Lab Assignment #6

EE-126/CS-146: Computer Engineering w/lab
Professor: Mark Hempstead **TA:** Parnian Mokri, Jessica Nordlund
Tufts University, Fall 2022

Due as per the class calendar, via 'provide'

The ultimate goal of this project is to design a pipelined version of a ARM processor that can detect control and data hazards. The functionality of the processor and its components should match the descriptions in the textbook unless otherwise noted. All work must be your own; copying of code will result in a zero for the project and a report to the administration.

Overview

List of assignments

- Lab 0: Set up modelsim, simulate an AND gate with 2 inputs and 1 output
- Lab 1: Basic Processor Components and Testbenches
- Lab 2: Remaining Processor Components including ALU, Memories, and control logic
- Lab 3: Implementation LEGv8: a single cycle processor that executes a subset of ARM v8-64bit ISA
- Lab 4: Pipelined processor with no hardware hazard detection
- Lab 5: Overcoming data-hazards using forwaring and stalling
- **Lab 6: Overcoming control hazards by resolving conditional/unconditional branches in ID and using flushing**
- Lab 7: Advanced Topics: open-ended team project (groups up to 2 people)

Lab Submission

Please submit your VHDL files *and* a PDF report via 'provide' command on the EE/CS machines. Please follow the announcement on Canvas about Provide to submit your labs and pay attention to messages you get when you try to provide.
VHDL Files: Submit the VHDL source files **(*.vhd)**[1] and any dependencies thereof. Use the entity descriptions provided at the end of this document.[2] These descriptions can also be found in assignment6.zip.
Scripts/Makefile/README: The course staff needs to know how to compile your code and run your testbenches. Please include a README and/or runscripts/Makefile if you have extra files. In addition, do no change the entity definitions or it will not run with our tests
Report: Submit your report as a PDF**(*.pdf)**. Demonstrate the functionality of your code by providing waveforms as detailed in the Deliverables Section. Label/annotate important signals and events in your waveforms and then provide a brief description of what is happening.

Lab6 Objectives

- Implement PipelinedCPU2: a modified version of PipelinedCPU1 that resolves conditional branches (cbz) and unconditional branches (b) in the **ID stage** as described in *Section 4.8* of the textbook, including statically predicting not-taken the flushing of instruction(s) when a mis-prediction occurs required.
- As dicussed in Section 4.8, you will need to move any logic to the ID stage to correctly evaluate a branch. For

[1]Do NOT submit your entire Modelsim project (including but not limited to ***.mpf** and files in **work/**)
[2]Submissions that fail to to follow any of these directions may be penalized at the discretion of the grader. If you have questions, contact the TA (Parnian Mokri: parnian.mokri@tufts.edu).

example, new comparator to the ID stage that can evaluate if a branch is taken or not taken without using the ALU; the PC adder should also be moved to the ID stage.

- The Hazard detection unit (used to stall the pipeline and insert a nop in Lab5) now must be modified to implement a flush. You could also choose to implement this flush logic as a seperate component.
- You are not required to implement forwarding logic to the ID stage comparator logic. However, you can choose to do so to adddress anew data hazard.
- You do *not* need to implement logic to handle the "exceptions" on section 4.9 of the textbook
- Demonstrate functionality by running the test program in this assignment.

