by two different arrays of numeric bits. They differ, in the complement system commonly used for parallel binary machines, by having ones and zeros interchanged. This duality is objectionable when doing non-arithmetic operations. The absolute-value system of the 701 provides the same numeric representation for numbers of opposite signs. On the other hand, there is a possibility in the 701 of the number zero having either a plus or a minus sign. An effort was made in the early stages of design to avoid a schizophrenic zero by incorporating "watch-dog" circuits which would force all zero results to have a plus sign. None of the schemes, however, were entirely consistent and they only led to a confusion of rules and exceptions. It was finally decided to permit either sign.

The difficulty, if any exists, is really one of logic and is not one introduced by the machine designer. In common usage, zero means "nothing," and there is no meaning to a sign of zero. In computer language, however, a zero must be represented by some configuration of bits including the sign bit which does not have, in addition to + and −, a third state of "no sign." The ordinary rules of arithmetic give no clue as to the choice between the two signs, and the only sensible thing to do is to establish an additional rule for the sign which the machine will attach to a zero result. The 701 has a "sticky sign" rule for addition or subtraction: When the result in the accumulator is zero, the sign previously in the accumulator sticks to that result. For multiplication and division the rule to be used is obvious; for example, $(+0)(-1) = -0$.

Actually, in the language of the machine, +0 and −0 are two different configurations of bits, and if −0 were arbitrarily suppressed by a special circuit the machine would be blind to one of the 2^{36} configurations of 36 bits. This would be a serious drawback in attempting to perform logical operations on non-numeric information. The machine would be unable to recognize, for example, the existence of a hole in only the sign column of a decimally punched card if that was the only hole in that row of the card.[1]

Multiplication and Division

For scientific computation the most important characteristic of the 701 is the high speed at which it multiplies or divides full words. If one adds together the time taken to obtain and execute a *Multiply* or *Multiply and Round* or *Divide* instruction, one obtains a total of 456 microseconds. However, the actual increase in the overall operating time for each multiplication or division inserted into the program of any real problem is, in general, very much less than that. The reason is that, following one of the instructions *Mpy* or *Mpy R* or *Div*,

a number of regeneration cycles are omitted during the execution of the next 12 instructions. Thus these 12 nstructions take considerably less time. (If the 12 instructions should include another *Mpy* or *Mpy R* or *Div*, the count of 12 starts over again.[3]) If we credit this saving against the *Mpy* or *Mpy R* or *Div* instruction which initiated it, we can state that the multiplication or division time has been reduced correspondingly below the norm of 456 microseconds. The theoretical minimum is 24 microseconds, although this is only reached if the 12 subsequent instructions happen to be of the *Store* instruction type. The actual time for a particular program can be calculated with the aid of TABLE I. In a practical program the effective time for each multiplication or division is usually observed to be around 150 microseconds.

Division should be the inverse of multiplication. Since multiplying two full-word (35-bit) factors generates a double-length (70-bit) product in the accumulator and *MQ* register, division has been similarly arranged to start with a double-length (70-bit) dividend stored in the accumulator and *MQ* register. This appears to be an elementary piece of logic, and it is surprising that a double-length dividend has not been provided in other computers of this type. A double-length dividend makes it much more convenient to program the division of integers or of mixed numbers (see section on *Shifting*). It becomes possible also to obtain a properly rounded 35-bit quotient without resorting to time-consuming double-precision techniques. It may be seen that, before starting to divide, one need only add one-half the divisor (that is, the divisor shifted one place to the right) to the low-order 35 bits of the dividend.

After a division without rounding, the proper remainder, with the sign of the dividend, is available in the accumulator. This is of great value for double-precision calculations as well as for certain logical operations.

Shifting

The four shift operations provide very flexible facilities for both arithmetic and logical purposes. The two accumulator shifts, *Accumulator Left* and *Accumulator Right*, cause the contents of the accumulator to be shifted to the left or right by the number of places specified in the address part. Excess bits at one end of the accumulator are dropped, and zeros are entered in the vacated places at the other end. The accumulator sign remains unchanged. There is no shifting into or out of the sign position.

The two long shifts, *Long Left* and *Long Right*, treat the accumulator and *MQ* registers as one long register with bit 1 of the *MQ* register joined immediately to the right of bit 35 of the accumulator register. Again the desired number of places by which the contents of the two registers are shifted jointly is specified by the address part. The signs of the two registers are made to agree: During *Long Left* the accumulator sign is set to agree with the *MQ* sign, during *Long Right* the *MQ* sign is set to that of the accumulator.

The long shifts are useful not only for moving entire words between the accumulator and *MQ* registers without going through memory, but also for positioning the binary point before a multiplication or division. Regardless of where the point is located in the fixed-point representation of two numbers, the 701 can multiply them and position the point in the final result in only four steps:

Load MQ
Mpy
Long Left or *Long Right*
Store

Rounding would take only one additional instruction. Division is equally easy, but the shifting is done before dividing instead of after.

For floating-point computations it is desirable to be able to calculate the difference of two exponents and then to shift a number of places equal to this difference. Since the difference can be quite large at times, the shift instructions have been arranged to accommodate shifts by as many as 255 places, although the result of a shift by a large number of places is, of course, just a string of zeros. The point is that, by providing for such a large number, a foreknowledge of the actual magnitude of a shift, which is to be calculated by the machine, is not needed to guard against exceeding the capacity.

At the other end of the scale, it is possible to specify zero places of shift. *Long Left* 0000 and *Long Right* 0000 do no shifting, but they do set the signs as described above. This feature provides a useful aid in programming.

Overflow Control

What to do about an overflow resulting from an arithmetic operation was recognized as a problem very early in the design of the 701, because known methods of dealing with an overflow were considered clumsy and inadequate. The method finally adopted for overflow control crystallized over a period of time, from ideas contributed by many different people, to become what is thought to be one of the most useful novel features in the 701 arithmetic.

As shown in Fig. 1, the accumulator capacity has been extended by providing two extra bit positions to the left of bit 1 which are not found elsewhere in the machine. These two overflow positions, labeled *P* and *Q*, have the same carrying, adding, and shifting facilities as the other 35 numeric bit positions of the accumulator, but they cannot receive information from memory nor store their contents in memory.

The overflow positions greatly simplify double-precision arithmetic. Double-precision addition, for instance, is reduced to the following steps:

1. Add the two lower halves and store the sum.

2. Shift the accumulator contents to the right by 35 places.
3. Add the two upper halves to the accumulator contents and store the sum.

Any overflow during step 1 is shifted to the right in step 2. It is thus automatically carried into the addition of step 3, regardless of the signs of the numbers.

The second overflow position was provided to take care of double-precision multiplication where, after four double-length partial products have been generated, one must accumulate up to three components of these partial products. This will give rise in some cases to a second overflow.[10]

Just providing storage for overflow bits is not enough; in many programs there has to be a way of recognizing that an overflow occurred. For this purpose an overflow indicator is provided (Fig. 1). The indicator is turned on by carries from bit 1 to bit P; it is also turned on when a binary one is shifted to the left out of bit position 1. The state of the indicator is tested by means of the *Transfer on Overflow* instruction (which then turns off the indicator). This permits the program to be altered, if desired, after an overflow. On the other hand—and this is just as important—the occurrence of an overflow can also be ignored. The 701 does not stop as a result of an overflow unless it is programmed to do so.

It should be noted that the overflow indicator does *not* indicate the contents of the overflow positions P and Q; rather it shows whether a binary one was carried or shifted out of bit position 1. Successive carries or a shift may have moved the one-bit well beyond the P and Q positions so that it is lost, but the indicator remains on. Also subsequent operations, other than *Transfer on Overflow*, will not turn off the indicator even though the P and Q positions may have been cleared. If overflows can occur at several stages of a program, the indicator may have to be tested only once at the end of a program.

It is important to have available both the overflow indicator and the overflow positions in the accumulator. The indicator provides a simple test to signal that an overflow occurred, the result without the overflow may be stored in memory, and yet the overflow bits can be recovered from the overflow positions merely by a shift to the right. It is worth pointing out that the two extra positions in the accumulator are not equivalent to building a 38-bit machine. Having two extra bits in the accumulator which cannot be stored in memory is perhaps more useful than increasing the capacity of the entire machine by two bits.

Rounding

Numbers have an unfortunate tendency to grow in length during the course of a calculation. To keep from losing the most significant bits at the left end of a number which has outgrown the capacity of the registers, the programmer may have to arrange for shifting numbers to the right at intervals. Thus, in order to save the most significant bits on the left, some of the least significant bits must be dropped on the right. This causes the numbers to be slightly in error, the error being cumulative unless it is compensated for. The programmer may reduce this error to a minimum by modifying, or "rounding," the remaining part of the number so as to compensate for the loss of the bits which were dropped.

This well-known problem in numerical analysis has no single solution. Different calculations may require different methods of rounding in order to reduce the residual rounding error to an acceptably low level. However, for the majority of applications a simple adjustment applied to the lowest remaining bit of the number to be rounded is sufficient, and it was decided, therefore, to provide a general-purpose *Round* instruction in the 701. More elaborate rounding must be programmed explicitly.

Two alternative methods of built-in rounding were considered. One was the short cut, found in many binary computers, of forcing a binary one in the lowest position which remains. The second, and somewhat more complicated, approach consists of adding a binary one in the highest position to be discarded and propagating carries, if any, to the left into the remaining portion of the number. The second alternative was adopted because the maximum possible rounding error is only half of that of the first method and because it is familiar to us as the binary equivalent of the rounding process commonly used in decimal hand-computing. For both methods successive rounding errors tend to cancel. If we deal only with numbers whose bits are randomly distributed, the probable error, obtained by averaging the errors of all possible bit combinations, is the same for both rounding methods and decreases as the number of bits to be dropped increases. The probable error vanishes if those numbers can have either sign at random, provided the rounding process is applied to the absolute value of the numbers. The absence of a strong bias to produce cumulative errors is an important criterion in the choice of a rounding process.

[10] The fact that three product components must be added is a necessary, but not a sufficient, condition to prove that a second overflow will actually occur. The proof will be supplied by an example: Assume that the product (01 11)(01 11) is to be formed by double-precision multiplication on a machine which, for convenience, has been reduced from 35 bits to a word length of 2 bits, excluding sign:

(01 11)(01 11)	Factors
10 01	Partial Products
00 11	
00 11	
00 01	
12 1	Carries
00 11 00 01	Product

The double overflow occurs when adding the 2-bit numbers in the second column from the right, so that two carries into the third column are needed. The same example can be applied to a 35-bit machine by adding 33 zeros at both ends of each factor.

In the 701 the *Round* instruction was designed to work in conjunction with the *Long Right* shift: *Long Right* places the bits to be dropped into the *MQ* register; *Round* then leaves a rounded number in the accumulator. Actually, the *MQ* register is not changed by the *Round* instruction. Its highest bit position is merely *tested*; if it contains a one, then a one is added to bit position 35 of the accumulator. The combination of just two instructions, *Long Right* and *Round*, provides a rounded right shift after the detection of an overflow. Because rounding after multiplication is even more common, a separate *Multiply and Round* instruction has been included. It does the same thing as *Multiply* followed by *Round* but needs no extra time for rounding.

Store Address and Extract

The *Store Address* instruction is used for the common logical process of changing the address part of an instruction. The contents of bits 6 to 17 of the accumulator are stored in memory in the corresponding part of the specified half word which would be the address part of an instruction. The other bits are left unchanged.

In some applications it is desirable to have a more general method of changing selected bits of a word in memory. The operation *Extract* has been provided, an operation somewhat different, however, from what is commonly known as "extract." In the 701 the *Extract* operation causes the specified full word in memory to be modified by changing every bit in memory to a zero (or + for the sign), if the corresponding bit in the accumulator is a zero (or +); wherever the accumulator has a one (or −), the corresponding memory position remains unchanged. Another way of putting it is that the operation causes every bit at the specified memory address to be replaced by the logical product of the bit previously there and the corresponding bit in the accumulator.

Extract is especially useful when storing more than two numbers, each of less than 18 bits, in a single full word, and when doing things such as counting the number of binary ones in a word. Non-arithmetic applications of this type may be rather limited, but the equipment required to incorporate *Extract* into the 701 turned out to be surprisingly simple.

Sense

The *Sense* instruction is a special means of communication between the computer and the outside world. It is used in a variety of ways to control input-output equipment and to turn on signal lights on the operator's panel. The pluggable control panels for the card reader, card punch, and printer have terminals which can be sensed or actuated by the *Sense* instruction. Of particular interest is its use in conjunction with six "sense switches" on the operator's panel.

Each of the switches can be addressed individually by the *Sense* instruction. If the switch is up, nothing happens and the program continues to the next instruction. If the switch is down, however, the program skips the next instruction. Thus *Sense*, followed by an unconditional *Transfer*, can provide the operator with a set of alternative programs under control of the front panel switches.

For example, one switch might be used to stop the computer at the next convenient break-point. Another switch might be set up to cause the machine to print selected data for the purpose of diagnosing troubles, the printing being normally suppressed to save time. The six switches have no pre-determined functions; the programmer may use them in any manner to the limit of his ingenuity.

Checking

In the absence of complete agreement among mathematicians as to the proper methods of checking scientific and engineering calculations, and because of the cost of building checking facilities of doubtful value, the policy adopted was to provide only very little built-in checking. The engineering effort, which would have been spent on designing extra equipment, was diverted to increasing the reliability of the functional parts of the computer. The experience already gained in running the laboratory model of the 701 as a full-fledged computing installation has amply justified this policy. It has confirmed the belief that the best approach is to rely almost entirely on programs to discover errors from any source, including the computer, the data, and the programs themselves.

External System

Program Control of External Units

Card readers, card punches, printers, magnetic tape units, and magnetic drum units represent a wide variety of input-output and external storage devices, all of which are under control of the stored program. Control is accomplished by a basic procedure which is common to all these external units. Only the instructions *Write*, *Read*, and *Copy* are required, as well as *Set Drum* for the drums. To this basic procedure are added a few special functions required by the tape units.

We need to define here a few terms which are convenient in describing the operation of the 701. It is to be hoped that they will be found useful enough to become standard terms in the computer field.

Writing consists of moving information from electrostatic memory to an output or storage unit (card punch, printer, tape, or drum).

Reading consists of moving information from an input or storage unit (card reader, tape, or drum; also checking pulses from the printer) to electrostatic memory.

A Unit Record is a sequence of full words, which are read or written as a unit during a limited period of time or between starting and stopping of the appropriate input-output device. Specifically, for a card

reader or punch, a unit record is the information contained on one card. For a printer, a unit record corresponds to one line of printing. On tape, a unit record is written as a continuous sequence of bits, the end of which is marked by an *end-of-record gap*. Thus two unit records on tape are always separated by an end-of-record gap which is long enough to permit stopping and starting the tape safely between records. (In the 701 the number of words in a unit record is variable and is determined by the program.)

A File is a sequence of unit records intended to be used together. It may be a deck of related cards or a page of printing. A file on tape consists of all the unit records on one reel starting at a standard point near the beginning of the reel and ending with an end-of-file gap. Any information beyond the end-of-file gap, which has been left unerased from previous use of the reel of tape, is not part of the file and is not read. The length of a file on tape is arbitrary, except that it cannot exceed the capacity of a reel of tape.

