

December 22, 1958

MEMO TO: Erich Bloch
SUBJECT: Sigma Divide Simulation

As a result of consultation with Joseph Stewart and Charles Freiman, it was decided that a simulation of the Sigma Floating Point Divide would provide needed verification of theoretical conclusions. This decision was later supported by E. Bloch and specifications and a general block diagram were drawn up to serve as an outline from which a 704 program might be constructed. On or about July 21, 1958, the 704 programming group was given the responsibility for the writing and obtaining results from the program. Werner Schanzenbach was chosen as the programmer. Concurrent with this effort was a similar one by the author - since this particular program was written and debugged during of-hours and consisted, in part, of different approaches from the Schanzenbach program, it was considered a reasonable effort to pursue and was later used as a check. A full description of the Sigma Divide method is outside the scope of this report and it is assumed the reader has a general familiarity with the method. The following report is merely a brief explanation of those parts of the simulation programs that would be considered of special interest to those seeking a more complete understanding of the simulation program differences and similarities with the original Sigma Divide Scheme.

H. E. Kolsky

H. Kolsky,

Erich Bloch suggested you might
like to look at this.

F. Bielawa

The information given at the end of this report is that obtained from only a partial reduction of the available data. The original data requirements covered the information needs of both J. R. Stewart and C. Freiman and at this point, only that of interest to J. Stewart has been presented. When further arrangements can be made to discuss the data with C. Freiman, an additional report can be given on the information peculiar to his needs.

Operands

1. Dividend

a. Generation

In the B program, use was made of an existing library sub-routine known as PE RANG. This routine generates pseudo-random numbers modulo N, by an extended "middle of the square" method. It was found that the routine was incapable of producing one of the possible full length (35 bit) numbers, namely $2^{36}-1$, and a modification was made to restore this number. Since a 48 bit dividend was required, two 35 bit random numbers were used together and were carried in two storage cells (A word contained 35 bits, B contained remaining 13 bits.)

The S program made use of similar techniques in the generation of random numbers and construction of operands. A major difference was evident, though, in the actual method of random number generation. While the library sub-routine used in B program consisted of essentially a "middle of the squares" method, it was modified by the random selection

of one of 16 preset random multipliers which, during each routine cycle, operated on the previously generated random number - the S program only used the "middle of the square of the previously generated random number. It should be noted that both of the generators are assumed non-degenerate and non-cyclic for at least the first one million numbers.

While a thorough test of the randomness of either sub-routine was not attempted, a test of the library sub-routine used in the B program was made of the positional probability of each of the 35 bits of 100,000 numbers. The occurrence of a 1 in each position is listed in included print-out and generally exhibits the expected probability of $1/2$. This, of course, only indicates that of n samples, approximately $n/2$ of them were 1's. This does not imply anything about the joint probability of two or more bits.

b. Normalization:

Normalization in the B program occurs immediately after 70 bits of dividend are formed. The entire 72 bits are shifted left until a 1 is encountered - 0's are inserted in vacated positions on right. At this point, the right most 22 positions (of the 70) are masked out which yields desired 48 bit number. Radix point is considered at the left.

Normalization in the S program is accomplished by simply placing a 1 in the left most position of the 48 bit random dividend. While this method is faster, the above more closely approximates the actual method of normalization.

2. Divisor:

a. Generation & Mask:

In the B program, the same generator as mentioned above is used to generate 70 additional bits of random divisor, right most 22 positions (of the 70) are masked out which yields required 48 bit number. Since it is desired to restrict divisors to specific ranges, the first 5 high order bits are masked and 5 new bits ranging incrementally from 10000 to 11111 are inserted. This then allows for 16 ranges of divisors with 2^{43} possible divisors in each range without repetition.

Generation and masking of the divisor in the S program is accomplished in the same manner as in B program.

b. The $3/4$ divisor multiple required is obtained by adding $(1/4 + 1/2)$ divisor. There is no overflow for maximum value. To obtain $3/2$ divisor, the number obtained for $3/4$ divisor is shifted left 1 position. Overflow may occur and is lost. The $3/4$ divisor multiple in S program is obtained as in B program. The $3/2$ divisor is obtained by shifting left one position. In this case, the overflow is retained.

