AN APPROXIMATE AND EMPIRICAL STUDY OF THE DISTRIBUTION OF ADDER INPUTS AND MAXIMUM CARRY LENGTH PROPAGATION

Oscar N. Garcia   Harvey Glass   Stanley C. Haimes

Department of Electrical and Electronic Engineering
University of South Florida, Tampa, Florida

This paper investigates, using sampled data, the commonly used hypothesis that integer operands reaching the adder of a computer are uniformly distributed. Questions raised on the validity of that hypothesis are reinforced and their impact on the calculation of the average of the worst case length of carry propagation is considered. An approximate formula is developed for the worst case carry chain length when the arithmetic operands are restricted in magnitude.

Keywords: binary addition, carry propagation, average worst case carry chain

Introduction

A central problem in the design of high-speed arithmetic units is minimization of the time required for carry propagation in a binary adder. In the addition of two numbers, several carry propagation chains may occur. Because addition time is proportional to the length of the longest propagation chain, the length of this chain is of particular significance.

An assumption frequently made in the analysis of an adder is that input values are evenly distributed among all possible combinations. This assumption simplifies the analysis of the expected length of the maximum carry propagation chain. Yuen (5) has observed that if the input values are not uniformly distributed that carry propagation will be significantly different than that obtained from uniformly distributed values.

This paper describes a study in which empirical data was collected during machine execution in order to investigate this problem. Input values to a binary adder on a specific computer in a particular environment were gathered and examined. The results indicate that the assumption of uniformly distributed input values may not be supportable.

Previous Research

Several authors have attacked the problem of calculating (or bounding) the expected length of the longest carry propagation chain. As early as 1946, Burks, Goldstine, and von Neumann (1) in a classical analysis showed that if two binary numbers to be added are uniformly distributed, an upper bound on the expected length of the longest carry in an n-bit adder is $\log_2 n$. In 1955, Gilchrist, Pomerene, and Wong (2) demonstrated a logical design to detect the completion of all carry propagation. A carry propagation will have completed when we may determine that for every bit position either a 'one' has been propagated, or conversely that it will not be propagated. In the latter case it is convenient to refer to zero carry and to consider the lengths of both zero and non-zero carry propagation chains.

Reitwiesner (3) in 1960 derived an algorithm for the determination of the expected length of the longest zero or non-zero carry propagation chain and Briley (4) in 1973 exhibited a tighter bound on non-zero carry propagation than that presented in the classical paper by Burks et al (1). For two's complement addition they new bound is $(\log_2 n - \frac{1}{2})(1 - \frac{1}{n})$ for the average worst case of the carry length. All of these authors assumed that the numbers added are uniformly distributed. As Reitwiesner points out, "That this hypothesis thoroughly represents reality is not contended; however, it is adopted here as a first approximation to reality."

The assumption of uniform distribution does seem inconsistent with our intuition. One would anticipate that more positive than negative integers will occur and that numbers of smaller magnitude will occur more often than numbers of greater magnitude. Yuen (5) cites these reasons in a letter commenting on Briley's results and points out that a designer of arithmetic units should be wary of using results based upon the assumption of uniformly distributed input values. Yuen demonstrates that if input values are not uniformly distributed different results for the expected length of the longest carry propagation chain may occur. He further suggests that it would be of interest to obtain some empirical data on the distribution of numbers during actual machine execution.

Method of Data Collection

There are serious difficulties with collecting a set of adder input values that may be described as "representative." The sample will depend upon:

- the machine
- the operating system, and
- the mix of programs run.

We chose not to be deterred and designed a means of collecting samples from a particular machine in a particular environment in the hope that the information would help to provide insight into the problem.

The samples were collected on the IBM 360/75 under the OS/MVT operating system at the Computer Research Center of the University of South Florida. Because the great majority of work processed at the Center is either a FORTRAN or COBOL compilation we have, to date, collected our data during the running of these compilers.

