D. RICHES

An analysis of Ludgates machine leading to a design of a digital logarithmic multiplier.

department of electrical and electronic engineering

university college of swansea wales united kingdom

university college of swansea

wales united kingdom

department of electrical and electronic engineering

author D. RICHES

title An analysis of Ludgates machine leading to a design of a digital logarithmic multiplier.

supervisor _____

approval _____

date _____

Abstract:

This paper documents an analysis of a design by Percy Ludgate, during the years 1903-1909, of a program-controlled mechanical calculator, or analytical engine. The machine is then compared with its modern counterparts, leading to a design of an electronic digital multiplier using a logarithmic principle.

C O N T E N T S

I N T R O D U C T I O N

A short account by Ludgate on his analytical machine appeared in the Scientific Proceedings, Royal Dublin Society, April 1909. A paper documenting a search for further information on Ludgate and his machine was carried out and documented by Randell in 1971. [1] No new information concerning the machine's design was discovered, making the 1909 paper the only source of information on which this report could be based.

Ludgate is rarely mentioned in the introductions to standard computing texts, with the result that they jump from Babbage's work (started 1834, ended 1871) to that of Aiken (started 1931) when considering program-controlled calculators.

One purpose of this paper is to document an analysis of Ludgate's machine after an attempt to 'expand' the information in the report.

The design is then compared to conventional systems with a view to determining any aspects of the machine not yet used in modern machines. This leads to a new design of digital electronic multiplier based on the logarithmic principle used in Ludgate's design.

Finally the practicability of this type of multiplier and further applications of the logarithms are discussed.

S Y M B O L S

A_n))) B_n)	Adder inputs to nth stage
C_n	carry from nth stage adder
CSA	carry-save-adder
LS	least significant
MS	most significant
ns	nanosecond
$\sum n$	sum from nth stage adder

See Appendix C for logic symbols

C H A P T E R O N E

HISTORY OF CALCULATING MACHINES

The intention of this section is to give an outline of the major advances in calculating machines, prior to the publication of Ludgate's paper in 1909. This has been summarised in a time chart (figure 1).

The examination and reading of calculating machine designs [ref. 2 to 10] greatly helped in the analysis of the proposed analytical engine.

The early digital mechanical calculators can be divided into three groups, desk calculators, difference engines and analytical engines. The desk calculators were intended to reduce the time and errors of simple calculations and were not automatic. The earlier type of this calculator, such as Pascal's, could only add or subtract. In 1671 Leibniz produced a design for a calculator that could also multiply. It was not until 1820 that the first calculator capable of all four basic arithmetic operations was made for commercial manufacture. This was the 'Arithmometer' of C.X. Thomas. The 'Millionaire', designed by Steiger, was a more notable early success and was again capable of the four arithmetic operations. The 'Millionaire' and a machine by Bollé were unusual examples of calculators in that they performed multiplication by a direct method, and not successive addition as in most other machines [4]. In these machines the operands had their separate digits multiplied by what was basically a table look-up

system.

Around 1786 a new type of calculator design was conceived by Muller. This was the difference engine. In 1812, probably unaware of Muller's work, Charles Babbage began the design of a difference engine and in 1822 completed a working model. A difference engine is only suitable for the automatic calculation of mathematical tables of functions whose higher order differences are constant. The engine or machine comprises a register for each order of difference and mechanism for adding the data in each register to that of the next lower register. A concise description of the principles of such a machine has been written by Babbage [2] . Despite its simplicity of action, it was not until around 1854 that a difference engine, with a useful number and size of registers, was constructed. This was designed and built by Scheutz. Babbage never completed the construction of a full-scale version of his difference engine and in 1833 turned his attention to designing an analytical engine.

Babbage's analytical engine was to be more powerful than any difference engine, with the results of calculations being able to affect the future instructions of the machine. It was to be capable of automatically computing any algebraic formula for which there was a solution, given the initial values. The machine was to be controlled and data communicated to it by punch cards.

The principles of punch card controlled machinery were first demonstrated by Bouchon between 1725 and 1745. By 1804 Jacquard was using this method of control for weaving cloth automatically. A short account of punch card control appears in Morrison [9] .

There are several descriptions of the principles of Babbage's analytical engine [2, 7, 8]. The original drawings and parts of the machine can be seen at the Science Museum, London. The designs for the engine included a mill where the arithmetic operations of addition, subtraction, multiplication and division were to be performed. Numerical data was to have been stored on 1000 columns of wheels. Each number was to have a separate column and the position of rotation of each wheel was to represent a separate digit. The numbers were to be to 50 decimal places. Babbage intended to use two different sets of punched cards for conveying numerical data and algebraic formulae (program) to the machine. Shortly before his death Babbage built the arithmetic and printing units. His son, Henry Babbage, continued working on the analytical engine after his father's death but it was never completed.

It should be realised that at the time Charles Babbage was designing and building his analytical engine, desk calculators with four arithmetic operations had not yet reached any notable commercial success.

It was in 1909 that Ludgate published what appears to be his only report concerning his design of an analytical engine [11]. He mentions here that in the early stages of his work he had no knowledge of Babbage's engines and that later on he only had a slight knowledge of them, limited mainly to their mathematical principles. It is clear from reading reports on Babbage's and Ludgate's analytical engines that they are mechanically completely different and that the principles behind each can only be compared on a general level.

Ludgate states briefly in a later paper [7] that he had nearly completed a second design,

" .... in which are combined the best principles of both the analytical and difference types, and from which are excluded their more expensive characteristics."

No further information on this design can be found.

From this summary of calculating machines it can be seen that Ludgate's work appears original and worthy of investigation. The following two chapters of this report refer in detail to his first design of an analytical engine.

CHAPTER TWO

DESCRIPTION OF LUDGATE'S ANALYTICAL MACHINE

2.1 DEFINITION OF AN ANALYTICAL ENGINE

It is convenient to start with a more detailed description of what Ludgate considered the necessary operations and facilities of an analytical engine. This will give a clearer picture of what Ludgate was designing.

In his 1909 paper Ludgate wrote that the object of his work was to design

> "machinery capable of performing calculations, however intricate or laborious, without the immediate guidance of human intellect."

The required operations of an analytical engine can be extracted from his report and can be enumerated as follows:

1. A form of communication between machine and operator.
2. ".... means of storing the numerical data of the problem", plus the intermediate results and final answer(s).
3. Capacity to submit ".... any two of the numbers stored to the arithmetical operation of addition, subtraction, multiplication or division."
4. The ability ".... to follow a particular law of development as expressed by an algebraic formula."

5. The ".... changing from one formula to another as desired, or in accordance with a given mathematical law."

6. The capacity to ".... 'feel' for particular events in the progress of its work"....." and also to make any pre-arranged change in its procedure, when any such event occurs."

It is worthwhile noting at this stage that these basic requirements also occur, but in a different form, in a much later report (1946) by von Neumann, Goldstine and Burks concerning "Preliminary discussion of the logical design of an electronic computing instrument." [12]

2.2 MATHEMATICAL THEORY BEHIND LUDGATE'S MACHINE

The fundamental action of Babbage's machines was addition, whereas in Ludgate's it was to be "direct" or "partial product" multiplication. In a machine whose fundamental action is addition, subtraction is performed by reversing the process, multiplication by successive addition and division by successive subtraction. In contrast the basic action of the arithmetic unit in Ludgate's machine was to be multiplication by a logarithmic method.

In his report he metnions that he originally intended to use ordinary logarithms to base ten,

"... but found that some of the resulting intervals were too large, while the fact that a logarithm of zero does not exist is an additional disadvantage."

He therefore arranged for each of the prime numbers below ten to have associated with it an index number. The indexes or logarithms of non-prime numbers, i.e. all possible products of prime numbers up to 9 x 9 = 81, are formed by adding the index of the prime numbers that form that product. Appendix 'A' contains an algorithm for writing such a system of logarithms.

The index numbers of the base ten system that Ludgate used are shown in table 1. The use of these index numbers in multiplication can be explained as follows. When two single digit numbers are to be multiplied then the corresponding simple index numbers are added. This result is called a compound index number and by referring to table 2 the corresponding product is found. Two examples are shown in table 1.

Multiplication of numbers of more than one digit is broken down into multiplication of single digits. An example of this, using tables 1 and 2, is shown in table 3. The allowance for the factors of ten in the multiplier and the 'carry' digits in the partial product addition were to be mechanical.

Addition was to be performed by using the same mechanism as in multiplication and was to be effected by multiplying the addendums by unity. These products were then to be summed as were the partial products of a multiplication. Subtraction was similar to addition except in the final stage. Here a train of gears for accumulating the partial products was to be rotated in the reverse direction.

Division was to be performed in what was then an equally unusual way as the multiplication [13] . The basis of the scheme was to

multiply the dividend by the reciprocal of the divisor. The reciprocal was first to be estimated by what may be considered a table look-up system. This estimate was then to form the basis of a highly convergent series in which only addition, subtraction and multiplication would be required to solve its sum.

The theory of the division is as follows:

Assume the Quotient $= \frac{D}{d}$

The reciprocal of the three most significant digits of 'd' is found by table look-up. Let this be 'A' (In the proposed machine A was to be a decimal of 20 figures.)

$\Rightarrow$ A.d begins with the decimal digits 100 ...

Let A.d be of the form 1.00 ... or 1 + x where 'x' is the small fraction

$$\text{Then } A.d = 1 + x$$

$$\therefore \frac{1}{d} = \frac{A}{1+x} = A(1+x)^{-1}$$

Expanding by the binomial theorem

$$\frac{1}{d} = A(1 - x + x^2 - x^3 + x^4 - \ldots \mp x^n \mp \ldots)$$

$$= A(1-x)(1+x^2)(1+x^4)(1+x^8)\ldots(1+x^{2n-2})\ldots$$

Therefore

$$\text{Quotient} = \frac{D}{d} = D.A(1-x)(1+x^2)(1+x^4)\ldots\ldots$$

By taking this last series as far as x^{10} the result is given correct to at least thirty figures. Ludgate states that the position of the decimal point was to be found independently of the formula

and by a mechanical method. This was to be so for multiplication, addition and subtraction also.

The numerical data was to be represented in the form of twenty digit variables with an extra sign digit (i.e. sign magnitude representation).

2.3 MECHANICAL DESCRIPTION

Randell's search for information [1] concerning the analytical engine suggests that Ludgate never succeeded in having his machine constructed. The drawings and manuscripts have not been traced despite a thorough search. Thus the 1909 paper is at present the only source of information on the machine and this was only written to serve as a short account of his work. From his report it has been possible to derive sketches of the machine, though there is no way of telling at present how closely these drawings follow the original design. They do, however, correspond exactly to the description in Ludgate's report and also serve in the understanding of the machine.

The Store

Ludgate intended his machine to have a store of 192 variables of twenty digits plus a sign digit. Each variable was to be stored in a shuttle, the individual digits being represented by protruding rods (see figure 2(a)). Two "co-axial cylindrical shuttle-boxes," divided into compartments parallel to their axis, were to hold the 192 variables. The inner and outer shuttle-boxes would each contain 96 shuttles, (see figures 2(b) and 2(c)) and the store would therefore

be divided into two distinct sections.

Ludgate states that both the number of variables and the number of digits in each could be increased. Also he mentions that new shuttles, representing new variables, could have been introduced after removing the old ones. To access a variable the store was to be rotated until the variable was brought opposite a 'shuttle-race' or track. The shuttle was then to be drawn out along the race. There were two 'shuttle-races' in the design, one for the outer store and one directly below for the inner store. Thus two variables on the same axis but different stores could be accessed simultaneously. This can be seen more clearly in figure 2(c).

It appears from considering aspects of the analysis not yet mentioned, that when programming the machine, any two variables to be multiplied may have to have been stored on the same axis. There is no suggestion in the report that two variables on different axes in different stores could be multiplied directly. This may have been possible, however, by taking a shuttle from one of the stores, rotating the stores, and taking another from the other store. The variables would have been multiplied and the shuttles returned to their respective locations. This type of operation could have been avoided but storage would have been wasted. The machine's capacity to do this would have determined its control mechanism complexity.

The Arithmetic Unit

A system of slides, called an index, was to convert the variable on the rods of an outer shuttle into distances corresponding to the simple index numbers of that variable. These slides were probably

to be stepped as shown in figure 3(c) and were to be in line before operation as in figure 3(a). They were to be released and moved forward until stopped by striking the 'type' of a shuttle as in figure 3(b). The distances moved are logarithmic displacements of the twenty digits in the shuttle.

In the design, the protrusion of the rods in a shuttle is governed by table 4. The slides are stepped in distances corresponding to the simple index numbers, as shown in figure 3(c).

Figure 4 is an overall sketch of the arithmetic unit of the machine and should be referred to when reading the following description.

The design includes only one bank of slides, running against the rods of an outer shuttle. A single slide moves in the opposite direction to these slides and runs against the most significant digit of an inner shuttle. This single slide is attached to the index, which then also moves a distance corresponding to the simple index of the most significant digit of the multiplier. The resultant displacements of the slides are compound index numbers, (i.e. simple index number of most significant digit of the inner variable added to each of the index numbers of the digits of the outer variable).

It is not clear in the report how Ludgate intended to convert these relative displacements into the partial product. He states:

> "The numerical values of the readings" (compound index numbers)"are indicated by periodic displacements of the blades mentioned, the duration of which

displacements are recorded in units measured by the driving shaft on a train of wheels called the mill".

After these compound index numbers had been converted to the partial product and added to the mill mentioned above, the process was to be repeated again. The single index rod then runs against the second most significant digit of the multiplier. The mill, being a train of gears, was to be capable of allowing for the carrying of tens when summing the partial products and allowing for multiplication by ten each time a partial product is added.

There is no mention of how the results in the mill were to be returned to the store. Figure 4 includes a possible method, again using stepped bars.

Multiplication of two twenty digit numbers was to have taken ten seconds and division about 1½ minutes maximum. Add and subtract were multiplication by unity. It appears from timings given in the report, that to add or subtract necessitated the variable being in the outer store. These operations were then to have taken a time of three seconds each. If it was possible for the variable to be in the inner store, it would have been multiplication of unity by the variable with a time corresponding to that of an ordinary multiplication.

There is no mention of how the machine was to deal with the double length words that may be produced in the arithmetic unit. Presumably there was to be no provision for numbers greater than twenty decimal digits. Neither does the report indicate whether

overflow of this type could be indicated by the machine and how such a number was to be reduced to a single word length.

Control, Input/Output

Ludgate's method of control was similar to Babbage's in that they both used a Jacquard system. Babbage was to use perforated cards while Ludgate designed his machine to use perforated tape or 'formula-paper'. The information in the perforations or holes was to be converted to mechanical displacements by rods. The absence or presence of holes was to determine the instruction in mechanical movements. One function of this 'formula-paper' was to select where the variables were to be stored and which shuttles were to be operated upon. Each row of perforations was to direct the machine one step of a calculation, i.e. a complete multiplication including the accessing of the two variables.

For calculations that did not warrant a 'formula-paper', the design included a keyboard. This was to control the machine and also to act as a means of punching a new 'formula-paper'.

A second keyboard was included by which numbers were to be communicated to the machine. The machine was also to be able to produce a 'number-paper' which could record numerical data. This could then be replaced in the machine and the data re-entered.

To avoid having to repeat a series of instructions on the 'formula-paper' every time a divide instruction occurred, Ludgate devised a subroutine. When a division was indicated on the 'formula-paper', control was to be passed to a dividing-cylinder. This cylinder was to contain the required perforations.

The reciprocal of the three most significant digits of any divisor was to be found from a cylinder, the 900 possible values being represented as twenty digit numbers in the form of holes one to nine digits deep. Rods were to transfer the reciprocal to a shuttle.

A logarithmic cylinder was also to be included. This in principle was similar to the dividing cylinder but with the perforations for calculating a logarithmic formula. Ludgate intended his machine to accept control cylinders of different functions.

One of the powers of the machine was to be its ability to change control from one formula to another "in accordance with a given mathematical law." It was to "feel" for events in calculations such as changes in sign or approaches to infinity. Ludgate only mentions this briefly. This form of conditional branching was probably to have been accomplished by skipping specified rows of instructions on the 'formula-paper'.

There is no way of determining exactly the instructions that were to control the machine. This is largely due to not being able to determine which pairs or single shuttles can supply variables for an arithmetic operation.

The ingenuity of the design can be appreciated when the size of the proposed machine is considered. Ludgate gave the dimensions as 26 inches long, 24 inches broad and 20 inches high. This is considerably smaller than the engine designed by Babbage which would have been measured in feet.

CHAPTER THREE

COMPARISON OF LUDGATE'S MACHINE WITH ITS MODERN COUNTERPARTS

One purpose of comparing Ludgate's machine with its modern counterparts is to determine whether there is any undeveloped aspect of his design which could be used in a modern computer.

3.1 LUDGATE'S MACHINE TODAY

Firstly the information about the machine has to be put into a form that can be compared with modern designs. The block diagram (figure 5) gives a basic layout of the engine. The control sections are omitted since they cannot be precisely defined. Each 'block' in the diagram is under the direction of a control system which in turn receives instructions from the keyboard or 'formula-tape'. It is not possible to judge whether the input/output of numerical data was to be via the accumulator or direct to the store.

The store is represented as two separate units. The removal of the shuttles from the store to access the data has been represented by a memory buffer. The register I_o corresponds to the slides of the index which were to effectively store the variable by their displacements, while multiplication took place.

An example of what one line of perforations across the formula-paper could direct the machine to do, is the selection of

two variables and the multiplication of them. This is stated in the report. A set of instructions on which the engine could operate has been derived and tabulated in table 5. It is possible to have such a machine working on the operation codes one to eleven in the table. This restricts its multiplication/division to variables with the same address but in different stores, (i.e. same axis). This may have been overcome by data being transferred around the store via the mill, though this process would have been very slow.

Operation codes twelve to fifteen allow temporary storage of data on the index slides and a temporary storage register 'T' common to both stores. These would have permitted faster arithmetic operations on variables stored anywhere.

The format of a possible instruction word is given in figure 6. This shows the store as being two interleaved stacks with double addressing for adjacent locations, i.e. two bit store address and seven bit word location address.

Figure 7 gives a flow chart of the sequence of stages in such a machine for the multiplication of two variables with the same address but different stores. An explanation of this is given opposite figure 7. Figure 9(a) gives the instruction words for such an operation following path ✳ of the flow chart. Multiplication of variables stored in different formations is explained in figure 8. From figures 7 and 8, it is obvious that for the lowest multiplication time, the variables must be stored with the same address in each store. This enables them to be accessed simultaneously.

3.2 COMPARISON

In the conclusion to his report and by considering the previous chapters it is clear that Ludgate intended his machine for mathematical use. Therefore it should be considered an 'early scientific computer'.

The basic layout of the machine closely follows that of today's machines as did Babbage's design. They all have arithmetic, control, input, output and store units.

Store

Besides the main store, the machine was to have a read only store for finding the reciprocals in the division routine. This can be described as a predecessor to some of today's read only memories.

It appears that Ludgate was attempting to reduce the number of separate store accesses by dividing the main store in two and and enabling double access. This facility has been included in the Atlas machine [14] . Here the core is split into four stacks, each having its own read, write and decoding mechanisms with a page of instructions spread over two stacks. Due to this, it is possible to read a pair of consecutively stored instructions in parallel. Another machine that Ludgate's engine can be compared to in this aspect is the I.B.M. 7094. This has a facility for requesting two 36-bit words as the actual memory has a 72 + 1 bit parity word for even and odd addresses [15] . The Control Data 'Star' computer has the facility of removing several operands from its store simultaneously.

The total capacity of 192-variables seems very low, although it was mentioned that this could have been increased. There was provision in the design for perforated paper storage outside the machine which can be compared to magnetic tape storage of today.

The proposed data word length of twenty decimal digits (requiri sixty bits for binary representation) is large even by today's standards. It is comparable with the I.B.M. Stretch computer [16] a the early electromechanical machine, Harvard Mk 1 which had twenty-four wheels representing twenty-three decimal digits and a sign digit.

The cyclic access principles of Ludgate's store were used in most vacuum tube computers but today they form the basis for some types of backing store (e.g. disc).

The removing of shuttles from the store to the index may be considered a destructive read. The variable must be re-written into store (i.e. shuttle replaced).

The advantages for Ludgate of storing instructions in the main store, as in most electronic computers, would have been greatly outweighed by the mechanical disadvantages of complexity and relativ slowness.

The Arithmetic Unit

Multiplication in modern computers is normally performed by repeated addition. This is similar to the 'pencil and paper' method The basic operations of these machines are addition and subtraction. Ludgate's design for a logarithmic multiplier appears to be unique. The potential of this principle is examined, with regard to modern

technology, in the following chapters.