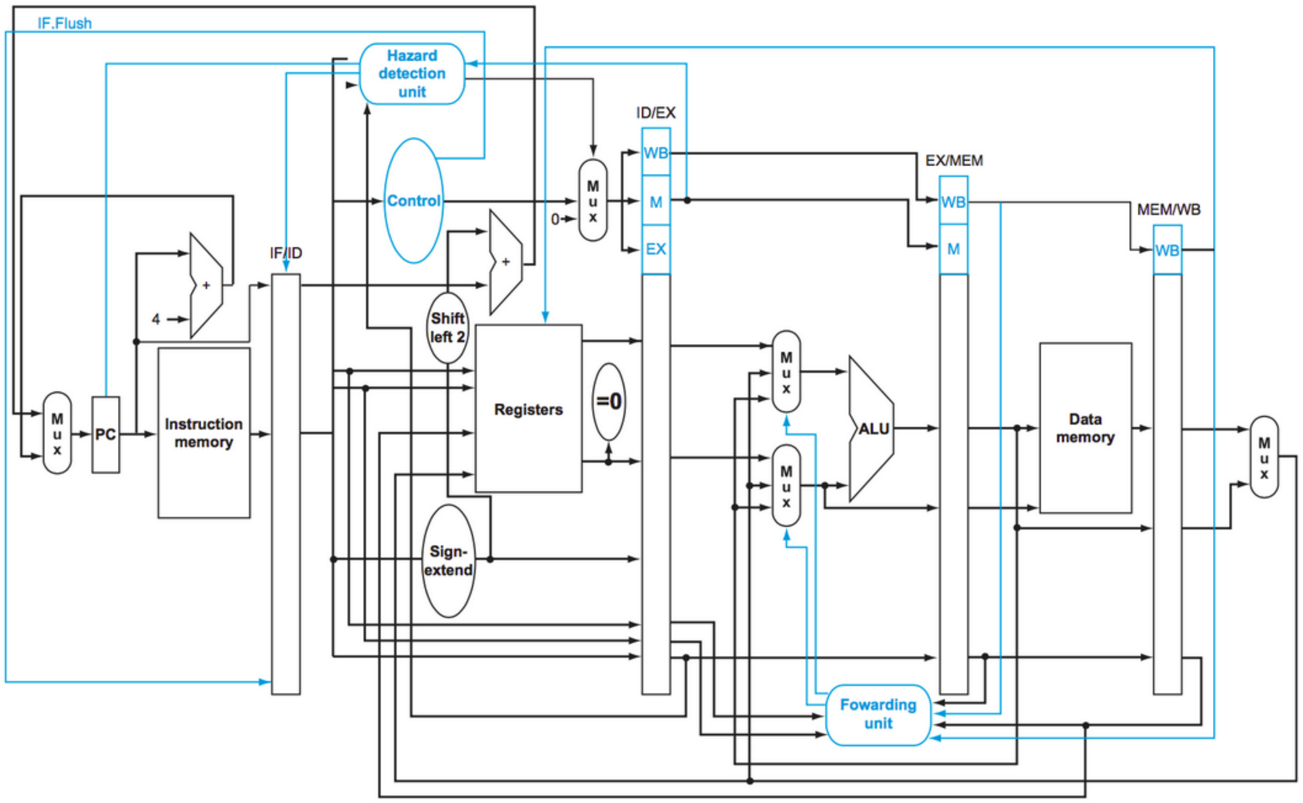


Figure 1: **The final datapath with some of the modifications to resolve control hazards.** The new additions are the IF.Flush signal and the equality test (=) on the outputs of the register file. NOTE: This diagram leaves out details of what additional logic is required and how to connect it. You should implement logic to enable conditional branches(cbz only) and unconditional branches(b only) to complete in the ID stage as described in *Section 4.8* of the textbook. This is Figure (4.63) in the textbook

Deliverables

VHDL Files

PipelinedCPU2 and all of the entities it uses.

Report

Provide waveforms—along with annotations, labels, and descriptions—that show the successful execution of the program defined later in this document. Be sure to:

- Include the most relevant signals in your waveform, including those related to overcoming control hazards
- Point out the following
  - Which instructions are executed, not executed, and which begin execution but are later "flushed"
  - When a conditional/unconditional branches is resolved

  - When/why a branch 'prediction' is correct/incorrect
  - When/why a flush occurs and what its effect is
- Clearly show what pipeline-stage each instruction is in

