COMPANY CONFIDENTIAL

John Griffith

PROJECT STRETCH
FILE MEMO #30
SUBJECT: Multiple Precision Addition and Subtraction
BY: W. Wolensky
DATE: April 9, 1956

The problem of Multiple Precision is one of requiring significant digits in excess of word size capacity in a given machine. A given word "A" if large enough can be divided into sections, and each section can be represented by a machine word. $A = A_1^x + A_2^{x-48} + A_3^{x-96}$ ------ or more simply $A = A_1 + A_2 + A_3$ ---- to the degree of precision required.

Arithmetic operations involving multiple precision numbers present machine handling problems unique to a machine because each machine word is only a portion of a complete logical word. Some of the problems encountered are herein defined, and a method of handling multiple precision add and subtract is illustrated.

A machine word of 60 bits is composed of an exponent of 10 bits, a fraction of 48 bits, one sign bit, and one overflow indication bit. In multiple precision problems the exponent is only significant in A_1, the first and most significant of the word sections. The exponents of A_2 and A_3 are progressively -48 from the preceding word section. It is planned that the exponent will always be positive because it is possible to realize all numeric values with a positive exponent and a fraction that can be either positive or negative.

For the purpose of this memo, zero is defined as an exponent of all zeros and a fraction of all zeros. In the case of multiple precision numbers, the fraction portions of all the word sections must be zero as well as the exponent of the most significant word section. It is evident that an increase in the exponent is the same as shifting the decimal point to the right. In the event that the exponent is increased beyond its capacity of 10 bits, an exponent overflow indication is provided. Should the exponent be reduced to the point where it will try to assume negative values, the exponent and the fraction will be set to zero.

Automatic single precision floating point operations cannot be applied directly to multiple precision operations. Floating point methods can be programmed for the particular situation at hand in a step by step procedure, likewise automatic normalizing procedures should not be applied to multiple precision applications. Normalization should be done in a programmed, step by step procedure only if the exponent is not all zeros.

Exponent differences between words involved in multiple precision addition and subtraction are very easily reconciled. The multiple precision word carries its exponent with its most signigicant word section (all fraction parts carry their own sign indication). Regardless of the value of the exponent for a given multiple precision word, the value of the exponent associated with the second word section is 48 less than the value of the preceding word section. (The difference is determined by the number of bits in the fraction part of each word section.) The value of the exponent associated with the third word section is 48 less than the exponent of the second word section and 96 less than the exponent of the first word section, etc.

When two words are brought together for addition or subtraction, one of the first operations that will be normally performed is the determination of the difference between the exponents. If the difference between the exponents is less than 48, all sections of the lesser valued word is shifted to make the exponents equal and the process of combining $A_1 + B_1$, $A_2 + B_2$, $A_3 + B_3$, etc. is continued. Because of the shifting of the lesser word, some of its least significant bits will not be involved in the operation.

A difference in exponent values that ranges between 48 and 96 implies that the lesser word has no counterpart to the most significant section of the greater word. In this case 48 must be subtracted from the exponent difference and the remainder indicates the amount of shifting necessary to properly align B_1 so that it is compatible to A_2. An exponent difference between 96 and 144 indicates that the most significant portion of the lesser multiple precision word will align itself to the third most significant portion of the greater multiple precision word, offset or shifted by exponent difference minus 96.

An exponent difference greater than the degree of precision times the exponent differential ($3 \times 48 = 144$) indicates that no addition or subtraction operation is to be performed. The larger of the two words is the result of any considered addition or subtraction.

One means of normalizing multiple precision numbers is similar to that used in the 702 and 705. A special Normalize and Transfer instruction is provided which causes a transfer when an insignificant zero is removed from the fraction part of a machine word. If there is no insignificant zero to be removed, no transfer is affected. The instruction permits removal

of any or all insignificant zeros and readily permits counting for exponent modification. It is presupposed that a single instruction will be provided which will fully normalize the word and the number of zeros eliminated will be available in the exponent counter.

Addition and subtraction are performed in the computer by an adder. Subtract can be defined as changing the sign of the second word and adding. Addition is split into two types of operation, addition with like signs and addition with unlike signs. Addition with like signs is a standard algebraic addition whereas addition with unlike signs is a process of complementing the second word and adding it to the first.