Subroutine methods of division by finding the reciprocal of the divisor have been discussed in several papers [17, 18, 19, 20, 26] . In 1946 von Neumann considered the use of an iterative scheme by this method. This was compared to a hardware design with the resulting recommendation that the hardware divider was built. This was due to cost/speed considerations. For Ludgate to have included a separate mechanical divider in his machine would have added to the cost and complexity considerably.

Flynn [17] describes an identical routine to that proposed by Ludgate for determining the reciprocal, while Wallace [18] and Ferrari [19] describe a more common approach using a Newton-Raphson iteration. Both methods require an initial approximation to the reciprocal. The accuracy of this approximation determines the number of stages of calculation needed to find the result to the required accuracy. Ludgate's method of table look-up to determine the initial approximation has been used in the I.B.M. System/360 Model 91 [21] . The division routine in this I.B.M. machine is discussed, and improvements suggested, by Ahmad. [26] The routine is identical to Ludgate's in theory, although it is applied to bit normalised binary number system.

Control, Input/Output facilities

In Ludgate's machine, these bear a strong resemblance to modern methods although they would have been primitive in most aspects.

His design included two paper-tape readers; formula-tape and number-tape. Thus instructions and data were kept completely separate.

These readers were also designed to produce new punch tape, thereby creating a backing store. In modern machines, technology has enabled these two types of punch to be combined into one unit.

It is not possible from the report to determine which unit (store or mill) was to accept or output data. Thus, no comparison can be made here although both methods have been used since.

His design permitted keyboard control of the machine and of a printer. This is a common facility today. The method mentioned earlier of conditional branching is primitive by today's standards, though it does compare with the conditional branching in which is now considered the first computer, the Harvard Mk 1 (1944).

3.3 CONCLUSION

Judging from the limited evidence provided by Ludgate's 1909 report, it would appear that in theory many features of his design are similar to equivalent aspects of modern computers. On the other hand there is little mention of practical considerations, and Ludgate himself states in his later report [7] that

> "the true calculating machine belongs to a possible rather than actual class."

C H A P T E R F O U R

ELECTRONIC LOGARITHMIC MULTIPLIERS

As noted earlier one unusual aspect of Ludgate's machine is his use of logarithms to effect multiplication. In this section it is intended to describe the development and design of an electronic multiplier based on the same principle as that proposed by Ludgate. The design was considered for construction in Transistor Transistor Logic (T.T.L.)

4.1 CHOICE OF LOGARITHMS

Ludgate wrote his logarithms to base ten, (decimal). This type of logarithm may be written to any base, (see Appendix 'A'). It is important when calculating these logarithms to make the simple index numbers as low as possible. This reduces the complexity and propagation delay time of an electronic multiplier when used as a basis for its design.

By writing the logarithms to base eight or base four it becomes possible to convert a binary number directly into its logarithm. This is done by examining and converting to indexes in three or two bit groups respectively.

With the base eight system a four digit octal word (12-bit binary)

can have its digits converted to simple index numbers in four separate parallel stages. This makes it possible to multiply two twelve bit numbers in four cycles by multiplying by three bit groups. A similar approach but using base four logarithms will require six cycles.

4.2 A MULTIPLIER USING LOGARITHMS TO BASE EIGHT

The logarithms or index numbers written to base eight are contained in tables 6 and 7. A general outline of a multiplier using these logarithms will be considered first.

General Description

A schematic plan of the multiplier is given in figure 10 and a labelled path through part of it in figure 11. The design is based on the multiplication of two twelve bit binary numbers. The design can easily be modified for other word lengths. It is more suited to numbers with bit lengths that are multiples of three as it 'views' three bits of the multiplier in one cycle.

The action of the multiplier can be described in the following sequences:

1. The four octal digits of the multiplicand and the least significant octal digit of the multiplier are converted in five parallel stages to their simple index numbers.
2. The simple index number of the multiplier is added to each of those formed from the multiplicand. Thus four compound index numbers of six bits each are produced.

3. These six bit compound index numbers are converted to six-bit (two digit octal) semi-partial products.

4. The most significant octal digit (3 bits) of the semi-partial products are added to the least significant digit of the next highest semi-partial product. This is similar to the procedure described by Ludgate for his machine.

5. The result of the previous stage may be considered a complete partial product since it is the product of multiplicand and three bits of the multiplier. This partial product is stored in a parallel load shift register.

6. The complete partial product and multiplier are now shifted right three bits. This shift aligns the next significant three bits of the multiplier ready for the next cycle. The shift of the complete partial product is to compensate for the multiplication factor of eight. (This is comparable to the multiplication by ten in the mill of Ludgate's machine).

7. The cycle is now repeated with the next three bits of the multiplier. The complete partial product formed by this cycle is added to the previous one and the sum and multiplier simultaneously shifted right three bits. Four cycles are required giving a final product of twenty four bits.

The connections through a path of the multiplier as shown in figure 11 have been labelled to correspond to the following descriptions.

Storage Register for Multiplier and Multiplicand

The multiplier and multiplicand are held in twelve bit parallel load/out registers, (assuming parallel mode of machine structure). Twelve binary latches are required for the multiplicand and a shift right register for the multiplier. It is advantageous in the next stage that both these registers should be able to supply the complement of the number stored.

Octal Digit to Simple Index, Converter

This logic circuit converts a three bit unit number into its simple index number which has a maximum range of six bits. (The six bits are required to accommodate the index of zero). One important consideration in the design is to produce a circuit with a minimum delay: Figure 12 is such a circuit with a propagation delay of two gates.

Simple Index Adders

Four six bit adders are required to add the simple index numbers. Since the length of the index numbers is only six bits, medium scale integrated (M.S.I.) circuit adders with ripple carry may be used. (Carry anticipate circuits compare with short M.S.I. adders in speed, [23] , this can be verified for a six bit adder using propagation delay data of 'Texas Instruments' components. [22] . The disadvantage of a more complicated circuit outweighs the advantage of an approximate five nanosecond reduction in delay.)

A seven bit number from the adders can only be produced if the index of zero is added to itself, (i.e. 0 times 0.) This gives a compound index result of octal 112. It is fortunate that in the logarithms to base eight there is no way in which a compound index of octal 12 can be produced. Therefore by ignoring the most significant bit of octal 112 and associating semi-partial product zero with compound index number 12, the semi-partial product is reduced to six bits.

Compound Index Number to Semi-Partial Product, Converter

This section of the logic converts the six bit compound index number (K, L, M, N, P, Q, see Fig. 11) to a six bit semi-partial product. (R, S, T, U, V, W) Again a main consideration is to produce a circuit with a minimum propagation delay.

A compromise between circuits with varying numbers of eight, four and three input logic gates has to be made.

The most significant bit of the input can be ignored for the first part of the design. A binary 'one' only occurs here for output zero and two other values. This reduces the Karnaugh Map analysis to five input variables. The Karnaugh Maps for this conversion are included to show the 'awkwardness' of the system (Table 8). The Boolean equations for the outputs, assuming a logic zero for the sixth input, are also included. This most significant sixth input is included in the final logic diagram, figure 13. The circuit takes into account the fan-out (max. 10, [22]) of the logic gates. This design requires 64 gates of which ten are eight input and has a maximum propagation delay time of four gates.

Without any optimisation in the design, a suitable logic circuit requires 69 gates, 31 of them being of the eight input type. It also has a maximum delay of five gates.

Four of these converters are required.

Semi-Partial Product to Complete Partial Product
(and addition of Complete Partial Products)

The operation of this section of the circuit can be described in five steps, in conjunction with figure 10.

1. The three most significant bits of the previous stage outputs added to adjacent least significant bits.

   i.e. outputs from previous logic

   m.s. - RSTUVW - l.s.

   added in following manner

   $R_n + U_{n+1}$, $S_n + V_{n+1}$, $T_n + W_{n+1}$

   where n = nth stage

2. Above result added to previous complete partial product. (Ripple carry adder bank.)
3. Result 'accepted' into parallel in/out shift register of twenty four bits.
4. Result and Multiplier shifted right three bits.
5. Copy into twelve bi-stable latches, which have outputs connected to second bank of adders, enabling sum of partial products to be added on next cycle.

There are several possible designs for this part of the circuit.

Figure 10 contains the simplest which is two banks of ripple carry adders. The carry runs simultaneously in both banks with the lower bank of fifteen adders dictating the maximum delay. With this design a large percentage of the total multiply time (approx. 45%) occurs at this stage. This percentage can be reduced by introducing parallel operation, carry save or carry anticipate techniques. The details of this are included later as they involve a lengthy discussion.

4.3 A MULTIPLIER USING LOGARITHMS TO BASE FOUR

The principle of this design is exactly the same as the base eight multiplier. The difference is the number of bits of the multiplier examined in one cycle. Two bits are examined and thus six cycles are required for a twelve bit multiplier.

Tables 9 and 10 contain the base four logarithms. The Karnaugh map analysis is easier in both logarithm converters, (L and L'). A complete logic diagram is shown in figure 14 and a detail of one path through the circuit in figure 15.

4.4 MULTIPLICATION OF NEGATIVE NUMBERS

The logarithmic multipliers just described can be thought to operate as a 'table look-up' system. This can be appreciated by considering them as 'black-boxes', where the variables are entered and the product appears after some delay. These 'black-boxes' have been designed to operate on numbers with a positive binary integer format. They cannot operate on one's or two's complement or any other format. This can easily be proved by taking some simple

examples.

To incorporate negative number multiplication it becomes necessary to use sign magnitude representation. This requires an exclusive - OR operation between the sign bits of the multiplicand and multiplier. The result of this is the sign of the product. This short operation can run parallel with the multiplication.

CHAPTER FIVE

TIMING AND PRACTICAL CONSIDERATIONS OF THE MULTIPLIERS

This chapter documents comparison of total delays through the two types of logarithmic multiplier described in the last chapter. Before this, suitable logic components are stated and it is on these that the delay times are based. Finally, methods of reducing the overall delay are discussed.

5.1 SUITABLE COMPONENTS

The data for the components mentioned below is based on texas 7400 logic series [22] .

Multiplicand register - three four-bit bit bistable latches supplying output and complement output (type SN7475).

Multiplier register - no parallel in/out shift register with complemented output manufactured by Texas. Delay times based on a suitable register constructed out of discrete logic gates and J/K flip-flops.

Logic gates of converters (L and L') - the propagation delays of these converters are based on Schottky T.T.L.

Simple index adders - As mentioned earlier, M.S.I. T.T.L. adders may be used. For the base four method: type SN7483 and base eight: type SN7483 (4 bit) and SN7482 (2 bit).

Partial product adders - Suitable combinations of four bit and two bit binary adders (ripple carry) and carry-save-adders (SN7483, SN7482, SN74H183 respectively.)

Partial product shift registers - fifteen bits of this 24-bit register have to be parallel in/out, right shift type (two SN74198). The remainder should be of serial in parallel out, right shift type (two SN7164).

Partial product temporary storage register - latches as in multiplicand register (three SN7475).

5.2 COMPARISON OF BASE EIGHT AND BASE FOUR MULTIPLIERS

The main difference in the complexity of the two designs occurs in the logarithm converters. To convert a twelve bit word to simple index numbers requires 60 logic gates in the base eight method and only 35 in the base four. To convert back to the semi-partial product requires 256 gates in base eight and 78 in base four design. This makes the base four design cheaper and easier to construct.

The propagation delays for both circuits are shown in figure 16. These are based on the maximum delays stated by Texas, and applied to the longest path through each multiplier. The timings assume that the multiplier and multiplicand registers have been previously

loaded.

When determining the total time for the multiplication the following points need to be considered.

1. The copy of the complete partial product into the latches after each cycle can be done parallel with the beginning of the next cycle.
2. The final cycle does not contain shift or latch copy operations.

The complete multiplication times are:

BASE 8 : 420 x 4 - 90 = 1590 ns.

BASE 4 : 360 x 6 - 60 = 2100 ns.

5.3 METHODS OF DECREASING THE TOTAL PROPAGATION DELAY

There are five methods of reducing the delay of the logarithmic circuits. They involve:

1. Parallel operation of sections of the circuit.
2. Carry anticipate
3. Carry store
4. Skip cycle if multiplier bits are zero
5. combination of 1 and 4 with 2 or 3

These techniques can be applied to both circuits but will only be considered for the base eight multiplier.

1. Parallel operation of sections of the circuit.

This requires that the first bank of adders in the partial product add stage should be of the carry-save type.

The principle of the parallel operation is to divide the circuit

into two. This is done by storing the carry and sum from the carry-save-adders in latches and operating the logic before them in parallel with that after. The logic diagram can be seen in Figure 17 and the timing diagram in figure 18 contains the total delay of this method. There is only an approximate reduction of 130 ns with this method.

There are two main disadvantages of this circuit. Firstly, the control signals become more complex with two separate shift controls required. Secondly, the cost increases due to the use of carry-save-adders and more latches.

2. Carry Anticipate

This is discussed by Lewin [23] . The principle is to examine all the inputs to the adders and simultaneously generate the carries for each stage. These carrie s are then applied to the appropriate adder stage to give the final sum. The delay for such a circuit is eight gates for the sum outputs and seven for the final stage carry output. The carry anticipate is, as normal, considered here in groups of five adders.

A bank of carry-save-adders has to be used again and with Schottky T.T.L. the delay time for the partial product stage is:

Carry anticipate: 110 ns + 18 ns for C.S. Adders

M.S.I. ripple carry: 188 ns.

The time for a base eight multiplication reduces from 1590 ns to 1350 ns but involves a large increase in circuit complexity and expense.

Figure 19 gives the circuit for carry anticipate.

3. Carry Store

In this system the adders are all replaced by carry-save-adders. The carry from each stage is stored and added back two adders to the right after the complete partial product has been shifted right three bits. After the final stage the carries are added to the appropriate sums in a fourteen stage ripple carry adder. This principle is shown in figure 20. This again requires extra control signals and logic. The delay is now reduced to 1420 ns.

4. Skip cycle if multiplier bits zero

When the multiplier bits are zero it is a waste of time to allow for the propagation delay of one complete cycle. An OR-gate connected to the outputs of the multiplier bits to be used in the next cycle will indicate if the shift control is to be doubled. This would skip over the three bit groups of the multiplier when zero.

The reduction in delay will obviously depend on the numbers multiplied.

5. Combination of previous methods

The above methods can be combined and the delay time can be reduced to approximately 1250 ns. The complexity and cost become considerable when this is done and it is worth considering more conventional methods and comparing them with the logarithmic multiplier first.

Appendix D contains an account of a software simulation process for checking the proposed designs of logarithmic multipliers.

CHAPTER SIX

COMPARISON OF LOGARITHMIC MULTIPLIERS WITH CONVENTIONAL MULTIPLIERS

The logarithmic multipliers described, differ in structure from other designs only in the way the partial product is derived. The methods described in the previous chapter for summing the complete partial products can be applied to the multipliers to be described here. Therefore in the comparison it is only necessary to consider the stages producing the complete partial product.

The logarithmic multipliers are suitable for sign magnitude multiplication and should therefore be compared only with other types using this negative number representation.

6.1 A TWO BIT SHIFT MULTIPLIER USING A CONVENTIONAL APPROACH

The multiplier described here can be compared with the base four design. It operates on addition alone and examines two bits of the multiplier in any one cycle. The appropriate action on examining two bits of the multiplier is given in table 11. A logic diagram of a multiplier using this principle is given in figure 21. The circuit is self-explanatory and it suffices to say that the shifting of the multiplicand to effect multiplication by two is accomplished

by interposing gates between the multiplicand and adder.

The propagation delay for this circuit can be reduced further by using carry anticipate or skip cycle techniques. This is not necessary for a comparison providing it is made with the base four multiplier in figure 14 as far as the first bank of adders.

The following are delay times through each multiplier to the complete partial product stage.

Delay through base four multiplier.

$$= L + \left[3 \text{ bit Adder } A_1 \rightarrow C_3\right] + L^1 + 2 \times \left[4 \text{ bit adder Co} \rightarrow C_4\right] + \left[4 \text{ bit adder Co} \rightarrow \Sigma_3\right]$$

$$= 10 + 60 + 15 + 96 + 60$$

$$= 241 \text{ ns}$$

Delay through conventional 2 bit shift multiplier

$$= 3 \text{ gates} + 2 \times \left[4 \text{ bit adder Co} \rightarrow C_4\right] + \left[4 \text{ bit adder Co} \rightarrow \Sigma_3\right]$$

$$= 15 + 48 \times 2 + 60 \text{ ns}$$

$$= 171 \text{ ns}$$

From these results and by inspecting the relevant logic diagrams, the conventional approach is preferable for cost and speed.

6.2 A THREE BIT SHIFT MULTIPLIER USING A CONVENTIONAL APPROACH

This is a similar design to the two bit multiplier described above. Three bits of the multiplier are examined in one cycle. The appropriate actions as regard to the multiplier bits are given in table 12 and the logic diagram in figure 22.

A comparison with the same considerations as in section 6.1 can be made (as far as partial product stage).

Delay through base eight multiplier

$$= L + \left[6 \text{ bit adder}\right] + L' + 2 \times \left[4 \text{ bit adder } Co \rightarrow C_4\right] + \left[4 \text{ bit adder } Co \rightarrow \Sigma_3\right]$$

$$= 10 + \left[48 + 42\right] + 20 + 2 \times 48 + 60$$

$$= 276 \text{ ns.}$$

Delay through conventional three bit multiplier

$$= 4 \text{ gates} + 1 \text{ C.S.A.} + 3 \times \left[4 \text{ bit adder } Co \rightarrow C_4\right] + 1 \text{ C.S.A.}$$

$$= 20 + 18 + 144 + 18$$

$$= 200 \text{ ns.}$$

Again there is an appreciable reduction in propagation delay with the conventional approach.

6.3 CONCLUSION

It is obvious from the two preceding chapters that the conventional approach to sign magnitude mutliplication is faster, less complicated and considerably cheaper than the proposed logarithmic method.

S U M M A R Y

No more information can be extracted from Ludgate's 1909 report. As this is the only information available about the machine [1], our appraisal of his design must be limited to the ingenuity of the theory. The detail of Ludgate's drawings, which largely determines the machine's feasibility, cannot be determined. Therefore, we must agree with a review in 1909 of Ludgate's report by Keys [25] that,

> "Until more detail as to the proposed construction and drawings are available it is not possible to form any opinion as to the practicability or utility of the machine as a whole."

It can clearly be seen by reading this report, that the base four and base eight multipliers have no advantages to offer over conventional designs. Multipliers written to bases higher than eight (i.e. 16, 32 ...) could be designed, but with difficulty due to their complexity. They would also still suffer the same disadvantages as the base four and eight designs.

One further application of this type of logarithm is in the design of a binary coded decimal multiplier. This is discussed further in appendix B. It appears from a superficial design consideration, that a logarithmic b.c.d. multiplier may have advantages over existing types. This aspect deserves further consideration.

This logarithmic method of multiplication may be applied to software table look-up multipliers. Here the restrictions on using logarithms to a low base number to reduce the circuit complexity are removed, although other restrictions will be imposed. It may be possible to apply logarithms to a b.c.d. table look-up multiplier. These aspects are also worthy of further consideration.

Other aspects of Ludgate's engine have since been used in later machines. Any further consideration of these aspects should be with these later computers where more detail is available.

Other than the logarithms and their applications, the analytical engine requires no further study unless new information on the machine is found.

R E F E R E N C E S

1. Randell, B. "Ludgate's Analytical Machine of 1909" University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 15. (1971)

2. Babbage, C. "Passages From The Life of A Philosopher." London, Longman 1864

3. Bowden, B.V. "Faster Than Thought". Pitman (1953)

4. Chase, G.C. "History of Mechanical Computing Machinery." Proceedings of the A.C.M. National Meeting, Pittsburgh 2 - 3 May 1952.

5. Hartree, D.R. "Calculating Instruments and Machines." Cambridge, 1950.

6. Encyclopaedia Britannica 13th ED. "Calculating Machines"

7. Ludgate, P.E. "Automatic Calculating Machines". Napier Tercentenary Celebration : Handbook of the Exhibition. Edinburgh : Royal Society of Edinburgh

8. Babbage, H.P. "On the Mechanical Arrangements of the Analytical Engine of the late Charles Babbage." Report of the Brit. Assoc. for the Advancement of Science (1888).

9. Morrison, P. and Morrison, E. "Charles Babbage and his Calculating Engines : Selected Writings by Charles Babbage and Others", Dover Publications (1961)

10. Wilkes, M.V. "Automatic Digital Computers". London, Metheuen 1956.

11. Ludgate, P.E. "On a Proposed Analytical Machine" Scientific Proceedings, Royal Dublin Society, April 1909

12. Burks, A.W., Goldstine, H.H., and von Neumann, J. "Preliminary discussion of the logical design of an electronic computing instrument." Report to U.S. Army Ordnance Department, reprint in Bell and Newell [15] .

