|  |  |
| --- | --- |
|  | 16-bit Five Stage MIPS Processor |
|  |  |
|  | Jared Mann V00187636, Taylor Long V00862856  CENG 450: Computer Systems and Architecture  4/19/17 |

# Contents

[I. Contents 1](#_Toc480328971)

[II. Table of Figures 4](#_Toc480328972)

[III. Objective 6](#_Toc480328973)

[Abstract 6](#_Toc480328974)

[Pipelining 6](#_Toc480328975)

[RISC-Like Instruction Set 6](#_Toc480328976)

[Project Objective 6](#_Toc480328977)

[IV. Design Requirement 7](#_Toc480328978)

[General Design Requirements 7](#_Toc480328979)

[Instruction Set 7](#_Toc480328980)

[Registers and Memory 7](#_Toc480328981)

[Pipelining 7](#_Toc480328982)

[Program Execution 8](#_Toc480328983)

[General Implementation Choices 8](#_Toc480328984)

[Data Forwarding and Hazard Detection 8](#_Toc480328985)

[Branch Prediction 8](#_Toc480328986)

[Design Process Guidelines Followed 8](#_Toc480328987)

[V. System 10](#_Toc480328988)

[High Level System Hierarchy Breakdown 10](#_Toc480328989)

[CPU Block Diagram 11](#_Toc480328990)

[VI. Individual Module Design 11](#_Toc480328991)

[Top Level Processor Unit 11](#_Toc480328992)

[CPU 12](#_Toc480328993)

[ALU 12](#_Toc480328994)

[Multiplexers 13](#_Toc480328995)

[Program Counter 13](#_Toc480328996)

[Pipeline Registers 13](#_Toc480328997)

[RAM Module 14](#_Toc480328998)

[ROM Module 14](#_Toc480328999)

[Register File 15](#_Toc480329000)

[Control Unit 15](#_Toc480329001)

[Data Forwarding and Hazard Detection 16](#_Toc480329002)

[Hex to Seven Segment Display 17](#_Toc480329003)

[VII. Simulation Result 18](#_Toc480329004)

[Format A Instructions 18](#_Toc480329005)

[Format B Instructions 19](#_Toc480329006)

[Format L Instructions 19](#_Toc480329007)

[VIII. Analysis and Discussion 20](#_Toc480329008)

[Final Simulation and Performance 20](#_Toc480329009)

[Final Test 1 Results 20](#_Toc480329010)

[Final Test 2 Results 21](#_Toc480329011)

[Hardware Verification Procedures 21](#_Toc480329012)

[IX. Conclusion 22](#_Toc480329013)

[X. Appendices 23](#_Toc480329014)

[VHDL Code 23](#_Toc480329015)

[All VHDL Code available at 23](#_Toc480329016)

[https://github.com/HansSanitizer/CENG-450-Project/tree/master/processor 23](#_Toc480329017)

[Control Unit 24](#_Toc480329018)

[ALU 29](#_Toc480329019)

[Pipeline Registers 30](#_Toc480329020)

[Multiplexers 33](#_Toc480329021)

[RAM Module 35](#_Toc480329022)

[ROM Module 36](#_Toc480329023)

[Register File 37](#_Toc480329024)

[Program Counter 39](#_Toc480329025)

[Hex to 7-Segment 40](#_Toc480329026)

[RTL Schematics 43](#_Toc480329027)

[Processor Top Level Schematic 43](#_Toc480329028)

[Processor Internal Connections 44](#_Toc480329029)

[CPU File Top Level Schematic 45](#_Toc480329030)

[CPU Internal Connections 47](#_Toc480329031)

[CPU Pipeline Stages 48](#_Toc480329032)

[Control Unit Top Level Schematic 50](#_Toc480329033)

# Table of Figures

[Figure 1 - System Hierarchy Breakdown 8](#_Toc480291681)

[Figure 2 - CPU Hierarchy Breakdown 9](#_Toc480291682)

[Figure 3 - CPU File RTL Block Schematic 10](#_Toc480291683)

[Figure 4 - ALU Block Diagram 11](#_Toc480291684)

[Figure 5 - RAM Module Block Diagram 12](#_Toc480291685)

[Figure 6 - ROM Module Block Diagram 13](#_Toc480291686)

[Figure 7 – Register File Module Block Diagram 14](#_Toc480291687)

