Lab Assignment #5

EE-126/COMP-46: Computer Engineering w/lab
Professor: Mark Hempstead **TA:** Parnian Mokri
Tufts University, Fall 2021

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.

Extra resources

Please refer to LabResourcesTutorials for more resources on using Questasim and VHDL tutorials. If you decide to use halligan’s windows machine for your labs, every machine in Halligan Hall **Room 122** has Questasim installed and running. Make sure you save your files in a write-able directory.

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 assignment5.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. In addition, do no change the entitity 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.

Lab5 Objectives

- Implement forwarding circuitry as shown in Figures 1 and 2.
- Implement Hazard Detection Unit as shown in Figure 1 to stall the pipeline and insert a *nop*.

[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).

- Demonstrate functionality by running the test program with data-hazards as defined later in this document.

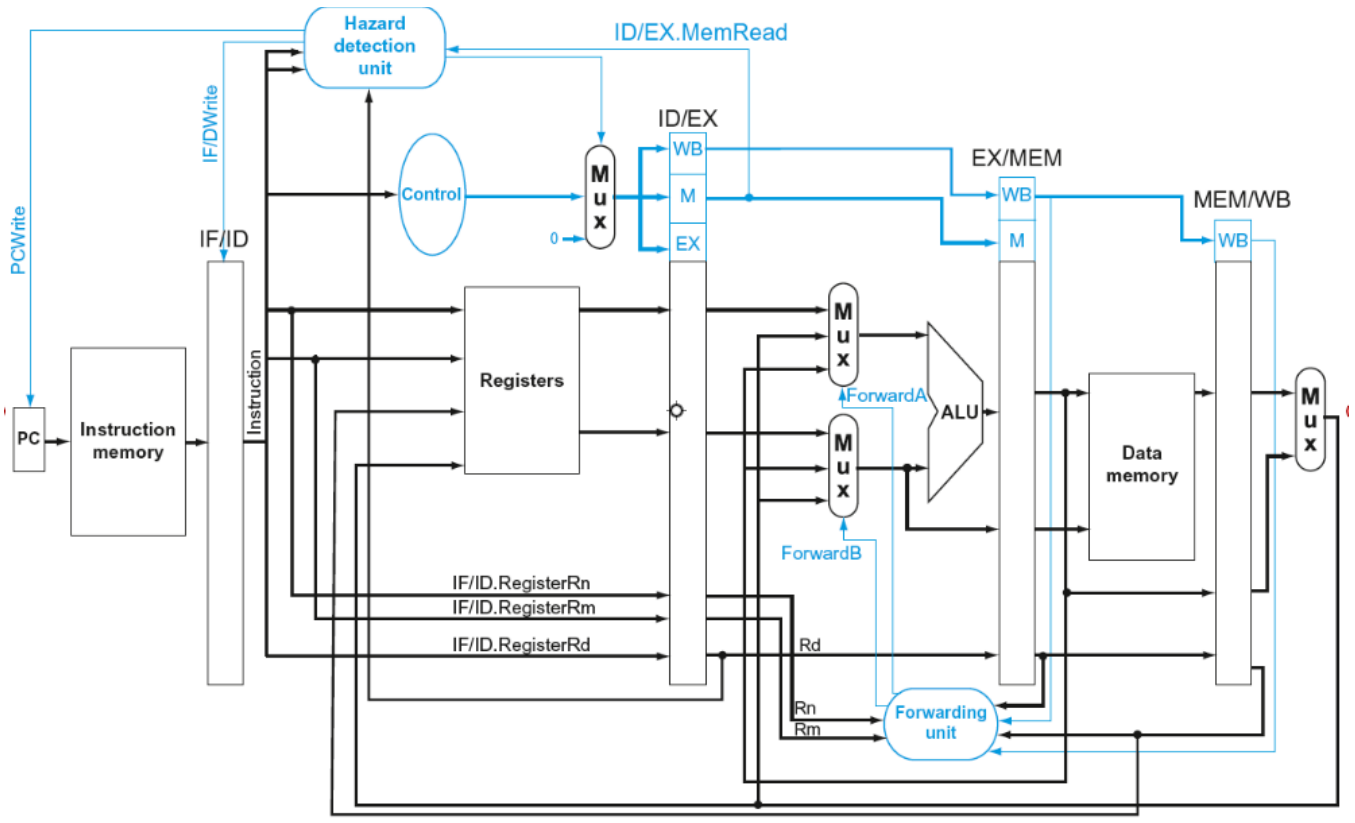


Figure 1: **Pipelined control overview showing the forwarding and hazard detection logic.** Note that this diagram leaves out some of the details of the full datapath. [Figure 4.55 from the textbook]

Deliverables

VHDL Files

PipelinedCPU1 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 data hazards
- Point out key events such as
  - When the pipeline is stalled
  - When values are forwarded
- Clearly show what pipeline-stage each instruction is in

Extra Credit

Consider running the test program with a *nop* inserted after the first instruction (l). Describe how the execution of that program would differ from the test program and explain *why* those differences would occur. In your explanation, include any differences, including those related to:

- stalling the pipeline
- forwarding values
- number of cycles needed to finish the program
- value of PC when the final *sw* occurs