13. Boys, C.V. "A New Analytical Engine". Nature 81,2070 1st July 1909.

14. Kilburn, T., Edwards, D.B.G., Lanigan, M. and Sumner, F., "One-level Storage System". IRE Trans EC-11 pp 223-235, April 1962.

15. Bell, C.G. and Newell, A. "Computer Structures: Readings and Examples". McGraw-Hill (1971)

16. Buchholz, W. "Planning a Computer System." McGraw-Hill (1962)

17. Flynn, M.J. "Very High Speed Computing Systems". IEEE Proc. Vol. 54 Dec' 66 pp 1901-1909

18. Wallace, C.S. "A Suggestion for a Fast Multiplier." IEEE Trans. on Electronic Computers, Vol. EC-13, No. 1 Feb. 1964.

19. Ferrari, D. "A Division Method Using a Parallel Multiplier." IEEE Trans. on Electronic Computers, Vol. EC-16, No. 2, April 1967 pp. 224-226.

20. Robertson, J.E. "A New Class of Digital Division Methods." IRE Trans. on Electronic Computers Vol. EC-17 Sept 1968 pp 218-222

21. Anderson, S.F., Earle, J., Goldschmidt, R.E. and Powers, D.M. "The IBM System/360 - Model 91 : Floating Point Execution Unit". IBM Journal of R. and D., Vol. 11, No. 1, Jan 1967 pp 48-53.

22. Texas Instruments Semiconductor Components - Digital Integrated Circuits, July 1971, Issue 2.

23. Lewin, D. "Theory and Design of Digital Computers." Nelson (1972).

24. Hollingdale, S.H. and Tootill G.C. "Electronic Computers" Pelican (1970).

25. Boys, C.V. "A New Analytical Engine." Nature 81,2070, 1st July 1909, pp 14-15.

26. Ahmad, M. "Iterative Schemes for High Speed Division." The Computer Journal, Vol. 15, No. 4, Nov. 1972.

27. "D.A. 70 User Manual", Automation Division, Poole. (software simulation of logic circuits).

A P P E N D I X 'A'

INDEX NUMBER SYSTEMS

Definition of A UNIT :

A positive integer in the range $[o, x-1]$, where x is the base to which the number system is written.

An Algorithm to derive:- index numbers of any number system.

1. Decide which base to write logarithms in.
2. Determine Prime numbers of the units of that number system.
3. Associate the simple index number 0 with unit 1.
4. Associate next unused index number with next highest prime number.
5. Calculate index numbers of all possible products of units so far produced (including units produced in this stage.)
6. Go to 7 if any product has two different index numbers associated with it or an index number has two separate products. Else go to 8 .
7. Erase last step 5 and associate next unused index number with last prime number. Go to 5 .
8. If any prime numbers left then go to Step 4 .
9. Associate lowest unused index number with zero.
10. Calculate index numbers of all possible products of units x zero.
11. If any index number associated with two products then go to 12, else go to 13 .

12. Erase lasts steps 9 and 10 and associate next unused index number with zero. Go to 10 .

13. END.

An example of the above algorithm is:

	UNIT	INDEX NUMBER
1	Logs. to be written to Base 10.	
2	Prime numbers are 1, 2, 3, 5, 7.	
3	Simple index of 1 is 0	
	1	0
4	Simple index of 2 is 1	
	2	1
5	Index numbers of products that are powers of 2	
	4	2
	8	3
	16	4
	32	5
	64	6
	128 not required since $> 9 \times 9 = 81$	
4	3 has the next highest index number	
	3	7
5	Calculate the index numbers of all possible products of units defined so far.	
	9	14
	27	21
	81	28
	6	8
	12	9
	18	15
	24	10
	36	16
	48	11
	54	22
	72	17

UNIT	INDEX NUMBER

Unit 4 already has an index number

4 Let 5 have the next highest unused index number

5	12

5 Calculate index of Powers of 5

25	24

5 Calculate products of 5 and previous unit variables

10	13
15	8

7 Error here due to 8 already being an index number. ∴ the index of 5 is changed. Same problem occurs at index 13, 18, 14, 20. ∴ 5 has index 23

5	23

5 Calculate index numbers that are products of previously defined units

25	46
10	24
20	25
40	26
15	30
45	37
30	31

By letting the index of 7 be 27, 29, or 32 the same problem as above occurs
Let 7 have index 33

7	33
14	34
28	35
56	36
21	40
42	41
63	47
35	56

Associate unit zero with highest available index. This, in this case eventually becomes 50.

Theory behind the derivation of the index numbers.

The above algorithm determines the lowest set of index numbers for any particular base. The principle of the algorithm is to associate the lower index numbers with the prime numbers or units that occur more often in forming all possible products of units.

An exception to the rule is the unit zero. It is convenient to give this the largest index number. Then all compound index numbers above a certain value may be regarded as zero.

The unit 'one', has to have a simple index of zero in every case (i.e. $\log z + \log 1 = \log z$)

Where two prime numbers are divisible into the same number of products, then their index numbers can be interchanged without increasing the index range. One example of this is the interchange of five and seven in Ludgate's logarithms to base ten.

A P P E N D I X B

A BINARY CODED DECIMAL HARDWARE MULTIPLIER

The intention of this section is to briefly consider and describe how the previous theory of logarithms can be applied to a binary coded decimal multiplier.

The normal approaches to binary coded decimal (B.C.D.) multiplication involve software table look-up methods. The programming approach to multiplication is inexpensive to include in a B.C.D. machine, although it does require storage locations for its 'working' and tables. Another disadvantage is that it is slow, although with this type of computer high speed is not normally necessary.

Conventional methods of hardware multiplication are not viable with a B.C.D. number format. This is due to it being necessary to examine the multiplier in four bits and also because the multiplicand cannot be shifted, then added, to allow for multiplication. Hardware conversion from B.C.D. numbers to ordinary binary and back again involves repeated multiplication and division, respectively, by ten [24] . This is a costly and lengthy process and makes conversion of the numbers to a form suitable for an ordinary multiplier, impractical.

A logarithmic multiplier has the advantage in that it examines the multiplicand in groups of bits, and not as a whole word as in conventional designs. The remainder of the logarithmic design can

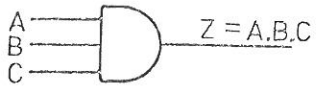
be based on the same principle as the base four and base eight designs described in the body of the report. The final stage will have to use B.C.D. adders to add the semi-partial products. This is a disadvantage due to B.C.D. adders being complicated and expensive. [23] .

It appears from this short account that a logarithmic B.C.D. multiplier may be feasible, if a fast multiplier is required.

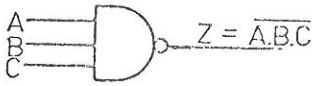
APPENDIX C

LOGIC SYMBOLS

1. AND GATE

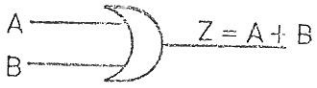


2. NAND GATE

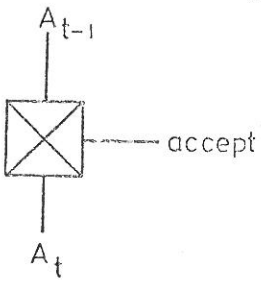


3. INVERTOR

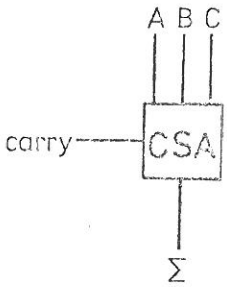
4. OR GATE



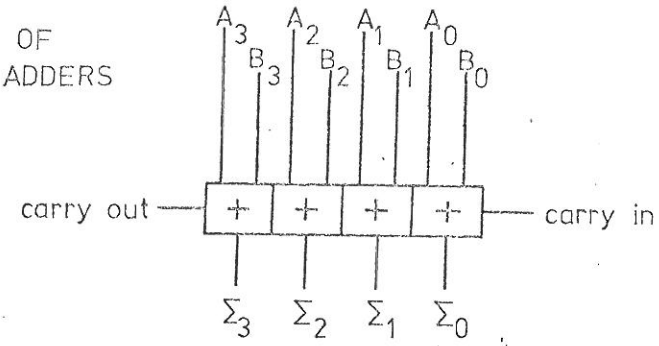
5. BISTABLE LATCH (STORE)



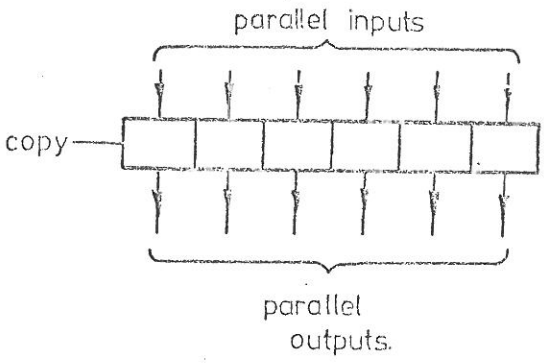
6. CARRY SAVE ADDER



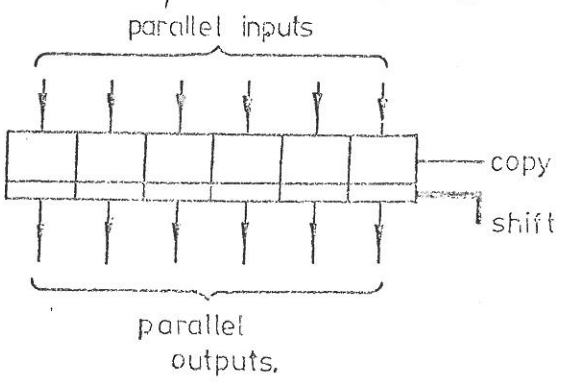
7. BANK OF FULL ADDERS



8. PARALLEL IN/OUT REGISTER



9. PARALLEL IN/OUT SHIFT REGISTER



10. BINARY NUMBER TO SIMPLE INDEX CONVERTER.

11. COMPOUND INDEX NUMBER TO SEMI-PARTIAL PRODUCT CONVERTER.

* { inputs to upper semi-circle,
outputs from lower semi-circle.

APPENDIX D

PROGRAM SIMULATION OF LOGARITHMIC MULTIPLIERS

This appendix contains an account of the procedure used to check the logic design of the logarithmic converter stages (L and L^{-1}), using the DA70 simulation program [27] .

Only one path through the multiplier need be simulated. The logic diagram is converted or coded to a form that is acceptable by the program; coding is explained in the 'DA70 USER MANUAL'. The simulation instructions are then written. Figure 23 contains 'suitable instructions' for checking the accuracy of the logarithmic design.

Firstly all inputs permanently low are set to zero (all unconnected pins automatically set to logic 1). The multiplier and multiplicand registers are defined as one register, (INP). Taking the base eight design as an example, a six bit register is declared with its three least significant bits acting as the inputs for the multiplier and the others for the multiplicand.

The register, INP, is set to zero and then repeatedly incremented until it has covered its complete range. Thus all possible inputs are simulated. After each increment the input pulses are allowed to ripple through the simulated logic. The product enters another register, (OUT). The contents of the two registers are then printed before the next increment. Using the printed information the design can be checked.

It is also possible to use the DA70 simulation suite to produce accurate timing diagrams. This is explained in the User Manual and only a brief account will be made here.

A modified version of the 'simulation commands' in fig. 23 is sufficient and this is given on page 54. The commands can be explained as follows;

DLAY - enables the propagation and edge times (from logic zero to one and one to zero) to be defined by the programmer. The actual delays are based on Texas components [22] .

PRUD - is used to set the propagation times of the adders. The alphabetic characters following each # PRUD refer to the pin connections to the adders. The delay is set in the connections which is effectively the same as being in the adder.

DREG and #SREG - define the input and output registers and set them to zero initially.

The remainder of the program is basically the same as that described on page 51. The following commands are inserted

MREG #SIDE and #RATE 4 - these produce the timing diagram. The format of the diagram is

+ equivalent to logic one

/ " " indeterminate state

- " " logic zero

The RATE 4 causes a symbol (+/or -) to be printed every fourth time-slot or four nanoseconds. (The average over the four nanoseconds is printed).

Examples of the timing diagrams for the base eight multiplier design are given on pages 55-60. Note that the corresponding final values of the input (INP) and output (OUT) registers are given after each timing diagram.

Due to an omission in the simulation commands fourteen of the sixty four timing diagrams produced by this program are inaccurate. For these

fourteen, 32 nanoseconds have to be added to the total indicated delay times.* These are not included in the examples, except for 5.

The maximum indicated delay occurs at 5 x 2

= 23 timeslots + 32ns error

= 92 + 32

= 124ns.

This test does not give the exact maximum delay. The delay is dependent on the initial conditions of the logic (from previous multiplication). Since the program takes into account the difference in propagation times from logic one to zero and zero to one, the delay is dependent upon the initial conditions of the logic, (i.e. the previous multiplication.)

The maximum delay could be found by further programming, but this could not be over 135ns. This compares favourably with the estimated delay which does not account for edge times

estimated delay = L + (6 bit adder Co $\rightarrow \Sigma_5$) + L^1

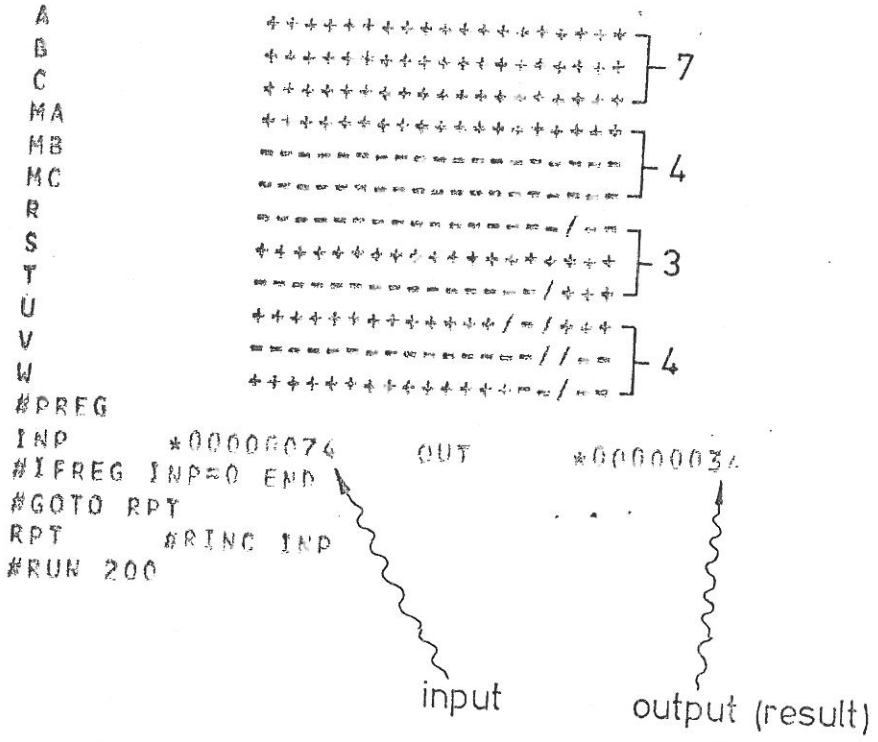
= 10 + (48 + 42) + 20

= 120ns.

*The error in the simulation is the omission of a # PRUD command to set the delay of the carry between the four stage adder and the two stage adder to 27, 35. (i.e. 27ns delay for logic 0→1 and 35ns for logic 1→0.) A nominal 4, 3 is automatically assumed, thus an allowance of 32ns is necessary on multiplications where a carry occurs between the adders.

```
DA70        DATE  10/04/73
#SIML SIML 1
#PROG
#DLAY 1NAND 5 5 1 1
#DLAY 2NAND 5 5 1 1
#DLAY 3NAND 5 5 1 1
#DLAY 4NAND 5 5 1 1
#DLAY 8NAND 5 5 1 1
#PRUD I=35 30 H=29 28 G=19
#PRUD F=14 15 MI=35 30 MH=2
#PRUD MG=19 10 MF=14 15
****

#PRUD ME=35 30 MD=29 28
#PRUD D=29 28 E=35 30
#PRUD Q=5 5 P=11 7 N=21 25
#PRUD M=26 20 L=5 5 K=11 7
#EXT 0V=0
#DREG INP=A B C MA MB MC
#SREG INP=0
#DREG OUT=R S T U V W
#STRT
#MREG INP OUT
#SIDE
#RATE 4
RPT #RINC INP
#RUN 200
#PREG
#IFREG INP=0 END
#GOTO RPT
END #END
```

```
#IFREG INP=0 END
#GOTO RPT
RPT      #RINC INP
#RUN 200

  11801
```




```
#PREG
INP      *00000074      OUT      *00000034
#IFREG INP=0 END
#GOTO RPT
RPT      #RINC INP
#RUN 200
```

input

output (result)

Example of timing diagram
produced by simulation program.

$7 \times 4 = 34$

octal

```
A                =-=-=-=-=-=-=-=-
B                =-=-=-=-=-=-=-=-
C                +-+-+-+-+-+-+-+-
MA               +-+-+-+-+-+-+-+-
MB               +-+-+-+-+-+-+-+-
MC               =-=-=-=-=-=-=-=-
R                =-=-=-=-=-=-=-=-
S                =-=-=-=-=-=-=-=-
T                =-=-=-=-=-=-=-=-
U                +-+-+-+-+-+-+-+-
V                =-=-=-=-=-=-=-//
W                +-+-+-+-+-+-+-=-
¥PREG
INP      *00000016      OUT      *00000004
¥IFREG INP=0 END
¥GOTO RPT
RPT      #RINC INP
¥RUN 200
```

2801

```
A                =-=-=-=-=-=-=-=-=-
B                =-=-=-=-=-=-=-=-=-
C                +-+-+-+-+-+-+-+-+-
MA               +-+-+-+-+-+-+-+-+-
MB               +-+-+-+-+-+-+-+-+-
MC               +-+-+-+-+-+-+-+-+-
R                =-=-=-=-=-=-=-=-=-
S                =-=-=-=-=-=-=-=-=-
T                =-=-=-=-=-=-=-=-/-
U                +-+-+-+-+-+-+-+-/-/
V                +-+-+-+-+-+-+-+-+-
W                =-=-=-=-=-=-=-=-/-
¥PREG
INP      *00000017      OUT      *00000007
¥IFREG INP=0 END
¥GOTO RPT
RPT      #RINC INP
¥RUN 200
```

3001

```
A                =-=-=-=-=-=-=-=-=-=-
B                +-+-+-+-+-+-+-+-+-+-
C                =-=-=-=-=-=-=-=-=-=-
MA               =-=-=-=-=-=-=-=-=-=-
MB               =-=-=-=-=-=-=-=-=-=-
MC               =-=-=-=-=-=-=-=-=-=-
R                =-=-=-=-=-=-=-=-/-+-//
S                =-=-=-=-=-=-=-=-+-+-//
T                =-=-=-=-=-=-=-=-=-=-
U                +-+-+-+-+-+-+-+-/-=-=-
V                +-+-+-+-+-+-+-+-/-=-=-
W                +-+-+-+-+-+-+-+-/-=-/-//
¥PREG
INP      *00000020      OUT      *00000003
¥IFREG INP=0 END
¥GOTO RPT
RPT      #RINC INP
```

```
                 -------------/---
 |               ---------------//
 |               -------------/+//
 |               -------------////
 PREG
 INP      *00000031      OUT       *00000003
 IFREG INP=0 END
 GOTO RPT
 RPT      #RINC INP
 RUN 200

    5001

A                -----------------
B                +++++++++++++++++
C                +++++++++++++++++
MA               -----------------
MB               +++++++++++++++++
MC               -----------------
R                -----------------
S                ----------------/
T                -----------------
U                ---------------++
V                +++++++++++++++++
W                +++++++++++++++/-
 PREG
 INP      *00000032      OUT       *00000006
 IFREG INP=0 END
 GOTO RPT
 RPT      #RINC INP
 RUN 200

    5201

A                ------------------
B                ++++++++++++++++++
C                ++++++++++++++++++
MA               ------------------
MB               ++++++++++++++++++
MC               ++++++++++++++++++
R                ------------------
S                ------------------
T                ----------------/+
U                ++++++++++++++----
V                ++++++++++++++/---
W                ----------------/+
 PREG
 INP      *00000033      OUT       *00000011
 IFREG INP=0 END
 GOTO RPT
 RPT      #RINC INP
 RUN 200

    5401

A                --------------------
B                ++++++++++++++++++++
                 ++++++++++++++++++++
MA               ++++++++++++++++++++
MB               --------------------
```

```
IC
I
I
'
I
I
I
IPREG
INP      *00000060      OUT      *00000000
IIFREG INP=0 END
IGOTO RPT
IPT      #RINC INP
IRUN 200
```

9601

```
A
B
C
MA
MB
MC
R
S
T
U
V
W
IPREG
INP      *00000061      OUT      *00000000
IIFREG INP=0 END
IGOTO RPT
IPT      #RINC INP
IRUN 200
```