The instruction sequence of a compiler in execution was sampled at intervals of 16.67 milliseconds with a specialized software monitor using the timer interrupt capability of the IBM 360. At the completion of each interval, the monitor determined whether the monitored program has received CPU service during the interval. When it has, the following information is recorded for later analysis:

- the next instruction to be executed,
- the values (contents) of operands addressed by the instruction, and
- the values in both the base and index registers referenced by the instruction for memory address calculations.

Note the last set of values. The parallel adder on this computer serves two purposes. It is used not only to implement the execution of arithmetic instructions but also to calculate effective memory addresses. In our analysis, this latter case has represented by far the greater volume of adder use in the machine.

Tabulation of the Empirical Data

After a sample is collected the data is examined by a program which tabulates the values of operands only for the machine instructions listed in Table 1. Note that this table contains both (two's complement)arithmetic and logical instructions. For our purpose we assume that the logical instructions operate exactly as their arithmetic counterparts. Their only difference lies in how the processor status bits are set for later interrogation by branching instructions. For all subtract instructions the operand value tabulated for the subtrahend is the two's complement of that value.

For each of the selected operations, either arithmetic, logical or address calculation, the operation is simulated with the actual operands in order to tabulate the length of the longest carry propagation chain. The instructions in the sample totaled 51,229. Of these 3993 or 7.79% were arithmetic instructions by our definition. Table 1 lists the frequency of each such instruction in the sample. Since the same arithmetic unit is utilized for the computation of the effective memory addresses of the operands of the sampled instructions, it was decided to analyze this data as well. The number of address calculations required by all these instructions totaled 56,502. Notice that some instructions in the IBM 360 system require no effective address calculation while quite frequently others require either one or two calculations. There are either one or two addition steps in each address calculation depending upon the format and the use of an index register in an instruction.

The tabulations of the results obtained from arithmetic instruction operands are kept separate from those obtained from effective address computation.

Table I. Instructions Sampled

	Number Times Observed	
	2,456	Add Arithmetic
	14	Add Logical
	1,523	Subtract Arithmetic
	0	Subtract Logical
Total	3,993	
	47,236	Other Instructions
Total	51,229	

Nature of Arithmetic Operands The operands corresponding to each of the binary arithmetic and logical operations mentioned above are 32 bits long. They are treated as positive and negative integers in two's complement representation. In performing the tabulation the full range of these arithmetic operands, which is from -2^{31} to $+2^{31}-1$, is evenly divided into 64 intervals. This means that there will be 32 intervals for the positive integers and 32 for the negative: the most significant six bits of the sampled operand determined the interval to which it was allocated. Each interval contained a tally for the occurrences of up to 2^{26} different integers in that interval. For an interval m for example, it would contain the number of occurrences of integers from $m \times 2^{26}$ to $((m + 1) \times 2^{26} - 1)$ where m is an integer $32 \leq m \leq 31$.

The results obtained are shown in Table II. The arithmetic (and arithmetic-logical) operations considered yielded 7989 operands. If these operands were uniformly distributed over the range of two's complement 32 bit integers, we would have obtained about 125 of them per each of the intervals. Table II indicates that this is far from being the case here. As a matter of fact, these results indicate that 97% of all values are in intervals -1 and 0, with interval 0 above accounting for 80% of the total. Since $2^{26}-1$=64, 108, 863 it is not surprising that we satisfy such a high percentile of the usage within that range for the most common integer operations. The results given in Table II are shown graphically in Figure 1.

Nature of Operands for Effective Address Computations In the IBM 360 system the largest effective address requires 24 bits in general, but in the machine used in this experiment, only 22 bits sufficed to address all of memory. Therefore, the tabulation of the operand consisted of treating the samples as 22 bit unsigned positive integers, i.e. integers in binary representation. Just as in the case of the arithmetic operations the range from 0 to 2^{22} = 4,194,304 addresses was divided into 64 intervals of equal size with 2^{16} = 65,536 different possible values within each interval. Similarly for a given interval m the values ranged from $m(2^{16})$ to $((m + 1)(2^{16}) -1)$ with $0 \leq m \leq 63$.

Table III shows the results obtained. The data again indicates a strong clustering around the small integer values, particularly around the intervals 0, 1 and 2. In effect the range from 0 to 3 x 2^{16} -1 contained again 97% of all effective address operands for the computation.

