Seminar on Digital Computing

REFERENCE LIST NO. 1

   NBS. ERA Publication No. 23 (30 November 1948) Available in ERA Library on
   3-day loan basis.

2. "Proceedings of a Symposium on Large-Scale Digital Calculating Machinery".
   Held January 1947 at Harvard University; published 1948.

   (October 1948).

   I.R.E. 36, 1452 (December 1948).

   1948) and 3, 69 (April 1948).

   Engineering 70, 975 (December 1948).

   (August 1947).

8. Project Whirlwind, Summary Report No. 2, in 22 volumes. Volume 4 is entitled
   "Introductory Material".

   Computing Instrument", Part I (second edition, 2 September 1947; first edition,

10. Goldstine and Von Neumann, "Planning and Coding of Problems, etc.", Part II
    Vol I (1 April 1947); also Part II, Vol II (15 April 1948). Inst. for Adv. Study.

    Harvard (1946).

* * * * * * * * * * * * * * * *

Your attention is directed to the quarterly publication,
Mathematical Tables and Other Aids to Computation (abbreviated MTAC). Each
issue has a section containing news and abstracts of articles in the field
of Automatic Computing Machines, in addition to feature articles on this
subject. Our library's file begins with the January, 1947, issue. Thumbing
through the last seven or eight issues will give you a good bird's-eye view
of the field today.
<table>
<thead>
<tr>
<th>KIND OF COMPLEMENT SYSTEM</th>
<th>10's</th>
<th>9's</th>
<th>2's</th>
<th>1's</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULUS M</td>
<td>$10^4$</td>
<td>$10^4 - 1$</td>
<td>$2^4$</td>
<td>$2^4 - 1$</td>
</tr>
<tr>
<td>HOW MANY DIFFERENT NUMBERS EXPRESSES?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPRESENTATION $A^*$</td>
<td>4999</td>
<td>4999</td>
<td>0111</td>
<td>0111</td>
</tr>
<tr>
<td>VALUE</td>
<td>+4999</td>
<td>+4999</td>
<td>+0111</td>
<td>+0111</td>
</tr>
<tr>
<td>MOST POS. EXPRESSES NUMBER:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPRESENTATION $A^*$</td>
<td>5000</td>
<td>5000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>VALUE</td>
<td>-5000</td>
<td>-4999</td>
<td>-1000</td>
<td>-0111</td>
</tr>
</tbody>
</table>

IN AN ADDITIVE MACHINE:
END-AROUND CARRY?
ZERO GENERATED BY MACHINE

IN A SUBSTRACTIVE MACHINE:
END-AROUND BORROW?
ZERO GENERATED BY MACHINE

RELATION BETWEEN VALUE $A$ & REPRESENTATION $A^*$:

$$A = A^* - \delta \cdot M,$$

WHERE

$$\delta = 0 \text{ for } A = 0, 1, 2, 3, 4$$
$$\delta = 1 \text{ for } A = 5, 6, 7, 8, 9$$

AND $\delta$ = LEFTMOST, OR "SIGN", DIGIT OF $A^*$

RELATION BETWEEN REPRESENTATIONS OF $A$ & $-A$

$$A^* + (-A)^* = M$$

PREP. BY 

Seminar Math
2 Feb 1949
PROBLEMS IN NUMBER NOTATION

1. Add two more columns to the DIGITAL MACHINE NOTATION table, for 4-digit octal (radix 8) integers, in 8's complements and in 7's complements.

2. Plot A vs. A* for 4-digit binary integers expressed in 2's complements and in 1's complements.

3. Plot A vs. A* for 2-digit decimal integers expressed in 10's complements and in 9's complements.

4. Plot A vs. A* for 2-digit octal integers expressed in 8's complements and in 7's complements.

5. Posted in every drafting room is a table of decimal equivalents. Make up a table of "octal equivalents" of fractions of an inch from 1/64 to 1 inch, by 64ths.

6. Convert 849.163 (decimal) to binary notation. Convert this same number to octal notation. Repeat, using several different methods.

7. Convert .763.102 (octal) to binary notation, employing several different methods.

PROBLEM IN PROGRAMMING

This problem is the evaluation and tabulation of a polynomial over a range of values of two independent variables.

