COMPANY CONFIDENTIAL

PROJECT STRETCH

FILE MEMO # 58

SUBJECT: High Speed Arithmetic in a Parallel Device

BY: J. Cocke and D. W. Sweeney

DATE: February 11, 1957

HIGH SPEED ARITHMETIC IN A PARALLEL DEVICE

When a machine performs one of the operations a + b, a - b, a x b or a ÷ b, the actual time to obtain the result will depend upon the numbers a and b. Machines in the past have made only slight use of the special properties of the numbers to be combined and have, in general, waited for the result, a time equal to the time necessary to combine the two numbers which take the longest. In this memo, it will be shown that the average time to combine two numbers is much less than the longest time. A general discussion of the various procedures which may be used to capitalize on the special properties of the numbers will be given. It will be shown how one may shorten the time for arithmetic operations without the need for faster components, however, necessitating additional components.

Addition

In addition, the principal cause for delay is the time taken to propagate carries. The Bureau of Standards has devised circuitry which will propagate carries faster, however, the hardware necessary for this device is enormous, and it will not be considered further. It has been noted (C. F. Bruks, Von Neumann, Goldstine - Logical Design of a Digital Computer, Vol. 1, Page 11) that the average number of carries, when adding two binary numbers of length n, is less than $\log_2 n$. Thus if a device were known which would inform the machine that the carries were

finished, an improvement in speed of $\frac{n}{\log_2 n}$ could be expected as compared to the conventional method. Such devices have been conceived and will be discussed. First, let us consider the general situation and show why the average carry time is small. Consider two random numbers each n characters long and of radix R.

The following table shows the ways in which a carry may originate.

a	b
1	R-1
2	R-1, R-2
3	R-1, R-2, R-3
.	. . .
.	. . .
.	
R-1	R-1, R-2, R-3 . . . 1

From this it is easy to see that a carry may originate in $\frac{R(R-1)}{2}$ ways. There are R^2 possible situations and thus the probability that a carry originate is $\frac{R-1}{2R}$. Now the value of the sum of the two must be exactly R-1 for a carry to propagate. Thus there are 0 + R - 1, 1 + R - 2, .., R - 1 + 1 or R ways that can happen so the probability that a carry be propagated is $\frac{1}{R}$. Now let $P_n(V)$ be the probability that a word of length n have a carry V. Then the probability that there be a carry of exactly V is $P_n(V) - P_n(V+1)$, and so the expected number of carries E_n is

$$\sum_{V=1}^{n} V\left(P_n(V) - P_n(V+1)\right) = E_n$$

Now suppose we have a word of length (n-1) and we know $P_{n-1}(V)$ and suppose we add a single symbol to the word, then $P_n(V)$ is the sum of two mutually exclusive probabilities. First, the probability that in the n-1 length word there is already a carry of length greater than or equal to V, $P_{n-1},(V)$, plus the probability that there is not a carry of length greater than length or equal V in the n-1 symbols but adding the extra symbol causes a carry of length V, $(1 - P_{n-V}(V))\frac{1}{R^{V-1}}\frac{R-1}{2R}$. Thus we see that $P_n(V) = P_{n-1}(V) + (1 - P_{n-V}(V))\frac{.1}{R^{V-1}}\frac{.R-1}{2R}$ so we see that

$$((P_n(V) - P_{n-1}(V)) \leq \frac{R-1}{2 \cdot R^V}$$

or

$$\sum_{i=V}^{n} (P_i(V) - P_{i-1}(V)) = P_n(V) \leq \sum_{i=V}^{n} \frac{R-1}{2RV}$$

$$\text{i.e. } P_n(V) \leq \underset{MIN}{\cancel{Max}} \left(1, \frac{(n - V + 1)(R-1)}{2 R^V}\right)$$

Now suppose that $R^k \leq n \leq R^k + 1$. We have the expected number of carries $E_n = \sum_{V=1}^{n} P_n(V)$

$$\leq \sum_{V=1}^{K-1} P_n(V) + \sum_{V=k}^{n} P_n(V)$$

$$\leq k - 1 + \sum_{r=k}^{n} \frac{(n - V + 1)\, R-1}{2 R^V} =$$

$$(1)\quad k - 1 + 1/2 \frac{(n - k + 1)\ R - (n - k + 2)}{R^{k-1}(R-1)} + \frac{1}{R^n (R-1)}$$

For R = 2 it is easy to see that $E_n \leq k-1 + \frac{n}{2^k} \leq \log_2 n$.