(It may be noted that the word "transfer" is being meticulously avoided in connection with moving information. To avoid confusion, the word "transfer" has been reserved exclusively for the function of transfer of control for which there appears to be no satisfactory alternative terminology.)

The instruction *Write* is used to specify the desired output unit and to prepare it for writing. Furthermore, it starts the mechanical motion of card punches, printers, or tape units; drums do not need starting since they are in continuous motion. Since punching, printing, and writing on tape are sequential operations, there is no need to define where to start writing on punches, printers, and tape units. Drums, however, require a separate instruction, *Set Drum*, to define the address on the drum at which writing is to start. *Set Drum* follows a *Write* which specifies the desired one of several drums.

After *Write*, and if necessary *Set Drum*, a sequence of *Copy* instructions is given. Each *Copy* specifies an address in electrostatic memory from which the next word is to be copied for the purpose of writing. As many *Copy* instructions are given as there are words in the unit record to be written. When the unit record is finished, the program proceeds to other operations. The lack of further *Copy* instructions causes the output unit to stop automatically. Before another unit record can be written on this or any other output unit, another *Write* instruction must be given. Thus *Write* always signals the start of a new record, and there is no doubt about whether a given *Copy* instruction was intended for the old or the new record.

Reading is similar to writing. A *Read* instruction selects an input unit and starts it if necessary. For drums *Set Drum* again specifies the starting address on the selected drum. A series of *Copy* instructions is then needed, one for each word to be read. Each *Copy* specifies where in electrostatic memory the word just read is to be stored. Again, each *Read* instruction indicates that a new record is to be started.

During reading it is necessary to be able to sense when the end of a record or a file occurs. This is done by placing two *Transfer* instructions immediately after the *Copy* instruction. The computer may then skip one or both *Transfer* instructions: one at the end of a file, two at the end of a record. Thus the program can be arranged to follow an appropriate course of action. In the middle of reading a record, there is no skipping after *Copy*.

Thus the basic series of instructions for writing or reading one unit record is:

Write or *Read*
(*Set Drum* for drums only)
Copy
Copy
Copy
.
.
.

Actually, the sequence of *Copy* instructions is usually replaced by a short iteration loop containing only one *Copy* instruction. The skipping feature mentioned was designed particularly for ease of interrupting such a loop at the end of a record or file.

When manipulating tape the above set of basic instructions is supplemented by three more: *Read Backward* for the purpose of reading a unit record on tape backward, *Write End of File* for writing the end-of-file gap previously mentioned, and *Rewind* for rewinding a reel of tape all the way back to the starting point. The printer uses the basic instructions in a rather special way when the printing is to be checked.[1]

Variable Record Length

From the above description it is evident that each word to be moved requires the execution of a separate *Copy* instruction which specifies the memory location of that word. Since an arbitrary number of *Copy* instructions can be given following *Read* or *Write*, the number of words in a unit record is likewise arbitrary within wide limits. The upper limit of the record length on tapes or drums is set by the capacity of the tape reel or the drum, which is far beyond practical requirements. The lower limit is one word. On cards the upper limit is set by the medium itself since no more than 12 rows and 80 columns are available. Actually, up to 24 words can be punched in binary form, covering all the rows and 72 of the 80 columns. Having a variable record length is particularly convenient in handling programs, since successive programs on a single program tape may be of widely different lengths. It also becomes unnecessary to fit data stored on tapes or drums into blocks of a fixed length. For instance, in matrix calculations each row (or column) of a matrix of any size can be arranged to be a unit record of appropriate length, and the complexity of the program is not affected by the order of the matrix.

The fact that external units are controlled by the pro-

gram on a word-by-word basis has other advantages. It is not necessary to store words in memory in a fixed sequence; the program can be arranged to handle words in ascending or descending sequence, or in no sequence at all. Also a considerable amount of equipment for controlling external units was saved.

Overlap of Internal and External Operations

Following the execution of a *Copy* instruction there is time for the execution of several other instructions before the next *Copy* must be given. The computer is fast enough to go through the short iteration loop previously mentioned, which calculates the address for the next word, even at the high word rate of magnetic tape. The time intervals between successive words are much longer with mechanical card equipment and printers, so that it is possible to proceed with binary-decimal conversion programs in the intervals which occur during card reading, punching, and printing. There is, of course, a limit on the time available for computer operation in the interval between words, the limit depending on the equipment being used. Since the timing of an external unit cannot be changed once the unit has been set in motion, it is up to the computer to synchronize operations and to wait for the external unit whenever necessary. Programs are written with these limits in mind.

Fortunately, it is not necessary for the average programmer to concern himself with these details. He can use already prepared subprograms which are known to work and which he may not even understand. All he needs to specify is which unit, where in memory, how many words, etc. Adequate input-output subprograms were devised along with the design of the equipment to make sure that the projected design would work well. Developing the subprograms was really part of the design of the 701 system. The input-output subprograms are an alternative to having more equipment which would perform similar functions. There is one important difference, however. The people who designed the original subprograms had no doubts that ingenious users would later devise much better subprograms which would make the same computer a better machine. With built-in equipment such improvements can be made only with great difficulty. This feature has already paid dividends in that many new and improved input-output subprograms have been written since the 701 design was completed.

Self-Loading Procedure

Another area where the programming facilities of the computer have successfully replaced physical hardware is in the loading of a new program into the computer. There is a load button and a selector switch on the machine, but they do just barely enough to get the process started. The rest is accomplished by a technique sometimes called the "bootstrap technique." The switch determines whether the program is to be loaded from tape, cards, or drum. Pushing the load button then causes one full word to be loaded into a memory address previously set up on the address entry keys on the operator's panel, after which the program control is directed to that memory address and the computer starts automatically. In effect, the load button has executed the short program:

$+Read \quad x$
$+Set\ Drum$ 0000 (applies to drums only)
$-Copy \quad y$
$+Tr \quad y$

where x is the identification of the input unit set up on the load selector switch and y is the (even) memory address set up on the address entry keys. So far one full word has been entered into memory, nothing more. The full word may, however, consist of two instructions of which one is a *Copy* instruction which can pull another full word or pair of instructions into adjacent addresses. The new pair may contain another *Copy* instruction to continue the process as long as necessary. For each *Copy* placed in memory another instruction is gained, so that one can rapidly build up a program loop which is capable of loading the actual operating program.

	Location	Instruction		
	0000	$-Copy$	0002	
	0001	$+R\,Add$	0003	
	→ 0002	$+Add$	0000*	
Loading	0003	$-Copy$	(0004)†	End-of-
Loop	0004	$+Store\ A$	0003	record
	0005	$+Tr$	0002	skip
	0006	Start of operating ←		
	0007	program to be loaded.		

* This instruction adds the contents of location 0000 to the contents of location 0003, which in effect adds 0002 to the address part of the *Copy* instruction at 0003 every time around the loading loop.

† The parentheses indicate that the address part will be modified during the execution of the program. 0004 is the initial value loaded by the first *Copy*.

Fig. 3—A self-loading program.

The self-loading technique is illustrated in the program of Fig. 3 where the starting address y is, for convenience, taken to be 0000. The first six instructions constitute the self-loading procedure. They are followed immediately by the instructions of the program to be loaded. The complete set of instructions forms a unit record which may be punched on a card or written on tape or drum. Pushing the load button causes the first pair of instructions to be loaded into addresses 0000 and 0001. The *Copy* instruction at 0000 loads the next pair into 0002 and 0003. The process may be seen to be self-sustaining. When the entire program has been loaded, an end-of-record signal is obtained which, as was mentioned earlier, automatically causes a skip from the *Copy* at 0003 to the start of the operating program at 0006. The above self-loading procedure is one of several in current use.

Tape Programming

The last section showed how a new program may be loaded from tape, drums, or cards. Tape and drum are similar in that they permit programs to be entered into memory at high speed. If repeated reference must be made to different parts of a long program which exceeds the capacity of electrostatic memory, these parts may be stored on drums and called into memory as needed. The drums must, however, be loaded first from an external source, that is, from tape or cards. In many computer installations a good deal of effort has gone into creating libraries of standard programs designed to simplify the programmer's task. Even a large library can be stored compactly on a single reel of magnetic tape. With the high speed of tape little time is wasted in skipping over unwanted programs in order to pick out the programs needed for a given job. For a large problem it may be more efficient to assemble a special program tape, particularly when the program may have to be reentered on many different occasions. The effort required to prepare the tape once is repaid in the saving of operating time.

It will be noted that no equipment has been provided to key new programs by hand directly on tape. Such equipment would be simple enough to build, but the nature of a continuous medium like tape and the invisible method of recording make it a difficult problem to correct errors made by the key-recording operator. Inserting a set of missing instructions would be especially awkward. Instead, new programs for the 701 are first punched on cards in decimal form, one instruction to a card, using standard equipment to punch and print the program. Corrections can be easily made at that stage by substituting or inserting cards. Once the program is known to be correct, it may be translated to binary form and transcribed on tape in a brief preliminary run on the computer.

Card Programming

For short programs it may be more convenient to have the computer condense the decimal cards, on which a new program has been punched, to binary cards. A binary card can hold more than 40 instructions. A program of several hundred instructions thus occupies only a few cards and is loaded into the computer in just a few seconds; such a program is manipulated more easily on cards than if it were condensed even further on a few inches of tape. Many programmers prefer to keep their own private library of program cards, modified to fit their special requirements. While at his desk the programmer may then assemble into a single deck of cards some prepunched library programs together with cards especially punched for the problem to be solved. He need only give the assembled deck to an operator who may then combine it with other program decks to run off a whole series of problems as time becomes available on the computer. Thus by assembling his cards complete with auxiliary loading programs, etc., each programmer has control over the machine as far as his program is concerned, and operating errors are held to a minimum even when the machine is being run by relatively inexperienced operators. All the operator usually has to do is to insert the tapes, if any, and press the load button.

This technique has been found useful even when using longer programs which are stored on tape. Cards may be employed to sequence the programs and to introduce auxiliary programs. For instance, when troubles arise, different program cards may be inserted to print out indicative information, to modify the program, and to make corrections. The manual controls available on the operator's panel could be used to do all these things, but it is much too easy to push the wrong button. A far safer procedure is to rely on the prepunched cards which give the operator a chance to double-check what he is going to do. Thus the card reader can serve as an extension of the manual controls. For that reason a card reader has been placed within easy reach of the operator when he is seated at the operator's station.

Operator's Panel

The manual controls which are used most often by the operator are very few in number. They include the load controls already mentioned, controls to start, stop, and reset the computer, and buttons to turn the power on and off. The sense switches described earlier also fall into this category. The other controls are more likely to be used by the engineer servicing the machine. Instructions can be executed directly from the operator's panel, by-passing the stored program. For this purpose a set of 18 instruction entry keys are provided to enter an instruction into the program registers. Another set of 18 keys, the *MQ* entry keys, may be used to enter a half word into the *MQ* register. Neon lights are provided to display the contents of all registers, and the contents of any memory address may be inspected by the use of controls which bring the desired word into the memory register.

When maintaining the machine, the engineer may want to analyze a certain part of a program. For that purpose he may start the program at high speed until the instruction counter neon lights show that the desired point is being approached. He may then interrupt the machine and slow it down to a rate of about 5 instructions per second by holding down the multiple-step key. This is slow enough to permit him to stop within one or two instructions of the goal. By pressing the single-step key he can then proceed one instruction at a time, observing the results on the neon lights. High-speed operation can be resumed at any time by pressing the start button.

A considerable design effort was made to simplify the operator's panel and the associated circuits. Whenever feasible, circuits already required for other purposes were utilized and elaborate sequencing circuits were

avoided. For instance, there is no special equipment for obtaining direct access to memory from the operator's panel. Information can only be entered directly into the *MQ* register; to store the information in memory, a *Store MQ* instruction must then be executed manually. The program loading procedure previously described is another example. The simple, yet effective, arrangement of the operator's controls was made possible by the decision to plan the operator's panel right at the start, along with the other input-output devices.

V. Conclusions

The primary characteristics of the new IBM Type 701 Computer, which are summarized in TABLE II, are high speed, large storage capacity, and an unusually complete array of input and output equipment. The 701 also includes a number of other new features which were described in this paper. Among them are:

1. Independently addressed half words and full words, providing direct access to each single-address instruction.
2. High-speed multiplication and division, requiring in effect only about 150 microseconds in many practical applications.
3. Division with double-length dividend and proper remainder.
4. Improved shift instructions.
5. Flexible control of arithmetic overflow.
6. Provision for altering a program manually in a manner predetermined by the program.
7. Integration of all input-output equipment with the computer.
8. Line-at-a-time printer capable of printing numeric and alphabetic characters.
9. Variable record length for input, output, and external storage.
10. Provision for flexible program control by tape or cards.
11. Programs substituting for equipment whenever possible, to achieve greater reliability and flexibility.

When taken together, these individual contributions to the art of building computers constitute a major improvement in the usefulness of such machines for scientific and engineering calculations.

Acknowledgments

The logical system design of the 701 was primarily the responsibility of an engineering group, headed by N. Rochester, which included M. M. Astrahan, J. H. Holland, R. W. Murphy, and the author. Another and larger group of engineers under the direction of J. A. Haddad, produced the engineering design of the 701, and W. F. McClelland was in charge of a group of mathematicians responsible for the mathematical planning and application studies; both of these groups made substantial contributions to the ideas reported here. Generous assistance was received during the preparation of this paper from many members of the IBM Engineering Laboratory in Poughkeepsie, N. Y.

Engineering Description of the IBM Type 701 Computer*

CLARENCE E. FRIZZELL†, ASSOCIATE, IRE

Summary—**The IBM Electronic Data Processing Machines Type 701 and Associated Equipment constitute a large scale electronic digital calculator. These machines, which incorporate the newest devices for input, output, and storage, are currently in production at the IBM Plant 2, Poughkeepsie, N. Y. This paper presents a general engineering description of the 701 with emphasis on what are believed to be new techniques and concepts.**

Introduction

THE IBM ELECTRONIC Data Processing Machines Type 701 and Associated Equipment constitute an electronic digital calculator designed primarily for large scale scientific and technical computing. The calculator operates internally in the binary number system and is controlled by means of a stored program. The outstanding features of the machine include large capacity high-speed storage, dynamic sequential control, and a flexible input and output system.[1,2]

This paper presents a general engineering description of the IBM Electronic Data Processing Machines. The characteristics and functions of the computer are described with emphasis on what are believed to be new techniques and concepts.

A typical 701 installation, shown in Fig. 1, consists of the following units:

(a) Analytical Control Unit
(b) Electrostatic Storage Unit
(c) Magnetic Tape Readers and Recorders
(d) Magnetic Drum Reader and Recorder
(e) Punched Card Reader

* Decimal classification: 321.375.2. Original manuscript received by the Institute, June 25, 1953.

† International Business Machines Corp., Engineering Laboratory, Poughkeepsie, N. Y.

[1] M. M. Astrahan, and N. Rochester, "The Logical Organization of the New IBM Calculator," IBM Corp.; 1952.

[2] L. D. Stevens, "Engineering organization of input and output for the IBM 701 electronic data-processing machine," (Review of input and output equipment used in computing systems)—*Joint AIEE-IRE-ACM Computer Conference*, pp. 81–85; March, 1953.

