Supplementary Material 3:
C code on your NYU-6463-RV32I

IMPORTANT: Only follow this guide if you have already completed the Supplementary Material Part 2: Getting assembly working. THIS IS NOT A REQUIREMENT FOR YOUR PROJECT TO GET FULL MARKS. IF YOU TRY COMPLETE THIS SECTION WITH A BUGGY PROCESSOR YOU MAY END UP WITH FEWER MARKS THAN IF YOU HAD JUST FIXED/FINISHED YOUR BASE PROCESSOR DESIGN.

In order to get C working it actually isn't a big step up from the previous document, as we already set up gcc and so on to assemble the assembly files. The major change at this point is actually architectural - we need to ensure that our hardware is able to *load data from instruction memory* (making it a "modified Harvard" architecture) with the LB, LH and LW commands. This is because the C compiler will allocate constant variables and so on to the instruction memory, and we need to be able to read these values as if they were data in order to use them in our program. If you have designed your processor well, you shouldn't need to change your main processor finite state machine.

There are two phases to this:

1. Extend your instruction memory to be *dual-port*, that is, it has *two* read addresses and *two* output data registers. The new read address is then wired in parallel to the data memory, with some additional *combinational* logic to deal with the address translation and multiplexing.
    a. Once you have finished this design and wiring, do not just try and write C code. First, *test the wiring* by writing an assembly program that reads out a known value (e.g. the instruction itself) into data memory. E.g. try loading the first word of your instruction program into the data memory, and make sure it matches the instruction in instruction memory. To perform this, use the load commands in assembly (LB, LH, LW).

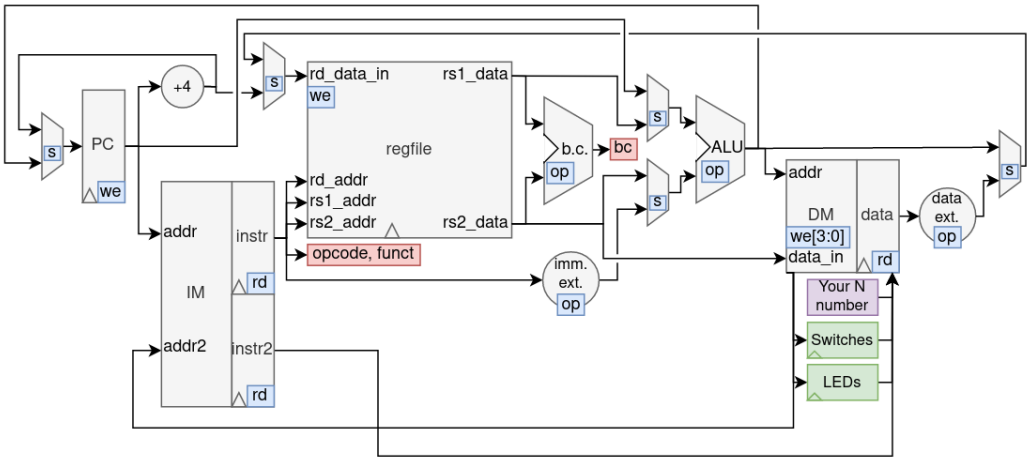

Figure 1: The extended instruction memory.

2. Once you have confirmed that it works, you are ready to move on to the next step, which involves the *set up* procedures for a program compiled in C to work. That is, we need to define the code that runs *before main()*. I have already written this code for you, but it is helpful to understand how it works.
    a. The first part of this setup sets up the *stack pointer* sp. This is required so that the C language model can allocate variables to the stack.
        i. The code required for this exists in start/nyu-6463-rv32i-boot.s in the standalone_c folder on brightspace.
    b. The second part of this setup copies in your initial variables to data memory from program memory.
        i. The code required for this exists in start/nyu-6463-rv32i-startup.c in the standalone_c folder on brightspace.

IMPORTANT: I have not provided instructions for using the C standard library, nor the files that may be required to have this work. Do not try #including stdio.h or any other libraries while writing your code.

The specific details for performing the above steps are included in the recorded video of the Dec 8 office hour.

You may also download from brightspace a "standalone_c" zip file which includes a main.c that you can alter and then run 'make' with to generate a 'main.mem' that you can include into your processor to execute.