MIPS Pipeline

Complete this HW with your final partner.

PROVIDED MATERIALS:

- start.circ (initial basic pipeline with no hazard handling)
- test_simple.dat (simple program you can load into instruction memory, contains no pipeline hazards), program instructions given below:
    - r0 = r0+1
    - r1 = r1+5
    - r2 = r2+5
    - r3 = r3+6
    - r0 = r0+2
    - r1 = r1+2
    - r2 = r2+2
    - r3 = r3+2
- test2.dat (simple program you can load into instruction memory, contains no pipeline hazards), program instructions given below:
    - r0 = r0+1
    - r1 = r1+3
    - r2 = r2+5
    - r3 = r3+0
    - r0 = r0+0
    - r1 = r1-r0
    - sw, r2 2(r3)
    - lw r3, 0(r3)
    - beq r0, r0, 7

Deliverables: A zip folder named with both of your usernames (please name the original folder with your usernames BEFORE zipping it. Your folder should contain 2 files – your nop-handling datapath in the file *nop.circ* and your final hazard-handling datapath in the file *final.circ*.

TEST STARTING DATAPATH

- *** See last datapath HW for description of instruction set and format ***
- It's important to understand well how the current basic pipeline works before attempting to modify it. I have made several "black-box" units by putting the circuitry into a subcircuit. At first trust that these black-box units do what they should. Read the **Simulation** section of the Logisim tips below, then load test_simple.dat to instruction memory and run it until you understand just the addi instruction.
- Now look at the subcircuits in more detail. They include
    - The pipeline registers are just collections of registers for all of the values that cross between stages.
    - The register file is the same black box we've been using, just now in a subcircuit.
    - The control is split into 2 subcircuits, one for the initial deconstruction of the op code (in the decode stage), and the other to choose to branch or not (can't happen until execute stage).

  Read the **Subcircuit** section of the Logisim tips below. Then go through each subcircuit (click on them in left menu) and understand what each one does, especially the 2 control units.
- Besides subcircuits, another way to clean up the circuit diagram and avoid the risk of crossing wires is to use tunnels. This simply hides part of a line/wire as if it tunnels below the visible circuit for awhile. Read the **Tunnel** section of the Logisim tips below and look at how the clock tunnels work. I have also added a tunnel for the regW control value that needs to be sent backwards from the last stage in the pipeline to the decode stage. Find the other end of that tunnel.
- Now test and follow the rest of the instructions through the pipeline. Load and run test2.dat which has several more instructions, still with no hazards. Make sure you actually use <cmd> r or "reset simulation" in the menu each time, then reload the test file to instruction memory, otherwise you may have lingering values from the previous test causing problems.
- Add your own instructions to test as well, make sure you try all the instructions!

HANDLING HAZARDS - NOP

- One method of handling pipeline hazards is to make the programmer/assembler fix them by adding NOP instructions to the assembled code before it even runs on the datapath. A sufficient number of NOP instructions must be added after each hazard-causing instruction in order to avoid the hazard. This is less efficient than handling hazards in the hardware, but easier to understand/debug.
- First you need to add a NOP instruction to the instruction set & datapath. We will use the instruction code 0000 0000 0001 as the NOP, as it's not used in the initial datapath (currently when op code is 000 for add, the last 3 bits of the instruction are ignored).
    - The basic idea is simple, you want to set all control values that actually "do anything" to 0 before they are used for this instruction.
    - First add control to detect if an instruction is a NOP, to do this you should modify the Control subcircuit in the decode stage.
        - Add an input to the subcircuit for the last bit of the instruction and hook that input up back in main.
        - Add another output to the subcircuit for "is NOP". Add the logic inside the subcircuit to use the op code and the last bit to set this output to 1 if the instruction is a NOP and 0 if not. Do NOT combine this bit with the original 8, it will only be used here in the decode stage (at least for now, if later in the HW you decide you need it in other stages, you can change this).
    - Now use your new control output to choose between using the actual control outputs and all 0's instead. Add a multiplexer before sending the 8-bit "ctl" value through the pipeline register ID/EX. The MUX will have 2 inputs, the actual combined original 8 bits of the control unit output, and an 8-bit constant value 0. The select bit to the MUX will be the newly added "is NOP" output of the control unit.
- Test that your datapath still runs correctly on the test programs, then that it runs correctly when NOPs are added.
- Save this datapath as *nop.circ* and include it in your submission folder.

