TD 01.01.002.005
September 23, 1960

GPD | technical record

IBM CONFIDENTIAL

DIGITAL COMPUTERS — LOGICAL PRINCIPLES AND OPERATION

by

George J. Saxenmeyer

IBM

General Products Division | Development Laboratory | Endicott, N. Y.

ABSTRACT

A brief survey of digital computation, beginning with the origins of early computing devices — including the abacus, the suan-pan, soroban, choreb, and others — is followed by a description of the theory and operation of Babbage's Differential Analytical Machine. Typical contemporary computing system functions are reviewed, along with data flow charts, binary number systems and adders, Boolean algebra, binary-coded-decimal and bi-quinary arithmetic. Fixed-bit information codes and self-checking and self-correction codes are discussed. A comparison of codes is made, along with various storage devices — including static storage (triggers and latches) — and dynamic storage, including magnetic drum and ferrite core storage. Switch cores and core logic are briefly surveyed.

GPD
Development Laboratory
Engineering Publications Department
Endicott, N. Y.

TABLE OF CONTENTS

ILLUSTRATIONS

FOREWORD

This report is a compendium of course notes used in an Engineering Training Program course presented at IBM Endicott during the latter part of 1959. The material has been compiled for distribution within IBM to serve primarily as a convenient reference source. The author hopes that this document, brief in content but comprehensive in scope, will serve the reader wishing to refresh his memory on some historical or contemporary aspect of digital computation.

I EARLY COMPUTING DEVICES — EVOLUTION OF THE ABACUS

The oldest known device which satisfies the basic definition of a digital computer is the abacus. Its historical origin has not yet been precisely established. In various forms and by various names it has been used since as early as the 6th century B.C. In China since the 12th century A.D. it has been called the suan-pan, with two binary and five quinary beads per position and as many as two dozen digit positions. The Japanese equivalent is the soroban dating back to the 16th century, with just one binary and four quinary beads since the other two are basically redundant.

The Russian version of the abacus is known as the ϵ'choty and contains ten beads per position, the equivalent of two Japanese sorobans laid side-by-side as mirror-images. Available sources do not describe its method of operation. In Turkey this device is called the coulba; in Armenia, the choreb.

Devices similar in technique were the bamboo counting rods used in China until the 19th century, the counting pebbles used by the ancient Egyptians (Herodotus, 450 B.C.), and the "line abacus" used throughout Europe as late as the 18th century. The line abacus was generally a wax-covered table with ruled lines to represent the decimal denominations in bi-quinary form and "X" symbols to take the place of the beads. Modified versions also evolved for such systems as pounds, shillings, and pence.

The suan-pan and soroban are still widely used today in native shops in Asia and in American Chinatowns. An interesting story concerns an American desk calculator manufacturer, whose "queen-of-the-line" machine was decidely bested at a public exhibition in Tokyo soon after World War II. The event was intended to be a promotional stunt to publicize the advantages of the desk calculator, compared to a soroban-equipped Japanese accountant.

In addition to the popularity of the suan-pan and the soroban mental arithmetic was a highly-developed art during the preceding three or four centuries. There are dozens of outstanding "arithmetic geniuses" recorded in mathematical history whose ability to "think with numbers" is amazing even today. Most of them based their abilities on a fantastic memory for odd numerical facts, such as a list of all prime numbers up to a million. It is interesting to note that almost all of them were self-taught and that no common techniques evolved from their experiences.

A. HALVING AND DOUBLING (MULTIPLICATION)

In another vein, a very interesting method of multiplication developed and is still widely used by some of the more primitive tribes in Africa. This method employed the system of "halving-and-doubling," in which by successive binary division of the multiplier and binary multiplication of the multiplicand produces a decimal product. The technique is so simple that with a little practice almost anyone can do it "in his head."

The following is a simple example of the basic process:

Problem: $13 \times 17 = ?$

Solution:

$2\overline{)13}$

$2\overline{)6}$ r. 1 → 17 → 17

$2\overline{)3}$ 34

$2\overline{)1}$ r. 1 → 68 → 68

→ 136 → 136

221 (answer).

II BABBAGE'S DIFFERENTIAL ANALYTICAL ENGINE

Charles Babbage lived in England and did his most significant work during the first half of the 19th century. Like Thomas Edison, James Watt, and other inventive geniuses of that era, he was born long before his time. That is to say the dreams and goals of these people could not satisfactorily be carried out within the bounds of their contemporary technology.

To Babbage belongs the honor of originating the basic principles of calculating equipment we know today as adding machines and desk calculators. But Babbage's most significant contribution of all to the art of digital computing devices was a system concept which never reached fruition during his lifetime. In spite of never having achieved the creative satisfaction of seeing his "brain-child" constructed, Babbage's definitions, organization, and specifications for his "analytical engine" are still today an excellent generic description of what we call "stored-program digital computers." This is true in spite of, rather than because of, the rapid evolutionary progress of mechanical, electrical, and electronic technology during the next century.

Probably the personage who understood Babbage best of all in his own time was a lady mathematician of some reknown, the Countess of Lovelace. The Countess helped Babbage's personal cause to the extent of explaining his work in terms which the other mathemeticians and their patrons and government agencies with available capital to finance it could understand. As a result of this help, Babbage was finally able to achieve a working model of his "difference engine" during his later life. This model is still in operating condition and is on display in a museum. In view of the rudimentary level of technology at that time, in which every part of an assembly had to be fitted by cut-and-try methods, it was manufacturing accomplishment of the highest order.

The following is a brief description of this analytical* engine by a contemporary, L. F. Menabrea, of Turin, Office of the Military Engineers:

> "Those labours which belong to the various branches of the mathematical sciences, although on first consideration they seem to be the exclusive province of intellect, may, nevertheless, be divided into two distinct sections; one of which may be called the mechanical, because it is subjected to precise and invariable laws, that are capable of being expressed by means of the operations of matter; while the other, demanding the intervention of reasoning, belongs more specially to the domain of the understanding. This admitted, we may propose to execute, by means of machinery, the mechanical

*Faster Than Thought, E. V. Bowden, Pitman Publishing Corp. New York, N.Y., 1953.

branch of these labours, reserving for pure intellect that which depends on the reasoning faculties. Thus the rigid exactness of those laws which regulate numerical calculations must frequently have suggested the employment of material instruments, either for executing the whole of such calculations or for abridging them; and thence have arisen several inventions having this object in view, but which have in general but partially attained it. For instance, the much-admired machine of Pascal is now simply an object of curiosity, which, whilst it displays the powerful intellect of its inventor, is yet of little utility in itself. Its powers extended no further than the execution of the four first operations of arithmetic, and indeed were in reality confined to that of the two first, since multiplication and division were the result of a series of additions and subtractions. The chief drawback hitherto on most of such machines is, that they require the continual intervention of a human agent to regulate their movements, and thence arises a source of errors; so that, if their use has not become general for large numerical calculations, it is because they have not in fact resolved the double problem which the question presents, that of correctness in the results, united with economy of time.

"Struck with similar reflections, Mr. Babbage has devoted some years to the realization of a gigantic idea. He proposed to himself nothing less than the construction of a machine capable of executing not merely arithmetical calculations, but even all those of analysis, if their laws are known. The imagination is at first astounded at the idea of such an undertaking; but the more calm reflection we bestow on it, the less impossible does success appear, and it is felt that it may depend on the discovery of some principle so general, that if applied to machinery, the latter may be capable of mechanically translating the operations which may be indicated to it by algebraical notation. The illustrious inventor having been kind enough to communicate to me some of his views on this subject during a visit he made at Turin, I have, with this approbation, thrown together the impressions they have left on my mind. But the reader must not expect to find a description of Mr. Babbage's engine; the comprehension of this would entail studies of much length; and I shall endeavour merely to give an insight into the end proposed, and to develop the principles on which its attainment depends.

"I must first premise that this engine is entirely different from that of which there is a notice in the 'Treatise on the Economy of Machinery,' by the same author. But as the latter gave rise to the idea of the engine in question, I consider it will be a useful preliminary briefly to recall what were Mr. Babbage's first essays, and also the circumstances in which they originated."

A. BACKGROUND OF BABBAGE'S MACHINE

It is well known that in the early part of the nineteenth century the French government, wishing to promote the extension of the decimal system, had ordered the construction of logarithmical and trigonometrical tables of enormous extent. M. de Prony, who had been entrusted with the direction of this undertaking, divided it into three sections, to each of which were appointed a special class of persons. In the first section the formulae were so combined as to render them subservient to the purposes of numerical calculation. In the second section these same formulae were calculated for values of the variable, selected at certain successive distances. Under the third section, comprising about eighty individuals, who were mostly only acquainted with the two first rules of arithmetic, values which were intermediate to those calculated by the second section were interpolated by simple additions and subtractions.

B. THE MATHEMATICS OF BABBAGE'S MACHINE

Since a similar undertaking was begun in England, Babbage decided that the operations performed under the third section might be executed by a machine. This concept was realized by a mechanism which had been partially assembled and to which the name "Difference Engine" was applicable because of the principle upon which its construction is founded. To give some notion of this principle it will suffice to consider the series of whole square numbers, 1, 4, 9, 16, 25, 36, 49, 64, etc. By subtracting each of these from the succeeding one we obtain a new series, which can be termed the "Series of First Differences," consisting of the numbers 3, 5, 7, 9, 11, 15, etc. Subtracting from each of these the preceding one we obtain the Second Differences which are all constant and equal to 2. We may represent this succession of operations, and their results, in the following table:

	A Column of Square Numbers	B First Differences	C Second Differences
	1		
		3	
	4		2b
		5	
a	9		2d
		7	
c	16		2
		9	
	25		2
		11	
	36		

Table 1 - Subtraction Chart of Whole Square Numbers

From the mode in which the two last columns B and C have been formed, it is easy to see that if, for instance, we desire to pass from the number 5 to the succeeding one 7, we must add to the former the constant difference 2. Similarly, if from the square number 9 we would pass to the following one 16, we must add to the former the difference 7, which difference is in other words the preceding difference 5, plus the constant difference 2. Or again, which arrives at the same result to obtain 16 we have only to add together the three numbers 2, 5, 9, placed obliquely in the direction ba. Similarly, we obtain the number 25 by summing up the three numbers placed in the oblique direction dc. Commencing by the addition 2 + 7, we have the first difference 9 consecutively to 7. Adding 16 to the 9 we have the square 25.

We see then that the three numbers 2, 5, 9 being given, the whole series of successive square numbers, and that of their first differences likewise, may be obtained by means of simple additions.

Now, to conceive how these operations may be reproduced by a machine, suppose the latter to have three dials, designated as A, B, C, on each of which are traced, say a thousand divisions, by way of example, over which a needle shall pass. The two dials, C, B, shall have in addition a registering hammer, which is to give a number of strokes equal to that of the divisions indicated by the needle. For each stroke of the registering hammer of the C, the needle B shall advance one division. Similarly, the needle A shall advance one division for every stroke of the registering hammer of the dial B. Such is the general disposition of the mechanism.

This being understood, let us at the beginning of the series of operations we wish to execute, place the needle C on the division 2, the needle B on the division 5, and the needle A on the division 9. Let us allow the hammer of the C to strike; it will strike twice, and at the same time the needle B will pass over two divisions. The latter will then indicate the number 7, which succeeds the number 5 in the column of first differences. If we now permit the hammer of the dial B to strike in its turn, it will strike seven times, during which the needle A will advance seven divisions; these added to the nine already marked by it, will give the number 16, which is the square number consecutive to 9. If we now recommence these operations, beginning with the needle C, which is always to be left on the division 2, we shall perceive that by repeating them indefinitely, we may successively reproduce the series of whole square numbers by means of a very simple mechanism.

C. THEORY OF BABBAGE'S MACHINE

The theorem of the construction of the machine we have just been describing is based on a particular case of the following more general theorem: That if in any polynomial whatever, the highest power of whose variable is m, this same variable be increased by equal degrees; the corresponding values of the polynomial then calculated, and the first, second, third, etc., differences of these be taken (as for the preceding series of squares); the mth differences will all be equal to each other. So that in order to reproduce the series of values of the polynomial by means of machine analogous to the one previously described it is sufficient that there be (m + 1) dials, having the mutual relations we have indicated. As the differences may be either positive or negative, the machine will have a contrivance for either advancing or retrograding each needle, according as the number to be algebraically added may have the sign plus or minus.

A "human computer" working at a desk needs a calculating machine, reference books of tables, pen and paper with which to record the intermediate results in his calculations, and instructions as to how to proceed. But that is not all. Computing is something of an art, and the human computer will be inefficient and may not get very far unless he has some power of discrimination which enables him to interpret his instructions in the light of results which his computations have produced, and if need be, to modify his procedure accordingly. The power of discrimination of which a human operator is capable cannot be exercised automatically by any of the machines which we have discussed so far. Their potentialities are therefore limited.

Babbage clearly understood the restrictions imposed by the inability of a machine to make decisions for itself. He was able to take the next step and to suggest how to endow a machine with the minimum amount of "intelligence" which it needs. As he expressed it, he made the machine "bite its own tail." It is entirely due to these ideas of his that the modern computing machines to which we shall devote the rest of this report are so fast, so flexible, and capable of such an astonishing variety of operations.

D. DATA PROCESSING SYSTEM REQUIREMENTS

If a data processing system is to perform the functions of a human computer, it must possess:

1. An arithmetic unit, capable of performing the normal operations of arithmetic. Babbage called this unit the mill.

2. A memory; that is to say a mechanism which will retain numbers needed in the calculation and also the instructions which will be needed to define successive stages in the computation. Babbage called this part the store. He planned to store 1,000 numbers, each of 50 decimal digits.

3. A built-in power of judgment, which will enable the machine to choose, according to prescribed criteria, the course which the computation has to take.

4. An input-output mechanism which allows the operator to feed numbers and "instructions" into the machine, and to extract from it the results of a calculation.

As we have already stated, Babbage planned to use punched cards for input. He also planned to use them for auxiliary storage; as tables of functions; further, Babbage also proposed using punched cards as one form of output. But realizing as he did the risk of error in copying tables by hand, he proposed making the machine set up its results in type where necessary. Modern machines use both tape and card for input and output and print out their results automatically as well. The sheets so prepared can then be reproduced without error by photo-lithography.

From this account it is evident that Babbage had a thorough understanding of the underlying scientific principles of modern electronic computers. He developed this understanding even though he was hampered by the primitive state of the mechanical devices and power sources of his day, by the non-existance of electronics, and by the experimental status of electromagnetism. This understanding is demonstrated by the operation of his definitions of "store," "mill," and "control" in virtually every modern data processing system. Today these terms are known respectively as "memory," "execution," and "instruction" functions, no matter how such functions are implemented in any given system.

It will become increasingly more evident later in this report that Charles Babbage deserves to be called the "father of digital computers."

III TYPICAL DATA PROCESSING MACHINE FUNCTIONS

The typical modern digital electronic computer or data processing system generally utilizes a number of generalized logical processing functions. Since one of the basic purposes of the system is to perform arithmetic operations on input data and intermediate results some means must be provided to execute the basic arithmetic operations of addition and subtraction, as well as multiplication and division in many applications. Since all other arithmetic operations, such as square root extraction, trigonometric function generation, and the solution of simultaneous equations can be accomplished by a more or less involved series of additions, subtractions, multiplications, and divisions, it is possible within most systems to "build" these so-called "macro-operations" into the operating program of the system.

A. ITERATION

For example, the extraction of the square root of any number can be accomplished readily by what is known as "iteration." Iteration is a repetitive which gradually "improves" a random or estimated starting value for the solution to the actual true answer to any specified degree of accuracy. Normally, an iterative calculation is terminated whenever the results of two successive iterations are equal, within the limits of the specified accuracy of the result. The following example demonstrates a popular method for iterative extraction of the square root:

General equation:

$$A_i + 1 = 1/2 \left(A_i + \left(A_i + \frac{N}{A_i}\right)\right.$$

Where:

A_i = any given approximation (including the starting value).

N = the operand whose square root is to be developed.

An intuitive analysis shows that the actual value of the square root must lie between the "first guess" (A_0), whatever value it happens to be, and the number of times that it is contained within the operand $\left(\frac{N}{A_0}\right)$. This truth holds for all of the successive A_i. It is evident that since the process is really one of averaging the immediate approximation and its "ghost" (which is inevitably on the other side of the true value of the square root) the next approximation will always be closer to the true value than the last one was. So as the iterations continue, the successive approximations gradually appraoch the exact square root. Finally, when the difference between successive values becomes less than the allowable error of the result, the iterations stop and the latest approximation is the answer.

Similar processes exist for all the commonly required functions. Generally these functions are generated as required by the program, since stored tables require expensive "parking space," and searching a lengthy table may require as much time as computing the desired value.

B. STORAGE DEVICES

So far the arithmetic section of a system has been reviewed. Although the arithmetic section is certainly the heart of the system, it is by no means the most essential. Data on which it operates must be made available to the system from within. This requires what is commonly called "storage," or sometimes "memory." Since all electronically-operated information-storage devices with sufficiently high accuracy and reliability to incorporate in a computing system are binary (two-valued) devices, and because the formal symbols which humans are accustomed to (the decimal numbers, the English alphabet, and the punctuation and mathematical operation symbols count up to four or five dozen discretely different marks) it would be very inefficient to use a separate binary position for each symbol. Instead, each character is "coded" in such a form that the entire character set only requires a few binary symbols. The subject of codes will be covered in later sessions on their structure, translation, error characteristics, etc.

Storage devices are usually supplied in the form of the remanent magnetic flux which can be stored either on the surface of a rotating drum or in the toroidal flux-paths of very small metallic cores of one of the manganese ferrites. The common desirable properties of magnetic storage are high operating speed, low driving power, and high "signal-to-noise ratio." The common design factors are:

1. Uniformity of magnetic properties.

2. Addressing (singling out a specific group of positions from the whole memory).

3. An ever-increasing operating speed.

These factors will be reviewed later.

Memories have been built having many millions of positions of binary storage which are in use in commercial-production systems today.

C. CONTROL LOGIC — STORED PROGRAMMING

The third basic functional element in a system is its control logic. Such logic consists of equipment necessary to accomplish the acquisition and execution of data and instructions. All large data processing systems, most intermediate-sized ones, and some small ones, now utilize what is known as "stored programming." This is usually achieved by coding instructions as if they were data, then storing them in memory with data. If instructions are coded numerically, they can be modified during the progress of the program which utilizes them. Such a method of operation provides the program with "learning" or "self-optimization" ability, as well as greatly increased flexibility for operating on ordered arrays of data or changing its own technique on a basis of intermediate results. These features account for the current popularity of stored-program machines.

D. CONTROL LOGIC — WIRED PROGRAMMING

The only commonly-used alternative to stored programming is "wired" programming. In such a system a "plugboard" or "control panel" is provided, containing sequentially-progressive program "exits" which are wired manually to the function "entries" to call out the desired succession of functional operations in sequential order. Usually limited means are also provided for automatic alteration of the program on a basis of intermediate results. However the overall flexibility is inevitably quite elementary compared to stored programming.

Summarizing, arithmetic functions, storage, and controls have been reviewed. These are three essential elements of any digital computer, electronic or otherwise. They date back in technological history to Babbage's "analytical engine," which contained all the logical concepts basic to any of our modern digital computing systems.

E. INPUT/OUTPUT DEVICES

However, there is a fourth essential element which was also admitted by Babbage. This is, input/output — the "arms and legs" of the system by which it communicates bilaterally with the external human world. This is a logical necessity since a computing system is a "machine," and machines are those combinations of functional devices which do useful work. This quality of usefulness is also one of the patentability criteria. There would not really be much point in building a digital computer unless it fulfilled a useful purpose, except for the mental exercise and academic achievement involved. These in themselves seldom justify expending our best engineering and design efforts.

Input/output commonly is provided in the form of punched card readers and punches (since they are the usual vehicle for internal accounting and scientific records), magnetic tape transports (which develop much higher operating speeds, in terms of processing record volume), typewriters, punched paper tape equipment, push-buttons and switches (for manual control-input), indicator lights and audible signals (for output to the operator), and many other special-purpose devices. Most of these devices will be reviewed in detail later in this report.

We have seen how — just as an automobile needs an engine, wheels, a frame, some seats, a steering wheel, some sort of brakes, and a gas tank before it can be called "complete" — a digital computer also has some very basic functional requirements which are just as common to all of them as the essential devices are to the automobile. In the next section we will see how this combination of functions can be symbolized in logical form to permit demonstration of the features of individual systems and comparisons between them.

IV FIRST-LEVEL DATA FLOW CHARTS

A first-level data flow chart of a computing system is a logical schematic representation of the various functional devices and units which give the system its inherent operating characteristics. The chart includes connecting lines to show the paths by which data and instructions are routed from unit to unit within the system during the execution of the program. Generally, these connecting lines are single lines which symbolize the several lines of a parallel-by-bit coded-information channel (which itself may be either serial or parallel by character). The functional units themselves are usually shown as boxes whose relative size may denote the comparative physical size of the circuitry and "hardware" they each contain.

In a previous section of this report the various essential sections of a general-purpose system were outlined and justified on a basis of the functional requirements of the system. Now the application of these devices in actual systems in current IBM production will be reviewed.

A. IBM 604 DATA FLOW

Figure 1 is the schematic layout of data flow in the IBM 604 Electronic Calculator. The basic machine contains 37 decimal digits of electronic (trigger) storage, plus a 13-position electronic "counter" which has an "add-subtract" accumulating function similar to the counters in an accounting machine. Factor Storage has the ability to read in from the attached card-operated Type 521 Computing Punch, and to read in or out at high speed under control of the control-panel wired program. Its 16 positions are subdivided into four units, of 3, 5, 3, and 5 positions respectively. The coupling features expand the unit capacities to a choice of 3, 5, 6, or 8 position combinations since either of the 3's may be coupled to either of the 5's,and one of the 3's may be coupled to the other. General Storage also contains 16 positions and is identical with Factor Storage except for the additional feature of the ability to read out to the 521 for punching of results into the card from which the input data was read during the previous card cycle. The Electronic Counter (13 positions) is similar to General Storage except for inability to read in from the 521.

The counter also holds multiplication products, division dividends, and the results of add and subtract operations by combining its original contents with program-selected data from the storage units. There is also a five-position M-Q (multiplier-quotient) storage unit which permits the use of a five-digit multiplier and the development of a five-digit quotient. The total count of decimal storage positions is then 50.

Since the 604 operates serial-by-bit, parallel-by-digit, there is also an eight-position column-shift unit which provides a parallel shift of the entire factor being transferred of 0-5 positions to the left. This feature permits decimal point scaling by programming and also the successive shifts required during multiplication and division.

B. IBM 305 RAMAC DATA FLOW

Figure 2 is the data flow chart of the IBM 305 RAMAC (Random Access Memory Accounting Calculator). The basic elements of the system are:

1. The processing drum.
2. The disk file.
3. The core buffer.
4. The peripheral equipment.

These elements will be discussed in that order.

Processing Drum — The processing drum contains 20 serial-by-bit tracks for instruction storage, four for data storage, four for arithmetic operations, one to buffer the console typewriter, two to buffer and check the card reader, and one for both the card punch and line printer. These 32 tracks contain 100 characters each. One character position contains eight bit-spaces for the seven bits of the character and an always-written space bit (which separates adjacent characters).

Disk File — The disk file contains fifty disks, coated on both sides with a magnetic iron oxide. There are 100 tracks on each disk-side, each divided into five 100-character sectors. The entire file then has a capacity of 50 x 2 x 5 x 100 = 100 = 5,000,000 characters, addressable in blocks of 100, the same size as a track on the drum.

Core Buffer — The core buffer has a capacity of 100 characters and serves as intermediate storage for both the drum and the file (whose time-bases are substantially different and unsynchronized with each other) by synchronizing itself with which ever one it is communicating at the moment. This buffer consists of an array of seven planes of 100 (10 x 10) cores each, a separate plane for each of the character bits. The cores are small (.080" O.D.) "doughnuts" of a special manganese ferrite with a square hysteresis loop and low coercivity (magnetic hardness). The core buffer is connected successively to the single-character "From" and "To" block-addresses specified by the current instruction in the stored program. This produces very efficient transfer operations on approximately

punched-card-size (100 character) blocks of information between the drum, the file, and the peripheral equipment, resulting in a well-balanced card-processing system. Also shown are some of the control panel "logic" functions, which will be reviewed later.

C. IBM 650 DPS DATA FLOW

Figure 3 is the data flow chart of the IBM 650 Data Processing System. Compared to the previous charts this one is very simple even though the 650 is more complex than the 305. This comparison demonstrates the latitude of data flow charting. The general flow of data is from the read feed of the IBM 533 Reader-Punch to read buffer storage on the drum. From the drum data is block-transferred to General Storage (which is addressable by word) and also on the drum. The stored program processes information from the drum a word at a time, data words to the Distributor and instruction words to the Program Register. Results are stored in General Storage where they are punched into blank cards with the punch feed of the 533. The word size is fixed, at 10 decimal digits plus arithmetic sign.

General Storage contains 2,000 words, the read and punch buffers 10 each. The instruction format consists of a two digit Operation Code, a four digit Data Address and a four digit Instruction Address. The program advances by executing the Operation Code, using the contents of the Data Address as the operand, and then going to the Instruction Address for the next instruction. The accumulator has a capacity of two words, and contains both products and dividends (a dividend being replaced during the division by the quotient and remainder). Since the "working" storage units, the program register, distributor, and accumulator are dynamic storage devices which do not produce static outputs, the Operation and Address Registers are used for a static analysis of operation codes and addresses which are normally required for a substantial period of time during the execution of a given instruction. Validity checks are used to constantly monitor the accuracy of the information in the various dynamic registers to detect internal malfunctions as soon as they occur.

D. IBM 7070 DPS DATA FLOW

The basic data flow of the IBM 7070 Data Processing System is shown in Figure 4. Most of the data flow paths are parallel-by-bit, parallel-by-digit, handling the entire word being transferred "broadside," 53 bits wide. These parallel transfers take place in either four or six microseconds per word of 10 digits plus sign. Paralleling permits a virtual speed increase over serializing of about 10 to 1 for a given type of circuitry at a cost ratio of less than 10 since the controls need be supplied only once. The heart of the 7070 is its multiplexed 5,000-10,000 word core memory, which operates at a speed of 6 microseconds per cycle. Instructions and input data are introduced into memory via punched cards, magnetic tape, and typewriter under the control of a stored program. The word format is similar to the 650 (10 digits plus sign). The instruction format is -OP-IX-FC-DATA, two digits plus sign for the Operation Code, two digits which specify the Index Register by which the Data address will be modified, two digits which specify which segment of the operand word is to be used, and a four digit Data address normally referring to core memory.

For a typical "add" operation, the augend will already be in one of the three accumulators (specified within the operation code). The addend is in memory at the data address from where it is transferred in six microseconds to the Arithmetic Register. Simultaneously the augend is transferred from the accumulator to the Auxiliary Register. Then the arithmetic and auxiliary registers read out serially to the adder, in synchronism with each other, with the sum going back into the arithmetic register, replacing the addend digits one at a time as they are used up. A parallel transfer of the sum from the arithmetic register to the active accumulator completes the operation. "Add-to-memory" is quite similar except that the sum goes back to the operand location in memory instead of to the accumulator. The instruction path from memory to the Program Register (which is immediately to the left of the auxiliary register in the figure) has been omitted from the chart.

Figure 5 is the data flow chart of the CPU (Central Processing Unit) for the IBM 705 Data Processing System. The box labelled "Acc & Aux Storage" in the lower left corner corresponds to the accumulators of the 7070 and consists of a 256-digit accumulator, and fourteen 16-digit and one 32-digit storage registers. Data flow is serial-by-character using a 51-character seven bit code which will be discussed later. CR1 and CR2 are one character registers which re-time the information for the adders. There are separate adders for the coded equivalents of the Hollerith (punched card) code. Multiplication (and division) are performed by using the Multiple Generator, which develops X1, X2, X4a, X4b and X5 multiples simultaneously. The proper multiples are selected by analyzing the immediate multiplier digit, resulting in the development of the product the entire multiplicand times one multiplier digit at a time. The two T/C boxes are true-complement control circuits which develop the complement of one of the input characters or the other when required. The remaining boxes are required for system "housekeeping" functions which will become apparent later in this report.

These five systems which have been reviewed were chosen as representative primarily to demonstrate the purpose of data flow charts.

Additional information is available from the Engineering Library and Stationery Stores for anyone who desires more detailed information regarding any particular system. The information presented so far will provide sufficient background knowledge for a good understanding of the succeeding topics and is required for a broad understanding of the underlying principles of any digital computer system.

