Lab Assignment #3

EE-126/COMP-46: 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 assignment3.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.

Lab3 Objectives

- Implement the single-cycle datapath (SingleCycleCPU) in VHDL
  Hint: You might need to use multiple instances of the same entity for different types of operations. **Hint:** To connect the entities together, define signals; for some entities you need to extract parts of the signals
- You do not need to implement the entire LEGv8 instruction set. Your processor should be able to implement

[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).

all instructions listed in our annotated greensheet posted on Canvas. This includes instructions ADD, ADDI, AND, ANDI, CBNZ, CBZ, LDUR, LSL, LSR, ORR, ORI, STUR, SUB, SUBI.

- Verify the functionality of your implementation by simulating the execution of the test program defined later in this document. To this end you must:
  1. Copy the test program machine code we provide into your environment. We have created the machine code by running the test program through a LegV8 assembler so you do not need to compile by hand.
  2. Initialize the contents of the IMEM with the test program and the `address in` should be 64 bit
  3. Set the initial states of DMEM and Registers as specified
  4. Run the simulation to verify correctness

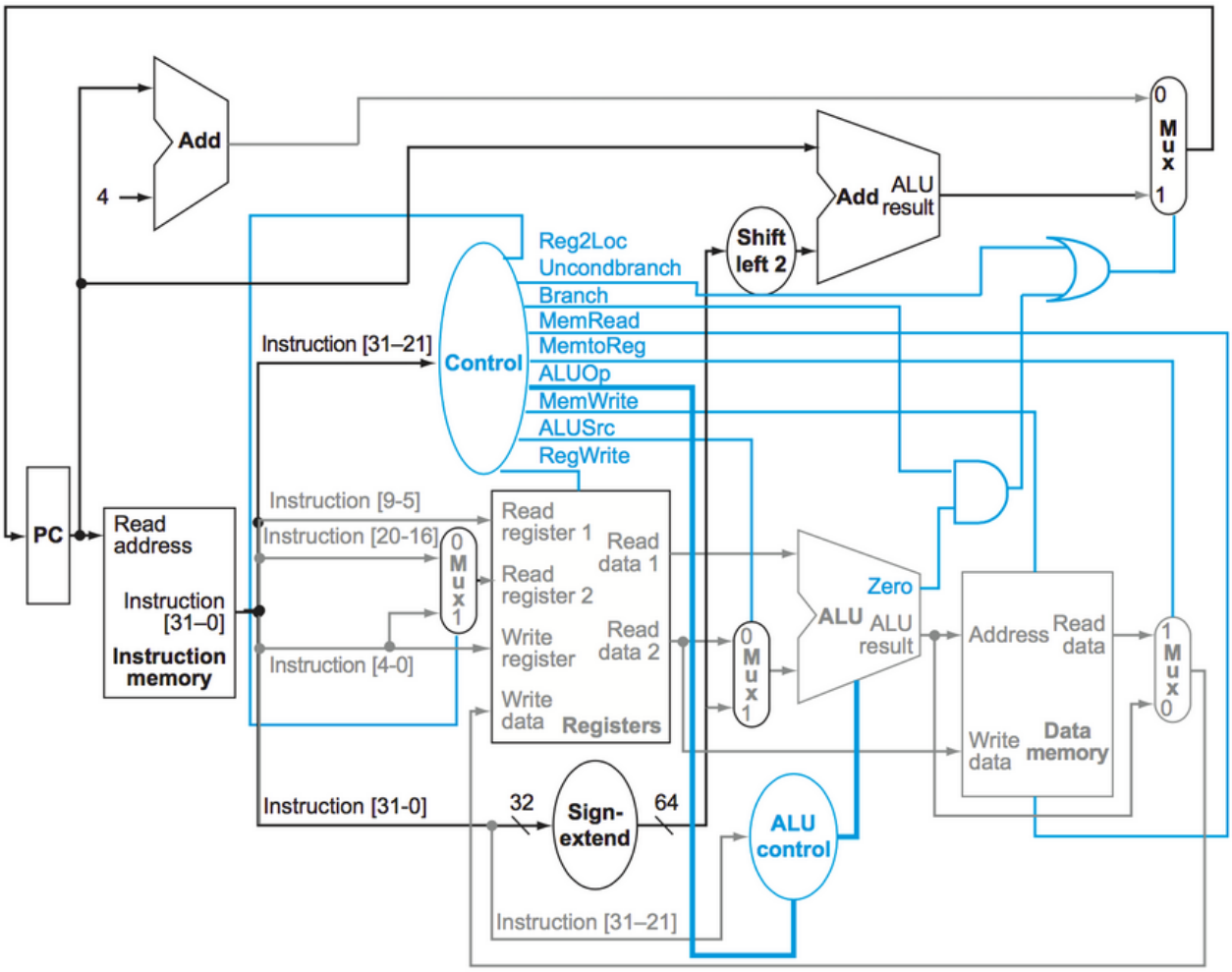


Figure 1: **Single cycle ARM (LEGv8) processor for Lab3.** This is Figure 4-23 in the textbook.