3. Subtraction

a. In both the S & B programs a complement number is formed and carried as a 2's complement number. This was necessitated by the fact that each "add-complement" operation consisted essentially of a double-precision subtraction and the 704 Add-Subtract operation would, because of its inherent character of representing a number as a sign and

absolute value, present logical problems when performing double precision operations.

b. Detection of Complement Result

In the B program, the radix point for the divisor and dividend are always considered to be to the left of the left most digit. The radix point of the result of an add-type operation is put in same relative position as that of operands. It is characteristic of this program to perform the add-type operation only on the digits to right of radix point. This means that add operations will involve 48 bit operands in the basic scheme and up to 50 bit operands in the Multiple scheme. The overflow bits to the left of radix point, incurred in the case of adding $3/2 \times$ divisor are ignored during an add-type operation. In the Basic scheme, the carry out of the first position to right of radix point indicates sense of result after add operation - 0 indicates complement result; 1 indicates true. In the Multiple scheme, the status of the first position to right of radix point indicates sense of result for add operations (of any of the possible operands) - 0 indicates true result, 1 indicates complement. This latter complementation detect method is well known and was used in this program for the sake of uniform computer functioning and has been verified by successful use in the program.

In the S program, the radix point for the divisor (1X) is considered as two positions to left of highest order bit. The radix point for $3/2$ and $3/4$ divisor as well as that of the result of add-type operations is placed in same relative position to that of 1x divisor. Complement detection is

accomplished in either scheme by means of sensing the carry out of the position left of radix point-where a 1 indicates true result and 0 indicates complement.

Rounding Procedure:

The time both programs were written it was felt that the divide operands should consist of a 48 bit divisor and a 96 bit number or actual dividend which consisted of 48 bits of original dividend and an additional 48 or 49 bits which would produce a rounded quotient. It was decided that initially the divisor would be subtracted from the 48 bit original dividend and the sense of the result would be detected. If this operation produced a true result, the divisor would be placed to right of first partial remainder with no separation, forming 96 bits of actual first partial remainder. If the result was complement the divisor would be placed to right of first partial remainder with a one position separation, this position being made 0 forming 97 bits of actual first partial remainder.

Normal Cycle Operation:

It is beyond the purpose of this report to describe the exact operation of either divide method. It is sufficient to state that both programs, when executing a problem solution by means of the Basic scheme, duplicate the actual machine operations after the initial subtraction; the program continues by shifting across on similar leading bits of partial remainder (normalization) and performing on add-type operation when normalization is complete. This cycle is repeated until problem terminates. The quotient is concurrently generated and its construction is dependent on the type of normalization and the result of

each add operation.

Likewise, in both programs, when executing a problem solution by means of Multiple scheme, actual machine behavior is also duplicated in that after the initial subtraction, the program continues by shifting across on similar leading bits of partial remainder (normalization) and performing an add-type operation when normalization is complete. In this case, the rules of quotient construction and divisor (multiple) selection are those illustrated in chart by J. R. Stewart dated July 28, 1958. The cycle is repeated until problem terminates.

It is of interest to note that the S program considered the Basic scheme normal cycle operation rules as a subset of the Multiple normal cycle operation set.

Termination:

The termination cycle, which is a function of the rounding operation, is handled in the same manner as actual machine termination. The rules for adjusting both quotient and remainder are derived from those stated in Notebook #3514 by J. R. Stewart (page 28). Since the Multiple scheme reverts to Basic-type cycles when 3 or less quotient bits remain to be generated, the termination rules are the same for both schemes. Both programs terminate identically.

Checking:

In order to insure proper operation of simulated divide schemes, a method for checking the problem results, i. e. the quotient, was devised for the B program. After each Basic scheme problem solution, the

generated quotient was multiplied by the original divisor to produce a double length product; the high order 48 bits of which should have been the original 48 bit dividend. This reconstructed dividend was then compared, bit wise, with original dividend and computer was stopped when a dis-similarity was detected. This obviously, neglected to compare the round portion of 96 bit actual dividend but it was felt that this particular check would catch all the probable errors. After a successful run through all problems in the Basic mode, the checking device was then replaced with another that essentially only compared the results, from a given set of operands, that were produced by operation in Basic and Multiple mode. Since the Basic scheme results were proven correct, they served as a standard for Multiple mode results.

The S program undertook to check results also and it is understood that much the same method was used as that given above with the exception that, additionally, the "round" portion of reconstructed actual dividend was checked.

In both programs, it was assumed that the statistical data obtained during the course of problem solutions would be accurate if the above test procedures indicated correct solutions.

Data Collection:

In the B program, data was collected for 1024 problems (a problem is defined as the generation of a 48 bit quotient from two 48 bit operands) in each of the 16 sub-groups of divisors making a total of 16,384 problems.

The S program solved 100 problems per divisor sub-group which yielded a total of 1600 problems. The specific information obtained from both programs was fundamentally the same and is listed below.

