NOTES ON THE LOGICAL DESIGN OF DIGITAL COMPUTERS AND ON SPECIAL CODING TECHNIQUES

Spring Term - 1951

Based on lectures given by

Charles W. Adams

as part of a special M I T course:

6.68 - Practice in the Use of Digital Computers

Prepared by

C. W. Adams

and P. A. Fox

ELECTRONIC COMPUTER DIVISION

SERVOMECHANISMS LABORATORY

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE 39, MASSACHUSETTS

TABLE OF CONTENTS

FOREWARD

The following pages contain a resume of the substance of a series of lectures given by C. Adams in the spring of 1951 as part of a course (6.68) on Practice in the Use of Digital Computers. Prerequisites for the course were experience with numerical methods and with programming for the Whirlwind computer. The course was divided into three roughly equal parts:

(1) lectures on logical design and special coding technique applied to computers in general,

(2) discussions, problems, and experiments on the logical design of, on coding for, and on the use of the Whirlwind computer, and

(3) preparation of a moderate-sized problem by each student for actual solution on the Whirlwind computer.

The notes here pertain largely to the first part of the course. The information on logical design is general and superficial; it is intended only as a survey to provide background for the coding work and to show the essential similarity of the many different automatic-sequenced computers. The material given here is based on extensive notes taken by Miss Phyllis Fox, so that much of the credit for them is due to her. All of the notes were edited by C. Adams so that any errors in them are solely his responsibility.

I NUMBER SYSTEMS AND REPRESENTATION IN COMPUTERS

A. Origin of Human and of Electronic Circuit Preferences

A digital computer is a device that performs sequences of arithmetic operations on numbers. Numbers are descriptive terms that arise in the process of counting, or of putting items into one-to-one correspondence with one another. People first learned to count by distinguishing between one and many; eventually they advanced a step further and counted one, a couple, a lot. When the need was felt for the ability to count higher, people began to enumerate against the most convenient standard of comparison available, and this was usually the digits on the end of their arms. This resulted in putting things into correspondence with fingers, and thereby learning to count up as one, two, three, . . ., nine, ten (which was two-hands-full). It was natural then to proceed two-hands-full and one, two-hands-full and two, etc. While there have been civilizations which counted in groups other than ten, most of those which amounted to anything counted in a decimal system largely because of the accident of having ten fingers.

When one goes to build a machine to be used by human beings, one wants a machine that works with decimal numbers. The usual desk calculators are accordingly all designed using the decimal system. While many of the large scale digital computers are also decimal, it is not always convenient, when numbers are to be handled by electronic circuits, to use a system which requires the distinguishing of ten different stable states. On paper, the number of convenient distinguishable symbols which can be defined is large; but, in vacuum tube work it turns out to be easiest to build reliable circuits which depend on the on-off, or switch, action of vacuum tubes without attempting to use different degrees of "on".

B. Comparison of Decimal with Binary Computers.

As a consequence, some of the large scale decimal machines (such as the ENIAC) fall back on ten different bi-stable circuits to represent a single decimal digit, while others (Mark II, Univac, SSEC, etc.) represent decimal digits by some selected ten of the sixteen ($2 \times 2 \times 2 \times 2 = 16$) different permutations of four bi-stable elements taken together. In other words, designers of electronic machines have almost universally built their machines making use of the preferentially binary quality of electronic

circuits even when the machine makes use of the decimal system throughout to allow for the decimal attitude of people who are to use the machines.

Many of the other computers (Whirlwind, EDVAC, SEAC, IAS, Raytheon, MIDAC, etc.) have made use of pure binary system, in which counting is by twos rather than by tens, so that each digit column represents a given power of 2. The digits then are simply 0 and 1.

one =	1.
two =	10.
three =	11.
four =	100.
five =	101.
⋮	
eight =	1000.
⋮	
seventeen =	10001.
⋮	
ninety-three =	1011101.

When a machine is to work with numbers in a binary system, some means must be provided for converting between the decimal system understood by the human users and the binary system understood by the machine. The conversion process is simple in principle but bothersome in execution; it will be covered later in these notes. The need for conversion naturally affects the engineering design of a computer, for one must decide whether it is harder to build a binary machine with a conversion method or a machine that works decimally throughout. The decision turns out to depend largely on how rapidly and in what quantity the machine must be able to handle decimal data and decimal results.

A pure binary system is an obvious choice for machines intended primarily for real-time applications such as control or simulation in which the machine communicates directly with a physical system, computes while the physical process is actually occurring, and forms results which are of fleeting or momentary value only; for here there are no numerical, tabulated data or results and the decimal system never enters the picture. For many so-called mathematical or scientific or engineering calculations, the data and results are necessarily decimal but they form a small percentage of the numbers used in the computation, and for other such calculations, graphical results are desired or, at any rate, are sufficient. In these cases, too, the pure binary system has obvious advantages. For some

engineering calculations, table-making, and business applications, the decimal system is evidently more desirable. In any event, the choice of the number system used is only one of several factors in the design of a computer.

C. Serial vs. Parallel Representation.

A second factor is the choice of the means of representing the several digits which make up each number. Generally, numbers are represented either as digits appearing at a given place one after the other in a time sequence or as digits appearing at several different given points all at the same time. The time-sequence representation is generally called serial; the place-sequence representation is called parallel. The choice here is very nearly dictated by the choice of the method of storage, as will be seen later.

There are, however, many compromises between strictly serial and strictly parallel types of operation. Some computers store serially and perform arithmetic in a parallel fashion while others store in parallel and perform arithmetic serially. In some decimal machines, the decimal digits are stored in serial fashion, but each decimal digit is represented by four binary digits, and the four binary digits comprising each decimal digit are stored in parallel, in four separate channels. So the serial-parallel distinction, though always blithely made, is not always a clear-cut one.

The manner in which the arithmetic is performed and the manner in which numbers are stored are closely interwoven with the choices between binary and decimal and between serial and parallel. It is necessary, then, to discuss arithmetic processes and storage techniques under several different assumptions as to the means of representing the numbers which must be operated on and remembered by any given machine.