9801

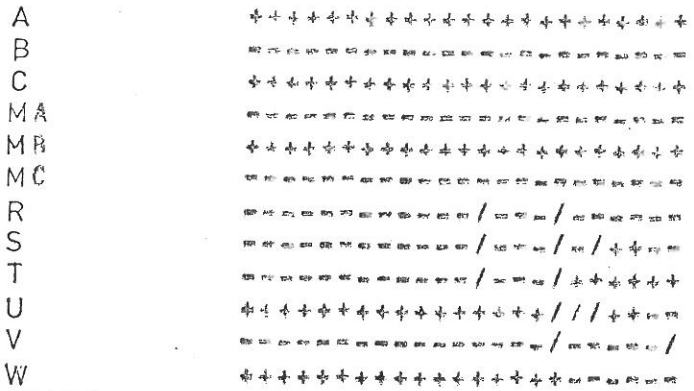
```
A
B
C
MA
MB
MC
R
S
T
U
V
W
IPREG
INP      *00000042      OUT      *00000016
IIFREG INP=0 END
IGOTO RPT
IPT      #RINC INP
IRUN 200
```

10001

```
A
B
```

```
#RUN 200

  11201

A                +++++++++++++++++++
B                +++++++++++++++++++
C                +++++++++++++++++++
MA               ====================
MB               ====================
MC               +++++++++++++++++++
R                ==========//========
S                ==========/++//
T                ==========//========
U                ==========////
V                ==========/+++
W                ==========////
#PREG
INP      *00000071      OUT      *00000007
#IFREG INP=0 END
#GOTO RPT
RPT      #RINC INP
#RUN 200

  11401

A                +++++++++++++++++++
B                +++++++++++++++++++
C                +++++++++++++++++++
MA               ====================
MB               +++++++++++++++++++
MC               ====================
R                ==================//
S                ====================
T                ==================//
U                +++++++++++++++++++
V                +++++++++++++++++++
W                +++++++++++++++++//
#PREG
INP      *00000072      OUT      *00000010
#IFREG INP=0 END
#GOTO RPT
RPT      #RINC INP
#RUN 200

  11601

A                +++++++++++++++++++
B                +++++++++++++++++++
C                +++++++++++++++++++
MA               ====================
MB               +++++++++++++++++++
MC               +++++++++++++++++++
R                ====================
S                ==================//
T                +++++++++++++/=====/++//
U                +++++++++++++/=====/+++
V                +++++++++++++/=====/++//
W                ==========/+++
#PREG
INP      *00000073      OUT      *00000028
```

8201




```
PREG
NP      *00000052      OUT      *00000012
IFREG INP=0 END
GOTO RPT
PT      #RINC INP
RUN 200
```

8401

Maximum delay occurs when multiplying 5 x 2,

= 23 timeslots x 4 + 32 ns error

= 92 - 32 ns

= 124 ns

N.B. This diagram is incorrect, see *, page 53

TABLES
nos. 1–12

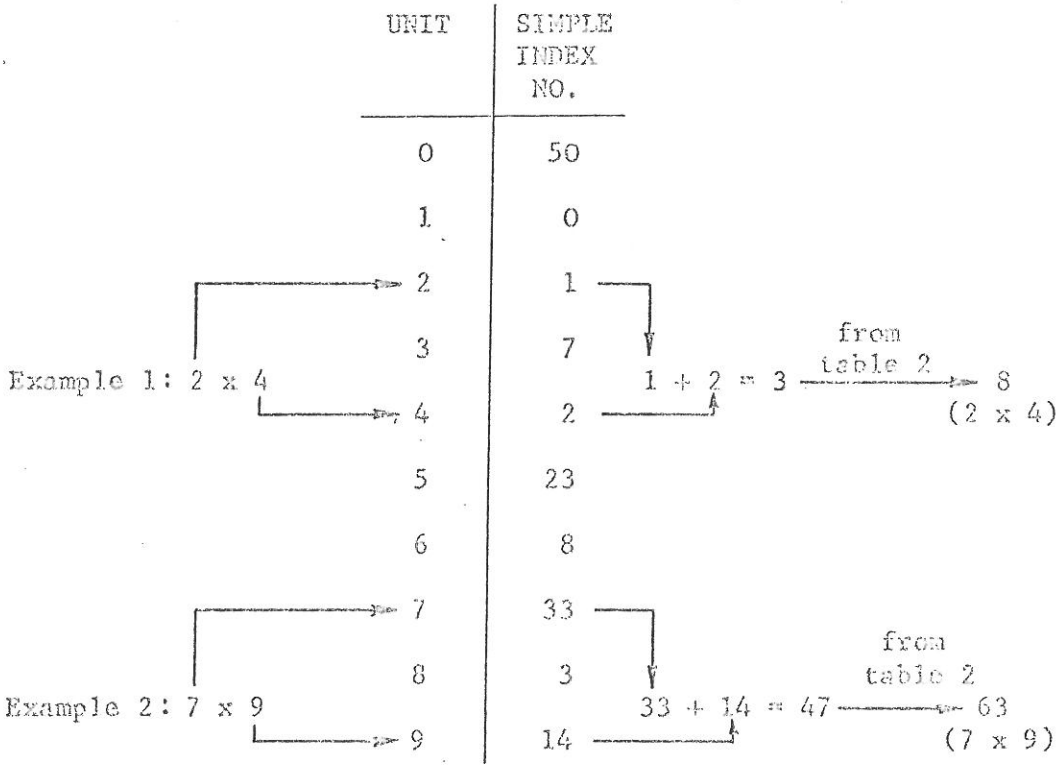
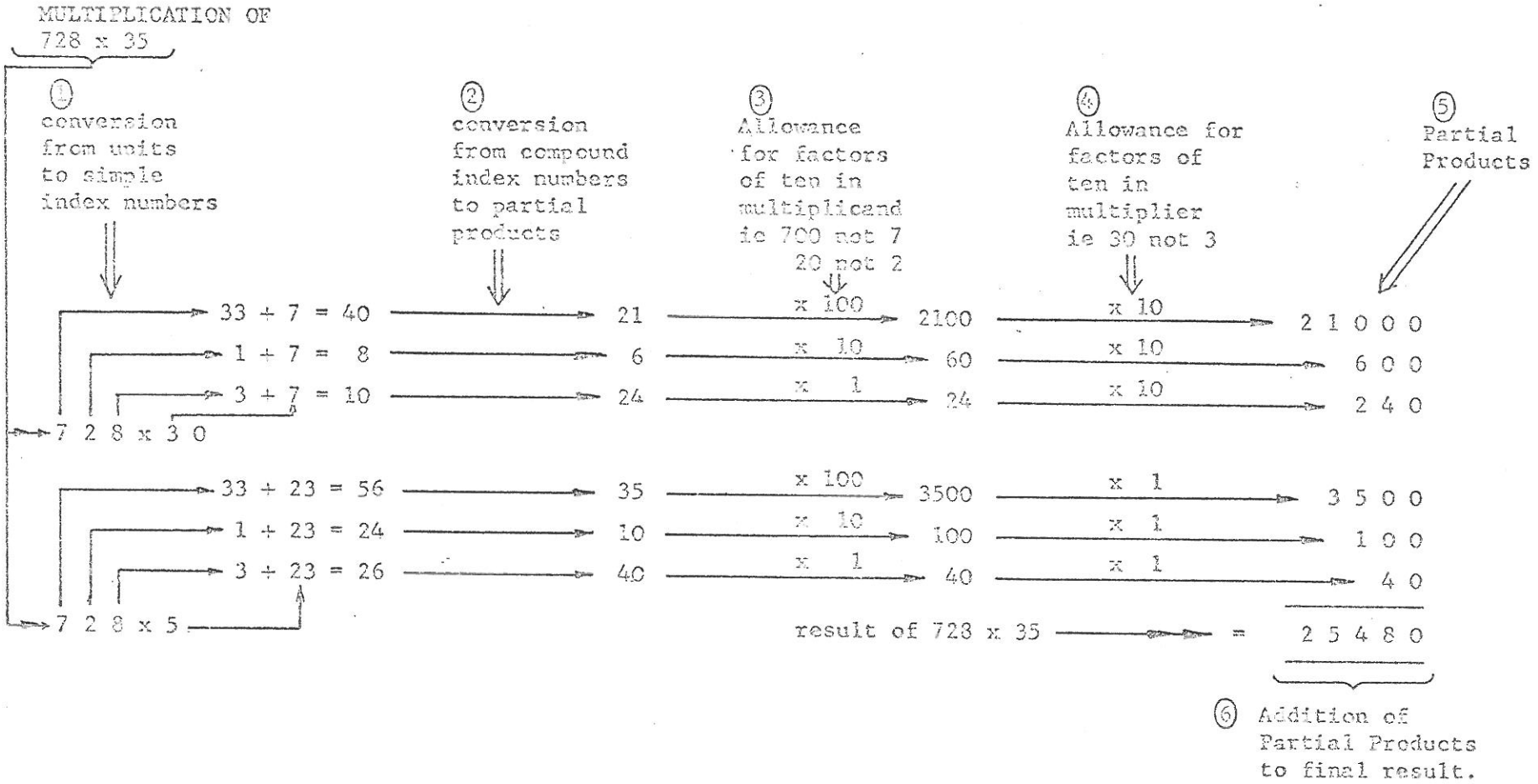


TABLE 1 - LUDGATE'S SIMPLE INDEX NUMBERS

TABLE 2 - LUDGATE'S COMPOUND INDEX NUMBERS

COMPOUND INDEX NO.	PARTIAL PRODUCT	COMPOUND INDEX NO.	PARTIAL PRODUCT
0	1	33	7
1	2	34	14
2	4	35	28
3	8	36	56
4	16	37	45
5	32	38	-
6	64	39	-
7	3	40	21
8	6	41	42
9	12	42	-
10	24	43	-
11	48	44	-
12	-	45	-
13	-	46	25
14	9	47	63
15	18	48	-
16	36	49	-
17	72	50	0
18	-	51	0
19	-	52	0
20	-	53	0
21	27	54	-
22	54	55	-
23	5	56	35
24	10	57	0
25	20	58	0
26	40	59	-
27	-	60	-
28	81	61	-
29	-	62	-
30	15	63	-
31	30	64	0
32	-	65	-
		66	49

T A B L E 2



T A B L E 3 - MULTIPLICATION EXAMPLE

[728 x 35]

Decimal Digit Represented by rod	Protrusion Distance of Rod [1 - 10 units]
0	1
1	10
2	9
3	6
4	8
5	3
6	5
7	2
8	7
9	4

T A B L E 4

TABLE 5 - POSSIBLE INSTRUCTIONS FOR LUDGATE'S MACHINE

Store Address (96 locations in each store, 7 bit address)

1. Rs .....store address (rotate store to position s)

Store Format

1. Bo:= So .....outer store variable to buffer
2. B_I:= S_I .....inner store variable to buffer
3. B_o := S_o ; B_I := S_I .....both store variables to buffers.

Data automatically re-written before accessing new location,
(i.e. shuttles replaced before store rotated)

Operation code

1. A := 0 .....clear accumulation
2. A := A + B_Q x 1 .....add buffer contents to accumulator
3. A := A - B_Q x 1 .....subtract buffer contents from accumulator
4. A := A + Bo x B_I.....add product of buffers to accumulator
5. A := A + Bo ÷ B_I.....*add division of outer by inner buffers to accumulator
6. B_Q^1 :=A .....store accumulator in buffer(s).
7. SK .....formula-paper conditional skip.

Input/Output

8. B_{Q1} :=K .....keyboard data to buffer(s)
9. B_Q :=P .....number-paper data to buffer(s)
10. Pu :=A .....output accumulator via Punch.
11. Pr :=A .....output accumulator via Printer.

12. Io :=Bo .....temporary store index register from outer buffer.
13. A:=A + B_I x Io .....add product of index register to inner buffer
14. T:=B_Q .....temporary store buffer
15. B_Q:=T .....load buffer from temporary store.

where Q = I for inner store
O for outer store

Q^1 = I or O or both

* assumed $\frac{Bo}{B_I}$

instructions 12 - 15 allow:

(a) direct multiplication of any two variables in different stores.

(b) direct transfer of data from one store location to another.

TABLE 5

UNIT TO BASE 8	SIMPLE INDEX NO.
0	45
1	0
2	1
3	6
4	2
5	17
6	7
7	22

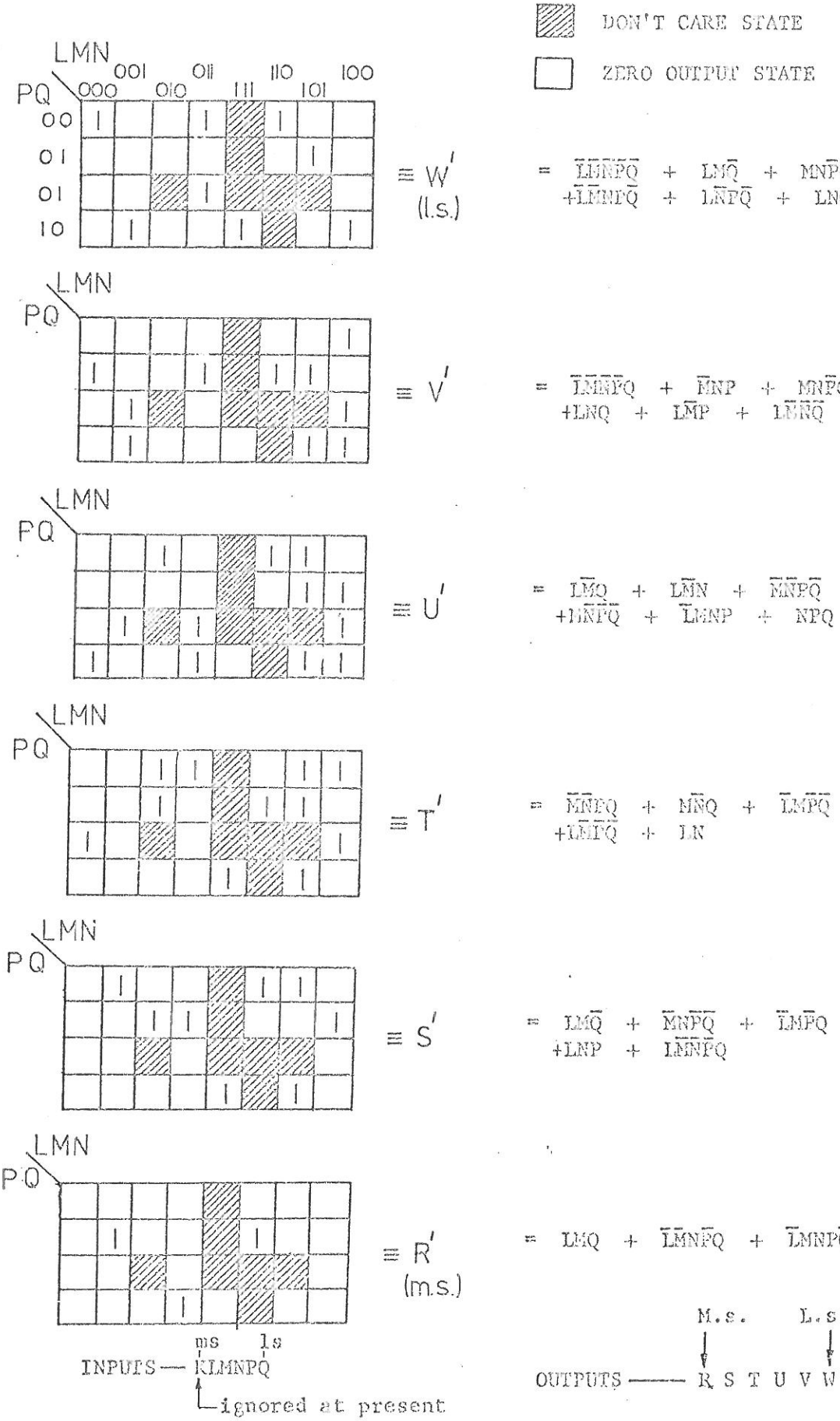
TABLE 6 - LOGARITHMS TO BASE 8

Compound Index Number	Semi-Partial Product	Compound Index Number	Semi-Partial Product
0	1	27	-
1	2	30	25
2	4	31	52
3	10	32	-
4	20	33	-
5	40	34	-
6	3	35	-
7	6	36	31
10	14	37	-
11	30	40	-
12	0	41	43
13	-	42	-
14	11	43	-
15	22	44	61
16	44		
17	5	45	0
20	12	46	0
21	24	47	0
22	7	53	0
23	16	54	0
24	34	64	0
25	17	67	0
26	36	112	0

T A B L E 7

COMPOUND INDEX NUMBERS OF

BASE EIGHT LOGARITHMS



T A B L E 8 - KARNAUGH MAPS FOR L' CONVERTER
(Assuming logic zero for most significant input)

UNIT TO BASE 4			SIMPLE INDEX NO.		
Binary	Decimal	Base 4	Base 4	Decimal	Binary
0 0	0	0	1 3	7	1 1 1
0 1	1	1	0	0	0 0 0
1 0	2	2	1	1	0 0 1
1 1	3	3	3	3	0 1 1
P Q					A B C

TABLE 9 - BASE FOUR, SIMPLE INDEX NUMBERS

COMPOUND INDEX NUMBER			PARTIAL PRODUCT		
Binary [R S T U]	Decimal	Base 4	Base 4	Decimal	Binary [V W X Y]
0 0 0 0	0	0	1	1	0 0 0 1
0 0 0 1	1	1	2	2	0 0 1 0
0 0 1 0	2	2	10	4	0 1 0 0
0 0 1 1	3	3	3	3	0 0 1 1
0 1 0 0	4	10	12	6	0 1 1 0
0 1 1 0	6	12	21	9	1 0 0 1
0 1 1 1	7	13	0	0	0 0 0 0
1 0 0 0	8	20	0	0	0 0 0 0
1 0 1 0	10	22	0	0	0 0 0 0
1 1 1 0	14	3	0	0	0 0 0 0

TABLE 10 - BASE FOUR, INDEX TO PARTIAL PRODUCT

$V = \overline{R}.S.T.\overline{U}$

$W = S.\overline{T} \vee \overline{R}.\overline{S}.T.\overline{U}$

$X = \overline{R}.\overline{S}.U \vee S.\overline{T}$

$Y = \overline{R}.S.T.\overline{U} \vee \overline{R}.\overline{S}.T.U \vee \overline{R}.\overline{S}.\overline{T}.\overline{U}$

Boolean equations connecting outputs of L'converter to its inputs.

MULTIPLIER BITS	ACTION
00	Do nothing
01	Add multiplicand
10	Add multiplicand shifted left one place (i.e. x 2)
11	Add multiplicand, and add multiplicand shifted left one place.