1. Problems using the n-th loop.

Since each problem consisted of a number of iterations of the add-shift sequence, it is of interest to know how many problems of each divisor sub-group used a particular iteration during the course of each problem solution. There are 48 possible iterations a problem may take before termination and each problem that requires the use of the n-th iteration, or loop, will add 1 to the sum accumulated for that iteration. By this means, data can be acquired which indicates, of r problems, what fraction, S, used the n-th loop during the process of problem solution.

It can be seen later, that the information given by this data implies the information to be presented by data in (4). Since each data group was generated by an essentially unrelated means, the concurrence of the two provides a check on the inherent data.

2. Sum of Shifts Taken on Iteration .

Part of the loop process consists of simply normalizing (either normal or inverted) the partial remainder and, at the same time, shifting quotient bits into the partial quotient. During each loop, the number of shifts utilized in each normalization is sensed and added to the accumulated sum of shifts for that particular divisor sub group. It is

important to realize that in both programs, the normalization was not limited and the amount of shifting was determined, in all cases except termination, by the leading bits of partial remainder. In the case of termination, normalization ceased at the point of generating the 48th (or 49th, in special case of initial subtraction yielding complement result) bit of quotient. The shifts taken up to this point in this particular loop are then added to the previous sum for that loop.

3. Sum of the (Shifts)² Taken per Iteration

The data gathered for this set is essentially the same as for (1) with the exception that the sum of shift amount squared is accumulated for each iteration (loop) instead of only the shift amount. This data was extracted with the anticipation that it would be useful in determining the standard deviation of the average shift per loop.

4. Problems Terminating in n-th loop

Whenever a problem termination occurred, a 1 was added to the accumulated sum of previous problems that terminated in that particular loop. As mentioned before, this information is implied also in the data taken for (1), but it more clearly presents the distribution of density of termination for each particular loop.

5. Shift Amount Distribution

In order to determine what percentage of the normalization is lost by placing a restriction on the maximum allowable shift amount, data was taken, whereby, for unlimited normalization (as stated) the shift amount taken for a particular loop during a problem solution is

sensed and a 1 is added to the accumulated sum of previous loops requiring a shift of that particular amount. All 48 possible sums, indicating a shift amount of at least 1 to at most 48, are extracted for each divisor sub group. The sum of each sum included in the first n sums compared to the sum of all sums indicated the percentage of cases capable of unrestricted shifting with a maximum shift limited to n.

Processed Data:

Included at the end of this section are two charts giving the computed averages for two basic parameters that are of particular interest. The first is a list of the average loops taken, by both the Basic and Multiple scheme, to terminate an average division problem. For the Multiple scheme with a shift limited to 6x, the average number of loops to terminate is 14.43. For the same conditions, it has been found that 94.3% of the shifts required will be a shift of 6 or less, which demonstrates that a shift limited to 6 is justified, in a theoretical sense, on the basis of its high effectiveness.

Conclusions:

Inherent in a study such as this are always a few doubts as to the usefulness of such inexact techniques to extract, more or less, exact information. The author acknowledges the need for understandable verification of the methods used and a more adequate proof of the present assumptions that the statistics drawn from the programs are valid, will be given in a later report.

Two general aspects of the simulation programs will be investigated - a more extensive analysis must be made of the random number generators in both programs to insure that the random variables are not biased (to an unreasonable degree). Another point that needs to be clarified is whether the method for obtaining the Average Loops to Termination is valid for establishing this average for the total population of random variables - it is not unreasonable to ask if it is valid to attribute to the total population characteristics, evident in a sub group, if the sampling technique is biased. It will be the aim of further studies to uncover what bias might exist in the data gathering methods.

Frank R. Bielawa

AVERAGE LOOPS TO TERMINATION AS FUNCTION OF SHIFT LIMITATION.

MULTIPLE DIVISOR (LEADING BITS)	SUM OF LOOPS PER 100 PERB PER DIVISOR			
	UNLIMITED SHIFT	SHIFT EQUAL TO 6 DEGREES	SHIFT EQUAL TO 5 DEGREES	SHIFT EQUAL TO 4 DEGREES
10000	1367	1447	1526	1694
10001	1367	1457	1540	1709
10010	1306	1376	1447	1629
10011	1333	1398	1466	1640
10100	1380	1464	1532	1687
10101	1315	1392	1479	1643
10110	1325	1372	1455	1622
10111	1321	1391	1473	1659
11000	1324	1409	1496	1671
11001	1326	1412	1473	1664
11010	1285	1372	1448	1633
11011	1363	1433	1497	1656
11100	1410	1475	1539	1702
11101	1435	1501	1593	1749
11110	1523	1593	1658	1817
11111	1534	1596	1647	1798