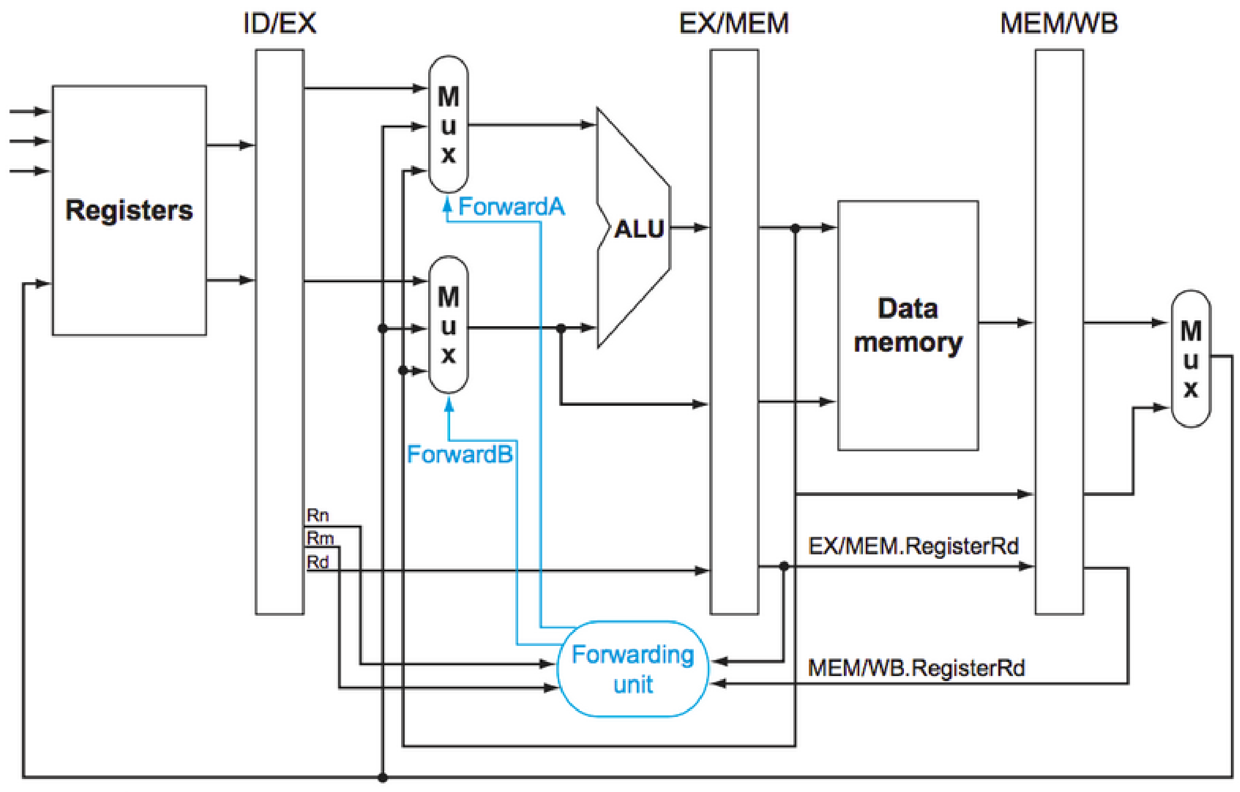


Figure 2: **Close-up of the forwarding circuitry in the datapath of Figure 1.** [Figure 4.53 from the textbook]

- the state of DMEM/Registers at the end of the program

ForwardingUnit

The Forward Unit, and attached muxes, should be able to forward data from the EX/MEM and MEM/WB registers to the input of the ALU. The details of the implementation including sample code and diagrams can be found in pages 320-322 in the textbook as well as the lecture slides from class. **(Note: in this design, data cannot be forwarded directly to the data memory for a store)**

Hazard Detection

The Hazard detection unit should detect load-use hazards and insert a bubble (nop) in the pipeline. For details, follow the logic described in Section 4.7 of the textbook, specifically pages 324 - 327.

Program and Simulation Specifications

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

```
LDUR  X9, [XZR, 0]          11111000010000000000001111101001
ADD   X9, X9, X9            10001011000010010000000100101001
ADD   X10, X9, X9           10001011000010010000000100101010
SUB   X11, X10, X9          11001011000010010000000101001011
STUR  X11, [XZR, 8]         11111000000000001000001111101011
STUR  X11, [XZR, 12]        11111000000000001100001111101011
NOP                         filled with zero - see imem
NOP                         filled with zero - see imem
NOP                         filled with zero - see imem
NOP                         filled with zero - see imem
```

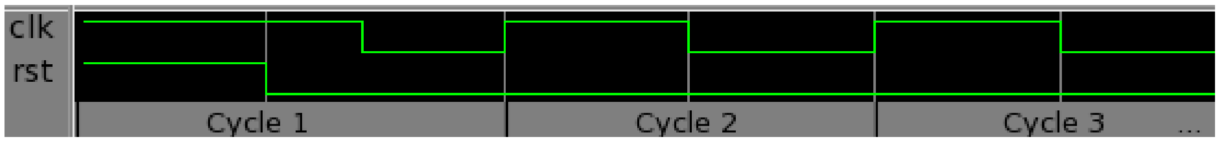
Register Values

```
$x9 =  1     $x19 = 0
$x10 = 0     $x20 = 0
$x11 = 0     $x21 = 0
$x12 = 0     $x22 = 0
```

DMEM Contents

```
DMEM(0)   = 0h1
DMEM(8)   = 0h0
DMEM(A)   = 0h0
DMEM(18)  = 0xh0
```

Reset Sequence



Partial Expected results

Entity Descriptions (provided in assignment5.zip)

Note that there are DEBUG ports again. There are **additional** ports compared to those of PipelineCPU0!
Pipelined CPU 1

```
entity PipelinedCPU1 is
port(
     clk :in std_logic;
     rst :in std_logic;
     --Probe ports used for testing
     -- 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 PipelinedCPU1;
```