# CS270 Programming Assignment 4
# "LC3 Assembly Programming"

Program due Friday, October 28, 2011 (via Checkin by 2:59pm)
Revised 10/21/2011 for clarification to subroutine 3
Revised 10/26/2011 for correction to Successive Division Technique

Assignments are to be completed individually.
Discussing implementation specifics of the program is **forbidden**.

## Goals

In this assignment, you will write an LC-3 assembly program (with subroutines) to right shift the bits of an LC3 word depending on two other input numbers.

## The Assignment

Write an assembly program that takes two LC-3 words **TOSHIFT_X**, and **TOSHIFT_Y**, and two IEEE 16-bit floating-point exponent values **X** and **Y**.

- The program first determines which of X, Y has the smaller exponent and then computes the difference. Note that you need to compute absolute difference.
- Next, depending on which of X, Y has the smaller exponent, right shifts the bits in either TOSHIFT_X or TOSHIFT_Y. For example, if X happens to be smaller than Y, then the program should right shift the bits in TOSHIFT_X.
- The amount that needs to be right shifted is given by the absolute difference of X, Y (i.e., |X-Y|)
- The result after shifting either TOSHIFT_X or TOSHIFT_Y must be stored into a variable called **SHIFTED**.

Note that X and Y are not 2's complement numbers, but integers in the format of IEEE 16-bit floating-point exponents we have been working with this semester. For a reminder, see the Wikipedia article at: http://en.wikipedia.org/wiki/Half_precision_floating-point_format

**Note on X and Y**:
X and Y are LC-3 words that only represent the exponent part of the 16-bit floating-point number. Do not confuse X, Y with a complete 16-bit floating-point number. In other words, only bits b0 through b5 should be used in either X or Y; the other bits should be 0. We will not test your programs with X or Y > 31.
For example:
Suppose X is: 0000 0000 0010 000.
The IEEE exponent value presented by X is: +16 (binary: 10000)
The actual exponent value is: 16 – 15 = +1 (note: bias is +15 in 16-bit IEEE floating-point format)

Example:
TOSHIFT_X= 0000 0000 1100 1000
TOSHIFT_Y= 0000 0000 0000 1000
X = 0000 0000 0001 0000 (=16 in IEEE 16-bit exp format)
Y = 0000 0000 0000 1110 (=14 in IEEE 16-bit exp format)

First, we see that XA = 16 – 15 = +1 and YA = 14 – 15 = -1. Now compute number of right shifts, RS = |+1 – (-1)| = 2. Next, figure out which of X, Y have the smaller exponent; in this case it would be TOSHIFT_Y. Finally, shift the word in TOSHIFT_Y to the right twice and store it in SHIFTED. The result in SHIFTED will be: 0000 0000 0000 0010. Note that right shifting a value will cause the least significant bits to be truncated. This is normal--it is exactly what happens in floating point addition.

You must implement the assembly program using subroutines described below. Note that you may declare additional subroutines as needed (for example, to compute which of X, Y has the smaller exponent)

**Subroutine-1:**

Name: **IEEE_EXP2ACTUAL**
Input: **IEEE_EXP**
Output: **ACTUAL_EXP**

The subroutine (which starts with the label **IEEE_EXP2ACTUAL**) takes an LC-3 variable called **IEEE_EXP** that is in IEEE 16-bit floating-point exponent form, and returns the actual value of the exponent in 2's complement form in an LC-3 variable called **ACTUAL_EXP.** For example: if IEEE_EXP = 16, then ACTUAL_EXP = 1. Note that the input to this subroutine is passed using variable **IEEE_EXP** and output is returned in variable **ACTUAL_EXP**.

**Subroutine-2**:

Name: **ABS_DIFFERENCE**
Input: **A**, **B**
Output: **R_ABS**

The subroutine (which starts with the label **ABS_DIFFERENCE**) takes two LC-3 variables called **A** and **B** that are in 2's complement form, computes the absolute difference of the numbers, and returns the result in variable **R_ABS**. Note that the input to this subroutine is passed using variables **A** and **B** and output is returned in variable **R_ABS.**

**Subroutine-3:**

Name: **RIGHT_SHIFT_SD**
Input: **RS_AMOUNT**, **RS_IN**
Output: **R_SHIFTED**

The subroutine (which starts with the label **RIGHT_SHIFT**) takes two LC-3 variables **RS_AMOUNT** and **RS_IN**, and right shifts the value in **RS_IN** by the amount in **RS_AMOUNT** and stores the resulting right shifted number into the variable **R_SHIFTED**. The right shifting process must be implemented using the successive division technique (described below) in this subroutine. Please note that we will only test this subroutine using a positive, 2's complement value for RS_IN (i.e., your subroutine does not need to handle the case when Bit 15 is 1.

Successive Division Technique:

1. Compute *divisor* = $2^{(RS\_AMOUNT)}$ using left shifts.
2. Set *result* = 0
3. Increment *result* by 1
4. Subtract *divisor* from RS_IN
5. If Zero or Positive, repeat from step 3.
6. Otherwise, subtract 1 from *result*
7. Store *result* into R_SHIFTED

**Subroutine-4:**

Name: **RIGHT_SHIFT_LSWA**
Input: **RS_AMOUNT**, **RS_IN**
Output: **R_SHIFTED**

This subroutine takes two LC-3 variables **RS_AMOUNT** and **RS_IN** and right shifts the value in **RS_IN** by the amount in **RS_AMOUNT** using 'left shift with wrap-around' operation. You will need to work out the algorithm for this subroutine. Note that left-shift with wrap around means, any bits that are shifted out of the MSB are inserted back into the LSB.

**Main program:**

As described earlier, the main part of your assembly program takes four input variables **TOSHIFT_X**, **TOSHIFT_Y** and **X**, **Y**, and calls the subroutines to right shift the value inside either of TOSHIFT_Y or TOSHIFT_X and stores the result in output variable **SHIFTED**. You can use either of **RIGHT_SHIFT_SD** or **RIGHT_SHIFT_LSWA** to right shift the bits.

**Rules:**

- You must use variables in your program named **X**, **Y**, and **TOSHIFT_X** and **TOSHIFT_Y** for the input variables and use **SHIFTED** for the output variable.

- You must name the subroutines and the input variables to these subroutines according to the description above. Note that each subroutine will be tested independently of the main program. You will lose points if you do not follow these guidelines

- You must write your program such that that these variables can be accessed by all the functions in your program that need them (therefore the total length of your program must be "small enough."

**Submission Instructions**

1. Name your assembly file rightshift.asm and store it in a directory called PA4.
2. Submit a tar.gz file generated from the following command (assuming you are running this command from inside PA4 directory):
   `$ cd ..; tar –czvf PA4.tar.gz PA4`

Do not submit the assignment with lc3tools inside PA4. You may lose points if you do so.

**Grading Criteria**

Points will be awarded as follows:

- Implementing Subroutine-1: 15 points
- Implementing Subroutine-2: 15 points
- Implementing Subroutine-3: 20 points
- Figuring out the algorithm and implementing Subroutine-4: 25 points
- Implementing main: 20 points
- Following submission instructions and style: 5 points

**Late Policy**

Late assignments will be accepted up to 48 hours past the due date, but with 10% deduction for every 24 hours late. Assignments will not be accepted past this period. If you were unable to submit via Checkin for any reason, contact us via email and we may be able to help.