USING -2 AS A BASE FOR A NUMBER SYSTEM TO REALIZE A COMPUTER

by

Jerrold Frederick Zimmer

Thesis submitted to the Graduate Faculty of the
Virginia Polytechnic Institute
in partial fulfillment for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

Chairman C. F. Smith

A. W. Bennett

R. H. Miller

June 1968

Blacksburg, Virginia

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The author would like to thank Dr. C. F. Smith for his initial suggestion which led to the present investigation as well as his assistance during its various stages.

Thanks also go to Mrs. Joyce Noell for typing and organizational assistance.

A special note of appreciation is due his wife, Susan, for making his entire education possible.

CHAPTER I

INTRODUCTION

Problem

Number systems for use in computers have been the object of many recent studies. While mathematicians have used and looked into number systems from a basically theoretical viewpoint for quite some time, it has not been until lately that the application by computers for these number systems have prompted electrical engineers to restudy the entire problem. In fact, most of the recent studies have been prompted by technological advances in such areas as memory devices and, most importantly, integrated circuits. This rapid change in device technology has stirred up so many areas of electrical engineering that one must be ready to change long held ideas overnight. For example, the theme of this paper is certainly not new or unique, but rather, the application of the results will hopefully be interpreted in the light of this new technology we have available.

History

By way of history we find that the old standby number system, base +10, was first used because of the ease of counting on the 10 fingers. The Chinese introduced the biquinary number

system with their abacus using five beads on one side of a wooden divider and two on the other side to count groups of fives. From then until 1930 most devices were based on the simple count of 10 system. For example, the work of Babbage used gears to realize a count of 10 system. However, in the late 1930's a Harvard student built the first computer based on simple relays and switches. Since relays are either open or closed, the binary system of counting was begun in the computer industry and todays' computers are still following such a system. [3]

Most of the work in the past decades has concentrated on ways of speeding up operation of the computer circuits. Vacuum tubes were replaced by transistors and integrated circuits are now replacing transistors as the control units in computers. What is being investigated in this paper is the possibility of using another number system as the basis for a computing system. The problem seems analogous to that posed by the automobile engine. The 4 cycle piston engine works fine and has been developed to a very high degree, but certainly there must be other equally fine ways of doing things. The jet engine took over in the airplane and the wankel engine is now making its bid for the automobile market. Can it be that another number system will be the successor to the binary number system in the design of computer systems?

Scope and Organization of Paper

This paper will use -2 as a base since this shows the most promising application and can be studied to provide a jumping-off point for other number systems. The primary reason binary number systems, be it positive or negative, are the most popular is the unreliable operation of circuits in anything but two states, the "on" and "off" states as they are so often called. Hopefully other systems can be devised that will be able to handle three or more states. For example, the new fluid logic technology might use a symmetrical ternary based logic (+1, 0, -1) represented by above atmospheric pressure, atmospheric pressure, and below atmospheric pressure respectively.

The first section of the paper will consider number systems with emphasis on those systems with a negative base. After a good understanding of negative base systems, the system using -2 as a base will be discussed.

The second section of the paper will attempt to explain some of the ways logic elements can be arranged to perform the necessary arithmetic operations in a computer. It will also be necessary to investigate methods of input and output on such machines.

The conclusion will present further topics for study as well as a summary of the possibilities for future promise of the present study. Note that of the reference in the Bibliography, very few refer to the machine concepts of negative base number systems, and most are references to the design of digital computers in general. The major stimulus for the work done in this paper was given by a series of articles by de Regt [9].

CHAPTER II

ARITHMETIC

Representation

Naturally the most basic decision faced by the designer of a digital computer is how to represent the numbers. Here, we want specifically to investigate the properties of the negative binary notation. A short review of some of the properties of other number systems would be helpful. One must remember that the system finally chosen for construction in the computer will have to be physically realizable with hardware, and as mentioned earlier, this considerably limits the choices.

The most striking aspect of negative-radix arithmetic is its simplicity. While somewhat confusing at first, a little practice will prove that negative-radix problems can be just as easy as the familiar positive-radix notation. In this paper we will try to stay clear of the deep mathematics so that the simplicity of the arithmetic becomes evident. Many examples will be given to demonstrate that the arithmetic does indeed work in all circumstances and algorithms will provide a relatively easy transition to the next section where the hardware portion of the computer is described.

The first rule of counting says that we must make use of positional notation. The familiar polynomial will be used to show this relation. Remember, any number can be represented by a sequence of n digits:

$$A_{n-1}\ A_{n-2}\ A_{n-3}\cdots A_2\ A_1\ A_0$$

This represents the quantity:

$$A_{n-1}r^{n-1} + A_{n-2}r^{n-2}\cdots A_2r^2 + A_1r^1 + A_0r^0$$

in which r is the base or radix, and A_i is taken from the set of r numbers $(0, 1, 2, \cdots, |r|-1)$. For example if we use the decimal system r = 10, A_i is chosen from $(0, 1, 2, \cdots, 9)$. Here, of course, we want to consider r = -2. For ease of understanding we should also look at r =-10. For example, the decimal quantity +1763 would be represented by 19823 in r = -10 notation. This stands for

$$1(-10)^4 + 9(-10)^3 + 8(-10)^2 + 2(-10)^1 + 3(-10)^0$$
$$= 10{,}000 - 9{,}000 + 800 - 20 + 3 = 1763.$$

Likewise, -1783 would be represented by 2397. In negative binary (base = -2), decimal -67, which is -1000011 in conventional binary, is represented by 11001101:

$$1(-2)^7 + 1(-2)^6 + 0(-2)^5 + 0(-2)^4 + 1(-2)^3 + 1(-2)^2 +$$
$$0(-2)^1 + 1 = -128 + 64 - 8 + 4 + 1 = -67.$$

We will use the term "negabinary" to refer to numbers whose base is -2 while just "binary" will be used by the base +2 system. This will remove any difficulty with the above term "negative binary."

Of course, one aspect which seems at first apparent is that in negabinary representation the lack of a sign bit will save one bit per word. However, this is not true. For any n there are 2^n combinations of the two symbols and if n = 2 there are 4 combinations, 00, 01, 10, 11, which could be interpreted as the integers from 0 to +3 or the integers from -2 to +1 as is the case with negabinary representation. So regardless of the interpretation there will still be just 2^n numbers represented.

Referring again to the polynomial representation which defines positional notation, the most significant non-zero digit is an n^{th} digit integer defined as $A_{n-1}r^{n-1}$. Assuming all $A > 0$ and $r < -1$, this term will be negative for even n and positive for odd n. Since the magnitude of this highest order term must be greater than the sum of the magnitudes of all the lower order terms in order to have a unique number, it can be said that a negative radix number having an even number of digits is negative, and one having an odd number of digits is positive.

However, just as in the example of the previous paragraph we see that the distribution of negative and positive numbers for a given quantity of digit positions might not be symmetrical. This is the case. Consider the negadecimal integers. For one digit position, we have ten possible representations: zero, and the numerals 1 through +9. Thus the lower limit is zero, the upper limit nine. For two positions, we have one hundred representations; the ten by one digit and ninety more, which are the negative numbers 19 (-1) through 90 (-90). The upper limit is nine, the lower limit -90. Table I below shows the upper and lower limits for each of the given quantities of positions for both negadecimal and negabinary.

Thus, the pattern for maximum and minimum number size is easily explained by reference to the general polynomial. Since even positions, and hence even powers, have positive values and odd positions, and odd powers, have negative values, a negadecimal number having 9's in the even positions and zeros in odd positions has the minimum value for the word size. Similarly, with 9's in the odd positions and zeros in the even, the number becomes its maximum size.

TABLE I

Upper and Lower Limits for Negadecimal and Negabinary Digit Positions

	Negadecimal Numbers		Decimal Values	
n	Lower	Upper	Lower	Upper
1	0	9	0	+9
2	90	09	-90	+9
3	090	909	-90	+909
4	9090	0909	-9090	+909
5	09090	90909	-9090	+90909
6	909090	090909	-909090	+90909

	Negabinary Numbers		Decimal Values	
n	Lower	Upper	Lower	Upper
1	0	1	0	+1
2	10	01	-2	+1
3	010	101	-2	+5
4	1010	0101	-10	+5
5	01010	10101	-10	+21
6	101010	010101	-42	+21
7	0101010	1010101	-42	+85
8	10101010	01010101	-170	+85

Table II now shows the sequence of numbers for negadecimal counting.

TABLE II

Negadecimal Counting

Negadecimal	Decimal
10	-10
11	- 9
12	- 8
13	- 7
14	- 6
15	- 5
16	- 4
17	- 3
18	- 2
19	- 1
0	0
1	+ 1
2	+ 2
3	+ 3
4	+ 4
5	+ 5
6	+ 6
7	+ 7
8	+ 8
9	+ 9
190	+10
191	+11
192	+12
193	+13
194	+14
195	+15
196	+16
197	+17
198	+18
199	+19

Similar numbers may be formed for any negative radix system having the numerals {0, 1, ···, |r|-1}. If |r| is represented by R, then the upper and lower limits become:

If n is even:

$$A_{UL} = (R-1)r^{n-2} + (R-1)r^{n-4} + \cdots + (R-1) = \frac{R^n - 1}{R + 1}$$

$$A_{LL} = (R-1)r^{n-1} + (R-1)r^{n-3} + \cdots + (R-1)r = \frac{-R(R^n-1)}{R + 1}$$

If n is odd:

$$A_{UL} = (R-1)r^{n-1} + (R-1)r^{n-3} + \cdots + (R-1) = \frac{R^{n+1} - 1}{R + 1}$$

$$A_{LL} = (R-1)r^{n-2} + (R-1)r^{n-4} + \cdots + (R-1)r = \frac{R(R^{n-1}-1)}{R + 1}$$

Now we consider how fractions are formed. To find the upper and lower limits use the infinite series where the non-zero terms are summed as follows:

$$A_{UL} = (R-1)r^{-2} + (R-1)r^{-4} + (R-1)r^{-6} + \cdots = \frac{1}{R + 1}$$

$$A_{LL} = (R-1)r^{-1} + (R-1)r^{-3} + (R-1)r^{-5} + \cdots = -\frac{R}{R + 1}$$

Here, the range depends only on the radix used, while the number of digits affects the precision. For negabinary, the range expressed in decimal is:

$$\frac{1}{R + 1} = +\frac{1}{3} > A > -\frac{2}{3} = -\frac{R}{R + 1}$$

The proper fractions which lie outside these ranges are represented by numbers with digits to the left of the radix point. If one (1) is added to the upper limit; and subtracted from the lower limit, the corresponding range includes all proper fractions of both polarities:

$$+\frac{4}{3} > A > -\frac{5}{3}$$

If we form all combinations of repeating radix fractions we can find the difference between the true value (infinite series of terms), and the actual value to within an increment (Δa) for a series of m digits. This is represented in Table III below.