[Figure 8 - Hex to Seven Segment Display Module Block Diagram 16](#_Toc480291688)

[Figure 9 - Processor Top Level Schematic 19](#_Toc480291689)

[Figure 10 - Processor Internal Connections 21](#_Toc480291690)

[Figure 11 - CPU Top Level Schematic 22](#_Toc480291691)

[Figure 12 - CPU Internal Connections (MUX's Omitted) 23](#_Toc480291692)

[Figure 13 - Pipeline Stages Connection Schematic 25](#_Toc480291693)

[Figure 14 - Control Unit Top Level Schematic 27](#_Toc480291694)

# Objective

## Abstract

### Pipelining

Modern processors implement pipelining to increase throughput. Many current processors implement pipelining with various stage depths having more than 5 stages such as Intel, AMD, and even current ARM processors. Pipelining is a standard across all modern architecture designs.

### RISC-Like Instruction Set

Reduced Instruction Set Computing is common in recent architectures (notably ARM). It is a design principle for developing an architecture using a small set of instructions, and obtaining highly optimized operation.

## Project Objective

Build a 16-bit processor capable of executing instructions from a RISC-like instruction set; using VHDL to describe the processor, and Xilinx to synthesize and implement the design on a Spartan 3E FPGA.

# Design Requirement

## General Design Requirements

### Instruction Set

The project must implement a 16-bit RISC-like instruction set. Following the format outlined on the lab website. The instructions are all 1-word long.

The processor must handle three types of instructions:

* A-Format (arithmetic instructions)
* B-Format (branch instructions)
* L-Format (load and store instructions)

### Registers and Memory

Registers and Memory require certain implementations:

* 8 16-bit general purpose registers, each 1-word long, with word alignment.
* A memory address space that is byte addressable.

### Pipelining

The design must implement pipelining, using a 5 stage MIPS-like pipeline. All pipeline hazards must be handled.

### Program Execution

Regarding these types of instructions, the processor must be able to execute any program written using this instruction set. All RAW and WAR hazards must be handled.

## General Implementation Choices

### Data Forwarding and Hazard Detection

Operand addresses are monitored in each stage so that possible data hazards can be detected during instruction decode. Data forwarding control signals are then assigned where applicable.

### Branch Prediction

Our design implements an assumption of branches being not taken. If the branch is taken, the previous two pipeline registers are flushed and replaced with NOP instructions.

## Design Process Guidelines Followed

During the entire design process some tenants were followed to allow for the smooth transition from a non-pipelined design to a five stage pipeline design.

* Maintain modularity

Using a modular design allowed for the implementation of pipeline registers and a control unit later without needing to go back and change the smaller modules of the design. Many modules were maintained and put into the pipelined design with little to no modification required.

* Use version control software to prevent loss of work, allow for diversification of workflow, and record the project for later use

Using git to track progress, backup our work, and allow for both of us to be working on different aspects of the project was paramount to efficient use of our time during the design process. There was also the bonus of having all the work backed up and recorded for later use.

* Time management and flexibility

Finding time to work on the project required time management and flexibility while working on other courses.

# System

## High Level System Hierarchy Breakdown

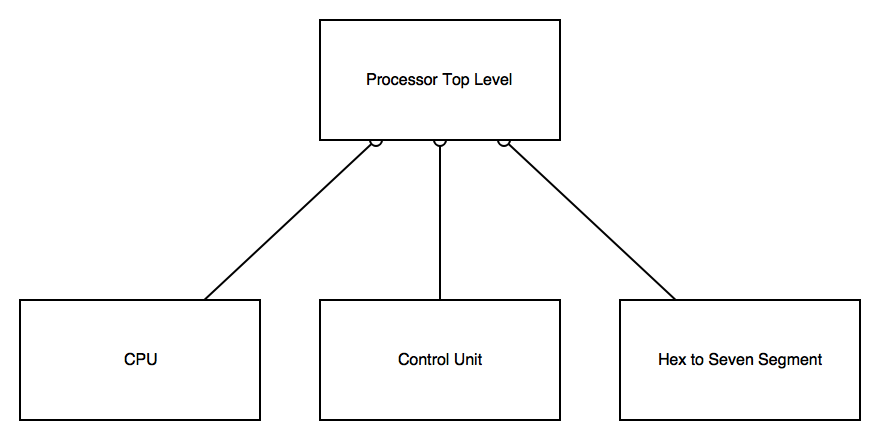


Figure 1 - System Hierarchy Breakdown

The processor is made up of a top-level file connecting the CPU, Control Unit, and Hex to Seven Segment sub-modules together. The CPU module contains its own submodules illustrated in the figure below.