*** MAKE A COPY of your working datapath WITH NOP implemented and modify that for the rest of the added functionality ***

HANDLING HAZARDS - FORWARDING

- See textbook pages 278-279, 304-310.
- You can use just forwarding to fix the hazard caused by an add or sub instruction where one or both arguments should be the result of a previous instruction that hasn't made it to the register write stage yet.
- Some examples:
    - r0 = r1+3; r0 = r0-5
    - r1 = r2+r2; r3 = r3+1; r0 = r1+r2
    - r2 = r0-3; r1 = r0+3; r3 = r1+r2
- The basic idea is to check in the execute stage if either argument should be obtained from the ALU result in one of the next 2 pipeline stages instead of from the standard reg value determined in the decode stage.
- You will have to deal with the fact that when the datapath first starts running, there are not real instructions in the last 4 stages yet, but there will still be values on those lines. The default values are likely to trigger your forwarding logic, which may cause problems.

HANDLING HAZARDS - BRANCH PREDICTION

- See textbook pages 317-319.
- All branch instructions create a hazard if the following instructions have any effects and the branch is taken. There is no perfect solution to this problem, as you cannot determine if a branch will occur until you've read the instruction itself and the required register values, and without a complete redesign of the pipeline itself, those can't both be done in the same clock cycle. The earliest you can determine if a branch occurs is in the decode stage, and only by adding a LOT of logic, which itself could increase the required length of a clock cycle. Branch prediction, however, is relatively simple and cheap to implement, so you will incorporate that into your datapath.
- You do NOT need to change the branch calculation to occur in the decode stage, keep it in the execute stage.
- you SHOULD predict that the branch will not be taken, and thus
    - if branch IS NOT taken, continue the datapath as normal, so there is no slowdown at all.
    - if branch IS taken, "throw out" the 2 instructions after the branch that have already entered the pipeline by updating any of their control values necessary for the rest of the pipeline stages so that those instructions have no effect (they essentially become NOP's from that point on).
    - There are many ways to handle this logic, most important is to implement a working solution. As a last step, try to make your design as efficient as you can.

LOGISIM TIPS

- Save your datapath OFTEN, at least after each significant update below, if your datapath breaks unexplainably, it is much faster to open an older working version and re-add the components needed than to debug the non-working circuit.
- If in doubt, add a MUX!
- There is a "comparator" component under Arithmetic that takes 2 inputs of any size and gives 3 single-bit outputs for >, =, <.
- Zoom: lower left of window can zoom to see whole circuit or closer view of particular portion.
- Tunnels: allows you to connect components without actually drawing the line between them. Don't get carried away with these as they are often harder to follow than actual lines, but you can use them occasionally if your circuit starts getting too messy with crossed wires. In the start.circ file I have provided I use tunnels for the clock and the regW control line. There is still just 1 clock pin that you can use to cycle the datapath, but a separate tunnel to send the clock input to different datapath components.
- Bit splitters can also be used as combiners, there are examples of this in the start circuit.
- Simulation
    - You can save/load memory states so you don't have to re-enter instructions every time or fill RAM.
    - Make sure simulation is enabled before trying to click the clock.
    - Make sure to reset your simulation (under the simulation tab) before running a new program or your program again, as opposed to just trying to set everything to 0 or you might get strange behavior.
        - You will have to reload your instruction memory after every reset but you get fast at it =)
        - Input pins will also be reset to 0, if you are using any.
    - To put some initial values in registers after a reset use add immediate instructions, e.g. r0=r0+5 will set r0 to 5 if the initial value is 0.
    - To zero out a register with a value, simply subtract it from itself, e.g. r0=r0-r0.
- Subcircuits
    - You will likely need to modify some of the subcircuits I have provided, just double click on the circuit name in the left panel and you will go to the circuit detail that you can change.
    - If your datapath starts getting clogged, add some of your own subcircuits that group chunks of your circuit into black boxes!
    - To add a new subcircuit to your project, go to the Project tab and choose add circuit.
    - To load a subcircuit from another file, choose load library from the Project tab, the Logisim library, and then open the appropriate file.
    - All inputs and outputs must be PINS in order to show up on the subcircuit black box. Remember you can add a pin and then change its # data bits and set

whether it is input/output. This applies to the clock input as well, it should just be a 1-bit pin, not the clock wire.

- You can change the shape and where the inputs/outputs go in the subcircuit black-box by clicking the rightmost icon in the 2nd row of the toolbar in the left panel. If you click on any little blue input/output circle/square, the part of the circuit it corresponds to will show in the lower right corner of the window, so you can tell which is which.
- To add a subcircuit you create to your main window, just click on the subcircuit name in the left panel, then click anywhere in your main circuit window, just like you would click on a MUX and then in main to add it.
- Label your subcircuit well to save yourself a lot of headache, any label you add on the screen where you change the shape of the circuit will show up when you add the circuit to main. Labels on the detailed view of the circuit itself will not show up.