These results are shown graphically in Figure 2.

The total number of operands for effective address computations found in the 51,229 sampled instructions was 113,004 or around 2.2 operands per instruction. This indicates that, on the average, slightly over one effective address computation takes place per instruction executed. If these operands had been uniformly distributed over the range considered we would have found 2568 in each of the intervals, far from what the experiment shows.

Distribution of the Lengths of Carry Chains

The approaches mentioned earlier to determine the worst case carry propagation chain have concentrated in finding the average value of such chain. We found that a study of the distribution of the frequencies of the longest carry chains for each operation is very illuminating. Such distributions are shown in Figure 3 for the binary arithmetic operations mentioned and in Figure 4 for the effective address computation.

These values were obtained by capturing the operands involved in either case and by simulating the addition and finding the longest length carry chain. The subtraction operations are performed as an addition of the complement of the subtrahend and the carries of this addition are counted; however, the carries which may be generated during the complementation operation are not included in the tabulation. The two figures show quite different distributions of carry lengths.

In Figure 3 the average value of the worst case carry length is 10.37. It is seen, however, that there are three major areas of interest in the distribution. One is on carries of length below six, another of length between 14 and 16 and another on carries of length greater than 27.

Table II. Distribution of Operands for Arithmetic Calculations

Interval	% of Total	Number in Interval
-32	0.	17
-31	0.	0
-30	0.	0
-29	0.	0
-28	0.	6
-27	0.	1
-26	0.	0
-25	0.	9
-24	0.	25
-23	0.	1
-22	0.	0
-21	0.	20
-20	0.	1
-19	0.	0
-18	0.	0
-17	0.	3
-16	0.	1
-15	0.	1
-14	0.	0
-13	0.	0
-12	0.	3
-11	0.	3
-10	0.	2
-9	0.	2
-8	0.	4
-7	0.	1
-6	0.	1
-5	0.	3
-4	0.	4
-3	0.	0
-2	0.	5
-1	17.	1377
0	80.	6361
1	0.	13
2	0.	0
3	0.	5
4	0.	4
5	0.	1
6	0.	2
7	0.	0
8	0.	3
9	0.	2
10	0.	0
11	0.	0
12	0.	2
13	0.	0
14	0.	0
15	0.	1
16	0.	7
17	0.	0
18	0.	0
19	0.	1
20	0.	30
21	0.	0
22	0.	2
23	0.	25
24	0.	13
25	0.	0
26	0.	0
27	0.	7
28	0.	0
29	0.	0
30	0.	0
31	0.	17

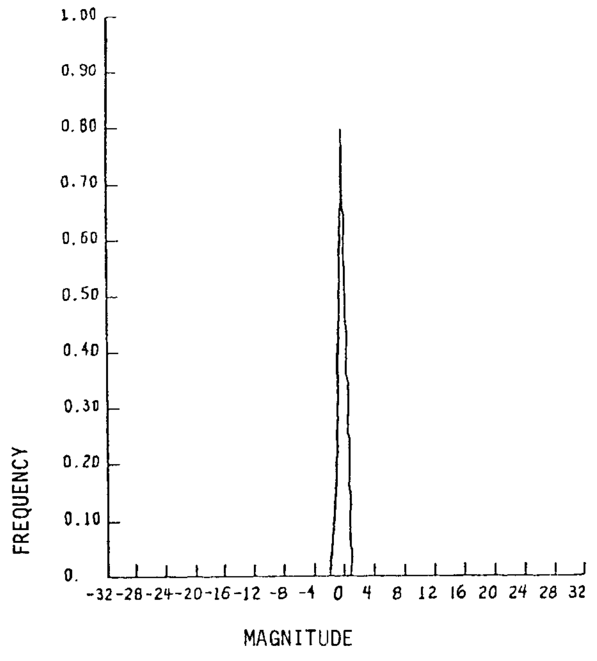

Figure 1. Distribution of Arithmetic Operands (N = 7,986)

