CHAPTER 6
DIGITAL ARITHMETIC

6.0 INTRODUCTION

In this chapter you will study digital arithmetic. Digital arithmetic is used in the internal calculations performed in many modern computer systems. You will learn about arithmetic using binary, hexadecimal, and BCD numbers. Circuits to perform digital arithmetic will be constructed and their operation analyzed.

6.1 OBJECTIVES

Upon completion of this chapter you should be able to:

- Understand binary addition with signed and unsigned numbers.
- Use the two's complement form of binary numbers to perform arithmetic.
- Multiply and divide binary numbers.
- Perform arithmetic operations on BCD and hexadecimal numbers.
- Implement digital arithmetic circuits.

6.2 DISCUSSION

Until this chapter, the numbers you have studied have been positive integers or fractions. In fact when we say that an N bit binary number can have a count of 2N - 1 we are implying that such a number is positive and uses all of the bits to express magnitude. Realize that this convention is arbitrary and that some of the bits can be used to indicate qualities other than magnitude.

6.2.0 Binary Addition

The positive binary integers used till now can be added a bit at a time by following a few simple rules which are shown in Figure 6-1.

FIGURE 6-1. Rules of Binary Addition.

$0 + 0 = 0$ $\quad$ $1 + 0 = 0 + 1 = 1$

and $1 + 1 = 0$ with a carry of 1

These rules are straight forward. Notice that the only operation resulting in a nonzero carry is the addition of two ones. These rules are useful for adding any binary digits and do not apply only when used with positive binary integers.

6.2.1 Signed Numbers

While the numbers used to this point are fine for counting, they have not allowed representation of negative quantity. Since numbers can be negative or positive, a single bit can be used to indicate the sign of a number while the remaining bits are used to indicate the magnitude of the number. The convention normally adopted is to have the MSB used for the sign bit and to have a zero in the sign bit indicate a positive number. This concept is illustrated in Figure 6-2.

FIGURE 6-2. Example of Signed Numbers.

01010111 = 87 $\quad$ 11010111 = -87

The rules used for addition will still work with these numbers. Some additional rules are needed to handle the sign bits. These rules are summarized below.

A. To add numbers with like signs add the magnitudes and use the common sign in the sign bit.
B. To add numbers with unlike signs find the difference in magnitude and use the sign of the number with the greatest magnitude.

Subtraction can also be accomplished with signed binary numbers in the same manner as is done with the decimal numbers. To subtract we merely change the sign of the subtrahend then add the subtrahend to the minuend following the rules of binary addition of signed numbers. Subtraction can also be carried out directly as with decimal numbers. The rules for binary subtraction are shown in Figure 6-3.

1 - 1 = 0 1 - 0 = 1 0 - 0 = 0

0 - 1 = 1 WITH A BORROW OF 2 FROM THE NEXT HIGHER BIT

FIGURE 6-3. Rules for Binary Subtraction.

Notice that the borrow in binary is a 2 instead of a ten as you are familiar with when using decimal numbers.

6.2.2 Complement Notation

A special form of writing numbers known as complement notation is used widely for binary arithmetic. This usage has become common because the complement form of a binary number is easily represented and manipulated by digital machines. The formula for the radix complement of a number is : (Rn) - N where R is the radix, n is the number of digits in the word representing the number and N is the number to be complemented. The radix-minus-1 complement is formed by subtracting one from the radix complement. Since you are concerned with binary numbers for use with computers, you will use the two's complement for the radix complement and the one's complement as the radix-minus-1 complement. Examples of these complements are shown in Figure 6-4.

NUMBER:	01010101	11011011
ONE'S COMPLEMENT:	00101010	10100100
TWO'S COMPLEMENT:	00101011	10100101

FIGURE 6-4. Examples of Complement Notation.

The left number is decimal + 85 and the right number is decimal - 91. Notice that the left most bits, the sign bits, are not changed by either of the complement operations. Also note that the one's complement is the complement of the bits of the magnitude with the sign bit retained.

This means that the hardware to form the one's complement is simply an inverter for each of the magnitude bits. The two's complement can be formed by adding one to the one's complement. This can be done using an EXOR gate and some additional circuitry. Subtraction of binary numbers is frequently performed by adding the two's complement of the subtrahend to the minuend. This process is illustrated in Figure 6-5.