TABLE 11 - TWO BIT MULTIPLIER ACTIONS

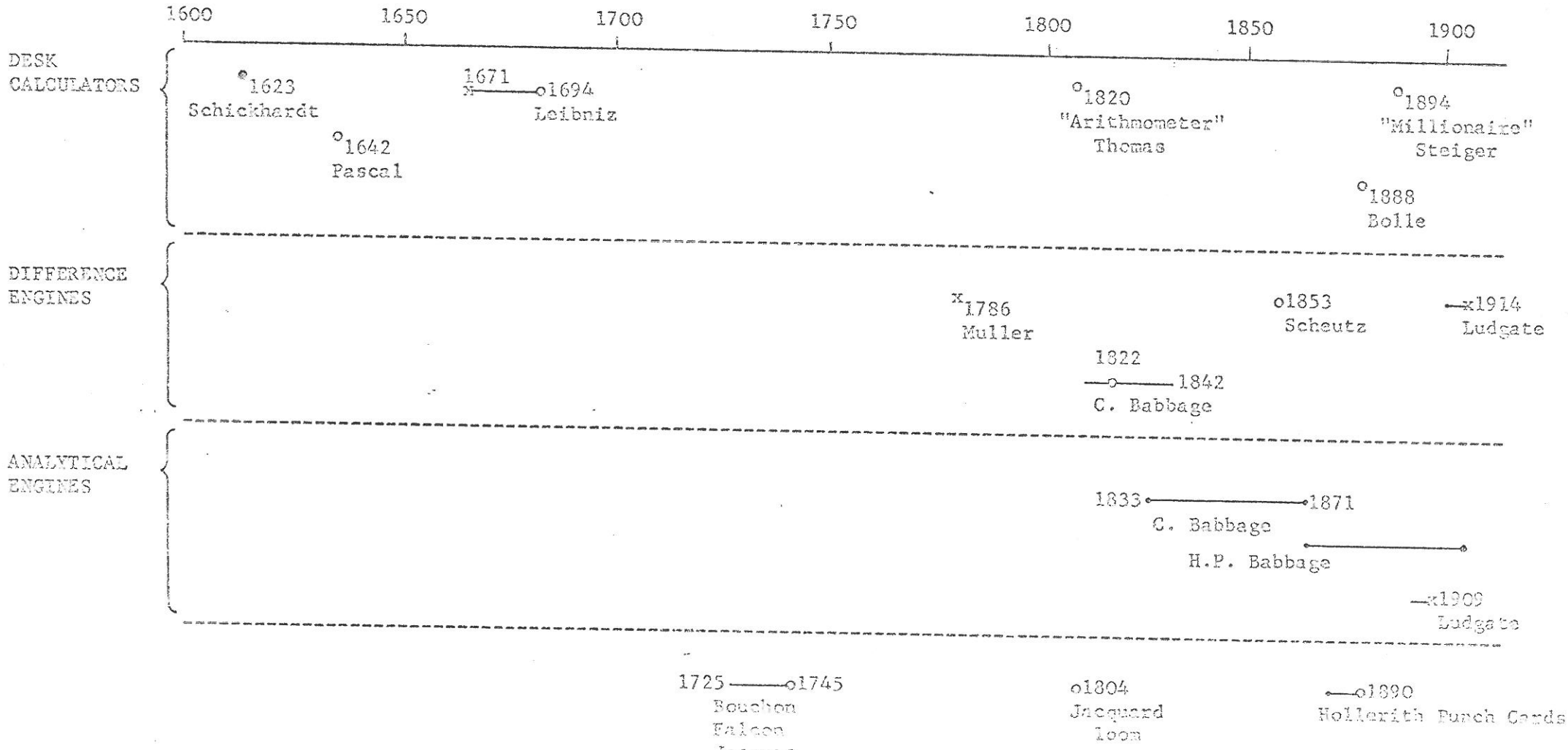
MULTIPLIER BITS	ACTION
000	Do nothing
001	Add MPCD
010	Add MPCD shifted left once
011	Add MPCD shifted left once, Add MPCD.
100	Add MPCD shifted left twice
101	Add MPCD shifted left twice, Add MPCD.
110	Add MPCD shifted left twice, Add MPCD shifted left once
111	Add MPCD shifted left twice, Add MPCD shifted left once and Add MPCD

MPCD = multiplicand

TABLE 12 - THREE BIT MULTIPLIER ACTIONS

FIGURES
nos. 1–23

FIG. 1. — T I M E C H A R T : Major developments in Calculating Machines up to 1920



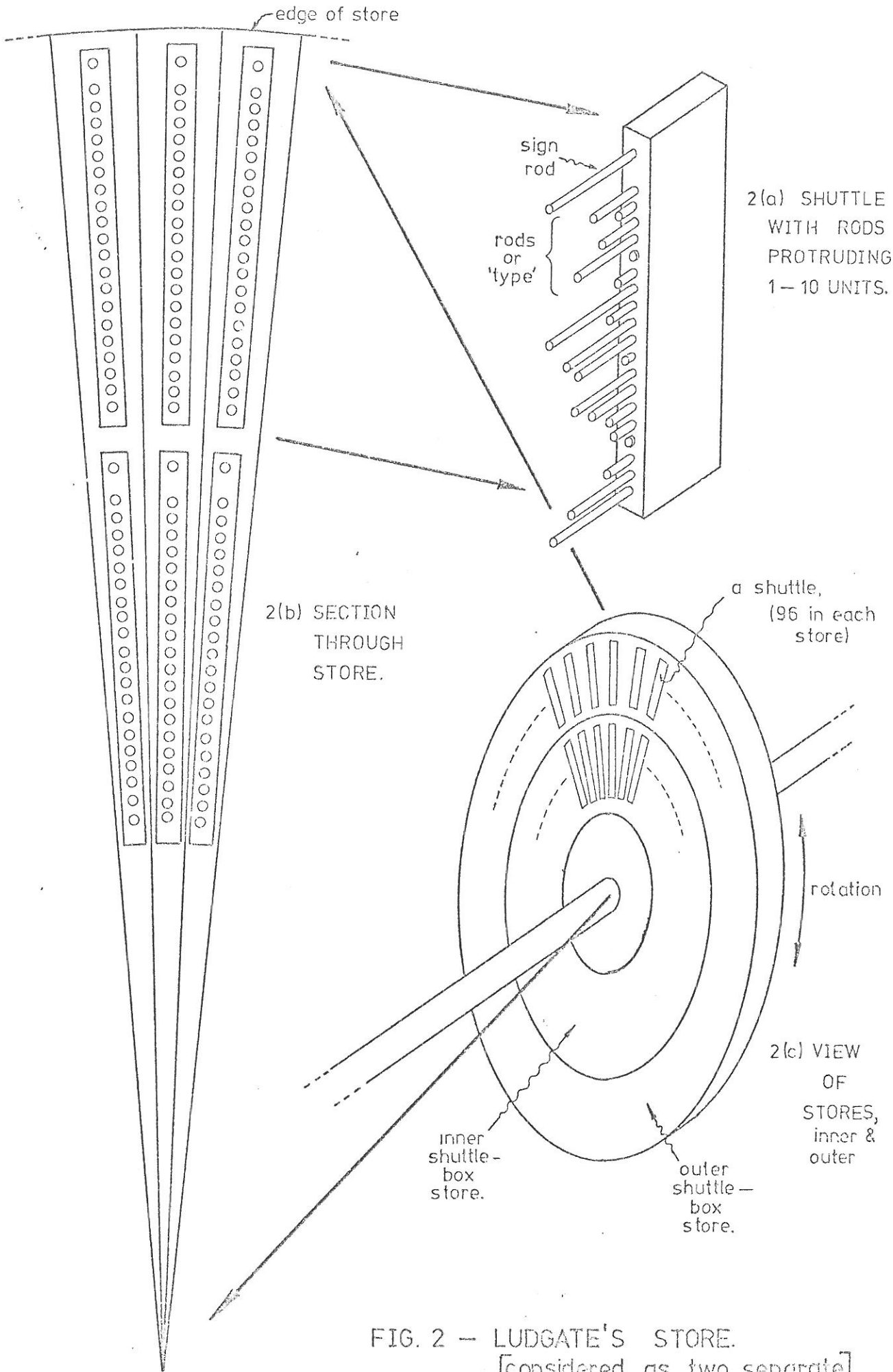


FIG. 2 – LUDGATE'S STORE.
[considered as two separate units – inner & outer stores.]

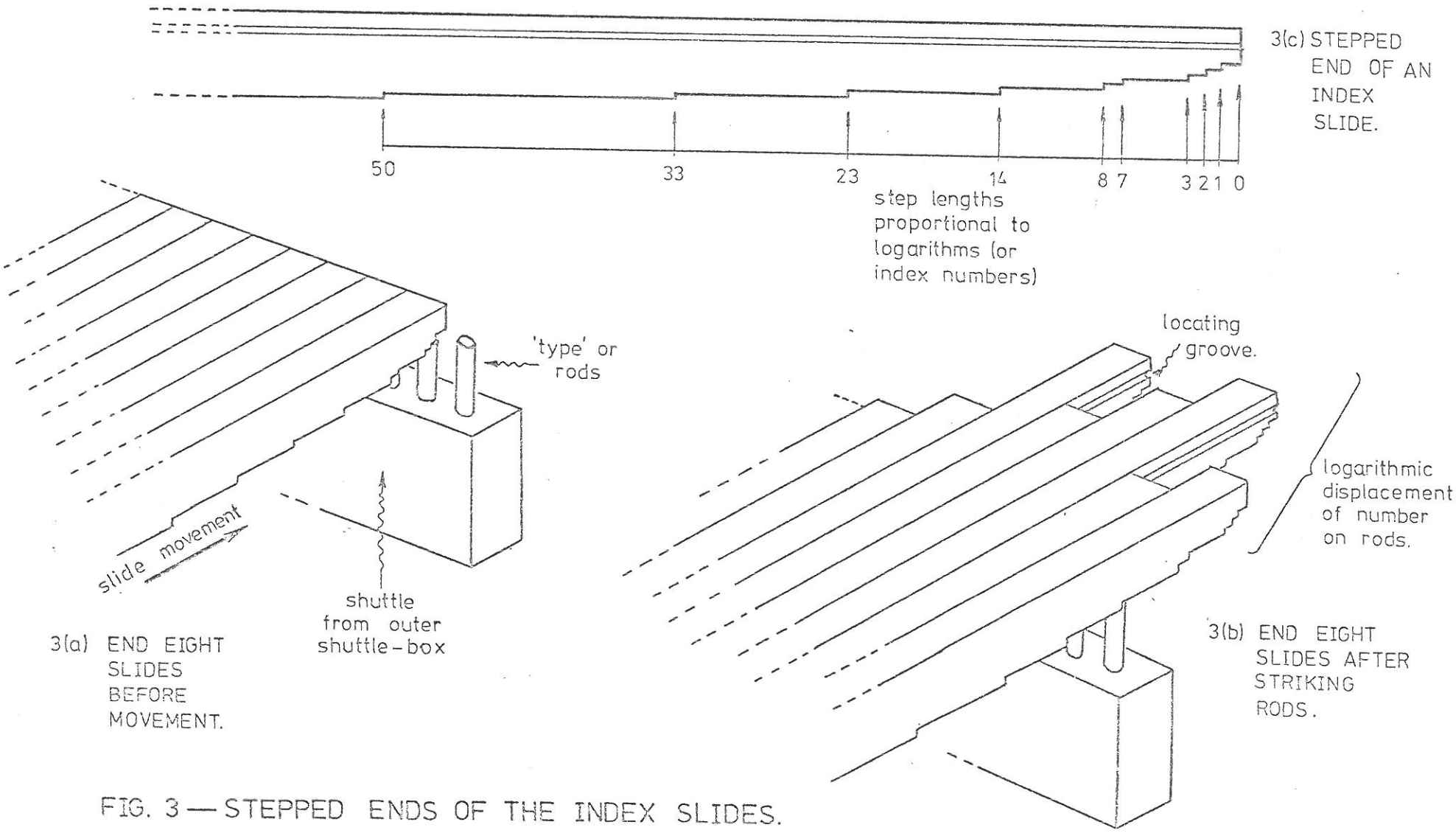


FIG. 3 — STEPPED ENDS OF THE INDEX SLIDES.

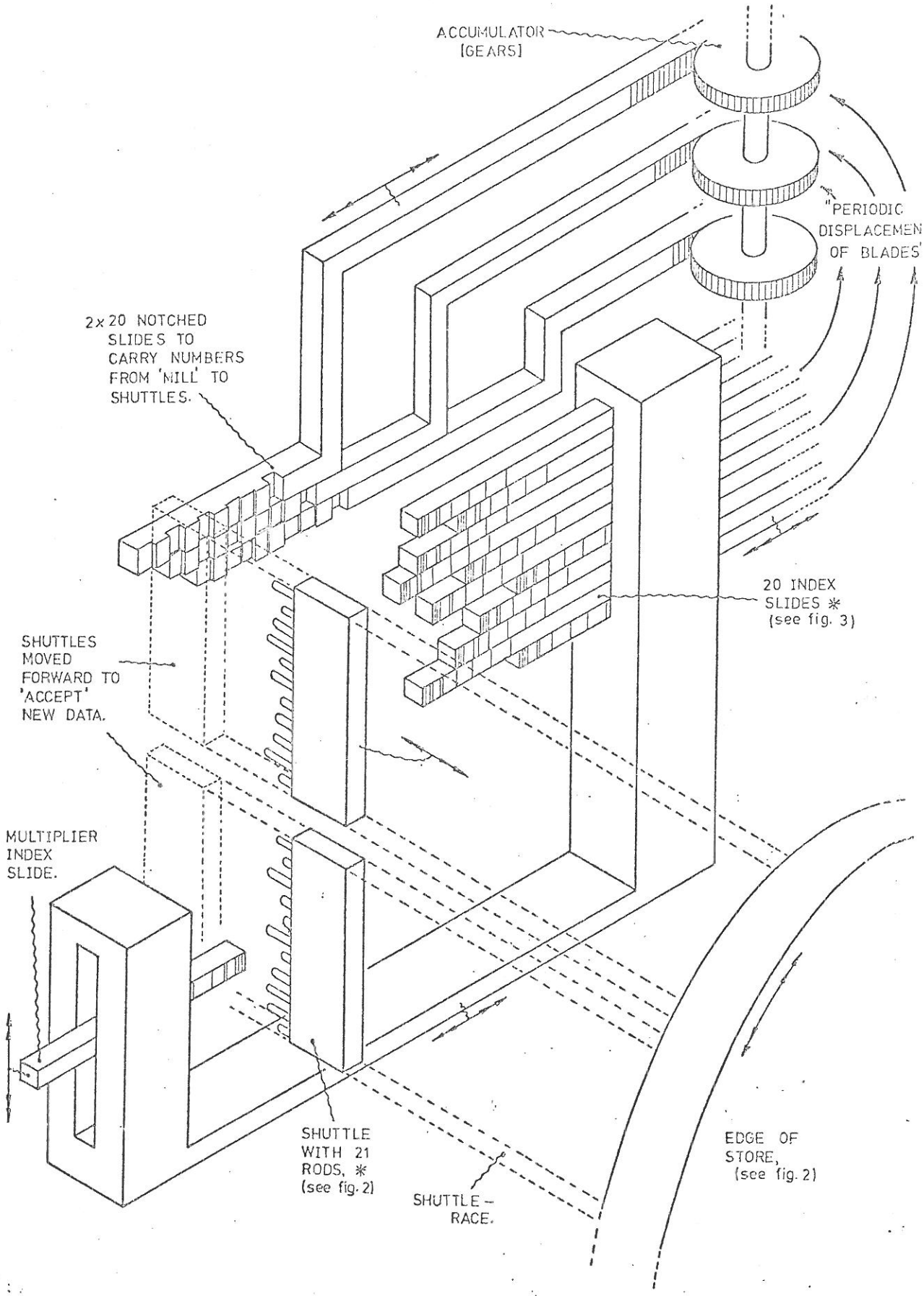


FIG. 4 — GENERAL SCHEMATIC OF THE ARITHMETIC UNIT.

* NUMBER OF SLIDES AND RODS REDUCED IN DIAGRAM FOR CLARITY.

DIRECTION OF MOVEMENT OF PARTS OF MACHINE

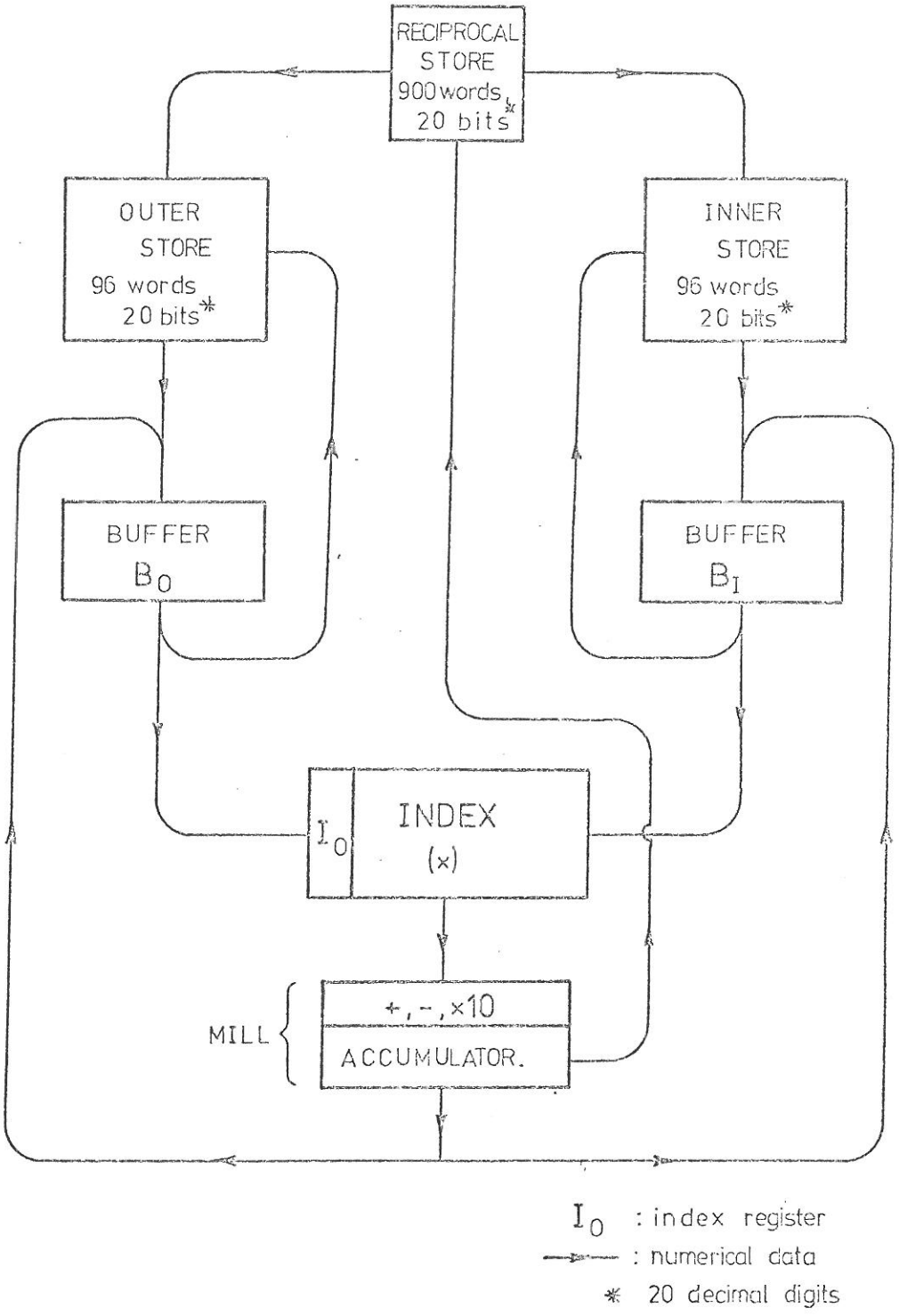
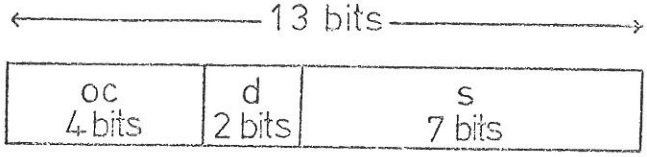


FIG-5.—BASIC BLOCK DIAGRAM
OF LUDGATE'S MACHINE.

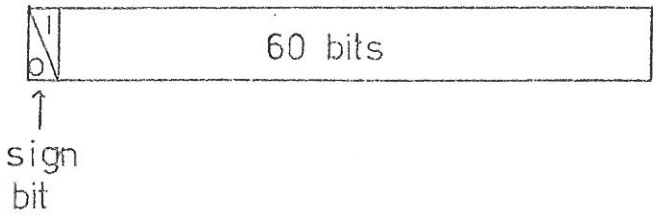


INSTRUCTION WORD FORMAT
(derived from table 5.)

oc - operation code

d - store format inner, outer or both

s - store address



DATA WORD FORMAT

FIG. 6

Explanation of Figure - 7

Both operands taken from same address, one from each store, and placed in buffers (B_O and B_I) and accumulator cleared. Multiplication proceeds and result forms in the accumulator. The branches in the flow chart describe the six possible ways of storing the result. The following numbered comments correspond to the diagram.

1. Result written over operand in inner store

   i.e. operand in inner store lost

   operand in outer store retained.

2. Result written over operand in outer store

   i.e. operand in outer store lost

   operand in inner store retained.