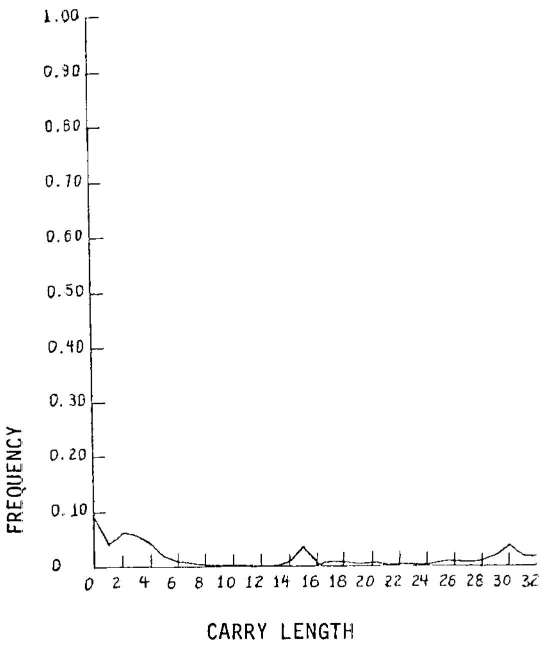

Figure 3. Distribution of Lengths of Longest Carry Propagation Chains for Arithmetic Calculations (N = 3,993 Average Length = 10.37)

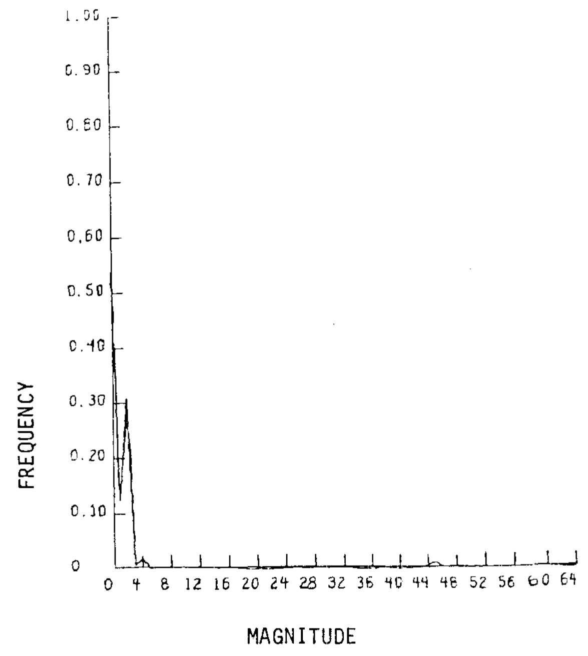

Figure 2. Distribution of Address Calculation Operands (N = 113,004)

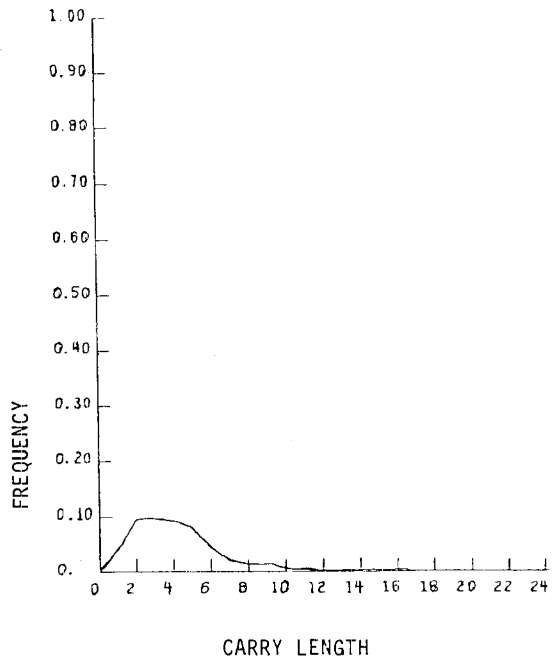

Figure 4. Distribution of Lengths of Longest Carry Propagation Chains for Address Calculations (N = 56,502 Average Length = 3.89)