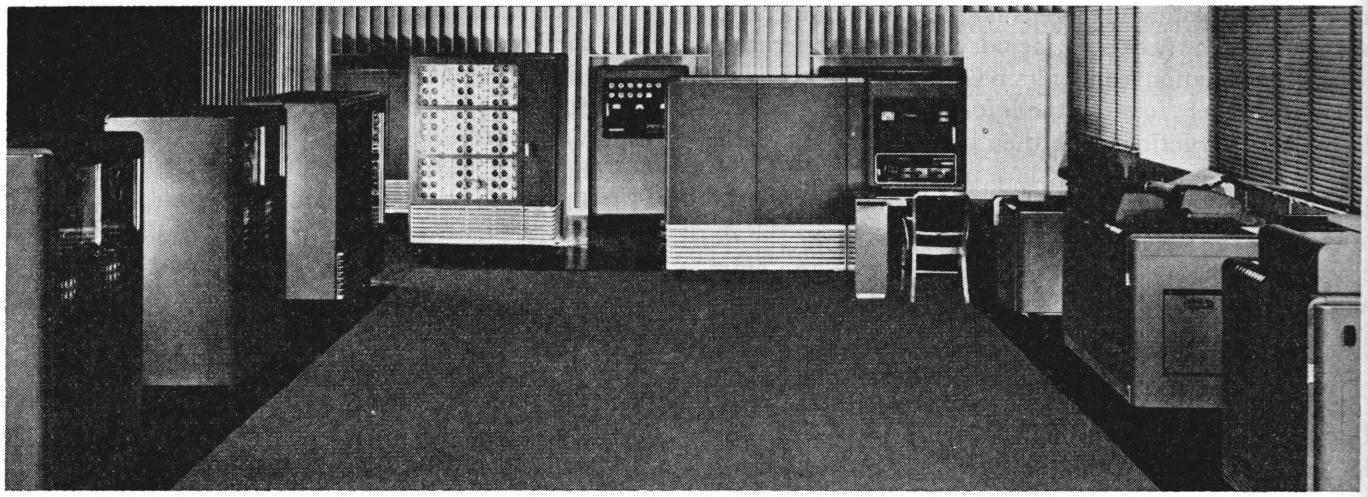


Fig. 1—A typical 701 installation.

(f) Alphabetical Printer
(g) Punched Card Recorder
(h) Power Distribution Unit
(i) Power Units.

The full length word used throughout the machine consists of 35 binary digits and a binary sign digit. The calculator uses a single address system which is capable of calling for 36-bit full words or either of two 18-bit half words, which may be stored in a full word location. In fact, all machine instructions are half words and include a 5-bit operation part, a 12-bit address part and a sign bit. The sign bit indicates whether an operation is to affect a full or half word. A full word, a half word, and the arrangement of the three parts making up a machine instruction are illustrated in Fig. 2.

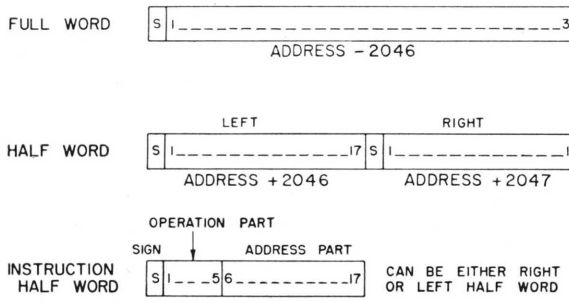


Fig. 2—Organization of words.

The high operating speed is made possible by the use of parallel transmission and arithmetic systems in conjunction with the rapid random-access Electrostatic Storage. Instructions are stored in the Electrostatic Storage and any one is available to the computer in one 12-microsecond machine cycle.

The calculator is capable of executing 33 different operations[3] as shown in Fig. 3. Twenty-five pertain to arithmetic operations, including shifts, transfers, and a programmed stop. The remaining eight control the various input and output units.

[3] W. Buchholz, "Systems design features of the IBM type 701 computer," PROC. I.R.E., pp. 1262–1275; this issue.

ARITHMETIC OPERATIONS	
ADD	SUBTRACT
RESET AND ADD	RESET AND SUBTRACT
ADD ABSOLUTE	SUBTRACT ABSOLUTE
STORE	LOAD M-Q
STORE ADDRESS / EXTRACT	STORE M-Q
MULTIPLY	DIVIDE
MULTIPLY ROUND	ROUND
LONG LEFT	LONG RIGHT
ACCUMULATOR LEFT	ACCUMULATOR RIGHT
TRANSFER	TRANSFER ON ZERO
TRANSFER ON PLUS	TRANS ON OVERFLOW
STOP AND TRANSFER	NO OPERATION
INPUT/OUTPUT OPERATIONS	
PREPARE TO READ	PREPARE TO READ BACKWARDS
PREPARE TO WRITE	WRITE END OF FILE
REWIND TAPE	SET DRUM ADDRESS
COPY AND SKIP	SENSE AND SKIP

Fig. 3—Calculator operations.

PROGRAM CONTROL

The Program Control section of the 701 consists of the following logical blocks, which are shown in Fig. 4: Instruction Register, Instruction Counter, Operation Decoder and a Memory Deflection Register. The Instruction Register receives an instruction from the Electrostatic Storage, via the Memory Register. The Operation Part provides coded information to the Operation Decoder (diode matrix) whose output line (one of 32) conditions all control circuits required to execute the current instruction. The Sign Part is normally directed to the Electrostatic Storage control circuits to indicate either a full or half word operation. (In one instance the Sign Part is used to distinguish between two operations using the same Operation Part, *Store Address/Extract.*) The Address Part is used to specify an Electrostatic Storage location when reading or storing. It is also used to specify an Input-Output Unit, a drum address or the number of columns to be shifted.

The Address Part of the Instruction Register (12 column) consists of a four position trigger register (high order end) and an eight position binary counter. Coded address information is stored in these two parts and may be transferred to the Memory Deflection Register to select the desired Electrostatic Storage location.

The Address Part is also used to specify a particular Input-Output Unit when it is associated with a general Input-Output select instruction such as *Read* or *Write.*

The address is decoded by a Class Selector Matrix which energizes a specific select line.

The Address Part of *Shift* instructions is used to specify the number of columns to be shifted. This information is read into the eight position address counter (lower eight positions provide a maximum shift of 255 places) which counts down in synchronism with the shifting operation. A count of zero indicates the completion of the operation.

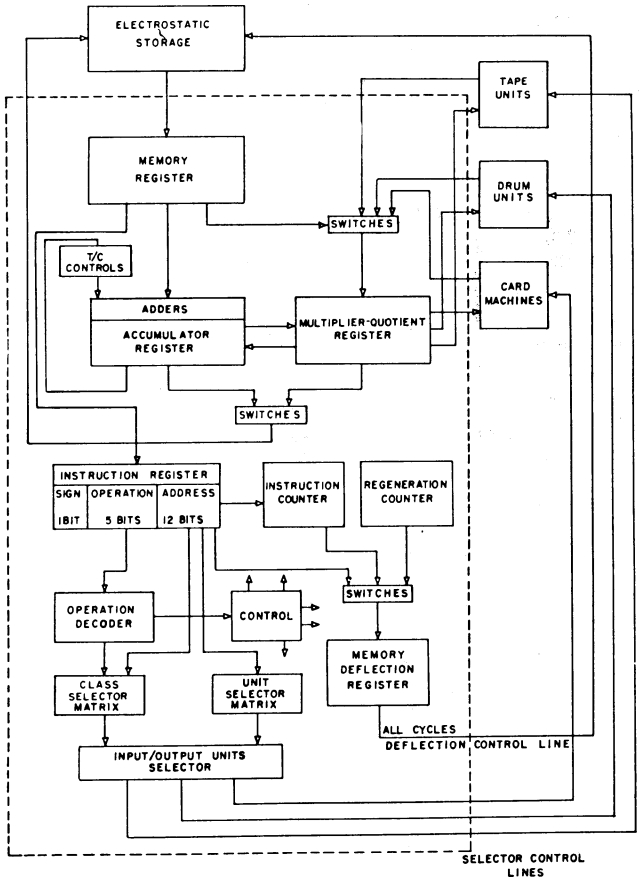


Fig. 4—Simplified block diagram of the IBM 701.

The Instruction Counter is a 12-position binary counter that may be advanced serially or read into in parallel. This counter contains the Electrostatic Storage location of the next instruction and is advanced by one at the end of each operation. Transfer instructions are executed by reading the 12-bit address from the Instruction Register into the Instruction Counter in a parallel manner.

The Memory Deflection Register, at all times, provides address information to the Electrostatic Storage Deflection Circuits. This information is obtained from the Instruction Counter, the Instruction Register or the Regeneration Counter. The Regeneration Counter provides the address of the next two full words to be regenerated. It is advanced by one, each regeneration cycle.

Arithmetic Section

All arithmetic is performed in the binary number system. All numbers transmitted throughout the calculator or stored in the Electrostatic Storage are true numbers. Ones' complement numbers are used when subtracting and the resulting difference, if it is a complement, is automatically converted to true form. The sign bit is then set in accordance with algebraic laws.

Addition is accomplished at the rate of one each 60 microseconds or 16,666 per second. This includes the time for obtaining the instruction to be performed and the number to be operated upon. This rate of performing addition also includes the Electrostatic Storage regeneration associated with the instruction and is discussed more fully in the section on Timing and Control Functions.

Multiplication is performed by the process of repeated addition and shifting. Division is performed by repeated subtraction and shifting. The time required for either operation is 456 microseconds. Furthermore, the net time required to execute a section of program, consisting of *Add* or *Store* or other instructions, is considerably reduced when this section of program immediately follows *Multiply*. (This technique is described in detail in the section on Timing and Control Functions.)

The principal elements of the Arithmetic Section, shown in the block diagram (Fig. 4), are as follows:

(a) Memory Register
(b) Accumulator Register
(c) Multiplier/Quotient Register
(d) Binary Adders
(e) True/Complement Controls.

The three registers used in the Arithmetic Section consist of Microsecond Delay Units[4] which were developed by B. L. Havens of the Watson Scientific Computing Laboratory. Each unit may be considered as a storage element whose output voltage level, during each one-microsecond period, is maintained at the same value as the input voltage level for the *preceding* one-microsecond period. Thus each unit will continuously store information when its output is connected to its input. These units may be conveniently used to form a one-megacycle shifting register by electronically switching the output of one unit to the input of an adjacent unit.

The delay feature is especially useful in the Accumulator Register. Here the Accumulator Register outputs serve as one set of inputs to the Adders and the Memory Register outputs serve as the other set. Each Adder is a germanium diode, matrix type, requiring static inputs to provide a sum at its outputs. In addition to providing information to the two inputs, time must be provided for carries to propagate and the resulting sum to reach a steady state. The delay feature, which enables the unit to read-in and read-out simultaneously, is used here to provide static Accumulator Register outputs to the Adders for a one-microsecond period while the Accumulator Register inputs are connected to the Adder outputs to read-in the sum. At the start of the next one-

[4] H. D. Ross, "The arithmetic elements of the type 701 computer," Proc. I.R.E., pp. 1287–1294; this issue.

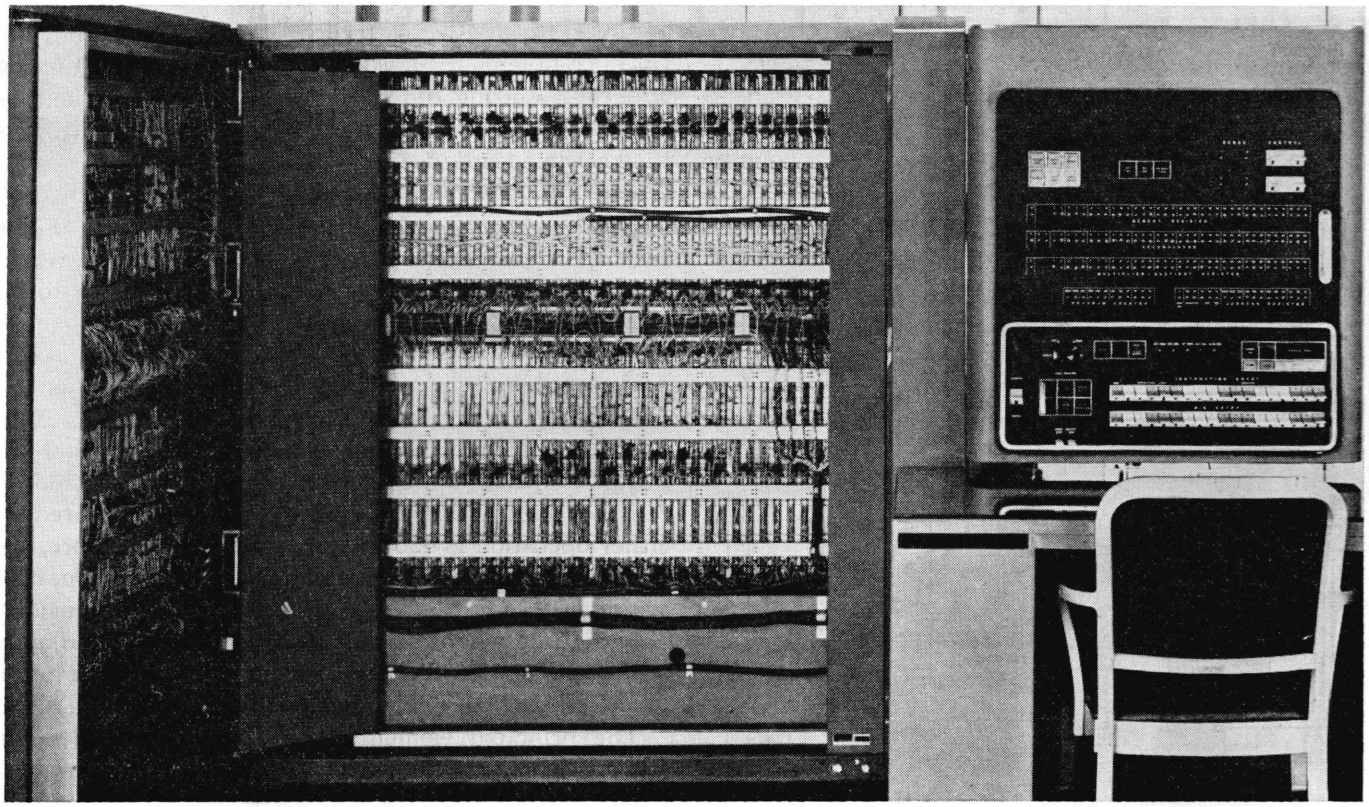


Fig. 5—Analytical control unit.

microsecond period the sum will appear in the Accumulator Register.

The Accumulator Register, having a capacity of 37 bits and a sign bit, stores the sum in the case of addition or the difference in the case of subtraction. It is also used as a shifting register, as product storage when multiplying, and as the dividend register when dividing. The Accumulator Register and Multiplier/Quotient Register are frequently coupled automatically to form a 72-bit register. In the case of multiplication, a 70-bit product may be developed with the most significant half (35 bits) appearing in the Accumulator Register, and the least significant half in the Multiplier/Quotient Register. The Accumulator Register is 38 bits long instead of 36 because it contains two additional positions on the left called Overflow positions. These are never stored directly but participate in arithmetic and shifting operations.

The Memory Register, having a capacity of 35 bits and a sign bit, functions as a storage register for all information coming from the Electrostatic Storage. In arithmetic operations this register stores the addend, subtrahend, multiplicand and the divisor.