II ARITHMETIC IN ELECTRONIC COMPUTERS

A. Binary Serial Addition - Adder.

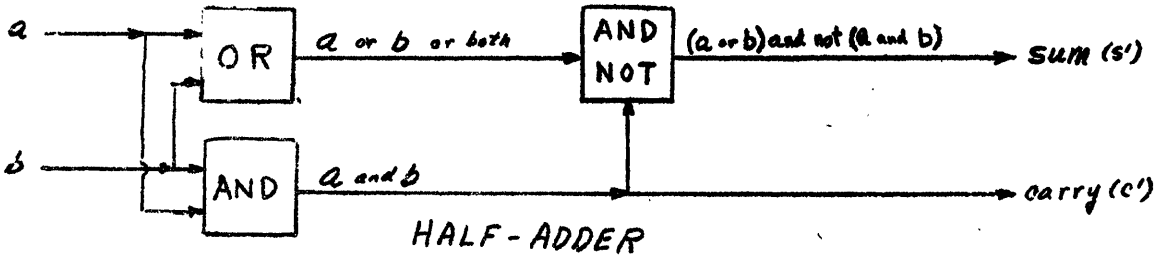
One of the basic (but not actually indispensable) abilities that any practical computer can have is the ability to add. Addition of two digits in the binary system is quite simple because there are so few combinations of digits. The addition table is simply:

$$0 + 0 = 0$$

$$0 + 1 = 1 + 0 = 1$$

$$1 + 1 = 10 = 0 + 1 \text{ to carry}$$

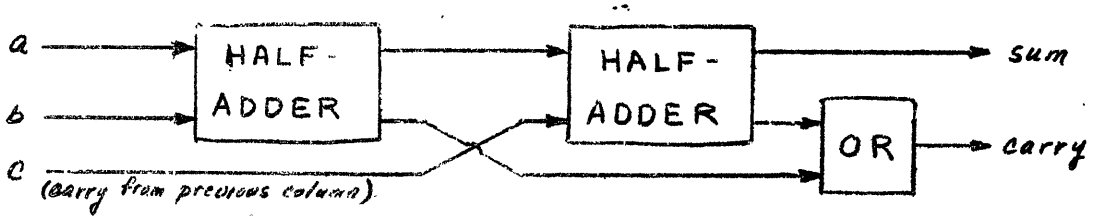
Assuming ones to be represented as the presence of voltage pulses and zeros as the absence of such pulses occurring at synchronized intervals, the sum of two digits can be formed fairly easily from a few easily-realizable logical circuits.



HALF-ADDER

In the diagram, the two digits (each represented by a pulse or no pulse appearing at a given instant) are shown as a and b. The OR circuit, or mixer, puts out a pulse if either a or b or both are ones. The AND circuit, a coincidence or gate circuit, puts out a pulse only if both a and b are ones(one pulse opens the gate and the other goes through it). The AND or gate circuit is usually formed from an electron tube with two control grids, both of which must be not negative before the tube will conduct. The same action can be obtained by judicious connections of two crystal diodes. The AND NOT circuit is really an AND with a NOT, or inverter, circuit on the bottom input. A pulse at the upper input will normally pass through the gate unless there is a pulse at bottom input, in which case the gate is closed. The circuit diagrammed above satisfies all the conditions required by the binary addition table given earlier.

In any except the rightmost digit column, however, an addition to be complete must allow not only for summing the two digits but also for the possibility of a carry from the previous digit column. It appears then that the circuit above performs only half the required task (hence the name, half-adder.) By putting two half-adders together, with an OR circuit thrown in, a whole adder can be obtained.



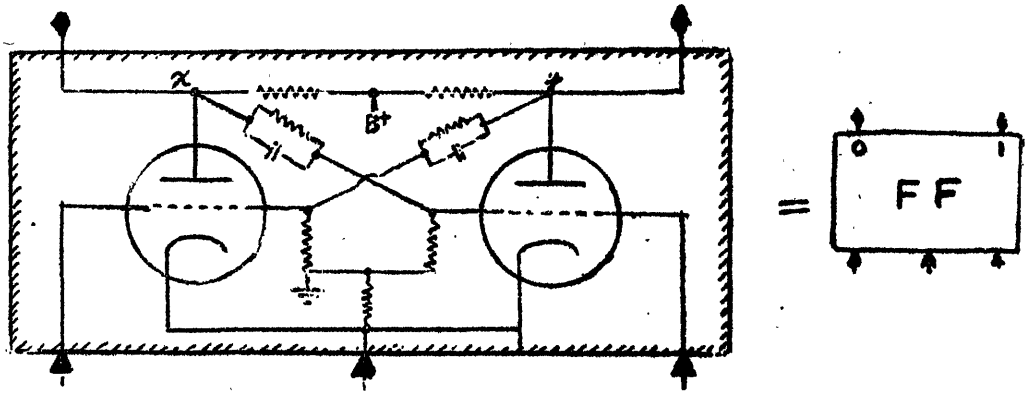
BINARY ADDER

B. Binary Parallel Addition - Accumulator.

Although it is much more commonly used in a serial machine, the above adder can actually be used in serial or parallel machines. Alternatively, binary numbers can be added in a parallel fashion by adding one number (appearing as pulses or no pulses on a set of parallel lines, one line for each digit column) to a second number stored in a set of flip-flop circuits, as is shown the (a) half of the attached drawing (B-34437). A circuit which thus accumulates sums is usually called an accumulator.