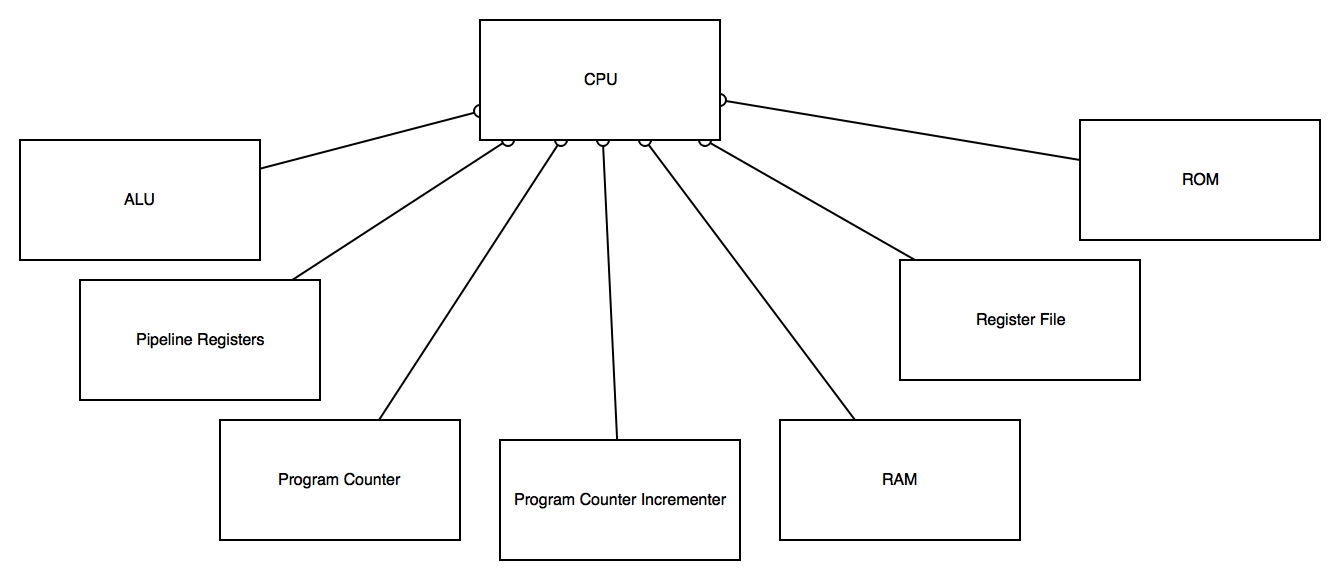


Figure 2 - CPU Hierarchy Breakdown

The CPU contains all the submodules required to perform an instruction execution. The Control Unit managed the CPU’s behavior to avoid data hazards, handle user I/O, and data forwarding.

## CPU Block Diagram

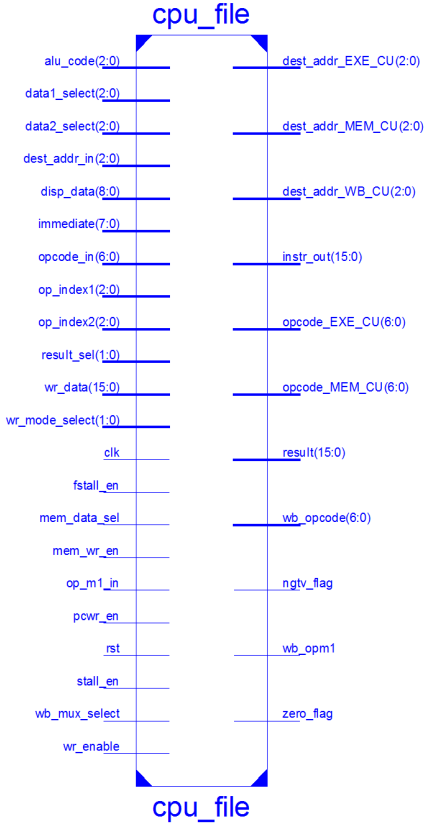


Figure 3 - CPU File RTL Block Schematic

# Individual Module Design

## Top Level Processor Unit

The Top-Level Processor Unit Module handles all connections and routing of the submodules as well as interfacing with the user I/O. The main internal connections connect the Control Unit with the CPU and Hex to Seven Segment Display module. *Figure 9* in the appendix shows a connection diagram of the Processor Modules internals.

