1. CPE 325: Laboratory Assignment #2
Base Systems and Numerical Data Types in Memory

Objectives: The purpose of this lab is to reiterate the representation of numerical values in different base systems and how those numerical data types are stored and accessed in memory. In particular, you will:

- Know how to convert values between base 2, 8, 10, and 16
- Understand how values are stored in memory
- Become familiar with the common data types used with the MSP430

1.1. 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 hex forms. It is important to be able to quickly interpret and convert values between binary, octal, hexadecimal, and decimal bases.

1.1.1. Binary, octal, hex, and decimal

As you have encountered before, numerical base systems can be used to express a value 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 associated with digital hardware.

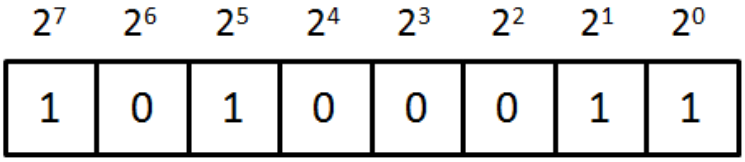
We are most familiar with using the decimal number system where each order of magnitude represents another power of 10. For instance, the value 163 in decimal is equal to:

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

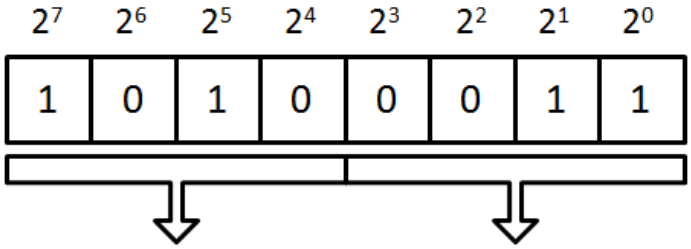
10^2	10^1	10^0
1	6	3

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:

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



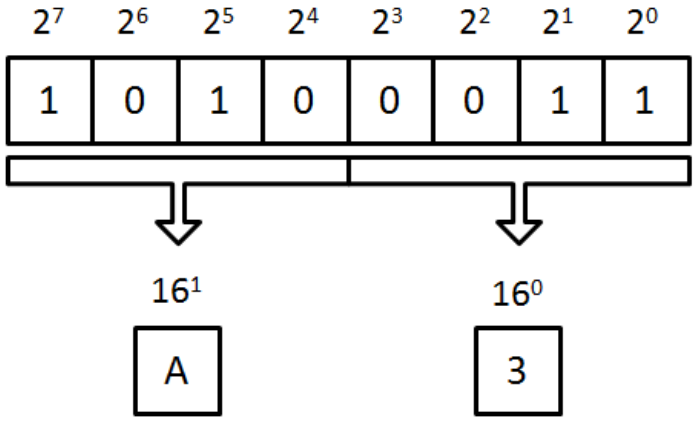
You can see that representing large values in binary can be difficult to do. You can quickly convert between binary and octal or hexadecimal in order to use less digits to represent the same number. The conversion is simple, and the method should become second nature. To represent a binary number in 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.

1.1.2. Converting using the Windows XP calculator

The Windows XP calculator can be accessed by clicking the Start menu > Programs > Accessories > Calculator. Once the calculator application has been started, click the View menu, and click on scientific. The scientific calculator interface is now open. Note a series of radio buttons on the upper left portion of the application.

