

EET363 Introduction to Microcontrollers

OIT Portland West, Fall 2010

Lab Assignment #2 — Assembly Language Programming Due October 28

Objective: The student will practice programming the 68HCS12 microcontroller, performing arithmetic calculations, iteration, and accessing arrays.

Project Description: Something (consider it a *deus ex machina*, if you must) has stored width of a square in centimeters into a 16-bit word at the start of data RAM (location \$1000). You are calculate the area in square inches and store as an ASCII character string starting at location \$1002 in an array that is 6 bytes long. Then send the characters to the terminal emulator so the temperature will appear on the PC display. The program should then stop by executing the *SWI* instruction, which will cause a return to D-Bug12 or stop the simulator.

Equipment and Software needed:

- Everything you used in Lab Assignment 1

However, you may do this lab with the simulator only (no Dragon12-Plus board) if you wish. It will take an additional step, outlined in Part 3.

General Instructions: The program is best divided into three parts. First, the calculation of the area and conversion of units. Second the conversion from a of the area value to the character string. The third part sends the characters to the display terminal.

When you test your program, you need to take on the role of the *deus ex machina* and initialize the width. Use two different test cases to run the program and compare the result with the calculation performed by hand or using a calculator.

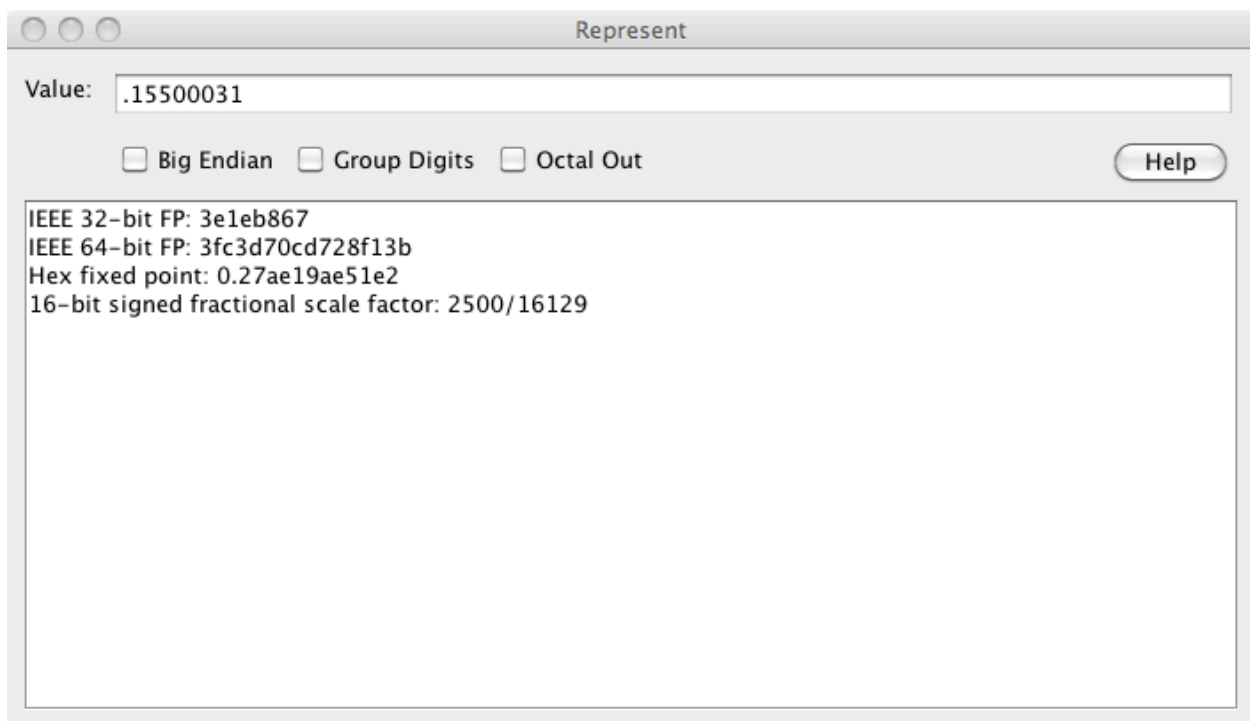
In the source code, include the registers.inc file at the start of the program. The statement "*#include registers.inc*" will do this. The # character should be in the first column of the line. This file defines constants for the start of the data RAM, *DATASTART*, and program, *PRSTART*, which are at \$1000 and \$2000 respectively. The start of the source code file should look something like this:

```
#include registers.inc
                                org    DATASTART
width:      ds      2           ; width in cm goes here
result:    ds      6           ; six characters for our result
                                org    PRSTART
; Code starts here...
```

PART 1 – Calculation of area and unit conversion

You can't have negative widths, so consider all values to be unsigned. The area of a square is coincidentally the square of the width. This could be a 32 bit value, but we will assume that the width will be small enough (255 cm or less) that area will fit in 16 bits. To help you out, I looked up the conversion from square centimeters to square inches. You will need to multiply by .15500031.

This may sound like an impossible task, but can be accomplished with multiplying by 2500 then dividing by 16129. Now you may wonder how I knew that. I used the program, *Represent*, that is provided on the CD and ends up on your computer's Start Menu. It's convenient for many calculations one ends up performing when programming microcontrollers (or other embedded programming tasks).



You should check your program at this point to insure it is doing the temperature conversion correctly.

PART 2 – Converting to the Character String

The character string is 6 bytes long. You will have to fill all six bytes even if the temperature value doesn't need 6 characters to be represented. To do this you will want leading space characters, so, for instance, 100 would be "◇◇◇100" and 12345 would be "◇12345" where the "◇" represents the space character.

The textbook has the algorithm to convert a value to digits (each character represents a digit), however we need to be a bit more creative because the ASCII

characters for the digits 0 through 9 don't have the values 0 through 9 but instead have the values 48 through 57.

So the algorithm is:

1. Initialize by setting P to the address of the last byte in the character array and V to the area in square inches.
2. Divide V by 10, putting the quotient in Q and the remainder in R .
3. Store R plus the value of the ASCII character '0' in location P .
4. Subtract 1 from P .
5. Put Q in V . If V is non-zero, go back to step 2.
6. If P is less than the address of the start of the character array go to step 8.
7. Store the value of the ASCII character ' ' (space) in location P , then subtract 1 from P . Go back to step 6.
8. You are done!

You should check your program again at this point to make sure that it is generating the correct string.

PART 3 – Displaying the string

Add to your result for part 2 the code to display the area to the terminal. Verify operation with two different widths and put your results (including the correct values you have calculated) in your report.

The following assembler instruction will write the byte in register B to the terminal:

```
jsr    [Putchar--4,pc]
```

This executes code within Dbug-12 that sends the byte to the terminal. The contents of other registers may change as a result of executing this instruction.

Your program should write the six characters in the character array to the display, then should write the CR (carriage return) and LF (linefeed) codes so the cursor will be on the next line.

If you use the simulator, you will have to start it in minidbug12 mode rather than student mode. In this mode using the simulator is slightly different than in student mode used earlier. After loading your program, issue the Reset command under the Files Menu, and then press "Go". When the program stops, it will be ready to start executing your program at location \$2000 (RAMSTART). The Putchar functions is implemented. Read the start of the DBUG12.ASM file for full information on use prior to attempting to run your program on the simulator. I recommend using the Dragon12-Plus board at this point.

You need to turn in your final commented program and screen captures showing the results of running the program.