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 review different number systems and demonstrate 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:

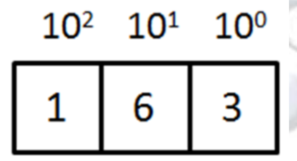
1 Numerical Base Systems

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.

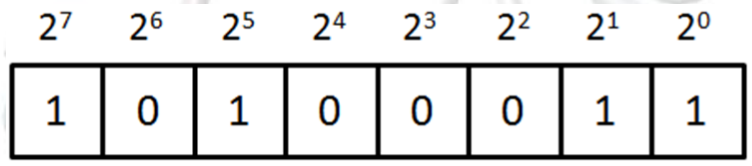
1.1 Binary, Decimal, and Hexadecimal

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:

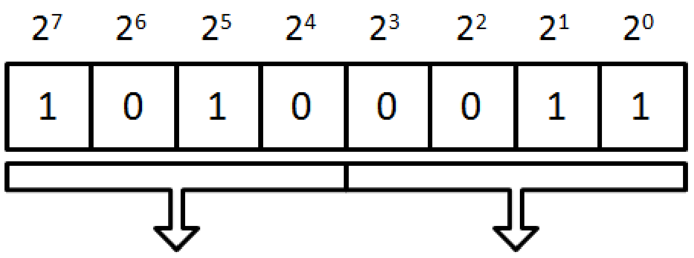
$$(1 \times 10^2) + (6 \times 10^1) + (3 \times 10^0) = 163$$



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



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 fairly simple. To represent a binary number in hexadecimal, group the binary value digits in groups of four, starting from the least significant bit position (with weight 2^0).

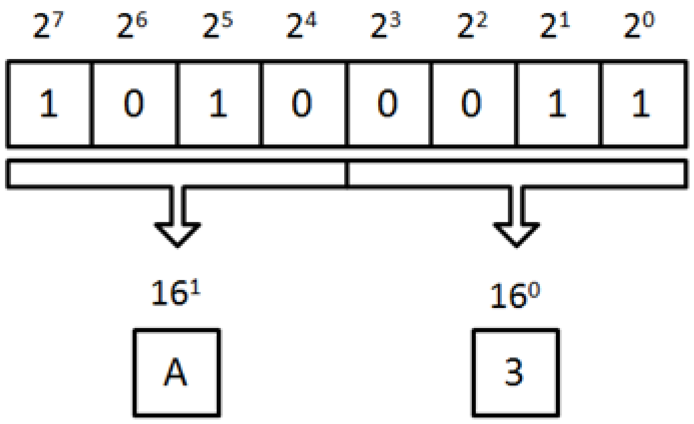


You can then convert each block of four bits to its corresponding hex value as seen in the following chart:

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. We prefer to indicate hexadecimal numbers using 0x in the prefix (0xA3) or letter h/H in the suffix (A3 h).



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.

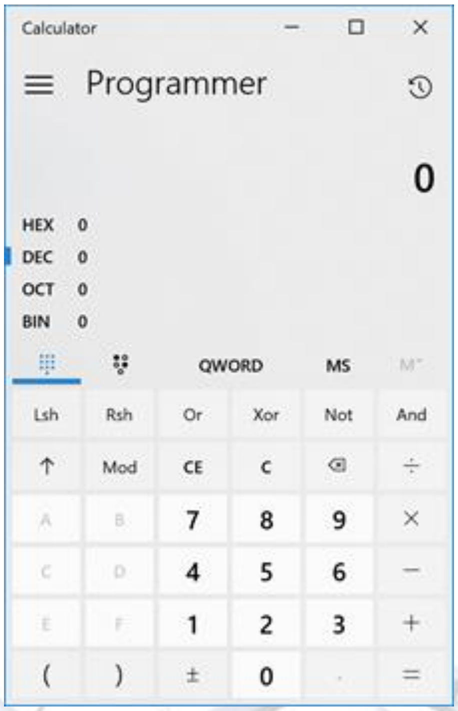


Figure 1. Windows Calculator Application

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 this 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 or 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).