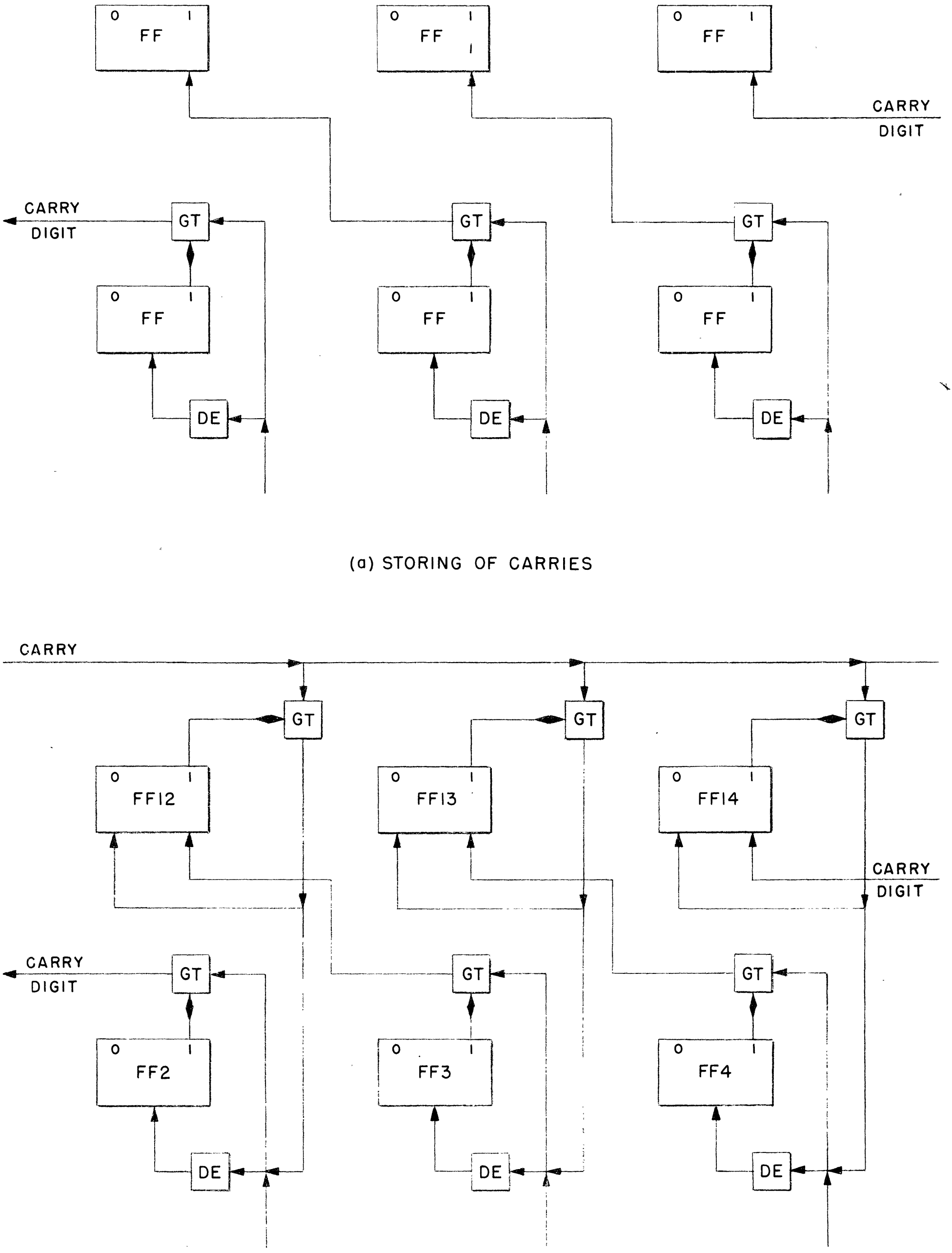
1. Flip-flops

a. Circuit. A flip-flop, or trigger circuit, or multivibrator, is a circuit containing a pair of electron tubes so connected that only one can be conducting at a given time. The two states are arbitrarily designated as 0 and 1. The basic (Eccles-Jordan) idea of the flip-flop is shown below.



When the left tube conducts, point x becomes more negative which drives the grid of the right tube negative and cuts the right tube off; if the right tube conducts, the action is reversed. When the common cathodes are driven positive, both tubes momentarily conduct but then the cross-coupling condensers cut off the previously conducting tube. As shown, there are three inputs, one to insert bodily a zero, one to insert a one, and one to trigger the flip-flop (i.e., to switch the flip-flop from whatever state it is in to the other state). And as indicated, the flip-flop has two outputs which are normally used to control gate tubes.

b. AC and DC Coupling - Restoration. The points x and y are normally both at rather high positive potentials, even though one is at a lower potential than the other. When gate tubes are connected



SIMPLE ADDER

to these outputs, the gates must be capable of operating with their grids at the flip-flop plate potential, or else the gates must be coupled to the flip-flop through condensers. When condensers are used, the flip-flop is called AC-coupled. AC-coupling requires that the flip-flop be flipped and flopped oftener than some minimum rate in order to maintain the proper charge on the coupling condensers. This minimum rate is guaranteed in the Whirlwind Computer by periodically (every 16 microseconds) stopping everything and complementing every flip-flop in the computer twice — the process being known as restoration. Need for such restoration can be avoided by using DC-coupled flip-flops, in which the plates of the flip-flops are usually operated at ground and the cathodes and grids at a large negative potential (eg. -150^{V}).

c. Logical vs. Electronic Schematics. It should be pointed out that the logical, or block, diagrams which are shown and discussed here do not show all the electronic details - for instance: arrowheads imply that circuits are so constructed that pulses travel only in the direction of the arrows (crystal diodes are used when necessary); delay blocks are used when and only when logical delay is necessary, whether or not actual electronic delay is or is not needed; etc.

2. Addition

In the diagram of parallel addition, for instance, a pulse, if any, coming in from the bottom in any one of the digit columns does two things: (1) it attempts to pass through the gate tube; and (2) after a short delay it triggers the corresponding flip-flop. If the flip-flop originally contained a zero, the gate is closed and the only effect of the adding pulse is to change the flip-flop to a one (0 + 1 = 1 and 0 to carry). If the flip-flop originally contained a one, the gate is open and the pulse gets through the gate to become a carry. The sum flip-flop is triggered back to a zero and the carry flip-flop is set to a one (i.e. the carry is stored) so that the result is 1 + 1 = 0 and 1 to carry.

As in the case of the serial half-adder, however, the addition process shown in B-34437a does only half the job, for the carries remain to be dealt with. There are two methods of treating these carries which are discussed under the names of "low-speed carry" and "high-speed carry."

3. Low and High Speed Carry

The low speed carry method is shown in Figure B-34437b. The lower flip-flops store the partial sums of an addition of two numbers, and the upper flip-flops store the carries. The addition of the carries is performed in several stages by a series of pulses on the carry line. A pulse on the carry line is gated through by any carry flip-flop containing a one to add the carry into the next column. Conceivably new carries may arise from this addition, so that the carry line must be pulsed again, and so on until no carries remain. For a number with n digit places, the process may become quite lengthy, requiring as many as n carries and it becomes desirable to devise a method for treating all the carries at one time.

The high-speed carry performs all the carries in one step. The logical difference between this and the sequential operation of the low-speed carry is shown by the figure below which shows the addition of two binary numbers.

