

**ECE471: Embedded Systems – Homework 3**  
Linux Assembly and Code Density

**Due: Friday, 28 September 2018, 10:00am EDT**

1. Use your Raspberry-Pi to work on this project.
  - Note, you can optionally work on this homework as part of a group of two students. It is fine to work on the homework alone. If you work in a group, make sure the names of both group members are in the README file.
  - Download the code from:  
`http://web.eece.maine.edu/~vweaver/classes/ece471/ece471_hw3_code.tar.gz`  
and copy it to the Raspberry-Pi.
  - Uncompress/unpack it with the command `tar -xzvf ece471_hw3_code.tar.gz`
  - Change into the `ece471_hw3_code` directory `cd ece471_hw3_code`
  
2. **Modify the `exit_asm.s` file to return the value 42. (1 point total)**
  - (a) Modify `exit_asm.s` where it tells you to add code
  - (b) Be sure to comment your code!
  - (c) Run `make` to generate an updated version
  - (d) To test, run `./exit_asm` followed by `echo $?` which will show you the last program's exit status.
  - (e) Some reminders about Linux GNU assembler (`as`) syntax:
    - `.equ IDENTIFIER, value` sets a macro replacement, like
    - `#define IDENTIFIER value` would in C
    - You can use `@` to specify a comment, like `//` in C
    - You prefix a constant value with `#`
    - (to move the number 5 into a register you would do `mov r0, #5`)
  - (f) Reminders about the Linux ARM EABI:
    - Arguments go in `r0` to `r6`
    - System Call Number goes in `r7`
    - Use `swi 0x0` to trigger a system call.
  
3. **Modify `hello_world.s` (3 points total)**
  - (a) Look at the provided decimal printing routine, `print_number`. Read the code to see how it works. Two comments are missing, labeled #1 and #2. Put into the README file appropriate comments for those two lines.

Note: do not just describe what the instruction does (“adds 10 to `r0`”) but try to conceptually say what the code is doing, i.e. something like “convert to degrees Kelvin” or “move pointer back to beginning of string”.

- (b) Modify the `print_string` routine so it works.  
 You'll need to add code so that it counts the number of characters in the string pointed to by `r1`, stopping at the first NUL (0) byte. Then store this count in `r2`. (I'm not grading on code density, but it is possible to do this with 5 instructions. It's probably not possible to do it in 4, but if somehow you manage to do so, let me know.)
- (c) After the above is done, after running `make` then running `./hello_world` should print  
`0: ECE471 is cool`
- (d) Now modify the code to loop from 0 to 15 printing the message with the first number changing from 0 to 15. (Much like the C example in HW#2). If your running code gets stuck in infinite loop, Control-C can break out of it.

#### 4. Convert the `print_string` routine to 16-bit THUMB code. (2 points total)

- (a) Copy your working code to `hello_world.thumb.s`  
`cp hello_world.s hello_world.thumb.s`
- (b) Modify the `print_string` routine to be THUMB code  
 First, comment out the `.syntax unified` line at the beginning. Next uncomment the `@.thumb` line to be `.thumb` and the `@.arm` to be `.arm`.
- (c) You can then try running `make`
- (d) If you get errors you will need to change things so that routine is only using 16-bit thumb instructions. Remember, no fancy addressing, no accessing registers above `r7`, no conditional execution.
- (e) HINT: Don't forget that you need to use `blx` when jumping to THUMB code otherwise your program will crash.  
 Extra HINT: be sure \*all\* locations that call the function get changed to `blx`

#### 5. THUMB2 code

Since `arm32` and `thumb2` code are compatible, you should also have a `hello_world.thumb2` executable by now, generated from the `hello_world.s` file. If you are running on a Pi2 or Pi3 it should run too.

#### 6. Something cool: (1 point total)

Copy your code to `hello_world.extra.s`  
`cp hello_world.s hello_world.extra.s`  
 and do one of the following:

- Easy: Make the counts backwards from 15 to 0
- Moderate: Print the count in hexadecimal
- Hard: Print lines in colors like HW#2
- Very hard: `read` a number from STDIN and print the message that many times.

## 7. Questions to Answer: (2 points total)

Put the answer to these in the README.

- (a) Compare the size of the ARM32 `hello_world` executable, the THUMB `hello_world.thumb`, and the THUMB2 `hello_world.thumb2` executables. (list all 3 values)  
You can get filesize with `ls -l` (that's a lowercase L) You will want to run the `strip` command on the executables first (i.e. `strip hello_world`) or your results might be unexpected.
- (b) Compare the size to that of the C executable you made for HW#2.
- (c) Which language might you use in space constrained system? Why?
- (d) Which piece of code was easier/faster to write, the C or assembly one?

## 8. Linux Command Line Exploration (1 point total)

Try out the `cal` program. This prints a calendar, by default the current month. You can also `cal 2016` or `cal 12 2016`. Beware not to do `cal 16` as that will give you year 16, not 2016.

- (a) Run `cal 9 1752` Is there a bug here? Can you explain what is happening?

## 9. Submitting your work.

- Run `make submit` which will create `hw3_submit.tar.gz` containing the various files. You can verify the contents with `tar -tzvf hw3_submit.tar.gz`
- e-mail the `hw3_submit.tar.gz` file to me by the homework deadline. Be sure to send the proper file!