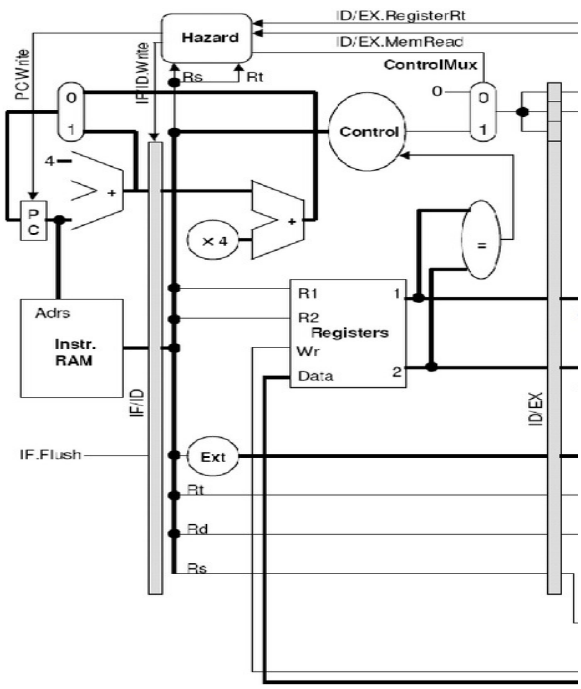
ECE369A Computer Organization
LAB 6

Pipelined Datapath with Hazard Detection

- In this phase you will implement the "hazard detection unit" in the ID stage of the pipeline and execute your test code from Labs 4-5 without the "nop" operations. In your revised datapath it is highly recommended that your branches and unconditional jumps are resolved in the decode stage to have a competitive advantage in the cycle count category.



- The hazard detection unit should detect dependencies and eliminate the need for "nop" instructions by stalling the pipelined execution till the dependency is resolved. The hazard detection unit should check for dependencies between the current instruction in the decode stage and the instructions that are in later stages.
- Figure on the left is just an illustration of the hazard detection concept for "lw" followed by a dependent instruction case. **You will determine the complete list of inputs needed by the hazard detection unit for various dependency scenarios.** For example, the dependency on $t0 register for the following code sequence needs to be detected by the hazard detection unit to stall the pipeline.

```
add   $t0, $t1, $t2
sub   $t3, $t0, $t4
```

- For such dependencies, hazard detection unit needs to stall the pipeline by controlling the "PC", "IF/ID pipeline register" and sending a "nop" instruction to the execution stage through the "MUX".

- Additionally, moving your branch resolution and decision logic into the ID stage is highly recommended as illustrated in the figure above.

With the Hazard Detection unit in place, your datapath now should execute the same code given in Labs4-5 document without the need of "nop" instructions as shown below:

```
PC=0     loop:  add   $t0, $zero, $zero      # t0=0, display 0, 0
PC=4     addi $t1, $zero, 6      # t1= 6, display 4, 6
PC=8     addi $t2, $zero 10      # t2 = 10, display 8, 10
PC=12          sw   $t1, 0($t0)                    # display 12, (no register written)
PC=16          sw   $t2, 4($t0)                    # display 16,
PC=20       lw   $s0, 0($t0)          # s0 = 6, display 20, 6
PC=24          lw   $s1, 4($t0)                    # s1 = 10, display 24, 10
PC=28       sub  $t3, $s1, $s0       # t3 = 10-6 = 4, display 28, 4
PC=32          sll  $t4, $t3, 3          # t4 = 4 << 3 = 32     , display 32, 32
PC=36          srl  $t5, $t4, 2          # t5 = 32 >> 2 = 8, display 36, 8
PC=40          j    loop                 # display 40,
```

  - Create your own test program without "nop" instructions in assembly that includes all the instructions listed in Table 1 of Labs4-5 document.
    - Assume that the first instruction of the test code corresponds to PC value of 0.
    - Translate this program to binary and initialize your instruction memory with this test program.
  - Synthesize and run the program in post-routing simulation
  - Suggested method: We strongly recommend you generate a program with a dependent sequence of operations covering all the operations.

- **Demonstration:**
  - Demo1: Functional verification by displaying the **value written into the register file after executing each instruction and the PC value of that specific instruction in post-routing simulation**.
    - **Do not change the name and port definitions for the instruction memory. During the demonstration day, we will replace your instruction memory with our version that has a precompiled program. It is highly critical that your "sw" is working properly. Our test program will write to the data memory a series of values using the "sw" instruction to initialize the contents.**
    - You need to demo through post routing (implementation) functional simulation of your data path. During demo you will pull out four signals
      1) 32-bit program counter.
      2) 32-bit write_data to register file.

  Make sure the above 32-bit signals are retained (untrimmed) in your post routing simulations.
  - Demo2: After verifying in post-routing simulation, load your bitstream onto the FPGA.
    - Verify by displaying the PC value and the value written into the register file for each instruction
    - If no value is written into the register file, set the display for the register write to zero but still display the PC.
  - **During the demo:**
    - **TA will download the group submission** from d2l. Then TA will copy the instruction and data memory file into a newly created project and will start the synthesis and implementation.
    - While the implementation is on-going, TA will ask questions about your implementation and details of pipelining to the team members and will take notes.
    - Once post-implementation simulation is ready, TA will check the waveform.
    - After checking waveforms TA will explain test cases that are failing if there is any. TAs won't be sharing actual test cases .
  - **Offline Testing:**
    - Functionality of individual instructions on the pipelined datapath will be tested offline based on post-routing simulation. Your datapath should still be able to execute all the instructions from Labs 4-5.
- **Deliverable:**
  - Turn in **.v files along with your constraint file** used in your design (not zipped, no other format will be accepted) using designated dropbox on D2L
  - Include the following notes **under comment section during your submission**
    - % effort
    - Number of pipeline stages:
      - We accept both four and five stage based pipelined datapaths
      - Indicate the number of pipeline stages in your design.
    - Branch decision and resolution stage:
      - Indicate the stage (ID or EX or MEM) where you are making branch decisions .
    - If you don't include any note, we will **assume** that it is a 5-stage pipeline and branches are resolved during ID stage during the offline testing.
- **Penalty Conditions:**
  - Percent effort not reported (20% penalty)
  - Late submission or late demonstration (15% per day)
  - Submitting files in a folder or in compressed form (zip/tar). (10% penalty)
  - Changing the file name or extension. (10% penalty)
  - Failing to demonstrate (80% penalty)
  - Missing testbench (15% penalty)
  - Design works in behavioral simulation but fails to synthesize (80% penalty)

- Design works in behavioral simulation, synthesizes with warnings but post-routing simulation fails (70% penalty)
- Design works in post-routing simulation but fails to run on the FPGA (30% penalty)
- All team members must attend the demonstration (missing partner receives maximum of 50pts)
    - Unable to answer questions about your implementation during demo 30% penalty