# Microcontrollers:
# The Arithmetic Logic Unit

An Online Continuing Education Course for Engineers

**Course Number: E-2112N**

**Credit: 2 Hours / 2 PDH / 2 CPD**

# Microcontrollers: The Arithmetic Logic Unit

Mark A. Strain, P.E.

## Table of Contents

## Introduction

The arithmetic logic unit (ALU) is the central core of a central processing unit (CPU). The ALU is simply a digital circuit that performs arithmetic and logical operations on binary numbers. They are combinational logic circuits which means that their outputs change asynchronously in response to changes to their inputs. ALU circuits perform operations on integer binary numbers. A floating point unit (FPU) is used to do operations on floating point numbers (or non-integers). This course focuses on the ALU.

An ALU has two integer inputs called operands and another input called an opcode. The opcode instructs the ALU which instruction to perform (like addition, subtraction, decrement, increment, AND, OR, NOT, XOR, etc.). The opcode code is a binary code that comes from the instruction set (or program) that is being executed. The instructions or program will contain both the operands or numbers to be used and the opcode that tells the ALU what to do with the numbers, e.g., add the numbers. These instructions are usually written in a higher-level programming language and are stored in the computer's main memory. A compiler will compile the higher-level language program and convert it to machine code. The computer executes one instruction at a time.

The CPU will go through a fetch-decode-execute cycle for each instruction. It will fetch an instruction from the program in memory, it will decode the instruction into binary codes that the ALU can use, and will execute the instruction.

## Binary Numbering System

In everyday life, we are used to the numbering system known as the decimal numeral system. Our common numbering system is based on Arabic numerals (or symbols). Our numbering system is base-10, which means there are ten symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9) used to represent every possible combination of numbers. Our numeral system is based on the number ten probably because long ago, we discovered that we each have ten fingers which are useful tools to count on when doing simple math.
Computer systems and other digital systems use a numbering system based on a number other than ten. Digital systems (such as a computer central processing unit) use a numbering system based on the number two. This base-2 numbering system is called the binary system. The binary system uses two symbols (0 and 1) to represent every possible combination of numbers.

## Decimal Notation

When we write decimal (base-10) numbers, we use positional notation. This means that each digit in a number is multiplied by a specific power of ten. The powers of ten (exponents) are positive on the left side of the decimal point, and the powers of ten (exponents) are negative on the right side of the decimal point.

For example, consider the decimal number 3854:

$= 3(10^3) + 8(10^2) + 5(10^1) + 4(10^0)$

$= 3000 + 800 + 50 + 4$

= 3854

Now consider the decimal number 1256.79:

$= 1(10^3) + 2(10^2) + 5(10^1) + 6(10^0) + 7(10^{-1}) + 9(10^{-2})$

= 1000 + 200 + 50 + 6 + 0.7 + 0.09

= 1256.79

| … | $10^5$ | $10^4$ | $10^3$ | $10^2$ | $10^1$ | $10^0$ | . | $10^{-1}$ | $10^{-2}$ | $10^{-3}$ | $10^{-4}$ | $10^{-5}$ | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 5 | 6 | . | 7 | 9 | | | | |

## Binary Notation

The binary system also uses positional notation. Each digit in the base-2 numbering system is multiplied by a specific power of two. Just as in the base-10 system, the powers of two are positive on the left side of the binary point, and the powers of two are negative on the right side of the binary point.

## Binary to Decimal

Each digit is a multiple of a power of 2. All digits to the left of the decimal point are positive powers of 2, and all digits to the right of the decimal point are negative powers of 2.

Consider the binary number 1011:

$= 1(2^3) + 0(2^2) + 1(2^1) + 1(2^0)$

= 8 + 0 + 2 + 1

= 11 (base-10)

Now consider the binary number 10101101.101:

$= 1(2^7) + 0(2^6) + 1(2^5) + 0(2^4) + 1(2^3) + 1(2^2) + 0(2^1) + 1(2^0) + 1(2^{-1}) + 0(2^{-2}) + 1(2^{-3})$

= 128 + 0 + 32 + 0 + 8 + 4 + 0 + 1 + 1/2 + 0 + 1/8

= 173.625 (base-10)

| … | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | . | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ | $2^{-5}$ | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | . | 1 | 0 | 1 | | | |