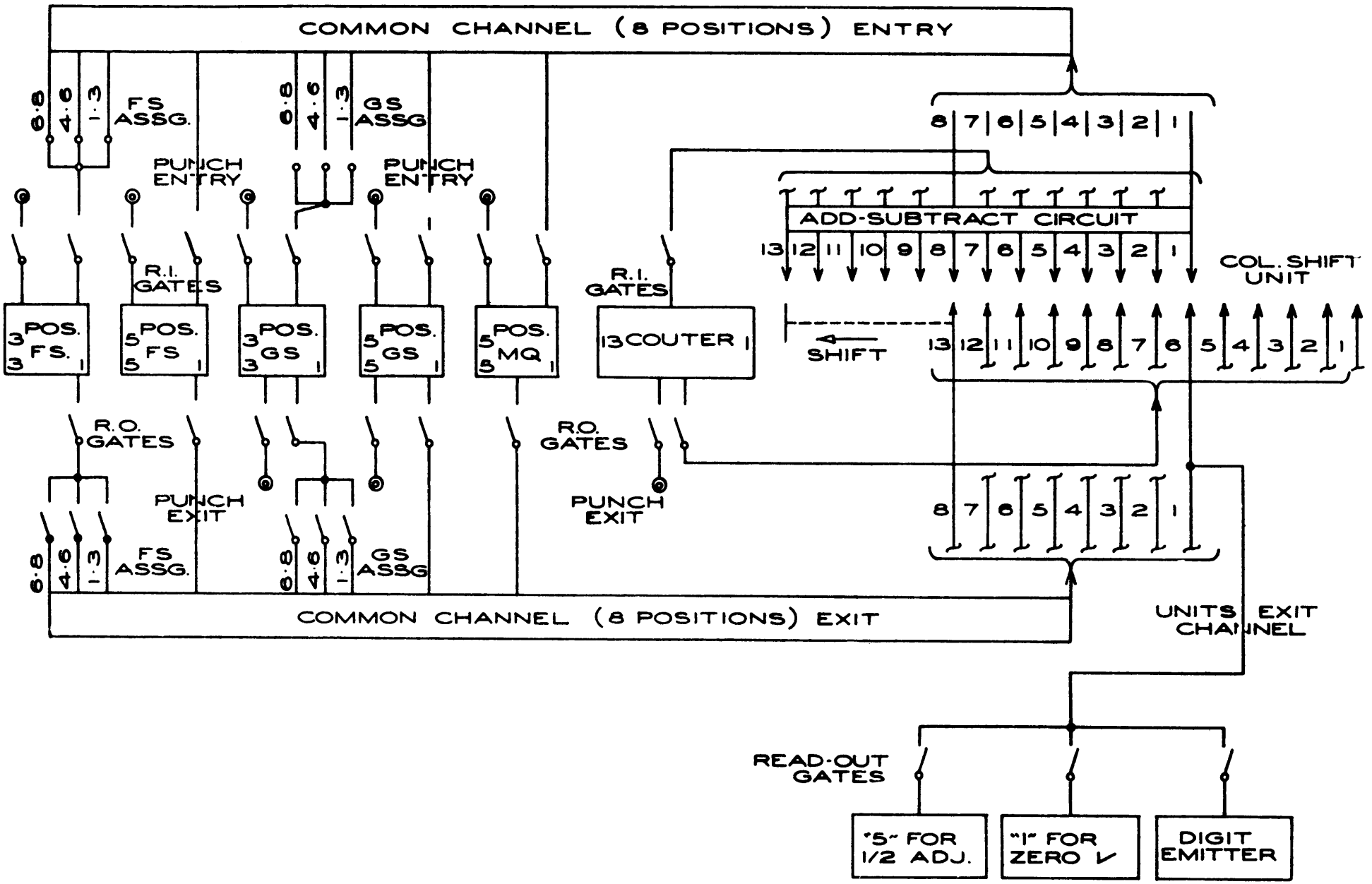


Fig. 1 - Schematic Layout of the IBM 604 Electronic Calculator

TD 01.01.002.005

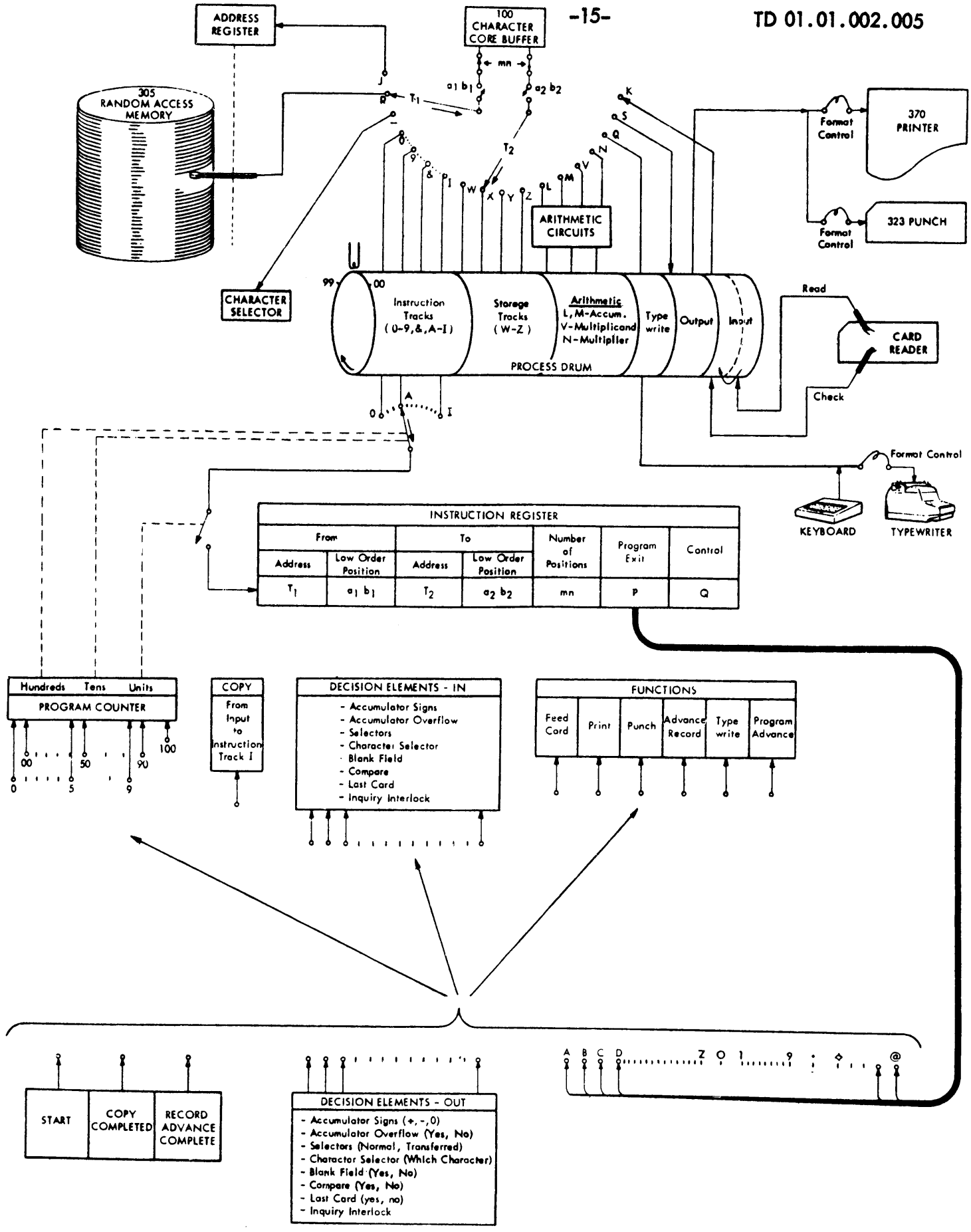


Fig. 2 - Data Flow Chart of the IBM 305 RAMAC

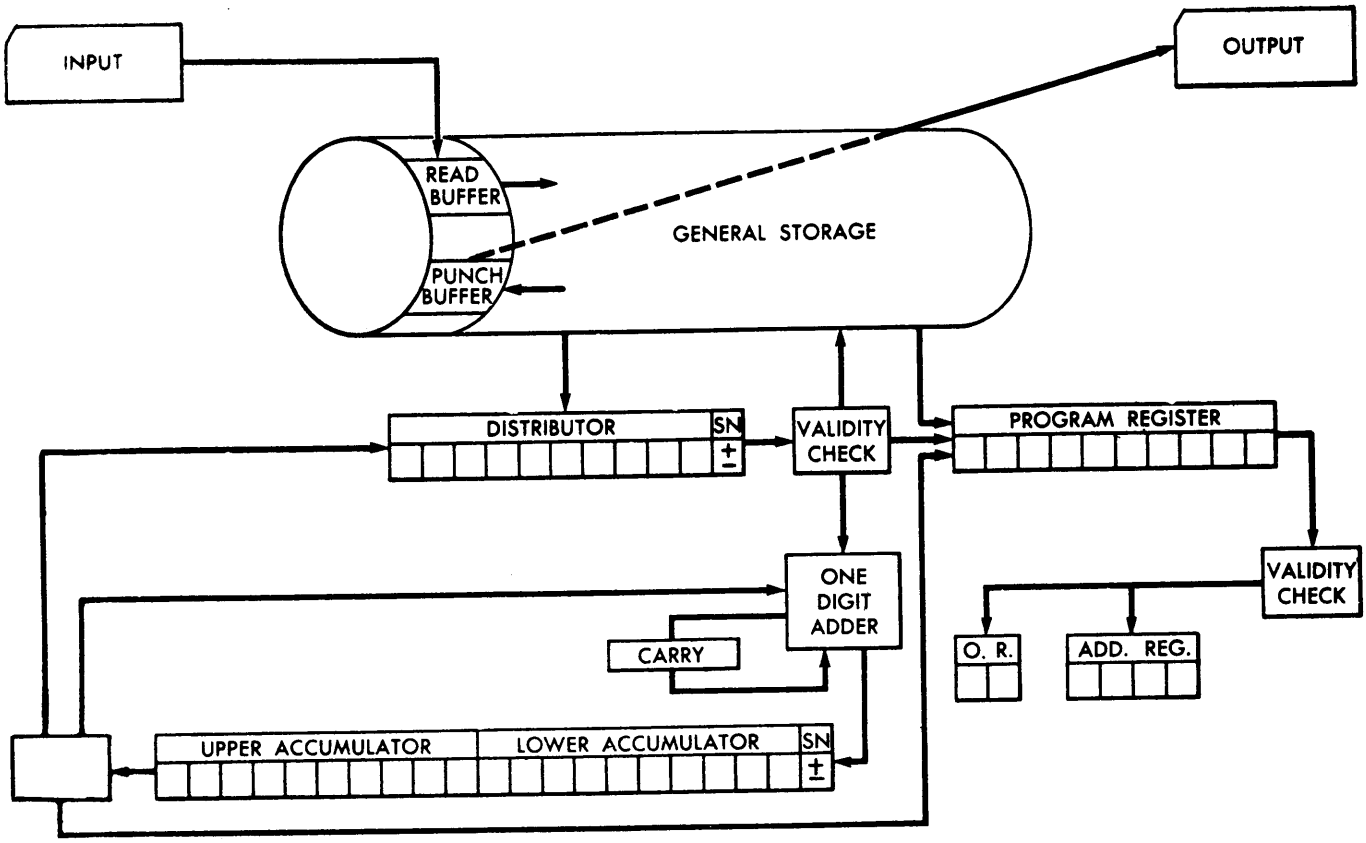


Fig. 3 - Data Flow Chart of the IBM 650 Data Processing System

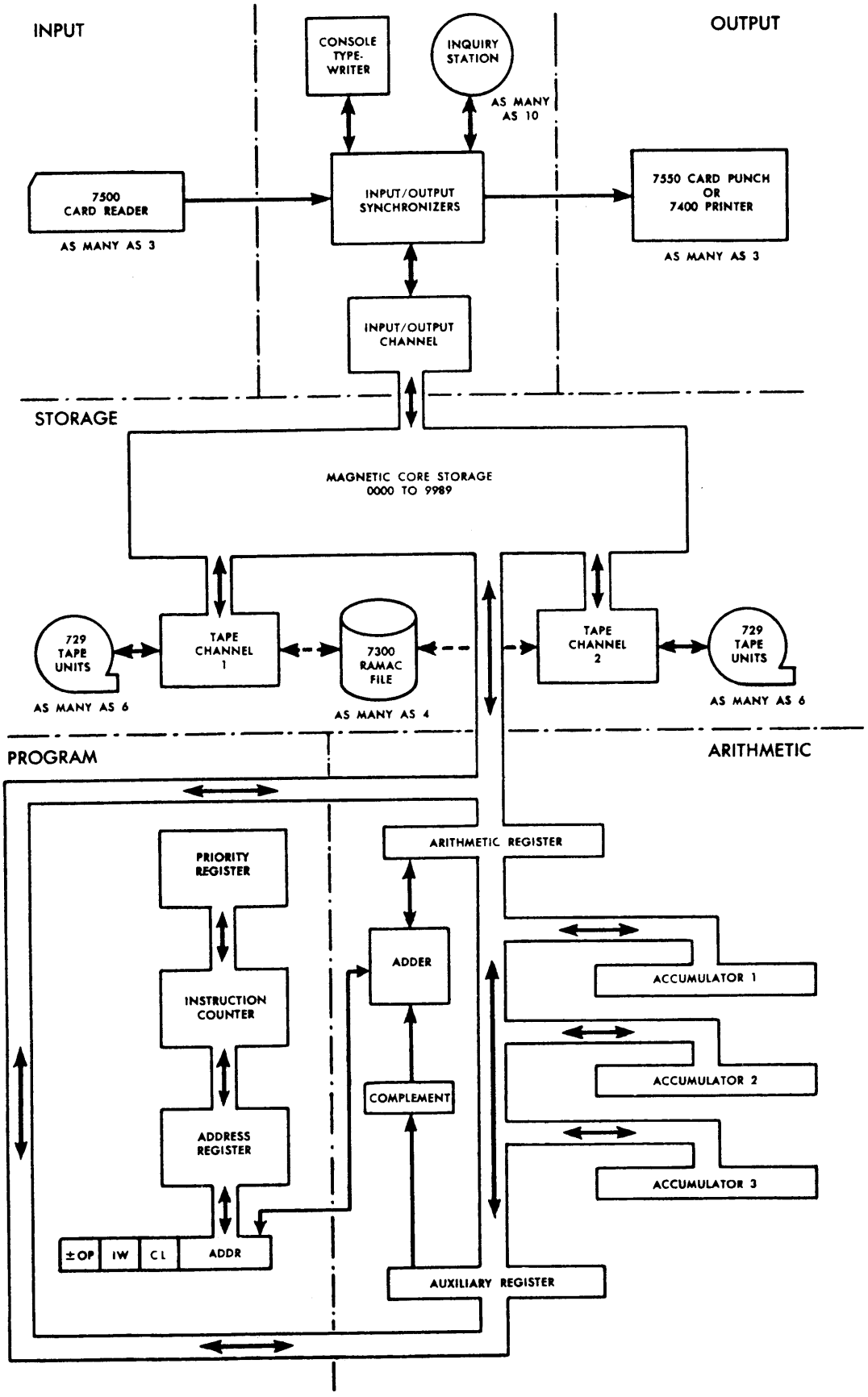


Fig. 4 - Data Flow Chart of the IBM 7070 Data Processing System

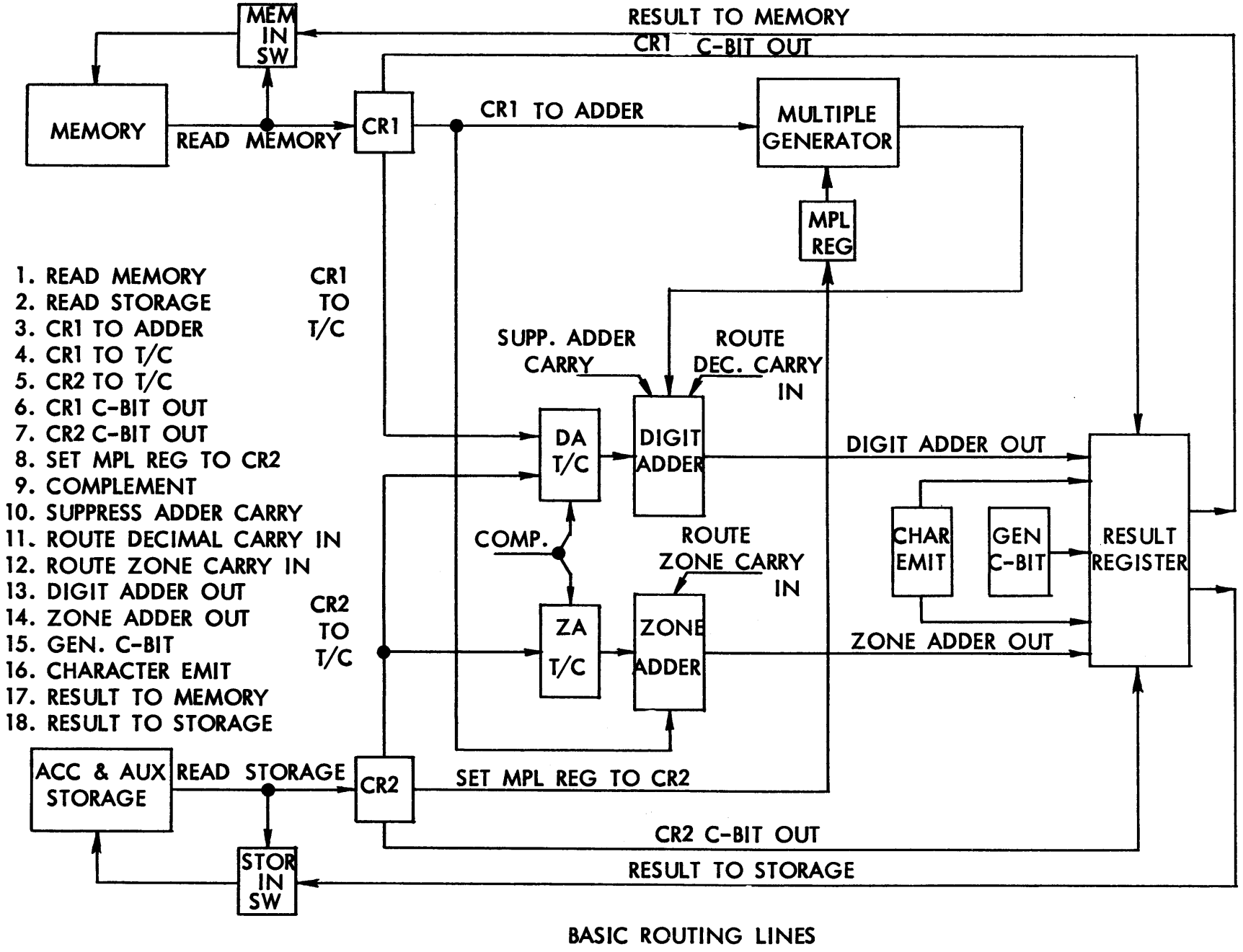


Fig. 5 – Data Flow Chart of the CPU of t' 'BM 705 Data Processing System

V THE BINARY NUMBER SYSTEM

The conventional decimal numbering system and its arithmetic owe their existence primarily to the fact that humans have ten fingers, which early in civilized history made convenient and readily available counters. Since ten fingers are divided on two hands, the bi-quinary numbering system of the abacus was also quite natural.

Similar reasoning shows that since many high speed electronically-operated memory devices have two well-defined "states" or "conditions," and each type of device "switches" abruptly under the control of its input signals, the most natural numbering system for use in electronic digital computers is binary (two-valued). For the moment the problem of bilateral conversion between binary numbers, and the decimal and alphabetic world we live in, can be ignored. Let us just try briefly to "think" in binary numbers.

To begin, assume that the number-symbols 2, 3, 4, 5, 6, 7, 8, and 9 do not exist. Thus, when we count (enumerate), the progression must go 0 to 1, 1 not to 2 but to 10, 10 to 11, 11 to 100, and so on. The following table correlates some common decimal numbers with their binary equivalents:

Decimal	Binary	Decimal	Binary	Decimal	Binary
0	0	10	1010	200	11001000
1	1	20	10100	300	100101100
2	10	30	11110	400	110010000
3	11	40	101000	500	111110100
4	100	50	110010	600	1001011000
5	101	60	111100	700	1010111100
6	110	70	1000110	800	1100100000
7	111	80	1010000	900	1110000100
8	1000	90	1011010	1000	1111101000
9	1001	100	1100100	1023	1111111111

Table 2 - Correlation of Some Common Decimal Numbers with Their Binary Equivalents

The following examples illustrate the basic binary arithmetic operations of addition, subtraction, multiplication, and division. Compare them with their decimal counterparts for simplicity.

Addition:

Decimal	Binary
276	100010100
+349	+101011101
625	1001110001

```
:  :::   :
:  :::   1
:  ::: _ 16
:  :: _ _32
:  : _ _ 64
:  _ _ 512
:_ _ _ 625
```

Subtraction:	Decimal	Binary		
	349	101011101	10010	(18)
	-276	-100010100	-1011	(11)
	73	1001001	111	(7)
		: : :		
		: : 1		
		: :--8		
		: --- 64		

Multiplication:	Decimal	Binary	
	23	10111	
	x17	x10001	
	161	10111	
	23	10111000	
	391	110000111	--------256
			128
			4
			2
			1
			391

Division:	Decimal	Binary	
	17	10001	(17)
	23)400	10111)110010000	
	23	10111	
	170	000100000	
	161	10111	
	0	01001	(9)

Fig. 6 - Basic Binary Arithmetic Operations

Referring back to the example of multiplication by "halving and doubling," it should now be apparent that the multiplication process is actually a systematic conversion of the decimal multiplier to its binary equivalent, followed by binary multiplication of the decimal multiplicand (by selective doubling). Decimal-to-binary conversion is accomplished ordinarily in just that way, by successive division by two, the successive remainders being the binary equivalent, in reverse order. For example, consider the decimal value 349 used in the following subtraction example:

2)349(1
2)174(0
2)87(1
2)43(1
2)21(1
2)10(0
2)5(1
2)2(0
2)1(1
0

The binary value is 101011101. This is the decimal method of conversion to binary. The binary method depends on the binary "code" chosen to represent the individual decimal digits in a binary computer or calculator. The problem is too complex to deal with here. Binary-to-decimal conversion is demonstrated in the checks of the binary arithmetic examples given above.

VI BOOLEAN ALGEBRA

Boolean algebra is a system which provides the symbology and operating rules for the expression and analysis of formal logic statements by the use of techniques similar to those of conventional algebra. This similarity is probably Boolean algebra's greatest difficulty since it is quite natural to lapse into conventional algebra during a Boolean algebra problem. If this potentiality is kept firmly in mind, it is possible to avoid it.

The following rules are listed to demonstrate the characteristics of Boolean algebra without resorting to the lengthy definitions used in a more formal treatment of the subject:

Function	Usage	Boolean Form
AND (A AND B)	Active only when both A and B are present.	$A \cdot B$
OR (A OR B)	Active if either A or B is present.	$A + B$
NOT (NOT A)	Active when A is not present.	$\overline{A}$

Table 3 - Rules Demonstrating Boolean Algebra Characteristics

Identities:

$\overline{A} + \overline{B} = \overline{A \cdot B}$; $\overline{A} \cdot \overline{B} = \overline{A + B}$; $\overline{\overline{A} + \overline{B}} = A \cdot B$; $\overline{A \cdot \overline{B}} = \overline{A} + B$; $A \cdot \overline{A} = 0$;

$A + \overline{A} = 1$; $A + B = B + A$; $A \cdot B = B \cdot A$

It is seen that the "plus sign" of conventional algebra is used to signify the OR and the multiplication symbol is used for the AND. The NOT is symbolized by a bar over the top of the function symbol(s) to which it refers. Boolean algebra will be used as a tool in the next nine sections to permit efficient coverage of the topics of information codes and their arithmetic.

VII THE LOGICAL BINARY ADDER

One of the more interesting devices in its various configurations is the logical binary adder which will now be analyzed in detail. The basic logical objective of the binary adder is the implementation of the simple charts of Figure 7 which are known as "truth tables." These tables provide the individual output results for the four possible combinations of inputs A and B on a basis of whether neither or either or both are present (as binary "ones").

The next step in the process of logical synthesis of the truth table is to convert the entries in the table into terms of Boolean algebra equations. The following two equations result for the "sum" and "carry" respectively:

1) $S = A \cdot \overline{B} + \overline{A} : B$ (either, but not both)

2) $C = A \cdot B$ (both, only)

This pair of equations represents the so-called "binary half adder." The reason for the qualification "half" is that a complete or "full" adder also has the arithmetic "carry-in" as a third entry, along with A and B. This is a natural requirement for a practical binary adder which operates serially (one bit at a time, low order to high), similarly to the mental process of binary addition demonstrated in the section of this report of the binary number system. Figure 8 contains the truth tables for sum and carry for the full adder. Since there are now three binary input variables, there are 2^3, or 8, entries in each table. Just as in the case of the half adder, these truth tables can be easily converted into Boolean equations:

3) $S = A\overline{B}\overline{C} + \overline{A}B\overline{C} + \overline{A}\overline{B}C + ABC$

4) $C' = AB\overline{C} + A\overline{B}C + \overline{A}BC + ABC$

These equations can be modified by factorization and other Boolean operations into a number of different equivalent forms, resulting in various corresponding equivalent symbolic logic circuits. The symbolic logic for equations 1) and 2) is shown in Figure 9. Figure 10 illustrates how two half adders can be cascaded to form a full adder in symbolic logic form. This form of the full adder can also be converted to its Boolean form, for comparison with equations 3) and 4), as follows:

5) $S = \overline{(A\overline{B} + \overline{A}B)\,\overline{C}' + (A\overline{B} + \overline{A}B)\,C'}$

6) $C = (A\overline{B} + \overline{A}B)\,C' + AB$

Multiplying equation 5) to eliminate the parentheses gives equation 3) directly. Equation 6) can be similarly checked against equation 4) by expanding the first term into two terms and then multiplying the other term by C' and $\overline{C'}$ in turn (since $C' + \overline{C'} = 1$, this operation is valid).

Now, if we think of the symbolic circuits in terms of, say, their transistor and diode equivalents, the count for the circuit is 30 diodes and 0 transistors, while the corresponding count for the circuit of Figure 10 is 18 diodes and 1 transistor. This means that by converting to the logic of Figure 11, we have effectively "traded" 12 diodes for 1 transistor, which is a quite significant saving in "hardware," since both circuits do exactly the same thing. There are actually many different forms of binary adders possible. The choice of any specific form for a given application is usually made on a basis of the other factors involved in the choice, such as economy, reliability, speed, etc.

SUM

		A	
		0	1
B	0	0	1
	1	1	0

CARRY

		A	
		0	1
B	0	0	0
	1	0	1

Fig. 7 - Simple Truth Tables for the Half Adder

SUM

	A:	0	0	1	1
	B:	0	1	0	1
C:	0	0	1	1	0
	1	1	0	0	1

CARRY

	A:	0	0	1	1
	B:	0	1	0	1
C:	0	0	0	0	1
	1	0	1	1	1

Fig. 8 - Truth Tables for Sum and Carry for the Full Adder

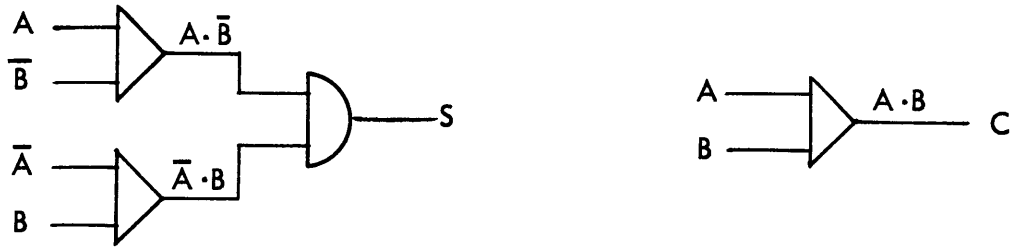


Fig. 9 - Symbolic Logic for Equations 1) and 2), page 23

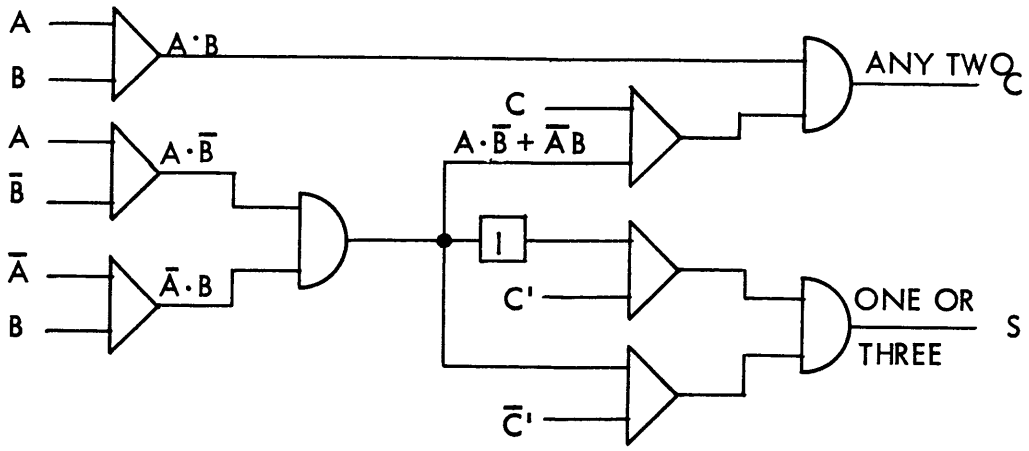


Fig. 10 - Two Half Adders Cascaded to Form a Full Adder in Symbolic Logic Form

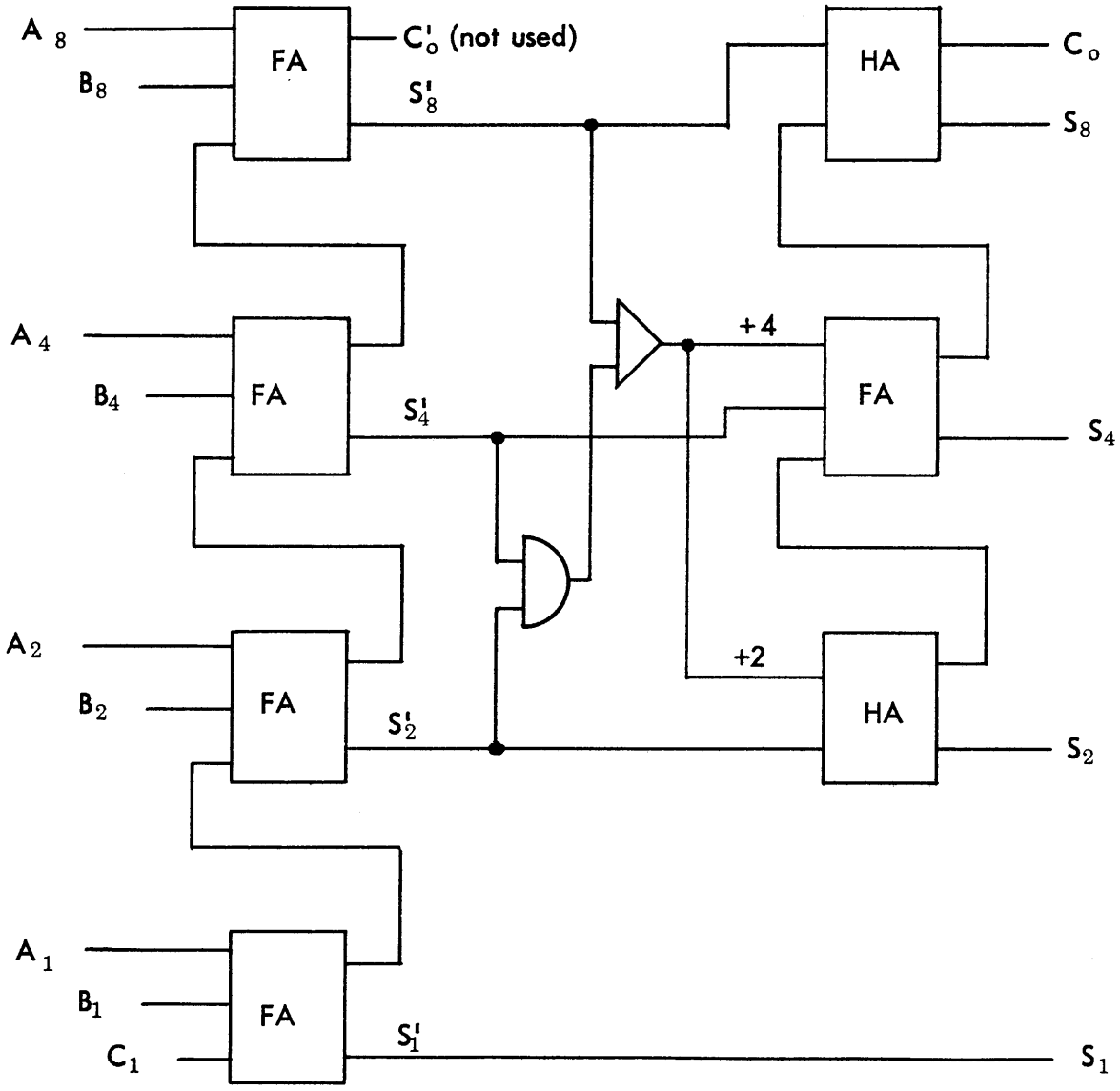


Fig. 11 - Symbolic Description of BCD Adder

VIII BINARY-CODED DECIMAL ARITHMETIC

Since the binary adder and its arithmetic are relatively simple, it is a very natural evolutionary step to recode the decimal symbols into binary form and develop a set of arithmetic rules to compensate for differences between the two number systems. It requires a four-position binary number to express (that is, contain all the possible different values of) a single decimal digit. This binary number, however, is actually the binary code for a radix-16 digit, since it has that many possible values. The effect of this characteristic is shown in the following examples of binary-coded-decimal (BCD) addition:

Example 1		Example 2		
decimal	BCD	decimal	BCD	
6	0110	7	0111	
+3	+0011	+5	+0101	
9	1001	(1) 2	1100	(12, base-16)
			+0110	(carry corr.)
			(1) 0010	(decimal 2, plus carry)

Fig. 12 - Examples of Binary-Coded Decimal (BCD) Addition

As illustrated in the second example, if there is a decimal carry required for the result, the sum will be six too low. This situation requires that there by a logic circuit monitoring the sum (of a BCD adder) to detect a sum greater than nine and apply the required correction of +6. This is included in the logic of Figure 12, which symbolically describes the BCD adder in terms of binary full and half adders logically cascaded to accept four binary entry-pairs broadside. The intermediate sums (with primes) are the uncorrected (base-16) outputs, which then in turn feed the inputs of a second partial adder to provide the +6 correction as required, on a basis of detecting the need for correction as 8 (4 + 2) in the intermediate sum. This 8 (4 + 2) signal is also used directly to provide a "6" for the correction adder.

A variation of the above BCD code (not widely used by IBM) is known as the "excess-three" code. Its characters are respectively three units higher in binary value than their BCD counterparts, ranging in binary "weight" from 3 to 11, as shown in the following table:

0	0011	5	1000
1	0100	6	1001
2	0101	7	1010
3	0110	8	1011
4	0111	9	1100

Table 4 - Example of "Excess-Three" Code, A Variation of the BCD Code

One of the unique advantages of the "excess-three" code is that it is self-complementing, since the nines complement of a given digit value may be obtained easily by a direct bit-by-bit inversion of the original value, independent of that value itself. Since a computer must ordinarily subtract as well as add, this is a significant advantage to a low-cost system.

The examples below show the operations of addition and subtraction in this code:

Addition

decimal	excess-3		decimal	excess-3	
3	0110		6	1001	
+6	+1001		+7	+1010	
9	1111	(15, base-16)	13	(1) 0011	(0, excess-3)
	+1101	(Correction, +10)		+0011	(correction, +0)
	(1) 1100	(9, excess-3)		0110	(3, excess-3)

Subtraction

decimal	excess-3		decimal	excess-3	
9	1100		4	0111	
-3	-0110		-9	-1100	
6	+1010	(complement)	-5	+0100	(complement)
	(1) 0110	(3, excess-3)		(0) 1011	(no carry)
	+0011	(0, excess-3)		0101	(recomplement)
	1001	(6, excess-3)		+0011	(Correction, +0)
				1000	(5, excess-3)

Fig. 13 - Addition and Subtraction in the "Excess-Three" Code

It is apparent that both addition and subtraction require a correction operation in "excess-3" code, resulting in somewhat slower operation, so that the ultimate choice between the two codes for a given application will very likely depend on a compromise of the requirements, since neither code has any overriding advantages.

IX BI-QUINARY ARITHMETIC

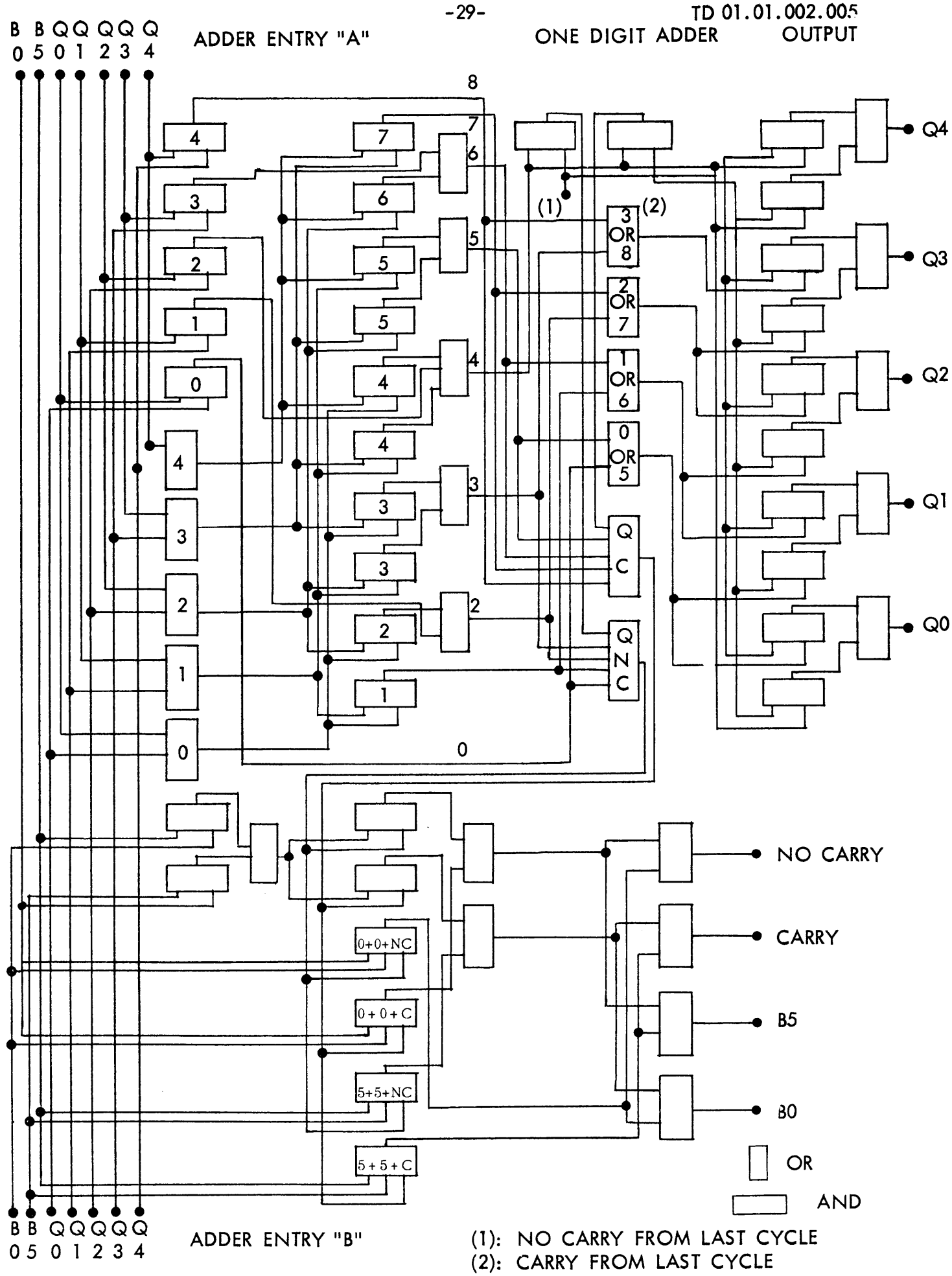
The earliest well-known use of the bi-quinary information code is the abacus. Its natural advantage in that application is that a decimal digit may be expressed in a pattern of seven beads, rather than the 10 required for true decimal representation. This code probably evolved by rationalization of the physical fact that the 10 human fingers, on which our decimal system is founded, are located five each on two hands.

The structure of the code, then, consists of a five-symbol minor quinary part and a major two-symbol binary part, one set for each decimal order of a number. Conventionally, the quinary "bits" are symbolized by their decimal arithmetic weights 0, 1, 2, 3, and 4, while the binary bits are weighted 0 and 5. Thus we have binary 0 and 5, and quinary 0 thru 4.

In arithmetic operations, "carries" are propagated similarly to the decimal system since a quinary carry (count beyond 4) advances the binary value for the same digit position to the other symbol, and a binary carry advances the quinary count for the next higher order digit by one. Thus, it is seen that the arithmetic is carried out as if the binary and quinary parts were successively alternating quinary and binary orders. It may help, to consider bi-quinary arithmetic in this fashion.

By way of historical reference, the program and arithmetic control circuits of the IBM 650 operate completely in bi-quinary code. Figure 14 is a logic diagram of the 650 arithmetic adder. The horizontal boxes are ANDs, and the vertical ones ORs. This circuit has some very interesting logic short-cuts included in it, as will be evident during an analysis of several cases of single-digit addition.

Another code which is very similar to bi-quinary is the qui-binary code used in a number of current development systems. The basic difference is that the binary and quinary parts of the code are transposed. As a result, the quinary bits have weights of 0, 2, 4, 6, and 8, and the binary bits weights of 0 and 1. The rules of arithmetic are otherwise the same as for bi-quinary. The minor advantage of this code over bi-quinary is that the translation to and from BCD is more direct, since for one thing the binary one-bit in qui-binary is the same as the one-bit of BCD. So is the quinary eight-bit the same as the BCD eight-bit, and so on. However, as is usual in such cases, there is a compensating disadvantage, which will be reviewed later in the section on "code translation."



(1): NO CARRY FROM LAST CYCLE
(2): CARRY FROM LAST CYCLE

Fig. 14 - Logic diagram of the 650 arithmetic adder

X BI-QUINARY COMPLEMENTING AND VALIDITY CHECKING

As seen during the study of the binary adder (Section VII), subtraction by a logical adder requires the complementation of the subtrahend, and also re-complementation of the difference if the minuend were the smaller value (resulting in a change of sign from minuend to difference). This same consideration applies to the bi-quinary adder — the subtrahend must be complemented.

The bi-quinary code has the useful advantage of being "self-complementing." This means that complementation can be accomplished on a bit-by-bit basis rather than character-by-character. In this code, complementing is done very simply by a consistent transposition of bits, as seen in the following table:

Decimal	B True	Q	B Complement	Q
0	0	0	5	4
1	0	1	5	3
2	0	2	5	2
3	0	3	5	1
4	0	4	5	0
5	5	0	0	4
6	5	1	0	3
7	5	2	0	2
8	5	3	0	1
9	5	4	0	0

Table 5 - Bi-Quinary Complementing by a Consistent Transposition of Bits

As deduced from this table, complementation requires the transposition of the binary 0 and 5 and the quinary 0 and 4 and quinary 1 and 3. The quinary 2 does not change since decimal 2 and 7 both have a Q2. The basic advantage of self-complementation is that no character analysis is required, since the same "rule" is always shared by more than one character. In addition, a valid complement can result only from a valid true value, preserving the self-checkability of the code which depends on there always being one and only one binary bit and one quinary bit. Figure 15 depicts the true complement logic used in the IBM 650.

Figure 16 shows the logical bi-quinary validity check used in the IBM 650. By way of introduction, the basic Boolean expressions for the validity check are as follows:

$$\text{Quinary Error} = \overline{(0 + 1 + 2 + 3 + 4)} + 01 + 02 + 03 + 04 + 12 + 13 + 14 + 23 + 24 + 34$$

$$\text{Binary Error} = (0 + 5) + 05$$

$$\text{Character Error} = \text{Binary Error} + \text{Quinary Error}$$

These equations are simplified by factorization to the form corresponding to the circuit of the figure. Incidentally, this logic is also applicable as is to the qui-binary code, since the code structure is equivalent, the only difference being in the bit-weights which have no bearing on the checking rules.

