Lab Assignment #4

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 assignment4.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.

Lab4 Objectives

- Implement the pipelined ARM (LEGv8) CPU as shown in Figure 1 *without* hazard detection or data-forwarding
- Demonstrate functionality and limitations by running the program defined in this document

[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).

- Please keep in mind that not all instructions the assembler can use are implemented by your processor. Please refer to the greensheet from assignment 3 for a list of instructions you are required to support. To use LegV8 assembler for imem, click here.
- We are looking at the first 4 words. You can set the rest of DMEM to zero.

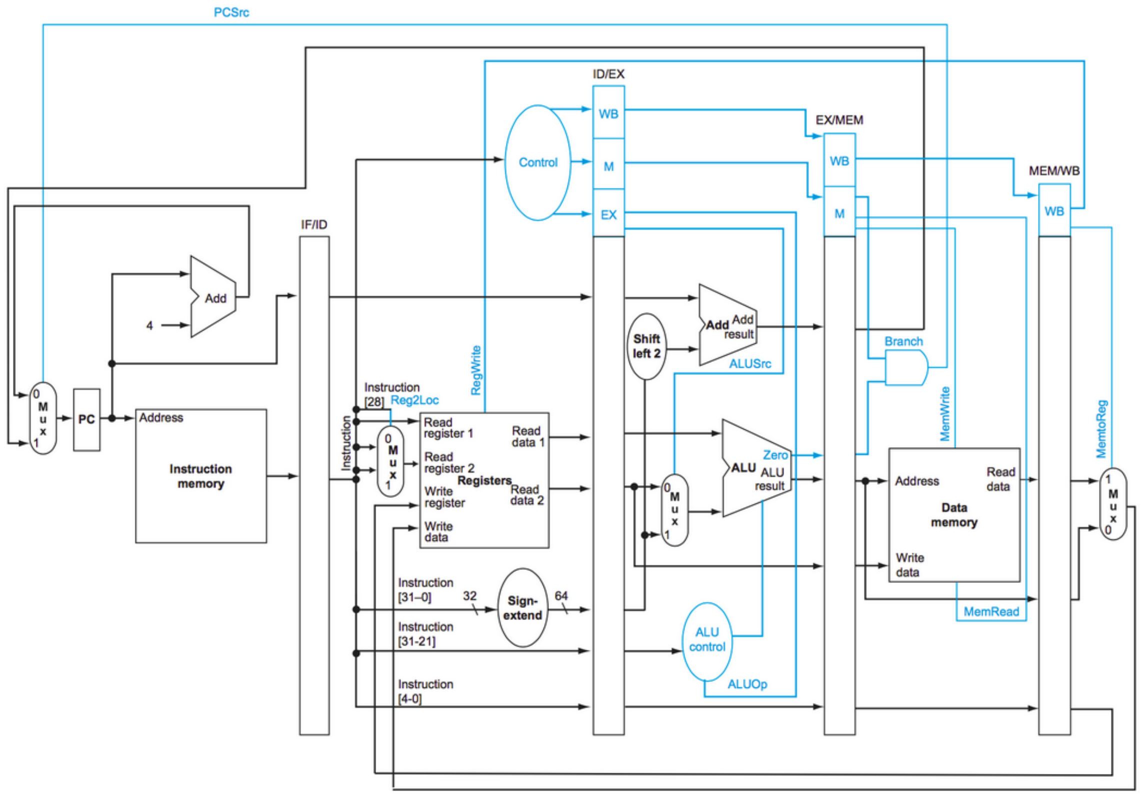


Figure 1: **Schematic of Pipelined ARM (LEGv8) processor for Lab 3.** This is Figure 4.50 in the textbook

Deliverables

VHDL Files

PipelinedCPU0 and all of the entities it uses.

Report

Provide waveforms that show the successful execution of the program defined later in this document. Be sure to:

- Clearly demonstrate what is going on each cycle using waveform-annotations and descriptions
- Include only the most relevant signals, such as the value of the PC, key control signals, any values read/written, etc.
- Show that correct output is stored in registers and data memory
- Point out where critical pipeline events occur, such as writing an incorrect value due to an unhandled hazard, overcoming a hazard with a **NOP**, etc.

Program and Simulation Specifications

Test Program (IMEM contents): partial machine code | raw assembly code where NOP's are defined as all 0s (see imem)

```
ADD X11, X9, X10        10001011000010100000000100101011
STUR X11, XZR,0         11111000000000000000001111101011
SUB X12, X9, X10        11001011000010100000000100101100
STUR  X11, [XZR,0]      11111000000000000000001111101011
STUR  X12, [X12,8]      11111000000000001000000110001100
STUR  X12, [X12,8]      11111000000000001000000110001100
ORR X21, X19, X20       10101010000101000000001001110101
NOP
NOP
STUR X21, [XZR,0]       11111000000000000000001111110101
NOP
NOP
NOP
NOP
```

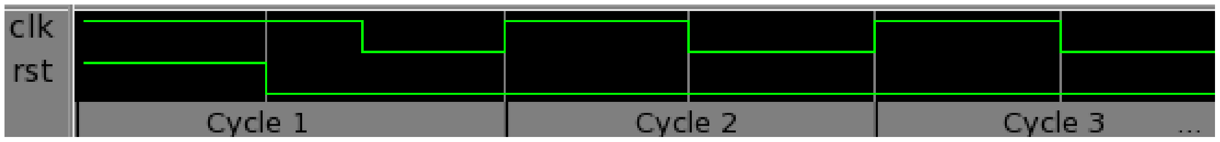
Registers

```
$X9  = 0x0000000000000010
$X10 = 0x0000000000000008
$X11 = 0x0000000000000002
$X12 = 0x0000000000000008
$X19 = 0x00000000CEA4126C
$X20 = 0x000000001009AC83
$X21 = 0x0000000000000000
$X22 = 0x0000000000000000
```

DMEM

```
DMEM(0x0)  = 0h1
DMEM(0x8)  = 0h2
DMEM(0x16) = 0h3
DMEM(0x24) = 0h4
```

Reset Sequence



Partial Expected result: cycle from reset

```
$X9-X12,X19-X22 =             as above before cycle 1
DMEM[0-32]      =             as above before cycle 1
$X11      = 0x0000000000000018    during      cycle 5
DMEM[0x0] = 0h2  at the end of cycle 5

...
```

Entity Descriptions (provided in assignment4.zip)

Note the output signals beginning with "DEBUG". These signals will be used for testing purposes and do not impact the functionality of the components, but they need to be hooked up properly or the tests will fail.

Pipelined CPU 0

```
entity PipelinedCPU0 is
port(
     clk : in STD_LOGIC;
     rst : in STD_LOGIC;
     --Probe ports used for testing
     --The current address (AddressOut from the PC)
     DEBUG_PC : out STD_LOGIC_VECTOR(63 downto 0);
     --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 PipelinedCPU0;
```