FIGURE 6-5. Subtraction Using Two's Complement Notation.

DECIMAL	BINARY
14	00001110
- 5	+ 11111011
9	100001001

Notice that the result of the subtraction has an overflow or carry bit. The result also will be in a signed magnitude format with negative numbers represented as two's complements. The carry bit is not used in this example hence it is discarded.

6.2.3 Binary Multiplication

The process for multiplying binary numbers is similar to the process used to multiply decimal numbers. Multiplication is accomplished by successive addition. For example 6 x 3 = 6 + 6 + 6. When multiple digit decimal numbers are multiplied, the digits of the multiplier are used one at a time to operate on the multiplicand. The partial products, which are the result of these operations, are added to form the final product. Each partial product is shifted one place to the left as the multiplicand digits

are used from LSB to MSB. This has the effect of multiplying each partial product by ten. A similar set of rules is used in binary multiplication. Binary multiplication is simpler since only two results can be obtained. The rules for binary multiplication are:

A. When a binary number is multiplied by one the result is the original number or multiplicand.

B. When a binary number is multiplied by zero the result is zero.

An example of multiplying a multi-digit binary number is shown in Figure 6-6.

```
  1101     MULTIPLICAND
X 101      MULTIPLIER
------
  1101     PARTIAL PRODUCTS
 0000
+1101
-------
1000001    FINAL PRODUCT
```

FIGURE 6-6. Example of Multi-Digit Binary Multiplication.

This method of multiplying will give correct answers and is very similar to the one you use to multiply decimal numbers. This is not the only way to perform binary multiplication nor is it the most hardware efficient. The same result can be obtained by shifting the previous result to the right instead of shifting each successive partial product to the left. This method is shown in Figure 6-7.

```
  1101     MULTIPLICAND
X 101      MULTIPLIER
------
  1101     INTERMEDIATE RESULTS
 00000
 001101    NOTE: ONLY THE LAST
           TWO TERMS ARE SUMMED!!
-------
1000001    FINAL PRODUCT
```

FIGURE 6-7. Alternate Method for Multiplying Binary Numbers.

This method of multiplication will also give correct results. Notice that for a zero bit in the multiplier the intermediate result is shifted right one position. For a one bit in the multiplier the intermediate result is shifted right one bit then the multiplicand is added to the result.

This method of multiplying is more hardware efficient in that it can be implemented using one fewer registers than would be required to implement the method shown in Figure 6-6.

6.2.4 Binary Division

Binary division can be accomplished by successive binary subtraction. This method will give accurate results but is slow and cumbersome for division of large numbers. The number of steps required to perform the division can be reduced by a process called shifting. An example of this process applied to decimal numbers is shown in Figure 6-8.

QUOTIENT 1500/5

We know that the farthest that 5 can be shifted and still be divided into 1500 with an integer result is two places, so 500 will be used for this process.

FIGURE 6-8. Alternate Method for Division.

```
 1500
-500          100 PARTIAL QUOTIENT
----
 1000
-500
----
 500          100 PARTIAL QUOTIENT
-500
----
   0          100 PARTIAL QUOTIENT
---
 300          FINAL QUOTIENT
```

This same principle can be applied to binary division. This method is frequently used in computer arithmetic. It can be implemented using subtraction circuitry in conjunction with some logic to determine the size of the number to be subtracted from the dividend. An example of this method, known as the restoring method, applied to binary numbers is shown in Figure 6-9.

The quotient is 1000001/1101

FIGURE 6-9. Binary Division by the Restoring Method.

```
1000001 can at least be divided by 110100 so,
-110100
-------
   1101      add 100 to quotient
  -1101      use 1101 for division
  -----
      0      add 1 to complete quotient
             quotient = 101
```

6.2.5 Hexadecimal Arithmetic

You could use a system of addition and subtraction using fifteens and sixteens complement notation for hexadecimal arithmetic. However, with the large number of binary arithmetic devices available it is easier to convert from hexadecimal to binary for performing arithmetic. When this is done, all answers will have to be converted back to hexadecimal after computation.

6.2.6 BCD Addition

BCD addition is frequently used in systems where the results are displayed as decimal numbers. Calculators are one example of this type of system. You have already learned about BCD notation. Some interesting things happen with BCD arithmetic because of the 6 unused states in a BCD digit. Figure 6-10 illustrates BCD addition problems.