Given: Constants A, B, C, D, E, F are stored at a, b, c, d, e, f, respectively.

Req'd: To evaluate the polynomial

\[ P(x,z) = Ax^2 + Bxz + Cz^2 + Dx + Ez + F \]

for \( x = 0, \pm1, \pm2, \ldots \pm15 \)

and \( z = 0, \pm1, \pm2, \ldots \pm15 \).

Store the P's in 961 consecutively addressed memory boxes. Assume that the coefficients are integers whose absolute values do not exceed, say, 2^10; this keeps the P's within range of the 30-digit memory boxes. Select numerical values for the coefficients if you wish. Utilize the ability of the machine to accumulate sums and sums of products. Coefficients need not be positive.
Binary Accumulator, mod \((2^n-1)\), Summary of Numerical Properties

A. The following properties are common to the \(n\)-place additive accumulator with end-around carry and the \(n\)-place subtractive accumulator with end-around borrow:

1. Range of \((2^n-1)\) different numbers.

2. Pos and neg numbers are expressed in the "1's-complement" system, in which:
   a. the neg of a number can be formed by subtracting the number from \((2^n-1)\); i.e., by replacing 1's with 0's and vice versa; and
   b. the left-most digit of a positive number is 0 and the left-most digit of a negative number is 1.

3. Absolute value of the numbers represented may not exceed \((2^n-1)\); e.g., +2047 and -3047 are the range limits in a 12-place system; +7 and -7 the range limits in a 4-place system.

4. Nature of left shift: The most useful form of shift is that in which shifting \(k\) places to the left is equivalent to multiplication by \(2^k\); i.e., adding the number in the accumulator to itself, repeating this process \(k\) times. The so-called "circular shift" has this property in the mod \((2^n-1)\) system. The digit occupying the left-most place shifts around into the right-most place on each shift. (Note that a \(k\)-place right circular shift would not be equivalent to division by \(2^k\). What kind of shift would?)

B. The following properties are peculiar to the \(n\)-place additive accumulator (abbreviated AA) with end-around carry:

1. A number transmitted to AA is added to the number in AA.

2. To subtract a number from the number in AA, the complement of the incoming number must be transmitted to AA.

3. Of the two forms which zero may take in a mod \((2^n-1)\) system, only the lll...1 form, or "neg zero" can be generated by AA. (Try to beat this on paper, in case you are not convinced.)

C. The following properties are peculiar to the \(n\)-place subtractive accumulator (abbreviated SA) with end-around borrow:

1. A number transmitted to SA is subtracted into the number in SA.

2. To add a number to the number in SA, the complement of the incoming number must be transmitted to SA.

3. Only the 000...0 form of zero, of "positive zero", can be generated by SA.
COMMENTS ON BINARY MULTIPLICATION

The following tabulation is a step-by-step analysis of what would take place in an arithmetic unit having an 8-place accumulator and a 4-place Q-register, during the automatic multiplication process. This abridged machine follows the multiply algorithm of the ERA-NES Computer (ERA Publ. No. 23, Fig. 4), except that the size of A, Q, and X are reduced. The example shown is the multiplication of +5 (multiplicand in X) by -6 (multiplier in Q), with A initially clear. X contains 0101 throughout the process; A sees this as 0000 0101.

<table>
<thead>
<tr>
<th>Step</th>
<th>A</th>
<th>Q</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>0000</td>
<td>0000</td>
<td>1001</td>
</tr>
<tr>
<td>Subtr X</td>
<td>1111</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>AL 1</td>
<td>1111</td>
<td>0101</td>
<td></td>
</tr>
<tr>
<td>Add X</td>
<td>1111</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>QL 1</td>
<td>1111</td>
<td></td>
<td>0011</td>
</tr>
<tr>
<td>AL 1</td>
<td>1111</td>
<td>0101</td>
<td></td>
</tr>
<tr>
<td>Don't add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QL 1</td>
<td>1110</td>
<td>1010</td>
<td>0110</td>
</tr>
<tr>
<td>AL 1</td>
<td>1110</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>Don't add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QL 1</td>
<td>1101</td>
<td>0111</td>
<td>1100</td>
</tr>
<tr>
<td>AL 1</td>
<td>1101</td>
<td>0111</td>
<td></td>
</tr>
<tr>
<td>Add X</td>
<td>1101</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>QL 1</td>
<td></td>
<td></td>
<td>1001</td>
</tr>
<tr>
<td>Add X</td>
<td>1110</td>
<td>0001</td>
<td></td>
</tr>
</tbody>
</table>

Result: Product, -30, in A; multiplier, -6, again in Q.

