Brown University
School of Engineering
ENGN1640 Design of Computing Systems
Professor Sherief Reda
LAB 04 (75 points).
Due in lab to TAs by Friday, March 11.

BROWN

---

1. [60 points] The goal of this lab is to build a mini assembler for ARM code. You can use your favorite programming language (e.g., MATLAB, C, Java, python, etc) to code the assembler, though lab support might be diminished if you choose an exotic programming language to code this assignment. Your assembler should accept as input a text file that gives the assembly code and produces as an output a binary file that has the machine code. We will not build a full assembler in one week, but rather choose a subset of instructions to assemble. The subset in this case is **data operation** instructions, which makes coding the assembler easier as you do not need to map labels to addresses (as in the case for memory or branching instructions). Note that the mnemonics of the ARM assembly are chosen to making parsing easier as the instructions are specified using the first three letters. Any subsequent letters indicate whether the instruction is conditional and whether the operation will impact the CPSR register. Here are specifically the instructions that you need to be able to assemble: MOV, MVN, AND, ORR, EOR, BIC, ADD, SUB, ADC, SUBC, CMP, TST, MUL. Your assembler must be able to handle all possibilities for each of these instructions such as ADD*cdS*, where *cd* is the condition (e.g., EQ, NE, CS, CC, etc; check Appendix B for all conditions) for the instruction and *S* indicates whether it impacts the CPSR register. Furthermore, the assembler should be able to handle all ways that can describe the second operand, such as immediate, register, or shifted register (either by a constant or a register). Test your code by assembling this trace of sample code and verify its correctness by comparing it against the machine code that gets assembled automatically using ARMSim#.

   [15 points] Also devise two other test cases for your assembler and verify their correctness. How are you going to prove to the TAs that your code can handle all possible data operation assembly instructions correctly without bugs?

   Test code (just a sample):

```
CMP     R3, R4
ADDEQ   R0, R1, R2
SUBNE   R0, R0, R2
ORR     R9, R5, R3, LSR #2
ADD     R5, R6, R7
SUB     R2, R3, #0xFF
ADD     R0, R1, #42
SUBS    R2, R5, R1 LSR #3
ADD     R7, R8, R9 LSR R2
```