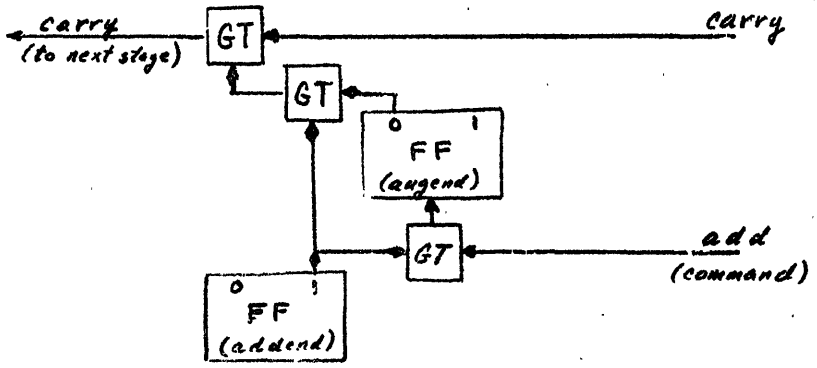
x =	1	1	0	1	1	0	1	0		x =	1	1	0	1	1	0	1	0
y =	1	0	1	1	0	1	1	0		y =	1	0	1	1	0	1	1	0
I	1↖		1↖		1↖				Carries		1	↙---1	↙---1					Carries
	0	1	1	0	1	1	0	0	Partial sum		0	1	1	0	1	1	0	0 Partial sum
II		1		1					Carries		1	1	0	0	1	0	0	0 0 Result
	1	0	1	0	0	1	0	0	Partial sum									
III		1		1					Carries									
	1	0	0	0	0	0	0	0	Partial sum									
IV	1	1	0	0	1	0	0	0	Result									

Sequential, Low-speed Addition — Simultaneous, High-speed Addition

Because the original addition of two digits cannot produce both a partial-sum one and one to carry, a carry arising at any stage may be passed along by special gate tubes to the left until it reaches the first position containing a 0. the 1's over which it travels all being triggered to 0. An electronic arrangement for performing the process is shown in Figure B 34438. A single pulse on the carry flip-flops, after the partial sum (and carries) have been formed, suffices to give a final result, although enough time must be allowed for the pulse to travel through at most n high-speed-carry gate tubes.

It is possible to entirely eliminate the carry flip-flops by

using a gate tube in the following way. The two input grids are controlled by the partial sum in the flip-flop and by the value of the addend (presumably stored in another flip-flop). Only when both grids are positive will the upper gate be open, permitting a pulse on the carry line to get through the tube. In the addition process, the partial sum is first formed.



This has no effect on the carry line as such. Then the carry line is pulsed and a pulse is transmitted only if <u>both</u> A is positive (0 + 0 = 0 <u>or</u> 1 + 1 = 0) and B is positive (addend = 1), i. e. only for the case 1 + 1 = 0 and 1 to carry.

C. <u>Binary Subtaction and Negative Numbers</u>

Several methods may be used to enable a computer to subtract. For instance, an entirely separate subtracting circuit may be constructed, or extra components may be inserted into the existing addition circuit to enable it to subtract. The resultant added hardware will in general be both undesirable and unnecessary. Subtraction of a number <u>y</u> from a number <u>x</u> may also be performed by adding to <u>x</u> the negative of <u>y</u>. For this, some means of handling negative numbers so that they add like positive numbers must be devised.

1. Tens Complement

Consider, for example, numbers in the range $-1 \geq x > 1$. Suppose the negative numbers are made positive by adding 2 to each of them. Then the $-1 \geq x > 1$ range is represented by numbers in the range $0 \geq x > 2$. The quantity $+\frac{1}{2}$ is represented in binary form as 0.1000, while the quantity $-\frac{1}{2}$ is represented by $2 - \frac{1}{2} = 3/2 = 1.1000$. Numbers can be added together exactly as in the addition circuits discussed above, except that:

1) when two negative numbers are added, 2 must be subtracted

from the sum. (i.e. $(2-x) + (2-y) = (2-2) + 2$)

2) when a positive and a negative number are added to give a positive sum, 2 must be subtracted from the sum. (i.e. $x + (2-y) = 2 + 2$)

This subtraction of 2 is automatic when the addition is carried out in a normal way if any carry off the left-hand end is simply disregarded.

In using the above complement scheme, there exists the possibility of confusion between a complement and a real positive number greater than one. Obviously, this possibility of results overflowing the allowed -1 to 1 range (i.e. of positive numbers greater than one and of negative numbers less than minus one) must be considered. This problem can be handled by:

1) not assuming responsibility for the operation of the computer when the program gives rise to an overflow, or by
2) building in circuits which detect the occurrence of an overflow. This latter can be done fairly easily, for the addition of two positive numbers must not give rise to a one in the left-most digit nor two negative numbers to a zero (addition of a positive and a negative number cannot possibly cause an overflow).

In the complement convention described above, the left-most digit is automatically indicative of the algebraic sign of the number (i.e. zero for plus, one for minus). The convention as described is called the tens, or twos, or radix, complement because the complement of x is obtained by subtracting x from the base or radix (two) of the number system being used. To obtain the complement of x, knowing x itself, one simply replaces zeros by ones and ones by zeros, and then adds a one to the right most digit column, performing carries where necessary. Thus, for example,

$$2 = 10.00000 = 1.11111 + .00001$$
$$20/32 = 0.10100 = \underline{0.10100}$$
$$2-20/32 = 1.01100 = 1.01011 + .00001$$

i.e. Tens complement = $1 \rightarrow 0$ and $0 \rightarrow 1$, + 1 added to right-most digit

In the decimal system, the tens complement of $\underline{x}$ can be taken to be ten-x, so that one forms the complement by subtracting each digit from 9 (rather than from 1 as was the case in the binary system) and by then adding 1 to the rightmost digit. Thus

$$10 = 10.00000 = 9.99999 + .00001$$
$$20/32 = 0.62500 = 0.62500$$
$$10-20/32 = 9.37500 = 9.37499 + .00001$$

Thus in the decimal system, complementary digits are 0 and 9, 1 and 8, 2 and 7, 3 and 6, 4 and 5. Notice that if the numbers are restricted, as before, to the range $-1 \geq x > 1$, the leftmost digit can be only 0, for plus, or 9, for minus. Therefore, the leftmost, or sign, digit could just as well be a binary digit in the event the allowable range were -1 to 1.