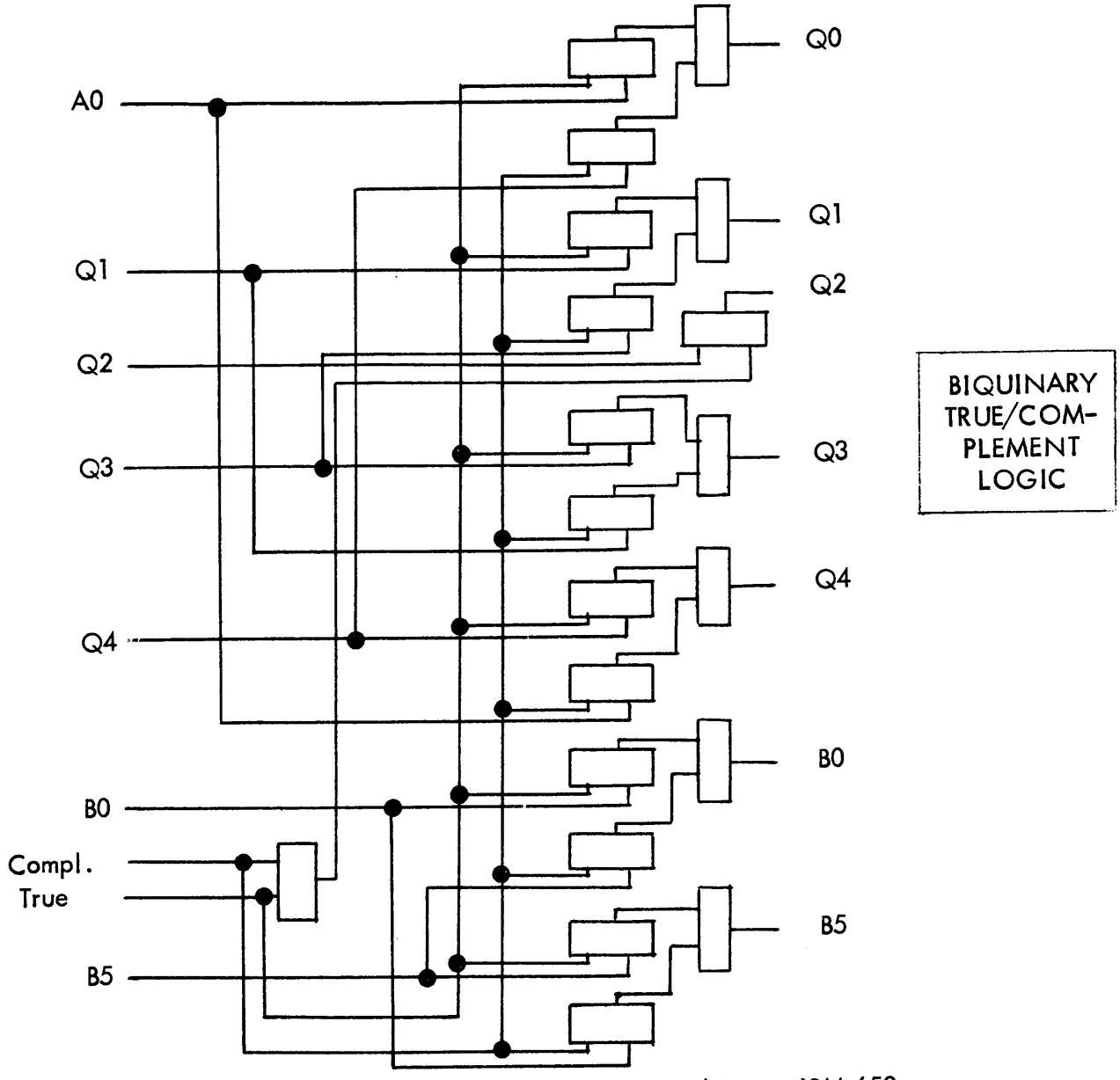


Fig. 15 - True complement logic used in the IBM 650

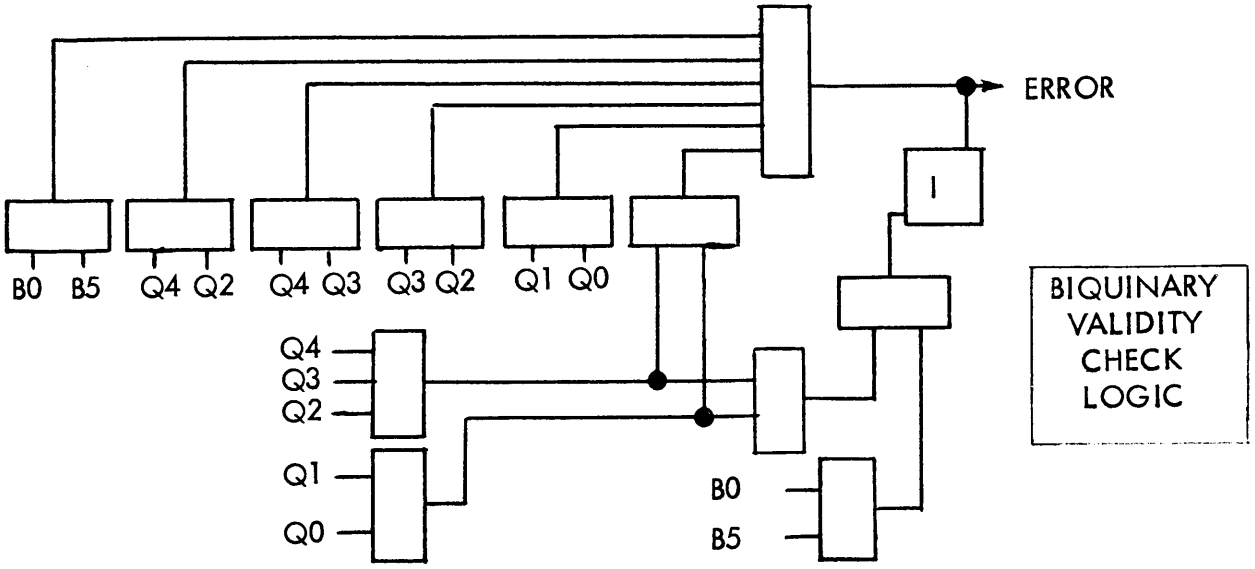


Fig. 16 - Logical biquinary validity check used in the IBM 650

XI FIXED-BIT INFORMATION CODES

Fixed-bit information codes are a class of codes having the unique property that all characters have the same bit-count (of 1's). The most commonly used of these codes is the two-out-of-five code used in general (drum) storage in the 650 and universally in the 7070. According to the permutation formula,

$$P = \frac{n!}{m!\,(n-m)!}$$

there are ten permutations of five elements taken two at a time. In internal form, they are: AB, AC, AD, AE, BC, BD, BE, CD, CE, and DE, where the sequence of the symbols is not significant. The 2/5 code of the 650 and 7070 assigns arithmetic weights of 0, 1, 2, 3, and 6 to the five bits transferring these weights to the literal symbols given above results in a character sequence of 01, 02, 03, 06, 12, 13, 16, 23, 26, and 36, with respective decimal values of 1, 2, 3, 6, 0, 4, 7, 5, 8, and 9. The decimal values 0 and 3 have ambiguous arithmetic weights of 3 units, since 0 + 3 = 1 + 2 = 3. Normally, however, this code is not used for arithmetic purposes, so the ambiguity is no problem. The bit weights are merely a mnemonic aid, mainly to the customer engineer maintaining the system out in the field (the logic designers, using Boolean algebra, could just as well have used the literal symbols A, B, C, D, and E).

Incidentally, of all the other possible 2/5 codes, there is only one equivalent to the 0-1-2-3-6 code which has all positive bit-weights and only the zero ambiguously weighted. It is the 0-1-2-4-7 code, whose weight for the zero is 11, the 4-7 combination.

The really powerful reliability advantage of the fixed-bit codes is their complete immunity to multiple bit-errors of a common polarity (0 to 1), which can never revalidate the affected character, as is possible in any of the parity-checked codes (which are checked by establishing the count of binary ones in the character as either even or odd). Multiple bit-errors are generally caused by polarized noise, excessive power supply regulation, or failure of one or more of the data flow control gates (which can permit whole characters to be superimposed one on the other bit-by-bit).

One larger-capacity fixed-bit code used in IBM equipment is the four-out-of-eight code of the IBM 65 Data Transceiver used to transmit the contents of punched cards to another transceiver at the remote end of a conventional land line. This particular code used 54 out of the possible 70 characters available in terms of eight elements taken four at a time $[8!/(4!)(8-4)! = (8 \times 7 \times 6 \times 5)/(4 \times 3 \times 2 \times 1) = 70]$. The coding chosen for the specific 54 characters of the code is based on its relatively straightforward translation to and from the Hollerith code of the punched cards that the machine processes.

Currently-produced systems use a 51-character seven-bit parity-checked code. A three- (or five-) out-of-eight code would provide 56 characters, and could conceivably be optimum for some future system. However, since the current seven-bit code has a capacity of 64 characters and both three-out-of-eight and four-out-of-eight naturally require the same storage and data-flow capacity in a given system, it would probably generally be wiser to adopt the "bigger" code as a company-wide standard for future systems which require fixed-bit information coding. This is true because it and the seven-bit parity-checked code would probably always co-exist.

XII SELF-CHECKING AND SELF-CORRECTION CODING

Due to IBM's direct responsibility to furnish field maintenance support for a major percentage of its computers and data processing systems, self-checked information coding has come to be a more or less traditional functional specification for any new or proposed system. Self-checking itself is accomplished by intentionally including sufficient redundancy (partial repetition or duplication) within each coded character to detect alterations of the characters due to system malfunctions.

In the case of the familiar BCD code reviewed earlier in this report, this redundancy takes the usual form of a parity-bit, which is simply a modulo-2 count of the information bits which are binary 1's for that character, as shown in the following table:

Decimal	8	4	2	1	C
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
(0)	1	0	1	0	1

Table 6 - Redundant Self-Checking by a Parity-Bit

In this table, the parity-check bit is chosen so that the total bit-count for a character is always odd. The parity bit is literally the 1's complement of the information bit-count modulo-2. The choice between odd and even parity is usually made on a basis of criteria such as insuring that all characters will contain at least one binary "one," which applies to the "zero" in the above table. If the 8-2 were chosen as the "zero" character, even parity would also satisfy this requirement.

Fixed-bit codes provide the required redundancy in more subtle form, as an unvarying bit-count. The unique advantage of this characteristic is unambiguity. Since the bit-count is constant, it is obvious that no character can be wholly contained within any other one. The result of combining any unlike characters by bit-by-bit superposition must necessarily result in an increase in bit-count, which violates the consistency of that count, which would indicate the occurrence of an error. This is a categorical advantage for all fixed-bit codes, as the succeeding study of the statistical error probabilities will prove.

Referring to Table 6, it can be seen that the 1, 2, and 4 characters are all contained within the seven character, so that superposition of an extraneous seven on a true 1, 2, or 4 will result in undetected conversion to a valid seven. This possibility defeats the basic ideal of system checking, which is a built-in guarantee of the validity and accuracy of the system's output. Assessment of the relative importance of this guarantee to the eventual field success of a given system is a very sophisticated process requiring a detailed knowledge of the system, and an intelligent estimate of the impact of undetected errors on the system's performance in actual field applications.

An interesting extension of parity checking is applied to IBM's magnetic tape equipment. This technique is known as "cross-parity" checking. It operates in the following manner. Consider a random-length "block" of information characters written on magnetic tape in parity-checked code. The parity bit for each character provides a constant modulo-2 bit-count for each character. The characters are written parallel-by-bit, serial-by-character on the tape. Then, at the end of the "block," an additional "character" generated by modulo-2 counting of the number of individual occurrences of each bit anywhere in the "block." This bit-by-bit parity-count then automatically forms an extra "character," which is written on the tape right at the end of the "block "

All information bits are thus checked twice, and the simplest error in the block which will satisfy the checking circuits must preserve both "vertical" and "horizontal" parity by creating an even number of changes in a rectangular error-bit pattern, requiring at least four simultaneous bit-errors. The 700- and 7000- series systems, as well as the 650, use this powerful checking technique in their respective magnetic tape features.

An additional advantage of cross-parity checking, not exploited by IBM, is that since a single bit-error in the information block on tape is identified two-dimensionally, its location is specified similarly to Cartesian coordinates. Correction of that bit then merely requires extraction, reversal (0 to 1 or 1 to 0) and re-insertion of that one bit to the same location. There are no unknown factors involved; this function is completely feasible. The basic reason cross-parity checking is not used by IBM is that a predominant type of error on magnetic tape is the drop-out of a string of the same bit for a long succession of characters. This is because the seven bit-channels occupy the full width of a half-inch tape, while successive characters are packed to a density of 200 to the inch (each character is nominally only, .005 inch long). Thus, a roughly round blemish on the tape will actually be about 14 times as many bits "long" as it is "wide." Since cross-parity checking could only correct one bit error in the block, it has been dismissed as unprofitable.

It is also possible and practical to include enough redundancy directly within the individual character to correct any information-bit error. This technique was developed by Hamming of Bell Telephone Laboratories during the course of some very early information error studies. The following table demonstrates the Hamming technique as applied to the BCD code:

Decimal	BCD					Hamming		
	8	4	2	1	(C)	X	Y	Z
0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	1
3	0	0	1	1	1	1	1	0
4	0	1	0	0	0	0	1	0
5	0	1	0	1	1	1	0	1
6	0	1	1	0	1	1	0	0
7	0	1	1	1	0	0	1	1
8	1	0	0	0	0	1	0	0
9	1	0	0	1	1	0	1	1

Table 7 - The Hamming Technique as Applied to the BCD Code

The X, Y, and Z bits are generated from the information bits in the following manner:

$$X = 4 \forall 2 \forall 1;$$

$$Y = 8 \forall 2 \forall 1;$$

$$Z- = 8 \forall 4 \forall 1;$$

where the symbol ($\forall$) means exclusive-OR or sum modulo-2. Identification of correctible bit-errors depends on the pattern of failure of the X, Y, and Z checks. If the character is OK, all three are satisfied. If X fails, but not Y and Z, the X-bit itself must be in error, since an information-bit error would fail to satisfy at least two checks. The Y and Z bits are detected similarly.

Now, if both X and Y, but not Z, fail, the error must be the two bit, since it is included in both X and Y parities, but not in Z. In similar fashion, $X\overline{Y}Z$ identifies the four bit, $\overline{X}YZ$ identifies the eight bit, and XYZ identifies the one bit as the error. This identification is independent of whether the error-bit should be a 0 or a 1. Retention of the original C-bit of the BCD code provides detection of double (two-bit) errors, making it possible to detect errors that cannot be successfully corrected. This is important because the correction can only identify one error-bit. If there is a double error, its symptoms will be ambiguous with some other unrelated single error so that the correction logic would, in the case of a double error, change a bit which was correct and ignore the real error, resulting in an undetected triple error in the end.

In passing, it may be noted that if the Hamming check bits are not used for correction, they (without the C-bit) provide double error detection.

There are many information codes, each with its own reasons for existence, on which we have not even touched. A thorough study of codes will prove quite interesting to anyone wishing to attempt it. This can be done fairly easily since most of the work to date has been well documented in the technical literature. IBM's Technical Information Service maintains an extensive bibliography on this and many other subjects.

XIII STATISTICAL ERROR PROBABILITIES

A. DEFINITIONS

1. Error: Any alteration of the bit-pattern representing a particular specific coded character.

   a. Total errors: The count of all possible bit-changes to all code characters.

   b. Error probability: Average probability of bit errors for the the entire code-set.

2. Detected Error: Any bit-pattern alteration which produces an invalid "character," by the rules of the particular code used.

   a. Detected error probability: The ratio of "detected" errors to "total" errors for all possible independent bit-errors, where double errors are considered two coincidental single errors, etc.

   b. Detected error rate: The average ratio of detected character errors to total character quantity. (P_e)

3. Undetected Errors: Any bit-pattern alteration which results in conversion of a valid character into a different valid one.

4. Independent Bit-Error: The reversal (0 to 1 or 1 to 0) of a single information bit caused by logic failure, superimposed noise, etc.

   a. "Zero" error rate: The rate at which errors in which "ones" change to "zeroes," expressed as P_0. The ratio of errors to total bit quantity during which they occur.

   b. "One" error rate: The rate at which "zeroes" change to "ones," expressed as P_1.

   c. Total error rate: The rate, expressed as P_e in terms of P_0 and P_1, at which errors of all types occur, detected or undetected.

   d. Undetected error rate: The rate at which undetected errors occur, expressed as P_u in terms of P_0 and P_1.

B. COMPARATIVE CHECKING RELIABILITY OF THE 2 OUT OF 5 AND BCD CODES

1. Total Error Probability:

a. 2/5 has 2 ones and 3 zeroes for every character, so that

$$P_e = 2 P_0 + 3 P_1$$

For example, if $P_0 \simeq 10^{-6}$ and $P_1 \approx 10^{-12}$ then

$$P_e = 2 \times 10^{-6} + 3 \times 10^{-12} \simeq 2 \times 10^{-6}$$

This means that 2 missing-bit errors will occur every million characters.

b. Consider now the following form of BCD Code:

Dec.	8	4	2	1	C	Dec.	8	4	2	1	C
0	0	0	0	0	1	5	0	1	0	1	1
1	0	0	0	1	0	6	0	1	1	0	1
2	0	0	1	0	0	7	0	1	1	1	0
3	0	0	1	1	1	8	1	0	0	0	0
4	0	1	0	0	0	9	1	0	0	1	1

The complete code uses 20 ones and 30 zeroes (five bits/character x 10 characters). Assuming equivalent usage of all 10 characters and the same P_0 and P_1 as for 2/5,

$$P_e = 2 \times 10^{-6} + 3 \times 10^{-12} \approx 2 \times 10^{-6}$$

which is the same as for 2/5.

2. Undetected Error Probability:

a. For the 2/5 Code, an undetected error must consist of the same number (1 or 2) of zeroes changing to ones and ones changing to zeroes, assuming a perfect error-free validity check, since this condition alone can maintain the constant bit-count of two ones. Using the same values for P_0 and P_1 as before:

$$P_u = (2 P_0 \times 3 P_1) + (P_0^{\,2} \times 3 P_1^{\,2}) =$$

$$6 P_0 P_1 + 3 P_0^{\,2} P_1^{\,2} \approx 6 P_0 P_1$$

$$\approx 6 \times 10^{-6} \times 10^{-12} = 6 \times 10^{-18}$$

b. For the BCD Code defined above, an undetected error is any error which results in no change of the character's bit-count (of ones) modulo 2; i.e., in this case, "odd." Accordingly:

$$P_u = 1/2(P_0 \times 4 P_1 + 6 P_1^2 + P_1^4) +$$

$$1/2(3 P_0 \times 2 P_1 + 3 P_0^2 + P_1^2) =$$

$$+2 P_0 P_1 + 3 P_1^2 + 1/2 P_1^4 + 3 P_0 P_1 + 1\ 1/2 P_0^2 +$$

$$1/2 P_1^2 = 5 P_0 P_1 + 1\ 1/2 P_0^2 + 3\ 1/2 P_1^2 + 1/2 P_1^4$$

$$\simeq 1.5 P_0^2 + 5 P_0 P_1 + 3.5 P_1^2$$

$$\simeq 1.5 \times 10^{-12} + 5 \times 10^{-18} + 3.5 \times 10^{-24}$$

$$\simeq 1.5 \times 10^{-12}$$

The P_u ratio between these two codes, BCD and 2/5, is:

$$(1.5 \times 10^{-12})/(6 \times 10^{-18}) = 2.5 \times 10^5$$

meaning the BCD has an undetected error probability 250,000 times as high as 2/5; this ratio is roughly equivalent to

$$P_0/P_1 = (10^{-6}/10^{-12} = 10^6 .$$

3. Rationalization of Assumptions

a. $P_0 = 10^{-6}$

This is a "missing" bit only every million characters, far from a "solid" condition where $P_0 = 1$.

b. $P_1 = 10^{-12}$

This is an "extra" bit once every trillion characters. Using the 7070 character rate of 250KC, the mean-error free interval comes out to 4×10^6 seconds, or about 15 months of elapsed running time, (which is reasonably pessimistic) for an error in any one functional area of a system.

4. Interpretation of Error Probabilities

a. P_e is strictly a function of the size and format of the code. The two codes used in the comparison are both five-bit codes averaging two ones and three zeroes, so their total error rates are necessarily the same.

b. The basic reason for the marked P_u superiority of 2/5 is that its undetected double errors must cancel each other, while in BCD they can have the same polarity. While common-polarity multiple independent bit-errors are no more plausible than opposite polarity ones, dependent multiple bit-errors of the same polarity are quite common in any real system. They generally result from distributed noise or simple gating or sampling failures. These conditions can cause undetected errors in BCD, but 2/5 is completely immune to them.

XIV CODE TRANSLATION

Typically in a modern data processing system such as the 650 or 7070, one code is not universally optimum from the standpoint of economy and/or reliability. For example, the seven-channel IBM magnetic tape is optimally coded since its information checking is based on two-dimensional bit parity, the seventh channel being the vertical (character parity)check and the horizontal (redundancy) check character providing the second dimension of parity check. Since each of the parity checks requires an even number of bit errors to defeat it, the simplest undetected bit-error pattern must be a four-bit rectangle within a given record. In other words, the code is triple-error-detecting. All single, double, and triple errors are detectable, and all but the rectangular quadruple errors.

However, it is impractical to carry this cross-parity check with the data during its trip through the system. The basic result of data processing is the alteration of old records by "updating" them to reflect current activity. This alteration inevitably destroys the original parity status, implying that the "new" record generates its own parity check information. This "bootstrapped" checking would require fail-safe processing operations, which are impractically expensive if not actually impossible with the BCD Code.

Consequently the 7070 system designers adopted the 0-1-2-3-6 version of two-out-of-five code for internal data transmission and storage, using the two-digit decimal representation of alpha-numeric characters initiated in the 650. The exceptions to universality of the 2/5 code in the 7070 are magnetic tape, the arithmetic adder, and the "one-uppers" (which update memory addresses sequentially for block-transferring of data and serial instruction-acquisition).

The 7070 adder operates in true decimal code using a ferrite-core matrix "addition table." The output of this matrix is recorded by the sense-winding pattern directly back to 2/5. The "one-uppers" logically decode individual address digits to true decimal, shift the value up one unit, and recode to 2/5. Since the decimal "code" is a fixed-bit code (one-out-of 10), the inherent reliability of fixed-bit data-flow can be preserved.

The same general philosophy applies to the 650, the basic difference being the use of the bi-quinary code where the 7070 uses decimal. The practical justification for the bi-quinary code in the 650 CPU is the state of computer circuit technology during its development phase. Magnetic cores were expensive and lacked uniform parameters, requiring that vacuum tubes and point-contact diodes (which also were quite new at the time) be used for circuit logic. The remarkable field performance record of the 650 completely justifies the design philosophy under which it was developed.

As a case study, consider the B/Q -to- 2/5 and 2/5 -to- B/Q translators via which general (drum) storage of the 650 communicates with the rest of the system, as shown in Figure 17. The effect of these translators, whose symbolic logic is given in Figure 18, is that the 650 "thinks" it has a bi-quinary-coded drum.

Since an actual bi-quinary drum would require seven tracks per band, general storage would require 280 tracks instead of 200, an increase of 40 percent. If about 200 heads is the limiting size of general storage, adding 10 more heads (210, total) would provide only 1500 words capacity, a reduction of 25 percent. Obviously, these two translators are vastly more economical than that.

Several current-development systems have adopted the seven-bit BCD code (1-2-4-8-A-B-C) for general data flow. They will translate to the qui-binary (Q: 0, 2, 4, 6, 8; B: 0, 1) code for arithmetic. Since the adder has two inputs (augend and addend), two BCD-to-Q/B translators and one Q/B-to-BCD translator are required. Generation of their Boolean expressions (in terms of input bits, one for each output bit) and conversion to symbolic logic form is left as an instructive exercise for the student.

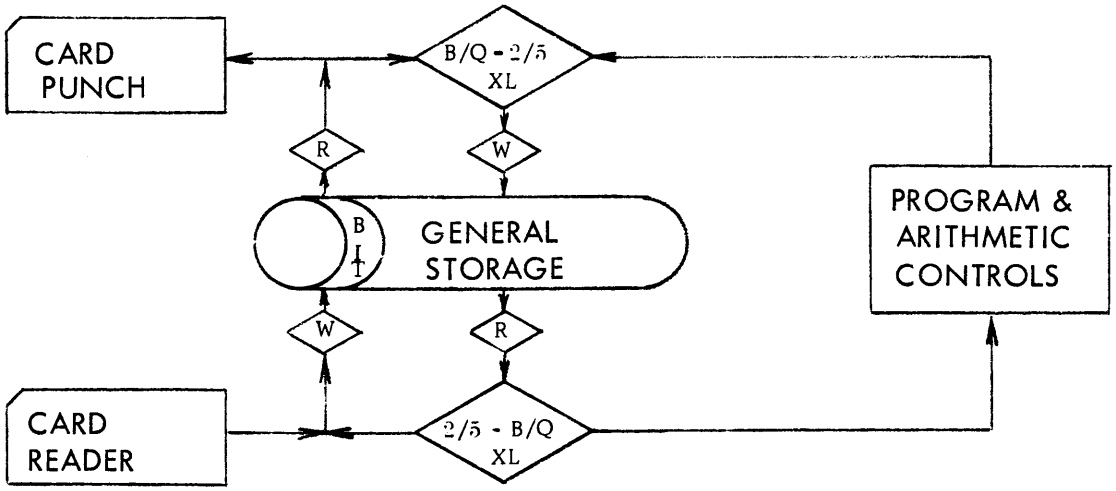


Fig. 17 - Data flow of B/Q-to 2/5 and 2/5-to-B/Q translators in the IBM 650

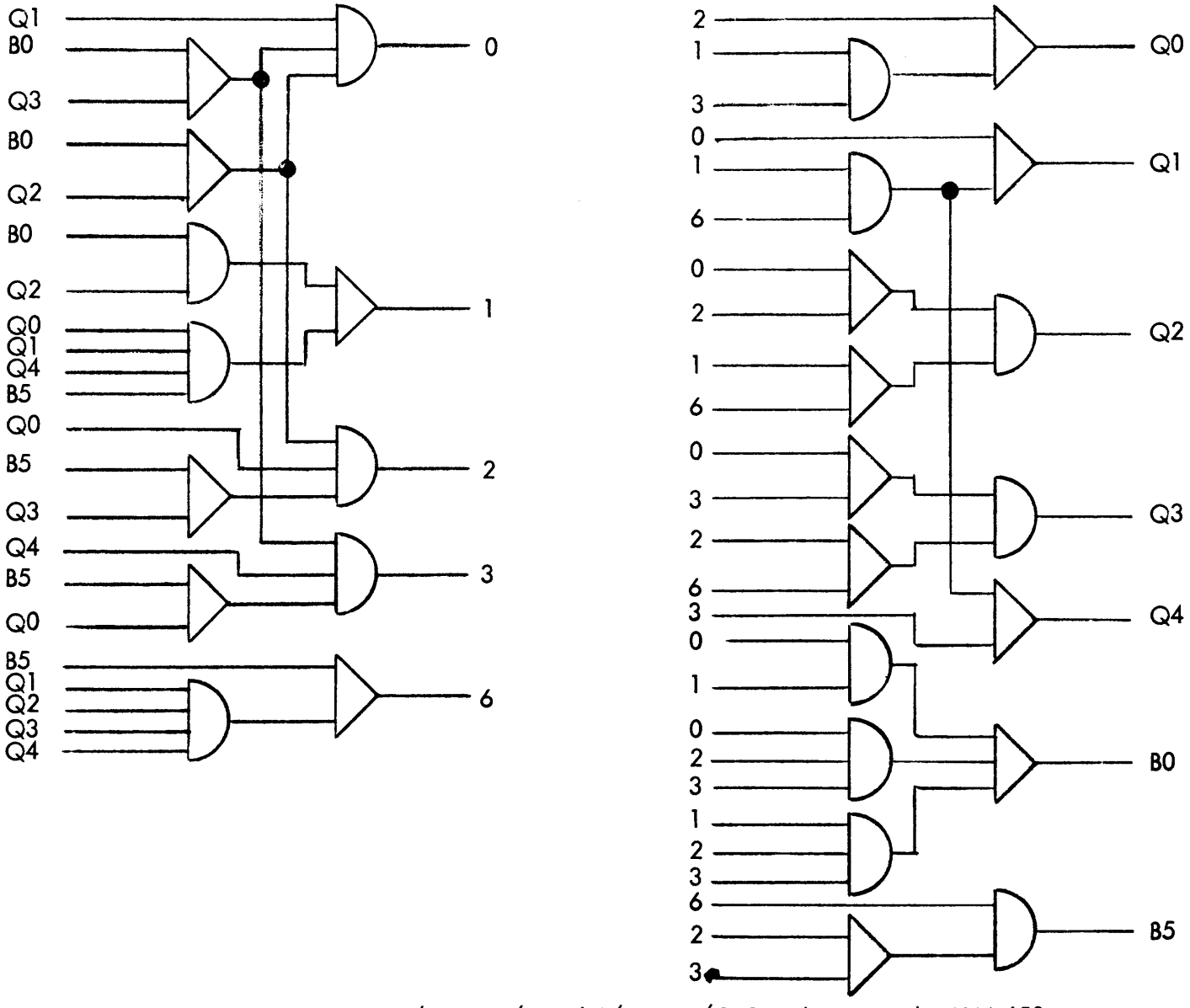


Fig. 18 - Symbolic Logic of B/Q-to-2/5 and 2/5-to-B/Q Translators in the IBM 650

XV COMPARISON OF CODES

The following codes have been reviewed.

1. Binary.
2. BCD, with and without a parity-check bit.
3. B/Q and its converse, qui-binary.
4. Two-out-of-five.
5. Seven-bit BCD, and
6. Three-, four-, and five-out-of-eight.

The use of decimal as a one-out-of-ten fixed-bit code has been mentioned. The objective of the current review is to establish the application areas within which each of these codes is optimum. They will now be considered individually, in the order listed above.

With respect to IBM production system applications, the 701, 704, 709, and 7090 are all pure-binary-coded systems. Even the Poughkeepsie "STRETCH" and Kingston "SAGE" computers are basically binary-mode systems. Outside IBM one finds about as many competitive binary as non-binary systems. This wide prevalence of application requirements for binary systems certainly justifies their "place in the sun." The basic reason for the popularity and almost "necessity" status of binary computers is due to their extreme performance in terms of speed and flexibility, for any given class of electronic "hardware."

As a case in point, the 704 and 705 are quite similar systems physically, both being built out of eight-tube pluggable units and both having the same peripheral units such as tape drive, card equipment, and wheel printers. However, functionally the two systems are vastly different. The 704 is a pure binary machine, serial by bit at one megacycle per second, with elegant logical instructions and a fixed 36-bit word length. On the other hand, the 705 is an alphanumeric machine, addressable by individual seven-bit BCD characters. Its instructions are less flexible and more varied than the 704's, providing more efficient programming for accounting applications.

The 701 and 704 have seen wide use in nuclear physics research computation and aerodynamic flight simulation. The 702 and 705 are broadly applicable to all types of large-scale accounting functions. As an illustration, one insurance customer has seven 705's. This demonstration of the faith of the market in BCD-coded systems is typical since most non-IBM competitive systems also use this code.

The ultimate in information reliability is obtained by the use of fixed-bit information codes, such as the bi-quinary and two-out-of-five codes, of the 650 system. The past years of 650 field experience have demonstrated a characteristic of extreme operating dependability, in the sense of the 650 seeming to have a "conscience" which almost inevitably prevents it from doing anything wrong. This "morality" comes about as a direct result of the adoption of information codes which are perfectly immune to common-polarity multiple-bit and character-superposition errors.

Such immunity permits the design of the control logic for after-the-fact validation of all arithmetic and logical operations. Basically, the control logic is designed so that internal failures will inevitably produce invalidation of the coding of the result of the operation. "Too few" or "too many" or "incorrectly timed" control signals can be detected consistently only if a fixed-bit code is used. Tacit admission of this superiority of fixed-bit codes is inherent in such areas of BCD systems as the memory address "decoder," in which BCD is converted into a four-out-of-eight or five-out-of-ten fixed bit code, to improve reliability in a functional area where component failures are extremely difficult to detect and still harder to analyze.

In the last analysis, the final choice of information code for any new system depends on a realistic assessment of the reliability and cost and function requirements of its intended applications. As IBM moves into the in-line data-processing and real-time process-control market, where information must be processed "right the first time" at almost the same instant of time in which it occurs, reliability will gradually predominate over cost, just as function has in the immediate past.

XVI STATIC STORAGE — TRIGGERS AND LATCHES

Our logic symbology arbitrarily has been limited to ANDs, ORs, and NOTs. Out of these three types of logical connectives, we have seen how such devices as a arithmetic adders and code-to-code translators are developed. Devices such as these operate continuously in the sense that their output signals have roughly the same timing as the input signals.

Obviously, a digital computer does not execute an entire program in one arbitrarily small instant of time. To begin with, arithmetic is normally serial since the intermediate "carries" must be propagated one digit at a time. This means that it will take about twice as long to develop a four-digit sum as a two-digit sum.

Even simpler in concept is the logical fact that arithmetic requires two operands, one number to add and another number to be added to. Since these two numbers are not available from a common memory simultaneously, one of them must precede the other out of memory. This means that it must be "remembered" somewhere outside of memory. Another direct requirement for auxiliary information storage is for instructions, especially the addresses of data contained in them, after their acquisition from memory, during their execution. In addition to data flow level examples such as these, there are many control storage requirements.

A universal requirement is the need to delay the output carry of an adder by one digit-time to become the input carry for the next digit (see the full binary adder and the bi-quinary adder). All in all, there is a universal requirement for logical memory devices which can be controlled and interrogated both randomly and continuously.

The best-known and oldest electronic device of this type is the trigger. It gets its name from its basic characteristic of abruptly changing state as the result of an input signal and then maintaining the new state until a "reset" signal returns it to its original or "off" state. This device was originated in vacuum tube form in 1923 by Eccles and Jordan. A modern version of it (used in the IBM 604 Electronic Calculator) is shown in Figure 19. Its inputs respond to the "fall" of negative-going signals. It is reset by reducing the grid bias to the right hand triode. Basically, it consists of two cross-coupled inverters so connected that the coupled plate-signal reinforces the input signal, thus "remembering" it.

A quite similar device is the latch shown in Figure 20, developed by E. S. Hughes of IBM and used extensively in the 650. It is really very similar to the trigger since it too uses two cross-coupled inverters. The basic point of difference is that one or both of the inverter outputs are cathode-followed-driven, providing an externally-controllable grid-level input signal, permitting external logic to intervene in the cross-coupling path. The cathode follower also has the effect of practically immunizing the latch against the static instability for which the trigger is notorious. The basic reason for the trigger's instability is its dependence on a high degree of similarity of characteristics between its two inverters.

The current SMS solid-state logic circuitry provides transistor-diode logic blocks which carry "trigger" and "latch" functional labels. However, the CTDL trigger has a two-inverter, two-emitter follower configuration, which actually qualifies it as a "double latch." This ambiguity of terminology is admittedly unnecessary, but is universally accepted temporarily in the interests of standardization and uniformity.

One of the common applications of triggers is the cascade connection called a "ring," which is required to generate sequential timing and/or gating functions. A functional operation in a computer is usually dissected by the logic designers into a prolonged series of sequential steps. For instance, a "multiply" operation consists of:

1. The acquisition of the multiplier and the multiplicand.

2. Repetitive addition of the multiplicand a number of times corresponding to the value of the first multiplier digit.

3. A one-position shift of the partial product with respect to the multiplicand.

4. Another series of repetitive additions of the multiplicand to the partial product, based on the value of the second multiplier digit, and

5. Successive one-position shifts alternating with repetitive additions of the multiplicand until the last digit of the multiplier has been used and the product is complete.

Logical rings or their functional equivalents are required to control the sequencing of these various steps in the operation since computer operations are intended to replace manual or mental operations based on consistent formal rules. These rules are naturally chronological because the human mind can usually do only one thing at a time efficiently.

There are two classical forms of trigger rings which were used in the tube circuits of the IBM 604 Electronic Calculator. The simpler of these is the Overbeck ring, which is shown in Figure 22. It advances one stage for each negative drive pulse. The drive pulse affects the ring by turning off the one stage which is on, whichever one of the "m" stages that happens to be. Next, the going-off of the stage which was on turns on the succeeding stage, as a result of the forward-coupled "going-off" signal overriding the drive pulse which the on-going trigger also "sees."

The main design problem with this type of ring is that the turn-on pulse must be substantially longer than the drive pulse, which in turn must be long enough for the off-going trigger to respond to it. Consequently the ring runs a lot slower than the limiting speed at which the same trigger will operate in logic circuits.

The other type of ring connection of triggers used in the 604 is the inverter-coupled ring shown in Figure 23. The addition of an inverter to every stage of the ring permits the drive pulse to be gated only to the stage succeeding the one which is on, to turn that next stage on. Then the going on of the following stage turns off the immediately preceding stage. In this type of ring the succeeding stage turns off the preceding one when it comes on. The opposite effect occurs in the Overbeck ring in which the turning off of the stage which was on turns on the following stage. It may be noticed that in the inverted ring, the "on" durations of adjacent stages overlap during the transition, while in the Overbeck ring there is a gap between them. In some cases this difference may be very significant to the designer.

Due to the similarity of function between the "trigger" and the "latch," a machine like the 650, which uses latches exclusively as logical memory devices, also uses latches in its rings. Because of the very stringent checking requirements of the 650's design criteria, the only rings used in it are the timing rings. These are driven by drum-derived pulses and checked against each other systematically.

Since the latch is basically a trigger with a cathode follower inserted between the plate of the second inverter and the grid of the first, the latch ring is quite similar in action to the Overbeck trigger ring in that the drive pulse, which is now a logical "reset" pulse, turns off the latch which is on, and the going off of that latch generates a negative signal which is capacitively coupled to the succeeding latch to turn it on, as shown in Figure 24.