TABLE III

Negabinary Repeating Radix Fractions

Special Numbers Form		Increments to obtain true limits for m digits $\Delta a = \frac{1}{3}(-2)^m$	
Limiting Decimal Value	Repeating Radix Fraction	Even m	Odd m
+ 4/3	1.0101···	$-\Delta a$	$2\Delta a$
+ 1	1.0000···	0	0
+ 2/3	1.1111···	Δa	Δa
+ 1/3	1.1010···	$2\Delta a$	$-\Delta a$
+ 1/3	0.0101···	$-\Delta a$	$2\Delta a$
0	0.0000···	0	0
- 1/3	00.1111···	Δa	Δa
- 2/3	00.1010···	$2\Delta a$	$-\Delta a$
- 2/3	11.0101···	$-\Delta a$	$2\Delta a$
- 1	11.0000···	0	0
- 4/3	11.1111···	Δa	Δa
- 5/3	11.1010···	$2\Delta a$	$-\Delta a$

The fact that some fractions (+1/3 and -2/3 in Table III) can be represented in more than one way becomes important in some arithmetic operations, namely, division. Naturally, analogous tables could be worked out for other radix systems.

Before continuing, we should look at one other number system. If we use the base -2 then we must use two unique symbols to

represent the possible numerals. In the negabinary system those numerals are {0, 1}. One other possibility is {0, -1}.

To avoid confusion, use the symbol N to stand for -1. Table IV below shows the counting scheme for both of these binary systems.

TABLE IV

Counting Scheme

BASE -2 (0, 1)	Decimal	BASE -2 (0, -1) or (O, N)
1010	-10	NNNNO
1011	- 9	NNOON
1000	- 8	NNOOO
1001	- 7	NNONN
1110	- 6	NNONO
1111	- 5	NON
1100	- 4	NOO
1101	- 3	NNN
10	- 2	NNO
11	- 1	N
0	0	O
1	1	NN
110	2	NO
111	3	NNON
100	4	NNOO
101	5	NNNN
11010	6	NNNO
11011	7	NOON
11000	8	NOOO
11001	9	NONN
11110	10	NONO

Notice in Table IV that 111 represents a decimal 3 while NNN represents a decimal -3. This leads to naming the new system the cobinary system. It is also interesting to note the arrangement of 0's and 1's in either system. The lowest order digit (value = 1) alternates between the two symbols each time the number increases or decreases. The next digit (value = -2) alternates each time the number increases by two. The next digit (value = 4) changes each time the number increases or decreases by four, and so on for each digit place. Notice, however, that this is exactly what happens in the regular binary system, but in negabinary system the relationship between the digits is changed. For example:

TABLE V

Decimal	Binary	Negabinary
+0	00000	00000
1	00001	00001
2	00010	00110
3	00011	00111
4	00100	00100
5	00101	00101
6	00110	11010
7	00111	11011
8	01000	11000
9	01001	11001
10	01010	11110

In Table V we see the two lowest order digits are the same in either case, but the next two higher order digits have their patterns shifted up two places in the negabinary system;

i.e. the 1's start with +2 and +6 in the negabinary and start with +4 and +8 in the binary. A similiar type of pattern can be observed for any digit position.

Addition and Subtraction

By now we should be able to write down decimal numbers in one of several bases. It appears that there probably is no advantage to writing numbers in this fashion except to be different. Well, the results will begin to show themselves in the following sections where we will consider the arithmetic operations. In demonstrating these arithmetic properties, many examples will be used. The format will be as follows:

Decimal Negadecimal Negabinary

That is, on the left the problem will appear in decimal form. Next to it will be the same problem in negadecimal form, and finally in negabinary form.

The system using a radix of -10, and the normal numerals 0, 1, 2, ···, 9 has the property that a carry-in is subtractive and a borrow is additive. By way of example, consider:

Decimal	Negadecimal
Add + 12	Add 192
+ 19	199
+ 31	171

For the negadecimal: 9 + 2 = 1 carry -1; 9 + 9 - 1 = 7 carry -1; 1 + 1 - 1 = 1.

For decimal addition one must not only perform the steps necessary to get the correct numerical answer but also the correct sign has to be determined. For example:

Decimal	Negadecimal
Add + 12	Add 192
- 19	21
- 7	13

Note that for the negadecimal example there is no sign decision to be made. The two numbers are simply added using the forementioned carry procedure. These operations can be described by the flow charts that follow. The "find" operation is just looking up in a table the correct answer. Figures 1 through 5 show flow charts for addition and subtraction for decimal, negadecimal, and negabinary.

Notice of course that N represents -1, just as before. For example, in negadecimal addition if the carry-in is N(-1), then 5 + 4 - 1 = 8. Likewise, 6 + 5 = N1, or 1 with a carry-out of N(-1) which demonstrates the verbal carry rule stated on the preceeding page. So we see that the tables have both the carry-in and the carry-out digits contained in them. Note

from the flow chart that essentially the same steps appear in all cases, but the regular decimal must find the correct sign for the answer. Now, while this all looks quite favorable toward the negative radix notation, we will see in the next section that there will indeed be other problems involved in actually implementing such a simple design. For example, in the negative base sum and difference tables there are two possibilities for the carry instead of just one.