The Multiplier/Quotient Register, having a capacity of 35 bits and a sign bit, serves many functions. It possesses shifting properties when coupled to the Accumulator Register. During multiplication, it is used to store the multiplier factor, which is eventually replaced by the least significant half of the 70-bit product; during division it stores the quotient as it is developed. The multiplier/Quotient Register also serves as a storage register for the asynchronous input or output devices.

Timing and Control Functions

The Analytical Control Unit, shown in Fig. 5, contains all master timing circuits, the arithmetic circuits, and the general controls required by the calculator.

The pulse repetition rate at which the computer functions is one megacycle per second. A twelve-stage ring of trigger circuits is used to generate the 12 one-microsecond pulses which together constitute one basic machine cycle. Since this is a synchronous machine, the selection of the basic cycle duration was based chiefly on timing requirements for the Electrostatic Storage.

The regeneration requirements of this type of memory may be met in a number of ways. In the 701, regeneration is considered a part of each instruction and is therefore distributed throughout the execution of a program. This is accomplished by assigning specific functions to each of four types of basic machine cycles: Instruction, Execution, Execution/Regeneration, and Regeneration. (These will be referred to as I, E, E/R and R cycles.) Since the major functions to be performed are the reading and interpretation of an instruction, the execution of an instruction and the regeneration of the electrostatic memory, the functional assignment by type of cycle is as follows:

(a) During an Instruction cycle (I) the next instruction is obtained from the Electrostatic Storage location specified by the Instruction Counter and set into an internal register called Instruction Register (Via the Memory Register). From here instruction is interpreted by a decoding matrix.
(b) During an Execution cycle (E) a reference is

made to the Electrostatic Storage at the address specified by the address part of the current instruction. The instruction determines whether a number will be stored or read out.

(c) During an Execution/Regeneration cycle (E/R) action is taken to execute the current instruction and simultaneously one spot on each of the 72 cathode-ray tubes is regenerated (the location is specified by a Regeneration Counter). This type of cycle is used for the execution of an instruction (follows I cycle for instructions such as *Transfer* and *Shift*) when no additional information is required from the Electrostatic Storage. It is also used, following the read-out of data from the Electrostatic Storage, when additional time is required to complete the execution operation. (This is described in the following discussion of the operation *Add*).

(d) During a Regeneration cycle (R) one spot on each of the 72 cathode-ray tubes is regenerated just as during an E/R cycle. This type of cycle is used to complete the regeneration associated with each instruction or to preserve the contents of the Electrostatic Storage when the machine is in a ready or stop status. The method of associating R cycles with each instruction is described later.

Each of the 33 machine operations consists of a predetermined pattern of these basic cycles. The type of cycles, the number of cycles required and their sequence depends upon the nature of the operation to be performed. The cycle pattern for four typical operations is shown in Fig. 6.

TRANSFER	I,E/R,R,R = 48 MICROSECONDS
ADD	I,E,E/R,R,R = 60 MICROSECONDS
STORE	I,E,R,R,R = 60 MICROSECONDS
MULTIPLY	I,E,E/R----36 E/R CYCLES=456 MICROSECONDS

Fig. 6—Cycle pattern for four typical operations.

An *Add* operation consists of the following sequence of cycles: I, E, E/R. These are usually followed by two R cycles, as will be described later. During the I cycle the instruction is obtained from the Electrostatic Storage and the operation part is decoded by the Operation Decoder as the operation *Add*. The combination of the two conditions, I cycle and the Operation Decoder output *Add*, is used to call for an E cycle next. It should be noted that all machine operations are initiated by a similar combination of conditions.

The 12 one-microsecond timing pulses which make up all basic cycles are used as an additional condition to specify the time within a cycle for an operation to take place, For example, the call for the E cycle just mentioned occurs at the end of the I cycle. This is accomplished by mixing the *Add* condition with the twelfth (or last) timing pulse of the I cycle to generate a *Go To* E cycle signal. Similarly, during the E cycle, the word to be added is obtained from the Electrostatic Storage and placed in the Memory Register. Since insufficient time remains to complete the addition, the *Add* Operation Decoder output is mixed with the E cycle and last timing pulse to generate a *Go To* E/R cycle signal. During the E/R cycle addition is completed, and the sum placed in the Accumulator Register. Simultaneously, regeneration automatically occurs.

The regeneration requirements of an instruction are maintained by a circuit known as the Regeneration Control. This circuit assures the Electrostatic Storage of a minimum of three regeneration (E/R or R) cycles per instruction executed. This amount of regeneration is sufficient to insure reliable operation with any program.

Since each instruction is made up of a predetermined pattern of I and E or E/R cycles and since the pattern differs for various instructions, a signal must be generated to indicate the completion of an operation. This signal is generated at the end of the last cycle of the execution operation and is called the End of Operation pulse. It is used to sample the status of the Regeneration Control unit to determine whether the calculator may proceed to the next Instruction cycle or call for a Regeneration cycle. Should the fixed cycle pattern of the instruction just completed embody a minimum of three E/R cycles, the Regeneration Control circuit would permit the calculator to call for an Instruction cycle and read out the next instruction. However, if an instruction such as *Add*, whose cycle pattern is I, E, and E/R, was the one just completed, then two Regeneration cycles will be required before proceeding to the next instruction. The Regeneration cycles underscored in Fig. 6 are those called for by the Regeneration Control circuit to meet the requirement of three Regeneration cycles per instruction executed.

It will be noticed that 36 E/R cycles are shown as part of the cycle pattern for the instruction *Multiply*. Since this number of cycles greatly exceeds the normal regeneration requirement, the Regeneration Control circuit permits the following twelve instructions to be executed at a reduced rate of regeneration. This function is popularly called the "Twelve Free Games" feature. It means that the only regeneration performed durthe twelve instructions following a *Multiply*, *Multiply* and *Round*, or *Divide* instruction (all three instructions have the same cycle pattern) is that which is provided by the E/R cycles.

The significance of the Free Games feature is that the instruction *Multiply*, which alone requires 38 machine cycles (456 microseconds), speeds the execution of the following twelve instructions so that, in effect, the multiplication time is reduced. This may be illustrated by assuming that the instruction *Multiply* is followed by six *Store* and six *Add* type instructions (see Fig. 6 for cycle pattern of each). The total time required for these 13 operations, without the Free Games feature,

would be 456+6(60)+6(60) or a total of 1176 microseconds. However, since the reduction of the regeneration requirement associated with each instruction is the basis of this feature, the Regeneration Control circuit does not require the execution of the three R cycles following each *Store* instruction (see Fig. 6) or the two R cycles following each *Add* instruction.

By this means the time required to execute 30 machine cycles, at 12 microseconds each, is saved. This represents a decrease of 360 microseconds in the time specified (1176 microseconds) to perform the 13 operations. Since this time saving is achieved by performing a multiplication it is reasonable to subtract the 360 microseconds from the 456 microsecond multiply time which results in an effective multiplication time of 96 microseconds.

To summarize, the effective multiplication time is a function of the 12 instructions which follow and may vary from 24 microseconds (a rather unlikely situation requiring 12 following instructions such as *Store*) to 456 microseconds in the case where *Multiply* is followed by *Multiply*.

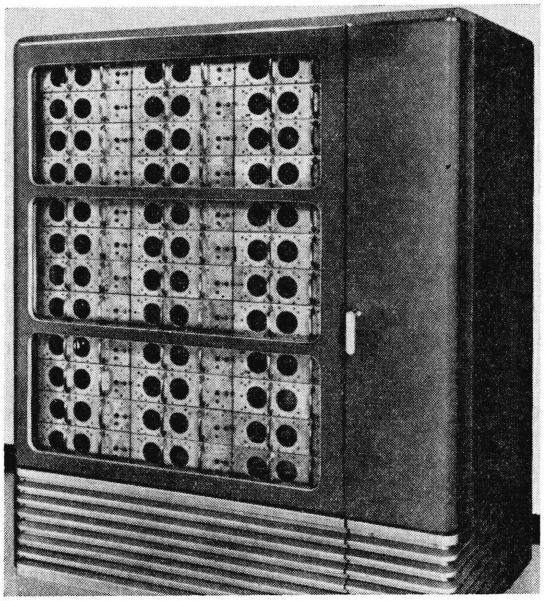


Fig. 7—Electrostatic storage unit.

Electrostatic Storage

The Electrostatic Storage Unit shown in Fig. 7 serves as the high-speed working memory and is an integral part of the 701. Instructions and data frequently required by the program are stored here, and all information transmitted between the various input and output units must pass through the Electrostatic Storage. The principle of storage used is essentially the dot-dash technique as described by F. C. Williams[5] and the access time may be considered to be one 12-microsecond machine cycle. The storage density is 1024 bits per c-r tube for a total high-speed storage capacity of 2048 full words. This is equivalent to approximately 20,480 decimal digits.

An Electrostatic Storage Unit consists of thirty-six pluggable Storage Drawers (see Fig. 8), a Deflection Gate, and special power supplies. Each Storage Drawer contains two 3-inch IBM-85 c-r tubes. This type is an improved version of the IBM-79 described by W. E. Mutter.[6]

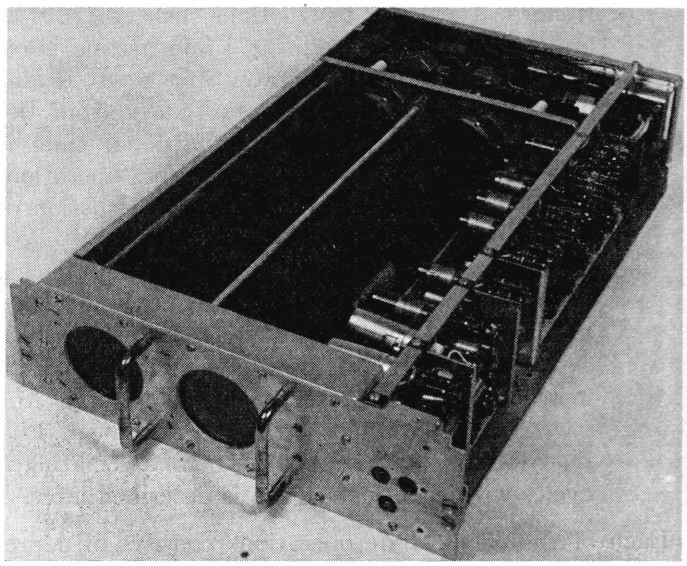


Fig. 8—Storage drawer.

One of the most serious difficulties encountered when using electrostatic storage is that of spill. Briefly, spill is the degeneration of the information stored on the face of a c-r tube as a result of repeated references to information in adjacent storage locations. In the 701 the effect of spill has been greatly reduced by the use of improved c-r tubes and by means of extensive address scattering.

The deflection system is common to all 72 c-r tubes and the selection of information is accomplished by control of the c-r tube beam-unblanking circuits. The assignment of address locations in the deflection system is arranged so that consecutively addressed half words, such as instructions, require the unblanking of a given c-r tube only once for each four instructions. That is, four consecutive address locations progress from left tubes (Storage Drawers S through 17), to left tubes (Storage Drawers 18 through 35), to right tubes (Storage Drawers S through 17), to right tubes (Storage Drawers 18 through 35).

Further scattering of addresses within the 1024 bit raster, as indicated in Fig. 9, provides a total of sixty-four consecutively addressed references before a reference is made to an address located adjacent to one pre-

[5] F. C. Williams, and T. Kilburn, "A storage system for use with binary-digital computing machines," *Proc. IEE (London)*, vol. 95, pp. 183–202; December, 1948.

[6] W. E. Mutter, "Improved cathode-ray tube for application in Williams memory system," *Elec. Eng.*, vol. 71, pp. 352–356; April, 1952.

viously used. For example, address 64 and address zero are adjacent and in the same tubes.

This address scattering so distributes the I-cycle references that it is possible to neglect their contribution to spill. The *E*-cycle references are a direct function of the program. That is, the number of references to a given address and the address location is determined by the programmer; no restrictions have been placed on his decisions. Therefore, in the calculation of the required storage-tube read-around ratio, only the *E*-cycle references must be considered. The most severe loop that is possible by programming makes 342 references to a specific address before all adjacent addresses have been regenerated.

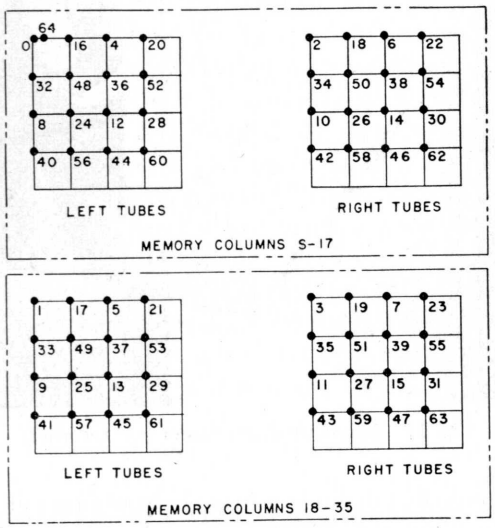


Fig. 9—Conventionalized view of storage tube rasters showing half-word address scattering.

The process of regenerating the Electrostatic Storage in the 701 has been speeded by the employment of two read-out pulses during each 12-microsecond Regeneration cycle. These read-out pulses unblank first the 36 left tubes and then the 36 right tubes. The resulting information is stored in triggers located in each Storage Drawer. At dash time, the time designated for writing, the storage triggers determine whether a dash is to be written or the existing dot is to remain. By this process two full words are regenerated each 12-microsecond Regeneration cycle, and only one video amplifier[7] is required for each two cathode-ray tubes.

The reduction in regeneration time afforded by improved cathode-ray tubes, address scattering and regeneration of two full words per regeneration cycle has greatly enhanced the efficiency of this computer. This reduction in regeneration time has been obtained while maintaining completely satisfactory operation of the Electrostatic Storage. The programmer is therefore not burdened by having to limit his demands on the Electrostatic Storage.

[7] J. C. Logue, A. E. Brennemann, and A. C. Koelsch, "Engineering experience in the design and operation of a large scale electrostatic memory," *IRE Convention Record;* 1953.

Magnetic Tape System

The Magnetic Tape Reader and Recorder, shown in Fig. 10, serves two functions in the 701 system. It may be used as either an intermediate speed, large-capacity storage for programs and data or it may serve as a high-speed input unit for information previously recorded on a reel of magnetic tape.

A Magnetic Tape Reader and Recorder contains two independent magnetic tape units. Each unit contains an 8-inch reel capable of holding up to 1200 feet of one-half inch, oxide-coated, non-metallic tape.

Recording is at a density of 100 bits per linear inch using the non-return-to-zero type of recording.[8] Seven parallel channels are recorded, six of which are information tracks and one is an odd-even check track. One full word is recorded in six groups of six bits per group, and information is transmitted between the Tape Unit and the Calculator in a serial-parallel manner. Since zero in a non-return-to-zero system is indicated by the absence of a pulse, the "one" bit recorded in the check track will insure that the arrival of an information group is indicated.