Deliverables

VHDL Files:
SingleCycleCPU and all of the entities it uses. These components are shown in Figure 1.

Report:

- Generate a waveform that shows the successful execution of the test program. This waveform should include the most relevant signals, such as the value of the PC, key control signals, any values read/written, etc. You may wish to split this into multiple figures.
- Annotate your waveform and describe what is going on each cycle

- Answer the question: what value would be written to memory by the test program if X9 = -1?
  - *HINT*: Validate the functionality of your processor then re-run with the new Register file initialization. Be sure to change the values in Register file back before submitting!

Tips and tricks

- Add immediate operations to your ALU as you see them in the assembly codes for each lab
- Reuse your DMEM from lab2 and initialize it for this lab.
- Using IMEMs:
  1. you can have different IMEMs that you initialize with different values for different programs.
  2. You can exclude one IMEM and add the new one from the compile order.
- You might want to have different sign extends for different instruction types
- The ISA.pdf is added to canvas. We only expect you to implement the instructions that are not blacked out in ISA.pdf

Program and Simulation Specifications

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

Computation test

```
ADDI X10, X11, 1
ADDI X10, X11, 2
ADDI X9, X9, 1
SUBI X9, X9, 1
ADD  X10, X9, X11
-------- Assembly Translation ---------
16'h0000:  out = 32'b10010001000000000000010101101010; // ADDI X10, X11, 1
16'h0001:  out = 32'b10010001000000000000100101101010; // ADDI X10, X11, 2
16'h0002:  out = 32'b10010001000000000000010100101001; // ADDI X9, X9, 1
16'h0003:  out = 32'b11010001000000000000010100101001; // SUBI X9, X9, 1
16'h0004:  out = 32'b10001011000010110000000100101010; // ADD  X10, X9, X11
```

Communication test

```
STUR X10, [X11, 0]
LDUR X10, [X9, 0]
----- Assembly Trasnlation ----
16'h0000:  out = 32'b11111000000000000000000101101010; // STUR X10, [X11, 0]
16'h0001:  out = 32'b11111000010000000000000100101010; // LDUR X10, [X9, 0]
```

Final test

```
 ADDI X9, X9, 1
ADD  X10,X9,X11
STUR X10, [X11,0]
LDUR X12, [X11, 0]
CBZ X9, 2
B 3
ADD X9, X10, X11
ADDI  X9, X9, 1
ADD   X21, X10, X9
----- Assembly Translation ----
16'h0000:  out = 32'b10010001000000000000010100101001; // ADDI X9, X9, 1
16'h0001:  out = 32'b10001011000010110000000100101010; // ADD  X10,X9,X11
16'h0002:  out = 32'b11111000000000000000000101101010; // STUR X10, [X11,0]
16'h0003:  out = 32'b11111000010000000000000101101100; // LDUR X12, [X11, 0]
16'h0004:  out = 32'b10110100000000000000000001001001; // CBZ X9, 2
16'h0005:  out = 32'b00010100000000000000000000000011; // B 3
16'h0006:  out = 32'b10001011000010110000000101001001; // ADD X9, X10, X11
16'h0007:  out = 32'b10010001000000000000010100101001; // ADDI  X9, X9, 1
16'h0008:  out = 32'b10001011000010010000000101010101; // ADD   X21, X10, X9
```

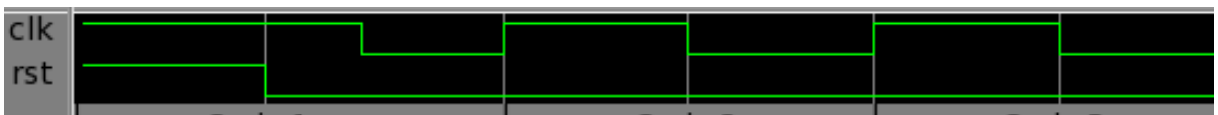
Registers

```
$X9 -> 0000000000000000
$X10 -> 0000000000000001
$X11 -> 0000000000000004
$X12 -> 0000000000000008
$X19 -> 0000000000000015
$X20 -> 0000000000000007
$X21 -> 0000000000000000
$X22 -> 0000000000000016
```

DMEM

```
MEM[0-7] -> 00000011 Hex
MEM[8-15] -> 00000011 Hex
MEM[16-23]-> 00000011 Hex
MEM[24-31]-> 00000011 Hex
```

Reset Sequence



Partial Expected Results

```
$x9,x10,x11m = as above   before        cycle 1
  DMEM[0-15] = as above   before        cycle 1
//  ... Final State After Final Test ... ///
// ... Temporary Registers ...//
After Cycle 1
X9  = 0x1
After Cycle 2
X10 = 0x5
```

Entity Descriptions **(provided in assignment3.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.

Single Cycle CPU

```
entity SingleCycleCPU 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 SingleCycleCPU;
```