Program and Simulation Specifications

Test Program (IMEM contents): **partial machine code | raw assembly code**

```
SUB X23, X20, X19    11001011000100110000001010010111
CBZ X23, 5           10110100000000000000000010110111
ADD X9,  X9, X9      10001011000010010000000100101001
SUB X24, X22, X21    11001011000101010000001011011000
CBZ X24, 3           10110100000000000000000001111000
ADD X10, X10, X10    10001011000010100000000101001010
ADD X11, X11, X11    10001011000010110000000101101011
ADD X12, X12, X12    10001011000011000000000110001100
B 2                  00010100000000000000000000000010
ADD X19, X19, X19    10001011000100110000001001110011
ADD X20, X20, X20    10001011000101000000001010010100
nop
nop
nop
nop
```

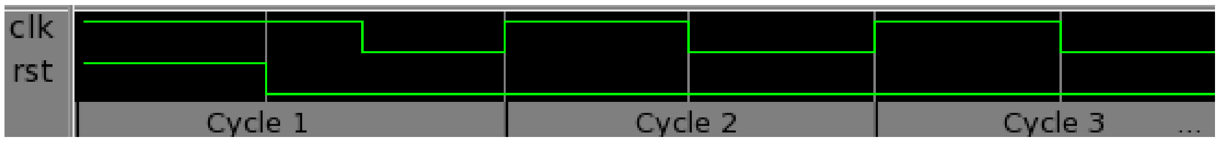
Register Values

```
$X9  = 0h1    $X19 = 0h1
$X10 = 0h2    $X20 = 0h2
$X11 = 0h4
$X21 = 0x000000008BADF00D
$X12 = 0h8
$X22 = 0x000000008BADF00D
$X23 = 0x0000000000000000
$X24 = 0x0000000000000000
```

DMEM Contents

```
DMEM(0x0)  = 0x00 00 00 09
DMEM(0x8)  = 0x00 00 00 08
DMEM(0x10)  = 0x00 00 00 07
DMEM(0x18) = 0x00 00 00 06
```

Reset Sequence



Entity Descriptions **(provided in assignment6.zip)**

Note that there are DEBUG ports again. There are **additional** ports compared to those of PipelinedCPU0 and PipelinedCPU1!

Pipelined CPU2

```
entity PipelinedCPU2 is
port(
     clk :in std_logic;
     rst :in std_logic;
     --Probe ports used for testing
     DEBUG_IF_FLUSH : out std_logic;
     DEBUG_REG_EQUAL : out std_logic;
     -- Forwarding control signals
     DEBUG_FORWARDA : out std_logic_vector(1 downto 0);
     DEBUG_FORWARDB : out std_logic_vector(1 downto 0);
     --The current address (AddressOut from the PC)
     DEBUG_PC : out std_logic_vector(63 downto 0);
     --Value of PC.write_enable
     DEBUG_PC_WRITE_ENABLE : out STD_LOGIC;
     --The current instruction (Instruction output of IMEM)
     DEBUG_INSTRUCTION : out std_logic_vector(31 downto 0);
     --DEBUG ports from other components
     DEBUG_TMP_REGS : out std_logic_vector(64*4 - 1 downto 0);
     DEBUG_SAVED_REGS : out std_logic_vector(64*4 - 1 downto 0);
     DEBUG_MEM_CONTENTS : out std_logic_vector(64*4 - 1 downto 0)
);
end PipelinedCPU2;
```

Tips and Tricks

- By moving the branch decision to the ID stage we have created a new data-hazard that cannot be addressed with the existing forwarding logic from lab5. We don't expect you to make an ID forwarding unit to detect data dependencies from a previous instruction to a branch.
- The program behavior of the branch needs to be the same no matter the pipeline architecture. You do not want to have to recompile the code (and branch offset) depending on the pipeline architecture.
- You needed to pipeline the PC so that the branch instruction has the value of the PC when it was fetched.
- DEBUG_REG_EQUAL is your ZERO FLAG (when registers are equal)
- Besides the branch instructions, the pipeline should have the same behavior (with piplining, forwarding and load-use stalls) as in Lab5 for all other instructions.