FIGURE 6-10. BCD Addition Examples.

DECIMAL	BCD	
3	0011	
+4	+0100	
7	0111	CORRECT RESULT

FIGURE 6-10. Continued.

6	0110	
+ 8	+ 1000	
14	1110	INCORRECT RESULT, NO CARRY
9	1001	
+ 8	+ 1000	
17	0001 0001	WRONG RESULT, CARRY GENERATED

This problem occurs because the carry for the decimal system occurs for sums greater than nine while the carry in the BCD system occurs for numbers greater than 15. To correct this, a six must be added to all sums greater than nine. To do this a machine would need to be able to recognize results of sums which are greater than nine and add six to those sums. Results of sums less than nine are correct in BCD arithmetic.

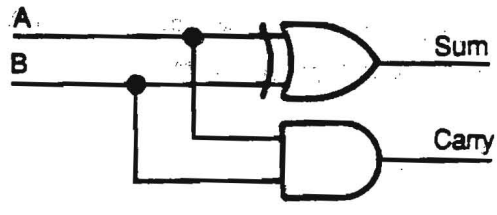
6.2.7 The Half-adder

Until now you have concentrated on the mechanics of binary related arithmetic and we have only hinted at how to use digital circuits to perform arithmetic operations. The next few sections will concentrate on implementing arithmetic circuits.

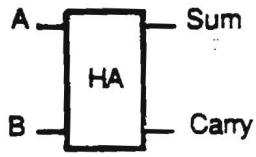
Your study of arithmetic circuits will begin with the binary half-adder. The schematic and truth table for the half-adder are shown in Figure 6-11.

FIGURE 6-11. Half-Adder.

Schematic



Block Diagram



Truth Table

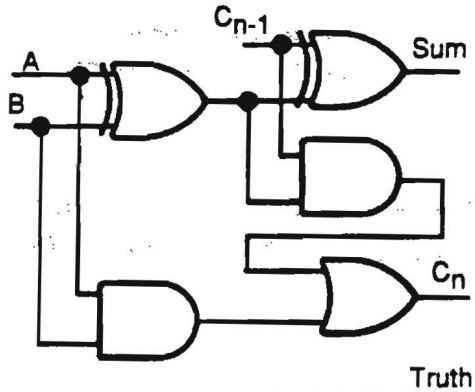
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Notice that this circuit has two inputs and two outputs. The EXOR gate performs the addition while the AND gate detects when both inputs are ONE and forms the carry output. This circuit is called a half-adder because it lacks the ability to accept a carry input from a previous addition.

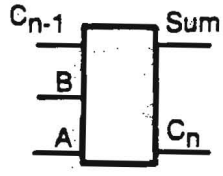
6.2.8 Full-adder

The full-adder has three inputs and two outputs. The inputs are the two bits to be added and a carry input from a previous addition. The full-adder has the sum and carry outputs. The schematic and truth table for the full-adder are shown in Figure 6-12.

Schematic



Block Diagram



Truth Table

A	B	C_{n-1}	S	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

FIGURE 6-12. Full-Adder.

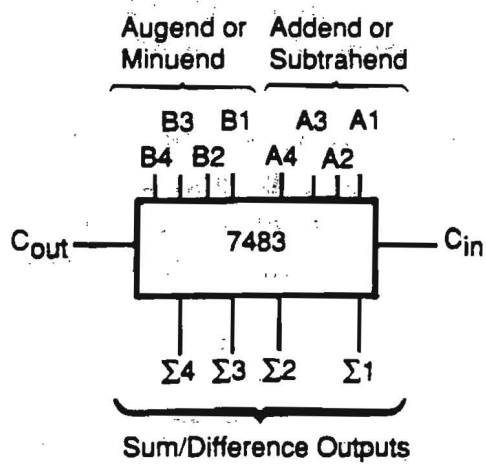
6.2.9 Parallel Binary Adder

A parallel binary adder will perform the addition operation on multiple bit binary numbers. The circuit which performs the function in the TTL logic family is the 74LS83. The circuit has some features that require discussion.