Table III. Distribution of Operands in Effective Address Computations

Interval	% of Total	Number in Interval
0	54.	60682
1	12.	13936
2	31.	34914
3	0.	465
4	2.	1824
5	0.	0
6	0.	0
7	0.	0
8	0.	0
9	0.	0
10	0.	0
11	0.	0
12	0.	0
13	0.	0
14	0.	0
15	0.	0
16	0.	0
17	0.	0
18	0.	0
19	0.	1
20	0.	0
21	0.	0
22	0.	0
23	0.	0
24	0.	0
25	0.	0
26	0.	0
27	0.	0
28	0.	0
29	0.	0
30	0.	0
31	0.	0
32	0.	0
33	0.	0
34	0.	0
35	0.	0
36	0.	0
37	0.	0
38	0.	0
39	0.	0
40	0.	0
41	0.	0
42	0.	0
43	0.	0
44	0.	0
45	1.	4608
46	0.	15
47	0.	48
48	0.	0
49	0.	0
50	0.	0
51	0.	0
52	0.	0
53	0.	0
54	0.	1
55	0.	0
56	0.	2
57	0.	0
58	0.	0
59	0.	0
60	0.	0
61	0.	0
62	0.	0
63	0.	47

Yuen (5) suggests that the non-uniform distribution of the operands in arithmetic operations coupled with the fact that both negative and positive integers are added would invalidate the tighter bound on the average worst length carry chain developed by Briley (4). He considers four cases for addition

1) two positive operands
2) first operand positive, second negative
3) first operand negative, second positive
4) both operands negative

Based on a probability p that a positive operand occurs and that most values are of less magnitude than 2^m for an n bit word, he develops an approximate formula for the average carry chain length. Actually his formula is flawed by assuming that the fourth case above has the same carry length chain as the third. A modified formula will be developed below to correct this.

First, let's postulate a uniform distribution of positive and negative n-bit operands, but limited in magnitude to be below $|2^m|$. Then if $X \in \{0,1\}$ with equal probability, the operands look as shown in Figure 5.

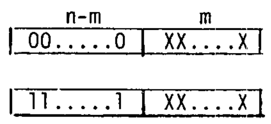

Figure 5. Positive and Negative Operands of Magnitude Less Than 2^m.

We will further assume that the bound found by Briley (4) is sufficiently tight to be used as the actual value of the average worst case carry length. Therefore, in case (1) above, which occurs with a probability p^2, the average worst carry length is $(\log_2 m - 1/2)$.

Cases (2) and (3) will have identical results because of commutativity, and will occur with a probability 2p(1-p). However, we can consider two subcases:

a) if no carry flows into the bit position of weight 2^m, then using (4) again the average worst case carry length will be $(\log_2 m - 1/2)$, and

b) if a carry flows into that bit position then the longest carry chain is $>(n-m)$.

If we let P_x be the probability that subcase b) above occurs then $(1-P_x)$ is the probability of subcase a). Let's compute the quantity P_x.

A carry will flow into the bit position of weight 2^m if it is generated at the bit position of weight 2^{m-1} or if it is generated at a lower order position at the beginning of the chain and transmitted through all consecutive positions. Given that for any of the m low order bit positions ones and zeroes are equally likely, the condition for carry generation is that the corresponding bit positions be both ones (which occurs with probability 1/2 . 1/2 = 1/4) and for transmission that a zero-one or one-zero combination occurs in corresponding bits (which occurs with probability 1/2 . 1/2 + 1/2 . 1/2 = 1/2). Then the probability:

$$P_x = \sum_{i=0}^{m-1} (1/4)(1/2)^i$$

If we consider the sum P_x above this is a truncated geometric progression of rate 1/2 and sum $\frac{1}{2} - (1/2)^{m+1}$. Therefore

$$P_x = \frac{1}{2} - (\frac{1}{2})^{m+1}$$

For $m > 8$, which is always reasonable, $P_x \simeq 1/2$.

If we further restrict $m \geq n/2$ then we can assert that in subcase (b) the length of the worst case carry chain will have an average of $n-m + \log_2 m - 1/2$.