## CPU

The CPU contains the data path of the processor. Control signals from the control unit implement the instructions read from the program memory. See *Figure 2* above for a hierarchy diagram of all submodules contained in the CPU and *Figure 11* in the appendix for the CPU’s internal connections and submodule diagram.

### ALU

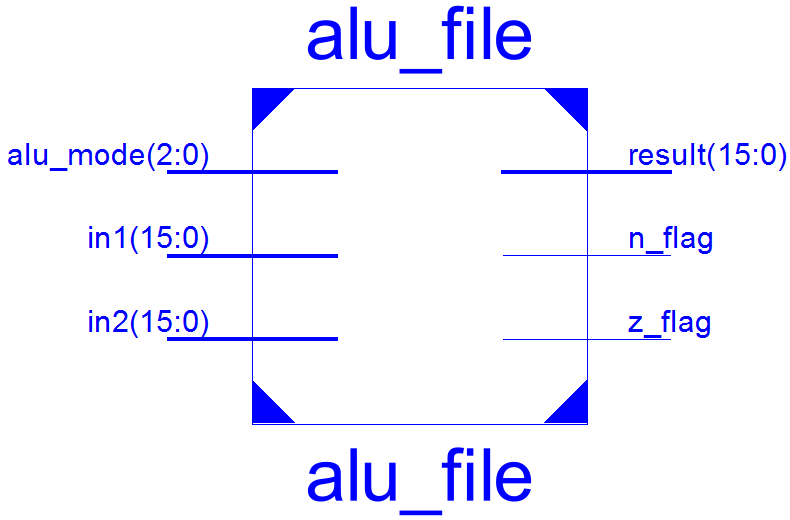


Figure 4 - ALU Block Diagram

The ALU executes arithmetic instructions. In the case of multiplication, the ALU will take the 8 LSB’s of both operands and multiply them together for a 16-bit result. The ALU was built to be purely combinatorial resulting in faster execution.

### Multiplexers

Data select signals from the control unit modify the flow of data and addresses based on the needs of the instruction and are used for the implementation of data forwarding.

### Program Counter

The Program counter continually increments through program memory addresses. The count can be held during a stall and branch instructions are able to overwrite the current address value.

### Pipeline Registers

These registers contain the necessary information that is needed by an instruction for the next stage of the pipeline. Information about the instruction that the control unit needs to monitor is also carried forward into the next stage through these registers. The results of the previous stage are written into the register on a falling clock edge and are released into the next stage on a rising clock edge. Reset signals zero out the entire register which effectively replaces the contents with a NOP instruction.

### RAM Module

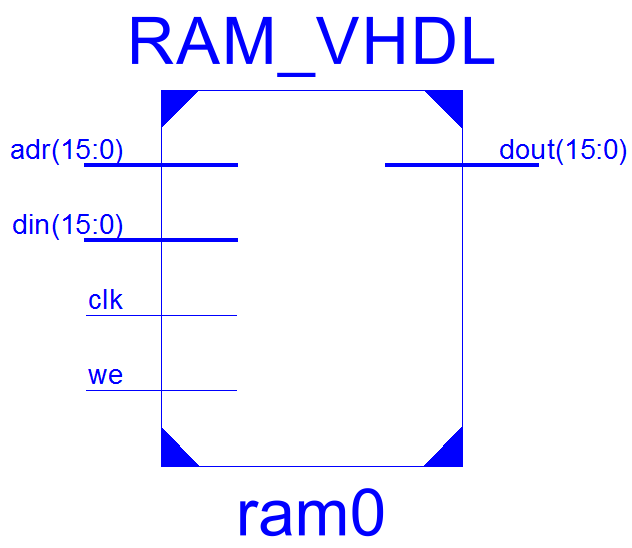


Figure 5 - RAM Module Block Diagram

This is the data memory. Load and store instructions interact with this module. The RAM module is 256 bytes in size and is byte addressable. Each word stored is 16 bits long comprising of two bytes. For example, reading byte zero will result in byte zero and byte one being available on the bus with byte zero being the least significant byte and byte one being the most significant byte. The RAM module is a combinatorial synchronous hybrid. Data reads are performed asynchronously while data writes occur on a rising clock edge to the clk pin while the we pin is set high.