3. Result written over both operands (outer and inner)

   i.e. both operands lost

4. Rotate store, then 5, 6 or 7 as below.

   i.e. both operands retained

5. Store result in outer store.

6. Store result in inner store.

7. Store result in inner and outer stores (having same address).

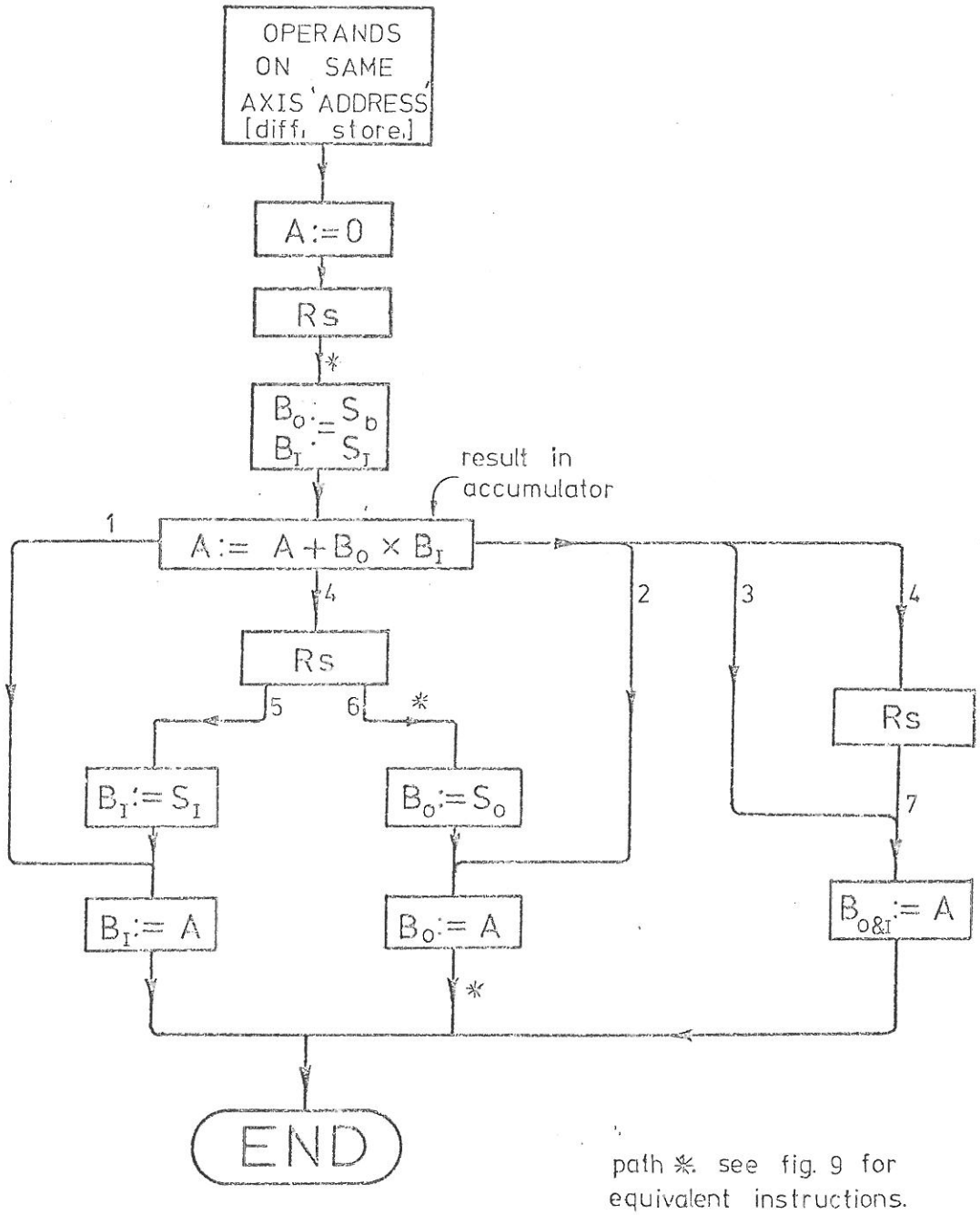


FIG. 7 — MULTIPLICATION FLOW CHART.

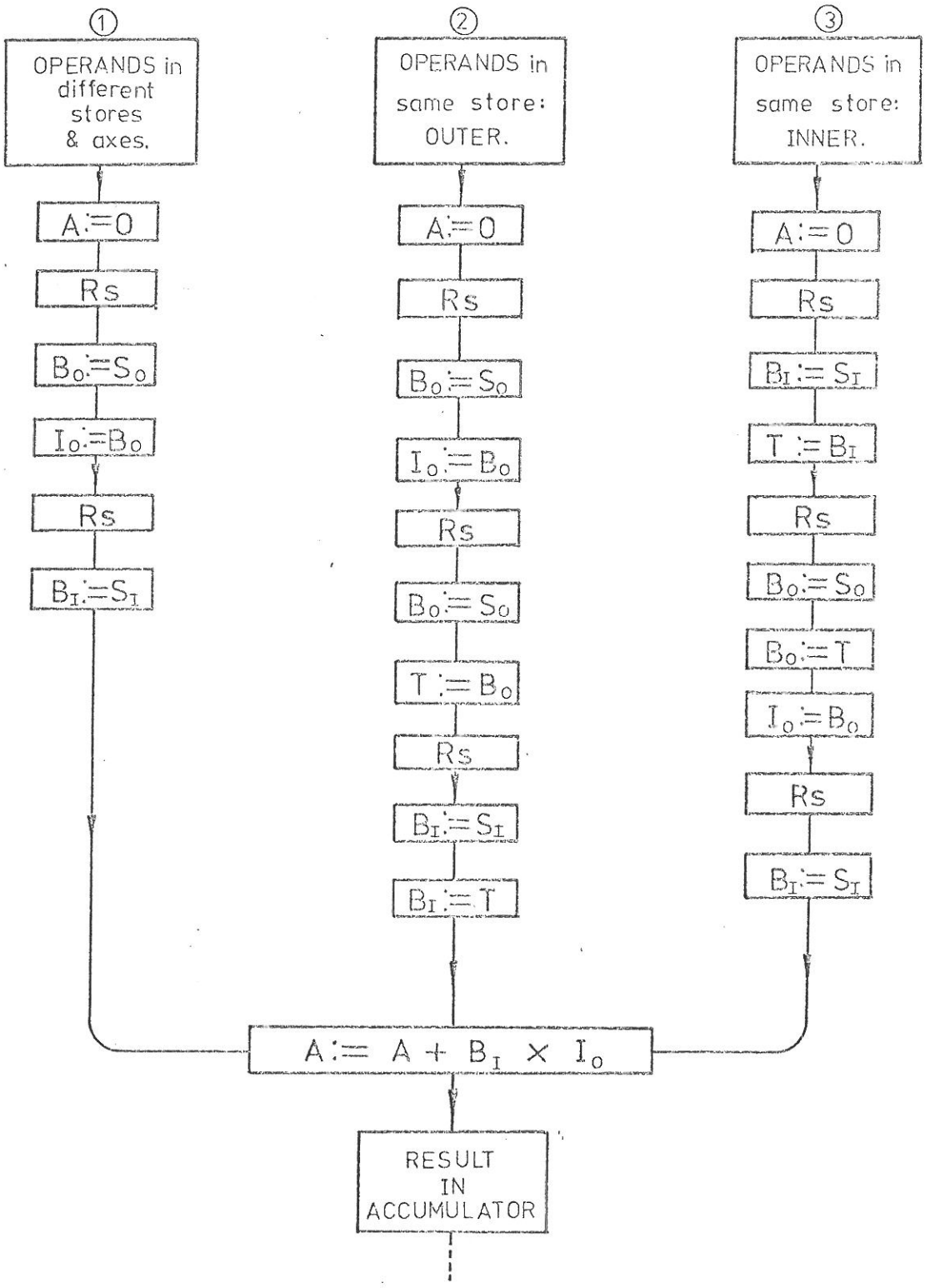
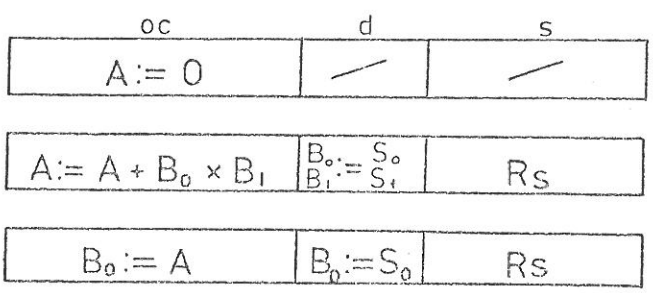


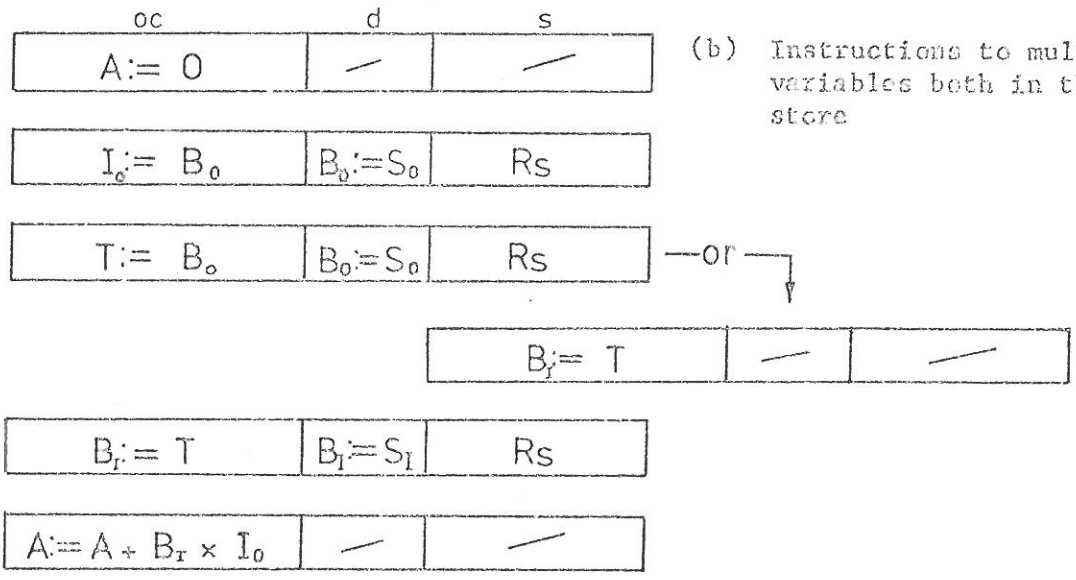
FIG. 8 — MULTIPLICATION FLOW CHART.

(a)



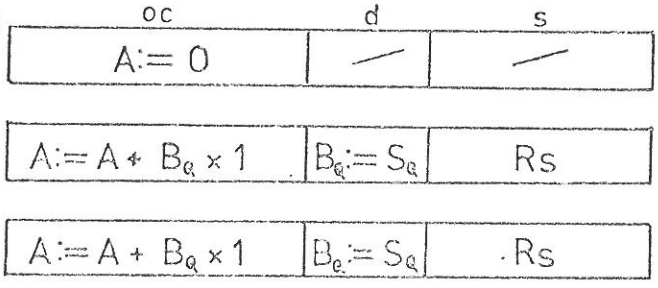
(a) Instructions to multiply two variables on same axis, result stored in another location in outer store [e.g. Fig. 9 Path*]

(b)



(b) Instructions to multiply two variables both in the outer store

(c)



(c) Instructions to add two variables to the accumulator, result in the accumulator

FIG. 9 - EXAMPLES OF INSTRUCTION WORDS

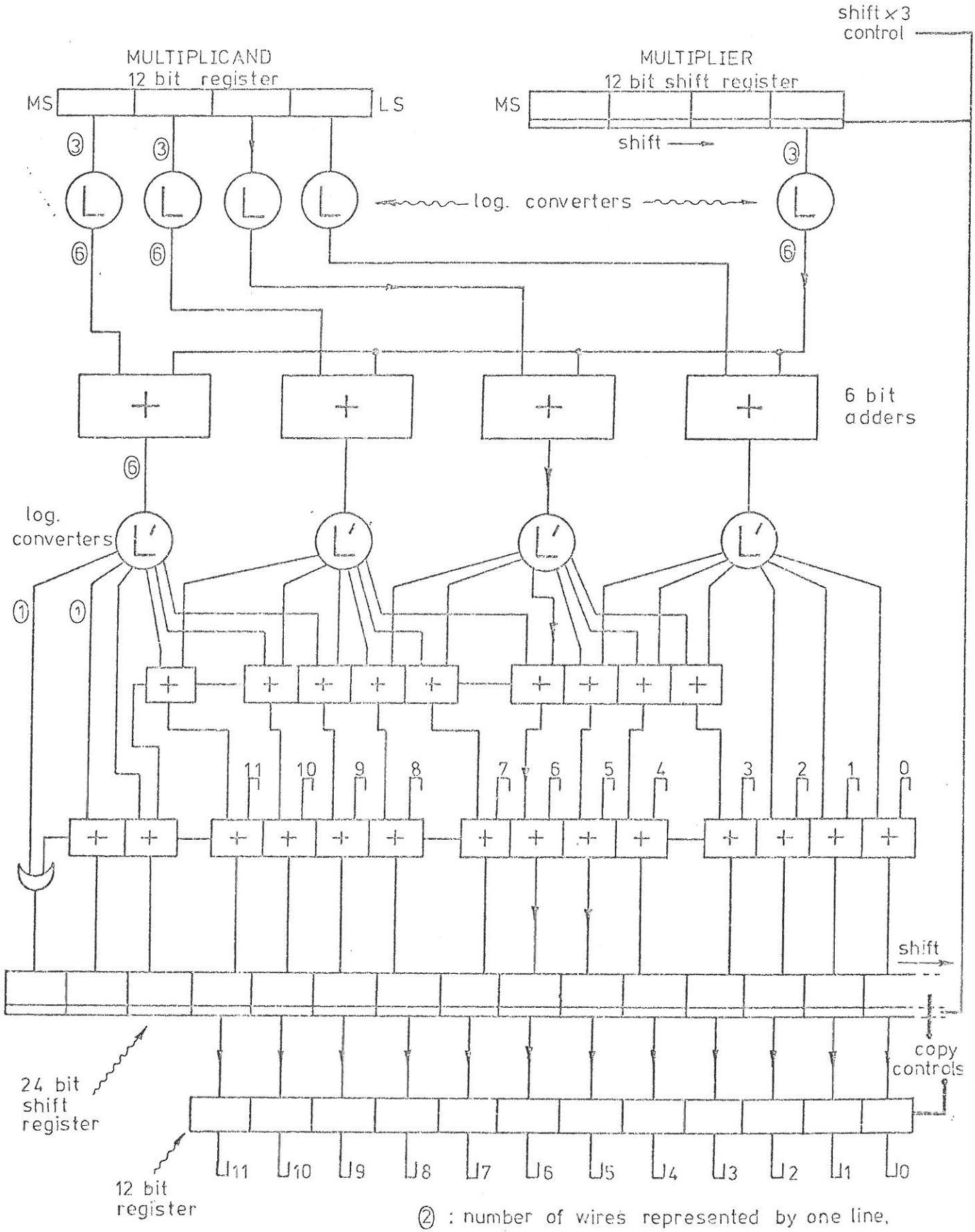


FIG. 10 — LOGIC DIAGRAM OF
BASE EIGHT MULTIPLIER.

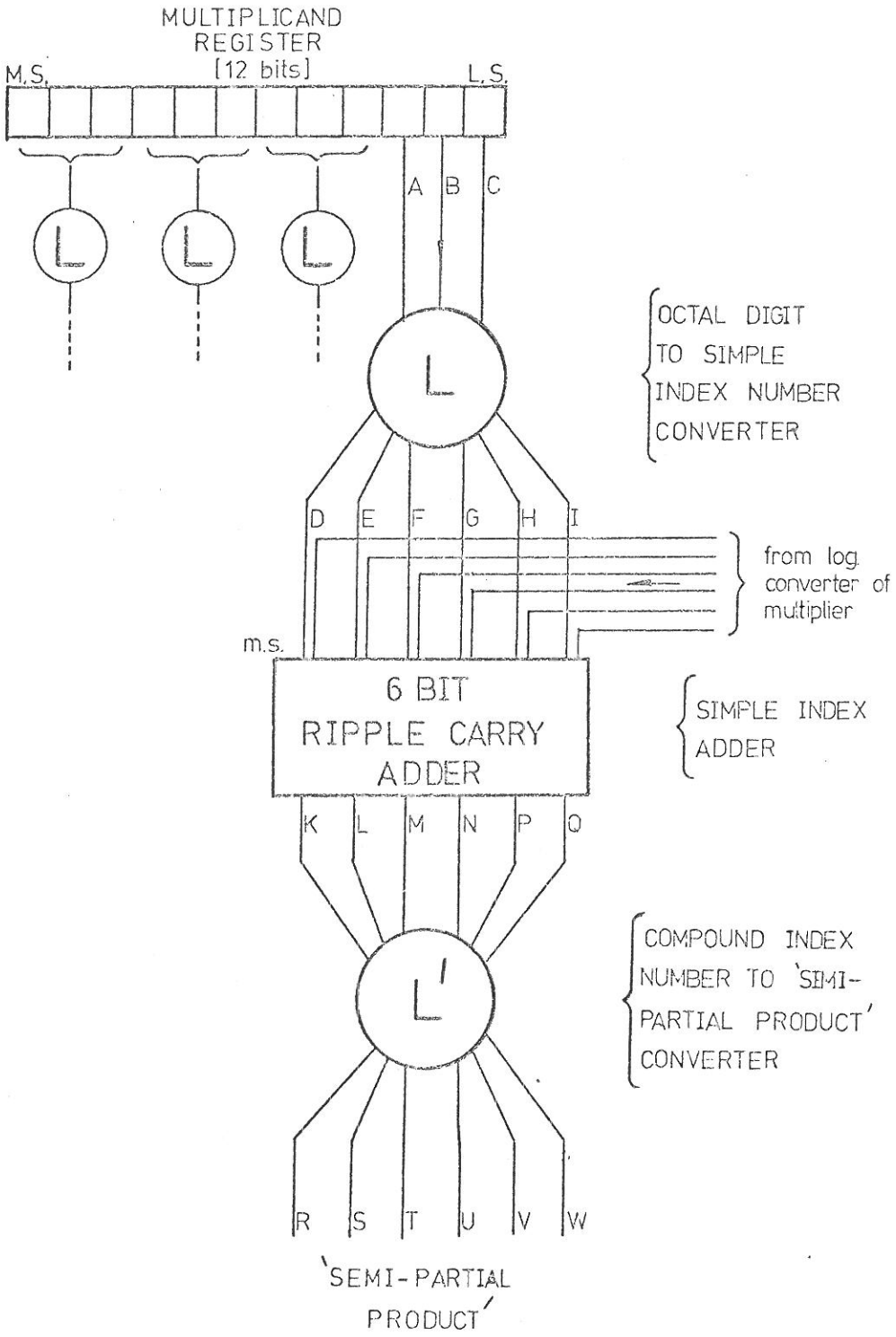
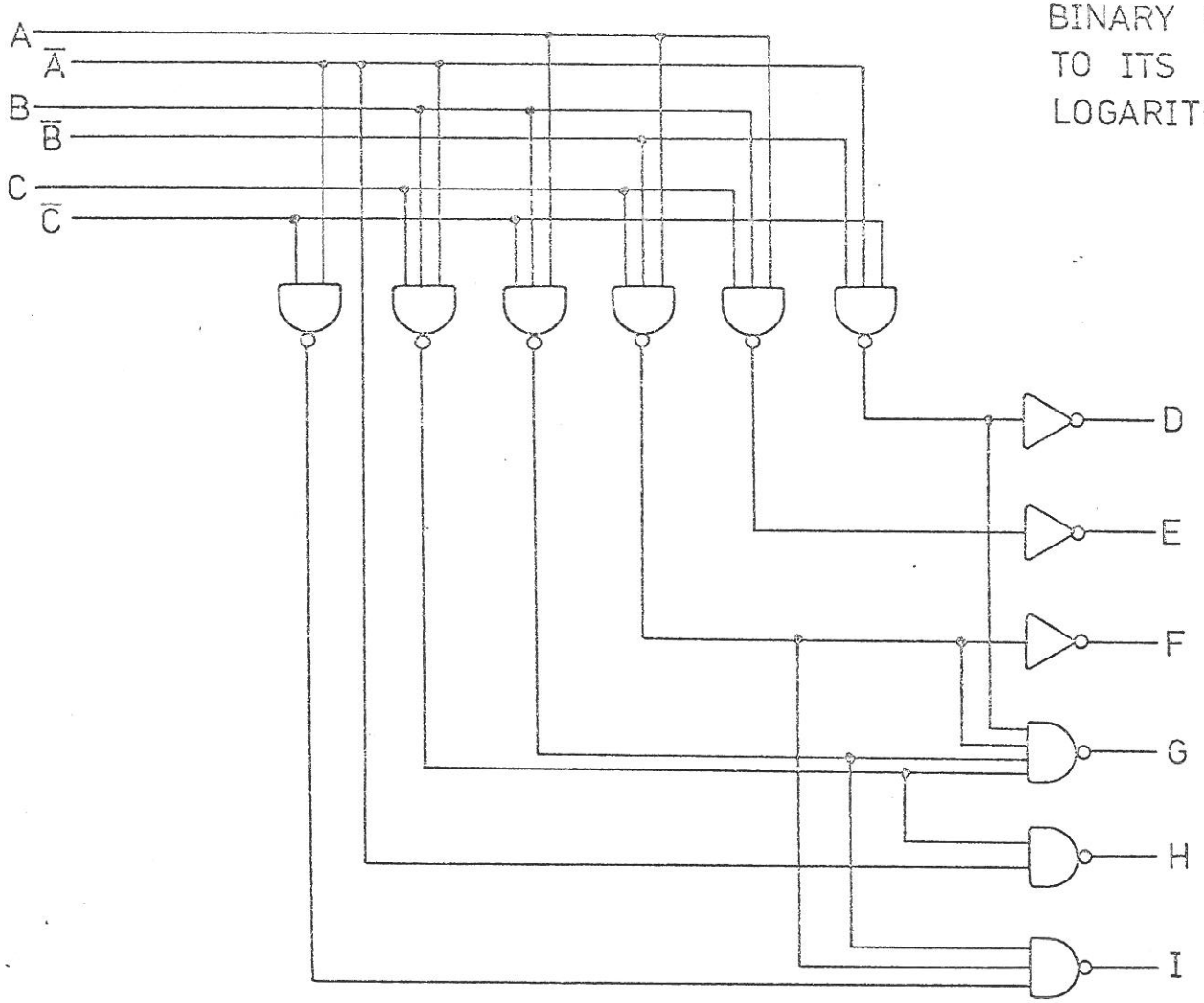


