Due: Friday, October 28 at 5:30 p.m.

Submission: For problems 1-3, hand me a paper copy or slide it under my office door (SCI 3809). For problem 4, please attach a printout of your code. Please also email me your code as an attachment; the attachment’s filename should begin with your last name.

Problems

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 (that is, the most significant byte should be on the left, and the least on the right).

   Recall that the binary form of each instruction is given in the detailed AVR Instruction Set posted on the course website. Note that the binary instructions given in the documentation are given in big-endian order. But remember that the AVR uses little-endian addressing (with the least significant byte before the most significant byte), so the executable string above is little-endian.

   The hex code code above was produced by avra, the assembler we are using. You can check your work by assembling the instructions you write. When you have correctly decoded it, the string of hex characters will appear in the assembler’s hex output, but it won’t be the full output.

2. In AVR assembly, one of ADD, CLR, CP, INC, LDI, and LDS is a pseudoinstruction. Identify which one and explain what other instruction it maps to.

3. (a) Translate the following C code into AVR assembly.

   ```c
   void swap(byte v[], byte k, byte j)
   {
      byte temp;
      temp = v[k];
      v[k] = v[j];
      v[j] = temp;
   }
   ```

   (b) Name an addressing mode that ARM supports but AVR lacks which would have allowed you to write more concise assembly code.

4. Complete exercise 4 from lab 8. Please comment your code and use appropriate labels and register names.

Optional extra problems

Do something creative involving procedures in AVR and make an appointment to show me what you’ve done!