**Before Doing Anything Else...**

1. Create a new directory called HW12 under your PennSim directory.
2. Download this file [[HW12.asm](https://canvas.wisc.edu/courses/201484/files/13265184/download)] to your HW12 directory (right-click the link and choose "Save As" or something similar from the context menu). It will be the starting point for your programming assignment.
3. Modify the comment at the top of the **hw12.asm** file to replace "NAME" with your full name as it appears in Canvas.
   * If you are working with a partner, the submitting partner's name should be listed first, and the non-submitting partner's name should be second.

**Assignment Description**

This code will make use of some of the routines you wrote and debugged in HW11.  Specifically the **MULT** routine and the **GET\_NUM** routine.

The main routine is provided and simply prompts the user to enter a number, calls a routine to calculate the square root of the entered number, and then displays the result.  You will write the two new routines **SQRT** and **DISP\_NUM**.

The number the user enters to take the square root of has to be in the range of 0-32767.  Positive range for 16-bit signed numbers.

How do you calculate the square root of a number on a ISA as lame as the LC-3?  By successive approximation with multiplication.  You have already written a multiply routine.

**SQRT**

The square root of a 16-bit number will fit in an 8-bit number.  The **SQRT** routine use **R3** as the input argument (**R3** will hold the number your are taking the square root of).   You start with a "guess" that has the MSB (of an 8-bit quantity) set (i.e. 0x0080).  You square it using your **MULT** routine.  Is the product > **R3**?   If so then 0x0080 was too big, so you clear that bit and set the next bit down (i.e. you are testing 0x0040 next).  If, however,  the product ≤ **R3** then you would keep that bit set and set the next bit down (i.e. you are testing 0x00C0 next).  You iterate using this method keeping or clearing bit by bit and testing the next bit down until you have decided to keep the LSB set or clear it.  So essentially in 8 iterations with 8 multiplies you are done.

You will need a mask pattern (0x0080, 0x0040, 0x0020, ...) to help you implement this algorithm.  Unfortunately the LC-3 is so lame it does not even have a right shift operation, so all of these masks will have to be stored in memory.  If you look at the provided template for HW12.asm it has a bunch of .FILL statements already provide for you to serve as these masks.

DISP\_NUM

So now you have the numeric value (square root of entered number) in register.  How do you display that as a decimal number on the console?  It is actually quite a pain.  It is made a bit easier by our knowledge that it can at most be a 3-digit decimal number (the largest value we can take the square root of is 32767, so our largest result to display would be 181).  We are also going to make our job a little easier by allowing leading zeros to be printed.  So if the answer was 9 we can print 009.

Essentially you have to process it place by place starting with the hundreds place.  So we first have to figure out how many times we can subtract 100 from the number.  That forms our first digit to print.  Of course we must be sure to convert that digit to and ASCII character by adding 0x0030 hex before **OUT**putting to the console.  After subtracting as many 100's as possible we need to next see how many 10's can be subtracted to form the next digit.  Then cast that digit count to ASCII and **OUT**put it as well.  Finally you have the remaining 1's and you simply cast that to ASCII and **OUT**put it.

When writing the subroutines be careful about context save/restore.  Recall that a subroutine that calls another subroutine has to also save **R7**.

Test your code thoroughly.  We will be testing your code directly in PennSim so it had better compile and run.

**Submit your completed HW12.asm to this dropbox.**

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Filename: hw12.asm

; Author: NAME - UWID

; Description: Prompts user to enter decimal number then compute the square

; root (through successive approx) and displays the result

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.ORIG x3000

START

; Prompt user to enter a decimal number

LEA R0, PROMPT ; get address of prompt

PUTS ; display prompt for number entry

JSR GET\_NUM ; number returned in R1

ADD R3, R1, #0 ; move number to R3

; Compute square root and display

JSR SQRT ; compute R2 = sqrt(R3) via call to SQRT

LEA R0, RSLT\_STR ; get address of result display string

PUTS ; display result display string

JSR DISP\_NUM ; wait for a key press

; Do it all again

BR START ; jump to start of program

;; Reserve Prompt strings

PROMPT .STRINGZ "\nEnter number (0-32767):"