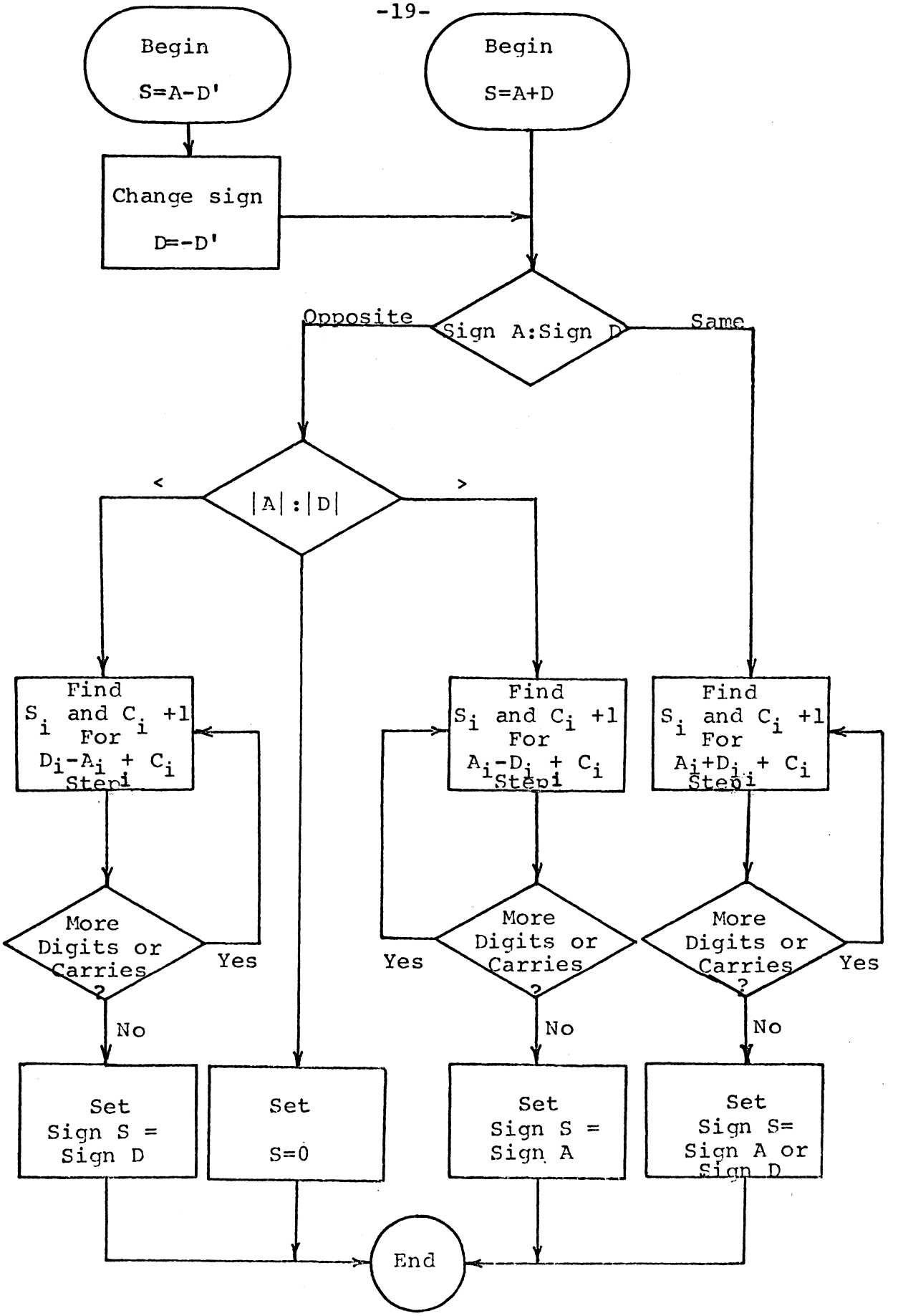

Figure 1. Decimal addition and subtraction

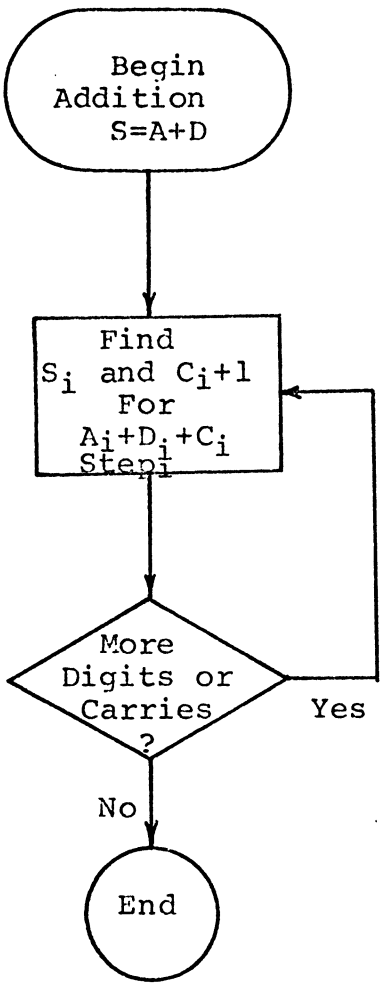

Figure 2.
Negadecimal addition

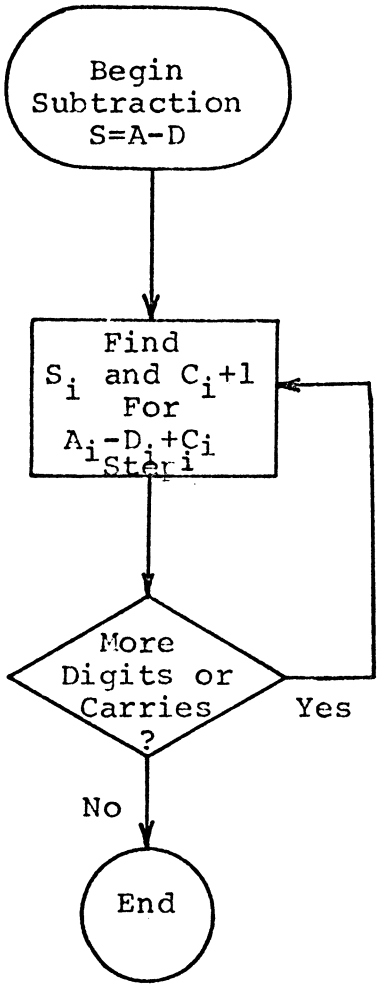

Figure 3.
Negadecimal subtraction

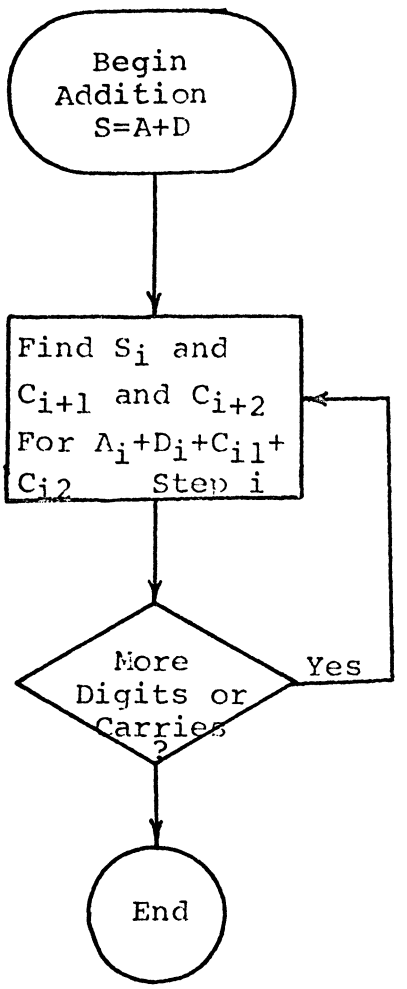

Figure 4.
Negabinary addition

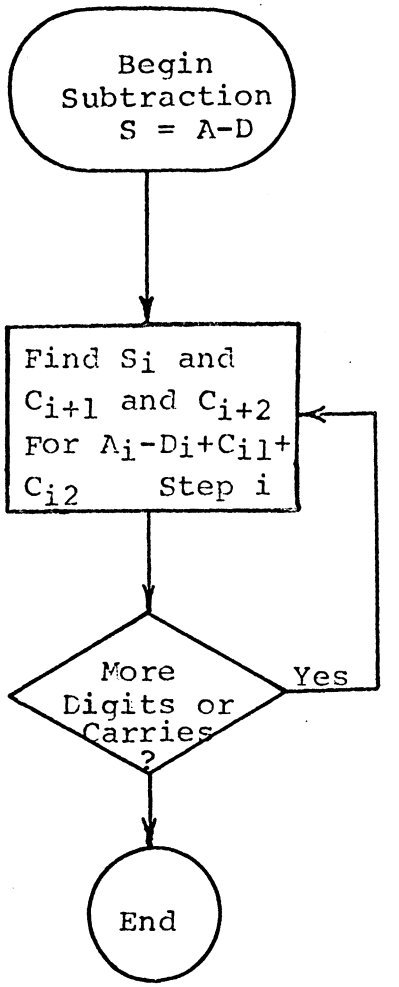

Figure 5.
Negabinary subtraction

TABLE VI

Decimal Digit Sum Table

2nd Operand Digit if Carry is 1	2nd Operand Digit if Carry is 0	First Operand Digit 0	1	2	3	4	5	6	7	8	9
-	0	00	01	02	03	04	05	06	07	08	09
0	1	01	02	03	04	05	06	07	08	09	10
1	2	02	03	04	05	06	07	08	09	10	11
2	3	03	04	05	06	07	08	09	10	11	12
3	4	04	05	06	07	08	09	10	11	12	13
4	5	05	06	07	08	09	10	11	12	13	14
5	6	06	07	08	09	10	11	12	13	14	15
6	7	07	08	09	10	11	12	13	14	15	16
7	8	08	09	10	11	12	13	14	15	16	17
8	9	09	10	11	12	13	14	15	16	17	18
9	-	10	11	12	13	14	15	16	17	18	19

TABLE VII

Decimal Digit Difference Table

Subtractive Digit if Carry is N	Subtractive Digit if Carry is O	Additive Digit 0	1	2	3	4	5	6	7	8	9
-	0	00	01	02	03	04	05	06	07	08	09
0	1	N9	00	01	02	03	04	05	06	07	08
1	2	N8	N9	00	01	02	03	04	05	06	07
2	3	N7	N8	N9	00	01	02	03	04	05	06
3	4	N6	N7	N8	N9	00	01	02	03	04	05
4	5	N5	N6	N7	N8	N9	00	01	02	03	04
5	6	N4	N5	N6	N7	N8	N9	00	01	02	03
6	7	N3	N4	N5	N6	N7	N8	N9	00	01	02
7	8	N2	N3	N4	N5	N6	N7	N8	N9	00	01
8	9	N1	N2	N3	N4	N5	N6	N7	N8	N9	00
9	-	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9

TABLE VIII

Negadecimal Digit Sum Table

Addend digit if carry is			Augend Digit									
N	1	0	0	1	2	3	4	5	6	7	8	9
0	-	-	19	00	01	02	03	04	05	06	07	08
1	-	0	00	01	02	03	04	05	06	07	08	09
2	0	1	01	02	03	04	05	06	07	08	09	N0
3	1	2	02	03	04	05	06	07	08	09	N0	N1
4	2	3	03	04	05	06	07	08	09	N0	N1	N2
5	3	4	04	05	06	07	08	09	N0	N1	N2	N3
6	4	5	05	06	07	08	09	N0	N1	N2	N3	N4
7	5	6	06	07	08	09	N0	N1	N2	N3	N4	N5
8	6	7	07	08	09	N0	N1	N2	N3	N4	N5	N6
9	7	8	08	09	N0	N1	N2	N3	N4	N5	N6	N7
-	8	9	09	N0	N1	N2	N3	N4	N5	N6	N7	N8
-	9	-	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9

TABLE IX

Negadecimal Digit Difference Table

Subtrahend Digit if Carry is			Minuend Digit									
N	1	0	0	1	2	3	4	5	6	7	8	9
-	0	-	01	02	03	04	05	06	07	08	09	N0
-	1	0	00	01	02	03	04	05	06	07	08	09
0	2	1	19	00	01	02	03	04	05	06	07	08
1	3	2	18	19	00	01	02	03	04	05	06	07
2	4	3	17	18	19	00	01	02	03	04	05	06
3	5	4	16	17	18	19	00	01	02	03	04	05
4	6	5	15	16	17	18	19	00	01	02	03	04
5	7	6	14	15	16	17	18	19	00	01	02	03
6	8	7	13	14	15	16	17	18	19	00	01	02
7	9	8	12	13	14	15	16	17	18	19	00	01
8	-	9	11	12	13	14	15	16	17	18	19	00
9	-	-	10	11	12	13	14	15	16	17	18	19

TABLE X

Negabinary Digit Sum

Addend Digit if Carry is			Augend Digit	
1	1	0	0	1
-	-	0	0	1
-	0	1	1	110
0	1	-	110	111
- 1	-	-	111	100

TABLE XI

Negabinary Digit Difference

Subtrahend Digit if Carry is			Minuend Digit	
1	1	0	0	1
-	1	0	0	1
-	-	1	11	0
0	-	-	110	111
1	0	-	1	110

Use of the flow charts together with the tables should allow anyone to do addition and subtraction problems. The next section then treats these problems.

Addition Examples

The possibilities that we would like to consider are summarized in Table XII. Here the first column gives the example number, the second and third columns refer to sign

of the augend (A) and addend (D) and the fourth column gives the relative magnitudes; the "don't care" entries indicate that differences in magnitude do not provide any additional variations when the polarities of A and D are the same.