For cases (2) and (3) the average worst case chain for $m \geq n/2$ will be

$$(1 - P_x)(\log_2 m - 1/2) + P_x(n-m + \log_2 m - 1/2)$$

Finally for case (4) which occurs with a probability $(1-p)^2$ the worst case carry chain will be $(\log_2 m - 1/2)$ bits long on the average, since the leading ones in the high order positions of the operands do not propagate a carry chain.

The average worst case carry length under these circumstances is given by $C_n \simeq p^2 \cdot (\log_2 m - 1/2) + 2p(1-p)\left[(1-P_x)(\log_2 m - 1/2) + P_x(n-m + \log_2 m - 1/2)\right] + (1-p)^2(\log_2 m - 1/2)$ or calling $\overline{m} = \log_2 m - 1/2$

$$C_n \simeq p^2\,\overline{m} + 2p(1-p)\,\overline{m} + P_x(n-m) + (1-p)^2\,\overline{m}$$

$$C_n \simeq 2_p{}^{(1-p)}P_x(n-m) + \overline{m}$$

Furthermore for $m > 8$ we can say

$$C_n \simeq p(1-p)(n-m) + \overline{m}$$

Comparison with Empirical Results

The values obtained empirically for the average worst case length of carry propagation for arithmetic computations was 10.37 and for effective address computation was 3.89. The former value is far above the bound given by Briley which would be 4.5 for n = 32. The latter value is closer to it for n = 22 which yields 3.96 using Briley's bound. While in neither case we had a uniform distribution, in the effective address computation experiment there were only positive operands.

In the case of arithmetic operations let us assume that 80% of the operands were positive. Since there were 1,523 subtractions out of the 3,993 arithmetic operations sampled, we can say that 38% of the operands changed signs or that we had 0.62 x .20 + 0.38 x .80 = 43% of the operands as negative integers. This yields p = 0.57. If we postulate m = 16 then $\overline{m}$ = 3.5 and $C_n \simeq (0.43)(0.57)(16) + 3.5 = 7.4$. This is still far from our experimental value of 10.38 but it is much closer than the value obtained assuming a totally uniform distribution of the operands.

The values shown in Figure 3 seem to indicate that it is likely that even when the magnitudes of the operands are restricted to less than 2^m, we do not have anything close to a uniform distribution. This would go along with our intuition about the abundance of positive and negative integers of values near $\pm$ 1.

Conclusions

It is obvious that the $(\log_2 m - 1/2)$ bound is not realistic because of the grouping of the integer operands around $\pm$ 1.

The main motivation for the study of the average worst case carry length is to consider the advantages of a carry completion adder. In this respect the observations of Reitwiesner (3) with regards to chains of carries and chains of no-carries or zero carries are also valid. These considerations have not been included in our approximate formula.

The experimental study reported has a number of acknowledged shortcomings and is far from complete. For example, it only includes addition and subtraction and ignores other arithmetic operations such as complementation, multiplication and division. Furthermore, the analysis only monitored a portion of the activity of the computer, specifically the execution of the COBOL and FORTRAN compilers. While these compilers account for a large portion of the processor activity it would be of interest to examine other types of programs at arbitrary times of processor execution.

Acknowledgements

The authors would like to express their sincere appreciation to S. Garrett of the Department of Electrical and Electronics Engineering and to R. Fascenda and P. Lyons of the Computer Research Center at the University of South Florida for their comments and assistance.

References

1. A. W. Burks, H. H. Goldstine, and J. von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," in Collected Works of John von Neumann, Vol. 5, A. H. Taub, Ed., New York: MacMillan, 1963, pp. 34-79.

2. B. Gilchrist, J. H. Pomerene, and S. Y. Wong, "Fast carry logic for digital computers," IRE Trans. Electron. Comput., vol. EC-4 pp. 133-136, Dec., 1955.

3. G. W. Reitwiesner, IRE Trans. Electron. Comput., vol. EC-9, pp. 35-38, Mar., 1960.

4. B. E. Briley, IEEE Trans. Comput., vol. C-22, pp. 459-463, May, 1973.

5. C. K. Yuen, IEEE Trans. Comput., Vol. C-23, p. 333, Mar., 1974.