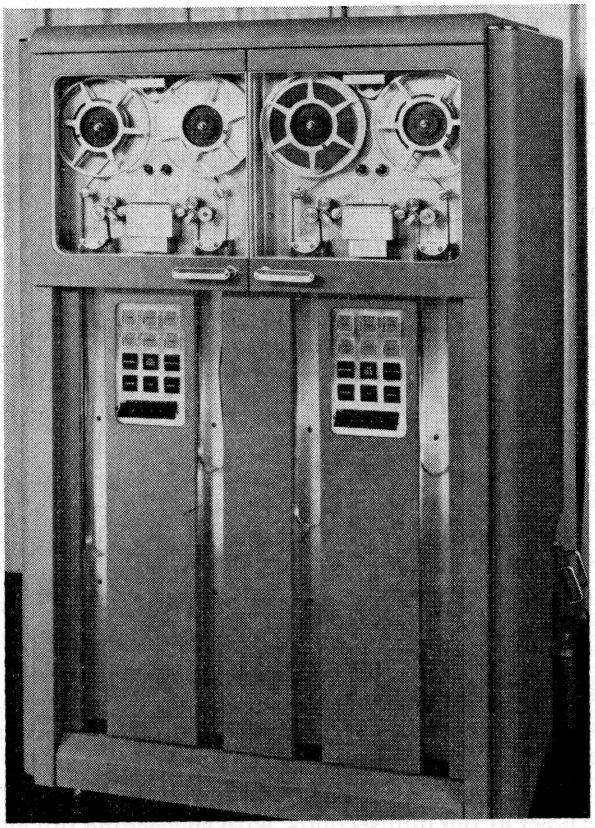


Fig. 10—Magnetic tape reader and recorder.

Two seven-bit registers are provided to serve as a temporary storage for each six-bit information group and the check bit. These registers consist of conventional trigger circuits. Their principal function is to

[8] H. W. Nordyke, "Magnetic tape recording techniques and performance," (Review of input and output equipment used in computing systems)—*Joint AIEE-IRE-ACM Computer Conference*, pp. 90–95; March, 1953.

compensate for time differences in the arrival of recorded bits due to tape skew. Since they have static outputs they also serve the valuable function of charging and discharging the capacitance of the signal cable between the tape unit and the calculator. After a suitable delay the seven signal lines are sampled and their status is read into columns S through 5 of the Multiplier/Quotient Register. The check bit is read into a Check Bit Trigger.

Since the 701 is a 36 bit, parallel machine, and the transmission of information between all other units is on a 36 bit parallel basis, it is desirable to assemble the six 6-bit groups of a full word, received from the Magnetic Tape Units, into this form. The function is performed conveniently by the Multiplier/Quotient Register. This register is inherently a high-speed shifting register; by connecting the output of column S to the input of Column 35, a six bit group is ring-shifted six columns to the left and temporarily rests in the Multiplier/Quotient Register columns 30 through 35.[9]

Successive six-bit information groups are read-in to columns S through 5 and ring-shifted left until six such groups have been received. Recognition that a full word is now located in the Multiplier/Quotient Register will initiate an immediate transfer of this word to the Electrostatic Storage, providing a *Copy* instruction is waiting. This will be discussed below.

During the shifting operation the number of one bits are counted and the result is compared with the status of the Check Bit Trigger. If this check fails a Tape Check light is turned on.

It will be remembered that the Multiplier/Quotient Register functions as a storage register for all information coming from input or output devices. A *Copy* instruction is required to transfer this information between the Multiplier/Quotient Register and the Electrostatic Storage. The twelve bit address part of this instruction specifies the Electrostatic Storage address at which the word is to be stored or read. In general *Copy* is a synchronizing instruction that acts as a buffer agent between the asynchronous input-output units and the synchronous calculator. It is supplied by the program at the rate required by the input-output unit in use.

The process of recording on magnetic tape is the reverse of the reading operation except that the Multiplier/Quotient Register columns 30 through 35 are used as outputs to the Magnetic Tape Unit.

The initial access time to information recorded on tape is set at ten milliseconds, with succeeding full words being available each 800 microseconds. Tape operating speed is 75 inches per second,[10] which makes available to the calculator 1250 full words per second. This is equivalent to approximately 12,500 decimal digits per second.

[9] See footnote 2, p. 1275.

[10] W. S. Buslik, "IBM magnetic tape reader and recorder," (Review of input and output equipment used in computing systems)—*Joint AIEE-IRE-ACM Computer Conference*, pp. 86–90; March, 1953.

Magnetic Drum System

The Magnetic Drum Reader and Recorder, shown in Fig. 11, serves as additional storage for programs or data which exceed the capacity of the Electrostatic Storage. This unit contains two physical drums, which are functionally organized into four logical drums, each capable of storing 2048 full words. The total storage capacity is 8192 full words which are equivalent to approximately 82,000 decimal digits.

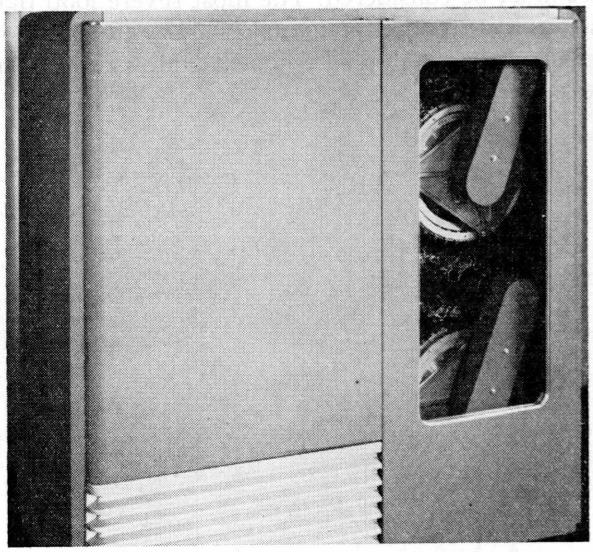


Fig. 11—Magnetic drum reader and recorder.

Each physical drum is a forged aluminum cylinder 13 inches in diameter, which is wound with Cunife (copper, nickel, iron) wire and ground to provide a true surface having the desired magnetic properties. The speed or rotation is 2929 rpm to obtain a surface speed of 2000 inches per second. A logical drum contains 36 information tracks, one index track and one timing track. The center to center spacing between tracks is 1/16 inch.

Each physical drum is enclosed in an aluminum housing containing 76 removable read/record heads which incorporate spring-loaded coil and lamination assemblies. A differential screw arrangement provides a vernier adjustment when setting the head to drum surface clearance.

Discrete pulse type of recording is used at a linear density of 50 bits to the inch and a drum timing track, consisting of 2048 recorded pulses, is used to locate the storage cells around the periphery of the drum.

The addressing system used consists of an eleven stage binary counter and a zero recognition circuit. After selecting one of the four logical drums by means of the instruction, (*Read Drum* or *Write Drum*) a *Set Drum* instruction may be given to specify the address of the first full-word to be read or recorded. Omission of the *Set Drum* instruction starts the reading or recording operation at drum address zero. The address part of this instruction is set into the eleven stage counter in parallel and programming may continue until the speci-

fied full-word is required. At this point in the program a *Copy* instruction must be given which, upon receipt of the index pulse, gates drum timing pulses to the address counter. This counter is connected so that it counts down and a count of zero denotes the arrival at the specified address.

Thirty-six bits of information are read simultaneously and set into a drum register. From this point on the process is similar to the Magnetic Tape operation previously described except that the full-word (36 bits) is set into the Multiplier/Quotient Register in parallel. From there it is stored in the Electrostatic Storage by the *Copy* instruction.

An additional feature of this addressing system is that a second zero recognition circuit is connected to the lower seven stages of the address counter. A count of zero occurs here every 1280 microseconds and it denotes the arrival of the next higher address (The drum addressing system is arranged so that consecutively-addressed storage locations are separated by 127 actual locations). By this means the programmer is provided with sufficient time to allow the machine to calculate the next electrostatic storage address and provide another *Copy* instruction. This type of operation provides a variable record length, by programming, which is limited only by the number of *Copy* instructions given and the capacity of the drum (0 to 2048 full-words).

Since four 36 track logical drums are located in one frame, relay switching is employed so that 36 amplifiers and 36 writing circuits can accommodate the 144 information tracks. The write circuits are switched at their outputs. The amplifiers are switched at the cathode of a preamplifier in the reading circuits.

Fig. 12—Card reader.

Punched Card Reader

Initial input to the calculator is by way of the Card Reader, which is shown in Fig. 12. Seventy-two columns of a punched card are read by brushes a row at a time. One such row of 72 columns constitutes two 36-bit full words, if binary punching is used. Thus a 12 row IBM card can contain 24 binary full words.

Cards are always read as though they contain binary punching and 24 full words, but if the information is punched in any other system, such as the familiar decimal punching, the calculator can be easily programmed to perform the conversion. The 72 columns can be read and converted, if necessary, at the rate of 150 cards per minute.

The signals from the brush circuits are set into Key Triggers. Key Triggers are flip-flop circuits having an integrating input for use with manual key switches and electromechanical devices such as brushes, cam operated contacts, and relay operated contacts.

Selection of the Card Reader is by means of the input/output instruction *Read* (Card Reader address), which sets the card feed in motion. Once selected, the program may continue for a maximum of 50 milliseconds, at which time the first row of the card has reached the reading brush station. One row of punched-hole information is automatically transferred to the 72 Key Triggers which are located in the Analytical Control Unit. Two *Copy* instructions are required to transfer this information, via the Multiplier/Quotient Register, to the Electrostatic Storage. Each group of 36 bits (one full-word) requires a *Copy* instruction to effect the information transfer and to designate the storage address. A period of 540 microseconds is available for programming between the first and second *Copy* instructions, thus permitting the modification of the *Copy* instruction address part. By this means loading may be accomplished by the use of a loop containing one *Copy* instruction and the few instructions required to modify its address part.

The time between information rows on the IBM punched-card allows the programmer complete freedom in programming for 15 milliseconds following the execution of the second *Copy* instruction. If the information being read is punched in a system other than binary, this period of time may be utilized to program the conversion.

A Control Panel is available on the Card Reader to provide facilities for the rearrangement of input data. Selective changes are made possible by the inclusion of selection relays which have multiple transfer contacts.

Printer

The line-at-a-time Printer, Fig. 13, is capable of printing 120 columns of information at a rate of 150 lines per minute. The transfer of information to the Printer is controlled by select (*Read* or *Write* Printer) and *Copy* instructions in a manner similar to that discussed in the section on the Card Reader. One basic difference is that information to be printed is transferred, via the Multiplier/Quotient Register, to 72 thyratrons (2D21). These thyratrons are located in the Analytical Control Unit and are used to energize either the print magnets, when printing, or the punch magnets, when the Card Recorder

is the selected output unit. The Card Recorder is discussed in the next section.

Data to be printed are usually results of calculations which have been stored in the Electrostatic Storage. However, information recorded on a magnetic tape or a magnetic drum may be printed by first transferring it to the Electrostatic Storage Unit. The arrangement of these data in the desired sequence for printing may be performed during the transfer.

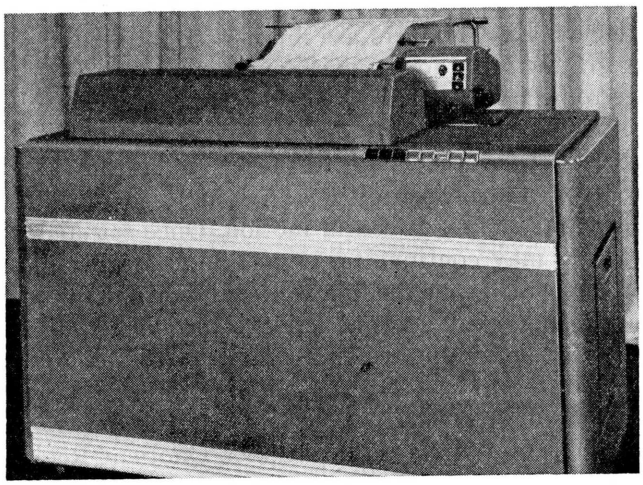


Fig. 13—Printer.

Conversion of the binary data into decimal form, as required by the Printer, is accomplished by programming. In order to reduce the lost time inherent in electromechanical input/output devices, the control circuits permit unrestricted programming for a period of 58 milliseconds after executing the Select Printer operation. This period is sufficient to convert by programming, seven binary full-words to seven 10 decimal digit numbers. At the conclusion of the 58 milliseconds period the program must provide two *Copy* instructions, in close succession, to effect the transfer of the first row (two words) of information to the Printer.

Checking of the actual printing is accomplished through the use of echo pulses which are generated by type-wheel contacts. These pulses indicate the time at which the type-wheel was set into motion, which determines the character to be printed. To perform this checking operation the select instruction *Read* (Printer address) and additional *Copy* instructions must be provided. The additional *Copy* instructions are required to transfer the echo information to the Electrostatic Storage where it may be compared by programming, with the information originally transmitted to the Printer.

Papers forms are handled by a punched-tape controlled carriage. However, all functions of this unit are available to the programmer and can be controlled by the program. Ten *Sense* exit lines from the calculator may be energized by *Sense* instructions to provide carriage control information.

Card Recorder

The Card Recorder, shown in Fig. 14, is a high-speed punch capable of punching information (usually decimal or binary) at the rate of 100 cards per minute. Operation of this output unit is similar to that of the Printer previously described. A *Write* (Card Recorder address) instruction is required to select this unit and set the card feed into motion. Following the execution of the select instruction, 70 milliseconds are available for programming before the first pair of *Copy* instructions are required. This time may be used to perform a binary to decimal conversion or other general programming. Two *Copy* instructions are required to transfer the information to be punched into each card row. Again the time between card rows is made available for unrestricted programming. When using the Card Recorder, this time is 31 milliseconds.

A maximum of 72 columns may be punched from the calculator exit busses. If desired, the remaining 8 columns may be used to gang-punch indicative information such as program code number.

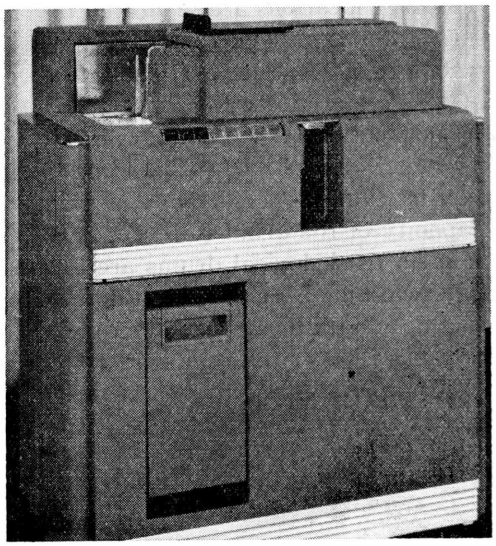


Fig. 14—Card recorder.

Operator's Panel

The Operator's Panel is located at the Analytical Control Unit of the calculator and is shown in Fig. 15. Considerable attention has been given to the selection of controls and indicators and their arrangement on the panel. Simplicity has been stressed. Yet versatile and powerful control over the calculator is at all times possible.

Facilities are available for the visual inspection of any full word stored in the Electrostatic Storage by use of the Memory Register neon lamps. The panel also contains neon lamps for the purpose of automatically displaying the contents of the Accumulator Register, Multiplier/Quotient Register, Instruction Register and Instruction Counter. In addition, lights are provided to indicate the status of calculator controls which are of interest to the operator.

Buttons and key switches are available for manual entry of data or instructions, execution of any instruction, slow-speed, step-wise operation, and program alter-

ation. Power *On* and Power *Off* Controls are also located here for the operator. In addition to the above, a Machine Cycle key switch and *DC ON* and *DC OFF* controls are available for the Calculator Engineer.