TABLE XII

Variations in Decimal Addition

Example No.	Polarity of A	Polarity of D	A:D
1	+	+	Don't care
2	-	-	Don't care
3	+	-	>
4	+	-	=
5	+	-	<
6	-	+	>
7	-	+	=
8	-	+	<

Example 1 (+, +, don't care):

+147	267	110010111
+ 82	122	1010110
+229	389	100100101

Example 2 (-, -, don't care):

-147	1953	10111101
- 82	98	11110010
-229	1831	1100101111

Example 3 (+, -, >):

+147	267	110010111
- 82	98	11110010
+ 65	145	001000001

Example 4 (+, -, =):

+147	267	110010111
-147	1953	10111101
000	0000	000000000

Example 5 (+, -, <):

+ 82	122	1010110
-147	1953	10111101
- 65	0075	11000011

Example 6 (-, +, >):

-147	1953	10111101
+ 82	122	1010110
- 65	0075	11000011

Example 7 (-, +, =):

-82	98	11110010
+82	122	1010110
00	000	00000000

Example 8 (-, +, <):

- 82	98	11110010
+147	267	110010111
+ 65	145	001000001

Several things should be noted about these examples: The calculation of the decimal problems requires that you must mentally organize the problem into a form that you are used to seeing. Some of the examples required that mentally there

was subtraction involved rather than addition. However, the negadecimal and negabinary examples were all worked the same way. Actually, the example numbers for negabinary were so big that you mentally didn't even try to interpret the problem, but simply, once given the string of 0's and 1's, proceeded to go by the addition rules.

Subtraction Examples

Refer to Table XIII for a summary of the example signs and mangitudes. Here, M is the minuend and S is the subtrahend and notice that there are two more examples that result from the elimination of the "don't care" type of problem in subtraction. Also, this time we will eliminate the negadecimal solution to the problems.

TABLE XIII

Variations in Decimal Subtraction

Example	Polarity of M	Polarity of S	M:S
1	+	+	>
2	+	+	=
3	+	+	<
4	-	-	>
5	-	-	=
6	-	-	<
7	+	-	>
8	+	-	<
9	-	+	>
10	-	+	<

Example 1 (+, +, >):

+4	100
+3	111
+1	001

Example 2 (+, +, =):

+3	111
+3	111
0	000

Example 3 (+, +, <):

+3	111
+4	100
-1	011

Example 4 (-, -, >):

-3	1101
-2	0010
-1	0011

Example 5 (-, -, =):

-3	1101
-3	1101
0	0000

Example 6 (-, -, <):

-2	0010
-3	1101
+1	0001

Example 7 (+, -, >):

+3	111
-2	010
+5	101

Example 8 (+, -, <):

+2	0110
-3	1101
+5	0101

Example 9 (-, +, >):

-3	1101
+2	0110
-5	1111

Example 10 (-, +, <):

-2	0010
+3	0111
-5	1111

The subtraction examples illustrate the ease of implementing the subtraction algorithm given in Figure 5 and Table XI. Negative radix subtraction is pure subtraction--there is nothing to do except begin with the lowest order pair of digits and apply the subtraction table. The operation is complete when there are no more digits or carries. The practical potential for addition and subtraction then, lies with using a machine since a digit-at-a-time arrangement

without first checking magnitude and sign is indeed a good thing to have in a machine. Again, the hardware to implement such an arrangement will be more complex, but a savings will occur because there are no operations needed at the word or register level as all manipulation is at the digit level. Thus the use of integrated circuits looks very promising.

Multiplication

Multiplication is actually about the easiest operation to perform with negabinary, and negadecimal multiplication is very much like its decimal counterpart. Figure 6 indicates a flow chart for both negative radix multiplication and decimal multiplication. Note that the only difference between the two methods is the choosing of the appropriate sign for decimal problems. Negabinary multiplication follows the rules in Table XIV. Negadecimal multiplication follows the same rules as decimal multiplication except that the sign of the carry is changed. The only problem comes when one wants to write both of the two product digits which are formed when a multiplier digit and the most significant multiplicand digit are multiplied together. This is all right in decimal but not in negadecimal. For example:

+147	267
+ 3	7
+441	19189

(1) (7 x 7) = 9, carry -4

(2) (7 x 6) - 4 = 8, carry -3

(3) (7 x 2) - 3 = 1, carry -1

(4) 0 - 1 = 9, carry +1 } not (7 x 2) -3 = 11

(5) 0 + 1 = 1

In other words, the carry must always be considered since it is a negative carry.

Multiplication Examples

For multiplication there are only four cases to be considered--(+, +), (+, -), (-, +), and (-, -).

Example 1 (+, +):

+18	198	10110
+ 7	7	11011
+126	286	10110
		10110
		00110010
		10110000
		010100010
		101100000
		0110000010

Example 2 (+, -):

+18	198	10110
- 7	13	1001
-126	154	10110
	1980	1011000
	1934	010000110

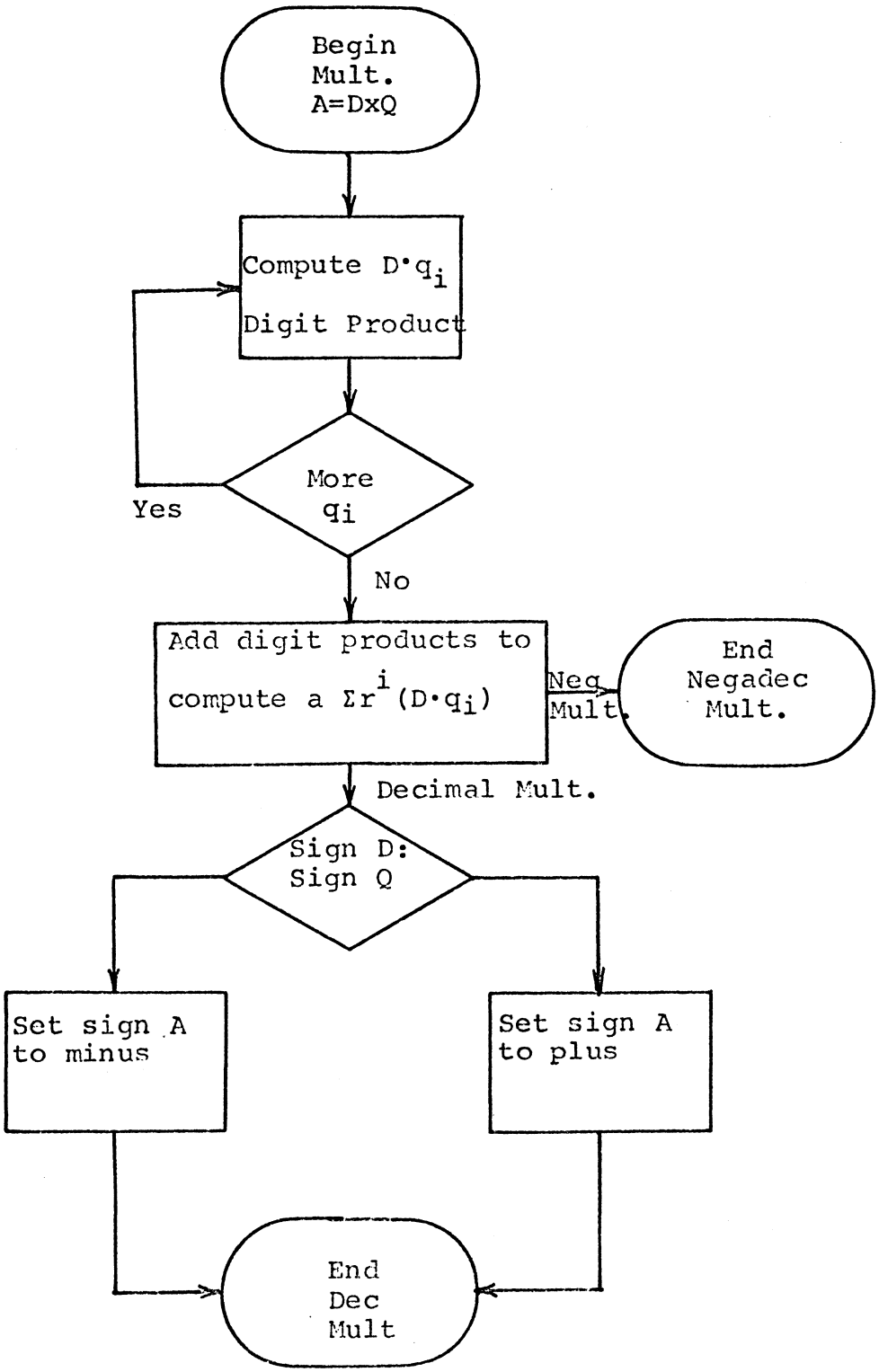

Figure 6. Flow Chart for Multiplication.

TABLE XIV

Negabinary Multiplication

No Carries	0	1
0	0	0
1	0	1

Example 3 (-, +):

-18	22	110010
+ 7	7	11011
-126	1934	110010
		110010
		0010110
		110010000
		111100110
		1100100000
		0010000110

Example 4 (-, -):

-18	22	110010
- 7	13	1001
+126	66	110010
	22	110010
	286	110000010

So from the examples it should be clear how multiplication works. The best way to deal with the negabinary examples, and in actual machine use, is to sum after each $D \cdot q_i$ partial product is formed so that all operations are kept simple.

Other Operations

There are ways to define other operations that apply only to the negative base number systems. Referring back to Table IV in which we used (0, -1) as the numerals for the base -2 system, we see that the numbers in this system are just the complements of the regular negabinary numbers. It turns out that if you use negative numerals in place of positive numerals, the number formed will indeed be the complement of the original number. To prove this, refer to the polynomial representation.

$$a_{n-1}r^{n-1} + \cdots + a_2r^2 + a_1r + a_0$$

For any negative base we have:

sign of each position
for positive numerals $\cdots -a_3 + a_2 - a_1 + a_0$

sign of each position
for negative numerals $\cdots + a_3 - a_2 + a_1 - a_0$

What we have then is the same number but with a different sign since the absolute numerical value of each bit position remains the same. For negabinary this is easy to do--just replace the 1's with N's. For negadecimal replace the +a's with -a's.

Complement

If we wanted to find the complement of a number, that is, the opposite sign of the number using the same numerals, then for normal decimal numbers we would simply change the sign. For negaradix numbers we must subtract from zero. This is accomplished by using the subtraction tables used earlier and noting that there will never be a double carry.