Using the formula (1) for E_n with n = 48 and R = 2 we obtain a value of 5.35, whereas $\log_2 48 = 5.57$ which shows that the ratio of improvement in speed is better than $\frac{n}{\log_2 n}$ as stated in the first paragraph of this section. For a radix 4 and n = 24, we obtain from formula (1) $E_n = 3.84$. Thus if a circuit which would obtain double carries as fast as single carries are obtained, one could achieve an increase in speed. A circuit for doing this is known but it may be pointed out that the increase in speed does not justify the hardware.

The first device known which enables one to stop an addition in a static machine was described in the I. R. E. Transactions, December 1955 and was due to Gilchrist, Pomerene and Wong. The principle is the following.

At each stage in the final answer, there must be a carry or no carry. The conditions which determine this are shown in the following truth table:

	Carry In	A	B	Carry Out
Carry out determined by	0	0	1	0
carry in.	0	1	0	0
	1	0	1	1
	1	1	0	1
Carry out determined by	0	0	0	0
addends.	1	0	0	0
	0	1	1	1
	1	1	1	1

Let $\overline{11}$ $\{\overline{00}\}$ stand for not both A and B 1 $\{0\}$. Then associate the following circuit with the adder.

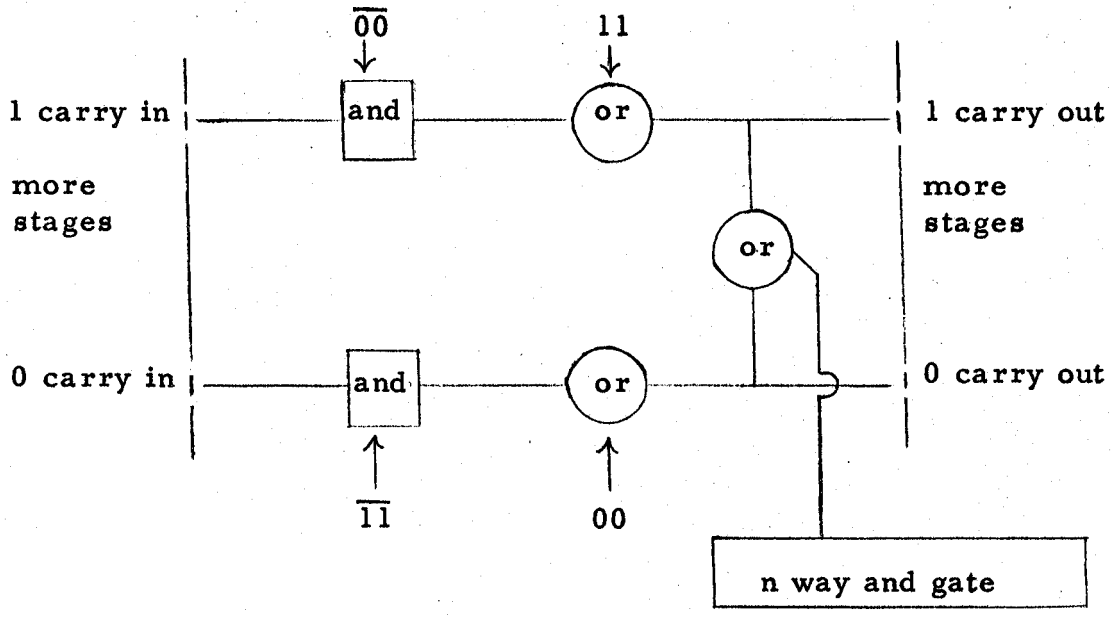

Fig. 1

A comparison with the truth table shows that this circuit will give the proper condition at each stage when there is either a carry of zero or one at each stage. When this has happened, there can be no further carries and the 48 way and gate will signal that the sum is formed. The expected number of carries is somewhat higher since we are propagating carries of 0 and 1. Thus the probability of originating a carry is 1/2 instead of 1/4 and using formula (1), and multiplying the second term by 2 we obtain a value of 6.7. Another method conceived by J. Pomerene while at I. B. M. which also determines the end of carries and provides a check for the correctness of the result is as follows:

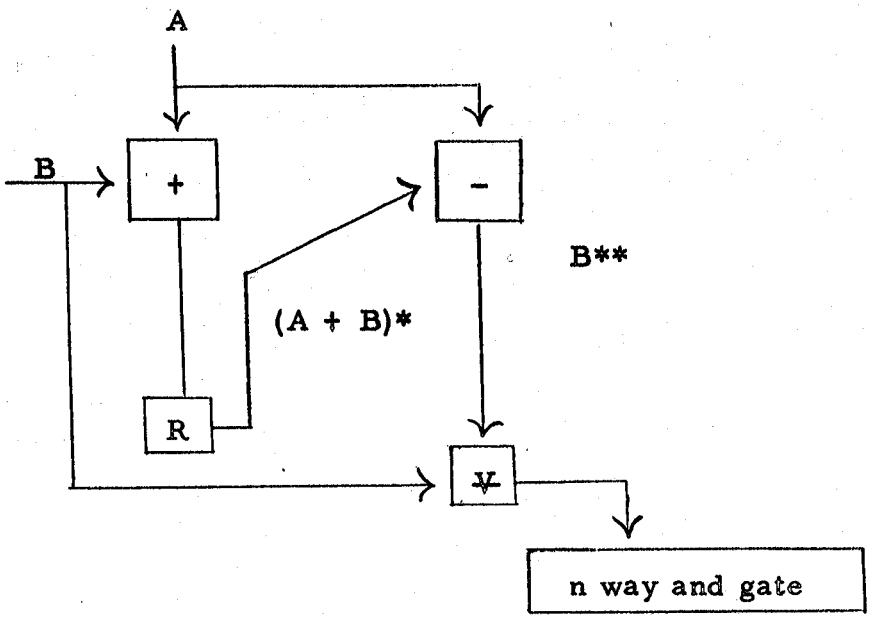

Fig. 2

The numbers A and B are put into the adder. This immediately forms the partial sum, i.e. the sum without all carries propagated. Call this sum (A + B)*. This goes into a register R and then A is subtracted from (A + B)* to form the partial difference of (A + B)* and A. Call this result B**.

B** is compared with B. If they agree, all carries have been completed and the correct answer is in register R. There is a time lag due to the fact that the subtract is started after the add and it is this time lag that causes B & B** not to agree until the operation is complete. However, if the subtract were to work faster than the add, it is possible that B & B** agree and cause the machine to think that (A + B)* was correct too soon. This probability can be lessened by increasing the time lag. Whether or not this would cause the operation to be too slow must be evaluated.

Another method which was an outgrowth of the above idea and provides a carry completion signal, as well as checking for error, was conceived by M. Marshall is as follows:

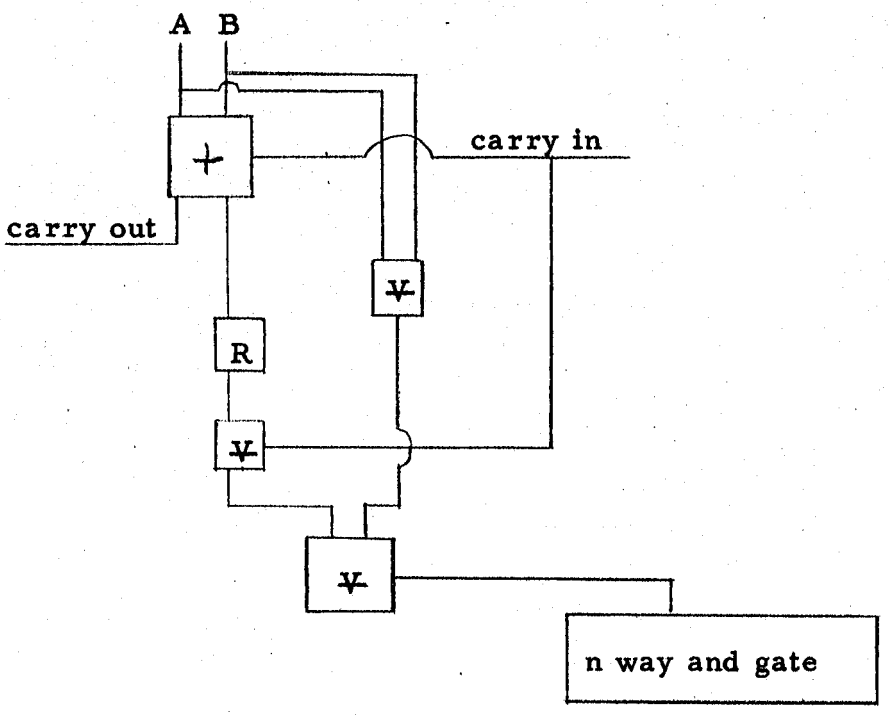

Fig. 3

The output of the adder is A ⊻ B ⊻ C_{in}. If this output is put into on ⊻ with C_{in} we obtain (A ⊻ B ⊻ C_{in}) ⊻ C_{in} = A ⊻ B, thus if this is compared with A ⊻ B the results should match. However, (A ⊻ B ⊻ C_{in}) ⊻ C_{in} takes 2 delays to be formed and so this does not compare with A ⊻ B until the carry has had time to change the output. This means that until all carries have settled out, there is at least one stage of the adder

which is not in agreement. This system is not subject to the possibility that successive stages may operate somewhat slower than others and thereby cause error due to a cumulative effect as the previous scheme was. However, the slow operation of an $\forall$ can cause the machine to think that the answer was correct and thereby cause error. It might be noted that the checking device in this scheme is somewhat simpler than the adder and thereby reduce the hardware requirements somewhat. Also, this method is faster than previous methods by a small amount due to the fact that it does not consider the propagation of zero and one carries but merely one carries, nor does it have to wait for subtraction to be completed.

A fourth method which combines in a way the first two methods of J. Pomerene is as follows:

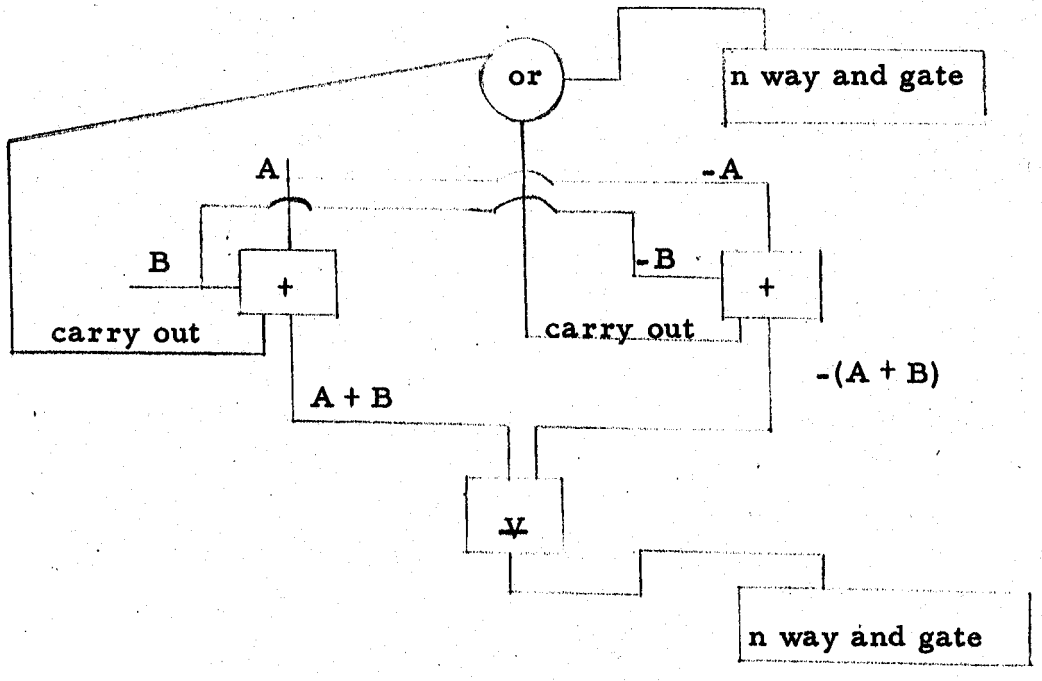

Fig. 4

A is added to B and -A is added to -B. Since A + B is the complement of -(A + B) the carries of zero will occur as carries of one in the complementary adder. Thus, putting the carries of one from each adder into an or circuit will give a signal at each input of the 48 way and circuit when all carries have been completed. Comparison of the outputs of the two adders gives checking. This method takes the time to propagate both zero and one carries, i.e. an average of 6.7 carries, but has the advantage that both adds start at the same time so there is no initial delay. Further checking is absolute in the sense that the speed at which either device operates does not enter the picture and the machine will not think that it has the correct answer until both adds are finished.

This method may also be of value in checking multiply, and this will be discussed in the section on multiplication.

Subtraction

Subtraction and addition are completely symmetric and all remarks of the above are appropriate.

Multiplication

There are numerous methods by which the multiplication speed can be increased. One is to have a built-in multiplication table and numerous adders. This seems to take too much hardware and the method to be described here is a way to speed up the multiply without necessitating

excessive hardware. The method is well known but will be described here for completeness. Also, an analysis of the number of necessary steps will be made. The method depends on doing three things. First, merely shift the multiplicand at each appearance of a zero in the multiplier. Second, for numbers like 7, 15, 31, etc. (i.e. any string of ones) in the multiplier instead of multiplying by seven etc., we multiply by eight etc., and subtract one times the multiplicand. That is, instead of adding at each one in a string of ones, we subtract one times the multiplicand at the beginning of the string and skip over until we find a zero and add in the multiplicand at that place. Third, instead of propagating the carries at each stage, the carries should be saved and added into the next succeeding stage until at the end they are added into the total and allowed to continue until the carry complete symbol is received. This is a rough description of the method. A precise description follows with a derivation of the number of steps required. Since it is possible to both add and subtract from the partial product and thus a sequence of n ones in the multiplier, such as $2^{-(i+1)} + 2^{-(i+2)} + \ldots + 2^{(i+n)}$ is equivalent to $2^{-i} - 2^{-(i+n)}$. N additions can in this way be replaced by one addition and one subtraction.

The sensing of two multiplier digits at a time and an indicator to tell if an add or subtract has just been performed, will determine the procedure.

The following table gives the rules.

	g_i	g_{i+1}	
add just performed	0	0	Shift right
	0	1	Add and shift right
	1	0	Shift right
	1	1	Subtract and shift right
subtract just performed	0	0	Add and shift right
	0	1	Shift right
	1	0	Subtract and shift right
	1	1	Shift right

The number of additions which will be required can be seen as follows. Suppose that one has an n bit word. Consider the digits three at a time looking at the middle digit. No add will be performed in the following cases: 000, 001, 100, 101,and 111. Adds will usually be performed in the cases 010, 011, and 110. The end digits must be handled separately since there is 1/2 a chance that an add be made at the beginning and end. Thus, the expected number of adds required is almost 3/8 $(n - 2) + 1/2 + 1/2 = 3/8\ n + 1/4$. However, looking at the cases 010, 011, and 110 we have counted too many adds since the cases ...xxx11011, xxx... ...xxx1101011 xxx...xxx110101011 xxx...,etc. require one less add than would be indicated by the above calculation. The length of each of