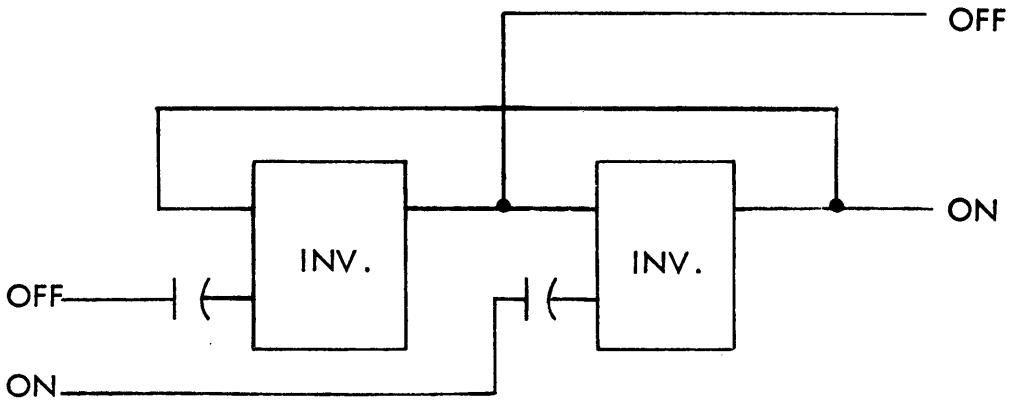


Fig. 19 - Trigger

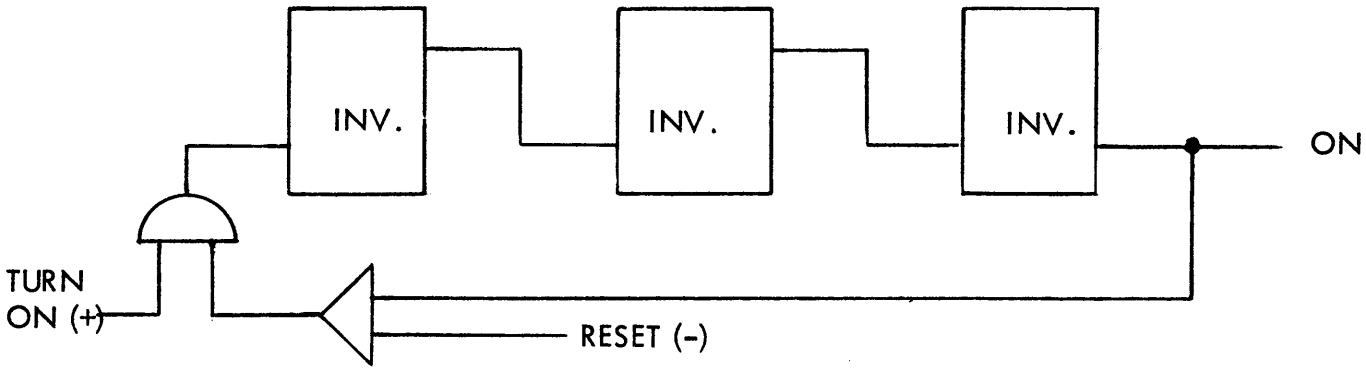


Fig. 20 - Single Latch

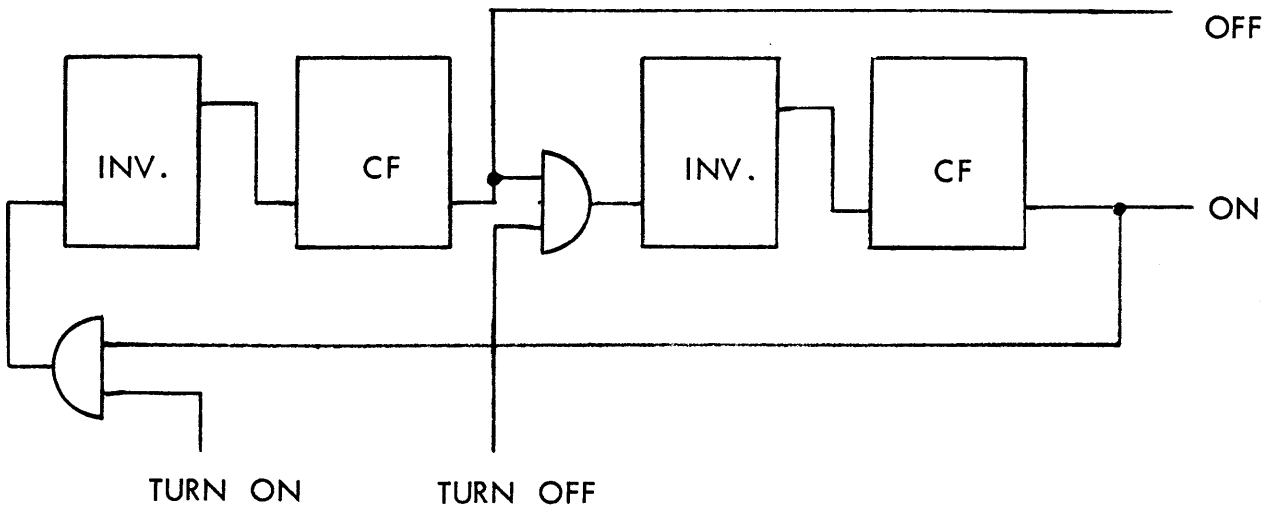


Fig. 21 - Double Latch

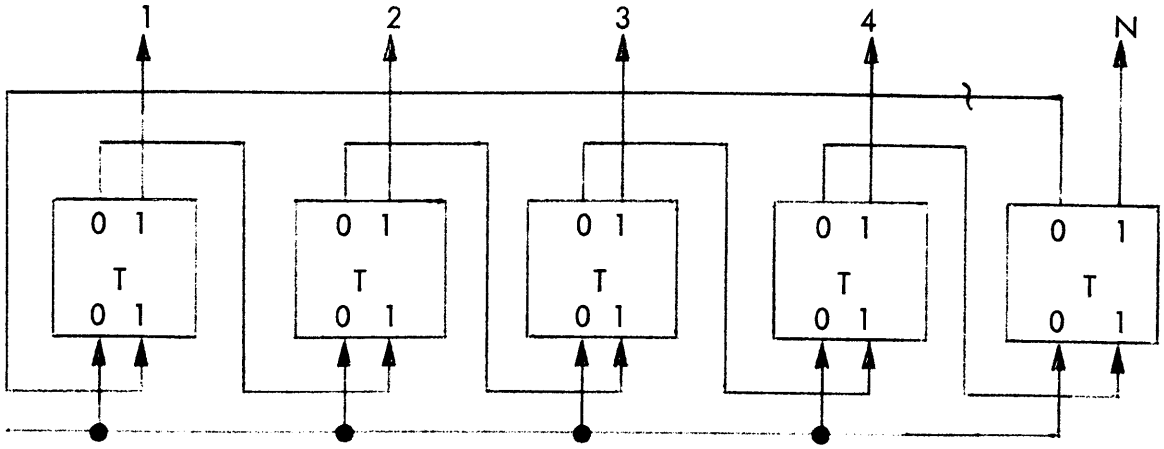


Fig. 22 - Overbeck Ring

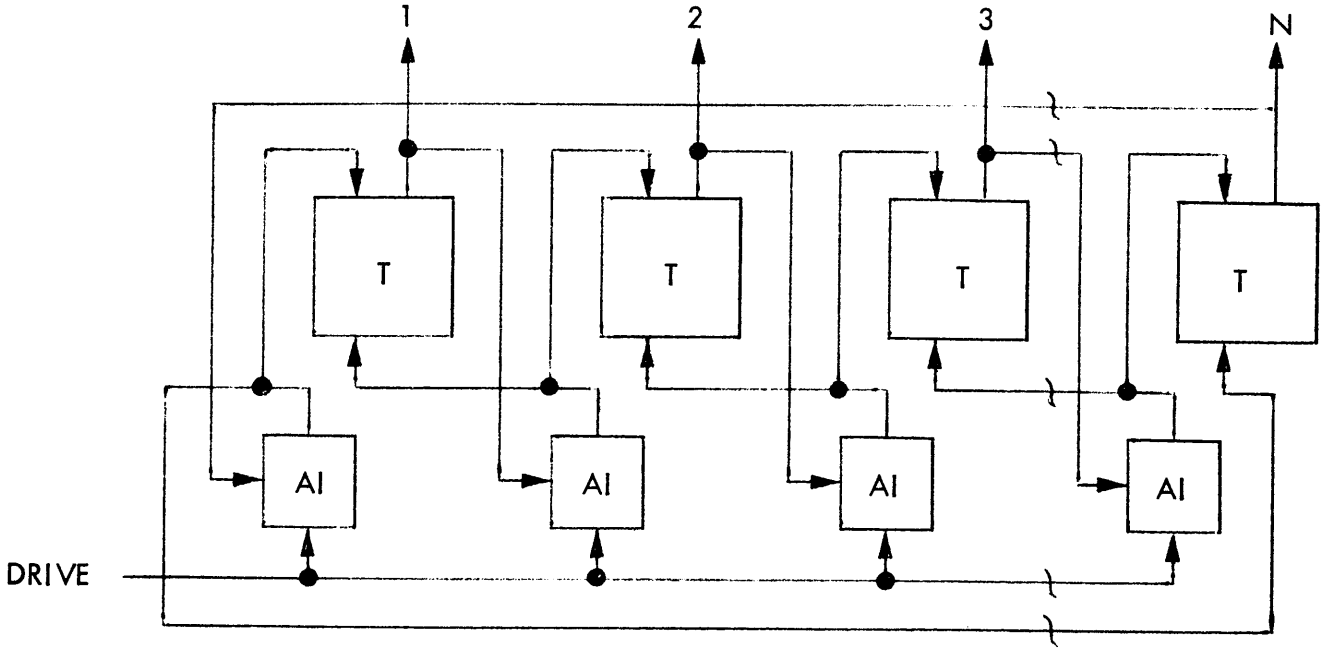


Fig. 23 - Inverter-coupled Ring

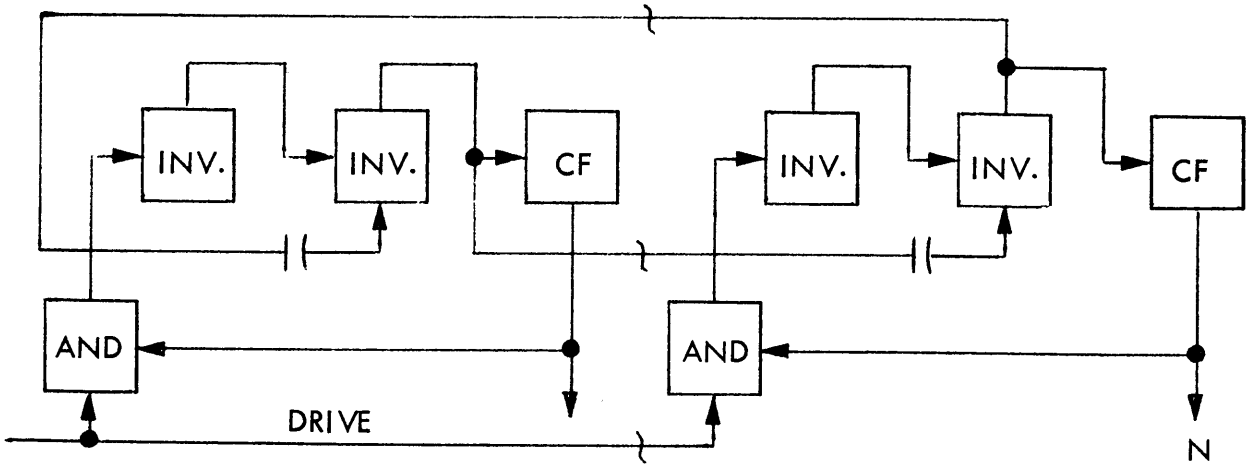


Fig. 24 - Latch Ring

XVII MAGNETIC DRUM STORAGE

The only local application of a magnetic drum as main storage in a computer which is of immediate interest is the 650, to which this discussion will be largely restricted.

The characteristics of the 650 drum are as follows:

1. Speed — 12,650 rpm.
2. Size — 4"D x 16"L.
3. Number of tracks used — 228 (200 for general storage, 14 for buffer storage, and 12 for timing tracks including a spare set of 6).
4. Track spacing — .045" axially.
5. Bit density — approximately 50 bpi.
6. Bit rate — nominally 125 kc.
7. Recording method — discrete spot.

The drum has an information capacity of 2,000 words, each word consisting of 11 five-bit digits. The physical addressing of a word on the drum is based on cylindrical coordinates. There are 40 bands distributed axially along the drum. Each band contains 50 words distributed angularly around the drum. Therefore, a drum address must be separated into its band (axial) and word (angular) components.

The group of five read/write heads which serve the desired band are selected by logical switching circuits specifically designed for the purpose, since they must operate at "non-logical" signal levels. This selection is accomplished by electronically raising the voltage level of the "read" winding of the drum head to "unblock" a vacuum diode and permit the signal induced in the winding by the moving flux from the drum to reach the voltage amplifier. This amplifier raises the weak signal from the head up to an amplitude of 20-30 volts, which then trips a Schmitt trigger (pulse generator or single-shot multivibrator) to product a pulse of sufficient time duration to be consistently sampled by a timing pulse, to turn on an "output" latch which develops a six-microsecond "information gate" as the logical output of drum storage information. These latches operate continuously as long as a drum address is in the address register. Thus all fifty words in the band are available during the course of a drum revolution. The problem of "angular" addressing then becomes one of dividing the periphery of the drum into 50 equal arcs. This has been done by dividing the drum period first into five sectors and then each sector into 10 words. This is done by providing a "sector ring" of five active stages and a "word ring" of 10 active stages.

There are obviously 50 unique combinations of sectors and words. The sector ring makes one complete cycle every drum revolution, the word ring makes a cycle every sector "time" of 1/5 drum revolution. Because a specific combination of sector and word repeats at a consistent point in every drum revolution, that combination of timing gates may be used as a specific dynamic angular address, singling out one word of the 50 from the band whose five heads are statically selected by the "band" component of the address.

The basic timing considerations involved in understanding the 650 drum are illustrated in Figure 25. The drum speed of 12,650 rpm provides a drum period of 4.8 milliseconds. A sector time is 1/5 revolution or 960 microseconds. A word time is 1/10 of a sector time or 96 microseconds. A digit time is 1/12 of a word time or eight microseconds. The use of the digit ring will become evident in the later review of capacitor storage. The digit time is further subdivided into four two-microsecond periods labeled A, B, C, and D time. The time-interval build-up then is 2 x 4 x 12 x 10 x 5 = 4,800 microseconds or 4.8 milliseconds per drum revolution.

The read/write technique used in 650 drum storage is as follows:

1. Information is sent to the drum in the form of six-microsecond gates which begin at the beginning of B-time and end at the beginning of the next A-time. The middle third of the gate is sampled by the Write Sample Pulse at C-time. The fall of the sampled output generates a write-current pulse via a current-pulse generator, which is connected by the band-selection switching to the record winding in the head. This winding contains 40 turns, and the current pulse is about 150 milliamperes, producing an mmf of about 6 ampere-turns for about one millisecond, during which time the drum moves angularly about .0025 and for the sake of simplicity may be considered stationary.

2. The above current pulse produces magnetic flux in the core of the head which contains a short air gap normal and adjacent to the surface of the drum. The head-to-drum spacing is on the order of .0015 so the gap flux fringes into the drum's magnetic coating (a nickel-cobalt alloy with very high remanence). After the write pulse terminates, a "spot" of magnetic flux will remain on the drum until the succeeding "write" operation. This flux-spot may be thought of as the equivalent of a tiny bar-magnet inlaid in the drum surface. The geometry of the situation is such that the spot is about .010 long, and successive spots have an interval of .020. "Erase" spots have the same flux density as "record" spots, but opposite polarity. Also, the background magnetic "bias" of the drum is weakly in the "erase" direction. The resulting flux pattern for several adjacent spots in the same track is shown in Figure 26.

3. At some later time during the course of the computer program, the recorded information is called for, to be read from the drum. The current induced in the read coil of the same head that "wrote" the flux is the time-derivative of the residual flux, and the resulting induced voltage wave form is shown in Figure 27. The approximate transfer characteristics of the drum and its read/write circuitry are best described by noting that the peak power during "write" time is the above-mentioned 150 ma times a peak voltage of between 30 and 50, say six watts peak power. The peak power developed during read time, assuming that the entire negative peak to positive peak signal slope is effective, is at least $(200\ mv)^2$ divided by 100K ohms, the value of the grid bias resistor in the voltage amplifier, or about 0.4 micro-watt. The power ratio, output to input, is on the order of 1/10,000,000, or -120db.

   The significance of this observation is that this is a measure of the overall power-gain requirement of the read/write circuit, stated in terms which may be readily appreciated by anyone with electronic circuit design experience.

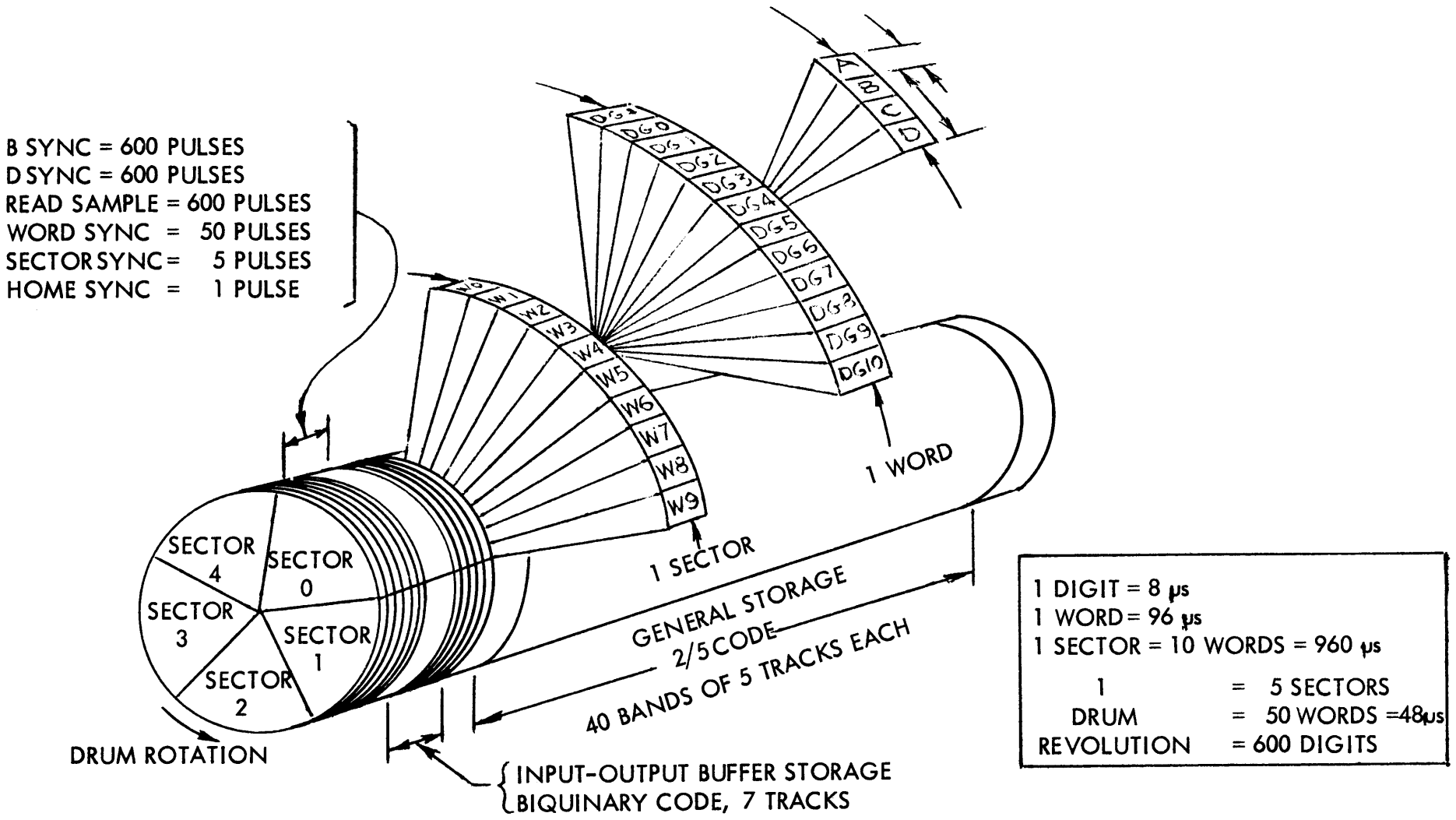


Fig. 25 - IBM 650 Drum Sectors and Timing

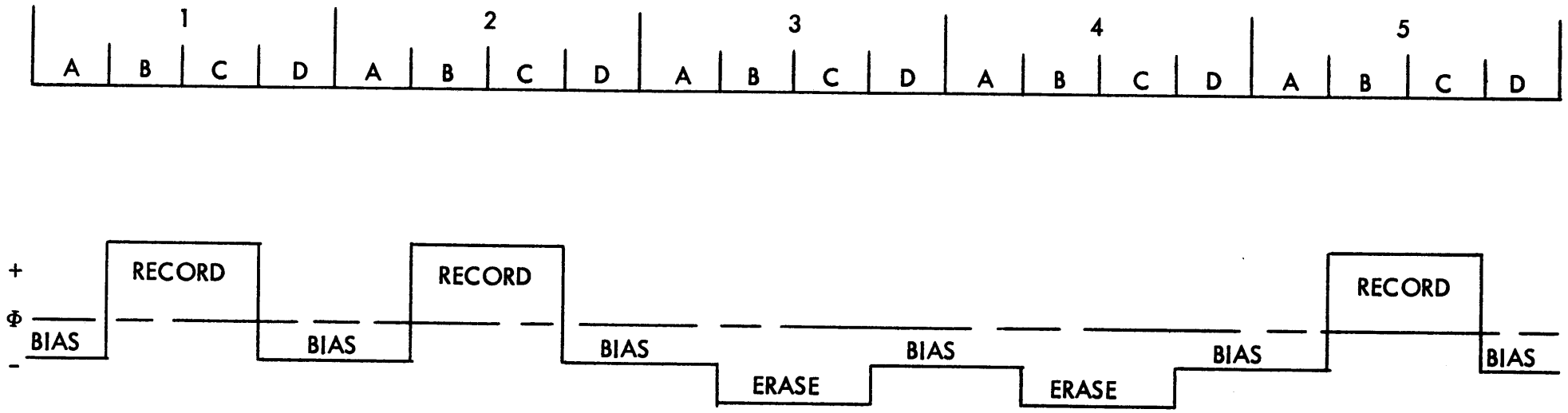


Fig. 26 - Residual Flux Pattern for Several Spots on Adjacent 650 Tracks

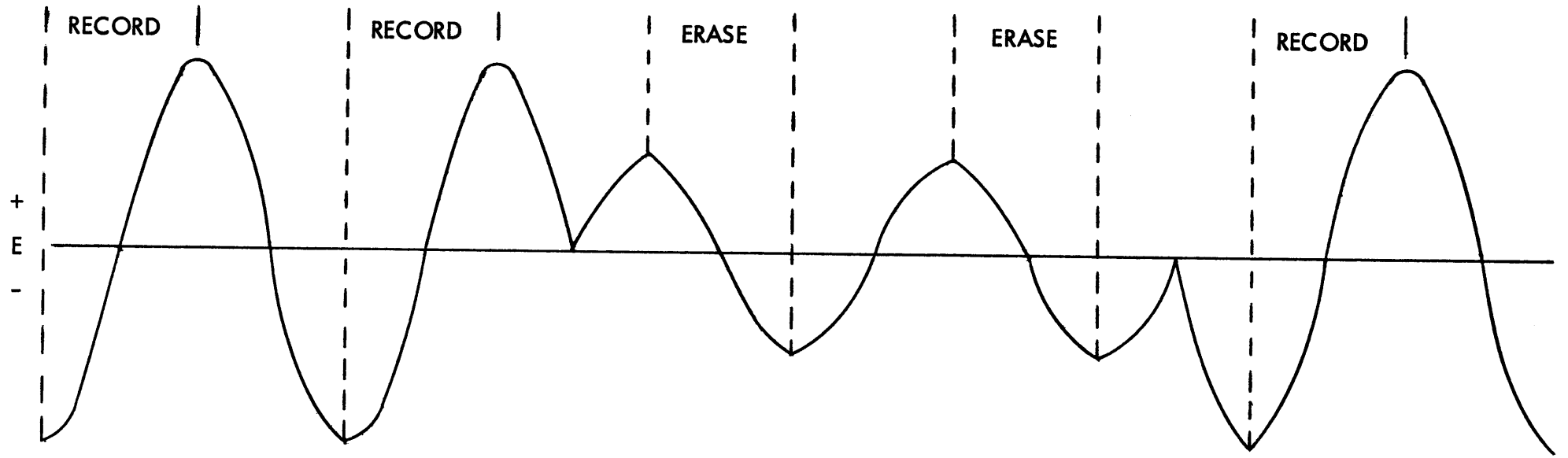


Fig. 27 - Induced Voltage Wave Form of Adjacent 650 Tracks

XVIII CAPACITOR STORAGE

Capacitor storage is used for the three high-speed storage registers of the 650 — the Program Register, the Distributor, and the Accumulator, which have information capacities of one, one, and two ten-digits-plus-sign words, respectively. These registers are in the program and arithmetic control section of the 650 system. As such they handle information in the bi-quinary code, as previously mentioned.

For the data flow logic of the 650 system, refer back to Figure 3. As this figure illustrates,the registers just described are essential units without which the system would be incapable of computing. These registers are the unique production-system application of capacitor storage in IBM, and certainly merit a study of their operating principles.

The essential components involved in the implementation of one "cell" or bit-position of capacitor storage are shown in schematic form in Figure 28A. Binary "zeroes" are stored in a low-loss ceramic capacitor (500 mmf) in the form of a charge of nearly 50 volts, static. However, this charge will not remain in even the best capacitor for an infinite period of time. Thus it becomes necessary to periodically regenerate this charge to restore it to its desired voltage level.

There are only two ways to sense this charge as information. The more straightforward of the two ways is to "look" at the charge with the grid of a class A vacuum tube. However, with a signal swing of nearly 50 volts, this is impractical since vacuum tubes normally have a cutoff to zero bias range of just a few volts, with the exception of the low-nu power triodes (2A3, etc.) and the beam power pentodes, both of which present other serious design problems. The alternative method ofsensing the charge on the capacitor is to attempt to charge it through a resistor and sense the voltage drop produced by the resulting current "spike."

Such a method is used in the 650. Its primary side effect is that charging the capacitor will place a binary "zero" in it if the capacitor had had a "one," necessitating the removal of the charge before the information it represented is lost. This is done by immediately, not instantaneously, discharging the capacitor.

Naturally the capacitor cannot be charging and discharging at the same time since this would presume current simultaneously flowing in both directions. What is required is a short time delay between "readout" and "regeneration." This is performed by what amounts to a two-stage open-ended latch ring. During readout, the information is stored in the first-stage latch, which is turned on by the amplified readout spike. Eight microseconds later, the information is transferred to the second-stage latch by resetting the first stage. This latch then has its output sampled by a timing pulse after it has stabilized to recharge the capacitor.

Let us consider now the details of Figure 28. The triode cathode follower on the extreme right is pulsed every time the capacitor should read out. In the distributor and program register, this is once a word time during a specific digit time. As the CF output voltage rises, it unblocks the right-hand vacuum diode and provides a circuit to charge the capacitor from +150V to -70V via points A, C, and D in the sketch. The "1" is sensed as a position spike on the OUT line. During the immediately-following digit time

the inverter at the extreme left goes into conduction; its plate voltage drops and unblocks the left-hand diode. If a"1"is to go back into the capacitor, the output of the CF driving point D comes up at the same time, literally "squeezing" the charge out of the capacitor from -50V via points B, C, and D to +150V through the CF. If the capacitor is not discharged, it will not recharge during the next readout operation, resulting in no output spike. This produces a binary "0" by not turning the latch on. The lower part of the figure shows the voltage signals at the labeled points in the circuit.

Figure 29 is merely a demonstration of how these capacitor storage cells are combined into a matrix, permitting the driving, sensing, and storage devices to be time-shared by many positions. If this were not done, capacitor storage would be hopelessly uneconomical since the necessary part of the job could be done with just one of the two latches in the delay circuit with no capacitor at all. In the 650 matrix, the only items that need be repeated 77 times (seven bits x 11 digits) are the capacitor and the pair of diodes (a 6AL5). This part of the circuit is packaged two sets to a one-socket preassembled pluggable unit.

Capacitor storage has been regarded traditionally by experienced computer designers as unfeasible commercially because of the charge-leakage problems and the drive requirements. But so was the trigger back in the days of Eccles and Jordan when triodes were hard to match in production quantities. Be that as it may, capacitor storage has been a success in the 650.

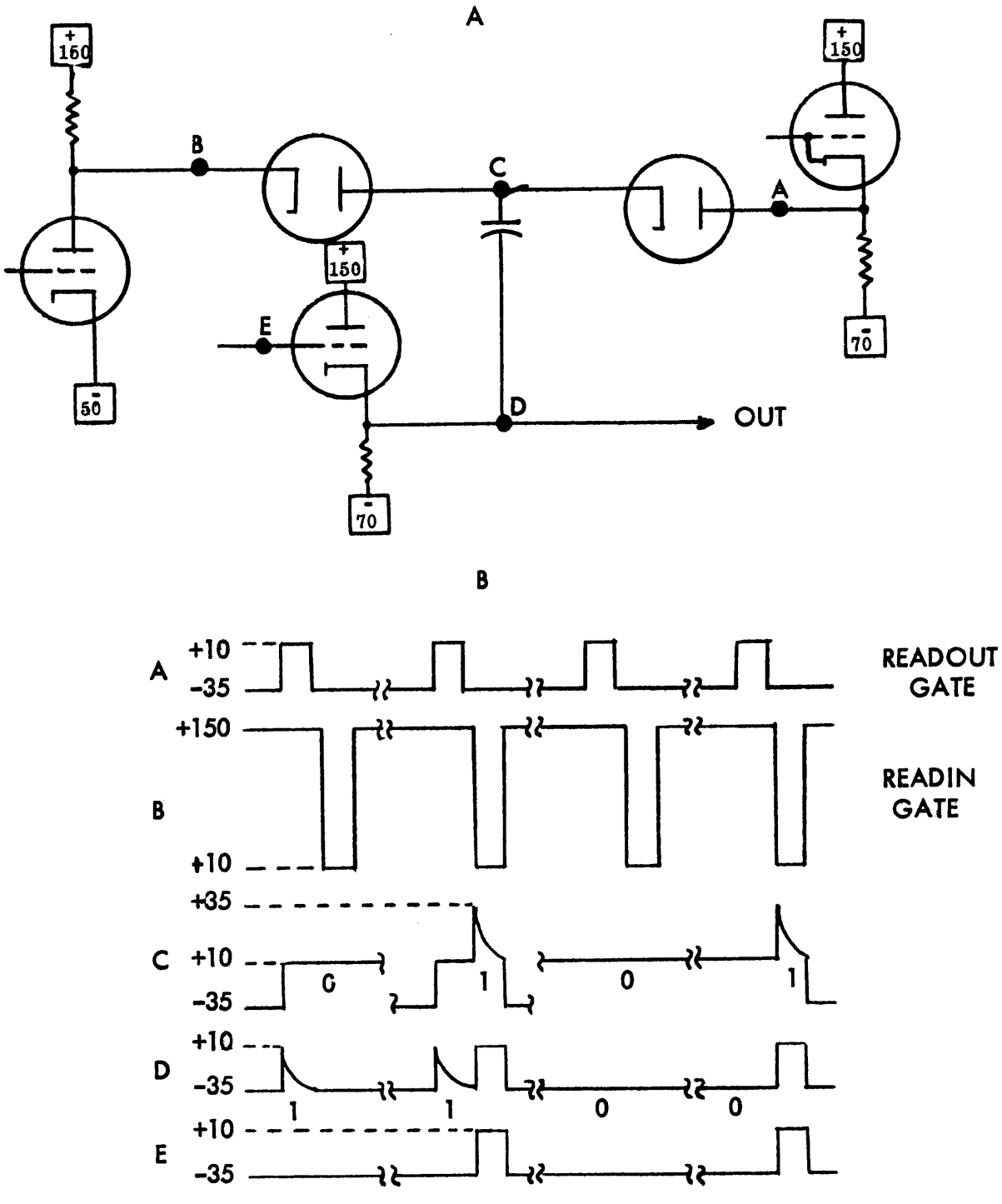


Fig. 28 - Capacitor storage

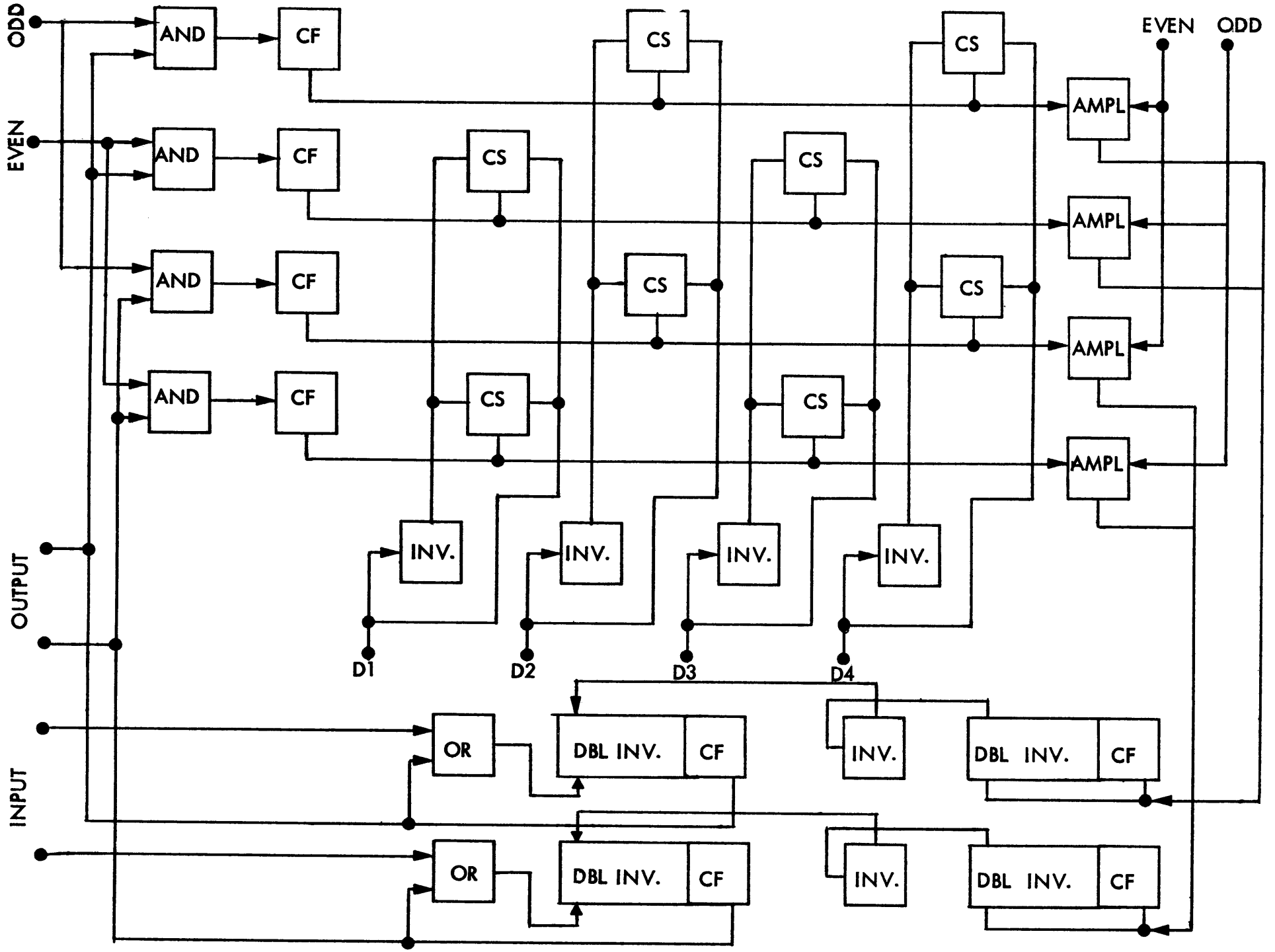


Fig. 29 - Capacitor Storage Cells Combined into a Matrix

XIX FERRITE CORE STORAGE

The 650 uses a magnetic drum as storage for its coded instructions, input/output data, and intermediate results. Most other modern computers, both in and outside IBM, use a box-shaped array of very small magnetic cores of one of several of the manganese ferrite alloys which are usually manufactured by a sintering (compression and heat-fusion) process.

A ferrite core in wide use in large memory arrays today is .030"ID x .050"OD x .025" thick, roughly the shape of the beads used in children's handicraft work. The arrays are generally fabricated by stringing a two-dimensional group of cores on the various windings which pass through them, to form a "core plane" which normally contains the same specific bit of all the addressable characters in either the entire memory or, in the case of a very large one, a simple fraction of it (such as 1/4 or 1/5). The completed plane is contained in a carrier usually called a "picture frame," which is what it most resembles. Then the required number of frames are stacked (like hotcakes) and the peripheral terminals of the individual frames are cross-jumpered to complete the three-dimensional array.

Probably the most fascinating characteristic of a ferrite memory array is the method of selecting the required address (a single character, a consistently-sized group of characters, or a fixed-length word) from the entire array by means of electronic current-pulses. The success of this technique depends on the peculiar shape of the hysteresis loop of the ferrite used for the cores. As shown in Figure 30, it is almost square, with sharply-defined second and fourth quadrant knees. The points of the loop, in the first and third quadrants, are quite pronounced and correspond to the saturation flux density. When one of these cores is heavily overdriven, these points become horizontal lines extending out until either the wires burn up from excessive current or the switching losses alter the shape of the loop by raising the temperature of the core.

The basic principle involved in address selection is called half-current coincidence. Referring again to Figure 30, it may be noted that perceptibly more than half the saturation mmf is required just to drive the flux density beyond the knee of the curve — in other words, to change its value substantially.

Now refer to Figure 31, which shows a small section of a hypothetical array and the routes of its windings. The two drive windings are usually designated as X and Y, after the Cartesian coordinate axes. It is evident that if there is approximately one-half of the saturation-mmf current in one each of the X and Y windings, only one core in a whole plane will "see" full critical switching currents, the amount necessary to change the polarity of the remanent flux in that core. $(N_x + N_y - 2)$ other cores in the plane will "see" half the critical current. The remainder of the plane will "see" no current at all.

Suppose now that the current in the drive windings is in the direction through the core which will reverse the remanent flux if the core is at the moment in the state of flux-polarity defined as "1". The drive currents will co-exist in only one core of the plane, and will reverse the flux polarity after the usual short time-delay required to re-align the magnetic demains in the core, which also is a function of the total driving current.

As the flux reverses polarity, this flux change induces a small voltage in all of the windings through the core, including the drive windings. One of the other windings is provided to specifically sense this weak signal, and is called the sense winding. Its route is so chosen that the noise voltage produced by the half-selected cores, which this winding also picks up, is canceled almost completely by reason of the fact that half of the half-selected cores induce a voltage opposite to that induced by the other half.