RSLT\_STR .STRINGZ "The square root is:"

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Subroutine: DISP\_NUM

; Description: Displays up to a 3-digit decimal number as ASCII to console

;

; NOTE: Use successive subtraction to figure 100's, 10's, 1's places

;

; NOTE: it is OK to print with leading 0's. If R2 was 23 could print 023

;

; Assumes R2 is argument of number to display (0-999)

; Returns Nothing

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DISP\_NUM

; YOUR CODE GOES BELOW HERE

; context save

; initialize

LD R3, ASCII\_ADD ; 0x0030 to convert cnt to ASCII char

; hundres place computation & display

; tens place computation and display

; ones place display

; context restore

RET

; allocate for context save and constants

< NEED .BLKW here for context save space >

PLACE100 .FILL #-100

PLACE10 .FILL #-10

PLACE1 .FILL #-1

ASCII\_ADD .FILL x0030

; YOUR CODE GOES ABOVE HERE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Subroutine: SQRT

; Description: Computes sqrt(R3) via successive approximation. It starts with

; MSB set (0x0080) and multiplies. If product is greater than R3

; the bit is cleared and the next bit is set and tested (0x0040).

; If, however, the product was less than R3 the bit would be kept

; set and the next bit would be set and tested (0x00C0). This

; process continues 7 more times till all bits have been tested

; and set/cleard

;

; NOTE: Assumes input argument R3 is in range (0-65535)

;

; Assumes R3 is argument to take sqrt of

; Returns R2 = sqrt(R3)

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SQRT

; YOUR CODE GOES BELOW HERE

; context save

; initialize

; compute sqrt using successive approx and check

; context restore

RET

; allocate for context save

SQRT\_R0 .BLKW 1

SQRT\_R1 .BLKW 1

SQRT\_R3 .BLKW 1

SQRT\_R4 .BLKW 1

SQRT\_R5 .BLKW 1

SQRT\_R7 .BLKW 1

MASK\_BASE .FILL x0080

.FILL x0040

.FILL x0020

.FILL x0010

.FILL x0008

.FILL x0004

.FILL x0002

.FILL x0001

.FILL x0000

; YOUR CODE GOES ABOVE HERE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Subroutine: MULT

; Description: Multiplies R1 \* R2, returns in R0

;

; NOTE: Does not need to work for negative values of R1 and R2, but

; should work for zero values

; NOTE: Can assume product will never exceeed 16-bit value (<65535)

;

; Assumes R1, and R2 are multiplier and multiplicand

; Returns R0 - the product of R1 \* R2

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

< Fill in your MULT routine from HW11 here >

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Subroutine: GET\_NUM

; Description: Returns decimal number user entered in R1

;

; NOTE: Processes characters till it sees x000A (new line) (Enter Key)

; User should only enter decimal digits and numbers less than 255

; Routine does not do any error checking of bad user data

;

; NOTE: This routine does call another routine (MULT10)

;

; Assumes Nothing

; Returns R1 - the decimal number user entered

;

; R0 will hold result of GETC

; R2 is general purpose temporary register

; R3 will hold -0x000A for compare to new line

; R4 will hold -0x0030 for casting char to num

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

< Fill in your GET\_NUM routine from HW11 here >

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Subroutine: MULT10

; Description: Multiplies number in R1 by 10

;

; Assumes R1

; Returns R1 => R1\*10

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MULT10

; context save

ST R0, MULT10\_R0

; perform computation

ADD R0, R1, R1; ; have 2X now in R0

ADD R1, R0, R0 ; have 4X now in R1

ADD R1, R1, R1 ; have 8X now in R1

ADD R1, R1, R0 ; have 10X now in R1

; context restore

LD R0, MULT10\_R0

RET

MULT10\_R0 .BLKW 1

.END