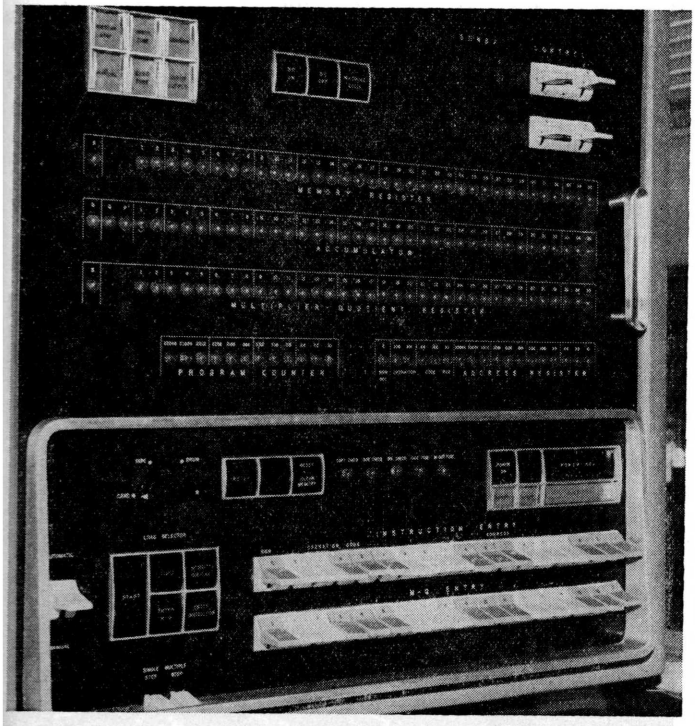


Fig. 15—Operator's panel.

Power Supply and Distribution Unit

The Power Supply for the Type 701 was designed and built to IBM specifications by an outside source. It operates from a 208-volt, 3-phase, 4-wire, Y-connected, 60-cycle power source. The calculator is rated at 88kva.

The direct voltages used throughout the calculator are as follows:

+500 volts	−30 volts
+220 volts	−100 volts
+150 volts	−250 volts
+15 volts	

The Power Distribution Unit, shown in Fig. 16, houses control relays, contactors, circuit breakers, fuses, timers, and a power control panel. In general it contains the equipment for the control and distribution of ac and dc power throughout the calculator.

All direct voltages are monitored at the Power Distribution Unit by ±5 per cent Sensitrols. These voltage monitoring devices protect the calculator components such as tubes, diodes, and resistors. This protection is accomplished by turning off all direct voltages in the event of a failure in one of the power supplies. A switch is provided on the Control Panel for each Sensitrol. These switches may be used to disable individual Sensitrols so that a particular direct voltage may be varied in excess of ±5 per cent for the purpose of marginal checking.

Fuses are all located in the Distribution Unit and are of the spring-plunger type. When a fuse is blown it will turn off all alternating or direct voltages, depending upon the circuit in which the fuse is located. The decision to place all fuses in the Power Distribution Unit was made for the protection of the inter-frame ac and dc service voltage cables. That is, all service voltages are fused at the distribution point rather than at the load. A secondary benefit is the convenience of having only one place to look for a blown fuse.

Voltages are sequenced on in five voltage groups for protection of the vacuum tubes and to reduce the line surge. A fast-discharge circuit is used to bring all voltages down simultaneously and the discharge time constant of each supply is adjusted to maintain the correct voltage ratio.

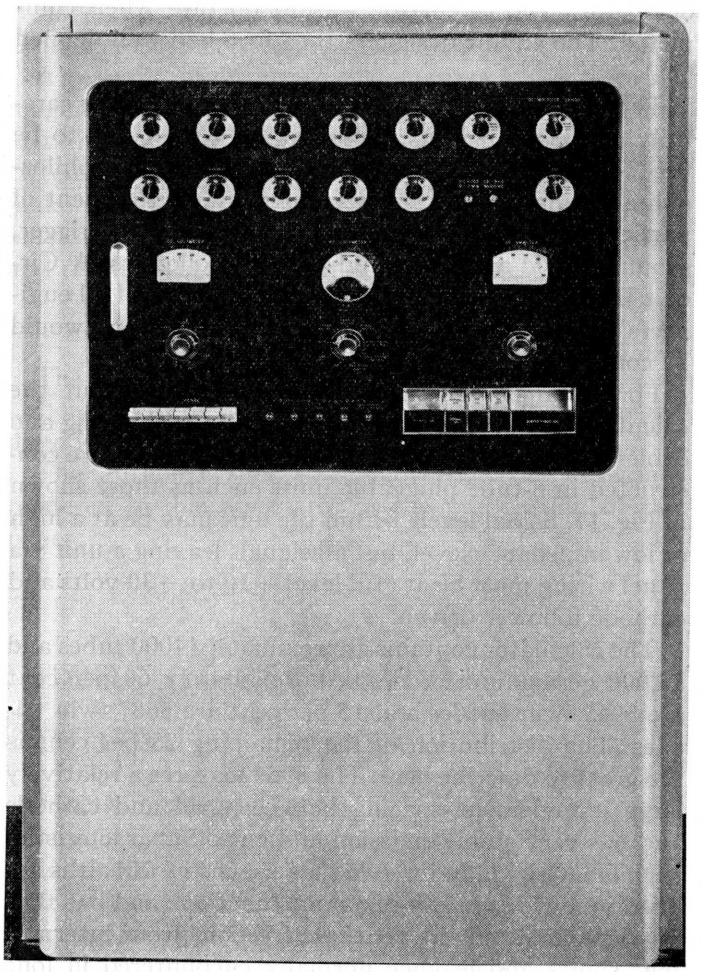


Fig. 16—Power distribution unit.

Construction and Design Details

The calculator was designed and built on a unitized basis to facilitate the production, testing, shipping and installation operations. The dimensions of all units have been restricted so that every unit can be moved easily through conventional doorways and elevators and can be transported with normal trucking or aircraft facilities.

The unitized construction was also extremely helpful during the initial testing phase of the project. Individual

test units were constructed for the Electrostatic Storage Unit, the Magnetic Tape Units, and the Drum Unit. A combined test unit was also constructed for the Card Reader, the Card Recorder and the Printer. These test units enabled engineering groups to work independently on all units. This speeded the testing operation and further assured a maximum degree of success when the units were initially coupled as a calculator.

The design of the calculator falls more or less naturally into two broad areas, the system planning and the engineering design, although obviously each was modified by the other. Of particular value to the circuit design engineers was an Operators Manual outlining the logical system. This manual was completed before circuit work was begin. In other words, before circuit work was started a manual was prepared describing in detail how the calculator was to operate when completed. The circuit designers therefore had well defined goals.

The engineering design was started by selecting carefully considered service voltages and tube types to be used, as well as developing a standard circuit philosophy. The latter was supplemented by development of particular circuit elements such as a high-speed trigger, grounded grid amplifiers and cathode followers. A Circuit Design Manual was prepared for the use of all engineers doing design so that all circuits developed would be compatible.

For instance, the signal level used throughout the calculator is +10 to −30 volts and all panel wiring and cables are driven by cathode followers. Circuits are constructed in 8-tube pluggable units such as those shown in Fig. 17. Signal levels within the unit may be at a high or low impedance level, but all signals leaving a unit via panel wiring must be at grid level +10 to −30 volts and cathode follower driven.

The calculator contains approximately 4000 tubes and 13,000 germanium diodes. Approximately 60 per cent are 5965 twin triodes and 15 per cent are 5687 twin triodes. The distribution of the remaining 25 per cent is too great to describe here. The 5965 features a relatively large interelement spacing between grid and cathode and has very stable emission and cut-off characteristics with life. Originally intended for use in the 701, it is applied in a wide variety of circuit functions and has thus far demonstrated the required freedom from intermittent shorts and leakage normally encountered in long life applications of this type. On the basis of defective tubes analyzed from operations to date (5000 hours) a failure rate of 0.3 per cent per 1000 hours has been established. This figure is expected to materially reduce as additional machines increase the population under study.

Two classes of germanium diodes are used in the 701. Class A diodes, those having a back resistance greater than 400,000 ohms at 50 volts, are the primary switching circuit components. Class B diodes, those having a back resistance greater than 200,000 ohms at 50 volts, are used as protective clamping on the signal lines throughout the calculator. They are also used in switching circuits where the lower back resistance is satisfactory.

Protection is provided for the many diodes used in switching functions by means of +15 and −30 volt lines throughout the machine. These protective voltage lines are connected, through Class B clamping diodes, so as to limit the upper and lower voltage level of all cathode follower outputs. Since cathode followers are the only driving source for diode switching circuits, a maximum back voltage no greater than approximately 45 volts will ever exist across switching diodes should any combination of the dc power supplies fail.

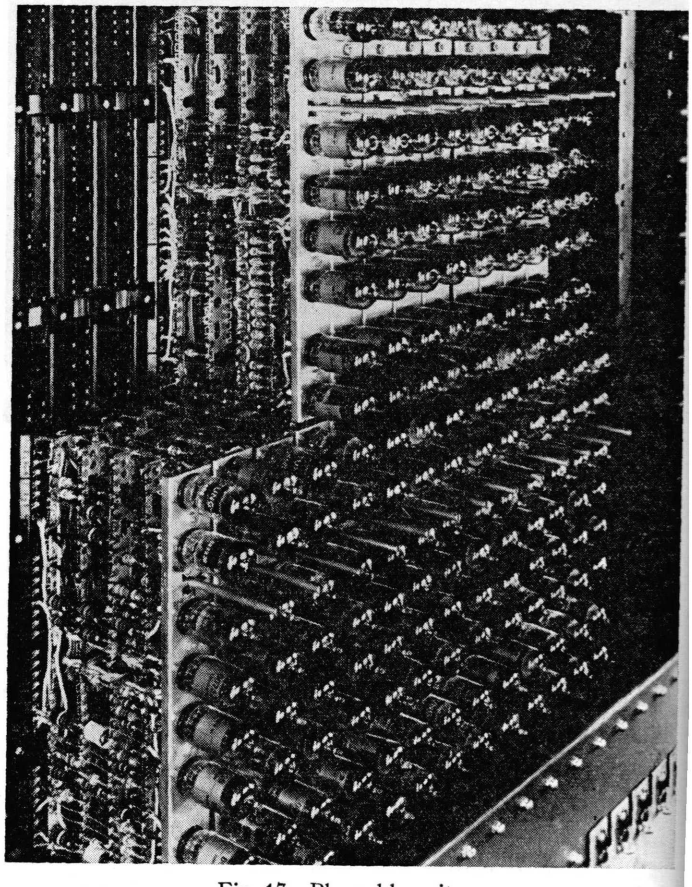


Fig. 17—Pluggable units.

This clamping action is required for only a short period of time since automatic voltage sensing through the use of plunger-type fuses and Sensitrols will immediately turn off all dc power. By means of this protection there has never been a loss of large quantities of switching diodes, as a result of their exposure to excessive back voltage.

All inter-frame cabling is laid in ducts beneath the floor. Signal cables are terminated in pluggable connectors and dc service cables are connected to screw-type terminal strips.

The machine is cooled by means of blowers located at the bottom of each gate or frame. The input air to these blowers must be maintained below 85°F by the use of air conditioning.

Machine Maintenance

The problem of maintaining a large-scale electronic calculator is one worthy of extensive planning. It is felt, by the engineers associated with this project, that regularly-scheduled preventive maintenance is the best approach to the problem at this time.

To facilitate the location of incipient troubles, the 701 has the following features:

(a) Direct coupling is generally used throughout, thus permitting static checking of signals.
(b) All dc power supplies may be varied approximately ±10 per cent for the purpose of marginal checking.
(c) The basic pulse repetition frequency (normally 1 mc) may be varied from 600 kc to 1.2 mc.
(d) The Machine Cycle key switch located on the Operator's Panel permits the execution of an instruction, one machine cycle (12-microsecond period) at a time. For example, the instruction *Multiply* would require 38 depressions of the Machine Cycle key switch and in doing so would permit analysis of the partial product as it is being developed.
(e) Facilities are also available to permit the high-speed repetitive execution of any instruction. This type of operation is extremely important when a timing problem or an intermittent condition exists.

In addition to the above, a large number of diagnostic programs[11] have been prepared which may be used to test thoroughly the performance of selective portions of the 701 system. Errors that occur during these tests are recognized by means of the program and reported to the engineer by means of lights, punched cards or printed results. Sufficient data may be stored and printed at a high rate of speed to permit an easy analysis of the difficulty.

A complete set of manuals (16) have been prepared covering the installation, operation, programming and maintenance of the calculator.

[11] L. R. Walters, "Diagnostic programming techniques for the IBM type 701 electronic data-processing machine," *IRE Convention Record;* 1953.

Conclusions

The IBM 701 is a large-scale, high-speed, electronic digital computer that is adaptable to a wide variety of problems. It incorporates the latest development in three principal types of storage and provides sufficient storage capacity to accommodate the many complex problems existing today.

A versatile input-output system using magnetic tapes, magnetic drums, the proven punched card and a line-at-a-time Printer, provides rapid communication between the machine and the operator. These features make possible the efficient use of a high-speed computer on problems containing large quantities of input data and resulting in many pages of printed results.

Acknowledgments

The Type 701 was developed at the IBM Engineering Laboratory, Poughkeepsie, N. Y. by groups under the direction of N. Rochester and J. A. Haddad.

The author wishes to acknowledge the assistance rendered by his associates in the preparation of this paper.

The Arithmetic Element of the IBM Type 701 Computer*

HAROLD D. ROSS, Jr.†, Associate Member, IRE

Summary—**The arithmetic element of the IBM Type 701 Computer contains a number of innovations, both in circuits and in logic, which distinguish it from that used in other high-speed, parallel, binary computers. The most significant of these is the storage element used in the arithmetic registers, in place of the more usual flip-flop or trigger circuit. Use of this new element enables a number of simplifications to be introduced in the methods of executing instructions.**

Introduction

A NUMBER OF large-scale, parallel, binary digital computers are now in operation and the circuits and logic used in their arithmetic units include a variety of techniques. Most of these machines make use of the flip-flop or trigger circuit as the basic storage component in arithmetic registers and at least one uses a re-circulating pulse type of storage device.

A still different type of storage is employed in the arithmetic element of the IBM Type 701 Electronic Data Processing Machines and Associated Equipment, the large scale, high-speed, electronic computer installations now in production by the International Business Machines Corporation.[1,2] This dynamic type circuit has the property that its output level during each one-microsecond interval is the same as the input level during

* Decimal classification: 621.375.2. Original manuscript received by the Institute, June 23, 1953; revised manuscript received July 23, 1953.

† International Business Machines Corp., Engineering Lab., Poughkeepsie, N. Y.

[1] C. E. Frizzell, "Engineering description of the IBM type 701 computer," Proc. I.R.E., pp. 1275–1287, this issue.
[2] W. Buchholz, "System design features of the IBM type 701 computer," Proc. I.R.E., pp. 1262–1275, this issue.

the previous one-microsecond period; that is, there is a one-microsecond delay through the unit. The output bears a marked resemblance to that of a flip-flop in that it is essentially a dc signal at one of two different levels. This circuit is used in conjunction with dc-coupled diode switching in order to achieve a number of simplifications in arithmetic processes.