This constraint is a powerful one, and makes the sense-winding pattern design a real achievement.

Summarizing, the reason for the noise voltage from the half-selected cores is that the top and bottom of the hysteresis loop are not quite level. Even though the slope is very small (and consequently the induced voltage from any one core is small), the fact that many cores are half-selected — and that the sense winding is the electrical equivalent of so many transformer secondary windings in series — results in these noise voltages superimposing into what would be a substantial spike (which could and would swamp the desired readout signal) without cancellation.

The readout signal, which because of the noise-cancellation requirement may have either polarity of voltage, is full-wave rectified via a high-efficiency pulse transformer and push-pull diodes. The resulting rectified signal, which is usually also stepped up through the transformer (typically 3:1) is then electronically amplified. This signal is finally "stretched" by a trigger or latch into a useful information signal which may then be prolonged into the "write" portion of the memory cycle. This cycle is normally adjacent in time and of about the same duration as the "read" portion.

At this point, it becomes important to review the address selection drivers. It has been found desirable through experience to use a large square-loop core to drive the ferrite array. One of the direct benefits is temperature compensation since as the array heats up its cores switch more easily and at the same time the output of the driver core falls off accordingly since it also takes less energy to switch. Almost as important is the fact that the driver cores can be specified to require an accurately-defined amount of energy to switch them. This consistent amount of energy generates a constant pulse of current, which is just what is required for accurate and reliable coincident-current selection. These conditions are much more a result of effort than ingenuity, and a lot of research into the significance of switch core parameters is still in progress.

Having described the driver switch-core, let us now review it in more detail. This core, which obviously is driven by a power-switching device such as a tube or transistor, has a square hysteresis loop and thus behaves similarly to the small cores in the array. In other words, it switches during "read" time, generating an output current pulse of a certain polarity. With a separate core obviously required to drive each of the select lines in the array, one core in each dimension of drive will have been "set." Therefore, all of the driver cores can be reset by auxiliary windings on them, all connected to a common reset driver with appropriate de-coupling, which "sees" only the load of the one core which was set, in the form of back emf and reset drive current.

Thus, during write time, the driver switch-cores "remember" which core was used during read time. This is a great convenience in a lot of applications since the

actual memory address is required only during read time, permitting controlled replacement of the "old" address with a "new" one during write time, enabling the memory clock to operate continuously, for maximum possible speed.

The remaining problem is that of how to get the information back into memory after its destructive readout (during read, all selected cores were driven to their "0" state). Since the information read out during read time was stored temporarily in triggers or latches, it may now be sampled by the memory clock during write time. To place the information back in memory requires the use of the fourth winding shown in Figure 31, the "inhibit" winding. If the result of the coincident read current was to flip all the selected cores to "0", the write current being in the opposite direction through the same windings would naturally reverse all the selected cores to their "1" state, which is not desirable.

This problem is solved by passing half reverse (read-polarity) current through the inhibit windings for those bits of the selected information which are zeroes, to "inhibit" the ones which would otherwise result. These three currents add algebraically, and net out to half-current in the zero cores and full-current in the one cores for the selected address, no current in the half-selected cores of the zero planes and half current in their one planes. Finally, the unselected zero cores see reverse-half-current and the ones no current. This satisfies all the functional requirements for a successful core memory, as we know it today.

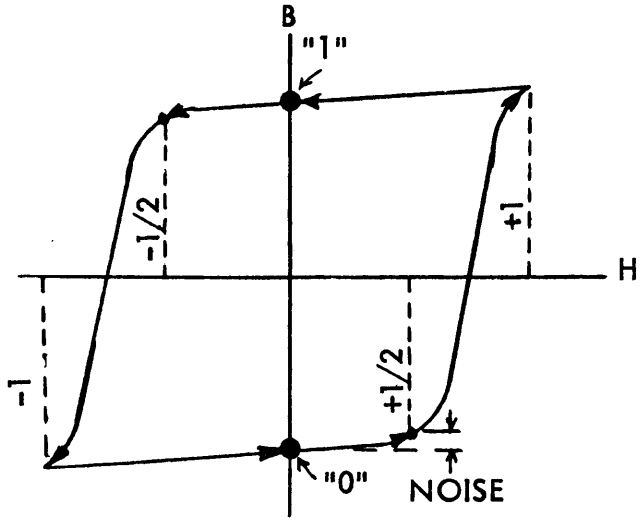


Fig. 30 - Ferrite hysteresis loop

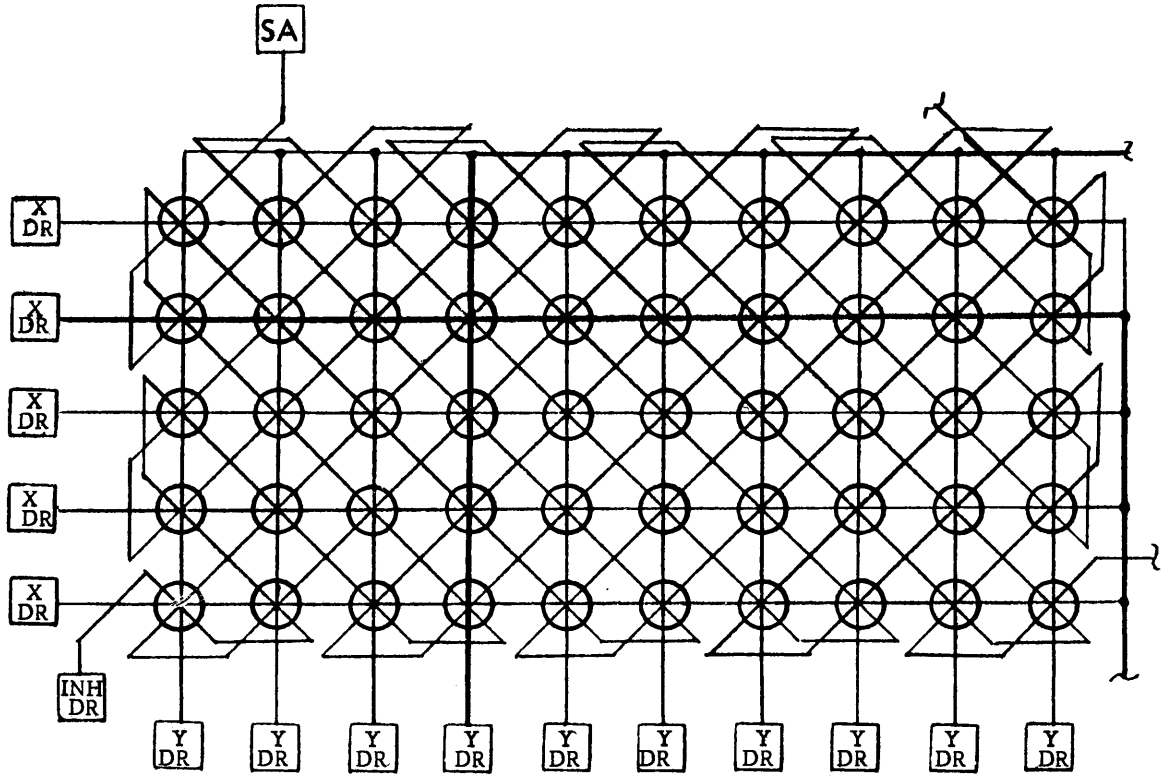


Fig. 31 - Ferrite memory plane

XX SWITCH CORES AND CORE LOGIC

Switch cores of the general type used as ferrite memory drivers perform a potentially useful memory function in remembering during "write" the address which was used during "read." In storage this characteristic provides the advantage of permitting the controlling address to be replaced during the write operation, since it may only have been required to "set" the desired switch cores during the read operation, the switch cores themselves addressing the memory during write as they are reset by a common unsteered pulse. This characteristic has been thoroughly utilized in the core buffer-register system of the 7070.

The basic characteristics of switch-core storage registers will be reviewed, as well as the unique functional advantages of a system of registers of this type.

The switch core, as a functional unit, will store one bit of binary information, which is also the natural capacity of a trigger or latch. However, unlike those feedback-controlled logical memory devices, the storage function of the switch core is provided by the energy contained within its magnetic hysteresis loop. This energy is required to drive the magnetic state of the core from maximum remanent flux in one direction to maximum in the other and back again. This is the same characteristic which is utilized in ferrite memory cores.

The real difference is in their relative switching power levels. Ferrite cores, especially in large arrays, are wound with single-turn windings literally "sewed" through the plane of cores. On the other hand, switch core memory devices normally contain multiple-turn windings to limit the current requirements to reasonable values (since they are physically larger, they require more switching power, and also accommodate more through-wires) and to provide higher output voltages.

The standard core used in the 7070 register circuits is .050"ID x .080"OD x .125"L, consisting of seven wraps of 4-79 Permalloy .000125" thick wound on a ceramic bobbin having about a .030" hole for the windings. A typical core, as will be detailed later, may contain 60 or more turns of very small size wire in its windings. The switching voltages, as a result of the relatively large energy and the multiple-turn windings, are on the order of 20 to 40 volts with less than a microsecond pulse-width.

The basic logical device, in which the core is a component, is a single position of a variation of the well-documented single-core-per-bit shift register. The various basic core shift register circuits are shown in Figure 32. Figure 33 details the circuit of one bit in a 7070-type core register which is packaged on a conventional SMS circuit card. The circuit shown contains four input windings and their diodes, three output windings and their transistors, and a drive winding.

The basic principle involved in the operation of a register switch-core is that if the core contains a "one" as a result of a previous input, the next pulse through the drive winding will reset the core, producing an output pulse via each of the output windings. The "diodes" formed by the base-emitter junctions of the output transistors limit the pulse amplitudes by damping their currents, making the intervening limiting resistors necessary.

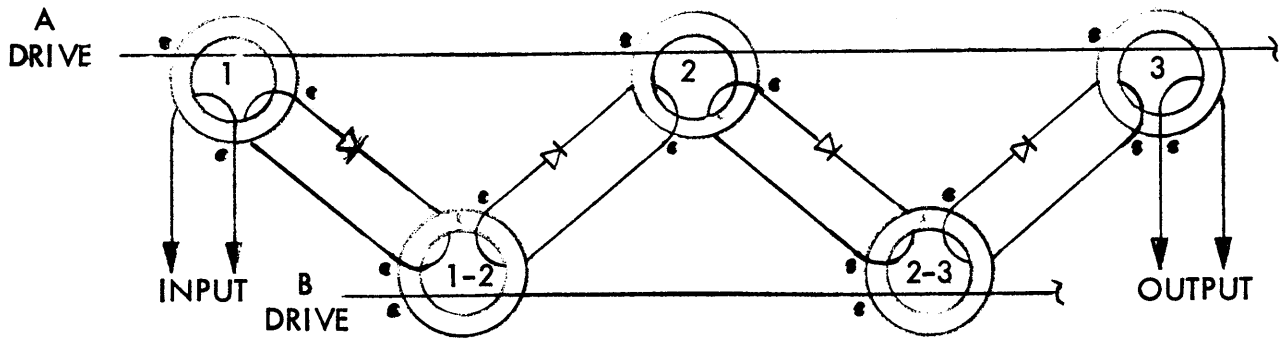
The pulses from the respective output windings turn on their associated transistors, whose collector outputs are capacitively loaded, typically with 2,000 pf. shunted by 27K ohms. The decay time constant for the charges the transistors put on the capacitors is many times longer than the period of the drive pulses, which are not gated and repeat every four or six microseconds. Since the core is being continually reset, its contents must be regenerated. Regeneration is usually accomplished by connecting one of the output circuits directly to one of the input circuits. The illustrated circuit does this internally within the card, to avoid using up any of the connector terminals since the regenerated information is not required externally.

The other input and output windings shown are used for inter-register communication since the registers are used functionally for the transfer and temporary storage of information, during the operation of the system on its stored program.

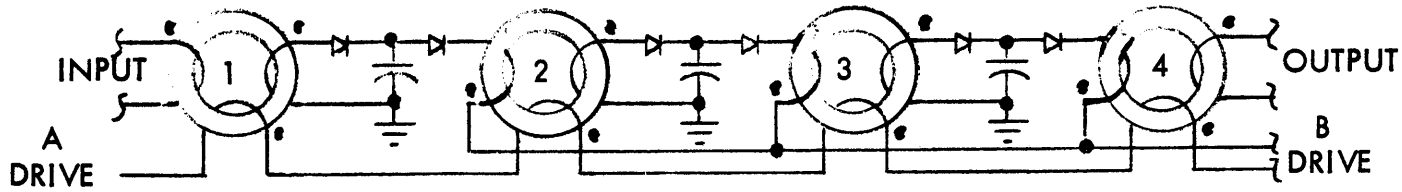
Figure 34 illustrates in simplified form how two registers can communicate with each other bilaterally via a single channel, with each register at the same time being able to read in and out serially to the right, independently. Since regeneration and shifting are mutually exclusive functions, they may share the same output winding as shown, the choice between the two functions being made by selection of the desired input winding.

The maximum number of windings provided for any of the registers in the 7070 is eight on each core, to permit parallel readin and readout two separate transfer channels, shift right, and regeneration. The parallel in/out paths each require two windings, shift and regeneration three more, and drive the remaining one.

The Auxiliary Register (see Section IV, Figure 4) requires such flexibility, to transfer in parallel via both the Arithmetic and Information Busses and serially to and from the Adder, as well as regenerating itself during indexing and certain other operations. The inherent advantages of the core registers in the 7070 system are low-cost parallel transfers (a ten-digit-and-sign numeric word in four microseconds) and simple read-in/read-out controls (a 53 bit register is controlled by a single switching transistor which selects the appropriate input winding on all the cores of a register at once). This makes possible a very elaborate register system, in comparison to the investment in circuitry required to provide the equivalent functions with triggers or latches and the associated turn-on and sample-out logic. The attendant drawbacks, which have been acceptedly compensated for in the design, are radiated electrical noise (produced by the severe current transients in the power supply distribution lines) and a fairly slow limiting speed of operation. Due to this speed limitation, this type of register is probably obsolescent, although switch cores operating in the megacycle range are available and in experimental use in IBM.

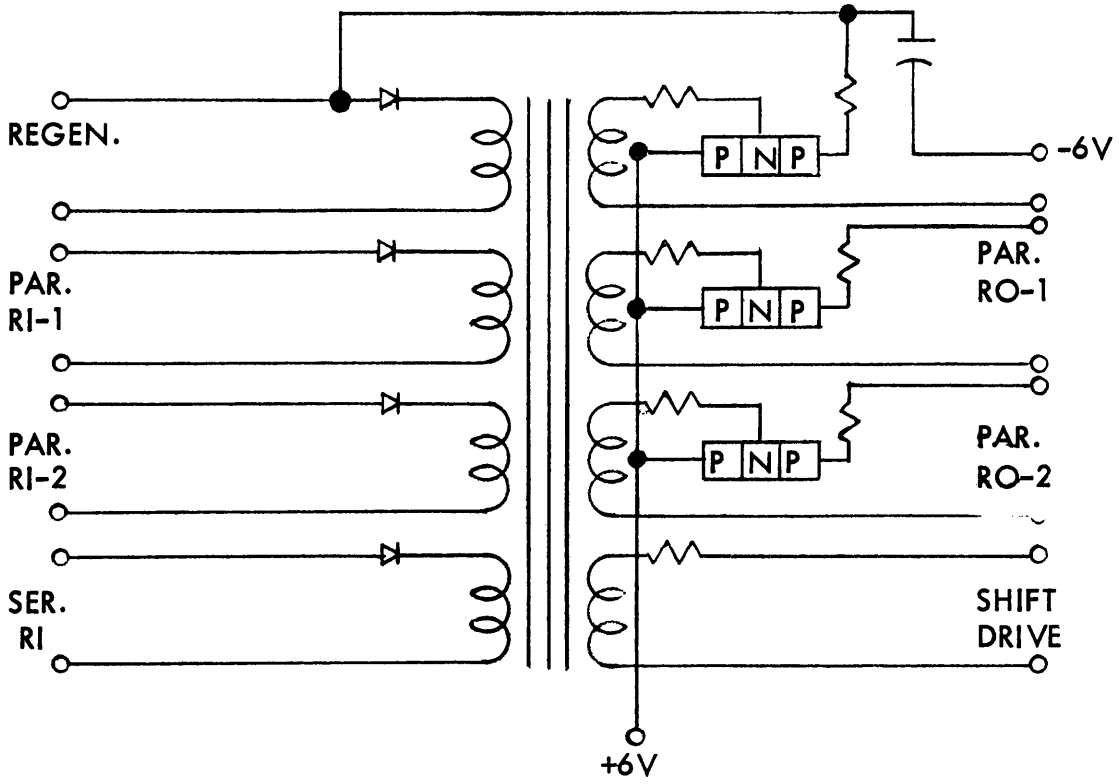


Two Cores Per Bit Shift Register



One Core Per Bit Shift Register

Fig. 32 - Basic Core Shift Register Circuits



Switching Core Register Circuit

Fig. 33 - Circuit of One Bit in a 7070-Type Core Register

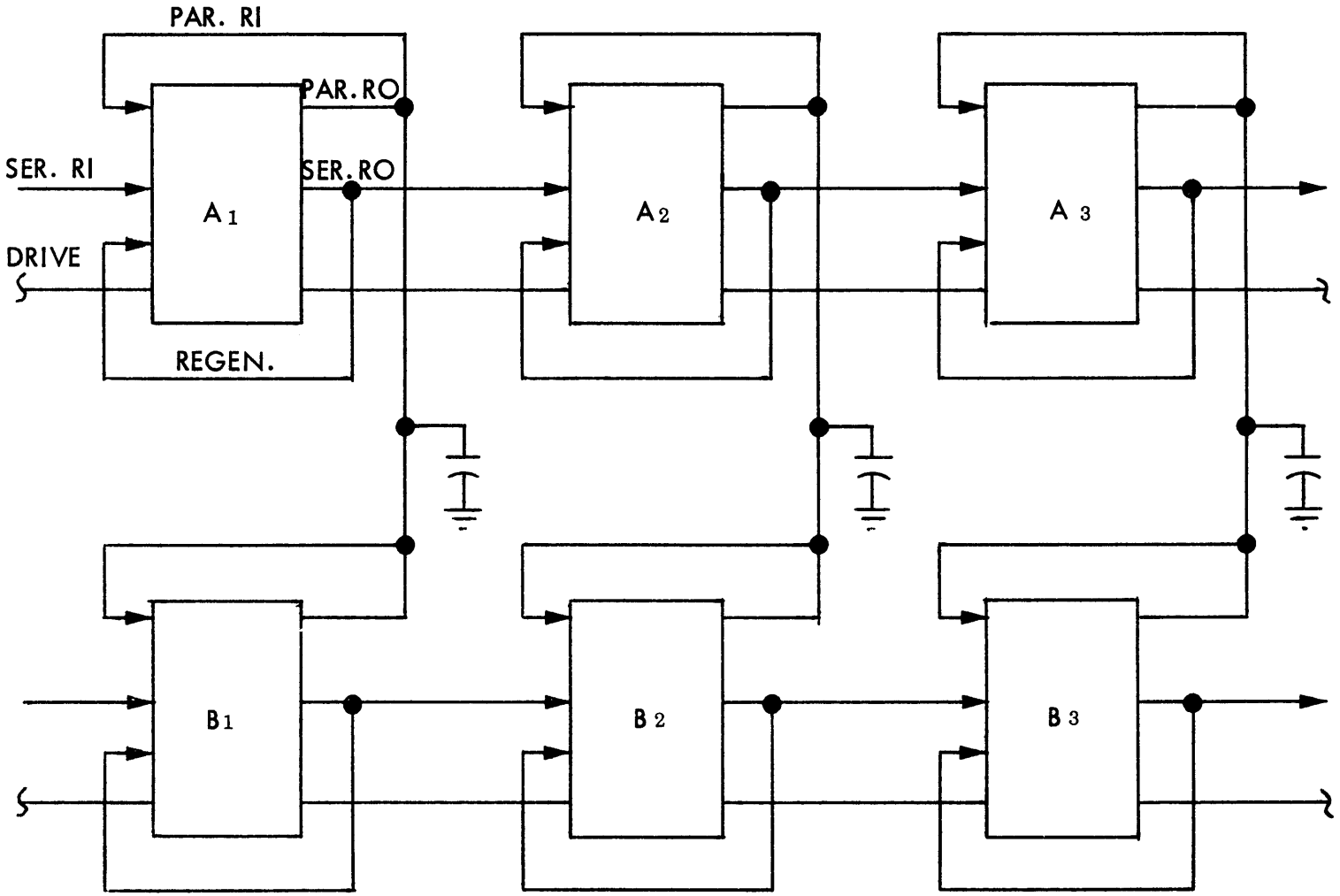


Fig. 34 - Core Register System

XXI CARD READERS AND PUNCHES

Conventional punched cards are a common input/output medium for data-processing systems. This is true for a variety of reasons. For example, most users of data processing systems are necessarily fairly large business organizations, to have the requirement and economic justification for their own full-time data processing equipment. In the competitive struggle of today's business world, economic health is signified by a process of continual growth. This growth would be inhibited to the point of stagnation if it were not for the availability of punched-card-operated cccounting systems of the conventional electro-mechanical type.

As a case in point, consider the complexity of payroll accounting — with its myriad of fixed and variable deductions (some of which are required by law) — for a typical industrial corporation with a manpower inventory of, for example, 50,000 men. The clerical staff required to provide just this one service on a manual basis would not even be available to train, much less have the necessary business experience, if all companies of that size in the nation were also competing for the same kind of manpower in a common labor market. Even more serious, from a financial point of view, would be the cost of supporting such an accounting organization. Instead of the prevailing ratio of accounting costs to gross business income of 5-10 per cent, these costs would probably be so high as to consume much of the profits, and force the company that tried to grow with its market right out of business.

IBM itself is a prime example of how mechanized accounting fosters corporate growth. Since accounting equipment is out stock in trade, we believe in it implicitly to the extent of being one of our own best customers. The result is that, by the commonly accepted yardsticks of corporate value, IBM has doubled in size every five years since its founding in 1914; This means that, in the intervening 45 years, we have doubled nine times to increase roughly 500 times. If this growth rate continues, IBM will probably reach its "natural" limit within a decade or so.

Other companies, those which are the giants of industry now, would be ponderously inefficient and hopelessly unproductive if it were not for their huge investments in mechanized accounting systems as well as the other forms of automation they now utilize.

The purpose of this discussion has been to set the stage for an exhibition of the peripheral card equipment used for computer input/output. It is rational to assume, since computers come in a limited range of finite sizes, that even branches and small divisions of the very large corporations can not afford them. This consideration also applies to the separate small accounting applications in their central offices. As a result, a substantial part of the total accounting work in a large corporation is still being done on conventional electro-mechanical accounting machines which utilize punched cards as their common processing medium. Thus there is a natural need for a computer, even a very large one, to be able to communicate with these satellite punched card systems in their native tongue, the punched card.

The limiting speed of a punched card accounting system is pretty well set by the mental and physical speeds of the human links in the chain of operations. On the other hand, electronic computers and data-processing systems are speed-limited only by the prevailing status of technology, which has improved all along at a much faster rate than human efficiency. Today a computer can do the work of thousands of accountants and mathematicians, and more quickly and accurately. As a direct result of their greater speed and computing power, electronic systems require higher input/output speeds in their peripheral card equipment as well. Consequently, card readers and card punches are continually being improved in speed as well as in function.

A. IBM 533 READER PUNCH

Figure 35 shows the feed schematics of the IBM 533 Reader-Punch used in the IBM 650 Data Processing System, the only card machine originally released for use with the 650. The reader section operates at a speed of 200 cards per minute, and the punch at 100 cards per minute. These speeds are sufficient to match the internal processing speed of the 650, which is based on now-obsolete electronic technology and market requirements. The particular feed mechanisms used in the 533 were chosen for two basic reasons — because they were available, and because their reliability had already been established in electro-mechanical applications.

B. IBM 537 COMPUTING PUNCH

A later development as a 650 attachment is the IBM 537Computing Punch shown in Figure 36. It feeds cards at 155 per minute and is designed to punch results back into the same card from which the input information was read. This requirement is basic to public utilities accounting since the card which contains meter readings usually has room for the billing amounts as well, and will only be used for one accounting cycle. Once the bill is paid, the card may be filed and forgotten.

In these illustrations the following system of abbreviations is used:

H - hopper,
R - read station,
P - punching station,
B - blank station (normally required for mechanical reasons),
and S-stacker.

C. IBM 7070 SYSTEM

Figures 36 and 37 show the schematics of the reader and punch attachments to the IBM 7070 Data Processing System. The Type 7500 Reader operates at 500 cards per minute in contrast to the 200 cpm speed of the 533. The Type 7550 Punch operates at 250 cards per minute, compared to the 100 cpm of the 533 punch and the 155 cpm of the 537. It can be seen that the speed has been improved by a factor of 2.5. This improvement, especially in the reader, is limited by the incidental nature of its card equipment to the 7070 system, rather than by current card-handling technology which has generated card transports running experimentally in the 3,000-4,000 cpm range.

D. BASIC IBM 650 SYSTEM

Figures 38 and 39 present the simplified data flow charts for card input and output in the basic 650 system. In Figure 38 "first read" and "second read" are the two reading stations required by the input card buffer, which collects the contents of the cards, one at a time, as they are read row-by-row while passing through the feed. Since the buffer is part of drum storage, it is serial by digit. However, the card is being read parallel-by-digit, so each horizontal row must be serialized as it is read. This is done by the "row buffer," which converts the information represented by a row of holes in the card into a binary pulse-train. This in turn feeds the "translator" (XL) which converts the serialized binary information train into its appropriate machine code. The coded output of the translator is then sent to the input buffer which accumulates the information, a row at a time, so that after the card has been completely read (all 12 rows) the buffer will contain the entire contents of the card record in valid machine code (biquinary, in the 650).

In the case of alphabetic and special character information, the 650 codes the characters into a two-digit numeric representation. Since acceptable characters in card-code form can range in hole-count from zero for the blank column to three for some of the special characters, it is obvious that any one hole can not generate the entire two-digit-biquinary character, especially since this particular hole may or may not be alone in its column. The actual relationship of the individual punched hole to the character which contains it must be "pre-sensed." This is done by a previous reading of the same card (one mechanical card cycle earlier) at "first read." The card information at the first read station must also pass through a row buffer and a translator to be serialized and coded. The information is then stored in the "pre-sense buffer," which in turn "tells" the second read translator how to translate a given hole in a given column for all holes in all columns of a given card.

In Figure 40 is shown the output data flow chart. Information to be punched in cards comes from "main memory" where it was developed by the stored program, to the "output buffer." From here it passes, once each drum revolution, to a translator which scans the entire contents of the output buffer for those coded characters which require holes to be punched in the immediate row on the output cards. This occurs row by row until the complete card has been punched. The output buffer data flow is straight-line, since the entire character representing a given column to be punched in the card is available to be analyzed for each required potential hole.

The storage buffers fulfill a very basic requirement — that of providing a "time delay" for the duration of the relevant mechanical feed cycle(an extremely slow process in terms of program execution time). In a buffered system like the 650 or the 7070, it is usually possible to do the entire processing of a given card record in less time than it takes to physically read the following input card and punch the preceding output card. If the processing unit were not buffered, the mechanical card cycles would pre-empt as much or more real time as is required to do the programmed processing. With buffering, these two necessities can be and are concurrent, for the duration of the shorter of the two. In other words, if it takes less than a card cycle to process a record, the program is finished, first and must wait for the completion of the card cycle in progress. If the program takes more than one card cycle, the card machine must wait until the program calls for another "read" and/or "punch" operation.

Thus the system paces itself automatically, the mechanically-generated interlock signals holding up the program if necessary, and the program delaying the following card cycle whenever it needs more real time. This process has the additional advantage that the card cycle need not be precisely synchronized with the program cycle since the two are mutually interlocked by the buffers' control logic.

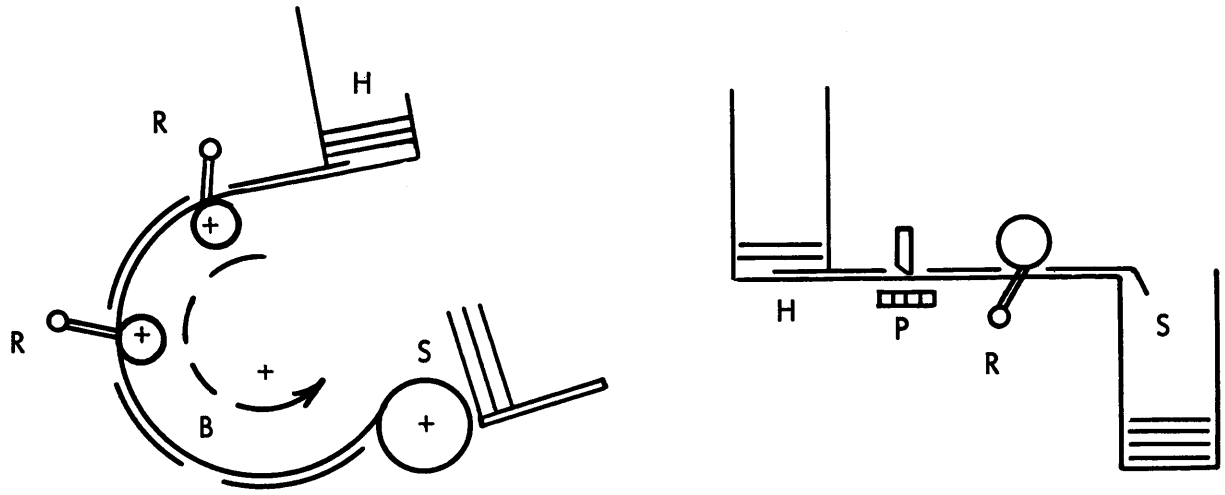


Fig. 35 - IBM 533 Card Reader and Punch

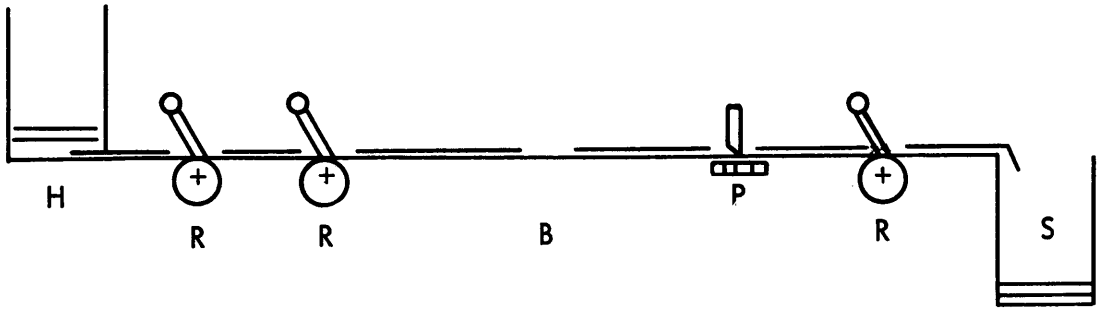


Fig. 36 - IBM 537 Computing Card Punch

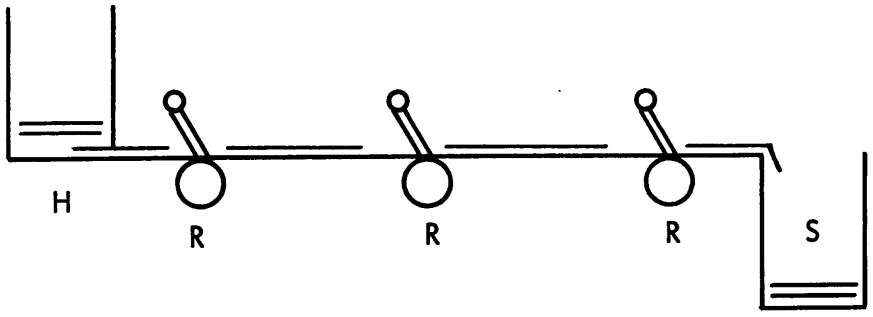


Fig. 37 - IBM 7500 Card Reader

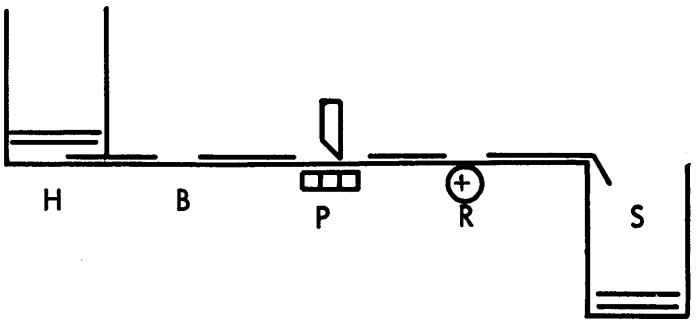


Fig. 38 - IBM 7550 Card Punch

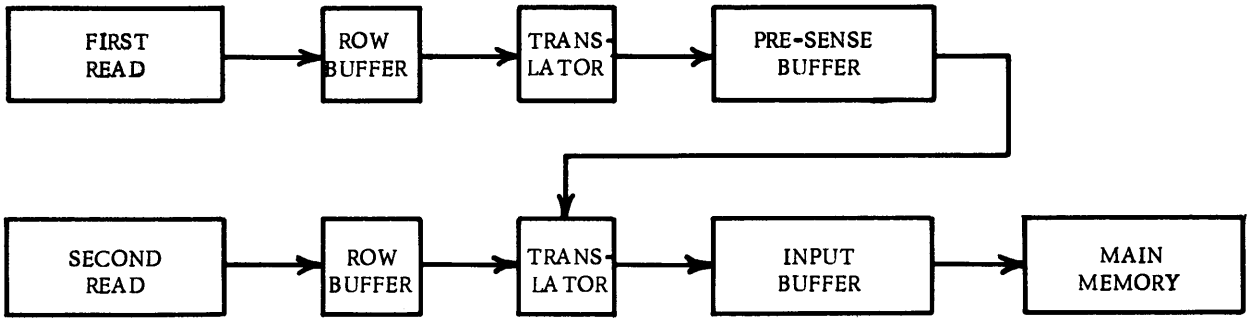


Fig. 39 - Card Input Data Flow for IBM 650

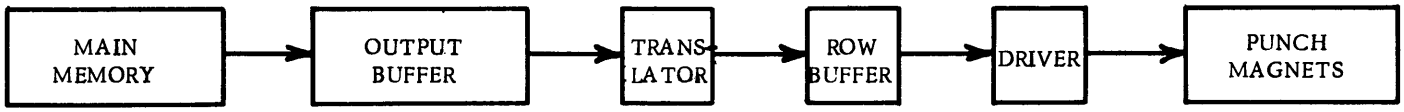


Fig. 40 - Card Output Data Flow for IBM 650

XXII WHEEL AND WIRE PRINTING

Another type of mechanical peripheral unit used with a great variety of data processing systems is the output printer. The IBM 717 and 718 Wheel Printers are available for attachment to the 700-series electronic computers, such as the 704, the 705, and the 709. The basic mechanism of these printers was developed in the 1940's and reached the accounting machine market in the form of the 407 Accounting Machine. The 407 can read punched cards, add their contents into electro-mechanical accumulating counters, list the information from the cards, and finally print totals called out on a basis of common identifying information punched on the cards.

The functions of the 717 and 718 are much more limited. Their basic application is the printing of intermediate and final results of the programmed processing of accounting of scientific information records. They are controlled by the stored program of the computer in the same manner as the readers and punches studied previously; in addition to the instructions, "read" and "punch," there is one which says "print."

The technique which the computer uses to communicate its output information to the printer is similar to that for the punch. The print magnets in the printer unit, which individually control a given horizontal position on all lines printed, are selected in the same manner as the punch magnets in the punch unit (i.e., the program controls the assembly of the output information using a group of memory addresses reserved for this purpose).

A. IBM 407 WHEEL PRINTER

Figure 41 is a detailed sketch of the basic print mechanism of the wheel printer taken from the Customer Engineering instruction manual. The succeeding figures will help to explain the operation of this mechanism. The purpose of Figure 41 is to demonstrate concretely the mechanical complexity of this device. Figure 42 is the mechanical timing chart for the print mechanism. The print cams may be considered the mechanical equivalent of the timing rings in the electronic circuits of the computer, since their function is also to control the ordered sequencing of steps which make up the operation. Figure 43 is a sketch of the typewheel — the component carrying the embossed characters which do the actual printing. Figure 44 shows the relative positions of the print mechanism parts when they are set up to print the character "V". Figure 44 is an enlarged portion of the mechanism shown in Figure 41.

The operation of the printing mechanism is as follows:

1. A timed impulse energizes the print magnet, attracting its armature.

2. The movement of the armature operates the upper pull rod and its four associated analyzer slides, which in turn mechanically "feel" the contours of the analyzer cams.

3. The analyzer cams select a unique combination of the four analyzer slides by allowing them to drop off their respective latches. This produces a mechanical storage effect which

is required to delay the information for a discrete amount of time so that the other parts of the print mechanism position themselves correctly.

4. After the predetermined delay time, the previously unselected analyzer slides, operated by the analyzer cams, release the lower pull rod. This trips the selector gear clutch by releasing the clutch pawl into one of the flutes of the revolving reamer shaft (named for its resemblance to a line reamer).

5. The rotation of the selector gear drives the typewheel around to position the selected character for printing.

6. Since the selector reamer shaft operates at a variable speed during the course of the operating cycle of the mechanism, the type wheel slows down during printing to a speed that matches the velocity at which the type-wheel hanger is cammed forward by the rotation of the print cam. The timing of the engagement of the print cam with its own reamer shaft, produced by a second (zone) impulse to the print magnet, controls which of the four characters in the selected group will print. Since there are four zones (because of the Hollerith code in the 407), there are four printing times, 15 mechanical degrees apart. This helps to distribute the impact of firing as many as all 120 printing positions at once. In addition to the 15-degree zone spacing, the print reamer shaft also has a 4-degree twist end-to-end further spread the impact.