For example:

Negadecimal

0	00000	0	00000
-(-2382)	3798	-(2382)	18422
+ 2382	18422	(-2382)	3798

Negabinary

0	000000	0	000000
-(19)	010111	-(-19)	111101
(-19)	111101	19	010111

Co-addition

In co-addition, an operation table is defined such that when two numbers are added the result is the complement of the answer obtained by normal addition. Use the symbol $\oplus$ to represent the co-addition operation, and this will be called the "co-sum". Thus the co-sum of A and B is defined as:

$$R = A \oplus B = - (A + B) = (-A) + (-B)$$

Naturally, this is very simple in terms of normal decimal addition, but because the complement of a negaradix number looks much different a new co-sum table is required, and this is shown in Table XV.

TABLE XV

Negabinary Digit Co-sum Table

Carry Digit			
1	0	0	1
0	-	1	0
1	0	0	11
-	1	11	10

Some examples follow:

Example 1

5	0101
$\oplus$3	0111
(-8)	1000

Example 2

5	0101
$\oplus$ (-8)	1000
3	0111

Example 3

3	0111
⊕ (-8)	1000
5	0101

Example 4

-3	01101
⊕ (-5)	01111
8	11000

Since there is no -1 carry, only one carry signal is required and thus a full negabinary co-adder has a structure like that of a regular binary adder. If the ⊕ represents an adder module, as we shall see in Chapter III, then addition and subtraction can be provided in the following ways:

$$A + B = (A \oplus B) \oplus 0 = -(A+B) \oplus 0 = -(-\{A+B\}+0) = A + B$$

$$A - B = (A \oplus 0) \oplus B = -A \oplus B = -(-A + B) = A - B$$

$$A + B - C = (A \oplus B) \oplus C = -(-\{A+B\} + C) = A + B - C$$

$$A + B + C + D = (A \oplus B) \oplus (C \oplus D)$$

Thus there are many possible arrangements of ⊕ modules to add and subtract numbers. Since the numbers can have either polarity we still preserve the benefits derived by using negative radix notation.

The $\oplus$ operation has the following properties--inverse, commutative, no identity, not associative.

Inverse,

$$A \oplus (-A) = - (A + \{-A\}) = -A + A = 0$$

Commutative,

$$A \oplus B = -(A + B) = -(B + A) = B \oplus A$$

Refer to Table XV

Co-multiplication

Co-multiplication is defined as the operation which produces the additive inverse of the product of two numbers. The symbol $\otimes$ will be used for this operation and it will be called the "co-product".

$$R = A \otimes B = - (A \times B) = (-A) \cdot (B) = (A) \cdot (-B)$$

Thus Table XVI contains the negabinary co-product table.

TABLE XVI

Negabinary Digit Co-product Table

Carry 1	0	0	1
-	0	0	0
-	1	0	11
0	-	1	1
1	-	1	0

Since we do not want to use regular addition we will have to use co-addition when adding the partial co-products. This will give us the correct answer without having to change the sign of the co-product. In other words, if $R = A \otimes B$ is formed using co-addition, then $R = (A \otimes B) = A \otimes B$. Remember, though, that in using co-addition, the co-addition must be done via $(A \oplus B) \oplus 0$. For example, we will show an addition example and then some multiplication examples.

Example 1

5	00101	A
+6	11010	$\oplus$ B
11	110101	A $\oplus$ B
	000000	$\oplus$ 0
	011111	$(A \oplus B) \oplus 0 = A + B$

Applying these same ideas to the following, we have:

Example 2

2	110	A	A x B = R
x3	111	B	
6	0010	$A \cdot q_1 = C_1$	
	0100	$A \cdot q_2 = C_2$	
	0010	$C_1 \oplus C_2 = D_1$	
	0000	$D_1 \oplus 0 = E_1$	
	00110	E_1	
	01000	$A \cdot q_3 = C_3$	
	11010	$E_1 \oplus C_3 = F_1$	
	00000	$F_1 \oplus 0$	
	01110	-R	
	00000		
	11010	R	

Example 3

3	0111	
-2	10	
-6	11010	-R
	00000	
	01110	R

There are two things to notice about these examples. First, since we want to add our q_i partial co-product results as we go along, to a running total, we must have several additions with zero. This result is then of the wrong sign (since

each q_i partial co-product changed the sign of its result to be added to the running sum) and a final sign inversion (or complementation) must be performed to obtain the correct answer. While this seems rather boring to perform on these easy examples it should be remembered that an actual machine could do these operations very rapidly. We are now prepared to write revised flow charts for these co-multiplication and co-addition operations. See Figures 7, 8, and 9. It will be seen in the next chapter that while either negabinary sum or negabinary co-sum seems relatively easy to understand from a manual viewpoint there will be definite machine advantages to the methods presented in the later portions of this chapter.

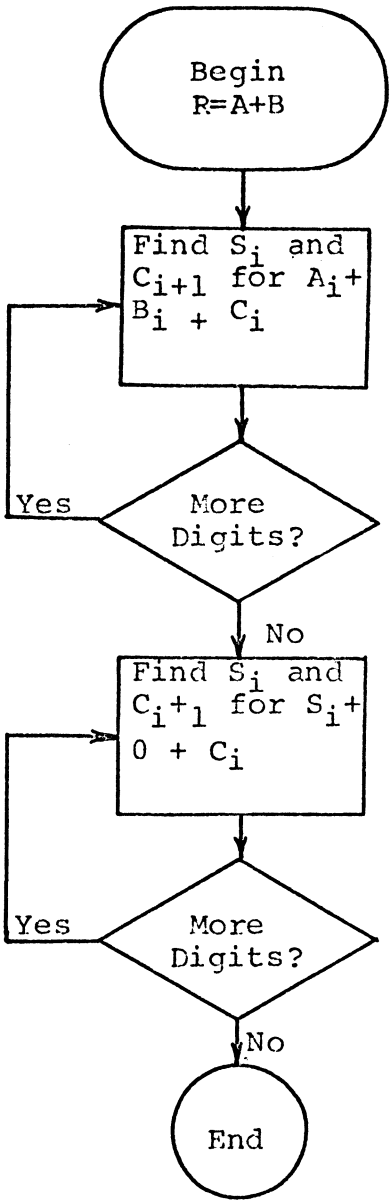

Figure 7. Negabinary addition using co-addition.

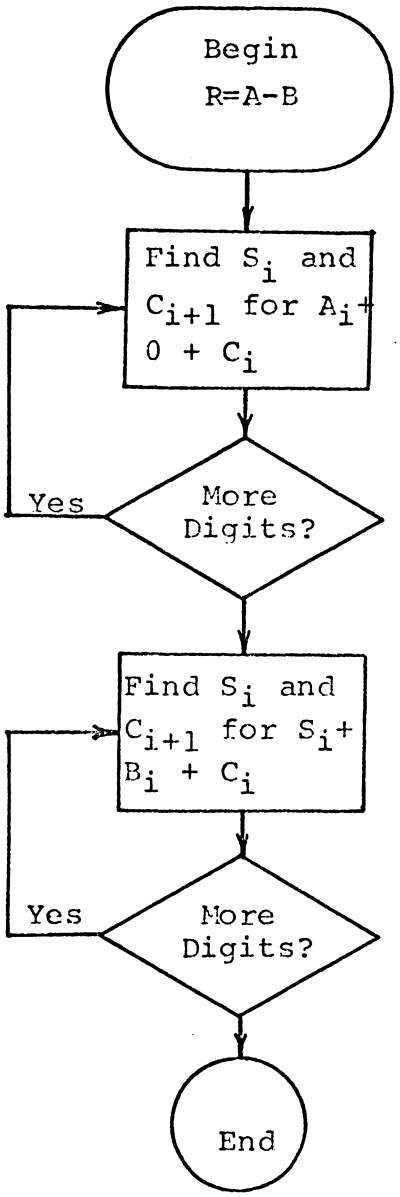

Figure 8. Negabinary subtraction using co-addition.

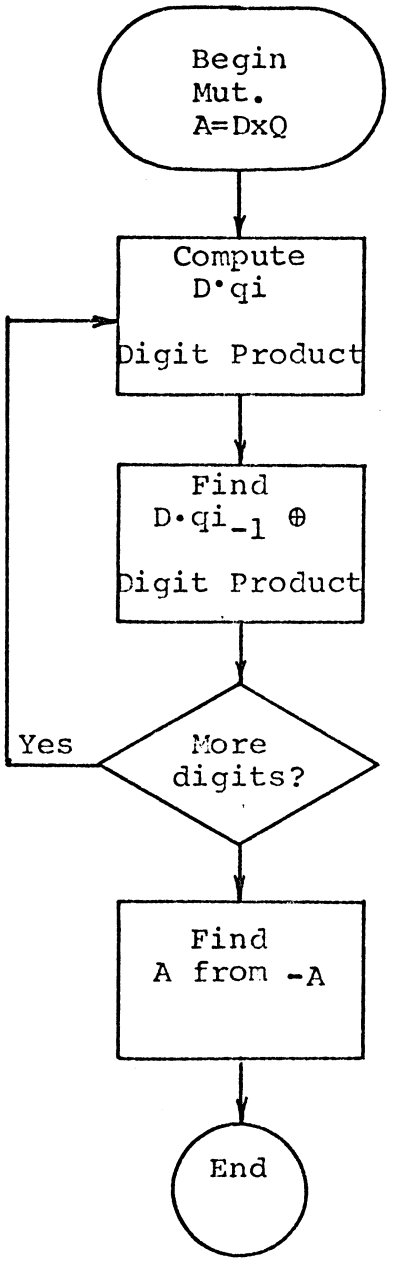

Figure 9. Multiplication using co-multiplication and co-addition.

CHAPTER III

SOME LOGIC REALIZATIONS

Introduction

After deciding that a computer using base -2 is really a serious thing we now want to try and implement some block diagrams and flow charts for performing the basic operations--addition, subtraction, and multiplication. It appears that regardless of the binary number system used, storage, retrieval, and shifting of information can be handled by presently available hardware. That is, transferring the information inside the computer would not differ from the present methods. Once the information is in a register then the differences in operation between different number systems come fully into play. It is not the purpose of this chapter, or this paper, for that matter, to exploit every detail of all the possible combinations of ways to accomplish the implementation of arithmetic operations using the previously defined numbering system. Rather, a few demonstrations will convince the reader that such systems are practical. Certainly as mentioned in Chapter I, the level of sophistication of the early computers were not what they are today, and the present subject of using -2 as a base is still in the infant stages of development. The emphasis, therefore, is not sophisticated

beauty but rather, simply, realization of the mathematics presented in the last chapter. Let us first look at some of the old operations performed with the "new" arithmetic.