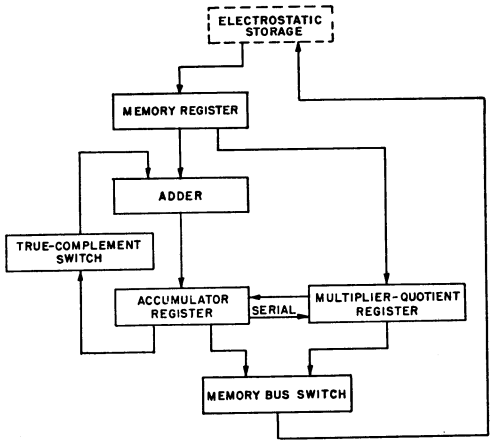


Fig. 1—Type 701 arithmetic element.

Arithmetic Element Block Diagram

A simplified block diagram of the arithmetic element appears in Fig. 1. In this diagram the major components are shown, with the paths of information flow during execution of an operation. The characteristics of the several units which make up the arithmetic element are:

1. Memory register. In executing an instruction requiring a word from memory, the word first appears in this register. It is not capable of shifting.
2. Multiplier-quotient register. This register may be entered from the memory register directly, or from the accumulator register by shifting. On input-output operation it acts as a buffer register. It is capable of shifting to the left or right. It holds the multiplier during multiplication and receives the quotient during division.
3. Accumulator register. In such operations as addition, this register holds one of the two operands and is connected to the adder by true-complement switching. It is capable of shifting to the right or left, either alone or in conjunction with the multiplier-quotient register.
4. Adder. This unit is mainly a switching array in which the outputs of the accumulator and memory registers are added together and the sum fed back to the accumulator register. It is described more fully in a later section.
5. Memory bus switching. This unit contains the switches necessary to connect the output of either the accumulator or multiplier-quotient registers to the electrostatic storage unit in order to store a word. Either the entire contents or only the more significant half of each register may be stored.
6. True-complement switching. This consists of an inverter and switching for each column of the accumulator register to permit either the true or inverted accumulator register output, or neither, to be connected to the adder unit.

Microsecond Delay Unit

The three arithmetic registers make use of a new storage unit called the microsecond delay unit. This circuit stores binary information in a dynamic fashion but its output appears as substantially a dc level of either +10 or −30 v, depending upon the binary digit stored. When suitable input switching is provided, information may be entered from one of several sources, such as another register or from the adjacent columns of the same register. When new information is not being entered, the status of the unit is maintained by an external feedback connection from its output to its input. As used in the 701, the output does not assume the new level until one microsecond after the input signal is applied.

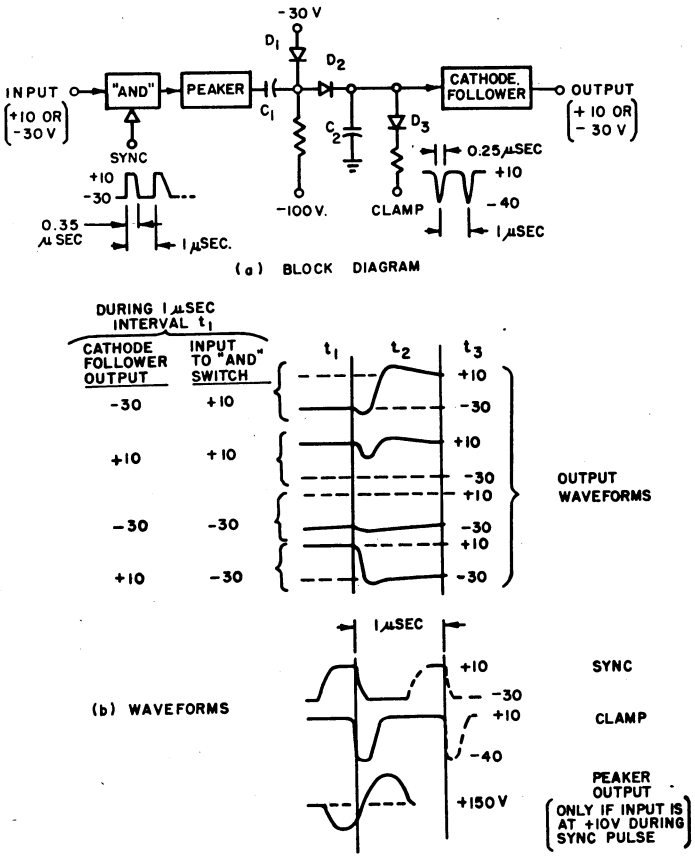


Fig. 2—Microsecond delay unit.

A simplified diagram illustrating the operation of this circuit is shown in Fig. 2a (a schematic diagram of the delay unit is given in Fig. 12). To facilitate understanding the operation of the circuit it is divided into four parts which are (1) an "and" switch in which the level of the input signal, either +10 or −30 v, is sampled by a sync pulse of about 0.35 μsec duration; (2) a vacuum tube peaker circuit driven by the "and" switch; (3) a

capacitor (and associated circuits) which stores the pulse energy from the peaker circuit for one μsec; (4) an output cathode follower driven by the capacitor.

Circuit Operation

In describing the operation of this circuit, there are four possible conditions to consider. These possibilities are listed in Fig. 2b. It should be pointed out here that the output continues unchanged until one μsec after the new input is applied, hence the table of Fig. 2b shows the various combinations of input and output that may be present simultaneously. Considering the first of these conditions, the input signal during t_1 is at +10 v and is applied to the diode "and" switch together with the sync pulse. Under these conditions, the sync pulse is gated through to the peaker tube which will conduct for the duration of the pulse. At this time a negative pulse will be coupled over to the junction of C_1, D_1 and D_2 but will go no farther because D_1 will conduct, holding this point to −30 volts. After the peaker tube is cut off, a positive pulse will be produced, of about 0.5 μsec duration, which will be coupled to the storage capacitor C_2 via C_1, D_1, and D_2. Since the output of the unit was assumed to be −30 v, this was the charge on C_2 (consideration of how this is done will be postponed momentarily). The positive pulse from the peaker charges C_2 to +10 v. The upper level is limited by conduction of D_3 when the normal +10 v upper level of the clamp signal line is exceeded. The voltage across C_2 is directly coupled to the output cathode follower stage. Fig. 2b shows the waveforms mentioned above.

The process just described began with the output of the unit being a −30 v level; consider now the process when the output is initially +10 v. An input level of +10 v will again gate the sync pulse through to the peaker tube, causing it to conduct and generate a negative pulse followed by a positive one. Once again the latter pulse proceeds via C_1, D_1 and D_2 to charge C_2 to +10 v. In the present case C_2 was already charged to +10 (except for leakage through the back resistance of D_2). The +10 v charge is renewed and the output simply continues at +10 v with this reservation: occurrence of the clamp pulse causes partial discharge of C_2, coming as it does during the initial portion of the positive peaker output pulse. Since the peaker output is of longer duration than the clamp, in the end it prevails and charges C_2 to +10 v as shown in Fig. 2b.

Both of the above descriptions assumed an input signal level of +10 v at the time the sync pulse occurred. Consider now the action if the input level is −30 v; coincidence will not occur in the "and" switch and the peaker tube will not conduct, consequently no charging pulse will be generated. If the charge on C_2 had previously been −30 v, it might have leaked off to some extent, through the back resistance of D_3 to the +10 upper level of the "clamp" signal. Fall of the clamp to −40 v will restore the charge. If, on the other hand, C_2 had previously been charged to +10 v by the process described, and no positive charging pulse is generated by the peaker, the clamp pulse will return the charge on C_2 to −30 v. These processes are illustrated by the last two output wave forms of Fig. 2b.

The delay feature is thus due partly to the action of the peaker circuit and partly to the storage capacitor, C_2. The usefulness of this delay is that the 0.6 μsec period while the sync pulse is at −30 v may be used to switch the input of the unit between several sources, one of which is ordinarily the output of the delay unit itself. When the unit is so connected, it maintains its dc signal level at either +10 or −30 v until this connection is interrupted and a new source connected to the unit. This connection is made for one μsec, following which the delay unit input is connected back to its output in order to retain the new information. The only requirement in switching the delay unit input is that the new input signal must have reached its final dc level by the time the sync pulse appears.

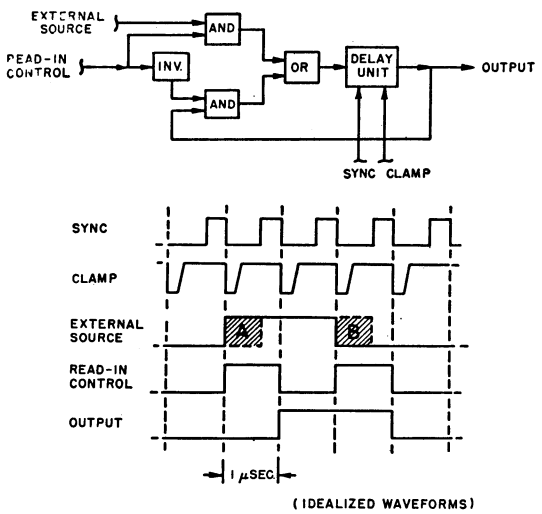


Fig. 3—Delay unit input switching.

Input Switching Associated with the Delay Unit

Fig. 3 shows a simple block diagram of a delay unit with the switching required to maintain continuously a +10 or −30 v level or to enter new information from an external source. Normally the "read-in" control line is at −30 v. When it is desired to enter new information into the delay unit this line is raised to +10 v for one μsec. Since the read-in line is normally at −30 v, the inverter output is normally +10 v, the lower "and" switch is conditioned and the delay unit output is connected to its input. Pulsing the read-in line for one μsec conditions the upper "and" switch and de-conditions the lower, effectively breaking the connection from the delay unit output. As the result of this one-microsecond "look" at the external source, the unit assumes the status of that source and at the end of the read-in signal, goes back to reading its own output. The occurrence of the read-in pulse is suitably timed with respect to the sync and clamp pulses as shown in Fig. 3. In this figure, the delay unit initially stored a −30 v level

(corresponding to a binary zero). The first read-in pulse caused the unit to store a +10 v level (corresponding to a one). Two μsec later a second read-in pulse occurred. At this time the external source presented a −30 v level and hence a zero was stored by the delay unit.

Another function performed by the action of the delay unit is that of re-timing; this is indicated by the shaded areas of the external source waveforms in Fig. 3. The shaded area *A* indicates that the rise of the waveform may occur anywhere within this interval and still give the output change at the time shown. Similarly, area *B* indicates that the fall of the waveform may take place anywhere within this interval and yet give correct operation. The re-timing is due to the fact that the input waveform is sampled by the sync pulse and effectively compensates for the effect of small accumulated delays.

Delay Unit Shifting Registers

If the "external source" for the circuit shown in Fig. 3 was the output of the adjacent delay unit of a register, a shifting register would result. A simple shifting register or ring, having no input switching, is shown in Fig. 4 with actual waveforms at several points. A pattern of two ones and a zero is shown, with the displacement of the pattern at successive stages by one microsecond, being evident.

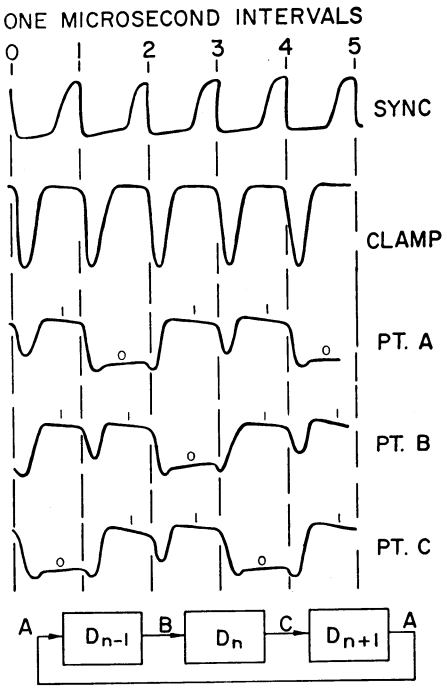


Fig. 4—Waveforms in a delay unit ring.

Fig. 5 shows three columns of a register, capable of shifting to the right or left as well as holding information statically. Each column consists of a delay unit with a three-way switch enabling it to read the output of the unit to the left, read its own output or that of the unit to the right. The function to be performed depends on the condition of the "shift right" and "shift left" control lines at the bottom. Normally these lines are at −30 v, de-conditioning the left and right "and" switch in each column. However, the center "and" switch is conditioned by way of the two-input "or" switch and inverter at the lower left. The output of this circuit is called the "hold" control line, since it conditions the center "and" switches which cause each delay unit to "hold" their status for an indefinite period. Shifting to the left by one column occurs if the "shift left" control line is pulsed to +10 v for one μsec at the proper time with respect to the sync and clamp pulses, in the same manner as the read-in control in Fig. 3. Pulsing either shift control line will cause the "hold" control line to go negative for the same period. A two μsec pulse will cause a shift of two places, and so on.

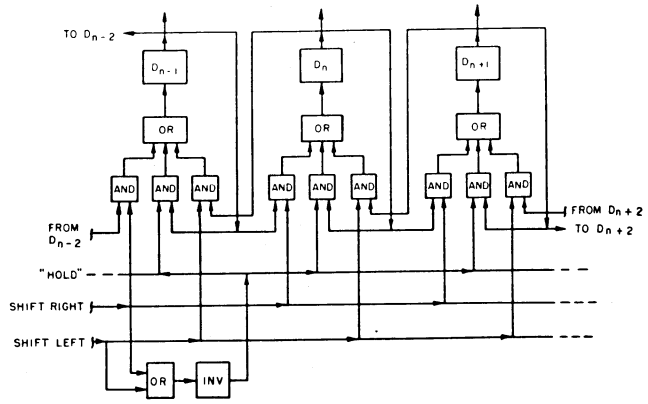


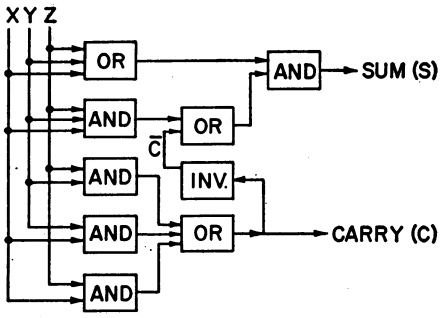
Fig. 5—Shifting register switching.

The circuit shown in Fig. 5, with up to six different inputs per column, forms the basis of the arithmetic registers used in the 701. It is felt that a high order of reliability, logical flexibility and freedom from close timing conditions has resulted from its use. Some of these points will be discussed further in the sections to follow.

Parallel Binary Adder

The means for performing addition in the 701 makes use of logical adders which are passive diode switching circuits of the type shown in Fig. 6. Also shown there is the "truth table" for this circuit. In both the circuit and the table *X*, *Y*, and *Z* are the three inputs, representing one digit each of the addend and augend, together with the carry to the next more significant column. The outputs are of course the sum and the carry from the next less significant column. From examination of the truth table it is evident that a carry is generated whenever there are signals present on two or more inputs and this fact is set forth in Boolean notation by the first logical equation. Further examination shows that a sum of one will be produced under two conditions: if there are signals present at all three inputs and if there is a signal present at one and only one input. This second condition may be re-phrased as fol-

lows: a sum of one will occur if there is a signal present at at least one of the inputs provided there is no carry output generated. This will also be evident from examination of the table. This last condition is exemplified by second Boolean equation in Fig. 6. Having these expressions for sum and carry, circuit follows at once.