Apparent from this description of its operation is the fact that this mechanism is very difficult to maintain. After a decade of concentrated re-design effort it has become a reliable and trouble-free device. This printing mechanism, an important part of the history of IBM engineering developments, is used in hundreds of computer output printers and thousands of accounting machines.

B. IBM WIRE PRINTERS

As mentioned earlier, the quest for greater operating speeds for peripheral equipment is countinuous. The basic 407 wheel printing mechanism has an optimum working speed of 150 lines per minute, which can not be exceeded significantly enough to justify the attempt. Because of the need for something faster was born the wire printer, which operates at a speed of 500 and 1,000 lines per minute. Figure 45 shows the print mechanism plate and the mechanical drive.

The character is printed by forming it in a matrix of 35 dots produced by the ends of selected print wires. These wires are vertically aligned adjacent to a selection mechanism so that they can be controlled individually by a common code rod. This rod is shown in Figure 46, with its guide tube, which positions the print wire ends.

Figure 47 shows the various combinations of rotational and lateral positions of the code rod and the correspondingly selected characters. Figure 48 shows the incremental values of two groups of mechanical wedges which operate as "adders" to develop the code rod positioning information.

Figure 49 shows the complete character set which the wire printer is capable of printing, including some previously unmentioned characters whose Hollerith equivalent codes are 8-5 and 8-6, extrapolated from the familiar 8-3 and 8-4 special characters. Figure 50 shows the dot pattern produced by a 5 x 7 print wire matrix for some of the printed characters.

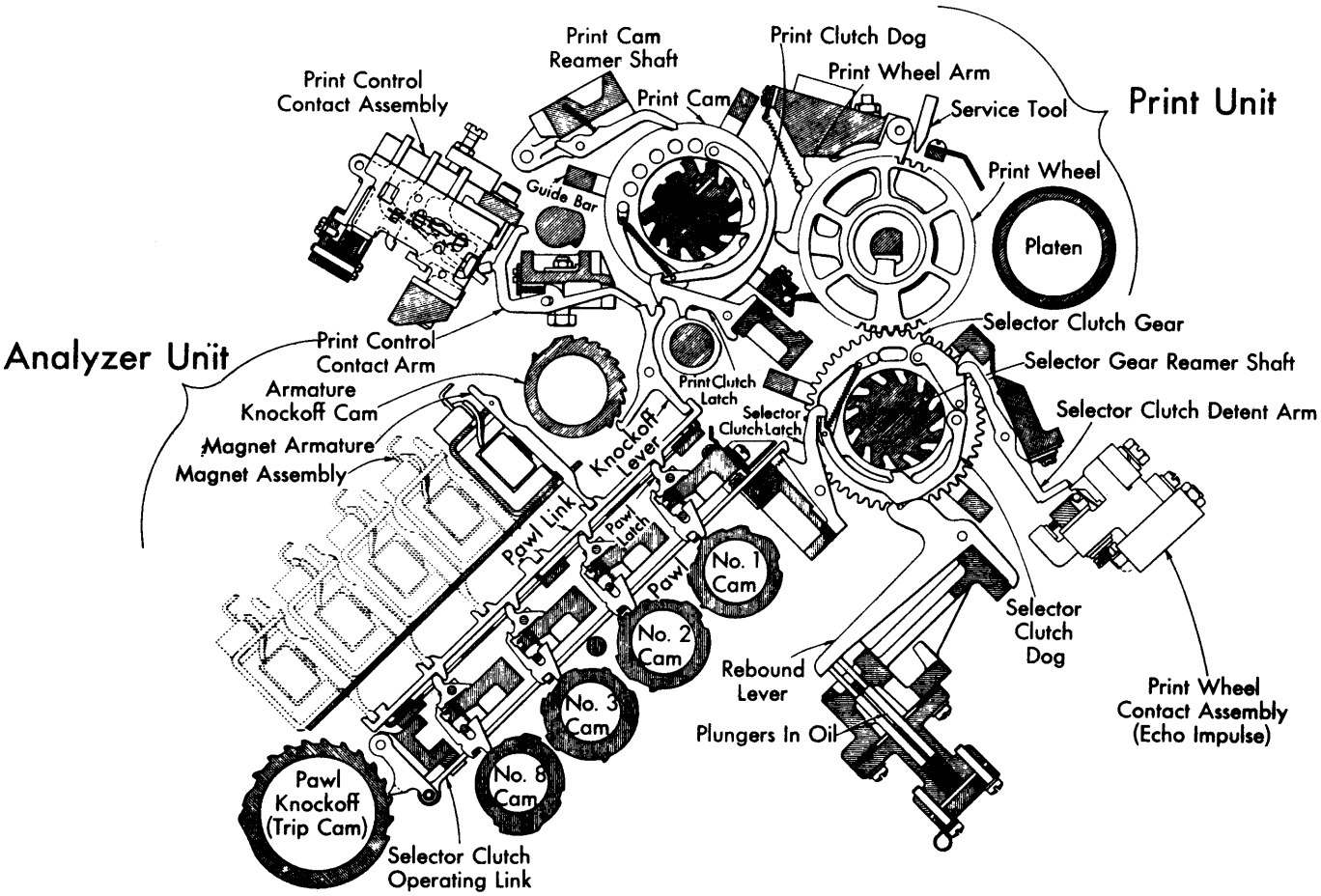


Fig. 41 - Printing Mechanism of the IBM 407 Accounting Machine

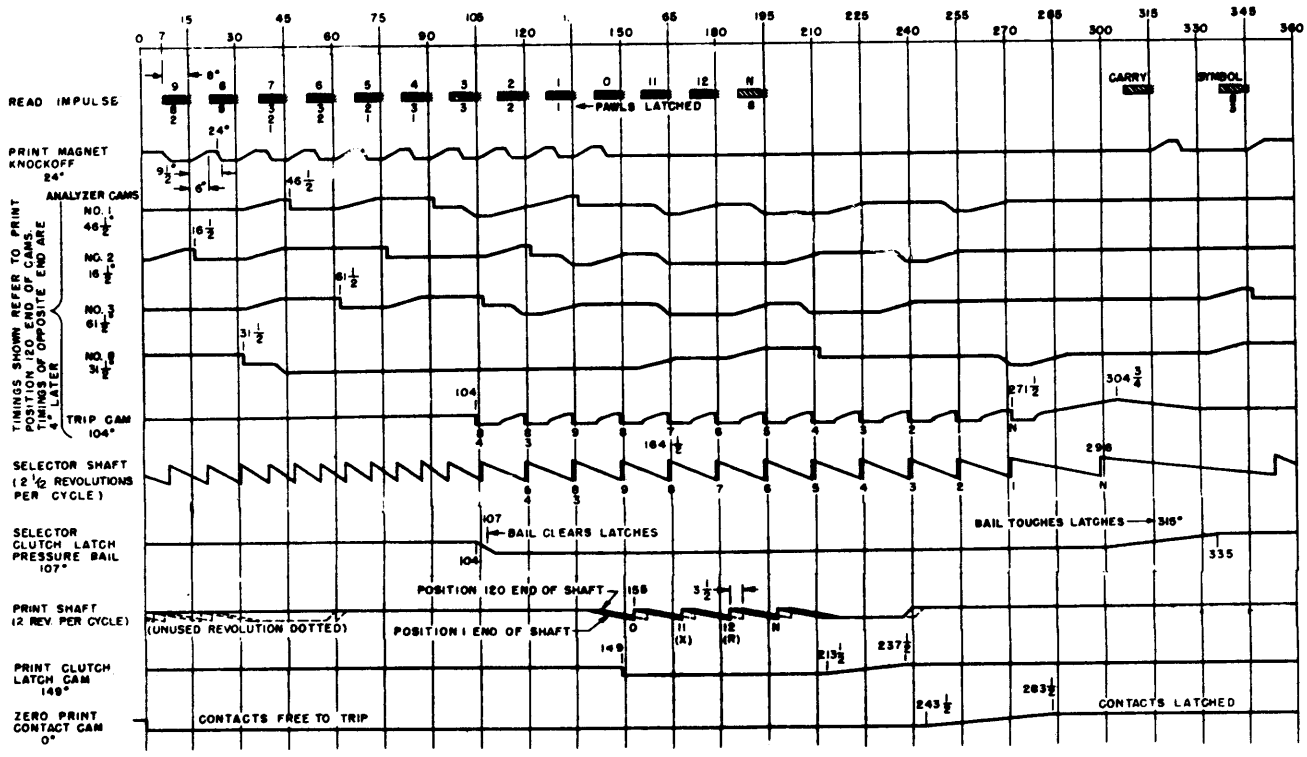


Fig. 42 - Mechanical Timing Chart for the IBM 407 Print Mechanism

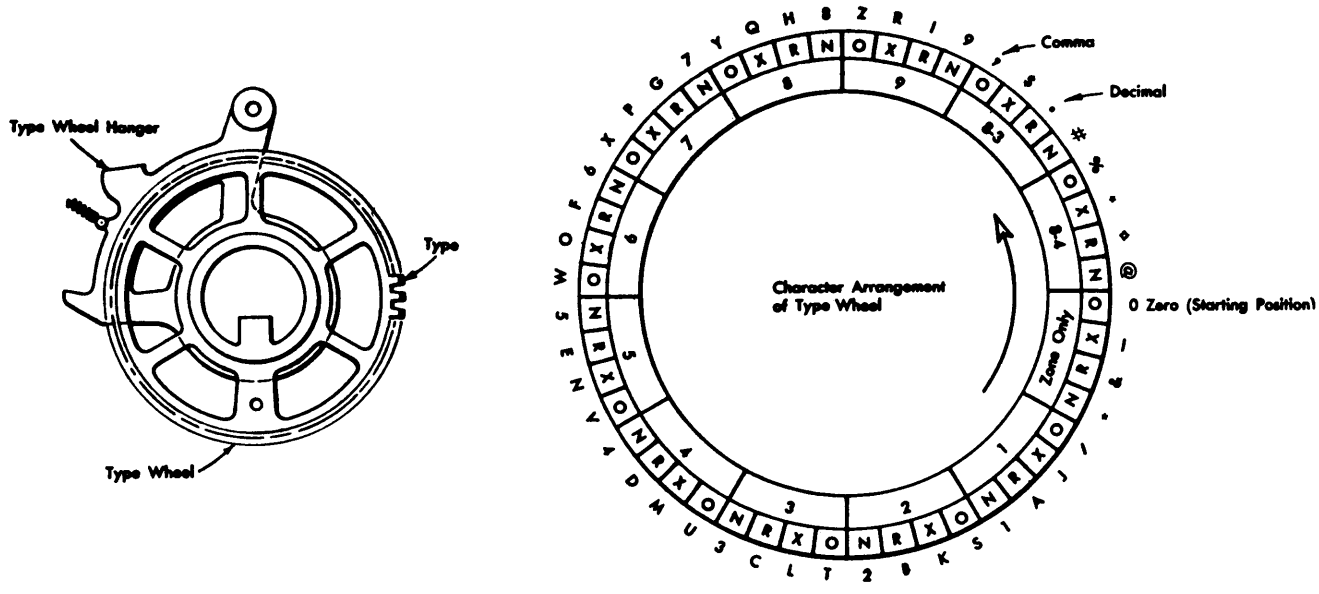


Fig. 43 - IBM 407 Print Mechanism Typewheel

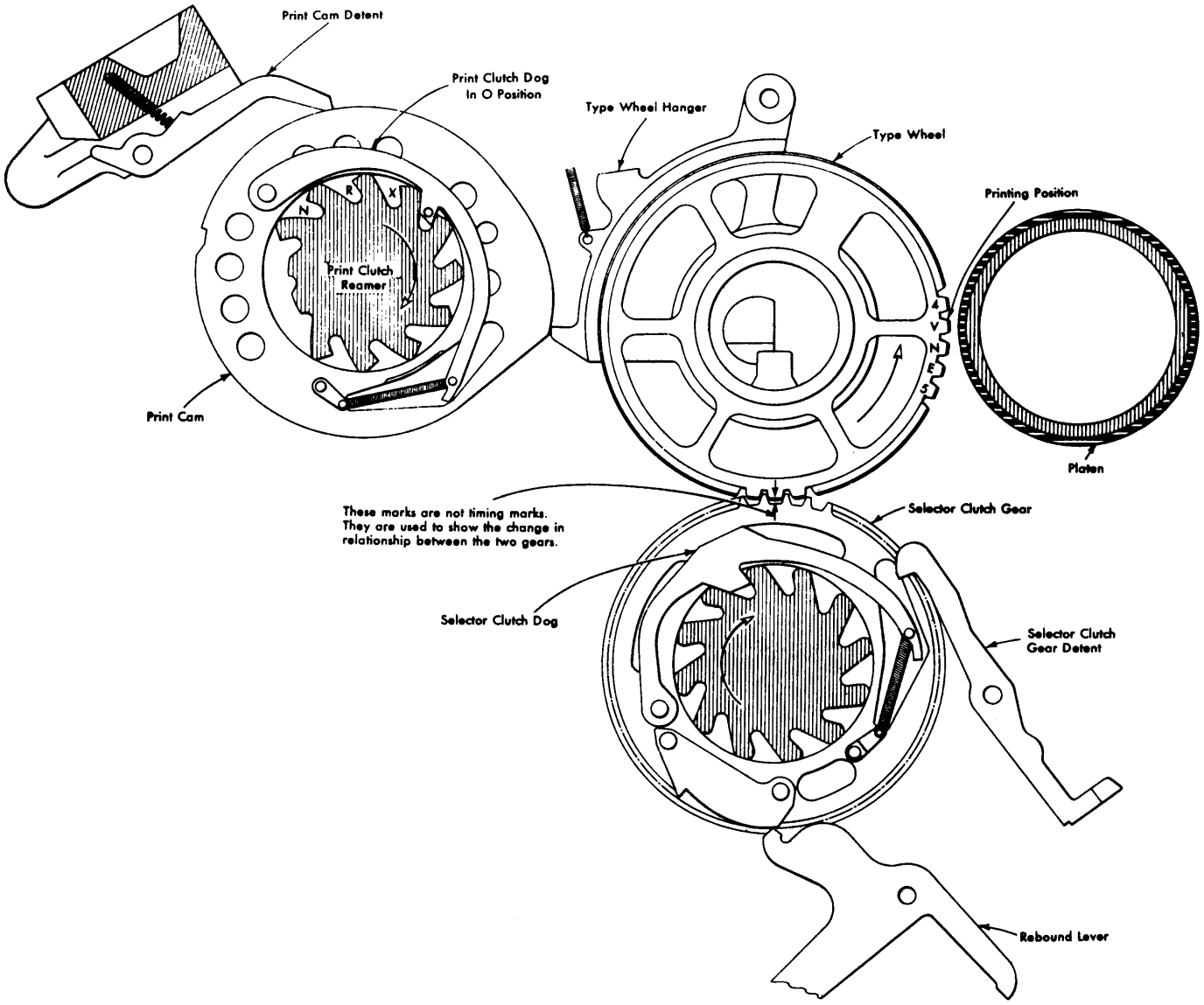


Fig. 44 - IBM 407 Print Mechanism Positioned for Printing V

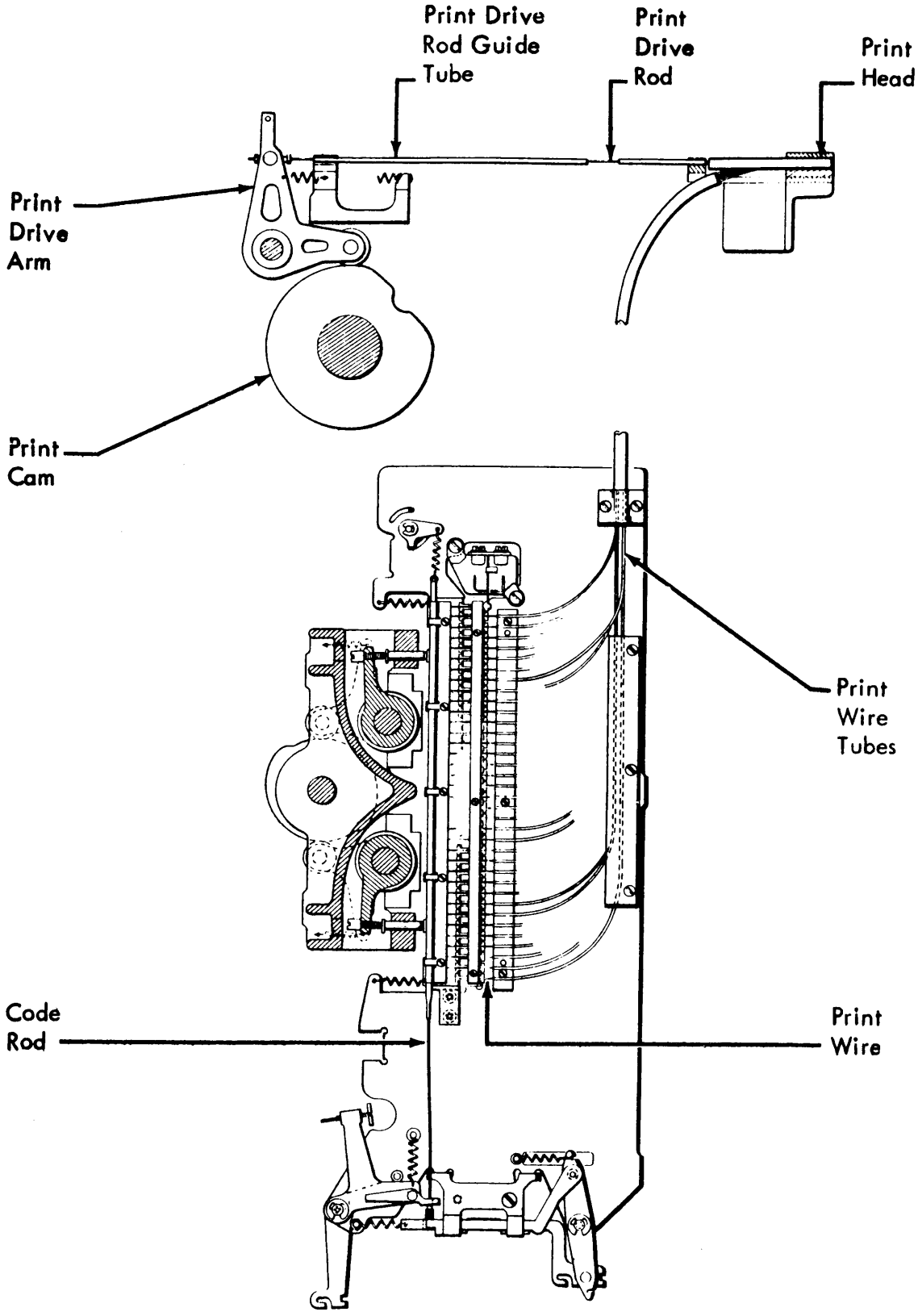


Fig. 45 - Print Mechanism Plate and Mechanical Drive

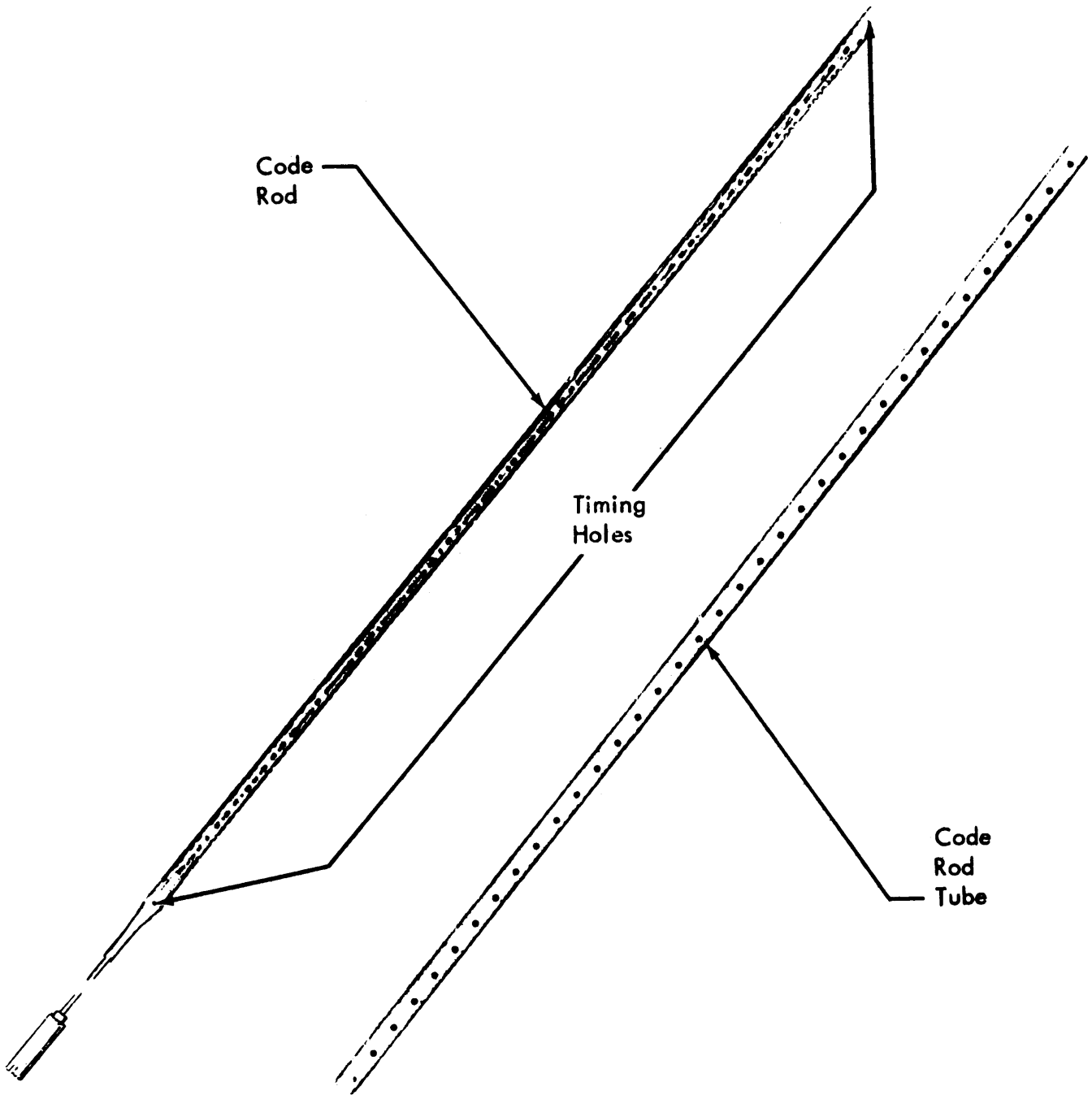


Fig. 46 - Code Rod and Tube for the IBM Wire Printer

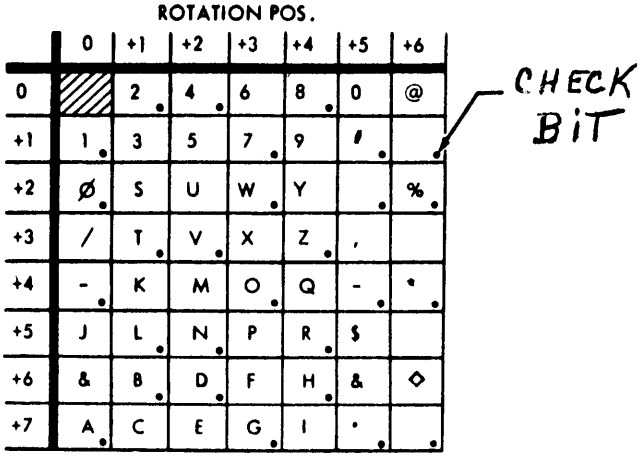


Fig. 47 - Rotational and Lateral Positions of the Code Rod

BINARY CODE	WEDGE VALUE	
X	+4	LATERAL
0	+2	
1	+1	
2	+1	ROTARY
4	+2	
8	+4	

Fig. 48 - Mechanical Wedge Values

CHARACTER	IBM	BINARY	CODE ROD ROT DISP.	CODE ROD LAT DISP.
A	12-1	X-0-1	0	+7
B	12-2	X-0-2	+1	+6
C	12-3	X-0-1-2	+1	+7
D	12-4	X-0-4	+2	+6
E	12-5	X-0-1-4	+2	+7
F	12-6	X-0-2-4	+3	+6
G	12-7	X-0-1-2-4	+3	+7
H	12-8	X-0-8	+4	+6
I	12-9	X-0-1-8	+4	+7
& TPM	12-2-8	X-0-2-8	+5	+6
.	12-3-8	X-0-1-2-8	+5	+7
◇	12-4-8	X-0-4-8	+6	+6
&	12	X-0	0	+6
SPARE	12-5-8	X-0-1-4-8	+6	+7
J	11-1	X-1	0	+5
K	11-2	X-2	+1	+4
L	11-3	X-1-2	+1	+5
M	11-4	X-4	+2	+4
N	11-5	X-1-4	+2	+5
O	11-6	X-2-4	+3	+4
P	11-7	X-1-2-4	+3	+5
Q	11-8	X-8	+4	+4
R	11-9	X-1-8	+4	+5
- TPM	11-2-8	X-2-8	+5	+4
$	11-3-8	X-1-2-8	+5	+5
*	11-4-8	X-4-8	+6	+4
-	11	X	0	+4
SPARE	11-5-8	X-1-4-8	+6	+5
/	0-1	0-1	0	+3
S	0-2	0-2	+1	+2
T	0-3	0-1-2	+1	+3
U	0-4	0-4	+2	+2
V	0-5	0-1-4	+2	+3
W	0-6	0-2-4	+3	+2
X	0-7	0-1-2-4	+3	+3
Y	0-8	0-8	+4	+2
Z	0-9	0-1-8	+4	+3
≠ TPM	0-2-8	0-2-8	+5	+2
'	0-3-8	0-1-2-8	+5	+3
%	0-4-8	0-4-8	+6	+2
ø (ZERO)	0	0	0	+2
SPARE	0-5-8	0-1-4-8	+6	+3
1	1	1	0	+1
2	2	2	+1	0
3	3	1-2	+1	+1
4	4	4	+2	0
5	5	1-4	+2	+1
6	6	2-4	+3	0
7	7	1-2-4	+3	+1
8	8	8	+4	0
9	9	1-8	+4	+1
0 TPM	2-8	2-8	+5	0
'	3-8	1-2-8	+5	+1
@	4-8	4-8	+6	0
BLANK			0	0
SPARE	5-8	1-4-8	+6	+1

Fig. 49 - Available Character Set

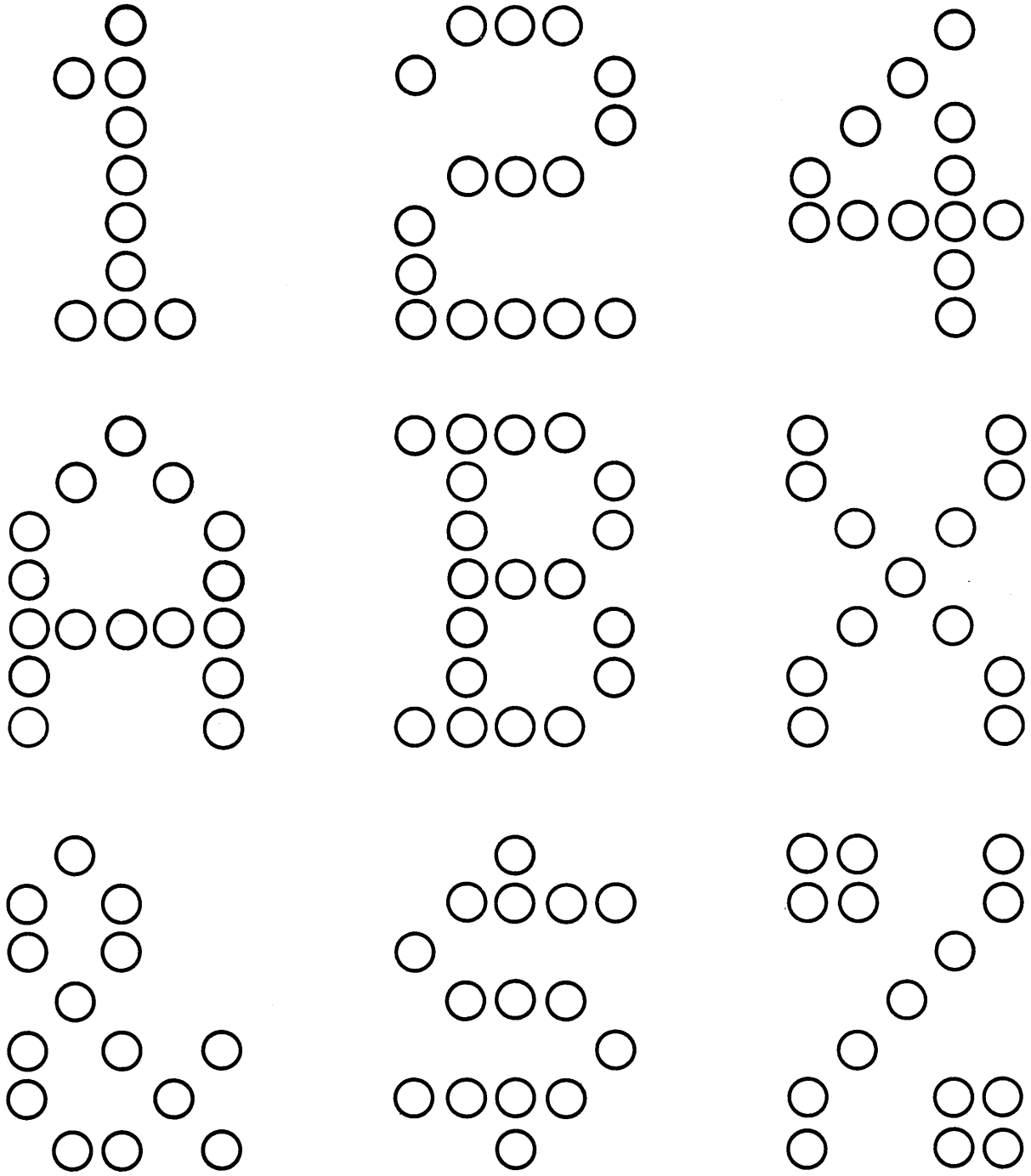


Fig. 50 - Dot Patterns Produced by a Print Wire Matrix

XXIII CHAIN, BAR-AND-HELIX, AND STICK PRINTING

A. IBM CHAIN PRINTER

A new line printer development is the chain printer, released for the 1401 Stored Program Accounting Machine. The basic 1401 consists of a transistorized (CTDL) processing unit, an opposed-feed reader-punch, and a chain printer. It is intended as a market replacement for multiple 407's. The 407 has a constant operating speed of 150 cycles per minute. In comparison, the 1401 reads cards at 800 per minute, punches at 250 per minute, and prints at 600 lines per minute. These speeds are maxima which progressively reduce the time requirements of the stored program. Normally the 1401 will read and punch concurrently, while the program is suspended, because the card equipment is unbuffered (uses main memory directly). Due to memory access limitations, printing is mutually exclusive with reading and punching and also suspends the program. The usual sequence of functional events is:

1. Read the next card and simultaneously punch the output card for the previous input.

2. Print the line corresponding to the card just punched.

3. Process (compute) the card just read.

4. Repeat this sequence for the balance of the run.

Figure 51 shows the layout of the print chain. There are five complete sets of characters in the chain, arranged in identical sequence. Figure 52 illustrates how printing takes place. The chain operates continuously. The timing of the drive pulse to the hammer magnet is critically synchronized with the immediate position of the character set on the chain, in order to strike the desired character in flight, through the paper form on which the character is to be printed. The operating time of the armature and hammer is in the range of one to one and one-half milliseconds. The contact duration between the hammer and the chain is sufficiently short to minimize the horizontal smudging or scuffing which takes place due to the continuous motion of the chain.

The data flow of printed information is shown in Figure 53. The print area of memory is read out serially-by-character as each new character is positioned for printing. There are 48 characters in the set, spaced on .150" centers in the chain. Since the hammers are on .100" centers(ten to the inch), every other character will line up with every third hammer. Thus, three "sub-scans" are required for each complete "print scan" of the hammers, for a total of 144 memory scans per line of printing.

The Print Scan Counter is used to hold the starting value for the Compare Counter for each sub-scan. The Compare Counter advances in synchronism with the memory readout. It distributes the information in memory to the correct printing positions on a basis of the relationship between the contents of memory and the respective characters available to the hammers during that particular sub-scan. The Hammer Scan Matrix distributes the print impulses to the corresponding Hammer Drivers, which in turn energize their respective Hammer Magnets.

B. IBM BAR-AND-HELIX PRINTER

Another interesting line printing technique is the bar-and-helix mechanism. The helix is carried on a rotating shaft and its potential point of contact with the horizontal bar during one revolution of its shaft generates one "line" of a TV-type raster (corresponding to the horizontal sweep in TV). The bar shuttles radially, producing point impressions which are the "elements" of the raster "line." At the same time, the paper form is moving vertically upward, generating the vertical component of the raster. Thus, the character raster is scanned by a combination of the horizontal sweep of the helix relative to the bar and the slower vertical motion of the paper. The radial motion of the bar with respect to the helix produces the elements of the character (the "video" signal). Figure 55 shows how the characters "1" and "S" are formed. There are seven possible impression points on each of nine lines of the raster, providing somewhat better visual definition of the character than does the wire printer. This mechanism, however, has not yet achieved acceptable performance for commercial use.

C. IBM STICK PRINTER

Another unusual printing device is the "stick" printer, which is used in the Type 370 Output Printer of the 305 RAMAC system. Like a typewriter, it prints only one character at a time. Figure 56 shows how the characters are laid out on the flat surfaces of the "stick" (a modified octagonal cylinder). A combination of axial and angular movements of the stick aligns the selected character with the platen, which moves to provide the impression impact.

The 370 prints ten characters to the inch (80 characters per line) at a maximum speed of 3,000 characters per minute. This printing speed is equivalent to 500 words per minute on a typewriter — the normal output of eight or ten typists.