the sequences is odd, so suppose they are of length $2k + 1$ $k = 2, 3, \ldots$ $\frac{n}{2}$. Then the number of ways that one of these sequences can occur is $n - 2k$ and the probability of a single occurrrence is $\frac{1}{2^{2k+1}}$. Therefore, the expected number of these types of sequences is

$$\sum_{k=2}^{\left[\frac{n}{2}\right]} \frac{n-2k}{2^{2k+1}} = \frac{n}{24} - \frac{7}{36} - \frac{1}{9 \cdot 2^n}\begin{cases}4 & n \text{ even} \\ 5 & n \text{ odd}\end{cases}$$

Since for n large $\frac{1}{2^n}$ is negligible, one can say the expected number of adds is $3/8\ n - \frac{n}{24} + 7/36 + 1/4 = 1/3\ n + 4/9$. Thus, if a method for fast shifting is devised and a circuit which senses two digits at a time and makes the add or subtract decision, it will be possible by the use of two registers (one continuing the number being added or subtracted and the other receiving the shifted quantity) to reduce the multiply time for an n digit quantity to the time necessary for $1/3\ n + 4/9$ adds. Further, if the carries are saved at each stage, the time will be reduced since only in the last add will it be necessary to assimilate the carries. Using the system for adding shown in Fig. 4, it would be possible to add the same number on each side and check the carries and the add at each stage, and thus provide a check at each step in the process. The circuitry necessary for this type of multiplication has been considered by Mr. Pomerene and it looks reasonably simple as compared to other methods of fast multiplication.

Division

If the process of division is examined closely, it is found that division is not the opposite of multiplication in the sense of successive subtractions instead of successive additions. Instead, division is a highly mechanized iterative procedure i.e., each succeeding stage depends upon the previous step rather than being independent of it as is the case in multiply. The process is (given a divisor, d, and a dividend, D) to form successive quotient digits of the type:

$$dq_1 + r_1 = D$$

$$dq_2 + r_2 = r_1$$

$$dq_3 + r_3 = r_2$$

etc.

The q_i at each step is chosen such that $\frac{|r_i|}{r_i} = \frac{|r_{i-1}|}{r_{i-1}}$ and $0 \leq r_i \leq r_{i-1}$ i.e., if the remainder changes sign choose a q_i one less than the trial q_i. Ordinarily, the q_i chosen is one digit and if we are using a radix R, $0 \leq q_i < R$. Obviously, one would like to be able to choose a q_i of one or more digits such that r_i reduces to a minimum value after each step. A method has been invented such that if the restriction $\frac{|r_i|}{r_i} = \frac{|r_{i-1}|}{r_{i-1}}$ is removed, it is possible to predict a q_i by examining r_{i-1} so that the r_i is minimized. This method will predict a possible string of digits composed of all 0's or all (R-1)'s. It is therefore of limited use for any radix except R = 2. It can be shown that if R = 2 that the expected number of digits in each

q_i approaches 8/3 digits rather than the usual 1 digit. The method depends upon always having the first digit of the divisor a 1. This is precisely the condition present when performing normalized floating point division. Ordinary fixed point division or unnormalized floating point division can be accomplished by pre-divide left shift of the divisor and dividend,and post-divide right shift of the divisor and remainder.

The two's complement of the divisor is added to the dividend or partial remainder if it is a true value. The true value of the divisor is added to the partial remainder if the partial remainder is a complement value.

If the partial remainder is true, the next quotient digits can be determined from this partial remainder to be the one represented by the carry out of the high order position and all succeeding zeros. If the partial remainder is complement,the next quotient digits can be determined from the partial remainder to be all of the leading ones except the last which must be inverted to a zero.