ADVANCE ASSIGNMENT

1. Become familiar with the automatic binary multiplication process by running through an analysis similar to the above for the various combinations of positive and negative multiplier and multiplicand. Be careful to start with numbers within machine range. Perform one analysis for the case where the number initially in A is not zero, and it is desired to follow the HOLD MULTIPLY, or MApy, command.

2. So that you may more efficiently follow next week’s discussion of division, set up a step-by-step analysis of the divide algorithm, Fig. 5, for a few combinations of positive and negative quotient and divisor. Read the comments on p. 22 of Publ. No. 23. Proofs will be given next time — until then, try the algorithm out to see if it works for your examples.

NOTES ON MACHINE DIVISION

Preliminary Considerations

Quotient \( Q \) and remainder \( R \) are related to dividend \( N \) (numerator) and divisor \( D \) (denominator) by:

\[
N = Q \cdot D + R, \quad \frac{N}{D} = Q + \frac{R}{D}
\]  \hspace{2cm} (1)

Several cases are of interest to the present discussion:

Case I. Here, (1) is restricted only by the requirement that

\[
0 \leq |R| \leq |D|.
\]  \hspace{2cm} (2)

Question: This limits (1) to how many sets of \( Q \) and \( R \), for a given \( N \) and \( D \)?

Case II. Here, (2) is retained, but in addition suppose \( Q \) is restricted to odd integers (because, say, of a property of some machine process). Then \( Q \) will have the same sign as \( N/D \); \( R \) may take on either sign.

Case III. Here, (1) is restricted only by the requirement of non-negative remainder whose absolute value is less than that of divisor. This is the DIVIDE operation defined on p. 19 of ENA Publ. No. 23.

\[
0 \leq R < |D|.
\]  \hspace{2cm} (3)

Question: How many \( Q, R \) for given \( N, D \)?

Restoring and Non-restoring Division Illustrated

Decimal, restoring. Example: \( 1634 = (65)(25) + 9 \).\n
\[
\begin{array}{cccc}
0 & 6 & 5 \\
+0025) & 1 & 6 & 3 & 4 \\
0 & 0 & 0 & 0 \\
+1 & 6 & 3 & 4 \\
1 & 6 & 0 & 0 \\
+0 & 1 & 3 & 4 \\
0 & 1 & 2 & 5 \\
+0 & 0 & 0 & 9 \\
\end{array}
\]

Subtr largest multiple of 25 that will not produce overcast, each time.

Remainder.

Decimal, non-restoring. Example: \( 1634 = (68)(25) - 16 \).

\[
\begin{array}{cccc}
+1 & 4 & +6 & = 106 - 40 = 66 \\
+0025) & 1 & 6 & 3 & 4 \\
-2 & 5 & 0 & 0 & \text{Subtr until overcast has occurred.} \\
-0 & 8 & 6 & 6 & \text{R,D opp sign.} \\
+1 & 0 & 0 & 0 & \text{Add until overcast has occurred.} \\
+0 & 1 & 3 & 4 & \text{R,D same sign.} \\
-0 & 1 & 5 & 0 & \text{Subtract, etc.} \\
-0 & 0 & 1 & 6 & \text{This is } R.
\end{array}
\]
Binary, non-restoring. Example: 29 = 5·5 + 4.

\[
\begin{array}{cccccc}
+1 & -1 & -1 & +1 & -1 \\
0 & 1 & 0 & 1 & 0 & 1 \\
\hline
0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

0, D same sign. Subtract D, put +1 in Q.

0, D opp sign. Add D, put -1 in Q.

The above process always produces an odd quotient, by virtue of the peculiar (+1, -1) number system used to express the quotient.