The tens complement system is simple, convenient, and natural. It is familiar to most mathematicians (and high school students) the form of co-logs (or complementary logarithms) used to eliminate the need for subtracting when summing columns of mixed positive and negative logarithms. It is also familiar as the form which negative numbers take on a desk calculator. It should be noted that restriction of the range to $-1 \geq x > 1$ and subtracting from two or ten was simply an example. In practice, any range can be used and complements can be formed by subtracting from 2^n or 10^n for any value of n for which 2^n or 10^n is outside the allowable range.

2. Nines Complement

In large-scale machines it is convenient to be able to form the complement of a number easily. The first step of forming the tens complement is easy, but the addition of the one at the righthand end causes troubles in that the complement must therefore be formed in a counting register to allow for possible carries. It is natural, then, to consider working with the complement on nines, or ones, or radix-minus-ones. In this system, the complement is formed by subtracting from 2^n-2^m, where 2^m is chosen as small as or smaller than the rightmost digit and 2^n as large as or larger than the leftmost digit of the numbers in the allowable range. With six-binary-digit numbers in the $-1 > x > 1$ range, for instance,

$$2^0 - 2^{-5} = 1.11111$$
$$\underline{20/32 = 0.10100}$$
$$2^0-2^{-5} - 20/32 = 1.01011$$

As with the tens complement system, addition is straight forward, but now the cases of adding two negatives, or of adding a positive and a negative to get a positive, result in numbers from which 2^n-2^m, rather than simply 2^n, must be subtracted. The subtraction of 2^n is again accomplished automatically by neglecting to carry beyond the leftmost digit. Furthermore, the same neglected carry pulse can be used to add the 2^m (by pumping the carry from the left end into the right end) thereby accomplishing the complete subtraction and yielding the correct result. The process is known as end-around-carry. Thus,

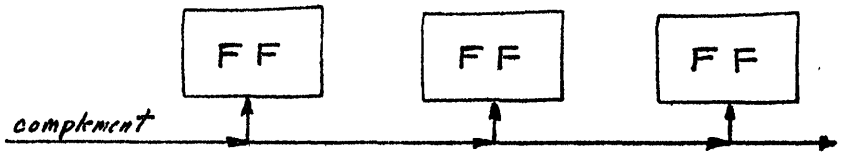
17/32 = 0.10001	-13/32 = 1.10010
-11/32 = 1.10100	-12/32 = 1.10011
$\overset{1}{0}$.00101	$\overset{1}{1}$.00101
6/32 = 0.00110	-25/32 = 1.00110

NINES COMPLEMENT: END-AROUND-CARRY

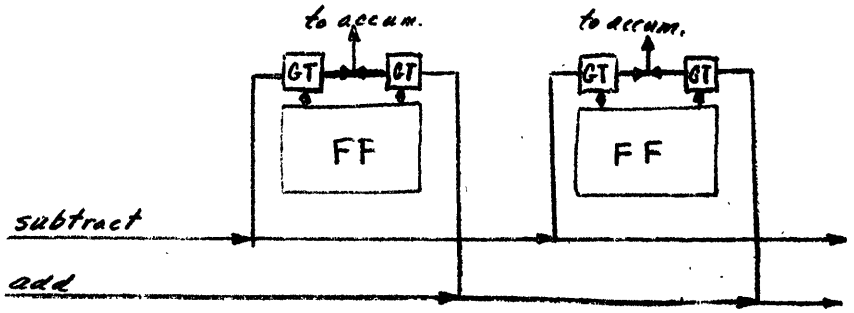
As in the tens complement system, the range must be d fined and overflows disallowed. Note that here the range is $-1 > x > 1$ rather than $-1 \geq x > 1$, because $(2-2^m) - 1 = 1-2^m$ so that both -1 and $+1 - 2^m$ have the same representation and cannot both be included in the allowable range. The loss of one possible number actually goes to compensate for the fact that 0 has a second, or negative, representation in the nines complement system, namely $2-2^m - 0 = 2-2^m$ (e.g. 1.11111). The inability to represent -1 and the ambiguity of 0 are the two major drawbacks of the nines-complement system. Actually, with an additive arithmetic unit such as has been described, only -0 and never +0 can arise, while if the arithmetic unit is subtractive (this has not been discussed here) only +0 and never -0 can arise. However, with an arithmetic control such as is used in Whirlwind, it is possible to generate +0 as the result of multiplication, divisions, etc., and this ambiguity is occasionally troublesome as will be seen later.

3. Circuits for Complementing and Subtracting

In the nines complement system, a number in a flip-flop register may be made negative by complementing each flip-flop.



The negative of a number can be read from a register without changing the register contents, by sending pulses through gate tubes on the zero-side of the flip-flop.



The same circuitry works for the tens complement system, except that a one must be added to the register after it has been complemented, and a one must be added to the accumulating register after it has been subtracted into.

D. Decimal Representation in Bi-Stable Circuits.

There are several general methods for representing decimal digits using bi-stable circuits. They may be represented in binary-coded form by four binary digits with some coding involving the assignment of one decimal digit value to each of some chosen ten of the sixteen possible permutations of four binary digits. They may be represented by a decade ring which has ten numbered bi-stable circuits, some one of which is in an abnormal state while the others are in a normal state, the one which is abnormal indicating the value of the digit. Decimal digits may also be formed from a binary digit in conjunction with five bi-stable circuits of which some four are in a normal state—the bi-quinary system—and in myriad other less common ways.

Among the binary-coded representations there are a number of choices, some of which are described below.

a) Straight Binary Representation

0 = 0000	5 = 0101
1 = 0001	6 = 0110
2 = 0010	7 = 0111
3 = 0011	8 = 1000
4 = 0100	9 = 1001

b) Excess 3's code. The code is as follows:

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

This system has the advantages that the complement of a number on a decimal basis is obtained by changing 1's to 0's and conversely, and that straight binary addition gives rise to a carry at the proper time. Of course the fact that the decimal range 0 - 9 is translated into binary 3 - 12 complicates the unit to the extent that a sum (formed binary-wise) is in error by either +3 or +13 and must be corrected. That is, if $x + y = z$, with $z < 10$, $(x + 3) + (y + 3) = (z+3) + 3$ and subtraction of the 3 is required, while for a sum $z > 9$, causing a carry, $(x + 3) + (Y + 3) = [(z+3) - 10] + 13$, so that 13 must be subtracted. This correction process is actually no harder than correcting the sum after adding in straight binary representation.

c) Two-star Four, Two, One Code

A third method of decimal representation by four flip-flops is called the 2*421 method. The term derives from the fact that the last three flip-flops represent the true binary values weighted 2^2, 2^1, 2^0 respectively, while the first flip-flop is weighted 2^1 rather than 2^3.