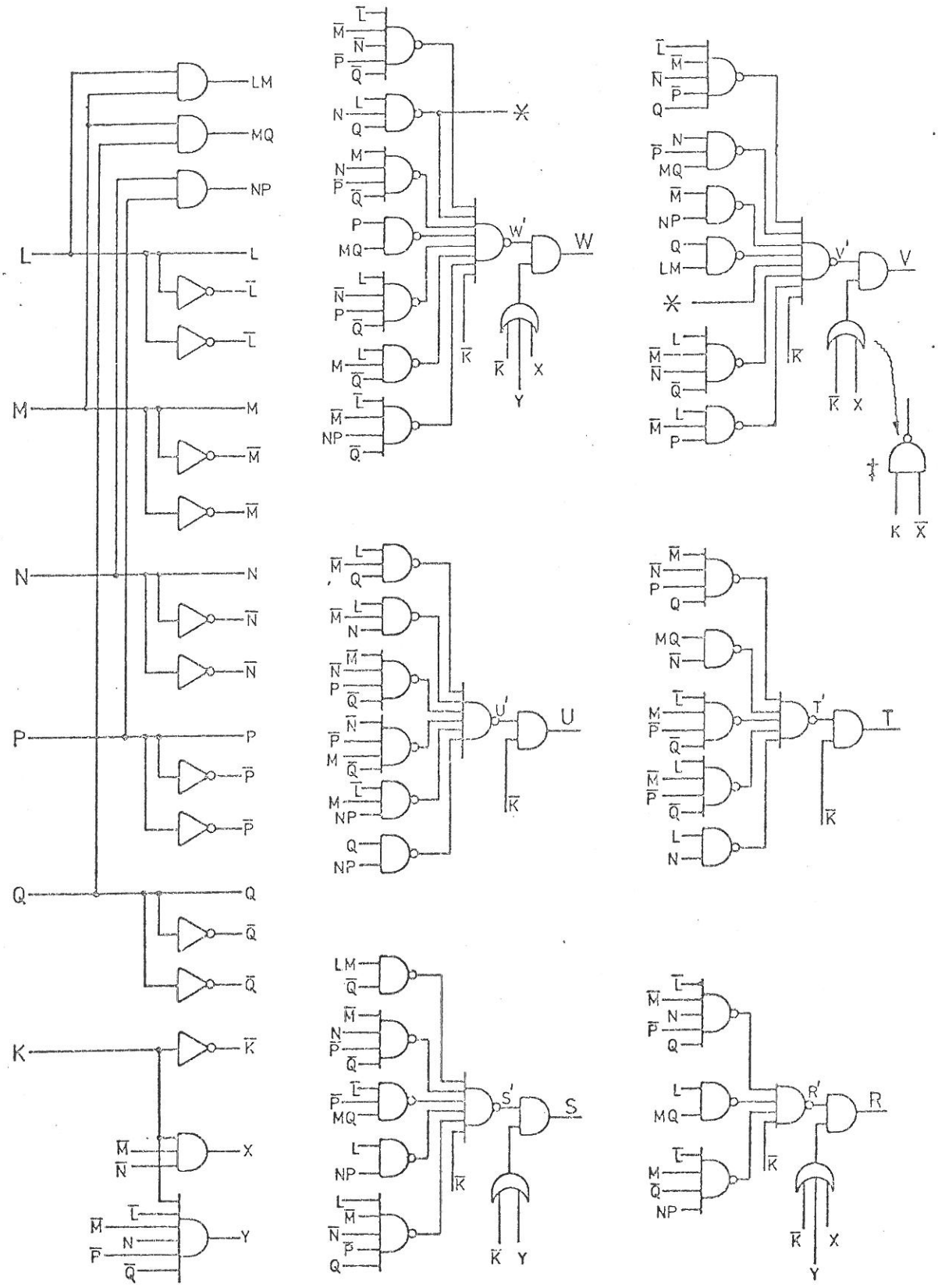
FIG. 11 –

ONE PATH THROUGH MULTIPLIER.

[BASE EIGHT DESIGN]

FIG. 12 — LOGIC CIRCUIT FOR CONVERTING A 3 BIT BINARY NUMBER [A,B,C] TO ITS 6 BIT BINARY LOGARITHM [D,E,F,G,H,I]



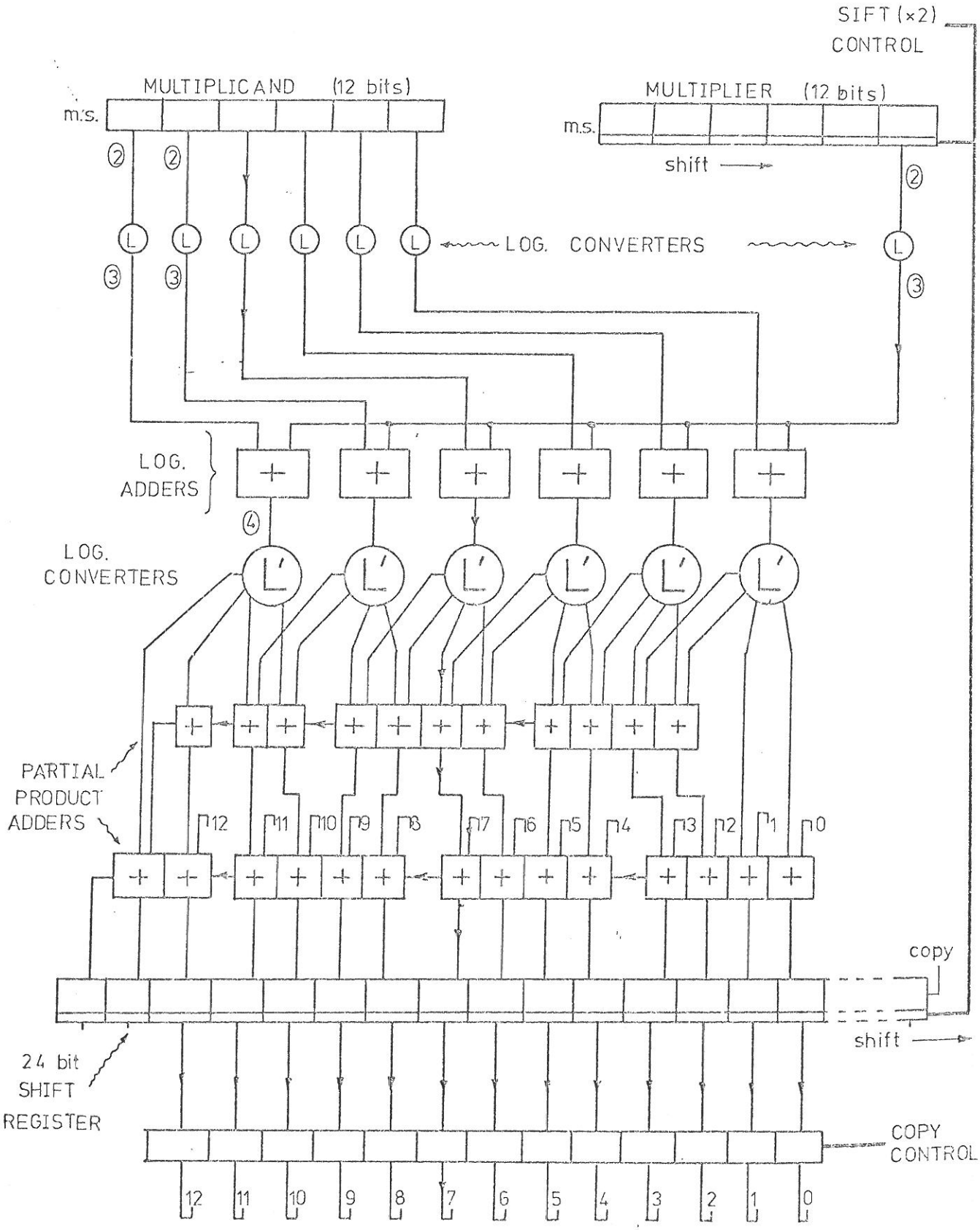


where 5 6 or 7 input gate occurs, assume 8 input gate with remaining inputs high.

† OR gates shown for clarity, may be replaced by nand gates.

FIG. 13—LOGIC DIAGRAM FOR COMPOUND INDEX NUMBER [K,L,M,N,P,Q] TO SEMI-PARTIAL PRODUCT [R,S,T,U,V,W] CONVERTER.

FIG. 14 — LOGIC DIAGRAM OF
BASE FOUR MULTIPLIER.



② : number of signals represented by one line,

2 : connect to other pin labelled 2,

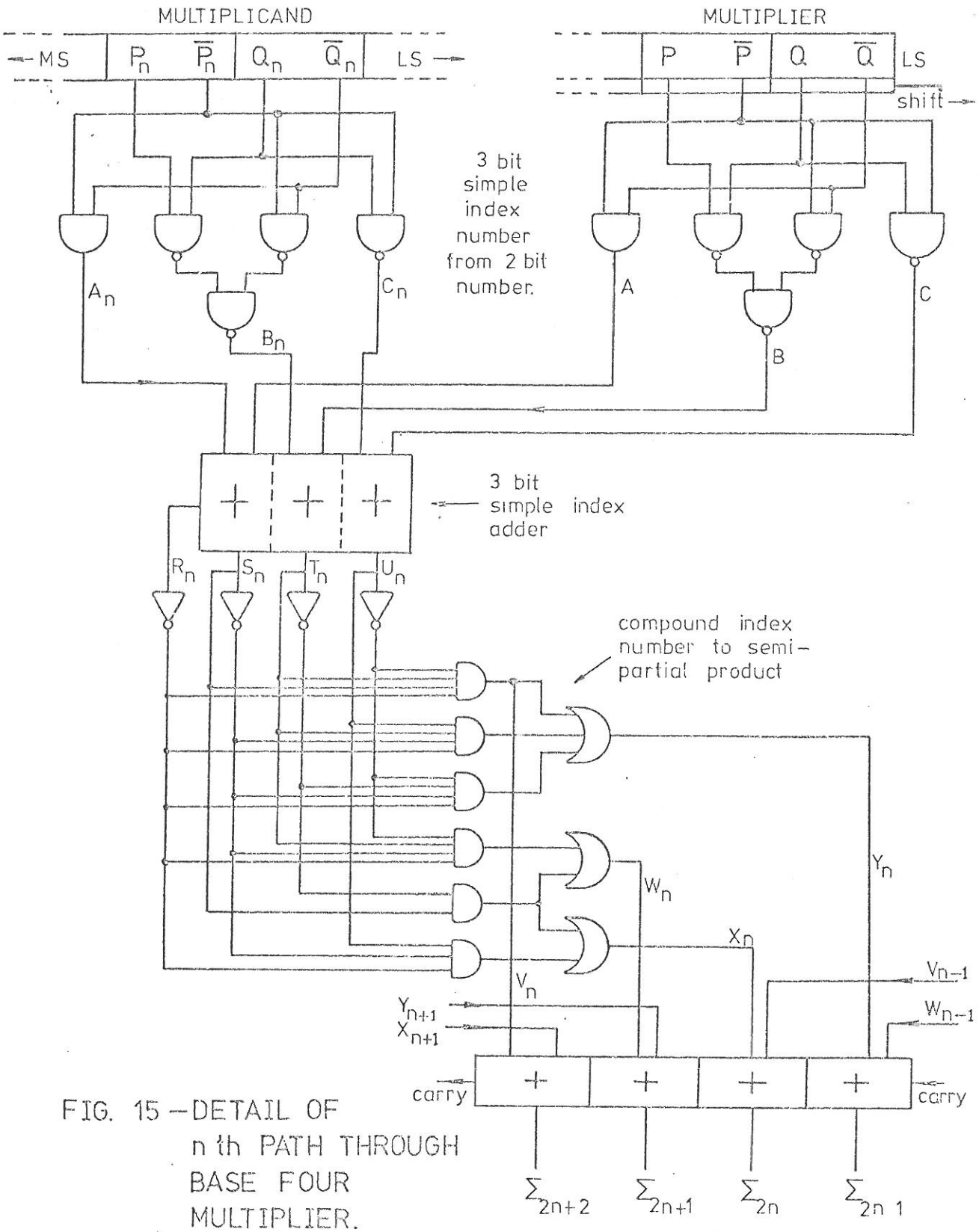


FIG. 15 – DETAIL OF n th PATH THROUGH BASE FOUR MULTIPLIER.

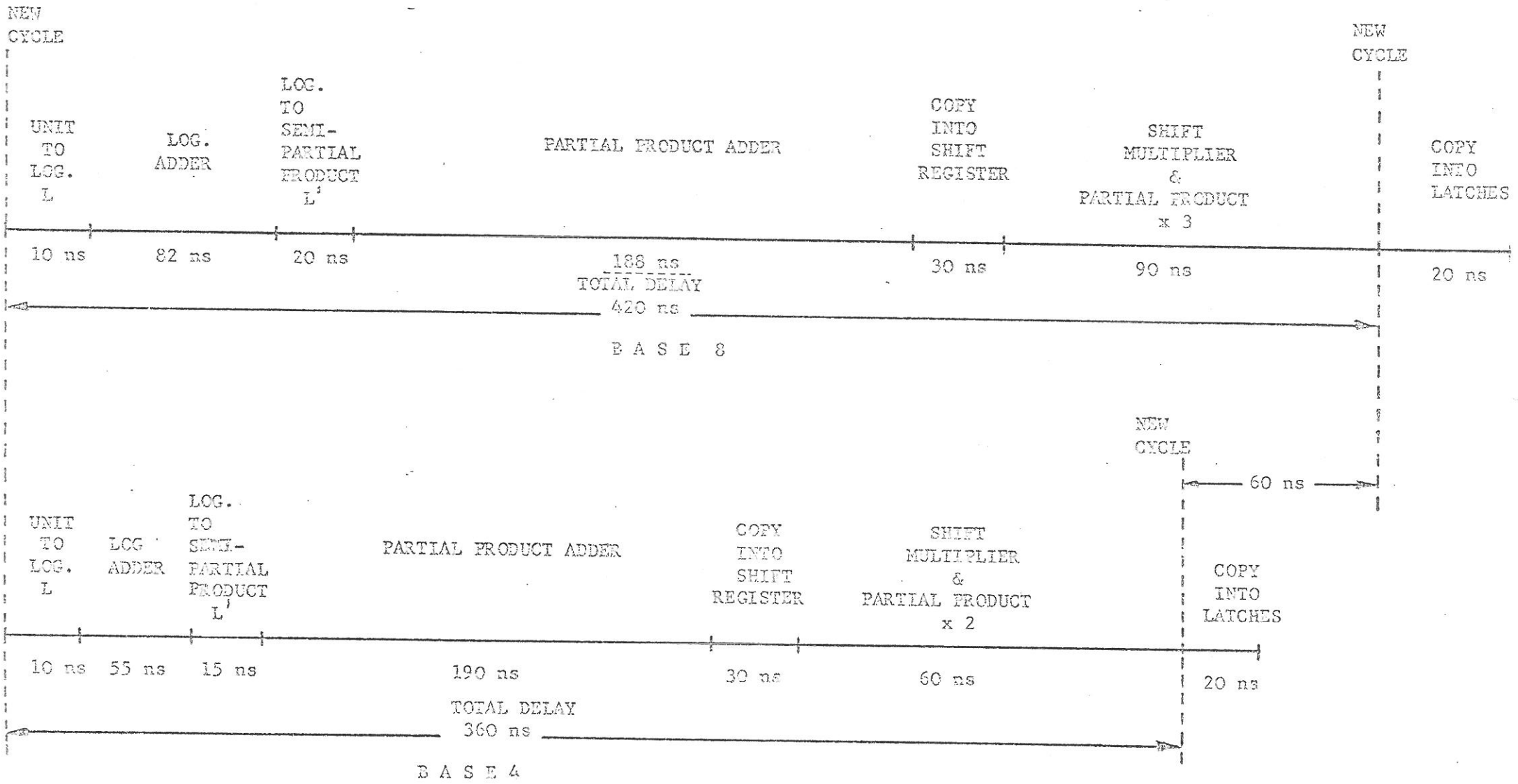


FIG. 16 - TIMING DIAGRAMS OF MULTIPLIERS FOR ONE CYCLE
(AS DESCRIBED IN FIGURES 2.2 & 2.6)

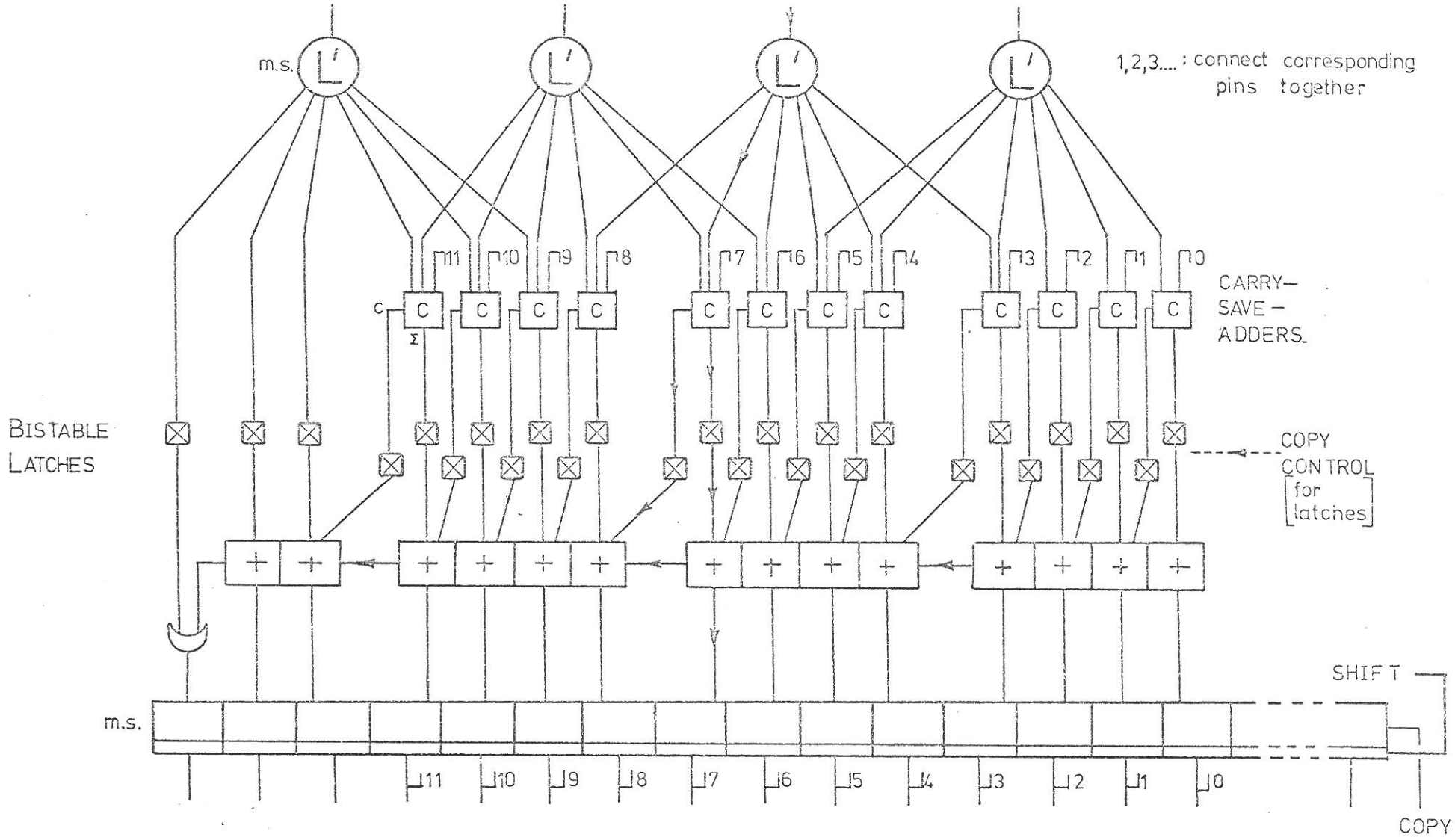
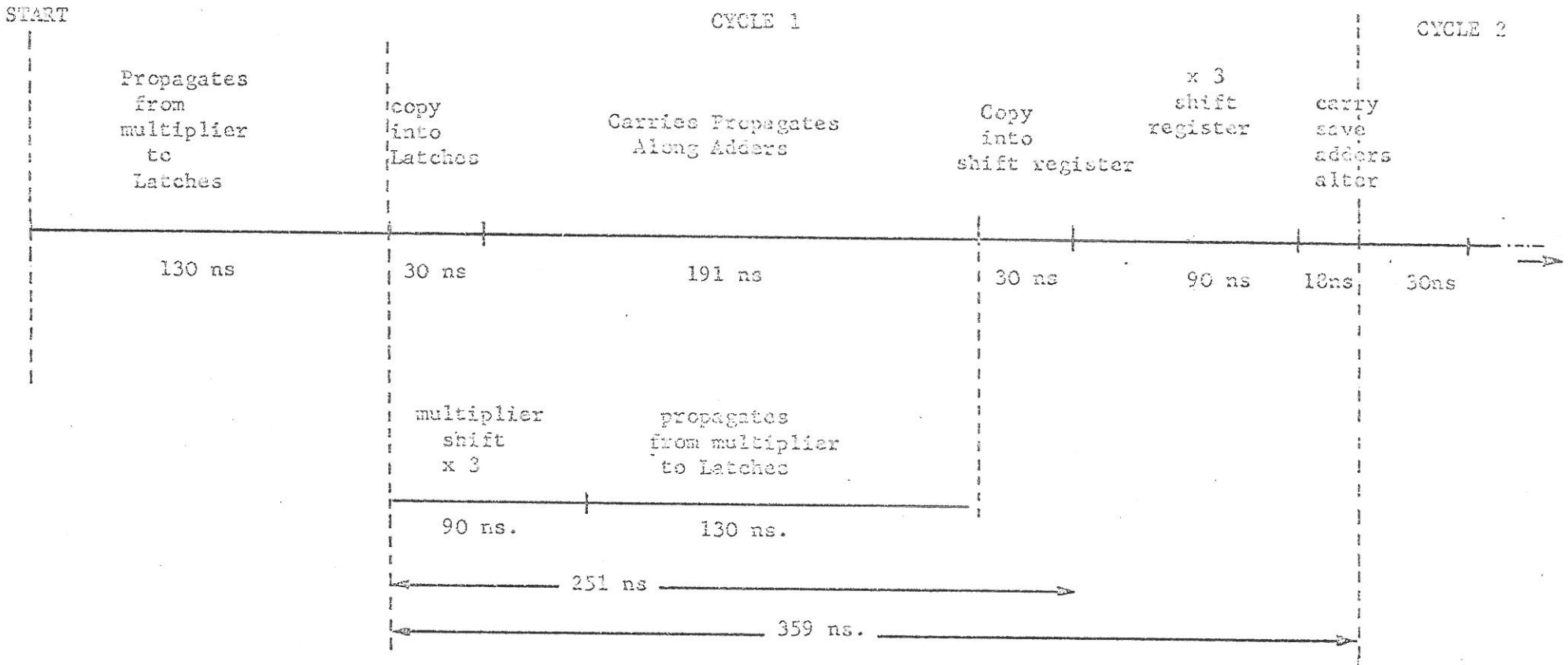