EMA-NBS Computer. Basic Divide Algorithm

Consider our DIVIDE algorithm, Fig 5. Let us first examine the "basic" portion of the flow diagram. This performs non-restoring binary division similar to the above example, except that 1's and 0's are (temporarily) put into Q in place of the required +1's and -1's, respectively. Except for a correction to take this substitution into account, the division process is actually all done just prior to the final "QL 1".

At this point the Q-register contains a pseudo-quotient P whose 1's represent +1's, but whose 0's indicate where -1's really should be. Suppose that the -1's had been kept track of in an imaginary register containing a number S composed of 1's and 0's such that the 1's in S represent the -1's; i.e., P and S are complementary to each other. The quotient we want is then (P - S). But since P and S are complements, this is the same as 2P. Therefore the required correction is simply to shift P one place to the left. Hence the final "QL 1".

At the end of the "basic algorithm" we have an odd quotient, as stated on p. 22. An odd number in our system of notation is characterized by inequality of the extreme right and left digits; i.e., q00 ≠ q1. (Exercises: (1) Verify this fact. (2) Analyze the steps in the algo by which these two digits come out opposite. (3) If they were to come out alike, what would this indicate?)

In other words, the "basic algorithm" performs the kind of division defined in Case II on the preceding page.

EMA-NBS Computer. Corrections Yielding Non-negative Remainder

It is required to perform the kind of division we have specified as Case III. The correction steps preceding and following the basic algorithm in Fig 5 accomplish this. These corrections will be justified in two stages. First, suppose that the basic algo has been followed, producing a quotient Q' and a remainder R' having the Case II properties:

\[
N = Q'D + R', \quad 0 \leq |R'| \leq |D|, \quad Q' \text{ odd.} \quad (4)
\]
We have three quantities \( N \), \( Q' \), \( R' \), each of which may be + or -; there are therefore 8 possible cases to deal with (the sign of \( D \) is determined by \( N \) and \( Q' \)). These cases are listed in Table I, together with the corrections necessary to insure a non-negative remainder. The sign of the correction to \( Q' \) is opposite to the sign of \( D \). This adds a negative absolute \( D \) to \( Q'D \) to compensate for the positive absolute \( D \) added to \( R' \).

These corrections are satisfactory, except that adding and subtracting are called for in the \( Q \)-register. This is expensive, since full carry or borrow facilities would be required in \( Q \). Note that the corrections for Cases 5 and 6 do not generate carry or borrow ("carrow") in \( Q \). Cases 7 and 8 therefore present the only difficulty.

Fortunately, the problem has a reasonable solution. Note that Cases 7 and 8 correspond to negative \( N \). The sign of \( N \) is known at the beginning of the division process. If \( N \) is negative, suppose we immediately make it more negative by an amount equal to the absolute value of \( D \). This pre-loading of \( N \) (to give \( N^* = N - |D| \)) must be compensated for later by adding +1 or -1 to \( Q \). Assuming we have pre-loaded \( N \), we then have at the end of the basic algo:

\[
N^* = QD + N^*, \quad 0 \leq |N^*| \leq |D|, \quad \text{ \( Q^* \) odd, \quad \text{\( N^* < 0 \).}}
\] (5)

The pre-loading step affects the 4 cases for which \( N \) is negative. 

Table I is still satisfactory for Cases 1-4. Table II re-examines Cases 5-8 in terms of \( R^*, N^*, Q^* \). The net correction is composed of two parts: a negative remainder correction (same rules as in Table I), and a further correction to \( Q^* \) due to the pre-loading of \( N \). Since \( N^* \) is too negative by an absolute \( D \), the correction to \( Q^*D \) must be a positive absolute \( D \); i.e., the correction to \( Q^* \) must be of the same sign as \( D \).

The net corrections are now all seen to be feasible without carrow facilities in \( Q \). If we make the right-hand (\( q_0 \)) place of the \( Q \)-register a "unit binary counter", then 1's may be transmitted to \( Q \) on a simple AWC (add-without-carry) basis. The corrections may then be collapsed into 4 cases (1-2, 3-4, 5-6, 7-8), as shown in Fig 5.