0	0 000	5	1 011
1	0 001	6	1 100
2	0 010	7	1 101
3	0 011	8	1 110
4	0 100	9	1 111

This amounts to replacing the range 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 by 0, 1, 2, 3, 4, 11, 12, 13, 14, 15, and has the same general advantages and complications as the excess 3 code.

E. Decimal Addition.

Addition of two numbers within a computer proceeds in one of several fashions according to the particular method of number representation in the machine, according to the speed desired, and according to the whim of the designer.

For a binary-coded machine (e.g. the Harvard machines, Univac, etc.) addition may be done either by a direct adoption of binary addition techniques, or by modified counting circuits, or by means of an auxiliary (e.g. wired-in or stored) addition table. In some cases, for instance, the binary numbers representative of the decimal digits in each decimal digit column are added as straight binary numbers. Necessary carries either arise naturally (if the excess-3 code is used for instance) or are generated in some fashion. The sum in each of the digit columns is then corrected by adding or subtracting something to it to give the correct binary-coded result of the decimal sum. In a binary-coded machine using a sequence of n pulses to transmit the decimal digit n, a binary counter modified to count through only the ten chosen combinations can be used. In the third method, an addition table is used, i.e. a circuit is designed with two ten-alternative inputs and two outputs, one with ten alternatives, giving the partial sum of the two input digits, and one with two alternatives, giving the 0 or 1 to carry. For the decade ring counters, (as used in the ENIAC, for instance) numbers are usually transmitted as sequences of pulses which are added (counted) serially into each digit position of the ring. Subtraction is accomplished usually by adding complements, although in the counter-type adders, subtraction can be done by counting backwards (e.g. a desk calculator).

II F. Multiplication and Other Processes.

By the definition of the process, multiplication is performed by adding the multiplicand to itself the number of times specified by the multiplier. As mechanized for a desk calculator, this process of repeated addition is normally modified so that the tens, hundreds, thousands, and other powers of ten appearing in the multiplier are dealt with by shifting the multiplicand one column to the left for each power of ten (note that this is exactly the technique taught in the grade schools which permits a student to multiply by a combined application of a short multiplication table, simple addition, shifting, and accumulation of the final sum product).

For example, given a multiplicand of 21 and a multiplier of 39, the multiplication is performed in steps, starting by adding the multiplicand, 21, into an accumulator 9 times, because the last digit of the multiplier is 9, to give 189, the first partial result. The multiplier then is shifted left (multiplied by 10) and is added 3 times (second digit of the multiplier) to the 189 already in the accumulator. In a binary system, this additive multiplication is relatively easy to mechanize in that the only possible digits in the multiplier are a one, implying a single addition, or a zero, implying no addition. In binary multiplication, an addition is required on the average for only 1/2 of the multiplier digits. On the other hand, for decimal multiplications, the repetitive addition process requires from zero to nine additions for each digit of the multiplier and is therefore quite long. For this reason, recourse is frequently made to direct use of decimal multiplication tables.

1. Decimal Multiplication

The product of 2 decimal digits will give a two-digit number with a first digit having some value between 0 through 8, so that a table of decimal products is more involved than a table of decimal sums, in which the result has as a first digit either 0 or 1. The two digits resulting from a multiplication of two single digits can be called the left-half and right-half of the product. Then the multiplication of a decimal multiplicand by one digit of a multiplier can be formed by pairing, in a suitable circuit, the multiplier digit in turn with each digit of the multiplicand, each pairing determining a left-half and a right-half digit of the product. The resulting strings of left-half and right-half digits, when added together, form the product of the multiplicand by the one digit of the multiplier. The process must then be repeated once for each digit of the multiplier. Notice that the left-half digits are really only the carries, stored temporarily in a separate register. The process is illustrated in the attached sketch. In the ENIAC, the process is shortened by finding at one time, in a suitably elaborate set of circuits, the entire left-half

and the entire right-half of a one-by-ten digit product. The process is repeated for each multiplier digit, accumulating the sums of all the left-half digits and all the right-half digits. These two are then added to give a final answer.

The multiplication-table method of multiplying may be performed in other ways, by making various compromises between speed and equipment. In one method, a table is formed and stored temporarily by repeated addition to give: 1 x multiplicand, 2 x multiplicand, etc., up through 9 x multiplicand. The digits of the multiplier are then used in turn to select from the temporary table the appropriate multiples to be added into an accumulator. In another method, one stores a table only up through, say, 5 x multiplicand, which requires less time and storage, and completes the table by using such equivalences as 7 x multiplicand =(5 x multiplicand) + (2 x multiplicand), or by using a complement scheme where 7 x multiplicand = (10 - 3) x multiplicand, the factor 10 being only a shift.

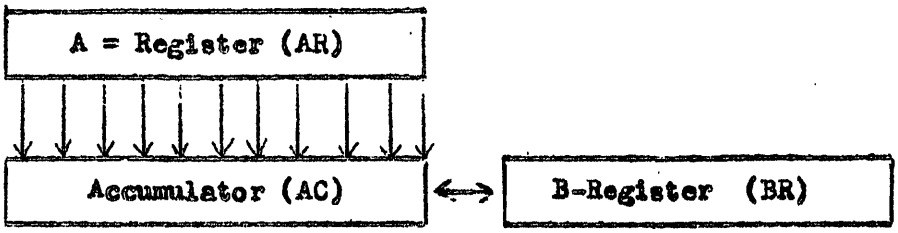
2. Binary Multiplication.