FIG. 17 — CIRCUIT FOR PARALLEL OPERATION.



TOTAL DELAY FOR MULTIPLICATION

$= 130 + 359 \times 3 + 251 = 1458$ ns

FIG. 18 - TIMING DIAGRAM FOR PARALLEL OPERATION OF BASE 8 MULTIPLIER

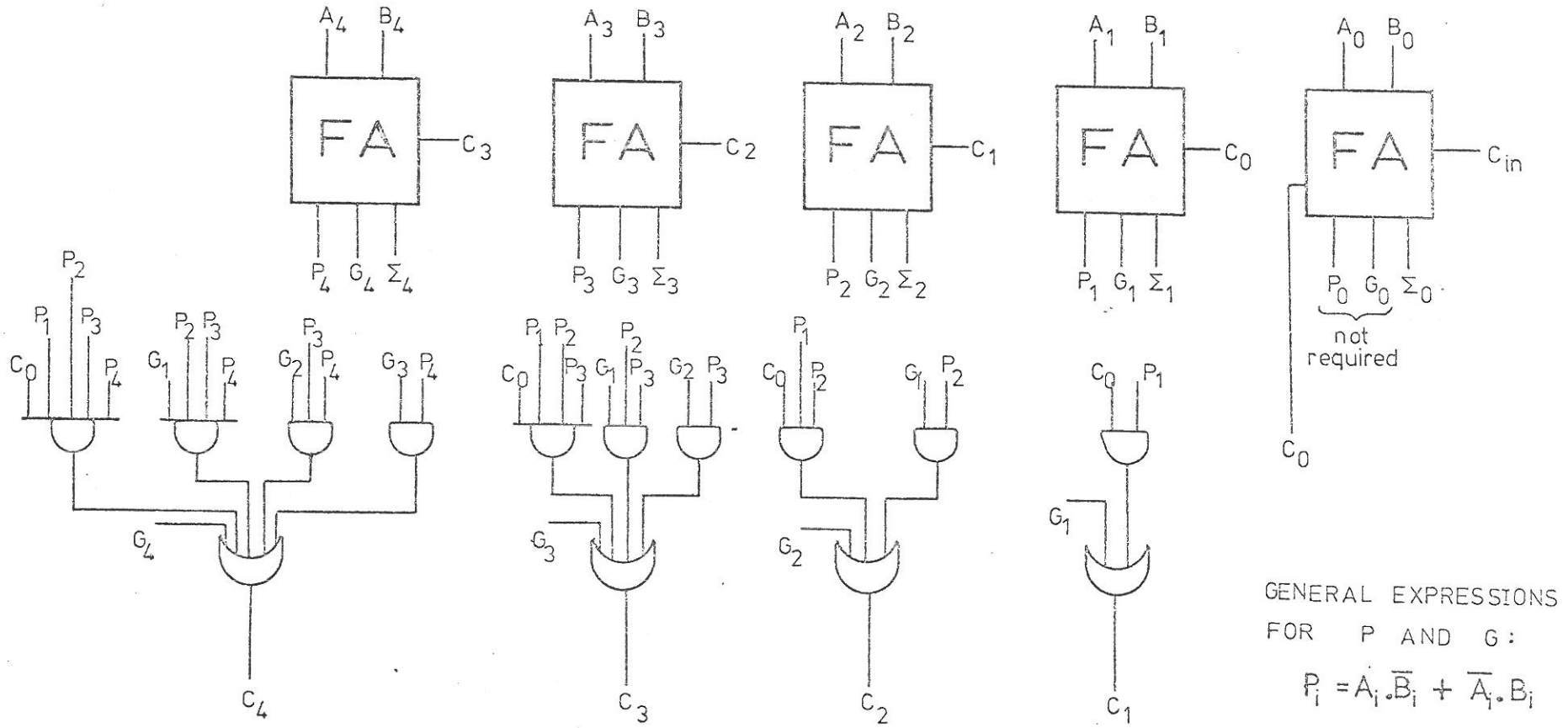


FIG. 19 — FIVE STAGE CARRY LOOK-AHEAD ADDER.

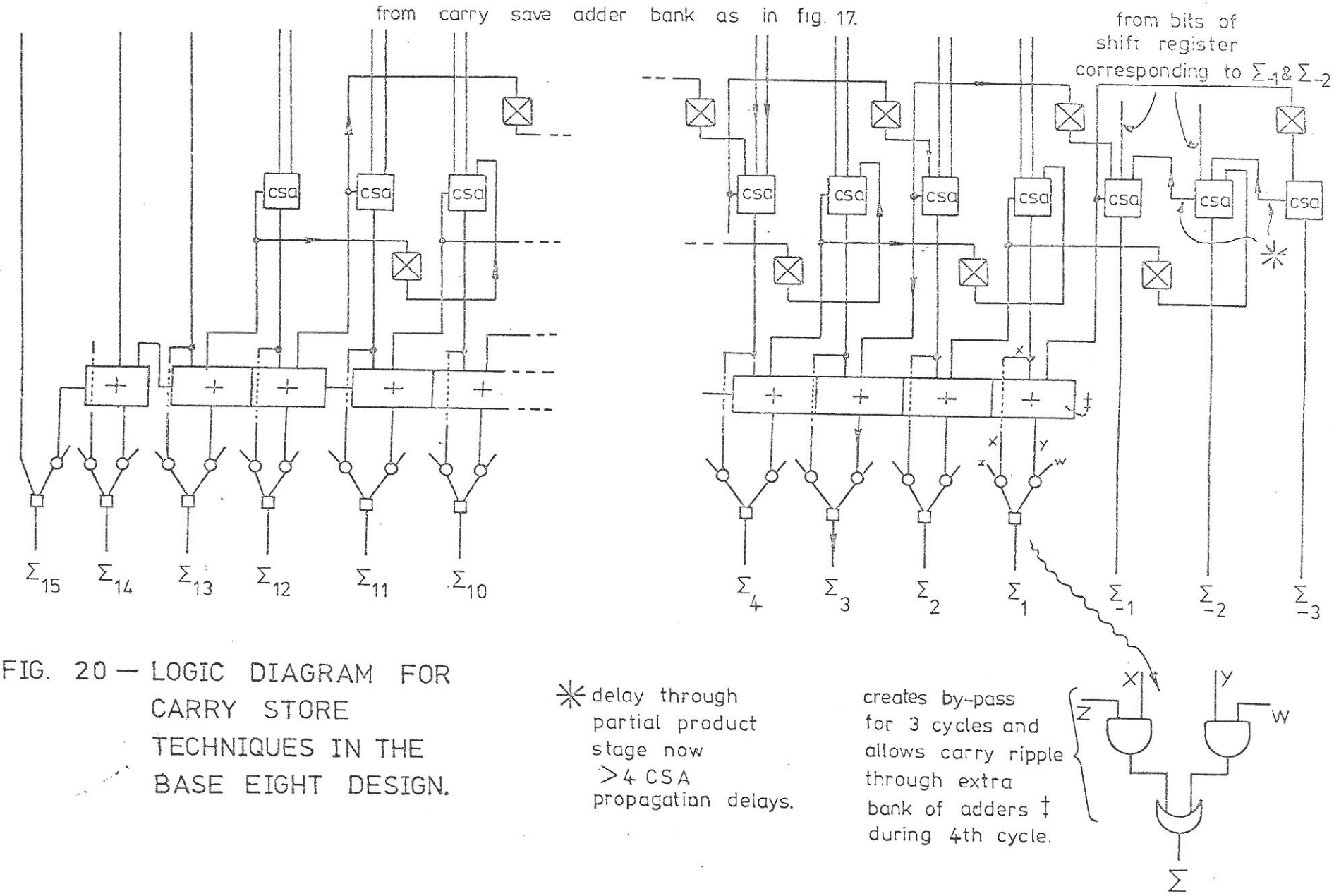


FIG. 20 — LOGIC DIAGRAM FOR CARRY STORE TECHNIQUES IN THE BASE EIGHT DESIGN.

✳ delay through partial product stage now >4 CSA propagation delays.

creates by-pass for 3 cycles and allows carry ripple through extra bank of adders ‡ during 4th cycle.

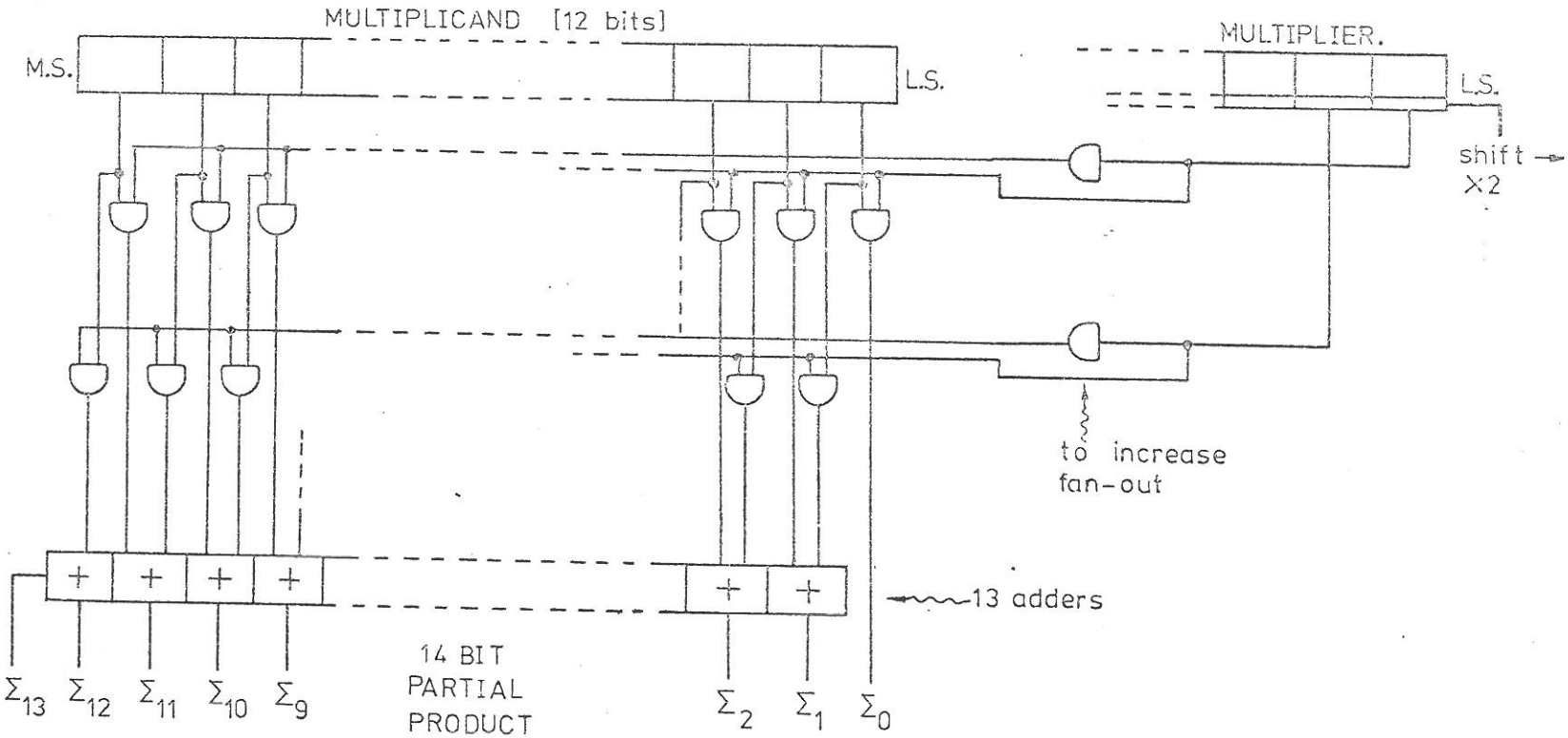


FIG. 21 — TWO BIT SHIFT MULTIPLIER.
[operating on addition alone]

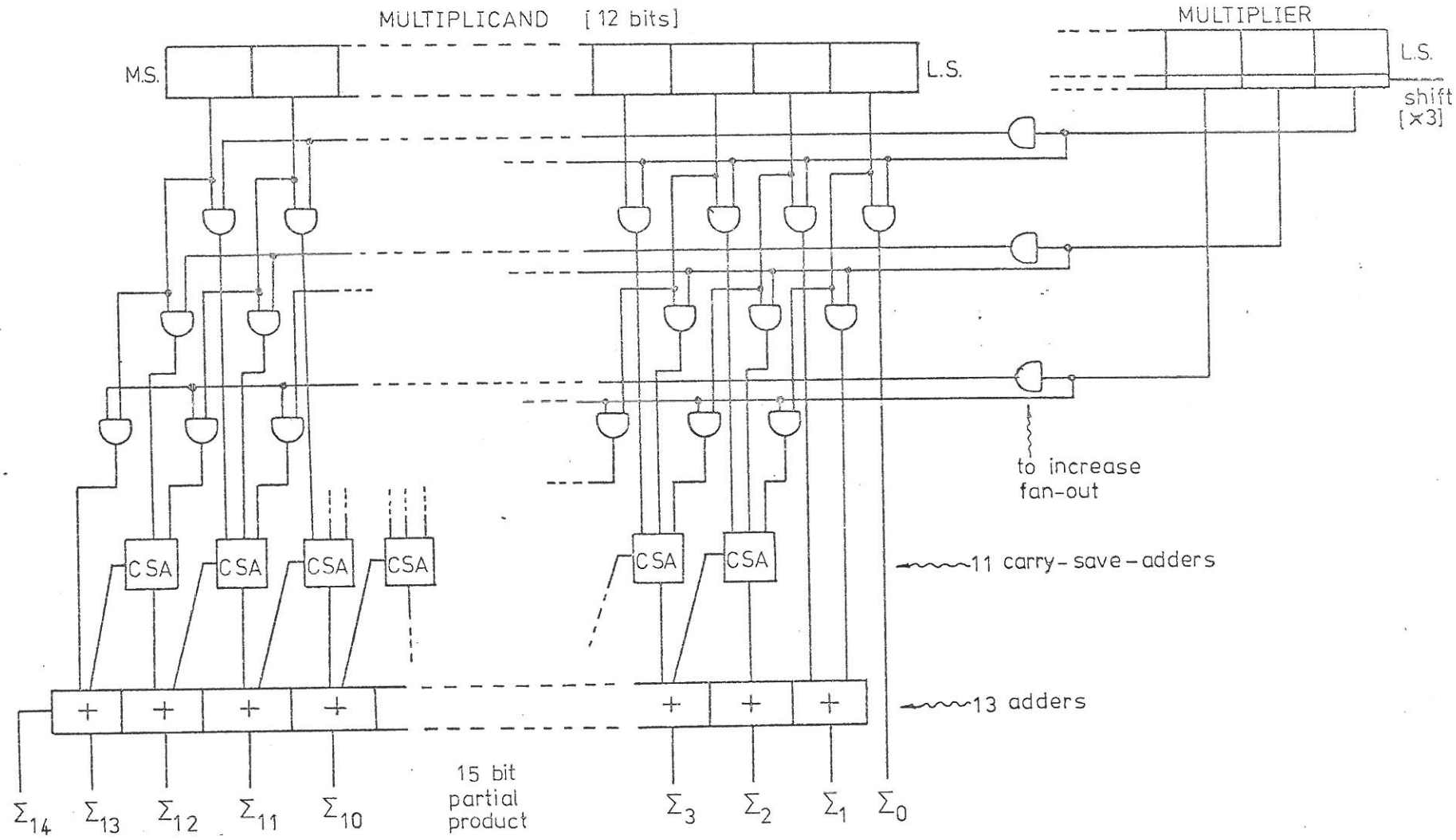


FIG. 22 – THREE BIT SHIFT MULTIPLIER.
[operating on addition alone.]

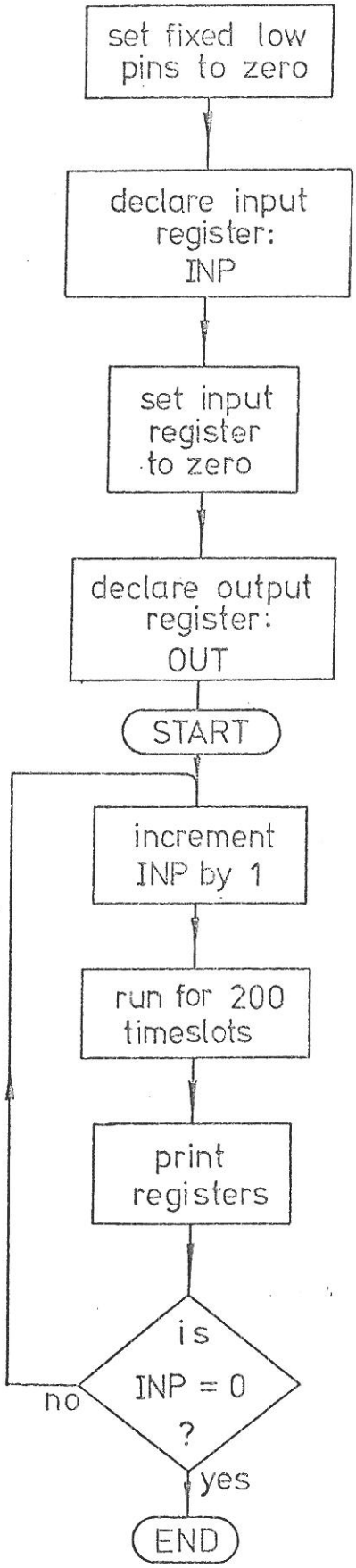


FIG. 23
SIMULATION PROGRAM
FLOW CHART.