To facilitate multiple precision operations, several special instructions and indications are suggested.

a) Add, if unlike signs suppress recomplementing, impose initial carry in.

b) Add, if unlike signs suppress recomplementing, suppress initial carry in.

c) Add, if unlike signs permit recomplement, permit initial carry in.

d) Add, if like signs force an initial carry in. (instruction **c** is the standard add instruction, modifications a, b, and **d** can be realized by coding specifically designated bits in the instruction code)

e) Carry indication: if addition results in the fraction overflowing the left most significant position, a carry indication is provided.

f) Load complement: the word specified is loaded in complement form into the accumulator.

Multiple Precision Addition (like signs) Given: Triple precision words +A and +B where $A > B$, therefore $A_1 > B_1$, and $A_2 < B_2$, $A_3 > B_3$ (see Fig. 1). A typical program for performing multiple precision add with like signs is presented ; detail is provided only in the areas that

specifically relate to multiple precision.

1. Test the signs of A and B; if the signs are alike and addition is to be performed, or if signs are unlike and subtraction is to be performed continue to step 2. (A and B both +, addition is to be performed).

2. Subtract the exponents of A_1 and B_1, determine the difference if any, and which of the two is larger. (For this example let the exponent of A_1 be larger than that of B_1 by one).

3. If necessary, modify index registers to properly locate the word sections to be involved in the operation.

4. Create B^1 which is B with the same exponent as A.

   4a. Load B_1 into accumulator shift right one position, store B'_1 in memory.

   4b. Ring shift accumulator left 49 positions (in the illustration of figure 1 a four bit fraction is shown, therefore ring shift accumulator left five positions).

   4c. Load B_2 into accumulator ring shift accumulator right one position, store B'_2 in memory.

   4d. Ring shift accumulator left five positions.

   4e. Load B'_3 into accumulator ring shift accumulator right one position.

5. With B'_3 in accumulator Add A_3 to contents of accumulator.

   5a. Store result in accumulator as Sum S_3 in memory.

   5b. Test for carry out, if carry out exists do special operation, if no carry out exists do 6 (no carry out in illustration).

6. Load B_2 into accumulator, add A_2.

   6a. Store result in accumulator as Sum S_2 in memory.

   6b. Test for carry out, if carry out exists do 7, if no carry out exists skip 7 do 8. (carry out is present)

7. Load B_1' into accumulator Add A_1 with special instruction imposing an initail carry in. Transfer to 9.

8. Load B_1' into accumulator add A_1.

9. Test for carry out, if no carry out store S_1 in memory and problem is finished, if carry out exists do 10. (carry out exists).

10. Add one to exponent of S_1, test for exponent overflow, if exponent overflow stop machine, if no exponent overflow do 11. (no exponent overflow exists).

11. Shift accumulator right (this is not a ring shift) one position because carry out digit is one position to left of normal fraction store S_1 as final answer.

   11a. Ring shift accumulator left five positions, Load S_2, ring shift accumulator right one position, store S_2.

   11b. Ring shift accumulator left five positions, load S_3, ring shift accumulator right one position, store S_3, problem completed.

Multiple Precision Addition (unlike signs)

Given: Triple precision words -A and +B where $A > B$, therefore, $A_1 > B_1$, and $A_2 < B_2$, $A_3 > B_3$ (see figure 2).

A typical program for performing multiple precision subtract (or add with unlike signs) is presented, detail is provided only in the areas that specifically relate to multiple precision.

1. Test the signs of A and B, if signs are unlike and addition is to be performed, or if signs are alike and subtraction is to be performed continue to step 2. (A is minus, B is plus and Addition is to be performed.)

2. Subtract the exponents of A_1 and B_1, determine the difference if any, and which of the two is larger. (exponents of A_1 and B_1 are equal therefore, no shifting or exponent manifpulation is required.)

3. Adjust index registers to properly sequence the word sections for the operations to be performed.

4. Load B_3 into accumulator. Complement and add A_3 to contents of accumulator, suppress recomplementing impose initial carry in.

   4a. Test for carry out; if a carry out does not exist continue to 4b; if a carry out does exist follow different procedure- (illustration of figure 2 does not have a carry out at this point.)

   4b. Store contents of accumulator in memory at S_3.

5. Load B_2 into accumulator. Complement and Add A_2 to contents of accumulator, suppress recomplementing and suppress initial carry in (Compare with step 4, initial carry in is suppressed here because there was no carry out from addition of B_3 and A_3)

   5a. Test for carry out; if carry out exists continue to 5b, if carry out does not exist follow different procedure which adds A_1 to B_1 and suppresses initial carry in. (illustration has a carry out at this point.)

