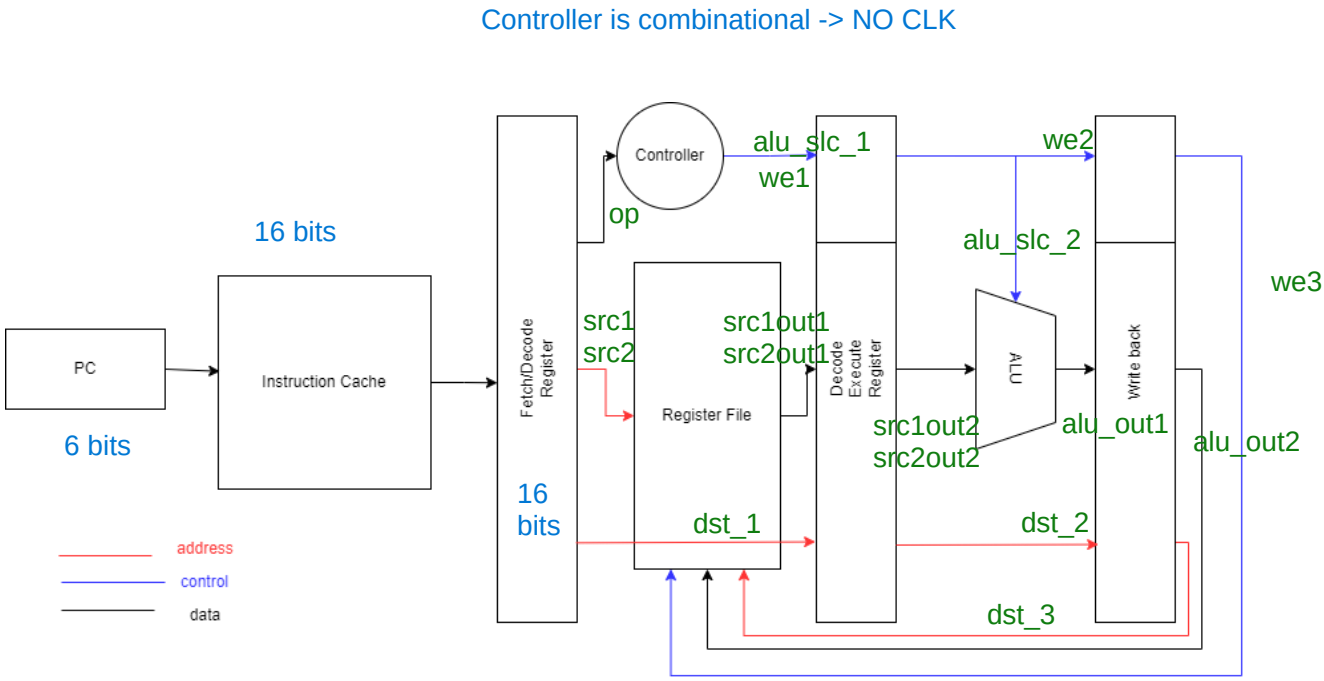
Cairo University
Faculty of Engineering
Computer Engineering

Computer Architecture
Lab 4

In this lab and the assignment, we will build a small pipeline like the one in the picture below:



- Use the ALU and Register files you created before.
- Also you will need the nDFF and Ram files from Lab3
- Our Instruction cache will be 16-bits width and has 2^6 entries. (what is the size of the address bus and what is the size of data bus and how many ports do we need.) (Hint: we don't write in instruction cache)
- A 6-bit counter (PC) that starts at 0 at reset and increments with 1 each clock cycle. The counter increments only if the enable is set to 1. (Hint: you can use the '+' sign or the adder, up-to-you)
- Each instruction consists of
    - 3-bits opcodes, 3-bits for src1, 3-bits for src2, 3-bits for destination, and the rest are unused bits.

    - 3-bits opcodes, 100 represents Not, 000 represents shift left and 010 represents NOP (no operation)
- The controller **reads** the opcode from Fetch/Decode register and **generate** the following control signals: the **ALU operation** (the ALU selectors from lab2) and the **write_enable** signal. The generated signals should be written in to the next pipeline buffer.
- The register file should read the read address1 and read address2 from the instruction and write the output to the next pipeline register. It also should read the write_address, write data and **the write_enable from the write_back register.**
- The write address should propagate from Fetch/Decode register to the Decode/Execute register.
- The ALU should read the selectors from the control signals and data stored in the Decode/Execute register. It should write its output to the write_back register.
- The write_enable and write_address should also be propagated from the Decode/Execute register to the next register.

Lab Requirements:

1- Implement PC in a separate file
2- Create an integration file the instantiate component from the memory as instruction cache, component from nDff as the Fetch/Decode Register and the PC. The file should integrate them together.
3- Convert the following instructions into OPCodes and save them to lab.mem
    a. Not R1
    b. Not R2
    c. Not R3
    d. Nop
    e. SHL R1
4- Create a “testbench” and “do file” testing the above scenario, you should reset at the first cycle, then run the simulation till all the instructions are read correctly.

Assignment:

1- Create the controller for the given instructions only.
2- Modify either register file or pipeline registers to work on opposite edges (for example, if pipeline registers work on +ve edge then the register file should work on the -ve edge and vice versa)
3- Create an integration file and connect all the components as shown above
4- Create a do file that run the scenario given in the lab and show the following signals in the wave:
    a. Clk
    b. Rst
    c. All Registers output
    d. PC

    (Hint.1: Don't forget to load the memory with instructions at the beginning of the simulation and don't forget to reset your processor as well).