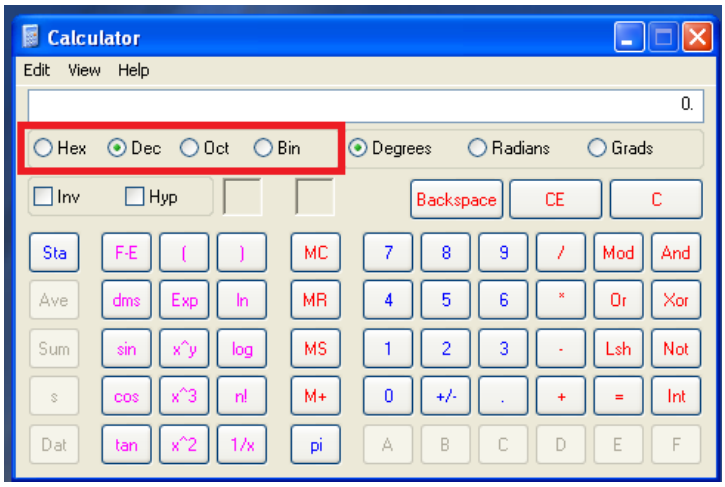


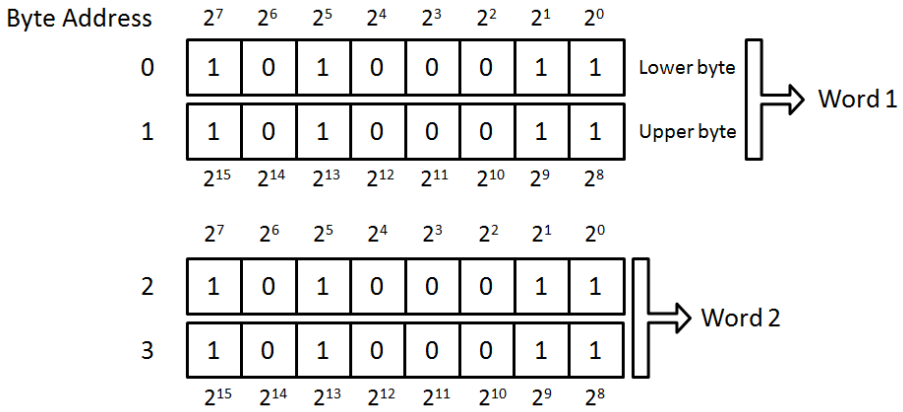
Figure 1. Windows XP calculator application

These buttons allow you to rapidly convert an entered value from one base to another. First, be sure the base for the initial value is correct. After entering a value, click on the desired base to see the converted value.

1.2. MSP430 Memory

Since we will be using the MSP430 architecture in this class, it's 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. The MSP430 memory is divided into 2-byte "words." In memory, each word begins at an even address. The MSP430 uses a "little endian" system where the first byte of a word (the least significant byte) is at the lower address.



While using the IAR workbench, you may also occasionally notice that the memory appears to be 20 bits long. This is part of the extended architecture that allows each word to store a 20 bit memory address. This is useful for easily pointing to addresses in memory.

1.3. 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 (2's complement) or unsigned (positive only)
long int	4 bytes	Same as int
float	4 bytes	Signed (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.

1.4. References

- Read pages 25 – 29 in the Davies text.
- More information for the printf function, float.h library, and limits.h library can be found at http://www.cplusplus.com

1.5. Assignments

Write a C program that will declare and initialize the following variables:

Variable name	Type	Initial value
c1	character (char)	‘a’
sui1	short unsigned integer	128
ssi1	short signed integer	-127
ui1	unsigned integer	1025
i1	integer	-100
li1	long integer	100000
f1	float	24.34

The program should display information about the variable type, name, size in bytes, value in decimal number system, value in hexadecimal number system, and value in octal number system. The floating point variable f1 should also be shown in floating-point and scientific formats. The information should be displayed in Terminal I/O and the output should be formatted as shown below (the output should include this header and one row for each variable).

```
+------------+-----+-----+------------+------------+------------+------------+------------+
Type         Var   Size  Dec.         Hex.         Oct.         FP           Exp
+------------+-----+-----+------------+------------+------------+------------+------------+
char         c1    1     97           0x61         0141
```

Below that, the program should print the ranges of common data types including char, short int, int, long int, unsigned int, unsigned long int, float, and double. Use definitions given in the limits.h and float.h header files for the ranges of data types