The repeated add-and-shift process of the desk calculators and the use of a one-by-ten digit multiplication table of the ENIAC reduce to an identical process when applied to binary numbers (Adding zero times is the same as multiplying by 0; adding once is the same as multiplying by 1). A more elaborate table may, however, be used to speed up binary multiplication if the binary digits are treated in groups of two, three, four, or more. For example, each threesome of binary digits can be equated to one base-8 or octal digit (e.g. 010 111 001 = 271 [base 8]) and an octal multiplication table may be used to find products, just as a decimal table was used in the methods described above. Such a procedure is quite uncommon in practice.

Most of the computers which use the binary system perform multiplication by means of a modest amount of control equipment coupled to the regular adding or accumulating circuits. While the details of such arithmetic control devices differ between serial and parallel machines, between nines and tens complement representations of negative numbers, and even between one machine and another of almost identical logical structure, the principles used are quite similar and are adequately typified by the circuitry used in Whirlwind.

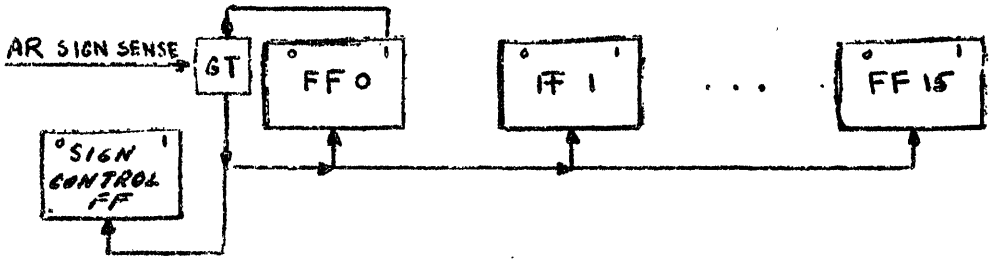
The first problem to be dealt with is to make allowance for the algebraic sign of the multiplier, for the straight add-and-shift algorithm is predicated on a positive multiplier. One way this can be handled is by sensing the sign of the multiplier at the outset, complementing the multiplier if it is negative and storing the sign of the multiplier so that the product, when formed, can be complemented if necessary. In Whirlwind, it is standard practice to carry out all separately-controlled processes (multiplication, division, and the various forms of shifting) entirely on positive numbers, and for this reason the sign of the multiplicand, as well as that of the multiplier, is sensed, stored, and made positive at the beginning of a multiplication order, before the multiplication process has commenced. (Actually, the correct sign of the product, rather than those of the two operands separately, is stored in a "sign-control" flip-flop.)

The arithmetic element of the Whirlwind computer consists primarily of three registers: the Accumulator (AC) which can both add and shift, the B-Register (BR) which is a shifting register that can be considered to be an extension to the right-hand end of the AC, and the A-Register (AR) in which numbers are stored which are to be added or subtracted into the AC.



The multiplication process is carried out in a number of steps, as listed below, assuming initially the multiplier to be in AC and the multiplicand in AR:

1. Sense the sigh of the multiplicand; complement the number if negative; and if negative, complement the sign-control FF (which initially contains 0). To do this, the AR sign sense line, shown below, is pulsed.



A-Register Sign Sensing

2. Sense the sign of the multiplier in AC; complement the number if negative; and if negative, complement the sign-control FF. The circuitry is identical with that shown for AR. The sign-control FF is seen to contain a 0 if the product is positive and a 1 if the product is negative.

multiplicand	sign-control FF	multiplier	sign-control FF	product
+	0	+	0	+
		-	1	-
-	1	+	1	-
		-	0	+

3. Transfer the positive multiplier from AC to BR.

4. If the right-most digit of the multiplier, in digit-column 15 of BR, is a 1, add the multiplicand, in AR, into AC. The mechanics of this is discussed below.

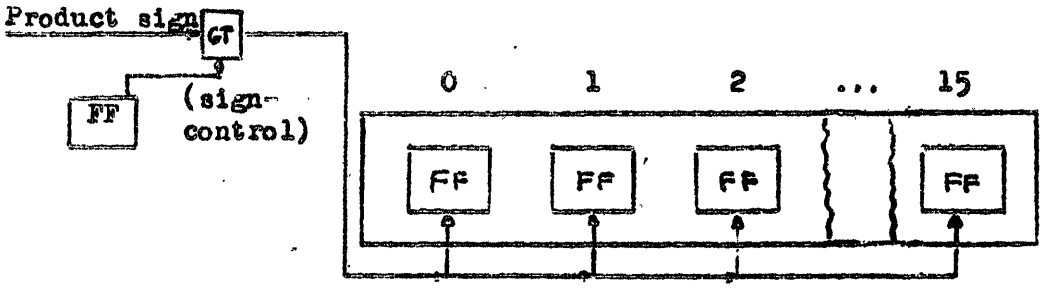
5. Shift the contents of AC and of BR to the right by one digit. The right-most digit in AC moves into the left-most digit column of BR; the right-most digit in BR is lost, being replaced by its neighbor from the left; a zero is put into the left-most column of AC. It is seen, then, that the process of shifting the multiplicand to the left is replaced by the logically-equivalent process of shifting the partial result to the right, just as it is in most desk calculators.

6. Repeat step 4.

7. Repeat step 5.

.
.
.

32. Repeat step 4 for the 15th time.
33. Repeat step 5 for the 15th time.
34. Round off the product (if desired) by adding a contents of the left-most digit of BR into the right-hand end of AC.
35. Compelment the product if the sign-control FF contains a 1. This is done by pulsing the product sign line shown below.



Product Sign Correction in the Accumulator.

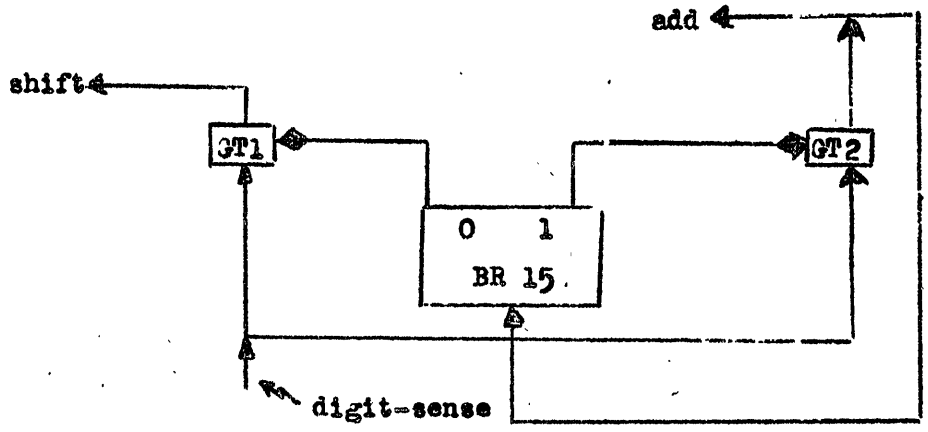
There are several features of the above process which need further discussion

1. Step 4 and the 14 repetitions of it will actually do nothing about half the time, since the multiplier will on the average contain half zeros and half ones.

