

Due: **Friday, November 9 at 9:00 am.**

Submission:

- (1) Please hand in a (tidy) paper copy of problem 1 at the beginning of class.
- (2) Please email your code for problem 2 as an attachment (named hw6.asm) to **submit211@cs.grinnell.edu** by the beginning of class.

1. For this problem, you will do some reverse engineering, converting executable code to assembly. The following is executable code for the AVR, given in hexadecimal format.

```
2091640030E0442743954217340F
```

Your task is to decode it. Specifically, give the corresponding list of instructions

- (i) in binary, and
- (ii) in AVR assembly language.

Hint: The only instructions included are ADD, CLR, CP, INC, LDI, and LDS.

To help make your solution readable, please put each binary instruction on a separate line, and please put a space between each nibble. Your binary solution should be given in big-endian order.

Recall that the binary form of each instruction is given in the “AVR Instruction Set” document that I posted on the course website. Note that the binary forms are given with b_{31} on the far left and b_0 on the far right. Even so, remember that the AVR is a little-endian computer. Thus, the executable string above is little-endian.

(Note that the executable code was produced by avra, the assembler we are using. When you have correctly decoded it, if you then assemble your solution, this string of hex characters should be in the output, but it probably won't be the full output.)

2. Write the assembly program described in Exercise 3 of Laboratory 8 (held on Tuesday, Oct 30). However, for this assignment, your program should use a function to implement the delay. Your function should accept one argument, the high-order byte of the counter that controls the delay, and that argument should be passed on the stack.

For (a small amount of) extra credit, write your function such that it is completely “safe” (i.e., it could be called from anywhere in a large program, regardless of which registers were in use by the calling function, and upon return all registers would retain the values they had when the function was called.)