SUM, Σ	21914	23090	24289	26973
AVE., $\frac{\Sigma}{N} \times 100$	13.696	14.431	15.181	16.858

BASIC DIVISOR (LEADING BITS)	SUM OF LOOPS PER 100 PERB PER DIVISOR			
	UNLIMITED SHIFT	SHIFT EQUAL TO 6 DEGREES	SHIFT EQUAL TO 5 DEGREES	SHIFT EQUAL TO 4 DEGREES
10000	2169	2236	2290	2404
10001	1855	1924	2000	2157
1010	1732	1786	1848	1992
10011	1679	1730	1785	1927
10100	1645	1717	1772	1904
10101	1652	1710	1761	1873
10110	1621	1689	1741	1865
10111	1667	1713	1762	1862
11000	1716	1771	1808	1913
11001	1850	1905	1946	2049
11010	1750	1714	2043	2162
11011	224	2166	2199	2309
11100	2194	2232	2268	2357
11101	2298	2331	2368	2448
11110	2378	2417	2452	2540
11111	2450	2484	2518	2609

SUM, Σ	30490	31795	32561	34265
AVE., $\frac{\Sigma}{N} \times 100$	19.369	19.872	20.351	21.478

(NOTE: ABOVE DATA TAKEN FROM
EXCHANGE EACH PROGRAM)

Dec. 22, 1958

(NOTES: ABOVE DATA TAKEN FROM STANDARD BENTON PROGRAM)

BASIC

151A (SCHEDULE) POSITION	CUMULATIVE SUM OF PROBS. USING SHIFT OF MAX AMOUNT, N (PER 100 PROBS./DIVISOR)							
	n=4	R	n=5	R	n=6	R	n=8	
10000	1956	.902	2050	.905	2103	.970	2169	
10001	1976	.950	1711	.922	1785	.962	1855	
10010	1890	.860	1621	.936	1679	.970	1732	
10011	1447	.862	1574	.937	1628	.970	1679	
10100	1402	.852	1521	.925	1573	.956	1645	
10101	1406	.875	1545	.935	1595	.965	1652	
10110	1410	.865	1525	.935	1574	.965	1621	
10111	1487	.872	1575	.945	1622	.973	1667	
11000	1529	.891	1626	.948	1663	.969	1716	
11001	1671	.903	1758	.950	1796	.971	1850	
11010	1748	.896	1861	.954	1916	.983	1950	
11011	1952	.919	2050	.965	2083	.981	2124	
11100	2037	.928	2121	.967	2156	.983	2194	
11101	2152	.936	2228	.970	2265	.986	2298	
11110	2222	.934	2304	.969	2339	.984	2378	
11111	2297	.938	2383	.973	2416	.986	2450	

SUM Σ	—	14303	—	15,176	—	15,574	—
AVE, $\frac{\Sigma}{16}$	—	.8939	—	.9485	—	.9734	—

R = RATIO

MULTIPLE

CUMULATIVE SUM OF PROBS. USING SHIFT OF MAX AMOUNT, N (PER 100 PROBS./DIVISOR)							
n=4	R	n=5	R	n=6	R	n=8	
1065	.779	1211	.886	1288	.942	1367	
1047	.766	1197	.876	1278	.935	1367	
997	.763	1172	.897	1239	.909	1306	
1044	.783	1204	.903	1268	.951	1333	
1095	.793	1231	.892	1296	.939	1380	
1011	.769	1155	.878	1240	.943	1315	
941	.716	1094	.763	1174	.893	1315	
1009	.764	1172	.887	1253	.949	1321	
1002	.757	1155	.872	1241	.927	1324	
1017	.767	1166	.879	1241	.936	1326	
974	.752	1148	.866	1219	.941	1295	
1089	.799	1231	.903	1294	.949	1363	
1124	.809	1282	.909	1345	.954	1410	
1132	.789	1278	.891	1369	.954	1428	
1242	.815	1339	.912	1453	.954	1523	
1286	.828	1422	.927	1470	.958	1534	

—	12.454	—	14.161	—	15.084	—
—	.7784	—	.8857	—	.9428	—

PERCENTAGE OF CASES WHERE SHIFT REQUIRED IS LESS THAN LIMITATION

P. CALVERT
 APR 22 1951

Section omitted pertains to SAP program written for 704 Computer.