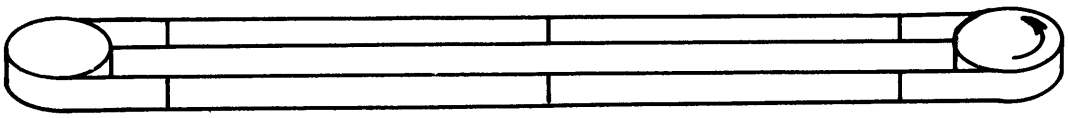


Fig. 51 - Positioning of the Print Chain

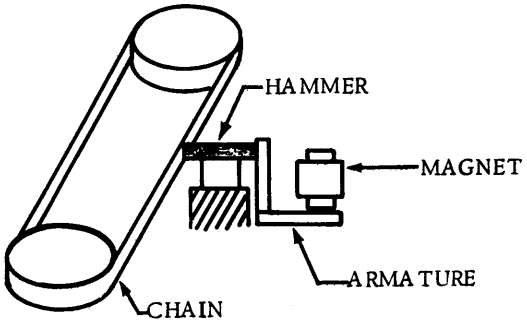


Fig. 52 - Descriptive Drawing of the Print Mechanism

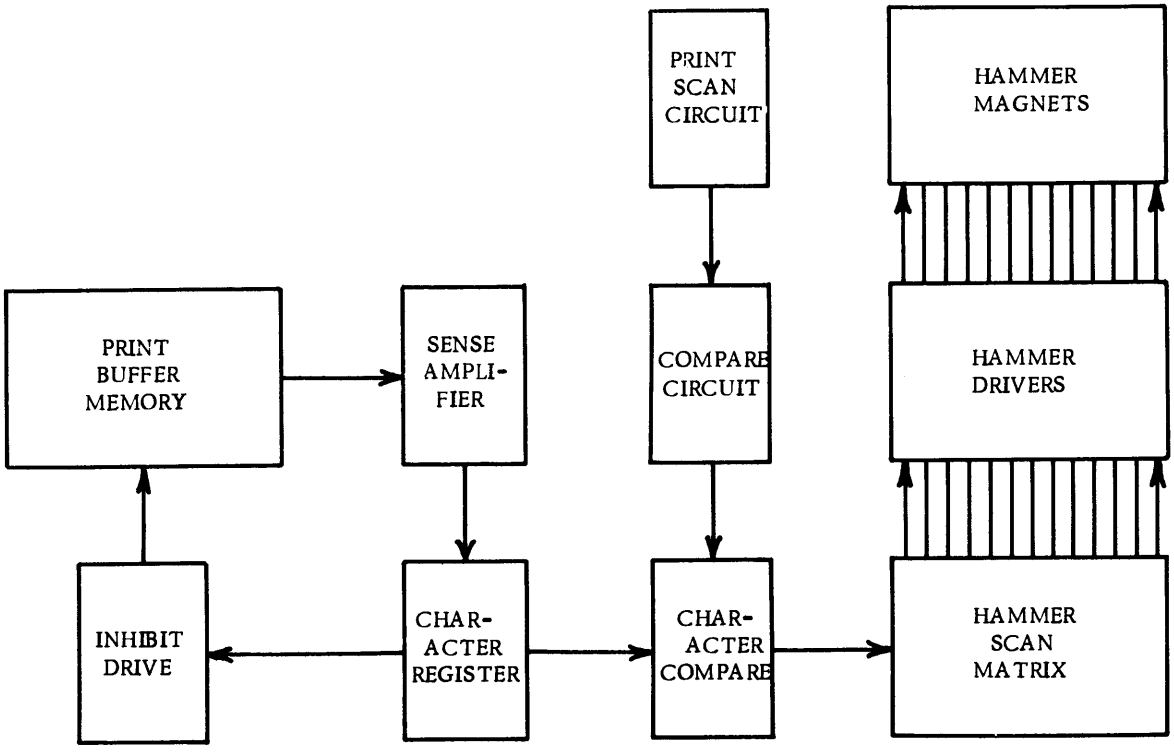


Fig. 53 - Data Flow of the Printed Information

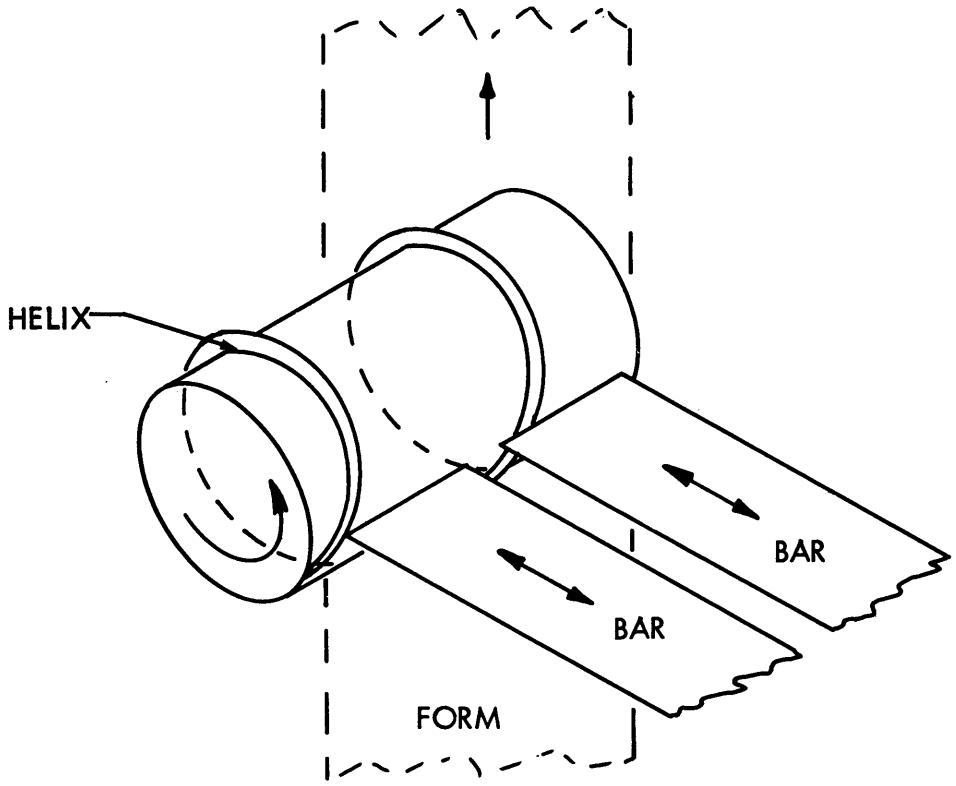


Fig. 54 - IBM Bar-and-Helix Printer

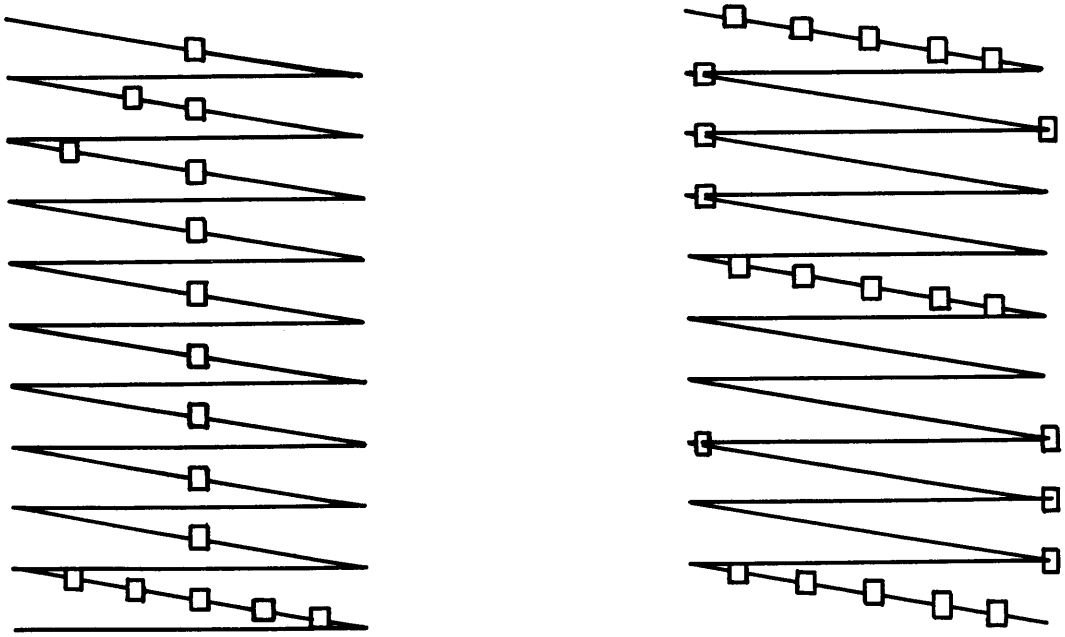
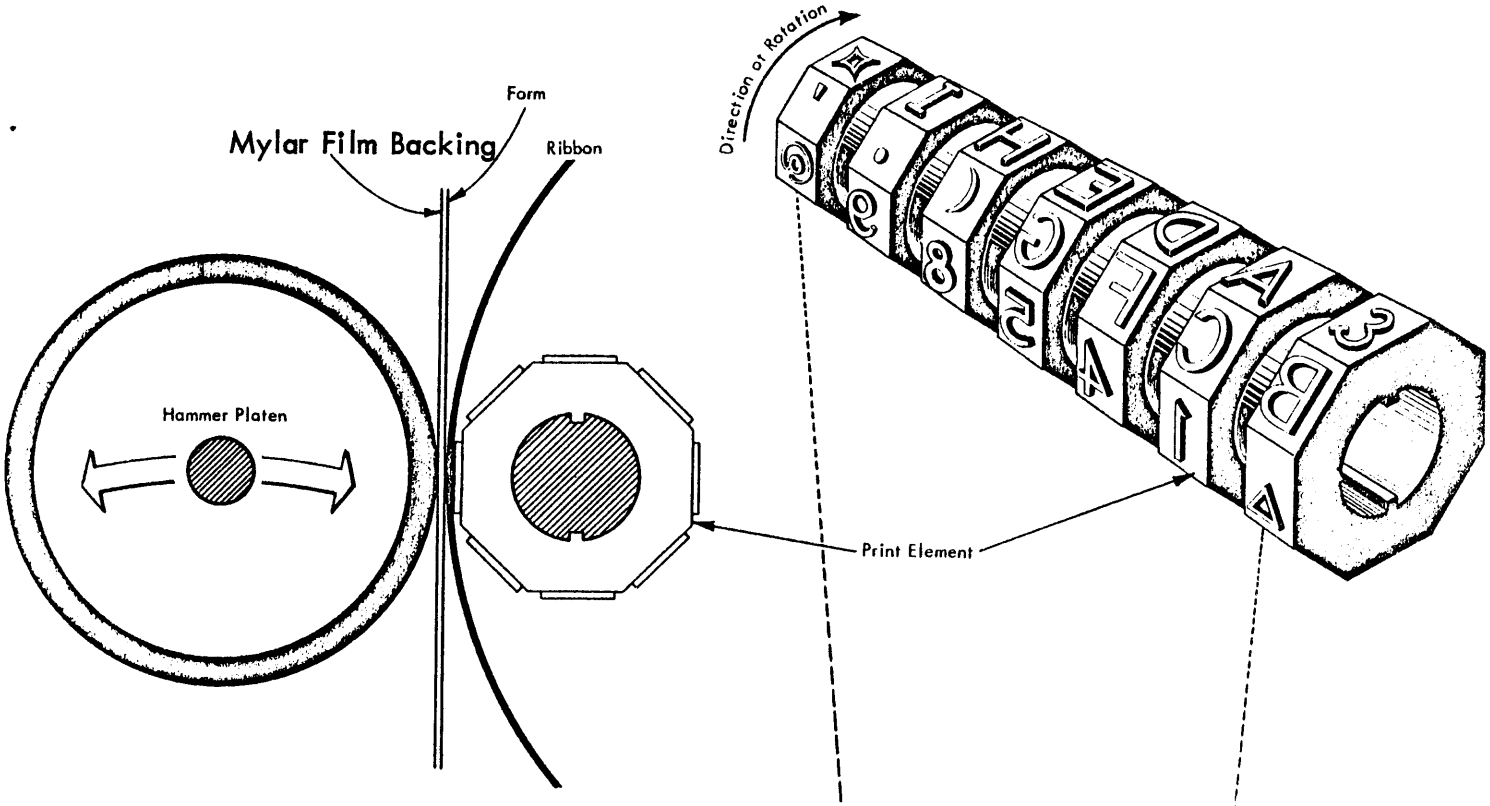


Fig. 55 - Formation of the Figures "1" and "S"



PRINT ELEMENT CODE AND MOTION VALUES

Binary Code	Motion Values	
X	+ 4	Rotary
0	+ 2	
2	+ 1	
1	+ 1	Horizontal
4	+ 2	
8	+ 4	

@	9	8	5	4	1	Home Δ		
11	#	10	7	6	3	2	1 R	2
%	Z	Y	V	U	/	Zero 0	2 R	0
° Degree	,	+	X	W	T	S	2-1 R	0-2
*	R	Q	N	M	J	—	4 R	X
"	$	)	P	O	L	K	4-1 R	X-2
¤	I	H	E	D	A	&	4-2 R	0-X
'		(	G	F	C	B	4-2-1R	0X-2
4 2 H	4 1 H	4 H	2 1 H	2 H	1 H	PRINT MAGNETS		
4 8	1 8	8	1 4	4	1	BINARY CODE		

Shaded Characters Are Not Obtainable With Valid Data Coding

Fig. 56 - Print Element Positioning in the IBM Stick Printer

XXIV TAPE DRIVE UNITS

Multiple-channel magnetic tape has become the most popular of all data-processing system input/output media. There are competitive (non-IBM) systems on the market today which rely exclusively on magnetic tape for input/output and are supported by "off-line" equipment for transcription from punched cards, punched paper tape, and keyboards onto magnetic tape. The system advantages of this arrangement are: a) the relatively slow input/output functions can be carried out without tying up the expensive electronic system just to acquire input information, and b) the system can produce output information in forms acceptable to the external man-controlled processes which link the system with the outside world. This method of establishing the input/output link (i.e., via magnetic tape) avoids the high cost of direct "on-line" buffering of the mechanical card, paper tape, and keyboard equipment by investing instead in one or more small satellite tape transcription systems. The choice between these two basic approaches depends largely on the nature of the data-processing application and the relative cost and availability of peripheral equipment (on- and off-line) for the given system.

In any case, the traditional application of magnetic tape equipment is on-line (directly tied into the electronic system and generally unbuffered), reading in and out of memory under program control. At IBM, this function is provided by the IBM 727 and 729 Magnetic Tape Drive Units and their associated control circuitry which are usually tailored to the immediate system using the tape drives. The attached illustrations are taken from the 727 customer engineering instruction manual and will be discussed in detail.

Figure 57 depicts the geometry of the information flux-patterns on the tape as used in the 727; the 729 does not differ markedly from these dimensions. There are seven tracks on the tape, provided by a seven-gap read/write head. Since the tape itself is one-half inch wide, the track-to-track spacing is about one-seventh of that, or .07 inch. The actual width of the flux-pattern in a track is .032 inch or about one thirty-second inch. This leaves slightly more than that as a "no-man's land" between adjacent tracks. The distance from the edge of an outside track to the edge of the tape is a little more than half the width of the "no-man's land" (.021 inch). The lateral spacing of successive bits in a given track is .005 inch (a bit density of 200 per inch).

Figure 58 shows the general shape of the hysteresis loop of the iron oxide coating on the tape, which has a cellulose acetate (cellophane) base. (This is the basis for one of magnetic tape's quaint nicknames, "rusty cellophane.")

The flux pattern on the tape is written by a read/write head, shown schematically in Figure 59a. The two coils shown oppose each other, one of them used for each of the two current polarities with which the flux can be "written." Figure 59b shows the flux-pattern of a typical sample of information in terms of flux density. This is the so-called NRZI (non-return to zero, IBM) technique, which represents binary information by reversing the polarity of the flux for each successive "one." The 111001 information shown produces four flux-reversals, whose timing identifies the location of the binary "ones" in the information. Figure 59c shows the resulting flux-pattern in terms of lines of force looking at the edge of the tape. The flux reversals produce concentrations of leakage flux because of the adjacency of the like magnetic poles. It is this flux leakage that is coupled to the read/write head to sense the flux-pattern during the reading of the tape.

Figure 60 shows the front view of the tape drive unit, illustrating its essential functional devices. This unit feeds tape between 2400 foot reels at a closely-regulated speed of 75 inches per second during reading and writing. Since the bit density is 200 bits per inch, this speed produces an information rate of 15,000 bits per second (15 kc). The total capacity of a reel of tape as written on an IBM 727 is 200 bpi x 7 tracks x 12 inches per foot x 2,400 feet or 40,320,000 bits of binary information. In a system like the IBM 705, which writes seven-bit coded characters, this reduces to 5,960,000 characters, provided the entire reel of tape was written as one intact record (not possible on the 705 due to memory limitations).

The basic function of the 727 tape drive is to feed the tape from the file reel to the machine reel past the read/write head. During this operation the tape moves from the file reel, down into the left-hand vacuum column and back up. This forms a loop which seals the vacuum column and puts gentle tension on the tape to provide complete control over the position of the tape as it passes through the drive unit. The tape then moves through the read/write head assembly, down and back up the right-hand vacuum column, and onto the machine reel. The vacuum columns operate at a suction head of about five or six inches of mercury (about .8 atmosphere) with substantial leakage of air past the edges of the tape in the columns for lubrication. The length of the tape loop in the column is controlled by the sensing or, either vacuum or ambient air pressure, by two vacuum switches in each column. A servo system keeps the bottom of the loop between the two vacuum switches. This servo action is based on the constant movement of the tape past the head. The tape is pressed against one or the other switch by the action of its idler roller, carried on a movable bracket. The drive unit also rewinds the tape under program control at a speed of 500 ips, a full-reel rewind requiring about one minute.

Figure 61 shows the arrangement of the control keys and indicator lights, which the operator and the tape drive use to communicate with each other. These will be considered later under the topic of the 650 tape system.

Figure 62 is a sectional view of the powdered-iron magnetic clutch used to drive the reels. The rotor is keyed to the drive shaft (not shown) which carries the tape reel spool. The three-sheave V-pulley is belted to the drive motor which supplies the mechanical power controlled by the clutch. The clutch coil is connected to its external control circuits by slip rings (not shown), which provide DC current. This magnetizes the iron powder contained in the chamber where the driven rotor is located. Magnetization of this iron powder causes it to congeal, "freezing" the rotor to the rotating housing and transfering the mechanical power to the rotor and its reel. Gradual application of current to the clutch coil allows the clutch to "pick up" its load smoothly, minimizing the tendency of the tape to slip or kink on the reel.

Figure 63 shows a rear view of the system of motors, belts, and clutches required to drive the unit. Notice particularly the capstan drive belt, which drives the two capstans in opposing directions. This is required since the tape must be pulled, rather than pushed, past the head. The take-up motor, used only at the beginning of the tape run for the purpose of slowly lowering the tape into the vacuum columns, feeds tape off both reels at the same time by rotating the m in opposite directions until the upper vacuum switches sense the arrival of the loops.

The IBM 727 is a very popular device, but it is now obsolescent because of its speed limitations and mechanical complexity. A series of IBM Type 729's is now in existence, which provide basic mechanical improvements, and in turn make it possible both to feed the tape faster (by 50 per cent at 112.5 ips) and to achieve higher bit densities (up to 555 bpi). The resulting increase in information rate provides a limiting bit frequency of 62.5 kc, more than four times that of the 727. However, the functional capabilities of the two units (read, write, backspace, rewind, load, and unload) are virtually the same.

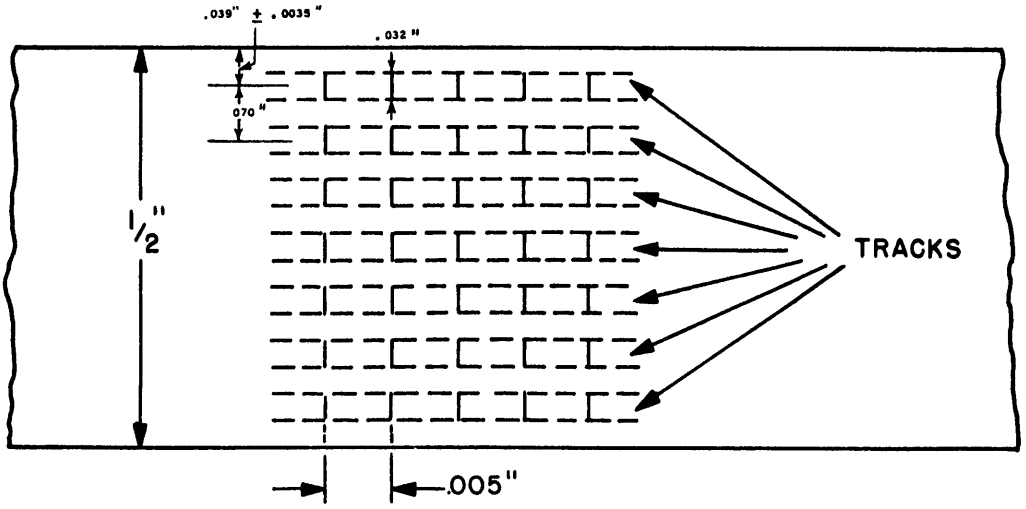


Fig. 57 - Geometry of Information Flux Patterns on Tape

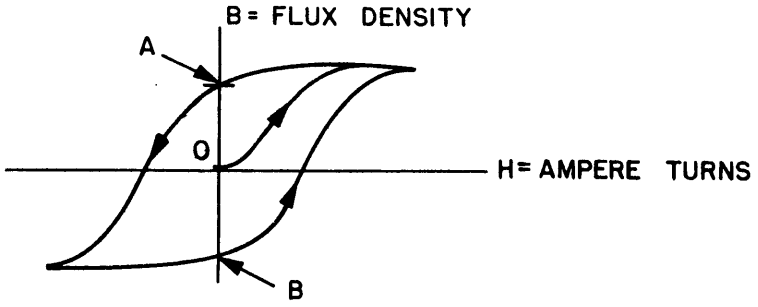


Fig. 58 - Hysteresis Loop of Tape Coating

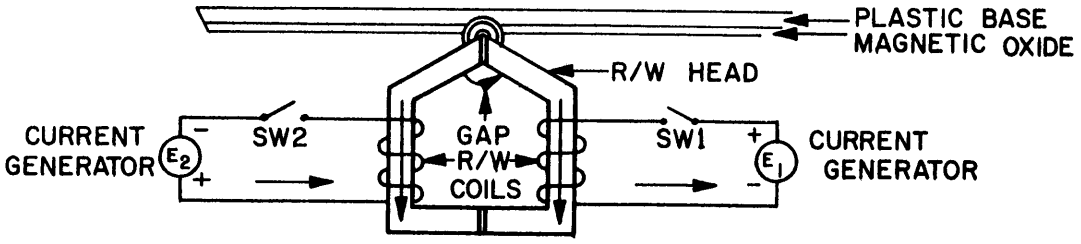


Fig. 59a - Schematic Diagram of Read/Write Head

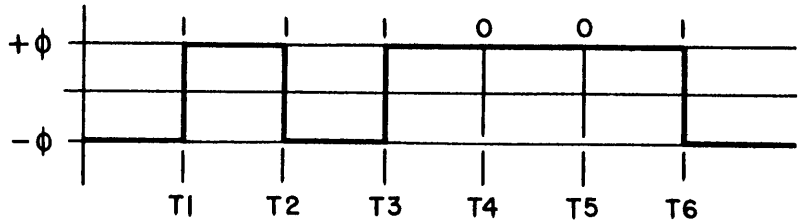


Fig. 59b - Flux-Pattern of Typical Information Sample

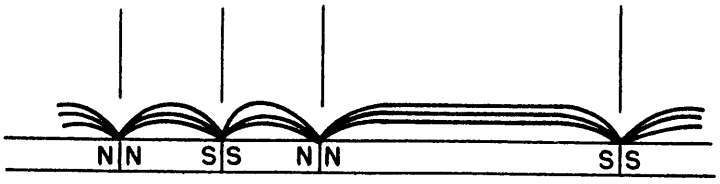


Fig. 59c - Resulting Flux Pattern

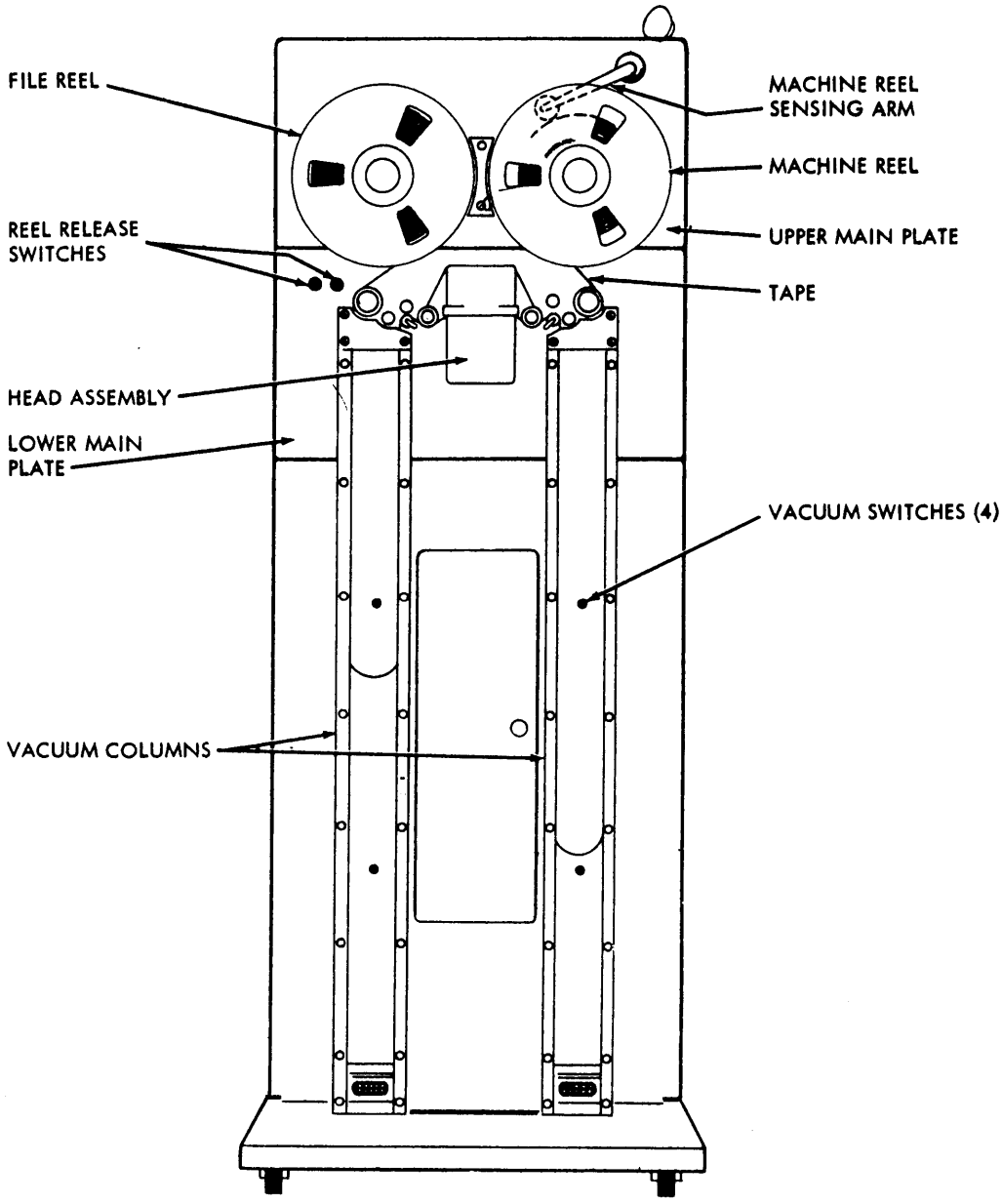


Fig. 60 - Front View of the IBM 727 Tape Drive Unit

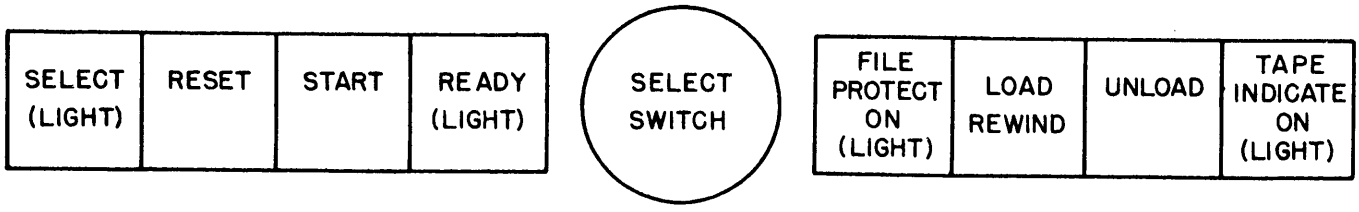


Fig. 61 - Tape Drive Unit Control Keys and Indicator Lights

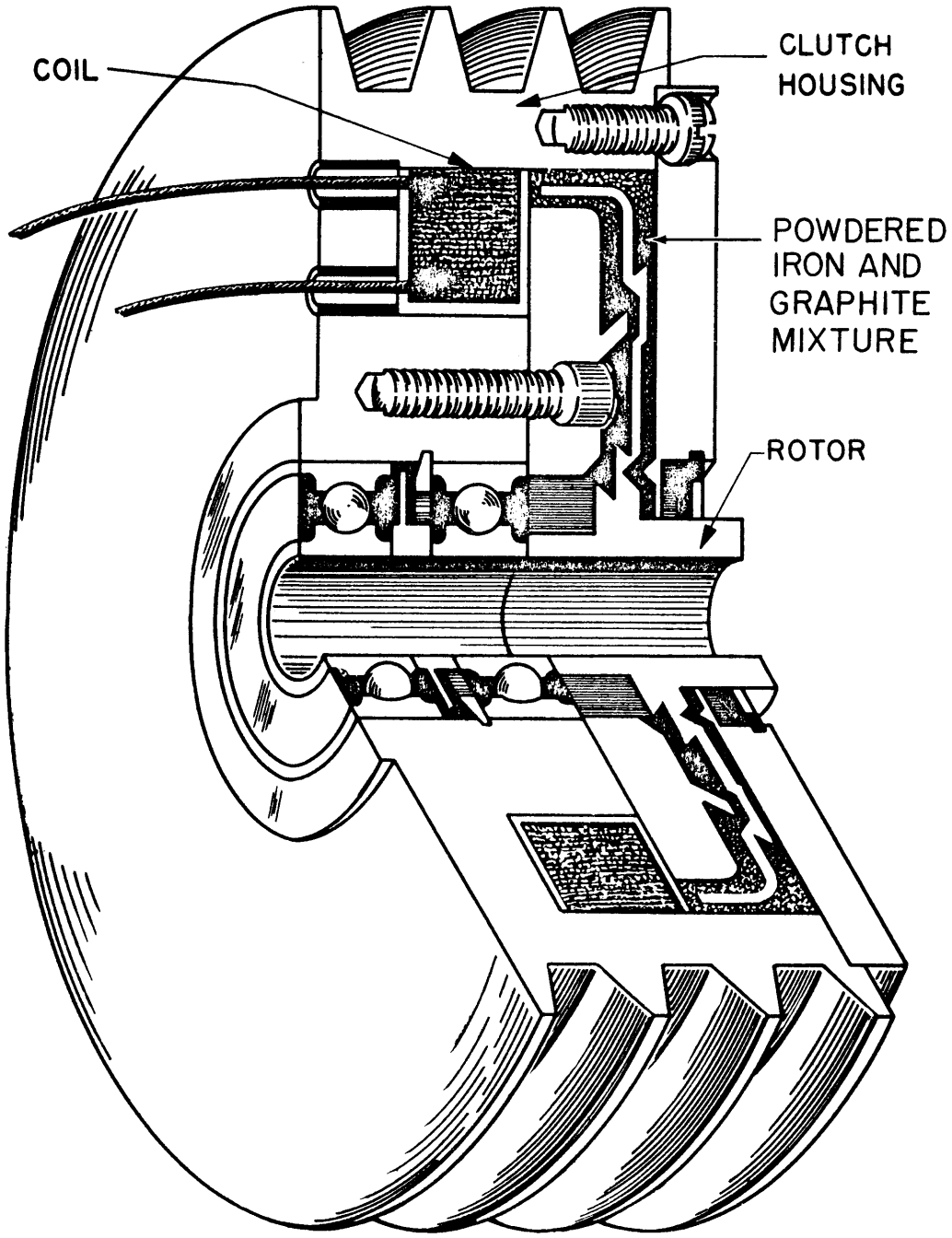


Fig. 62 - Sectional View of Magnetic Clutch

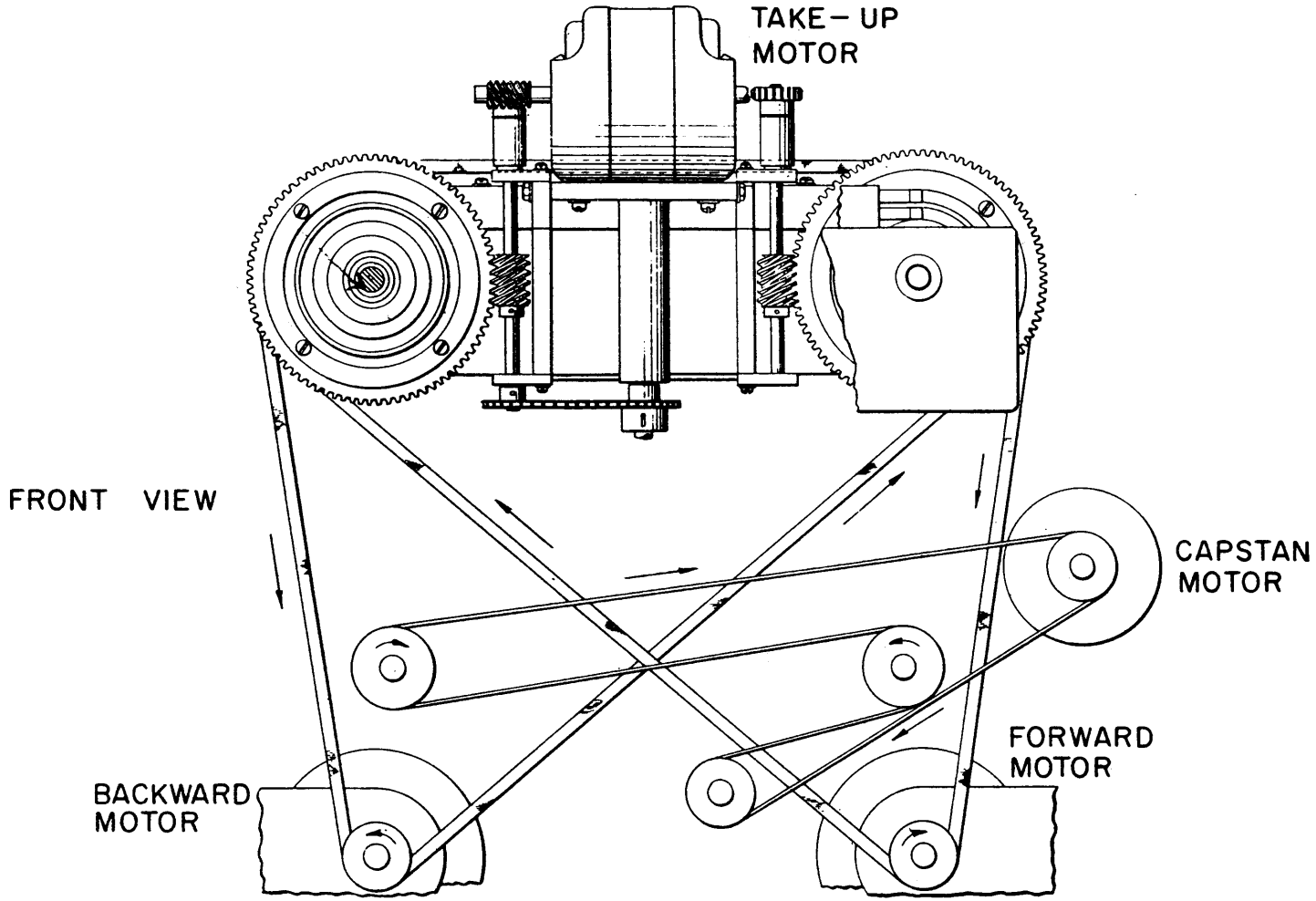


Fig. 63 - Drive Motors and Pulleys

XXV RANDOM ACCESS DISK FILE MEMORY

The random access disk file memory is unique to IBM data processing systems. It was developed originally for the IBM 305 RAMAC* (Random Access Method of Accounting and Control) system by the San Jose Product Development Laboratory. The 305 is the first system of a new line of equipment specifically designed for what has come to be known as "in-line" data processing. This accounting method is basically a mechanized version of the traditional "journal ledger," which is an accounting record book maintained in the chronological order of the transactions being accounted.

In an old-time retail sales operation, at the time of occurrence of the sale of one or more items, an entry would be made on a page reserved for each item affected, immediately below the last previous activity for that same item, to provide a continuous current record of the status of each item in stock. Sales would reduce inventory; received orders would increase it, the running balance representing the amount immediately on hand. In a historical era where life was simple and merchandising was very primitive and where the typical retail store might stock several hundred items at the most, this accounting method was quite satisfactory. However, in the course of the last several decades of merchandising and accounting history, the art of retailing has grown immensely.

Consider the very large department stores, chain groceries, and drug stores which form an important part of our personal lives today. They stock tens and hundreds of thousands of different items whose only common denominator is that they are sold under the same roof. Imagine the fantastic confusion that would result if this hugh operation were required to be accounted by the "journal ledger" method. Each time a clerk sold something he would have to go to the "book" and record the effect of his sale on the inventory balance. Very likely, before he even found the item he was looking up in the book, several more clerks would line up behind him for their turns to record a sale. In a busy store there would probably be more clerks than customers. Extrapolating this manpower situation to the entire economy, so many people would be required for selling that there wouldn't be anyone left to do the buying. At least a natural balance would be struck, where there would be just enough people left as customers to keep all the others busy as clerks.

In contrast, retail sales accounting has been mechanized by the use of cash registers, adding machines, price tags, and punched cards to a point where accounting costs require only a minor percentage of the price of a sales item. RAMAC accounting is a pioneering attempt on the part of IBM to drive this cost down even further. In this system the old "journal ledger" is replaced by the disk file, which, in the 305 system, has a capacity of 50,000 100-character records and access times of 50 minutes minimum, 250 minutes average, and 750 minutes maximum. Compare this to the time required for accounting using the ledger-book. The remainder of this review will cover the electro-mechanical aspects of the disk file. The attached figures are taken from the IBM 305 customer engineering instruction manual.

Figure 64 shows the disk array, which consists of 50 aluminum disks, 24 inches in diameter and coated on both sides with iron oxide (similar to magnetic tape). The array is driven from the bottom, through a gear-box, at 1200 rpm. The resulting revolution period of the array is 50 milliseconds. There are 100 concentric information tracks on each of the

*Registered Trademark, IBM Corp.

100 disk sides in the array and these are further divided angularly into five sectors of 100 characters each, providing a total file capacity of 5,000,000 characters. This is comparable to the character capacity of a 2400-foot reel of magnetic tape. The record access time is much less than with tape because it is direct rather than sequential.

Figure 65 is an exploded view of the access arm which straddles the selected disk. It carries two read/write heads of the type shown, one for the top and one for the bottom of the straddled disk.

Figure 66 shows the relative positions of the access arm and the array in top and side views.

Figure 67 presents the construction details of the read/write head. The upper view demonstrates how the coils are assembled on the core, "potted," and machined. The other two views show the read/write head mounted in the air head, which, lubricated by a film of moving air, floats the read/write head against the disk surface.

Figure 68 shows the mechanical relationship between the track detents and the access arm. The arm is required to be positioned accurately and repeatably on each of the 100 tracks. In operation the access arm is driven very close to its final position by a servo system controlled by the position error of the arm. Then the track detent locks the arm "dead on" the selected track.

Figure 69 illustrates the basic elements of the access mechanism, which drives the arm off the disk from which it starts, up or down to the newly selected disk, and in to the selected track. The access arm is driven out, up and in, or down by a reversing clutch. The arm is coupled to the clutch by an "elevator" cable. The arm can move up or down only if it is mechanically fully retracted from the array. It can move in or out only if the disk detent, shown unlabeled behind the access arm, is seated.

Figure 70 is a detailed drawing of the magnetic clutch assembly showing its construction to be quite similar to that of the drive clutches on the 727 tape drive unit.

Figure 71 shows the arm retraction mechanism, which removes the arm from and drops it below the array when the file "shuts down" for any reason including failure of the normal access drive controls.

