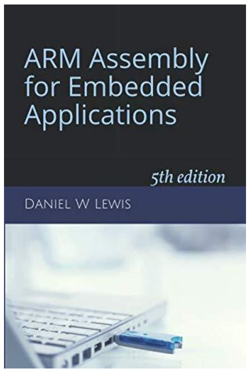


Programming Lab 2B

Interpreting Binary

Click to download Lab2B-Main.c

Topics: Conversion from binary to octal, hex, decimal, 2's complement and signed magnitude.

Prerequisite Reading: Chapters 1-2 & Appendix B
Revised: January 3, 2022

Background: This lab exercises your understanding of binary number systems. Since you have not been introduced to assembly language yet, this assignment is to be coded entirely in C. Successful completion of this assignment will also reinforce your familiarity with the workspace environment.

Assignment: The main program will compile and run without writing any code. However, your task is to create equivalent replacements in C for the functions shown below that are found in the main program **without using any library functions**. The original versions have been defined as "weak" so that the linker will automatically replace them in the executable image by those you create; you do not need to remove the versions. This allows you to create and test your functions one at a time.

1. Use "`make cleanall`" to remove any existing files in the `src` and `obj` subdirectories of your workspace folder.
2. Click on the link above to download the C main program and store it in the `src` subdirectory of your workspace folder.
3. Use your favorite text editor (not a word processor) to create a second C source code file in the `src` subdirectory that implements the five functions shown below; you do **not** need to create a function for 1's complement. Do not use filenames containing spaces or filename extensions with uppercase letters. Compile the program using "`make`".

```
void Bits2OctalString(uint8_t bits, char string[]) ;    // radix 8
void Bits2UnsignedString(uint8_t bits, char string[]) ; // radix 10
void Bits2HexString(uint8_t bits, char string[]) ;      // radix 16

void Bits2TwosCompString(uint8_t bits, char string[]) ;
void Bits2SignMagString(uint8_t bits, char string[]) ;
```

The program uses these functions to display various numeric interpretations of the displayed binary pattern, which may be adjusted using the slider and decrement (-) and increment (+) buttons. If there is an error in one of your functions, the corresponding value will be displayed in **red**.

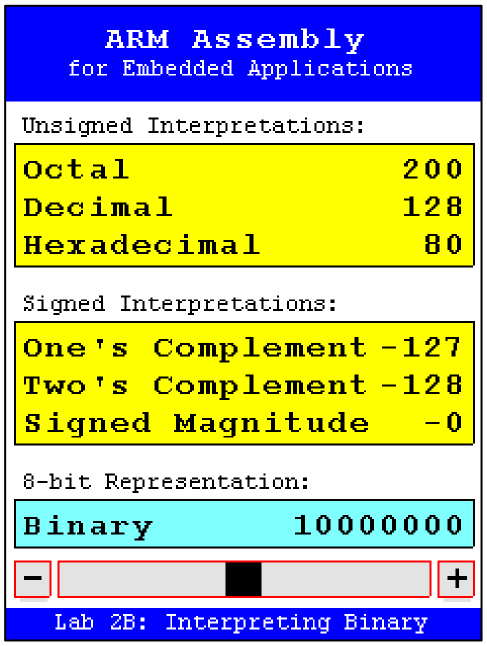
Hint: For all unsigned conversions, use division and remainders to extract one *integer* for each radix digit in the unsigned number `bits` and convert each such integer to a corresponding *character* using a table lookup as in:

```
static const char hex[] = "0123456789ABCDEF" ;
unsigned digit = ... ; // a number from 0 to 15
char ch = hex[digit] ; // one character of string
```

Store the characters into the array `string`, terminated with a NUL character (`'\0'`). Inside `Bits2SignMagString` and `Bits2TwosCompString`, compute the sign and magnitude separately and call `Bits2UnsignedString` to create the digits.

Alternative: Create one general-purpose function to convert an unsigned magnitude into a string of digits in a specified radix, and then use this function to handle the magnitude part of each of the assigned functions.

Remember: There are two representations of zero (+0 and -0) in signed magnitude and 1's complement.



```
ARM Assembly
for Embedded Applications

Unsigned Interpretations:
Octal                200
Decimal              128
Hexadecimal           80

Signed Interpretations:
One's Complement    -127
Two's Complement    -128
Signed Magnitude      -0

8-bit Representation:
Binary          10000000

Lab 2B: Interpreting Binary
```