A physically simpler way of correcting \( Q^* \) is to OMIT \( q_2 \) TO \( q_0 \) instead of AWC 1 TO \( q_0 \), since a pair of shifting gates are already available for this purpose. Then the AWC 1 TO \( q_0 \) step within the basic part of the algo is made simply INSERT 1 IN \( q_0 \).

Exercises:

1. Tabulate the steps of the divide algo for several examples, assuming a machine with 4-digit numbers and 8-place accumulator. Be sure to test some of the peculiar cases like 0/0, 0/5, -3/0, etc.

2. Justify (or refute) the second paragraph on p. 22 ("Prior to division . . . ").
TABLE I

<table>
<thead>
<tr>
<th>Case</th>
<th>N</th>
<th>Q</th>
<th>D</th>
<th>q₀</th>
<th>Corr to R¹</th>
<th>Corr to Q¹ corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>e</td>
<td>5</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>1</td>
<td>Abs add D -1</td>
</tr>
<tr>
<td>f</td>
<td>6</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>0</td>
<td>Abs add D +1</td>
</tr>
<tr>
<td>g</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>1</td>
<td>Abs add D +1</td>
</tr>
<tr>
<td>h</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>Abs add D -1</td>
</tr>
</tbody>
</table>

TABLE II

<table>
<thead>
<tr>
<th>Case</th>
<th>N²</th>
<th>N¹</th>
<th>Q²</th>
<th>D</th>
<th>q₀</th>
<th>Corr to N¹ for neg N¹</th>
<th>Corr to N¹ for neg N¹</th>
<th>Corr to Q¹ due to N¹</th>
<th>Net corr to Q¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>7</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>None</td>
<td>None</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>b</td>
<td>8</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>0</td>
<td>None</td>
<td>None</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>1</td>
<td>Abs add D</td>
<td>+1</td>
<td>-1</td>
<td>None</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>0</td>
<td>Abs add D</td>
<td>-1</td>
<td>+1</td>
<td>None</td>
</tr>
</tbody>
</table>

For summary, see flow diagram, Fig 5, Publ. No. 23.
Mapy uses Parts I and II.
MPpy uses Part II only.

Initial Contents:
X: MultipliCand
Q: mutiplier
A: Initial number (zero for MPpy)

Final Contents:
X: MultipliCand
Q: mutiplier
A: Initial number plus product

Figure 4. MULTIPLY ALGORITHM (portion of Mapy and MPpy)
INITIATE DIVIDE

CASE I: \( \frac{40}{20} = 0 \) CASE II: \( \frac{40}{20} = 1 \)

ABB SUBTR X TO A

AL 1

SUBTR X TO A

AWG I TO QO

ADD X TO A

QL 1

SHIFT COUNT = S

S = 30

S < 30

CASE I

II

AWC I TO QO

X \times 11 \frac{q_2}{2^2} \& q_0

CASE II

I

II

AWC ADD X TO A

AWC I TO QO

ABS ADD X TO A

END DIVIDE

AWG = ADD WITHOUT CARRY

INITIAL CONTENTS:

X: DIVISOR
Q: ZERO
A: DIVIDEND

FINAL CONTENTS:

X: DIVISOR
Q: QUOTIENT
A: REMAINDER

Figure 5. DIVIDE ALGORITHM (portion of DPpy)
PROGRAMMING OF ILLUSTRATIVE ROUTINES

Example: To find the most positive number in a list.

Given: 1030 numbers stored at \( \alpha \) to \( \alpha + 1029 \).

Req'd: To find the most positive number in the list and store it at \( \mu \).

1

A.1 PE / A.2 / 1029
A.2 YQ / B.1 / \( \alpha \)

(\( * \) Best number so far is now in Q)

B.1 QP / B.2 / ----
B.2 OS / B.3 / \( \alpha + 1 \) *
B.3 TS / D.1 / C.1

- \( +,0 \)

C.1 YQ / D.1 / \( \alpha + 1 \) *

D.1 TE / E.1 / F.1

done not done

D.1

E.1 EP / E.2 / B.2
E.2 AY / E.3 / B 2
E.3 EP / E.4 / C.1
E.4 AY / D.1 / C.1

* Initial value shown - this changes for each traversal.
UNITS MARKED (*) ARE USED EXCLUSIVELY FOR INPUT