   5b. Store contents of accumulator in memory at S_2.

6. Load B_1 into accumulator. Complement and add A_1 to contents of accumulator, suppress recomplementing, and imposing initial carry in. (initial carry in provided because preceeding operation contained a carry out.)

6a. Test for carry out; if carry out exists store accumulator into memory as S_1 and transfer to 8, if no carry out exists do 6b. (no carry out exists)

6b. Complement contents of accumulator. Place proper exponent with S_1. Change sign of S_1 to minus.

6c. Load S_2 into accumulator. Complement contents of accumulator store S_2 in memory, make sign minus.

7. Load S_3 into accumulator. Complement contents of accumulator. Add one to accumulator contents. Store in memory as S_3 change sign to minus.

7a. Test accumulator for carry out if no carry out exists transfer to step 8, if a carry out exists do 7b.

7b. Add one to S_2 and check for overflow, if no overflow transfer to step 8,

7c. Add one to S_1, and check for overflow, if no overflow transfer to step 8, if overflow exists stop machine.

8. This is an optional procedure provided to normalize the multiple precision word if desired.

8a. Load S_1 into accumulator. Normalize and transfer,if a zero is removed a transfer is effected to another protion of the program where the number of zeros removed is counted and control is then returned to 8a. If no zero is removed control is switched to 8b.

8b. Test zero removal count if count is equal to zero, no normalizing is to be done, if count is greater than zero do 9.

9. Load S3 into accumulator. Shift accumulator left number of places equal to zero removal count (one in this case per figure 2) Store S_3 in memory.

9a. Shift accumulator right five positions. Load S_2 into accumulator. Shift accumulator left one position. Store S_2 in memory.

9b. Shift accumulator right five positions. Load S_1 into accumulator. Shift accumulator left one position. Subtract zero removal count from S_1 exponent. S_1 is now in final form store S_1 in memory, solution is complete.

The illustrations of figure 1 and figure 2 with their accompanying solution procedures outline or define practically all the possible situations that can arise in multiple precision addition and subtraction. The purpose of analyzing the problems and providing a means of solution for multiple precision add and subtract has been completed. It is expected that the ideas presented herein will be reviewed and modified after the basic stretch logic has been more firmly established.

X =	X_1 Exp.	X_1 Fract.	+ Exp.	X_2 Fract.	+ Exp.	X_3 Fract.	
B =	010	1001	—	1100	—	0111	
A =	011	1100	—	1011	—	1011	
B′ =	011	0100	—	1110	—	00111	(4)

$$\begin{array}{rcl} 00111 & = & B'_3 \\ 1011 & = & A_3 \\ \underline{00110} & = & Ci \\ 01110 & = & S_3 \end{array} \quad (5)$$

$$\begin{array}{rcl} 1110 & = & B'_2 \\ 1011 & = & A_2 \\ \text{CARRY OUT} \rightarrow 11100 & = & Ci \\ \underline{11001} & = & S_2 \end{array} \quad (6)$$

$$\begin{array}{rcl} 0100 & = & B'_1 \\ 1100 & = & A_1 \\ \underline{11001} & = & Ci \\ 10001 & = & S_1 \end{array} \quad \begin{array}{l} (7) \\ \\ \\ (8a) \end{array}$$

S = 100	1000	1100	1111	(11)

Figure 1

Multiple Precision Add (like signs)

X	=	X_1 Exp.	+ Fract.	X_2 Exp.	+ Fract.	X_3 Exp.	Fract.
-A	=	010	1110		0011		1010
+B	=	010	1001		1110		0011
$-\overline{A}$	=	010	0001		1100		0101

0011 = B_3
0101 = $\overline{A}_3$
NO CARRY OUT → 01111 = Ci
00001 = S_3

1110 = B_2
1100 = $\overline{A}_2$
CARRY OUT → 11000 = Ci
11010 = S_2

1001 = B_1
0001 = $\overline{A}_1$
00011 = Ci
01011 = S_1

1110
0001
00000 Ci

	- 010	0100	0101	
S =	- 010	0100	0101	1111
S* =	- 001	1000	1011	1110

Figure 2

Multiple Precision Add (unlike Signs)

* normalized