Lab Assignment #6

EE-126/COMP-46: Computer Engineering w/lab
Professor: Joel Grodstein **TA:** Dave Werner
Tufts University, Fall 2019

Due as per the class calendar, via 'provide'

The ultimate goal of this project is to design a pipelined version of a MIPS 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.

List of assignments

- Lab 1: Basic Processor Components and Testbenches
- Lab 2: Remaining Processor Components including ALU, Memories, and control logic
- Lab 3: Single Cycle MIPS-32 processor implementation
- 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 branches/jumps 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. The easiest way to do this is via the web interface, which you can find from the course home page.

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.

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 branches (beq) and jumps (j) in the **ID stage** as described in *Section 4.8* of the textbook, including the flushing of instruction(s) when required
- You do *not* need to implement logic to handle the "complicating factors" on page 318-319 of the textbook
- Demonstrate functionality by running the test program in this assignment. As usual, feel free to use the pipeline tracker to aid your debugging.

[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 (Dave Werner: David.Werner@tufts.edu).

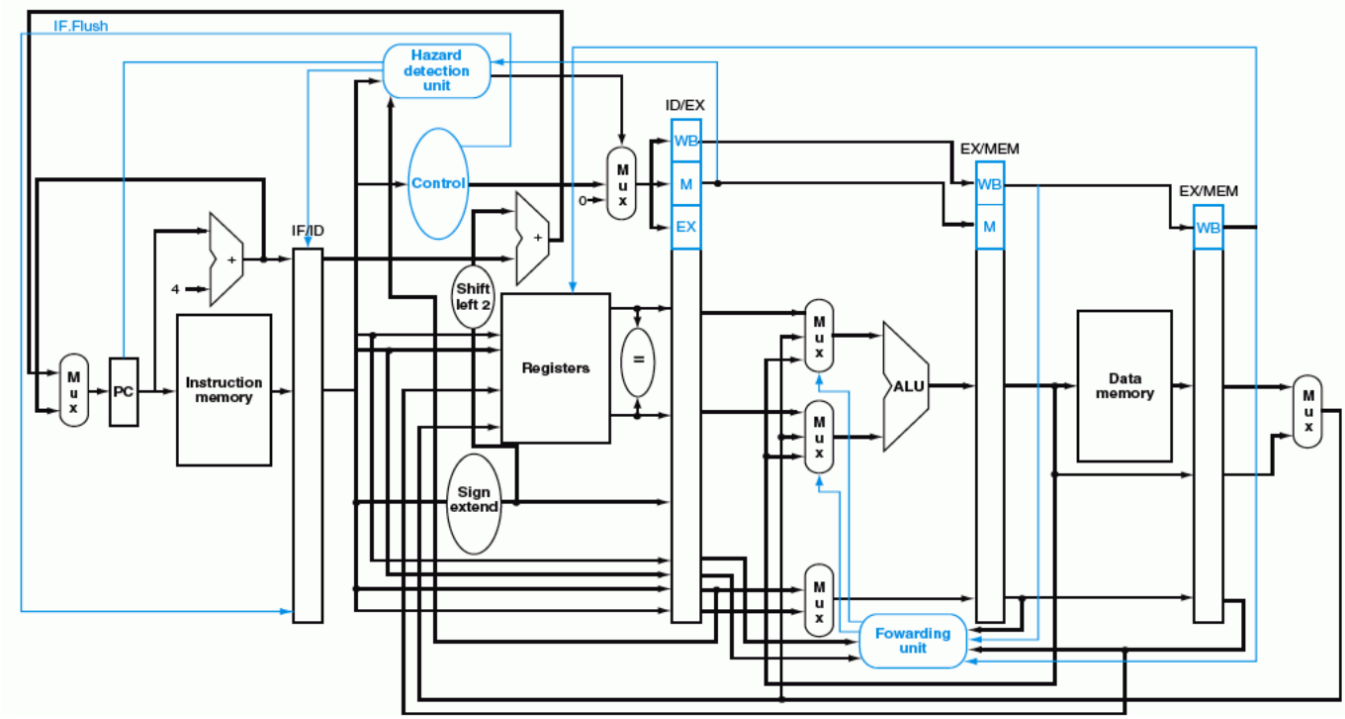

Figure 1: **The final datapath with some of the modifications to resolve control hazards.** The new additions are the IF.Flush signal and the equality test (=) on the outputs of the register file. NOTE: This diagram leaves out details of what additional logic is required and how to connect it. You should implement logic to enable branches(beq only) and jumps(j only) to complete in the ID stage as described in *Section 4.8* of the textbook. This is Figure 4.65 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 branch/jump 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

```
????????????????????????????????  |         beq $s0 $s1 L1  0
00000001000010000100000000100000  |         add $t0 $t0 $t0  4
00010010010100110000000000000010  |         beq $s2 $s3 L2  8
00000001001010010100100000100000  |         add $t1 $t1 $t1  12
00000001010010100101000000100000  |  L1:    add $t2 $t2 $t2  16
00000001011010110101100000100000  |  L2:    add $t3 $t3 $t3  20
????????????????????????????????  |         j exit           24
00000010000100001000000000100000  |         add $s0 $s0 $s0  28
00000010001100011000100000100000  |  exit:  add $s1 $s1 $s1  32
00000000000000000000000000000000  |         nop // add -> ID
00000000000000000000000000000000  |         nop // add -> EX
00000000000000000000000000000000  |         nop // add -> MEM
00000000000000000000000000000000  |         nop // add -> WB
```

Register Values

```
$t0 = 1     $s0 = 1
$t1 = 2     $s1 = 2
$t2 = 4     $s2 = 2343432205
$t3 = 8     $s3 = 2343432205
```

DMEM Contents

```
DMEM(0x0) = 0x00 00 00 09
DMEM(0x4) = 0x00 00 00 08
DMEM(0x8) = 0x00 00 00 07
DMEM(0xC) = 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 or for the tracker.
     DEBUG_IF_SQUASH : 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 (in various pipe stages)
     DEBUG_PC, DEBUG_PCPlus4_ID, DEBUG_PCPlus4_EX, DEBUG_PCPlus4_MEM,
               DEBUG_PCPlus4_WB: out STD_LOGIC_VECTOR(31 downto 0);
     -- instruction is a store.
     DEBUG_MemWrite, DEBUG_MemWrite_EX, DEBUG_MemWrite_MEM: out STD_LOGIC;
     -- instruction writes the regfile.
     DEBUG_RegWrite, DEBUG_RegWrite_EX, DEBUG_RegWrite_MEM, DEBUG_RegWrite_WB: out std_logic;
     -- instruction is a branch or a jump.
     DEBUG_Branch, DEBUG_Jump: out std_logic;

     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(32*4 - 1 downto 0);
     DEBUG_SAVED_REGS : out std_logic_vector(32*4 - 1 downto 0);
     DEBUG_MEM_CONTENTS : out std_logic_vector(32*4 - 1 downto 0)
);
end PipelinedCPU2;
```