Addition

Table X is again reproduced below for easy reference. From Table X the Boolean expressions to realize the table could be written.

TABLE X

Negabinary Digit Sum

Addend if Carry is			Augend Digit	
2	1	0	0	1
-	-	0	0	1
-	0	1	1	110
0	1	-	110	111
1	-	-	111	100

If the carry bit that transfers only one place to the left is called C and the carry bit that transfers two places to the left is called D, and A and B are the two numbers to be added, then Table XVII can be formed for the sum with no carry.

TABLE XVII

Negabinary Sum Truth Table--No Carry

A	B	Sum	C	D
0	0	0	0	0
0	1	1	0	0
1	0	1	0	0
1	1	0	1	1

It appears that Table XVII would not be particularly difficult to implement, and this is certainly true. As we saw in the addition examples in the last chapter, however, the fact that there are two carries results in the possible build up of two carries as seen in the left-most column of Table X. The logic can of course be realized but there are too many gates for good reliability and low cost. A better solution to the problem is the use of the co-adder modules and the truth table (Table XVIII) shown below.

TABLE XVIII

Co-addition Truth Table

A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	0	0	1
0	0	1	1	0
0	1	1	0	0
1	0	1	0	0
1	1	1	1	1

The circuit for this is very much like that for a full adder using the normal binary numbers. Figure 10 shows such a co-adder.

Now to perform the addition operation requires that A + B = (A ⊕ B) ⊕ 0, which can be accomplished as follows. On the first clock pulse enter A and B into the co-adder and store the results -(A+B), in register X. On the second clock pulse enter X and Zero into the second co-adder and put the results in the output register. In block diagram form it looks like Figure 11 below.

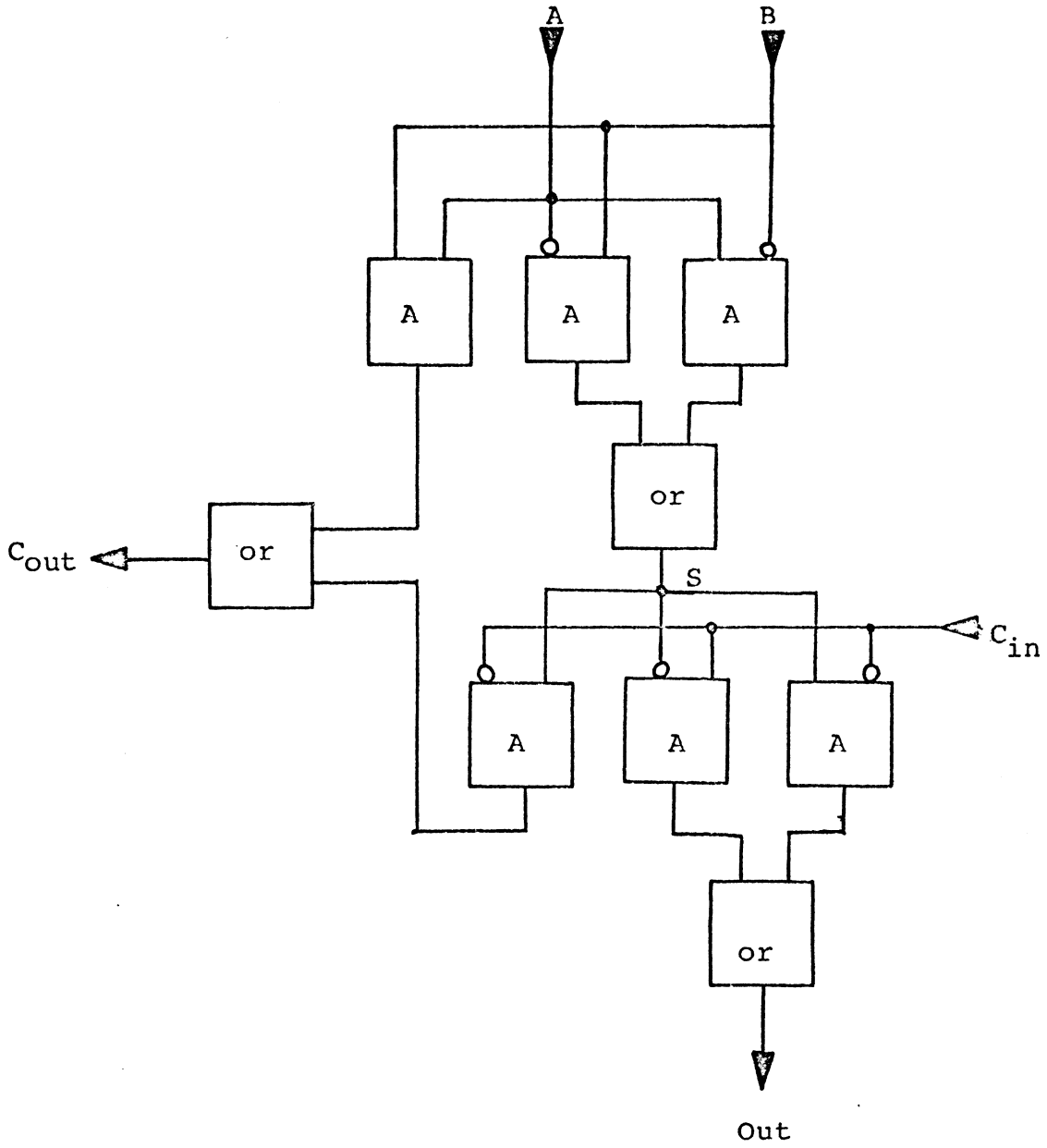

Figure 10. Block Diagram of a Co-Adder Module

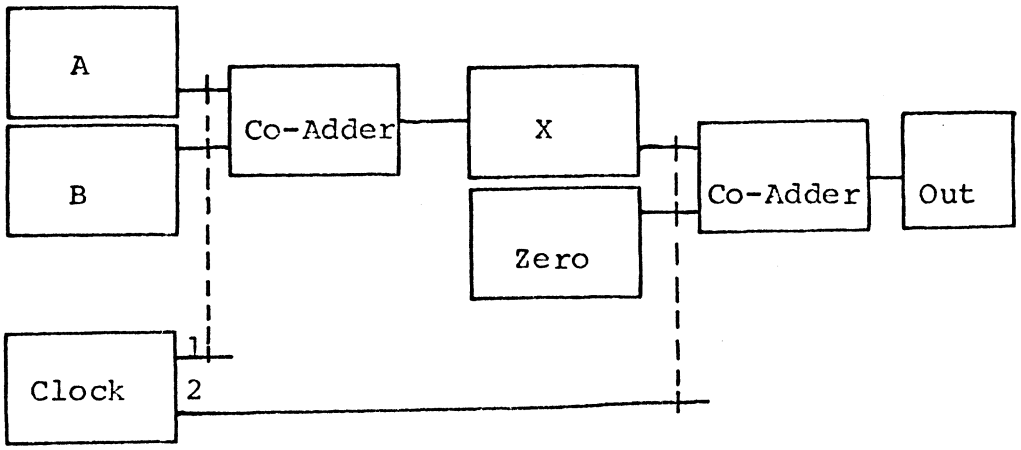

Figure 11. Block diagram for full addition.

Remember that for subtraction using the co-adder scheme all that needs to be done is revise the formula to read A - B = (A ⊕ 0) ⊕ B. This is shown in Figure 12.

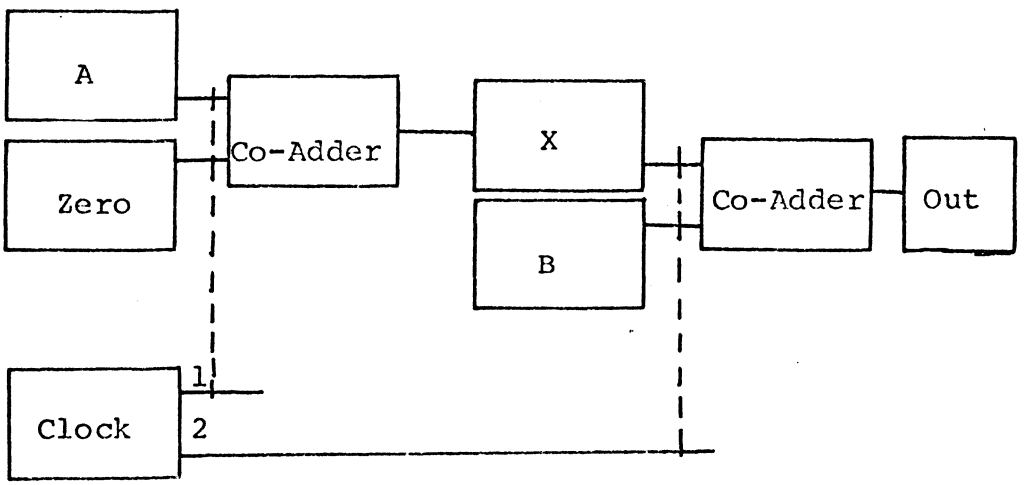

Figure 12. Block diagram for full subtraction.

Notice how simple and easy these operations are--both full addition or full subtraction can be accomplished in just two clock pulses. However, remember that this is just one way to solve the problem. Obviously there are other solutions that may be faster or possess other advantages applicable to a specific type of machine.

Multiplication

The next problem is that of multiplication. From previous work it appears that the simpliest way to do this is to obtain the partial co-product of the digit bits and then co-add the results to arrive at an answer. In words, the operation might proceed as follows. Put the first number to be multiplied in register A and the multiplier in register B. Now begin left shifting the number in B, out of B, and into register M until the first 1 is encountered in the M register (this is the most significant). Obtain the partial co-product and store it in register N. Shift M and N once to the left, check to see if the bit in register M is a one, and if it is, find the partial co-product, storing it in register P. Now co-add N and P and return it to N. Shift M and N, find the partial co-product of M and A if M is a one, store in P, co-add N and P and store in N. Continue until B is empty, that is, count the shifts and stop when m equals the number of places in the register. Now do the final complementation on the answer since R = -(A x B) when co-multiplication is used.

