

CPE 325: Embedded Systems Laboratory

Laboratory #2 Tutorial

C Data Types and Their Representation in Memory

Aleksandar Milenković

Email: milenka@uah.edu

Web: <http://www.ece.uah.edu/~milenka>

Objective:

This tutorial will familiarize you with different number systems and how to convert representations between them. It includes the following topics:

Converting representations with bases 2, 8, 10, and 16

Understanding how data are stored in memory

Understanding the common data types used with the MSP430

Notes:

The previous tutorial introducing the TI experimenter's board and the Code Composer Studio software development environment is required for successful completion of this lab.

Required reading: [CPE 323 Review – Data Types and Number Representations in Modern Computers](#).

Contents:

| | | |
|-----|---|---|
| 1 | Numerical Base Systems..... | 2 |
| 1.1 | Binary, Decimal, and Hexadecimal..... | 2 |
| 1.2 | Converting Using the Windows Calculator | 3 |
| 2 | MSP430 Memory..... | 4 |
| 3 | C data Types..... | 5 |
| 4 | References | 5 |

1 Numerical Base Systems

In microcontroller applications, it is very common for numerical values to be represented in several different bases. In physical memory, values are stored in binary; however, representing large binary values can be cumbersome and inefficient. Often these values are represented in either octal or hexadecimal forms. It is important to be able to quickly interpret and convert values between binary, octal, hexadecimal, and decimal bases.

1.1 Binary, Decimal, and Hexadecimal

Different numeral systems can be used to express one value in multiple ways. While we generally use and think in base 10 (decimal), digital hardware exists in only two states – on or off. For that reason, it makes sense to use base 2 (binary) to represent values kept in digital hardware.

We are most familiar with using the decimal number system where each order of magnitude represents another power of 10. For instance, the value 163 in decimal is equal to:

$$(1 \times 10^2) + (6 \times 10^1) + (3 \times 10^0) = 163$$

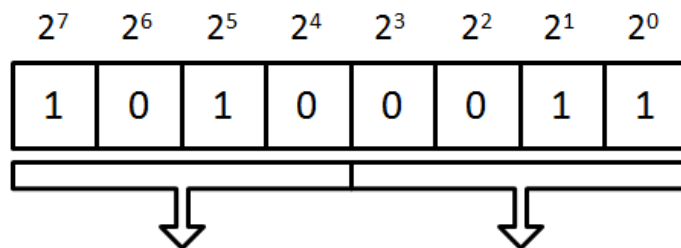
| 10^2 | 10^1 | 10^0 |
|--------|--------|--------|
| 1 | 6 | 3 |

In the binary system, instead of each order of magnitude being a power of 10, they are a power of 2. The same value that we represented as 163 in decimal is represented as 10100011 in binary:

$$(1 \times 2^7) + (0 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 10100011_2 = 163_{10}$$

| 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

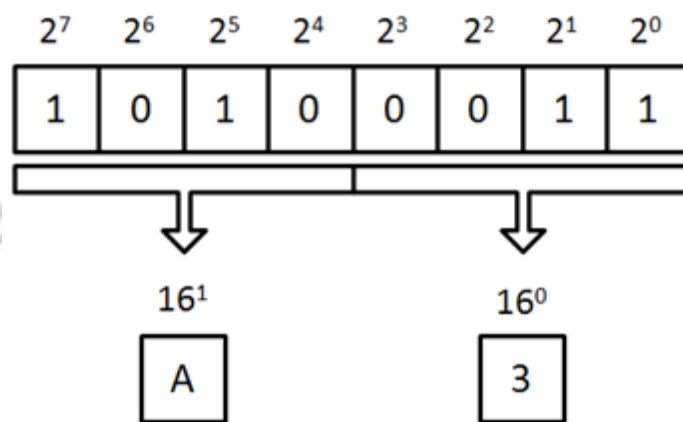
You can see that representing large values in binary can be cumbersome. You can quickly convert between binary and octal or hexadecimal in order to use fewer digits to represent the same number. The conversion is simple, and the method should become second nature. To represent a binary number in hexadecimal, group the binary value digits in groups of four.



You can then convert each block of four values to its corresponding hex value as seen in the following chart:

| | | | |
|---|------|---|------|
| 0 | 0000 | 8 | 1000 |
| 1 | 0001 | 9 | 1001 |
| 2 | 0010 | A | 1010 |
| 3 | 0011 | B | 1011 |
| 4 | 0100 | C | 1100 |
| 5 | 0101 | D | 1101 |
| 6 | 0110 | E | 1110 |
| 7 | 0111 | F | 1111 |

Therefore, the same binary value shown above can be represented in hexadecimal as A3.



You can easily go from hexadecimal to binary by using the reverse method. Each hex digit breaks out into four binary digits. Likewise, the method can be used with octal by grouping three instead of four binary digits per octal digit.