The 74LS83 is a four-bit binary adder with fast carry. The fast carry is made possible by circuitry which is called a "look ahead" carry circuit. This circuitry samples the output of each individual adder thus saving the time required for a carry to ripple through each adder stage. The 74LS83 also performs math in the true logical sense. This means that outputs will all be true. For one's complement arithmetic this means that the end around carry can be directly implemented. An End Around

Carry or EAC is needed when the result of the addition of two numbers with unlike signs is positive (>0). The block diagram for the 74LS83 is shown in Figure 6-13.

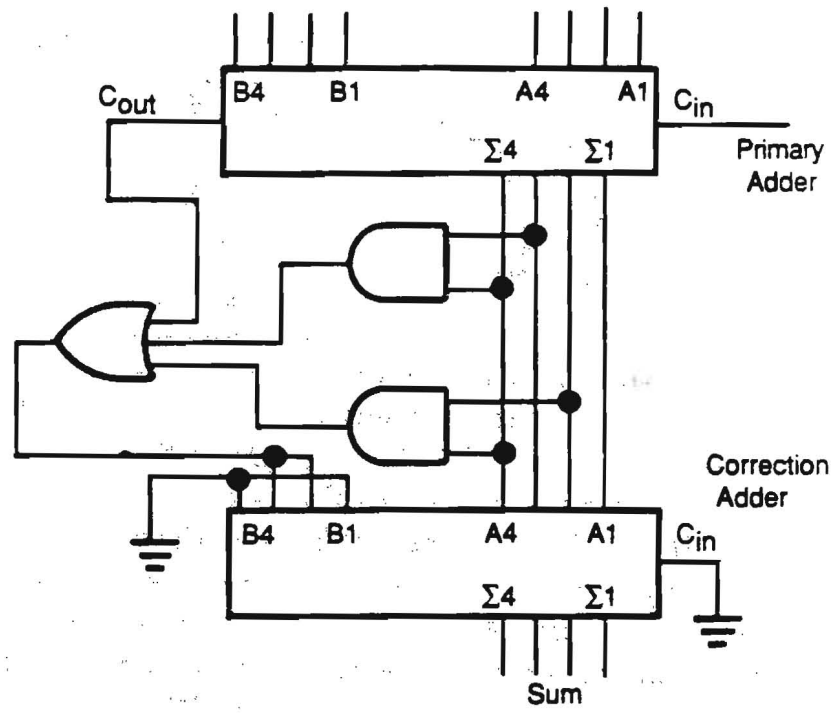
FIGURE 6-13. Parallel Adder IC.



6.2.10 BCD Adder

A BCD adder can be formed from two four-bit adders and some additional circuitry. The schematic for a BCD adder is shown in Figure 6-14.

FIGURE 6-14. BCD Adder Circuit.



Notice that the first adder performs the basic addition, while the second adder will add six to outputs that are nine or greater. The second adder is controlled by the AND and OR

gates which detect when the output of the main adder is nine or greater.

6.2.11 Binary Multipliers

A reasonable variety of IC multipliers are available in the TTL logic family. A dedicated multiplier is generally used only where speed is very important. An example of this type of circuit is the 74LS261, a two-bit by four-bit binary multiplier, capable of producing a five-bit output in 26 nS. Where speed is less of a consideration an Arithmetic Logic Unit or ALU is frequently used.

These devices can be used to perform the multiplication function and other arithmetic and logic functions. The 74LS181 is an example of an ALU. It performs arithmetic and logic operations on two four-bit binary numbers.

6.3 SUMMARY

In this chapter you have learned about binary arithmetic and the circuits required to perform binary arithmetic. You have learned about the half- and full-adders, binary multipliers, BCD adders, and ALUs as means of performing binary arithmetic. These items form the backbone of digital arithmetic computation.

6.4 REVIEW QUESTIONS

1. What is the sum of 101 and 011 ?

_____

2. Compute the difference between 011 and 010.

_____

3. What is meant by a half-adder ?

_____

_____

_____

4. Divide 111001 by 10010.

5. Multiply 101 by 110.

6. Convert FF hexadecimal to binary.

7. Convert FF hexadecimal to decimal.

8. What is an ALU ?

9. Write -120 in the two's complement binary format.

10. Write -120 in the one's complement binary format.

11. What is the range of signed number in an eight-bit register using two's complement notation ?

12. Convert 120 to BCD.