100,000 runs

	Δ
1 50006	+16
2 49942	-58
3 50316	+36
4 50276	+276
5 50031	+81
6 50168	+168
7 49959	-41
8 49809	-91
9 49811	-189
10 50270	+270
11 50324	+324
12 49964	-36
13 50130	+130
14 50133	+133
15 49639	-361
16 49971	-29
17 49913	-87
18 49964	-36
19 50459	+459
20 50071	+71
21 50018	+18
22 49969	-31
23 49938	-62
24 49839	-161
25 50258	+258
26 49746	-254
27 49793	-200
28 49810	-190
29 49696	-304
30 50231	+231
31 49865	-185
32 50062	+62
33 49825	-176
34 49884	-116
35 50172	+172

RANDOM NUMBER GENERATOR

TEST RESULTS

$ERS = 50000/100000$

Buy notes - Sept 19, 1944

(DUH replaced with 665 35 to avoid dropping issues)

B PROGRAM PRINT-OUT

ING BITS DIVISOR EQUAL 16

(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)
1024	2252	7752	0	0	1024	2252	7752	0	0
1024	2232	7614	0	9466	1024	3741	16377	0	951
1024	2346	8658	0	5694	1024	3844	17400	0	2827 2996
1024	2394	8622	0	2965	1024	3832	17322	0	6018 4480
1024	2334	8152	0	1487	1024	3850	17172	0	7924 2560
1024	2344	8142	0	841	1024	3805	16731	0	3881 1327
1024	2463	9011	0	561	1024	3831	17351	0	4433 766
1024	2320	8176	0	275	1024	3853	17255	0	7702 373
1024	2286	7948	0	181	1024	3889	17811	3	4859 217
1024	2324	7904	0	69	1021	3760	16568	22	4129 97
1024	2414	9058	1	33	999	3668	16066	67	49
1023	2292	7850	1	26	932	3209	13251	151	26
1022	2317	8069	7	9	781	2330	8510	260	9
1015	2339	8125	11	5	521	1463	5263	251	6
1004	2287	7975	21	3	270	602	1784	175	5
983	2162	7260	38	0	95	183	473	67	0
945	2081	7119	54	1	28	43	85	26	1
891	2024	6794	88	1	2	2	2	2	1
803	1688	5436	108	1	0	0	0	0	1
695	1479	4833	111	0	0	0	0	0	0
584	1170	3376	118	0	0	0	0	0	0
466	940	2698	108	0	0	0	0	0	0
353	667	1713	109	0	0	0	0	0	0
244	3	1379	108	0	0	0	0	0	0
141	227	477	63	0	0	0	0	0	0
78	139	339	36	0	0	0	0	0	0
42	71	153	14	0	0	0	0	0	0
28	44	104	13	0	0	0	0	0	0
15	20	32	7	0	0	0	0	0	0
8	10	16	6	0	0	0	0	0	0
2	5	13	0	0	0	0	0	0	0
2	2	2	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

[MULTIPLE SCHEME]

1024	2941	9725	0	0	10968	1024	2541	9725	0
1024	2145	6725	0	0	6628	1024	3654	15542	0
1024	2112	6314	0	0	3213	1024	3576	15198	0
1024	2194	6944	0	0	1537	1024	3599	15377	0
1024	2177	6857	0	0	1537	1024	3599	15377	0
1024	2146	6618	0	0	763	1024	3635	15289	0
1024	2183	6675	0	0	393	1024	3605	15445	0
1024	2128	6358	0	0	181	1024	3573	15071	0
1024	2193	6977	0	0	90	1024	3544	14742	2
1024	2205	6577	0	0	48	1024	3568	14850	17
1024	2110	6822	0	0	26	1005	3489	14377	42
1023	2205	7117	2	5	963	963	3306	13506	95
1021	2153	6679	3	3	868	2779	2779	10919	154
1018	2118	6712	4	3	714	2044	2044	7240	205
1014	2096	6246	8	1	509	1369	4729	199	205
1006	2019	5727	19	0	310	717	4729	199	2
987	2048	6234	28	0	310	717	4729	199	0
959	1967	5953	42	0	157	323	913	104	0
917	1852	5050	75	0	53	97	239	31	43
842	1633	4351	76	0	22	33	65	19	0
766	1438	3870	81	0	3	4	6	3	0
685	1275	3365	100	0	0	0	0	0	0
585	1110	2954	103	0	0	0	0	0	0
482	868	2140	96	0	0	0	0	0	0
388	696	1788	88	0	0	0	0	0	0
298	508	1170	75	0	0	0	0	0	0
225	362	812	65	0	0	0	0	0	0
156	246	488	50	0	0	0	0	0	0
108	181	297	44	0	0	0	0	0	0
64	107	235	33	0	0	0	0	0	0
31	48	22	16	0	0	0	0	0	0
15	24	50	5	0	0	0	0	0	0