1.2 Converting Using the Windows Calculator

The Windows calculator can be accessed by typing in Calculator into the search line. Once the calculator application has been started, select Programmer view to get screen shown in Figure 1.

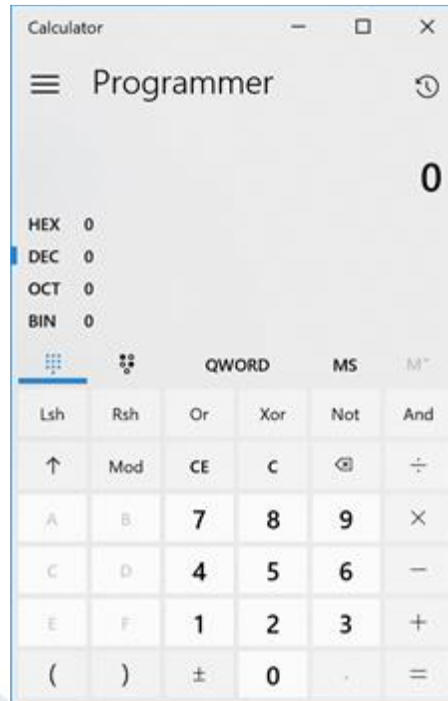


Figure 1. Windows Calculator Application

Side fields HEX, DEC, OCT, and BIN allow you to select a desired numeral system. By changing them you can rapidly convert an entered value from one base to another.

2 MSP430 Memory

Since we will be using the MSP430 architecture in this class, it is good to become familiarized with the way the MSP430 stores and recalls memory. You will learn detailed information about the MSP430 architecture in class; however, there are a few basic concepts that you should review for the lab.

As you likely already know, each binary digit in memory is referred to as a bit. Likewise, a byte is formed of 8 sequential bits and is smallest addressable unit in memory. The MSP430 memory can be viewed as an array of bytes or an array of 2-byte “words.” In memory, 2-byte words are aligned to even addresses, i.e., each word begins at an even address. Another important policy defines how multi-byte objects are stored in memory. The MSP430 uses a **“little endian”** placement policy where the first byte of a word (the least significant byte) is at the lower address in memory (see Figure 2).

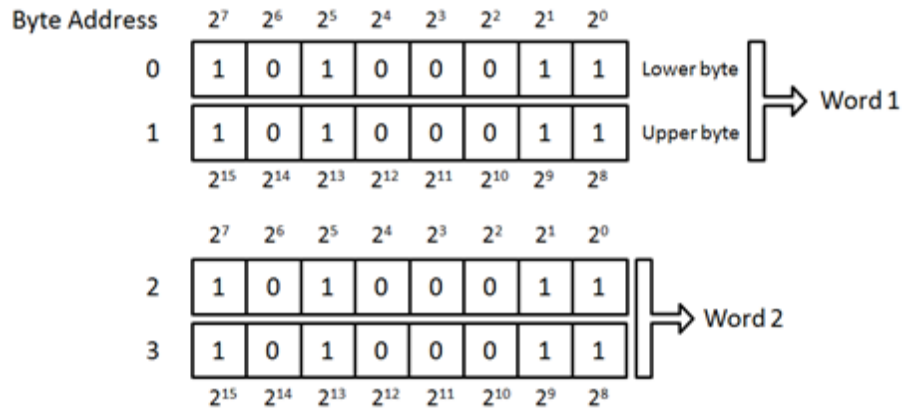


Figure 2. Little-endian Placement

While using the Code Composer Studio, you may also occasionally notice that registers appears to be 20 bits long. This is part of the extended architecture that allows each register to store a 20 bits of memory address. This is useful for easily pointing to addresses in memory. In the first lab this was disabled when the code and data models were changed to small.

3 C data Types

In this lab, you will use the C language to write programs for your MSP430. It is important to become familiar with the different data types and how they appear in memory. Below is a list of some of the data types that you will be using in this lab:

| Data type | Size | Characteristic |
|--------------|---------|--|
| bool | 1 byte | true/false flag |
| char | 1 byte | ASCII translated value |
| int | 2 bytes | Signed integer (2's complement) |
| unsigned int | 2 bytes | Unsigned integer |
| long int | 4 bytes | Signed (2's complement) |
| float | 4 bytes | Single-precision floating-point (sign bit, 8-bit offset exponent, 23-bit mantissa) |

You should become familiar with each of these data types, especially how they look in memory. One of the key concepts that you should recognize is that the same values in memory can be interpreted different ways depending on its associated data type. You will learn more about these data types as you explore their output in this lab while using the C printf statement.

4 References

- Read pages 25 – 29 in John H. Davies' MSP430 Microcontroller Basics.
- More information for the printf function, float.h library, and limits.h library can be found at <http://www.cplusplus.com>