As an example, consider the following.

Example 1 (Refer to Tables XV and XVI)

```
 5    A   101    A  0101        A  0101
x2    B   110    M    100       M   110

                  N  1111        N 11110
                                 P 01111

                                   01111
                                   00000

                                 N 00101
```

```
     A   0101   A  0101
     M    110   M   110

     N   0101   N  1010        N  1010
                               P  0000

                                  1110 = Ten
```

In order to provide a circuit that would perform the co-multiply operation, we need to refer to a truth table based on Table XVI. This is indicated in Table XIX.

TABLE XIX

Truth Table for Co-Multiplication

A	M	Carry In	P	Carry Out
0	0	0	0	0
0	1	0	0	0
1	0	0	0	0
1	1	0	1	1
0	0	1	1	0
0	1	1	1	0
1	0	1	1	0
1	1	1	0	0

However since we only want to multiply when the M digit is a one, lines one, three, five, and seven can be removed. The result is shown in Table X X.

TABLE X X

Reduced Truth Table for Co-Multiplication

A	Carry in	P	Carry out
0	0	0	0
1	0	1	1
0	1	1	0
1	1	0	0

Table X X indicates that a circuit that would perform the co-multiplication operation would be the exclusive--or gate arrangement as shown below in Figure 13.

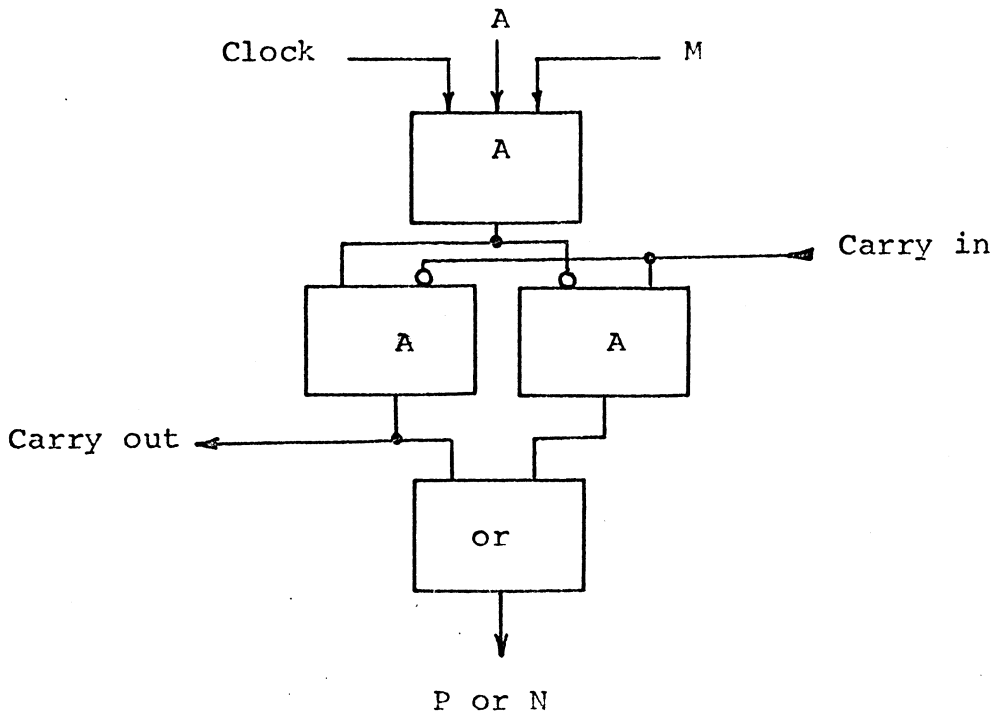

Figure 13 Co-Multiplication Gate Network

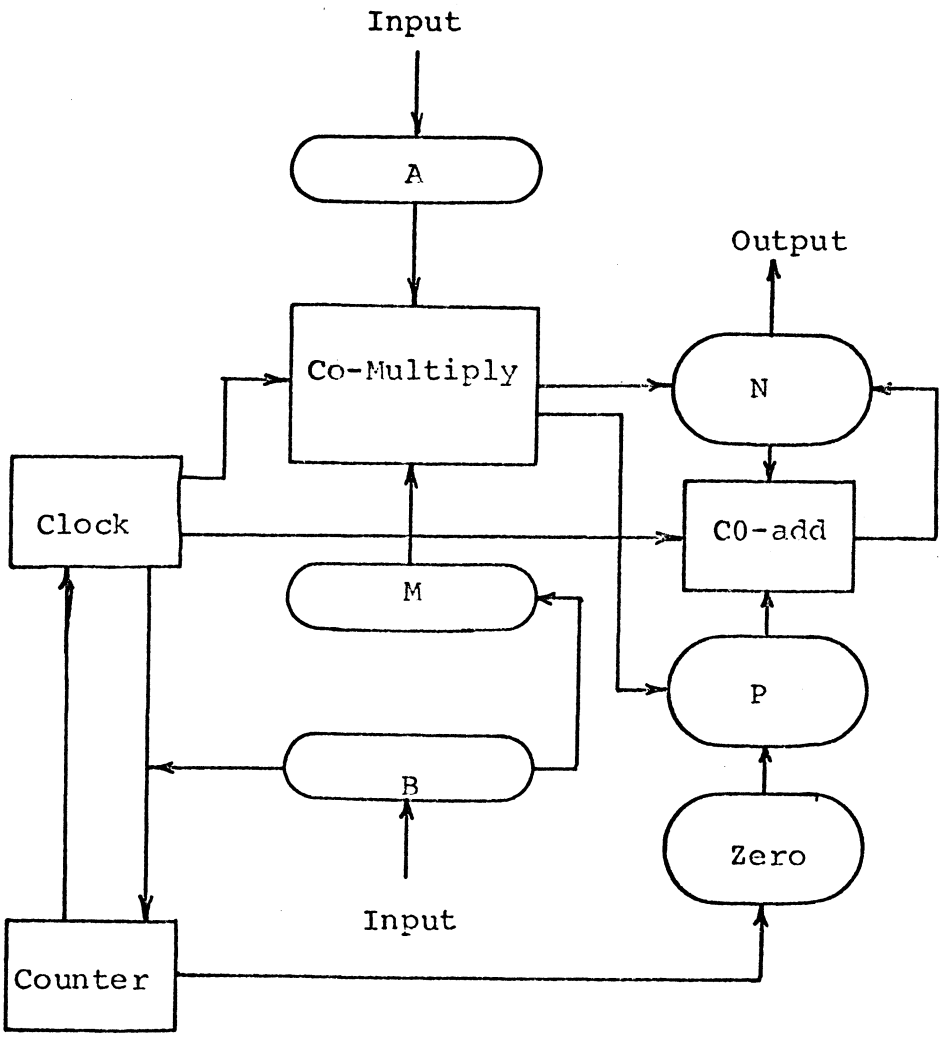

Figure 14. Block diagram for Multiplication of Negabinary Numbers by Co-products and Co-addition.

Notice that the M digit is brought in to turn the co-multiplier gate on and off. So now the final diagram for performing multiplication can be made from the above descriptions of the operation. This is found in Figure 14.

Of course there are other easier ways to multiply. One way is simply to transfer A into N if the digit under consideration in M is one. Then arrive at the result using co-addition.

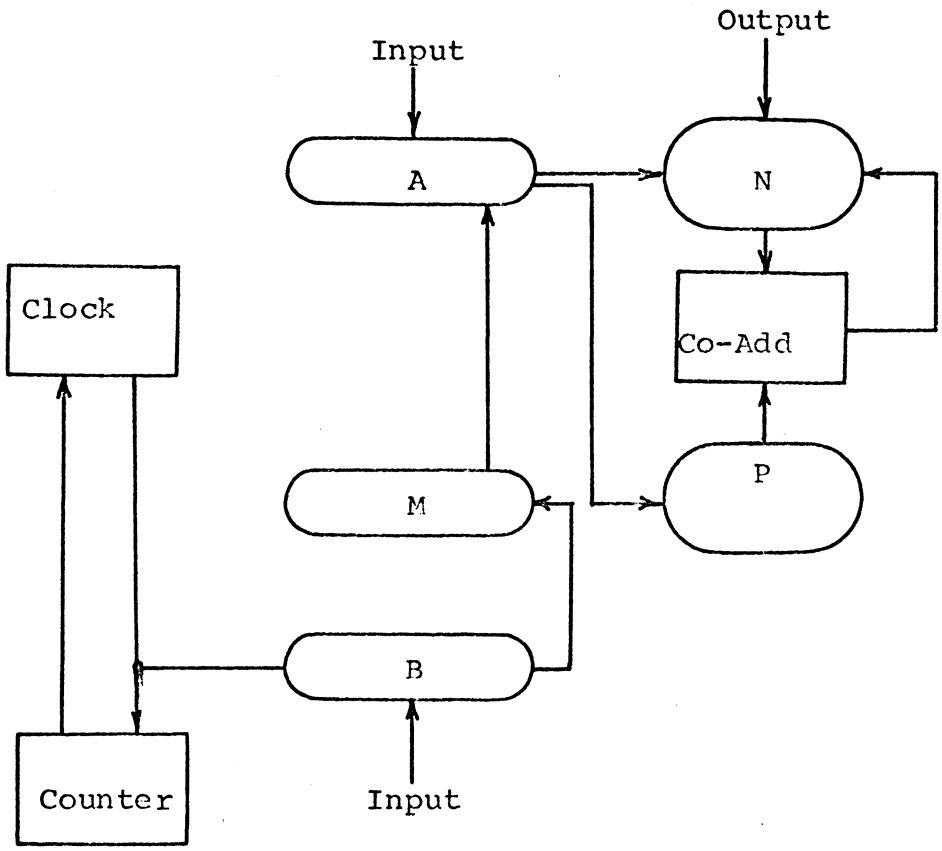

Figure 15. Block diagram for Multiplication of Negabinary Numbers by Co-addition.