Figure 72 is a schmeatic of the logic of the access control circuits. A specific one out of hundred taps on a Markite* potentiometer is selected by the "disk" portion of the file address by the central processing unit of the system. The uniform voltage gradient across the potentiometer generates a position-error voltage at the wiper, which moves with the up-down motion of the access mechanism. The polarity of this voltage determines via the clutch amplifier whether the arm moves up or down. When the potentiometer "nulls," the disk detent operates pneumatically to align the fork of the access arm accurately with the selected disk. It also operates the disk detent switch, which feeds the track position-error voltage to the clutch amplifier for the arm-in motion. When the wiper of the potentionmeter "nulls" with its controlling track address, the track detent operates and the access operation is completed.

* Trade name

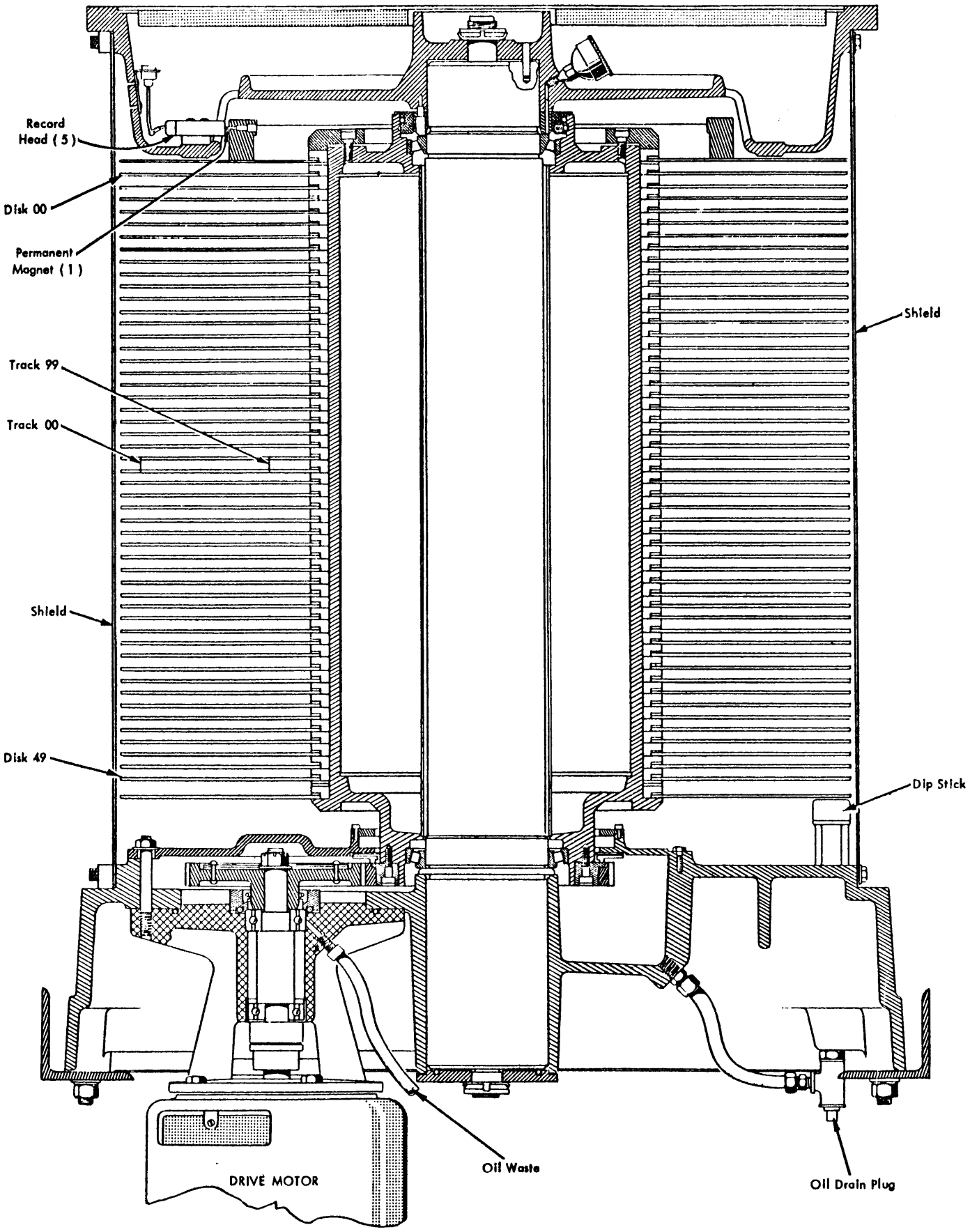


Fig. 64 - Disk Array of the RAMAC Disk File

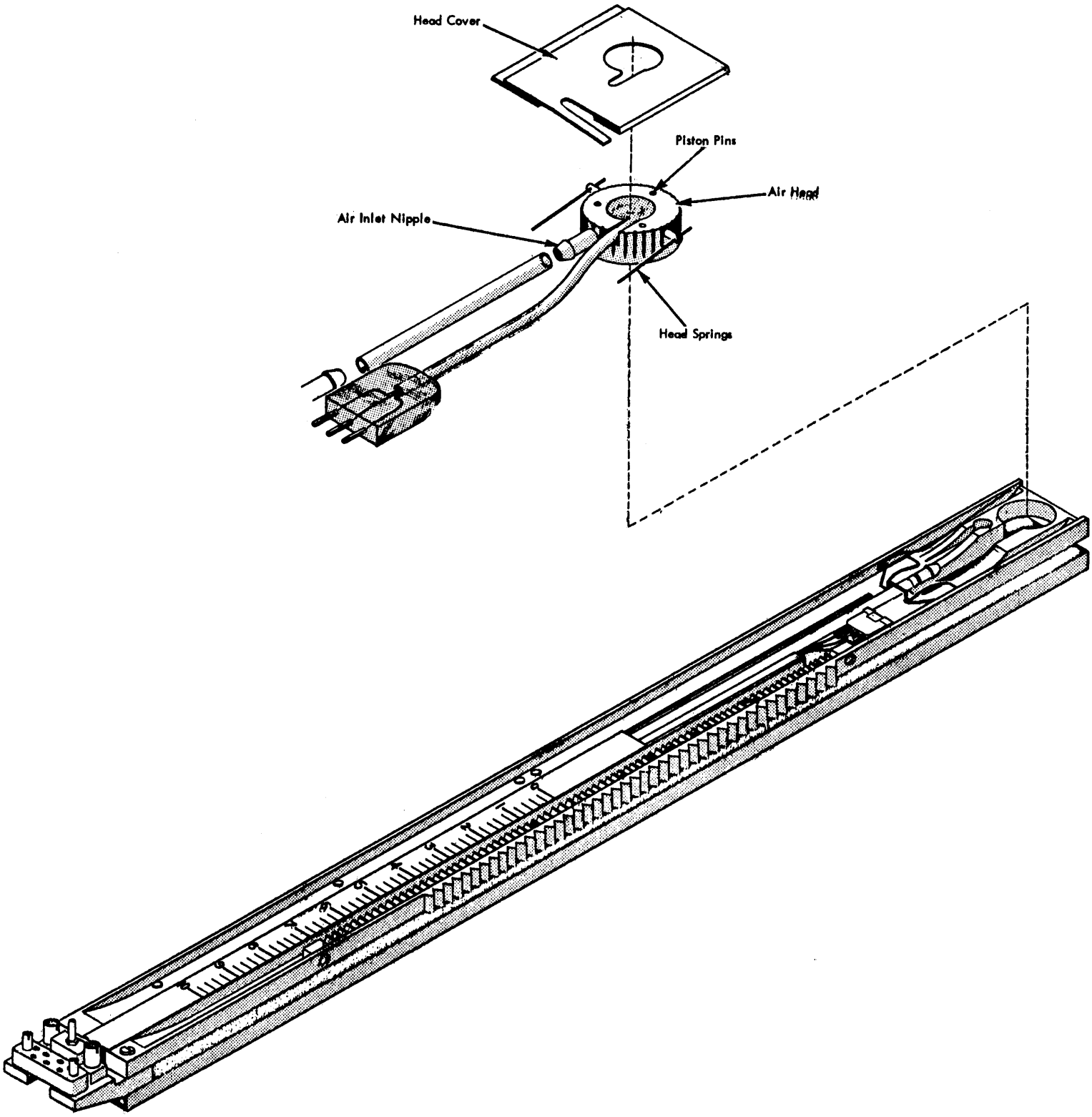
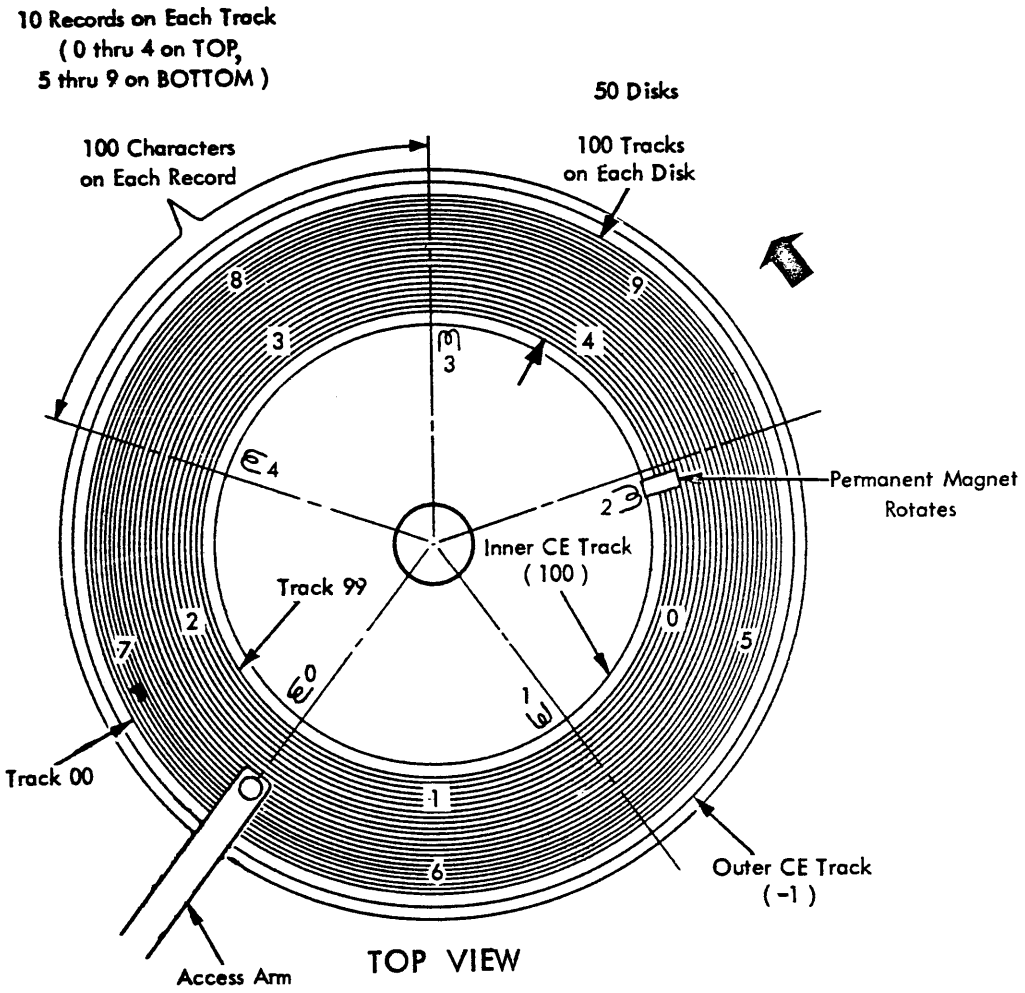


Fig. 65 - Exploded View of the RAMAC Access Arm



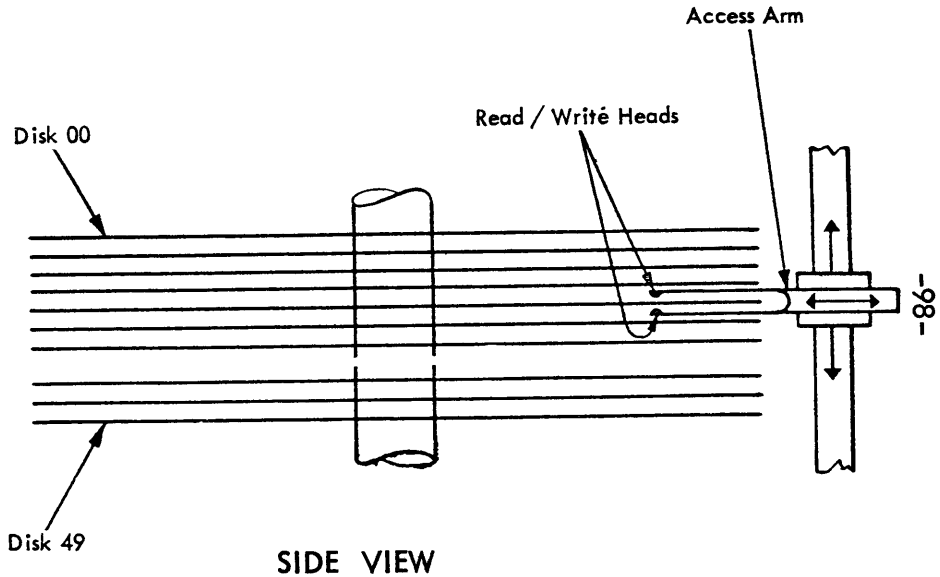
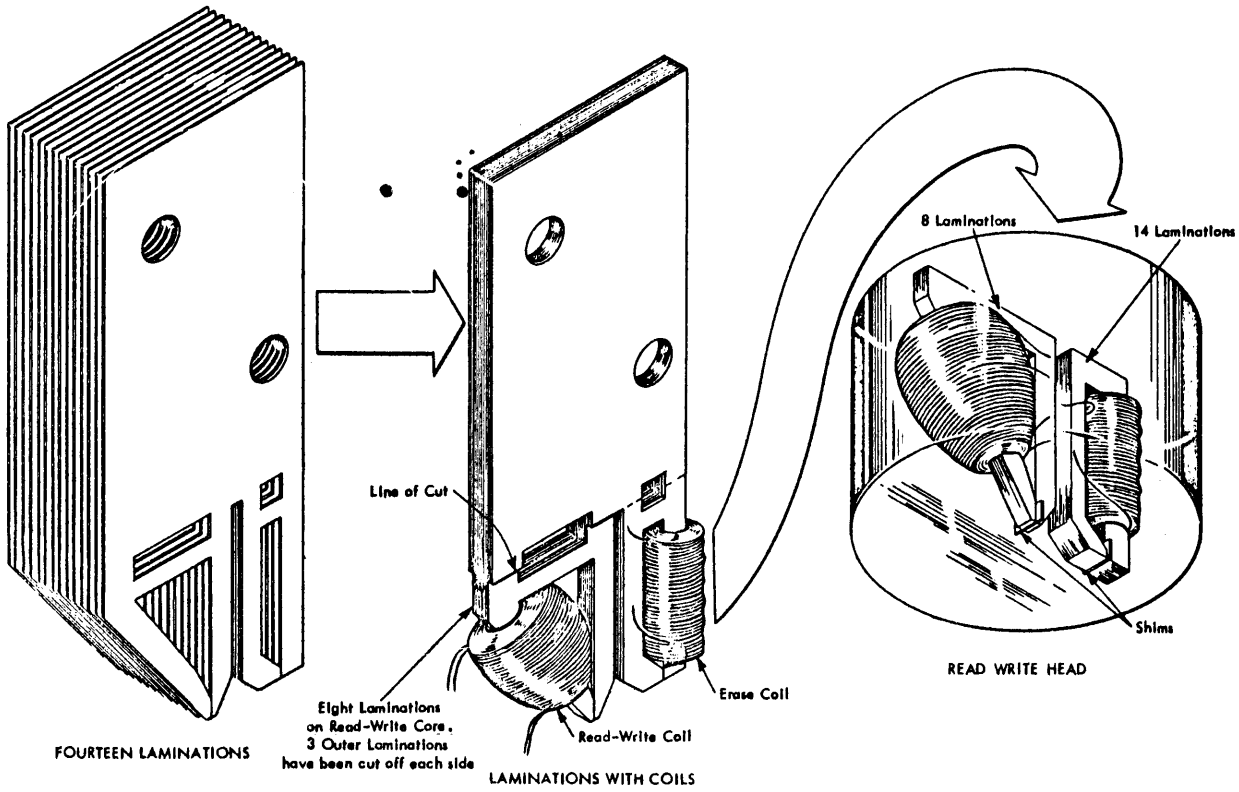
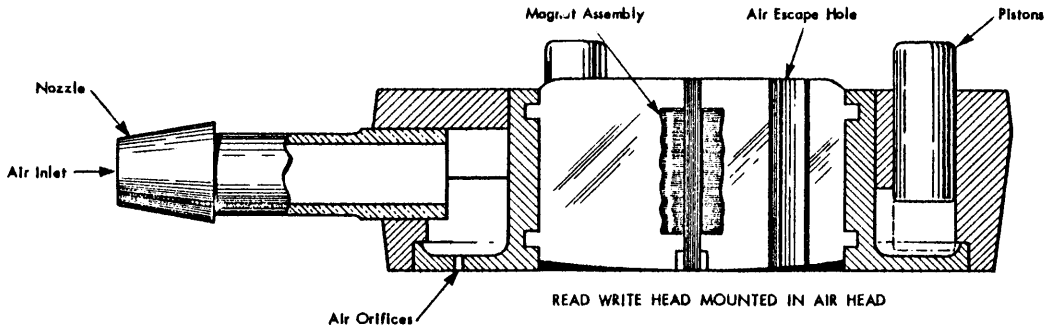
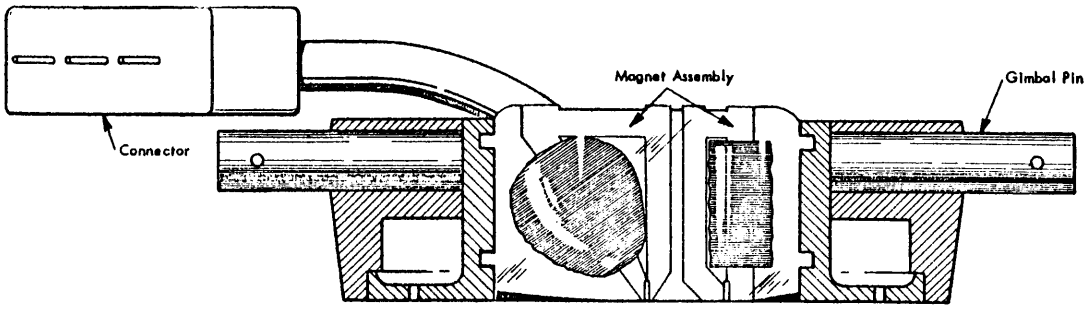


Fig. 66 - Positioning of the Access Arm and Disk Array







ALL OF THE ABOVE
DRAWINGS ARE FIVE TIMES SIZE

Fig. 67 - Construction Details of the Read/Write Head

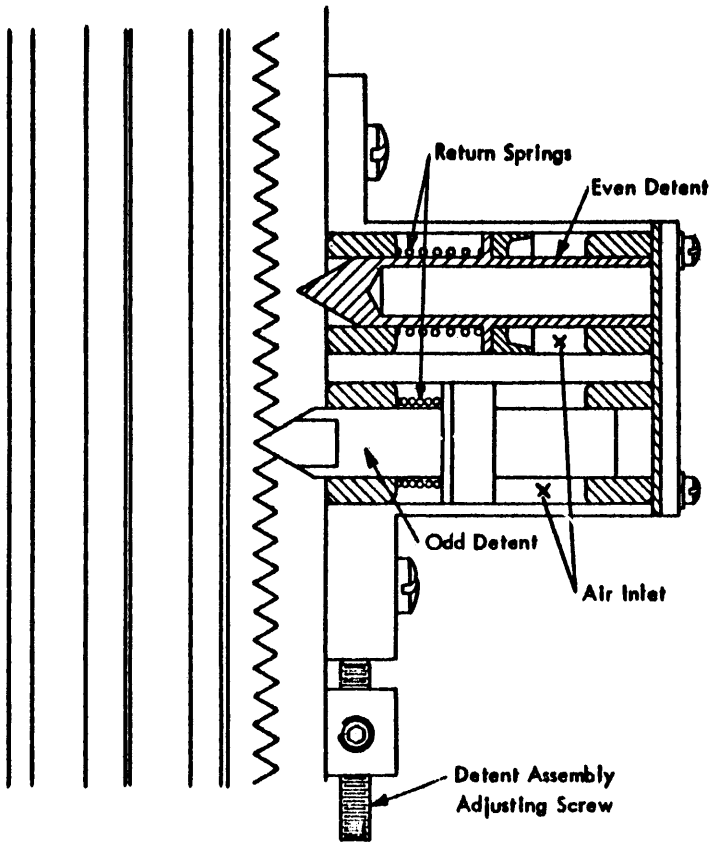


Fig. 68 - Track Detent

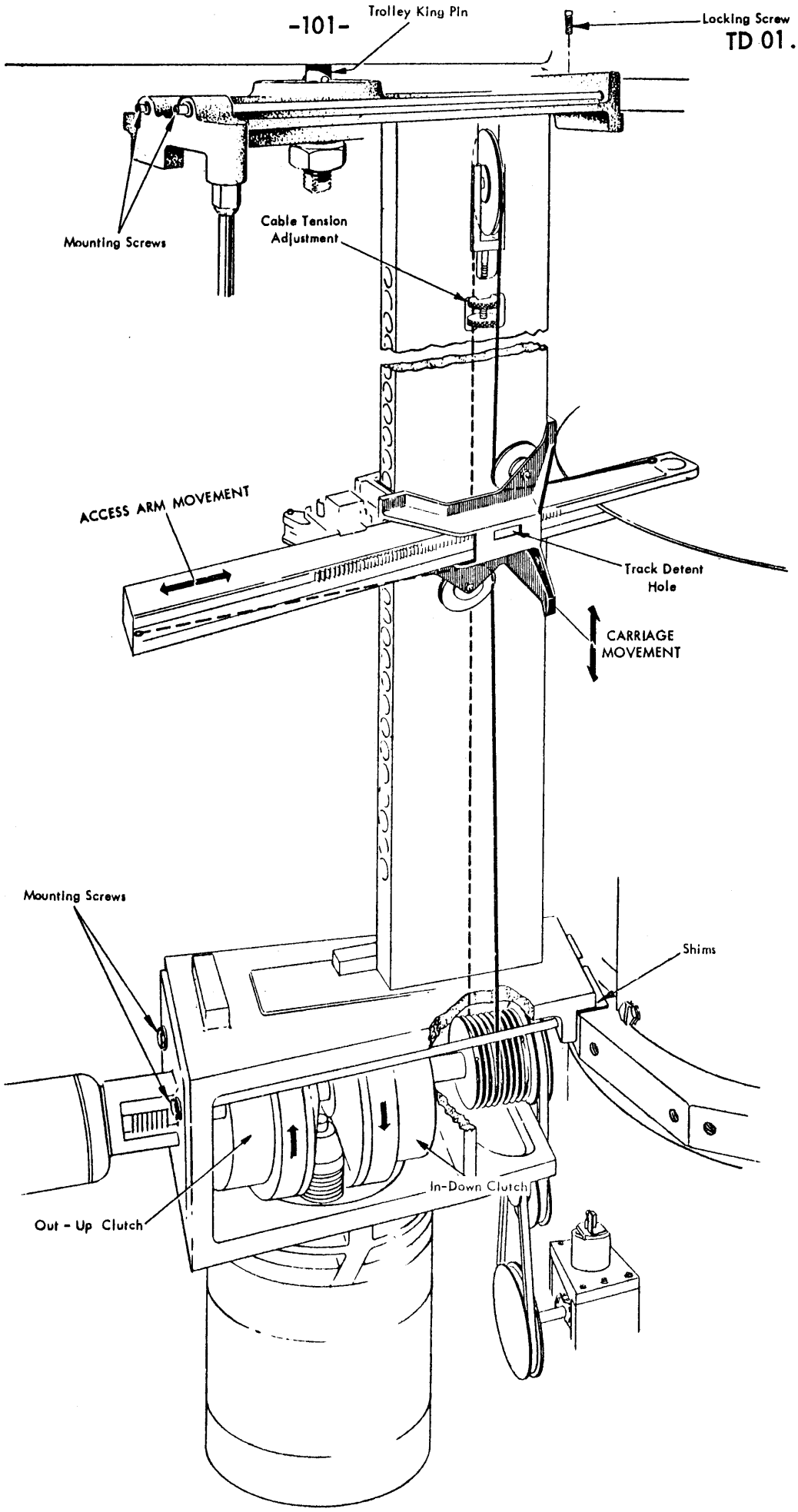


Fig. 69 - RAMAC Access Mechanism

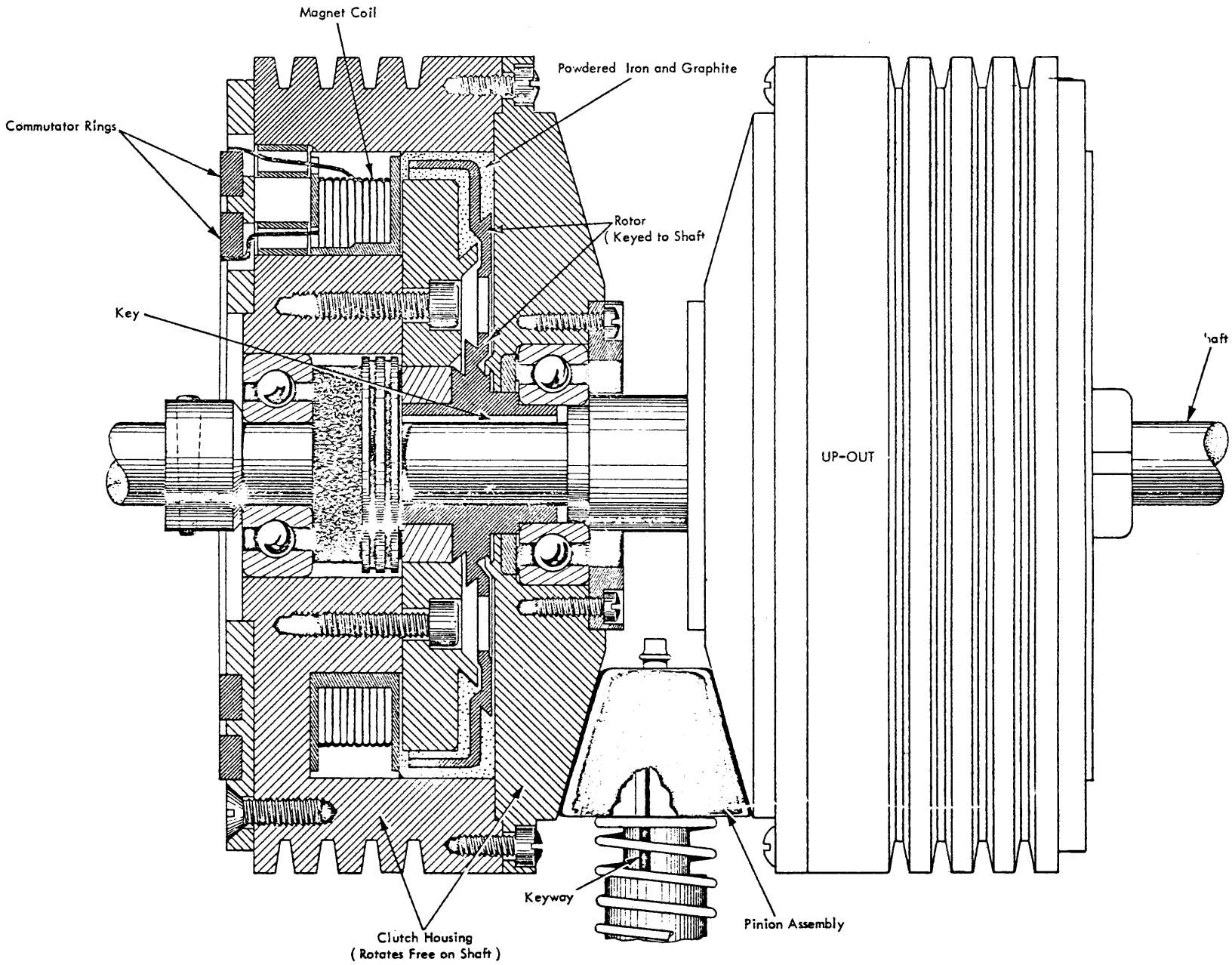


Fig. 70 - Magnetic Clutch Assembly

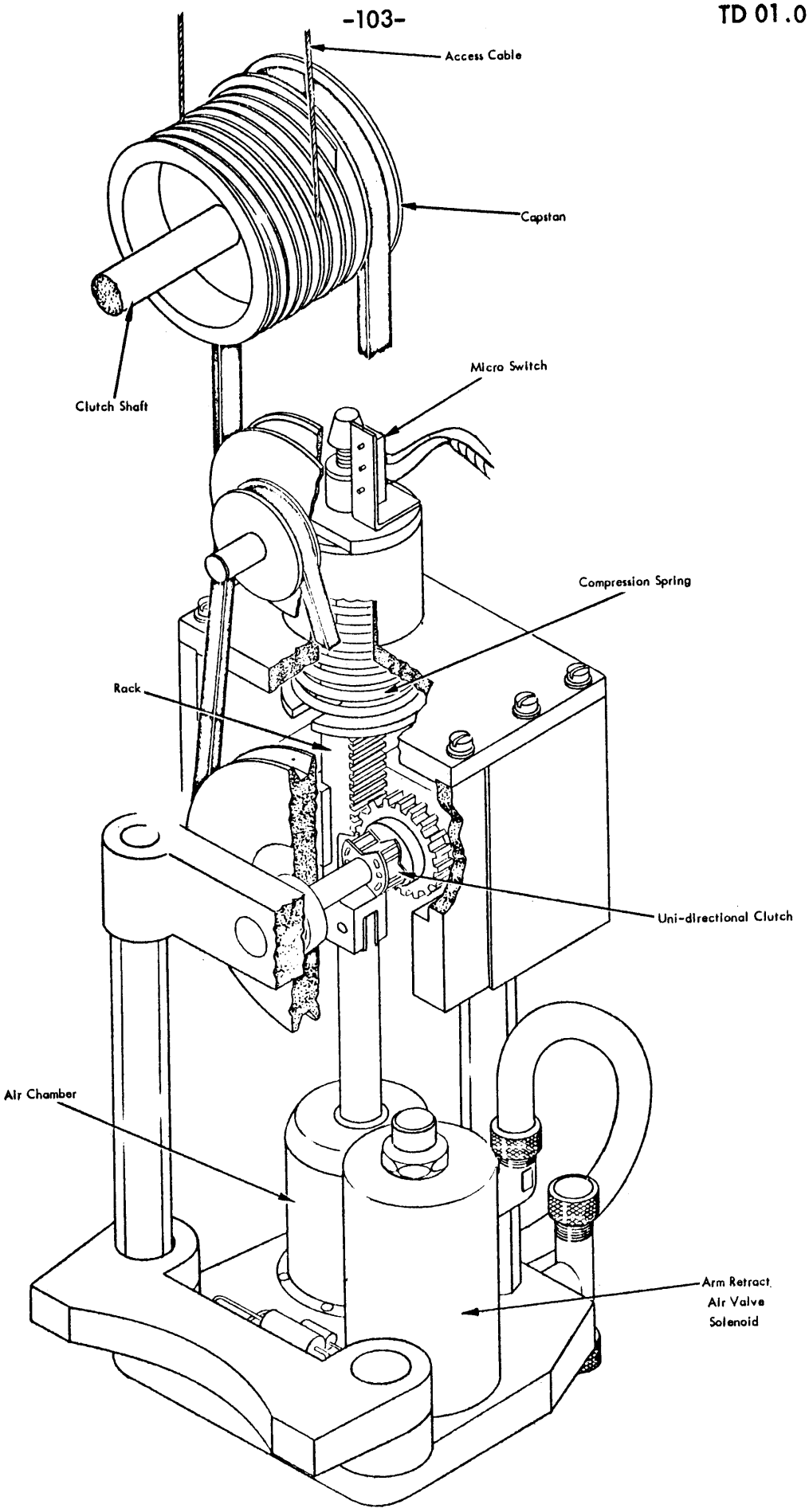


Fig. 71 - Arm Retraction Mechanism

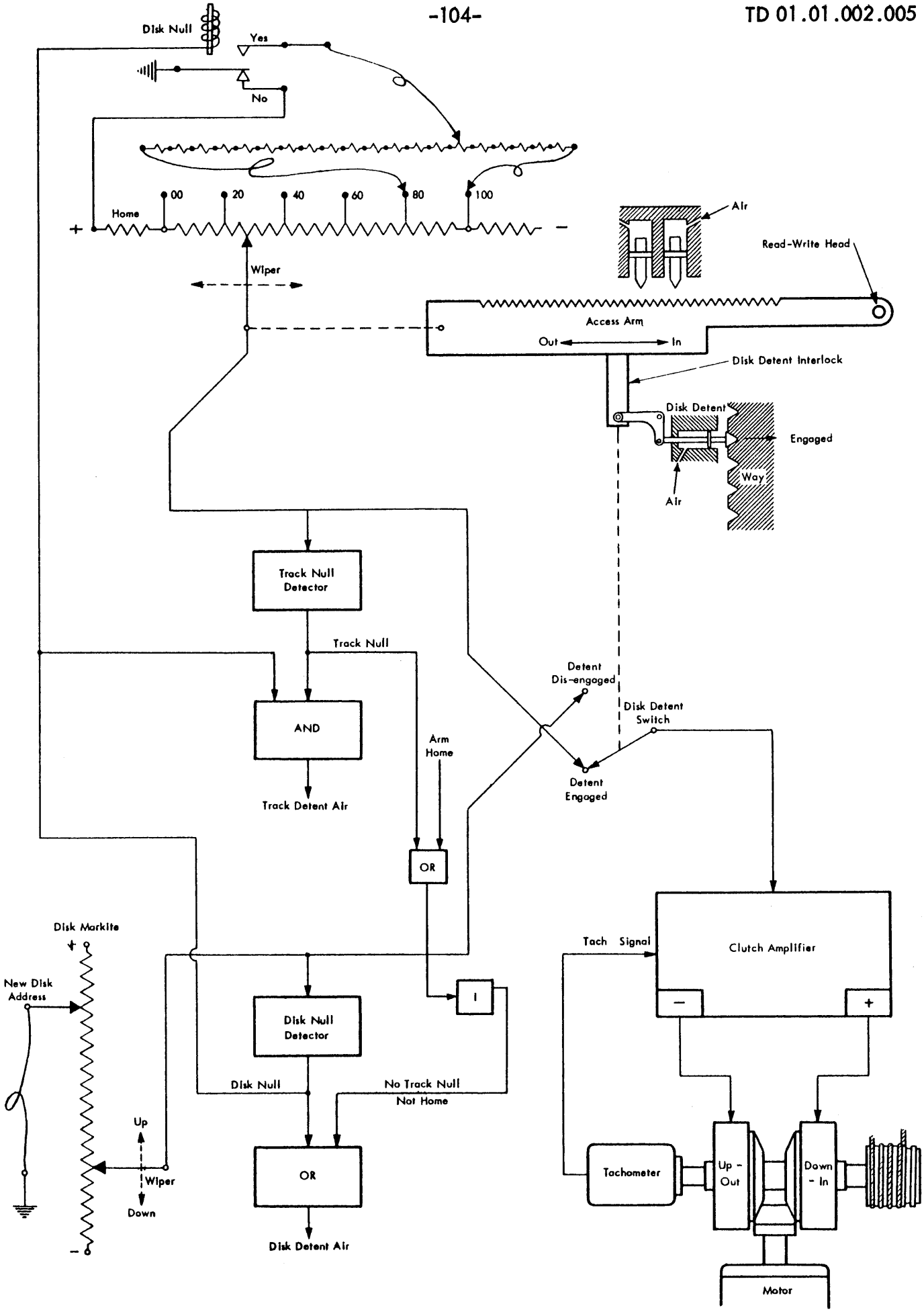


Fig. 72 - Access Control Circuit Logic

XXVI CONCLUSION

Just as each of the systems discussed has added a list of accomplishments to the history of data processing, succeeding generations of processing systems will make significant contributions toward advancing the collection, processing, and evaluation of data. Though many trends stand out in the history of digital computation, the same three logical functions — AND, OR, and NOT — are always used as a foundation for new equipment.

While memory access times and operation times have greatly decreased, memory capacities have correspondingly increased. For example, the IBM 305 system has a memory capacity of 50,000 100-character records and random access time of 750 milliseconds maximum, while a popular form of abacus has a capacity of 13 characters and an access time dependent upon the skill of the human operator.

Along with the improvements came an increase in equipment volume, power requirements, and circuit complexity. To solve these problems, computer engineers have developed new solid-state techniques and diagnostics concepts.

It is significant that in the history of digital computation the philosophy has remained fundamentally unchanged; the use of some discrete entity for counting, whether it be beads or electrical impulses, has remained the foundation upon which the present data-processing system is built. It is possible that this foundation may be significantly altered in the future. Attempts are now being made to formulate and exploit multi-valued logic to replace the Aristotelian logic which has been used so effectively in the past.

XXVII BIBLIOGRAPHY

1. Faster Than Thought, E. V. Bowden (Pitman, 1953).

2. 305 RAMAC Reference Manual, Form A26-3502.

3. 650 DPS Bulletin Form G24-5000
   650 DPS Bulletin Form G24-5003
   650 DPS Bulletin Form G24-5005

4. 7070 General Information Manual, Form D24-7004.

(For more detailed technical descriptions of the 604, 305, 650, and 7070, the reader is referred to the respective Customer Engineering Instruction Manuals, listed in the Stationery Stores Forms Catalog and available on the same basis as the above.)