(a) LOGICAL ADDER

X	Y	Z	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$C = XY + YZ + XZ$

$S = (X+Y+Z)(XYZ+\bar{C})$

(b) "TRUTH TABLE"

Fig. 6—Three-input binary adder.

It should be mentioned that the circuit as shown in Fig. 6 contains logical elements only; cathode followers required to drive the diode circuits have been omitted. As a result of including the necessary cathode followers a loss in signal amplitude is encountered resulting in a rise in the lower level of the carry signal of about three volts per follower. It thereby becomes necessary to introduce means for restoration of signal amplitudes and these means, which involve over-driven stages, introduce delays into the circuit operation.

Inasmuch as the 701 performs addition in parallel upon 37 columns (35 bits plus two accumulator overflow positions) the delay mentioned above accumulates so that for a full length carry about 3.5 μsec are required. Actually about five μsec are allowed for this process. Since facilities for addition, but not subtraction, are built into the computer, provision must be made to complement one of the operands in order to carry out the latter operation. Complementing is performed on but one of the two factors, namely the one from the accumulator register and a simple diode switch and inverter are required in each column to connect the adder to the accumulator register contents either as a true or a complement number.

Logical Aspects of the Operation "ADD"

In the 701 all numbers are represented as magnitudes with signs. Furthermore, means are provided to obtain the complement of only the accumulator register contents in subtraction, it having been determined that the occurrence of zeros of either sign was not only acceptable but desirable. Fig. 7 contains examples to illustrate the rules by which the form of the result of an arithmetic subtraction is determined. In case 1, the ones' complement of a positive number in the accumulator register is added to a larger negative number in the memory register. An end-around carry results, indicating that the result is in true form and that the sign of the accumulator register must change. In case 2, the magnitudes are reversed and the absence of an end carry indicates a result in ones' complement form and that the accumulator register sign will not change. In order to put the result into true form, a re-complementing process is undergone. Cases 3 and 4 are similar except that, since the magnitudes of the factors are alike, the result will be zero in each case. Here, the sign of the zero result is the initial sign of the accumulator register.

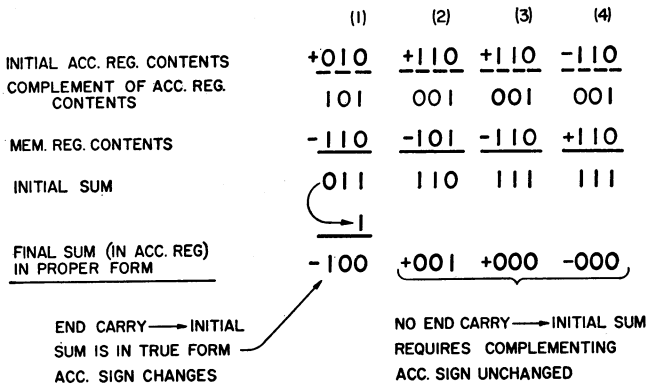


Fig. 7—Examples of ones' complement arithmetic when only accumulator operand can be complemented.

The ones' complement system, while causing a certain inconvenience in having to deal with end-around carries, makes the re-complementing process less painful, in that no carries are produced. The twos' complement system would have required allotting time for possible lengthy carries in re-complementing, as illustrated in Fig. 8. In other words, re-complementing in the ones'

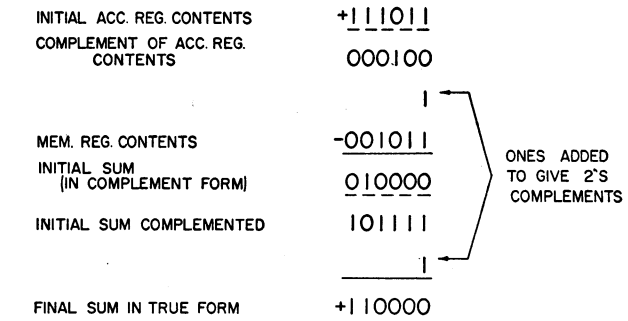


Fig. 8—Example showing occurrence of long carry in changing a twos' complement sum to a true number.

complement system consists simply of inversion, a process requiring little time, whereas in the twos' complement system a complete addition is involved. The table in Fig. 9 summarizes the situations that may arise in adding two numbers of varying sign and magnitude.

SIGN OF FACTOR IN ACCUMULATOR	SIGN OF FACTOR IN MEMORY	RELATIVE MAGNITUDES	FORM OF INITIAL SUM	IS RECOMPLEMENTING REQUIRED?	SIGN OF FINAL, TRUE SUM
+	+	——	TRUE	——	+
−	−	——	TRUE	——	−
−	+	$\|M\| > \|A\|$	TRUE	NO	+
−	+	$\|M\| < \|A\|$	COMPL.	YES	−
−	+	$\|M\| = \|A\|$	COMPL.	YES	−
+	−	$\|M\| > \|A\|$	TRUE	NO	−
+	−	$\|M\| < \|A\|$	COMPL.	YES	+
+	−	$\|M\| = \|A\|$	COMPL.	YES	+

Fig. 9—Logic of sum generation and addition.

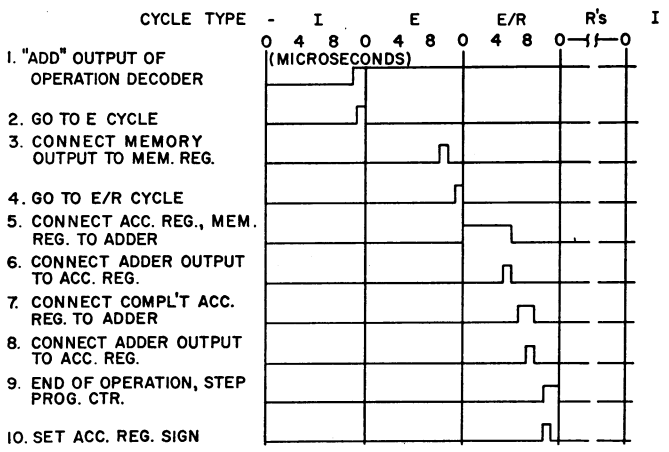


Fig. 10—Timing for the operation "ADD."

Control and Timing of the Operation "ADD"

The timing of the various commands necessary for the execution of the operation ADD is illustrated in Fig. 10. Execution of any operation begins when the output of the operation decoder assumes its new value, near the end of an "*I*" (Instruction) cycle. In this particular case, the output was the one signifying the operation ADD, hence an "*E*" (Execution) cycle is desired next. An *E* cycle is required whenever a reference to memory is needed, either to store a number or to withdraw one from memory. After nine microseconds, the word from memory is available to be entered into the memory register. There being insufficient time remaining in the cycle for the commands yet to come, an "*E/R*" (Execution/Regeneration) cycle is next called for. In this cycle, two full words of the electrostatic memory are regenerated while the actual addition is done. Addition begins by connecting the memory register to the adder and the accumulator register to the adder in true or complement form as required. After five μsec, the output of the adder is sampled and set into the accumulator register in place of the operand originally there. Depending on the relative magnitudes and signs of the operands, the result at this point may be in complement form as described in the preceding section. If so, the complement of the new accumulator register contents is connected to the adder for two μsec; the other adder input is left disconnected and represents a row of 35 zeros. After one μsec the new adder output is sampled and the result, now in true form, returned to the accumulator register. Finally, the sign of the accumulator register is set, according to the table in Fig. 9, and a signal sent out indicating the execution of the operation is complete. This last pulse also advanced the program counter one step in preparation for obtaining the next instruction. Following the *E/R* cycle, two "*R*" (regeneration only) cycles may be called for to satisfy the regeneration requirements of electrostatic memory.

For different operations, the sequence of cycles will vary, depending upon the need for a reference to memory, and upon the amount of time needed for manipulation of words by adding, shifting, and so on. The arithmetic speed is limited by timing requirements of electrostatic storage rather than by arithmetic circuits themselves. Advantage was taken of this where possible to avoid using faster, more elaborate circuits.

The Control of Shifting

The number of places by which the contents of the accumulator register or the combined accumulator and multiplier quotient registers are to be shifted, is controlled by a counter into which the address part of the shift instruction is entered. As shifting proceeds, pulses are sent to this counter, causing it to count down in synchronism with the shifting process. When the contents of the counter reach zero, this fact is sensed by an "and" switch and the shift gate to the registers involved is terminated. Shifting takes place at a one megacycle rate and up to eight places may be traversed within a 12 μsec cycle. A variable number of cycles is required for the execution of a shift depending upon the length of the shift. The maximum length of a shift is 255 places; such a shift, while resulting in a string of zeros in the registers affected, facilitates the programming of floating-point computations.

Multiplication

Multiplication in the 701 consists of repeated addition and shifting, the addition being under control of the currently least significant multiplier digit. Thirty-six successive 12-μsec *E/R* cycles are required for the 35×35 multiplication, following the *I* and *E* cycles used to obtain the instruction itself and the multiplicand. During each *E/R* cycle one addition of the multiplicand may occur together with a shift of the accumulator and *MQ* registers to the right. These functions require but little more than half the 12 μsec allotted; the actual time taken for multiplication might have been reduced by packing the successive additions closer together but in practice as the additional time is used to regenerate the electrostatic storage, the time expended is recouped in that succeeding instructions may be executed in less time. By using the 12-μsec ADD-type *E/R* cycle, control of multiplication is simplified.

Division

One of the consequences of the use of dc coupled diode adders and delay unit registers is that division becomes somewhat simpler than would otherwise be the case. A system which is superior to either the restoring or nonrestoring scheme is used; it is made possible by the ability to examine the result of a subtraction before discarding the original number. Thus, if the result of a subtraction is an overdraw, this result may be ignored (that is, not put into the accumulator register) and the previous remainder retained and shifted left, whereupon another subtraction may be tried. If the subtraction resulted in a remainder of unaltered sign, the result will be in complement form and must be complemented prior to shifting left. In the former case a quotient digit of zero is inserted into the lowest order multiplier-quotient register column prior to shifting and in the latter case a quotient digit of one is inserted. This is repeated in successive cycles, so that the quotient builds up in the *MQ* register on the right as the less significant part of the dividend is shifted out into the accumulator register on the left.

THE DIVISION PROCESS	(DIVISOR) MEM. REG.	(DIVIDEND) ACC. REG.	MQ REG.	
1. DIVISOR & DIVIDEND IN TRUE FORM	+ 10010	+ 00 01100	10111	
2. ACC. REG. COMPLEMENTED		11 10011		
3. DIVISOR		10010		
4. SUM; END CARRY INDICATES NEG. REM. (QUOT. WILL BE LESS THAN ONE)		C→ 00 00101		
5. ACC. REG. CONTENTS AS IN (1)		00 01100		
6. ACC. & MQ SHIFTED LEFT ONE COL.		00 11001	0111_	
7. ACC. REG. COMPLEMENTED		11 00110		
8. DIVISOR		10010		
9. SUM; NO CARRY INDICATES POS. REM.		NC 11 11000		1 QUOT. BIT=1
10. REM. COMPLEMENTED TO GET TRUE NO.		00 00111		
11. ACC. & MQ SHIFTED LEFT ONE COL.		00 01110	1111_	
12. ACC. REG. COMPLEMENTED		11 10001		
13. DIVISOR		10010		
14. SUM; NEG. REM. AS IN (4)		C→ 00 00011		0 QUOT. BIT=0
15. ACC. REG. CONTENTS AS IN (12)		00 01110		
16. ACC. & MQ SHIFTED LEFT ONE COL.		00 11101	1110_	
ETC.				

Fig. 11—The division process.

The process just described is illustrated in Fig. 11. Here a typical division is shown using abbreviated binary numbers of five-bit length (the signs are handled separately). The repeated subtraction cycle is shown for two different cases in steps 6–10 and 11–15. In the first case the result of subtraction was a positive remainder; this fact is indicated by the absence of an end carry which therefore causes the adder output to be inserted back into the accumulator register. In the second case a carry appeared and so the adder output was ignored, the previous remainder shifted to the left, and a new subtraction started, thus it is seen that no restoration is necessary. A further point is illustrated by the example: namely, the method of testing for a quotient less than one. The first four lines illustrate this; the divisor is subtracted from the dividend with no preliminary shift. An end carry in this process indicates an overdraw and consequently a quotient of less than one. If no carry were obtained, an alarm would be generated and calculation stopped. In making this test the end carry is the only result desired, the sum is not put in the accumulator register. Actual division then begins with a shift to the left of the original accumulator register contents together with the *MQ* register.

The complementing cycle mentioned above, which is required when subtraction did not cause a sign change, could have been avoided by use of additional switching but there was little to be gained by the increase in speed, as ADD-type cycles of the sort described previously are made use of here without change in timing. Thus 36 cycles are required for the complete operation, resulting in a surplus of regeneration for the electrostatic memory. This surplus usually enables some regeneration to be dispensed with in the operations immediately following so that a portion of the time spent in dividing is usually made up elsewhere.

Testing for Zero in the Accumulator

The incorporation of the operation "Transfer on Zero" requires a method for detecting whether the accumulator register contains zero. The contents of the accumulator register are to be unchanged as a result of this test and this requirement is easily complied with by the scheme described below. Once again, the use of delay units in the accumulator register simplifies the process.

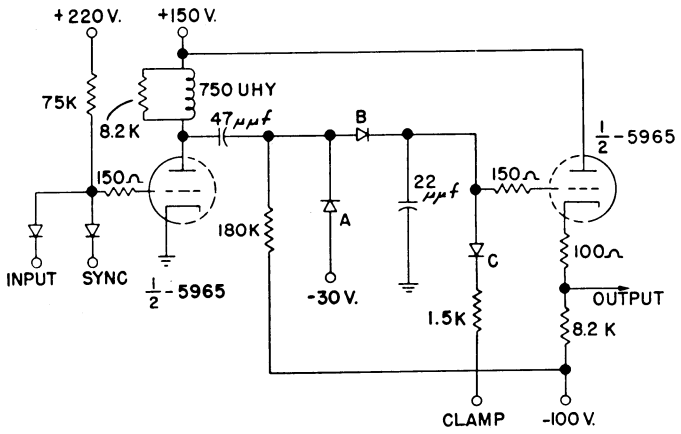


Fig. 12—The delay unit circuit.

The process consists of adding the complement of the accumulator register contents, to zero (obtained by leaving the memory register disconnected from the adder) and adding one in the lowest order. If the contents were initially zero, a full length carry will result, giving a carry from the highest order. This carry is used to cause the next instruction to be taken from the address specified in the Transfer on Zero instruction. The adder output is not set into the accumulator and hence the contents are undisturbed. This ability to examine the adder output without putting it into the accumulator register is facilitated by the one-microsecond delay in the delay units and the use of dc coupled diode adders.

Conclusion

The arithmetic element of the IBM Type 701 Computer makes use of a new electronic storage circuit, the microsecond delay unit, whose output may be either of two dc levels, yet uses dynamic pulse storage techniques. Through the use of this device, in conjunction with dc coupled diode adding circuits, important simplifications are realized in shifting registers, and in the execution of division and testing for zero in the accumulator.

Acknowledgment

The microsecond delay unit described herein was developed by B. L. Havens of the Watson Scientific Computing Laboratory. His support, encouragement, and assistance contributed greatly to the success of the 701 project and his kind permission to describe this unit is greatly appreciated by the author. The author also acknowledges the valuable assistance of other members of the project in the preparation of this paper.