It will be noted therefore that each true (complement) partial remainder has a leading one (zero) after the predicted quotient digits are removed. Therefore, the next cycle will always produce at least one quotient digit. The probability of producing one or more quotient digits is:

No. of Digits	Probability
1	1/4
2	5/16
3	13/64
4	29/256

The probability of developing one more digit can be gotten from the last probability by doubling the numerator and adding three and by multiplying the denominator by four. The probability of developing all quotient digits in one stage is $1-\sum_{i=1}^{n-1} P_i$. For example, if we assume a four digit divisor, an eight digit dividend, and wish to develop a four digit quotient we have the following probabilities:

No. of Digits	Probability
1	1/4
2	5/16
3	13/64
4	15/64

Now, if we evaluate these probabilities to get the expected number of digits at each stage, the formula is $E_n = \sum_{k=1}^{n} k P_k$ and we get the following table:

n	E_n	
1	1	= 1.000
2	$E_1 + \frac{2^2-1}{2^2}$	= 1.750
3	$E_2 + \frac{2^3-1}{2^4}$	= 2.187
4	$E_3 + \frac{2^4-1}{2^6}$	= 2.421
10	$E_{10} + \frac{2^{11}-1}{2^{22}}$	= 2.662
∞	8/3	= 2.667

Therefore, with respect to any divisor length over 10, we can expect to develop 8/3 quotient digits per additions. For a 48 bit divisor length, we can expect an average of 18 additions to develop a 48 bit quotient.

This method is certainly the simplest to conceive and construct as it is never necessary to generate quotient digits. Instead the appropriate high order bits of the partial remainder are the quotient.

A modification of this method in which the three high order digits of the divisor and dividend are compared after each stage and the divisor is entered as double, the same, or one-half its value (rather than always the same) will reduce the probability of developing only one quotient digit from 1/4 to 1/8 and distribute this 1/8 over the probabilities for developing more than one quotient digit. This will raise the average expected number of quotient digits to about three, but this method will require more hardware not only to do the comparisons but in developing the quotient digits themselves.

Obviously, this modification can be extended to comparing more digits and introducing other multiples of the divisor as trial, but the gain appears to be slight for the increased complexity of the system.

A Discussion of Speed

The basic add time can be said to be composed of three parts. First, the time lag from the time the signal gates the registers, which contain the two numbers, to the time sufficient voltage is on the logical circuit to make it function. Second, the time for addition to take place. Third, the time for the register containing the answer to be set after the addition is finished. The first and third of these times are fixed, but the second varies according to the number of carries. The present data flow model which is six stages long can be used to estimate approximately the average add time for a 48 bit word since its longest carry of six stages is about the average number of carries for a 48 bit word. (This is an approximate estimate since the two fixed times use a higher percent of the time for adds in the short adds than they do for the longer ones. However, this should be good to 10 milli microseconds one way or the other.)

The present model, using the circuit shown in Fig. 2 not using drift transistors, will perform an add which requires that a carry be propagated six stages in 180 milli microseconds. With drift transistors, we should hope to cut this time in half, thus giving us an average fixed point add time of less than . 1 microsecond. Hence the floating point add speed will largely

be determined by the shifting speed and the speed of the various logical circuits used in determining the amount of shift.

Since shifting can be carried on in parallel with addition by the use of two registers, the multiplication speed is a function of the shift speed or the add speed. If the shift time is greater than the add time, then the time to multiply will be roughly 16 shift times. If the add time is greater than the shift time, the time will be 16 add times. If the add time is much greater than the shift time (as it is hoped), then by saving carries and shifting during time logical add is formed, the time will be 16 logical add times plus one carry assimulation time. The logical add time requires the first and third fixed times mentioned before, but the second time is always a minimum and based on evidence from the data flow model the sum of the two fixed times plus a minimum add time should be about half the average add time. Thus if shifting is fast enough we can expect a multiply time of about 1 microsecond.

In division, the picture is not quite so bright. Based on methods so far proposed, division should require approximately 20 add times to develop the quotient and proper remainder with each add requiring a carry assimulation. Further shifting time cannot be done in parallel with the add time, since one does not know how far to shift until the remainder appears. Thus using the method of the last section and expected speeds, it would seem that one would be provided with a divide speed of about

3.0 microseconds. Several more complex methods for improving this speed have been considered, but so far the additional complexity does not seem to be justified in the light of the small gain in speed that these methods provide.