## Decimal to Binary

It is often necessary to represent a decimal fraction in binary. The integer part (to the left of the decimal point) is converted to binary by continually dividing by 2 until you get to 1. The fractional part is converted to binary by continually multiplying by 2 until you get 0 or a repeating sequence or you get tired.
Consider the decimal number 142.378:

First, convert the integer part:

Divide by 2 Remainder

| | |
|---|---|
| 142 / 2 | 0 |
| 71 / 2 | 1 |
| 35 / 2 | 1 |
| 17 / 2 | 1 |
| 8 / 2 | 0 |
| 4 / 2 | 0 |
| 2 / 2 | 0 |
| 1 | 1 |

Now, read the remainder part backward, and that is the binary representation of the integer part of the number: 10001110

Next, convert the fractional part. Start with the number to the right of the decimal point (0.378). Multiply the number times 2 and record what is to the left of the decimal place after this operation. Then take this number and discard whatever is to the left of the decimal place and continue.

0.378 * 2 = 0.756 → 0
0.756 * 2 = 1.512 → 1
0.512 * 2 = 1.024 → 1
0.024 * 2 = 0.048 → 0
0.048 * 2 = 0.096 → 0
0.096 * 2 = 0.192 → 0
0.192 * 2 = 0.384 → 0
0.384 * 2 = 0.768 → 0
0.768 * 2 = 1.536 → 1
0.536 * 2 = 1.072 → 1
0.072 * 2 = 0.144 → 0
0.144 * 2 = 0.288 → 0
0.288 * 2 = 0.576 → 0
0.576 * 2 = 1.152 → 1
0.152 * 2 = 0.304 → 0
…

Now, read the number forwards, and that is the binary representation of the fractional part of the number: 0.011000001100010

Therefore, 142.378 = 10001110.011000001100010…

## Negative numbers in Binary

In mathematics, negative decimal numbers can be represented by using a minus "-" sign. In digital circuits, numbers are represented only by a sequence of bits, either a 0 or a 1 and no plus or minus sign. The most popular methods of representing signed numbers in binary are the one's complement and two's

complement methods. These methods allow subtraction to be performed by adding the complement of a number instead of subtracting the number. The utilization of one's complement and two's complement greatly simplifies digital circuits since addition is a fundamental operation.

## One's Complement

The one's complement of a binary number is obtained by negating the number by inverting all of the bits. This is accomplished by changing all of the 0s into 1s and all of the 1s into 0s. The one's complement of a number behaves as the negative of the original number. An addition operation is a fundamental operation in digital systems. There is no fundamental subtraction operation. Subtraction of two numbers is equivalent to adding one number to the negative of the other number. Therefore, any subtraction operation is equivalent to inverting one of the numbers and adding it to the other number.

Consider taking the one's complement of the number 0000 1100 (base-2):

```
0000 1100
```

Invert all of the bits

```
1111 0011
```

One's complement is seldom used in digital systems because when a one's complement number is added to another number, the result is offset by –1. In other words, the result of a subtraction operation (using one's complement) is off by –1.

Consider subtracting 8 from 12 using one's complement:

One's complement of 8:

```
0000 1000
```

Invert all of the bits

```
1111 0111
```

Now subtract 8 from 12:

12 – 8 = 4

Add the one's complement of 8 to 12: (12 – 8)

```
0000 1100
1111 0111
----------
0000 0011
```

= 3 (base-10)

Note that the result is off by one. This problem is resolved by performing a two's complement operation instead of a one's complement.

## Two's Complement

Two's complement representation of a binary number has widespread use in digital systems. It solves the problem of the –1 offset that one's complement produces.

Consider the same subtraction operation as above, but this time using two's complement of 8:

```
0000 1000
```

Invert all of the bits and add one

```
1111 0111
0000 0001
-----------
1111 1000
```

(Note: Binary digits are often group

Now subtract 8 fro

Now consider subtractin

```
0000 0001
-----------
1110 1001
```

Now subtract 23 from 17:

17 – 23 = –6

Add the two's complement of 23 to 17

```
0001 0001
1110 1001
-----------
1111 1010
```

= –6 (base-10)