2. In the Whirlwind accumulator, addition requires a separately-ordered carry, so that each repetition of step 4, or at least each actual addition, would seem to require a separate carry operation.

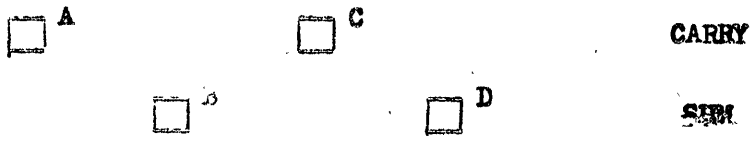
3. During the process, after an addition and before a shift, the number in AC may easily exceed one in magnitude, even though this is outside the allowable range for Whirlwind. The product will never exceed one, however.

The need for performing useless steps, as mentioned in Item 1 above, can be eliminated by using the following circuit. Instead of a sequence of alternate add and shift pulses, a sequence of digit-sense pulses are supplied and these become either add or shift pulses as necessary.



Where BR15 holds a 1, GT2 is open and the sense pulse orders an addition and sets BR15 to 0, so that GT2 will be closed and GT1 will be open for the next sense pulse. When BR15 holds a zero, GT2 is closed, no addition takes place, but a shift is ordered instead. Thus a sequence of sense pulses gives a sequence of alternate add and shift pulses, omitting unnecessary add pulses. A counter is used to count the shift pulses (not the add pulses) and to stop the flow of sense pulses after the 15th shift.

The need for a separate carry pulse after each addition is eliminated in Whirlwind, with a considerable resultant economy in multiplication time, by combining the carry with the process of shifting ordinarily required after each addition. This combined shift and carry process can be understood by considering two of the partial sum flip-flops of AC together with their associated carry flip-flops. In the sketch, A is the carry flip-flop for B, and C for D, and it is desired to use one pulse both to add in any carries and to shift the resultant contents of the AC one to the right (i.e., what B had, D will have).



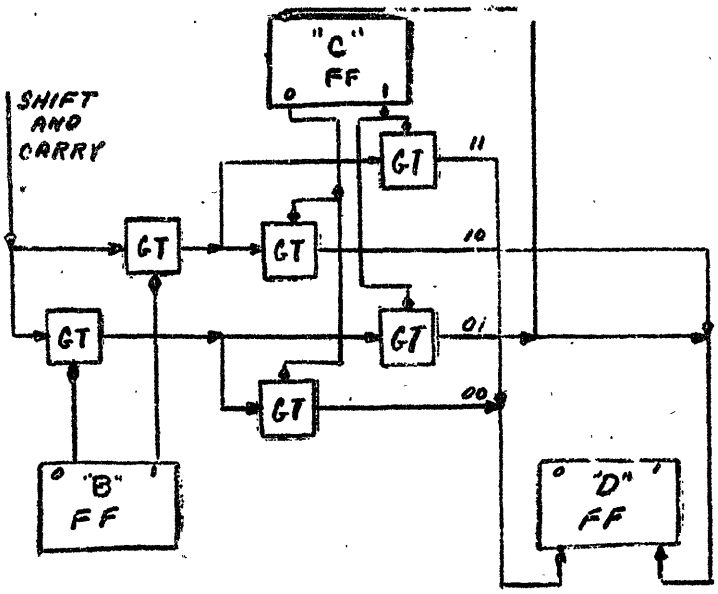
The first column in the following table represents the possible configurations arising from the first step of an addition; the second column the

result of one application of a low-speed, or partial, carry, and the final column the result of an ensuing shift order.

Add B=C	Partial Carry B=A	Shift Right D=C
00	00	00
01	10	10
10	10	10
11	01	01

It is not difficult to arrange a circuit to deduce the final column from the first directly, omitting the second step entirely, so that one pulse suffices for both partial carry and shift. A partial carry after each addition is sufficient in this case, because (as can be seen in the table) after its completion no digit location will ever have 1's in both the sum flip-flop (D) and the associated carry flip-flop (C). Thus the possible addition of the multiplier to the partial product during the next step of the multiplication proceeds without difficulty, resulting at worst in a one in both C and D. After all 15 shift-and-carry steps have been performed, the product remains partly in the sum and partly in the carry FF's, and a complete (high-speed) carry is necessary to complete the job.

A circuit is shown below for performing the combined operations upon receiving a shift and carry pulse. The arrangement of gate tubes (so that a pulse comes out on one of four lines depending on the setting of two FF's) is sometimes called a "whiffletree".



IIF 3. Division, Square-rooting, etc.

By developing and mechanizing reasonably simple and repetitive algorithms, any other arithmetic processes, such as inversion, division, and the taking of roots and of powers, and many special procedures such as finding values in tables, can be added to the bag of tricks automatically performed by a computer. However, any of these processes, and in fact the process of multiplication and even of addition, can also be built up out of simpler abilities and given to the machine in the form of a sequence of instructions rather than in the form of hardware.

The decision as to which processes should be built-in and which may be left to be programmed is as much a matter of taste as of engineering. Built-in operations are performed quicker and require less storage than programmed operations, but they require extra equipment which increases cost and tends to reduce reliability. The wisest choice would seem to be based on building-in only those operations which are likely to be needed in every problem and to program all others, spending the time and money thus saved on increasing the speed and storage capacity of the computer. A small increase in speed and storage capacity can far surpass in value a large amount of built-in equipment for performing special processes.

In any event, all present-day computers have the built-in ability to add, subtract, and multiply. Many have built-in division; some have built-in square-rooters; a few have built-in table hook-ups for finding logarithms, exponential and/or trigonometric functions. It hardly seems worthwhile to discuss the mechanization of any of these processes within these rather superficial notes.