# Programming Project Part 3

University of Nevada, Reno

CS 219 - Computer Organization

# 1. Theory

In the first and second programming projects, your code processed commands from a text file and displayed the results of each command. In part 3 of this project, you will leverage the ALU functions from part 2 to process commands in assembly language format.

As before, the input for your simulator will come from a text file. Each line of input will contain one assembly language instruction. The format of these instructions will be similar to the ARM version 7 ISA we have studied in class. The instructions will operate on a set of eight 32-bit registers (named r0-r7) implemented in your code. Each instruction will be in one of these three formats:

MOV Rd, #IMM
Moves the immediate operand into Rd
Rd can be r0-r7
IMM can be any 32-bit constant expressed in hex with a 0x prefix

OPCODE Rd, Rn, Rm
Does the specified operation on Rn and Rm, stores the result in Rd
OPCODE can be ADD, SUB, AND, ORR, XOR, CMP, TST Rd,
Rn, and Rm can be r0-r7
Flag values will be affected if S is added with the OPCODE (ADDS, SUBS, ORRS, XORS)

OPCODE Rd, Rn, #N
Does the specified N-bit shi on Rn, stores the result in Rd
OPCODE can be LSR, or LSL Rd
and Rn can be r0-r7
Flag values will be affected if S is added with the OPCODE (LSLS, LSRS)

# 2. Implementation Details and Requirements

This project must be written in C or C++.

Your program should accept all strings (opcodes, register names, and immediate operands) using upper or lower case. As a suggestion, you might want to convert all letters to upper or lower case to make it easier to parse the commands.

Your program will implement the registers r0-r7. These registers should be implemented using the uint32_t type and should be initialized to 0x0 before your code begins reading its input. Changes made to a register by one instruction should persist and be available in subsequent instructions.

Flag values (NZCV) will also be initialized to 0 before your code begins. For each subsequent instruction update your affected flag values and keep other flag values like the previous state.

For example, for OPCODE without any S no flag values will be affected. In that case, the flag values will remain unchanged like the previous state. **(DO NOT MAKE THEM ZERO)**

Your code will read each input line, perform the specified instruction, and print the updated values of r0-r7 registers. You may format the output in any way that makes the results easy to view. At a minimum, the output must show which register was changed by the instruction, the instruction, and the new value for that register; all of which should be easily understood by anyone running the code.

# 3. Result

Your code should follow proper code commenting and indentation procedures. Please check the sample input and output file on canvas.

**Extra credit:**

**There will be an extra 10 points if you can show the register and flag values for CMP and TST OPCODE.**

You have to submit a README file with your working code. In your README file, make sure to describe how to run your code, your code's working process, and the result.

# 4. Submission Requirements

For submission of this assignment, you are expected to submit your working code at a minimum. All submissions are expected by the due date, and submission is done on canvas. Your code must be executable via remote computers so that the TAs can compile/run your code with ease. Make sure to contact the TAs if you have any confusion over working with remote computers.

No extensions will be given except as permitted by the course Syllabus. 30 minutes of grace will be given a er the due time, assuming the students may face internet/submission/device issues. However, after the grace period, all the submissions will be treated as LATE SUBMISSIONS.

IF YOUR SUBMISSION IS LATE BY 1 DAY, YOU WILL HAVE A 20% PENALTY.

LATE SUBMISSIONS AFTER THAT WILL BE ACCEPTED FOR AT MOST 50% CREDIT. In case of late submissions, the 30-minute grace in due time will not be given.

So, you need to submit a zipped file with the following items:

1. A running code (C/C++) with proper documentation on each part.
2. A README file with the necessary information on how to run this code.
3. You must describe your result and working process in your README file.
4. Your zip file should be named with your name as 'first-name_last-name_proj3'.
5. If you have multiple files such as c/cpp and header, provide a make file so that the TAs can compile and execute your project easily.

# 5. Grading

**Non-working code will result in a 0 score**

**(If you receive a zero on an assignment due to compilation issues on the grader's end, but the code runs correctly on your system, reach out to your TA with your running code to update your score)**

**56 points**
7 points will be given for each operation being handled correctly.

**20 points + 10 points**
20 points will be given if you represent registers r0-r7 and 10 points will be given if you represent flag values in an appropriate manner.

**14 points**
14 points will be distributed based on properly documented code and your description of your result and working process in the README file.

**10 Extra points**
If you work with CMP and TST OPCODE and update the register and flag values