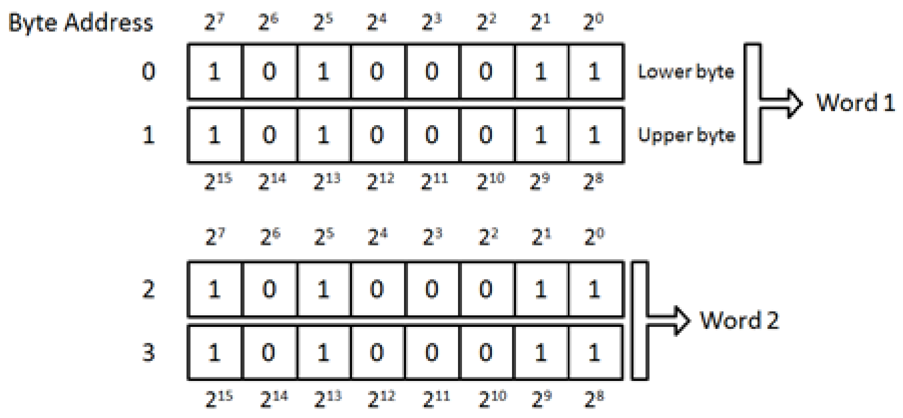


Figure 2. Little-endian Placement

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 so the address space is extended to 2^{20} bytes or 1 MB. 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.

Consider a demo program shown in Figure 3. It declares and initializes several variables of different types. For each variable, a C printf statement is given that prints the variable in its decimal, hexadecimal, and octal form. Please pay attention to the format fields used in the printf statements (those fields are marked by double quotes). This C string includes one or more format specifiers - subsequences beginning with %. For example, consider the statement in line 30. This printf prints the value of variable three times as follows:

- in decimal format (%6d – 6 indicates that the width of the field is 6 characters, d indicates signed decimal integer),
- in hexadecimal format (%#04.4x – # ensures that the number is printed with prefix 0x in case of hex representation, 4.4 specifies width of the filed (4) with all 4 digits printed, including leading 0s, and x indicates hexadecimal representation), and
- in octal format (%#06o - – # ensures that the number is printed with prefix 0 for octal representation, 6 specifies width of the filed including leading 0s, and o indicates octal representation).

If you want to learn more about the formats, please visit the following web site: http://www.cplusplus.com/reference/cstdio/printf/.

```
1
2   /*------------------------------------------------------------------------------
3    * File: Lab2_D1.c
4    * Description: This program demonstrates common C data types and P
5    *              printf formatting options.
6    *
7    * Platform: EXP-MSP430F5529lp
8    *
9    * Author: Aleksandar Milenkovic, milenkovic@computer.org
10   * Date:    August 3, 2020
11   *
12   *------------------------------------------------------------------------------*/
13  #include <msp430.h>
14  #include <stdio.h>
15
16  int main(void) {
17      int i1 = 11, i2 = -6;           // 16-bit integers
18      unsigned int u1 = 65535;        // unsigned 16-bit integer
19      long int l1 = 100000;           // 32-bit signed integers
20      long int l2 = -2;
21      char c1 = 'A';                  // 8-bit character
22      float f1 = 1.25;                // single-precision floating-point
23      unsigned long int * p1 = &f1;   // p1 points to memory where f1 is stored
24      unsigned long int l3 = *p1;     // interpret floating-point number as long
25  integer
26
27      WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
28      printf("Each variable below is printed in decimal, hex, and octal
29  representations.\n");
30      printf(" i1 = %6d, %#04.4x, %#06o\n", i1, i1, i1);
31      printf(" i2 = %6d, %#04.4x, %#06o\n", i2, i2, i2);
32      printf(" u1 = %6u, %#04.4x, %#06o\n", u1, u1, u1);
33      printf(" l1 = %ld, %#08.8lx, %#011lo\n", l1, l1, l1);
```

```
    printf(" l2 = %6ld, %#08.8lx, %#011lo\n", l2, l2, l2);
    printf(" f1 = %6.2f, %#08.8lx\n", f1, l3);
    printf(" c1 = %c, %#02.2x, %#03o\n", c1, c1, c1);
    printf("That's all folks\n");

    return 0;
}
```

Figure 3. Common C data types and their representation in memory using printf formatting (Lab2_D1.c)

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
- Code Composer documentation: MSP430 C/C++ Language Optimization
- Printf function in stdio library: http://www.cplusplus.com/reference/cstdio/printf/