ECE256 Computer System Interfaces, Fall 2020
Lab 1 – Exercise
Introduction to ARK Keil MDK

For all labs in ECE256:

- Use an NXP FRDM-KL25Z board with ARM Keil MDK v5.31
  NOTE: A free version of Keil MDK can be downloaded and installed on your laptop or PC. Keil MDK v5.31 is installed on the PCs in the Embedded Systems Laboratory (MK228).

- **Write a short (2 page minimum) report detailing your result. Before submitting the lab report, you must also demonstrate your working lab to either the TA or professor, at which point they will make not of your successful demo**.

***Task 1*:** Work through all steps of the Lab 1 Tutorial (ARM Keil MDK). (35%)

Calling Functions and Passing Arguments – When a function calls a subroutine, it places the return address in the link register LR. The arguments (if any) are passed in registers R0 through R3, starting with R0. If there are more than four arguments, or they are too large to fit in 32-bit registers, they are passed on the stack.

Temporary Storage – Registers R0 through R3 can be used for temporary storage if they were not used for arguments, or if the argument value is no longer needed.

Preserved Registers – Registers R4 through R11 must be preserved by a subroutine. If any must be used, they must be saved first and restored before returning. This is typically done by pushing them to and popping them from the stack.

Returning from Functions – Because the return address has been stored in the link register, the BX lr instruction will reload the PC with the return address value from the LR. If the function returns a value, it will be passed through register R0.

1. Run the program from the tutorial until the opening brace in the main function is highlighted. Open and examine the Registers Window (View -> Registers Window). What are the values of the stack pointer (R13), link register (R14) and the program counter (R15)?
2. Open and examine the Disassembly window (View -> Disassembly Window). Which instruction does the yellow arrow point to, and what is its address? How does this address relate to the value of PC?
3. Step Over once using the F10 key while the Disassembly Window is open. Which two registers have changed (they should be highlighted in the Registers Window), and how do they relate to the instruction just executed?

4. Look at the instructions in the Disassembly Window. Do you see any instructions which are four bytes long? If so, what are the first two?
5. Continue execution (using F10) until reaching the BL.W my_strcpy instruction. What are the values of the SP, PC and LR?
6. Execute the BL.W instruction (use F11 this time). What are the values of the SP, PC and LR? What has changed and why? Does the PC value agree with what is shown in the Disassembly Window?
7. What registers hold the arguments to my_strcpy, and what are their contents?
8. Open a Memory Window (View -> Memory Windows -> Memory 1) with the address for src determined above. Open a second Memory Window (View -> Memory Windows -> Memory 2) with the address for dst determined above. Right-click on each of these memory windows and select ASCII to display the contents as ASCII text.
9. What are the memory contents addressed by src?
10. What are the memory contents addressed by dst?
11. Single step (F11) through the assembly code watching memory window 2 to see the string being copied character by character from src to dest. What register holds the character?
12. What are the values of the character, the src pointer, the dst pointer, the link register (R14) and the program counter (R15) when the code reaches the last instruction in the subroutine (BX LR)?
13. Execute the BX LR instruction. Now what is the value of PC?
14. What is the relationship between the PC value and the previous LR value? Explain.
15. Now step through (F11 or F10) the my_capitalize subroutine and verify it works correctly, converting b from “Hello micro!” to “HELLO MICRO!”

***Task 2*:** For this task you will write an assembly code subroutine to approximate the square root of an argument using the bisection method (see Wikipedia for details). All math is done with integers, so the resulting square root will also be an integer. (65%)

Requirements – Your code must approximate the square root of an integer between 0 and 2^{31}-1. Using integer math is sufficient (no floating point or fractions are required); your code will return the truncated (integer portion) of the square root.

Your code must be in an assembly language subroutine which is called by a C function for testing (see Tutorial 1 for examples). Be sure to use registers according to the ARM calling convention.

Software Design – Base your software on the following pseudocode:

```
Approximate square root with bisection method
INPUT: Argument x, endpoint values a, b, such that a < b
OUTPUT: value which differs from sqrt(x) by less than 1
```

```
done = 0
a = 0
b = square root of largest possible argument (e.g. ~2^16).
c = -1
do {
        c_old <- c
        c <- (a+b)/2
        if (c*c == x) {
                done = 1
        } else if (c*c < x) {
                a <- c
        } else {
                b <- c
        }
} while (!done) && (c != c_old)
return c
```

Testing – In the main function, write code to test that your subroutine works correctly.

Deliverables:

- Answers to the questions listed in Task 1.
- Code (C/Assembly) for your square root subroutine – **include as Appendix A in report**
- Code (C) for the main function used to test the subroutine – **include as Appendix B**
- Snapshots of Stack Window when running your square root subroutine – **in writeup**
- **TA or professor note that lab was successfully demonstrated**