This would be almost analogous to the normal accumulator method of multiplication with binary numbers, but, of course, there would be no concern for the polarity of a number with the negabinary number system. A block diagram for this method of multiplication is shown in Figure 15.

Input-Output

Now that the arithmetic operations have been performed the next thing that should be considered is the input and output of the numbers that somehow have to get in and out of the various registers. The easiest way to translate the decimal numbers into the negabinary numbers is through a diode matrix. Of course, this gets quite complicated for large numbers so that a negabinary coded decimal (NBCD) arrangement could be used. Again, many other schemes could be used to solve the problem but one method is shown in Figure 16 for the coding of the decimals -9 to +9.

Basically, the input-output problem is similiar to the same problem in the present computers. Remember that when an output is needed the sign of the number can be obtained by checking to see if the highest order bit is either in an odd or even bit position in the number.

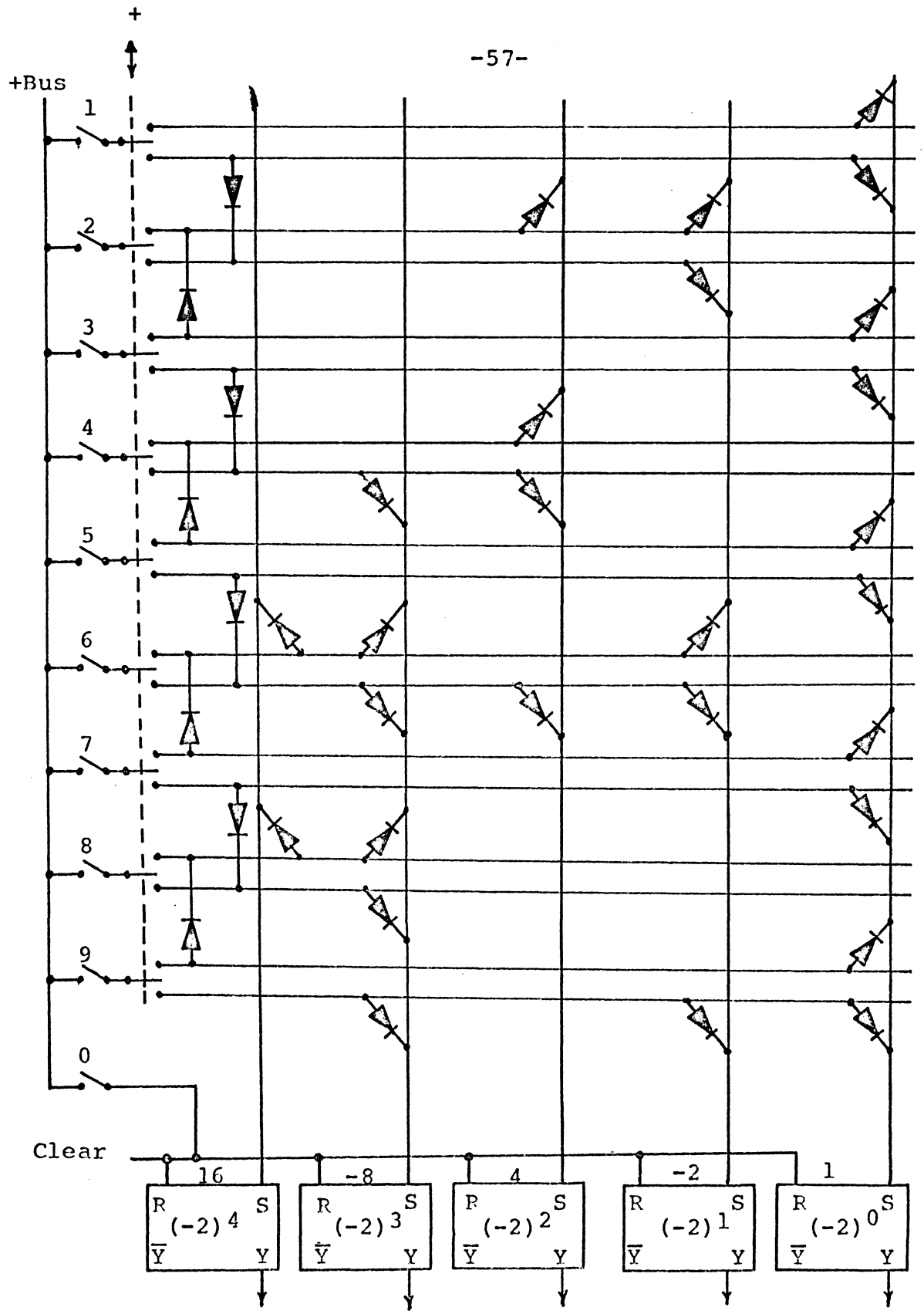

Figure 16. Negabinary coding of the decimal numbers from -9 to +9.

Other operations

The operations of shifting and storage are no different than present machines and present no difficulty for machine implementation. Counters do not necessarily have to count in the negabinary mode although if the input to a computer came through an analog-to-digital converter such a converter would probably use such a counting scheme. The address and control portions probably would remain in their present format, since they really have no number in an arithmetic sense associated with them.

Conclusion

We see then that a machine that makes use of the tools presented in this chapter do not necessarily have to differ greatly from the present systems. It might be described as an application of the unwritten "beauty of mathematics" law that something so seemingly different, so strange to think about, so much unlike anything humans do in their day to day contact with mathematics, could be reduced to a problem of minor changes in our thinking patterns and working machines.

CHAPTER IV

SUMMARY AND CONCLUSIONS

Summary

Some of the mathematical concepts behind the negative base number systems were presented in Chapter II. Special emphasis was placed on the negative based binary system since it is most useful to the application of computers and logic circuits. Chapter III demonstrated that logic circuits can be built with relatively minor changes to perform the arithmetic operations required by the new number system. Probably the most notable aspects of the new system may not be readily apparent from the previous discussions.

Conclusions

As previously discussed the advantages and disadvantages of making a computer in the large scale integration are present right now. The negabinary number system helps bring out the advantages of integrated circuits because no knowledge of the number is needed before an arithmetic operation is begun. In all present day computers, for example, the subtract operation involves complementing of some sort, end around carries, and other problems which use valuable processing

time. It is hoped that the discussions previously have shown the ease of coping with some of these problems using the negabinary system. Another significant factor is the fact that, since there is no separate sign bit, it becomes possible to chain several accumulator registers together to any desired length without worrying about end-around carries, overflows, numerical shifts, etc. Along the same lines it becomes just as easy to partition a register as lengthen it. These simple matters can now be made programmable. It appears that these concepts will make a small, fully software programmable computer a much more attractive proposition where as presently a larger instrument would have to be employed. With regard to integrated circuits then, it is possible to put an extremely long register on a single wafer with all the necessary interconnections on the chip and then program the way the register will be divided when it is actually put in use at the customer's location. This lowers production cost and raises quality since if it is found that a single module of a chip is not functional then the programmer can simply bypass that module. It can be seen then that a complete computer could be built with few high level connections and a great number of low level chip connections and just a few varieties of chips. Again, lower cost and higher reliability.

Areas for Further Study

As pointed out in earlier chapters there are many possibilities left to realize a more optimum machine than are presented here. Certainly the biggest area left unexplored is that of division. Space simply did not allow such considerations to be included in this paper. The area concerned with which combinations of numbers to choose for multiplication and/or division is left largly undone. For example, might co-multiplication have been more attractive if negative numbers, instead of positive numbers, had been used? This and numerous other examples could be cited to show that the area of investigation using a negative basis for a number system is largly unexplored.

Hopefully this paper has presented a background for the uninitiated and a specific base for those who might want to persue these matters further.

CHAPTER V

BIBLIOGRAPHY

1. Benrey, Ronald, Understanding Digital Computers; John F. Fider Publisher, Inc., New York, 1964.

2. Buchholz, W., Planning A Computer; McGraw-Hill Book Company, New York, 1962.

3. Corliss, William R., Computers, United States Atomic Energy Commission, Oak Ridge, Tennessee, July, 1967.

4. Frankel, S. P., The Logical Design of a Simple General Purpose Computer; IRE Transactions on Electronic Computers, Vol. EC-6, March, 1957.

5. Garner, Harvey L., "Number Systems and Arithmetic," Advances in Computers, edited by F. L. Alt and M. Rubinoff, Vol. 6, pp. 131-194, Academic Press, New York, 1965.

6. Lazarkiewicz, A. and Balaskinski, "A Simple Experimental Computer with Negative Basis," Mathematics of Computation, Vol. 15, No. 75, pp. 275-285, July, 1961.

7. Millman, J. and H. Taub, Pulse and Digital Circuits; McGraw-Hill Book Company, New York, 1956.

8. Phister, M., Logical Design of Digital Computers; John Wiley & Sons, New York, 1958.

9. de Regt, M. P., "Negative Radix Arithmetic," Computer Design, Vol. 6, No. 5, pp. 52-63, No. 6, pp. 56-65, No. 7, pp. 36-43, No. 8, pp. 36-39, Vol. 7, No. 1, pp. 62-66, May, 1967 to January, 1968.

10. Scott, N. R., Analog and Digital Computer Technology; McGraw-Hill Book Company, New York, 1960.

11. Siegel, P., Understanding Digital Computers; John Wiley and Sons, New York, 1961.

12. Smith, Charles V. L., Electronic Digital Computers; McGraw-Hill Book Company, Inc., New York, 1959.

13. Wadel, L. B., "Negative Base Number Systems," IRE Trans. on Electronic Computers, Vol. EC-6, p. 123, June, 1957.

14. Ware, Willis H., Digital Computer Technology and Design, Volume II: Circuits and Machine Design; John Wiley and Sons, Inc., New York, 1963.

The vita has been removed from the scanned document

USING -2 AS A BASE FOR A NUMBER SYSTEM TO REALIZE A COMPUTER

by

J. F. Zimmer

ABSTRACT

The first section demonstrates the mathematical feasibility of using -2 for a base as a possible number system in a computer. All arithematic operations are defined and many examples are given. The second section shows how the various mathematical operations might be realized in an actual computer. Flow charts and block diagrams are used extensively.