### ROM Module

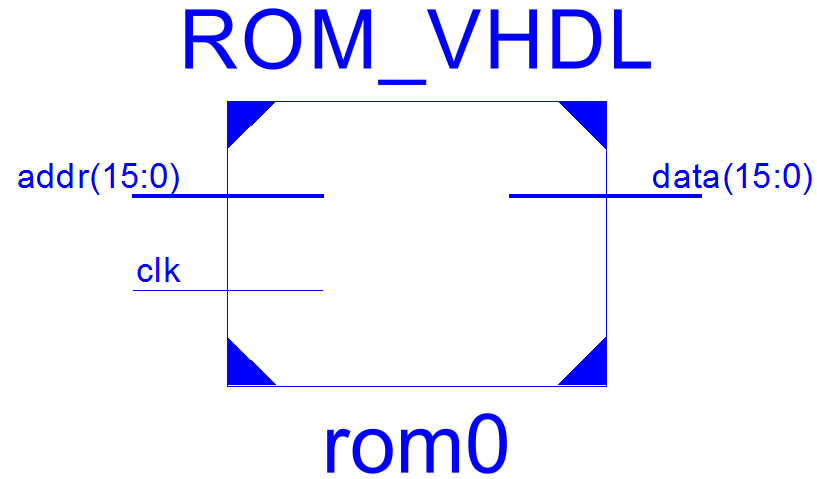


Figure 6 - ROM Module Block Diagram

This is the program memory and contains the instructions that will be executed by the processor. The program counter sets the address to fetch the instruction from. The ROM module contents are byte addressable and word aligned. The address line is read on a rising clock edge and the corresponding instruction is written to the data bus on a falling clock edge.

### Register File

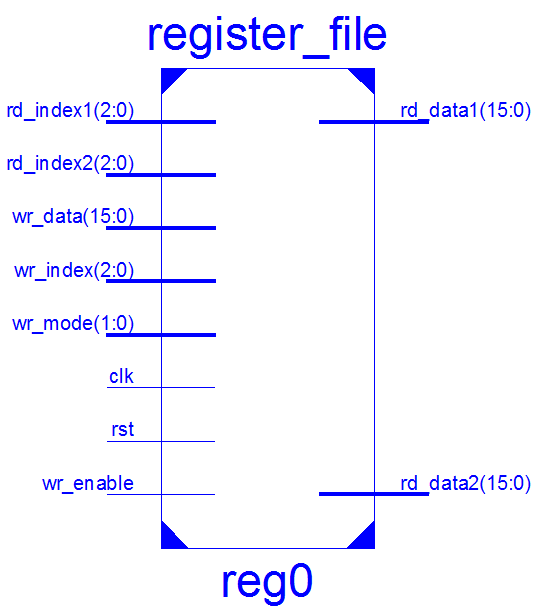


Figure 7 – Register File Module Block Diagram

The Register File contains the 8 general purpose registers used by the processor. The register file is

## Control Unit

The design goal for the control unit was to have it be purely combinatorial. The control unit operates like a combinatorial state machine but uses some synchronous aspects when dealing with user I/O. See *Figure 19* in the appendix for a block diagram of the control unit.

### Data Forwarding and Hazard Detection

Write addresses throughout the pipeline are monitored and compared to the operands of the incoming instruction during decode. A 6-bit code is used to by a process to determine whether a hazard is present and whether to perform data forwarding or a stall.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 4 | 3 | 2 | 1 | 0 |
| RA | | RB | | RC | |

|  |  |
| --- | --- |
| **Code** | **Possible Hazard Location** |
| 00 | No Hazard |
| 01 | Execution |
| 10 | Memory |
| 11 | Write Back |

Based on the current instruction in decode the process checks the code and data forwards from the appropriate stage when possible. For hazards where the data isn’t ready until a later stage stalls are introduced until forwarding can occur. Forwarding is implemented by setting the data select lines or the multiplexers appropriately.

## Hex to Seven Segment Display

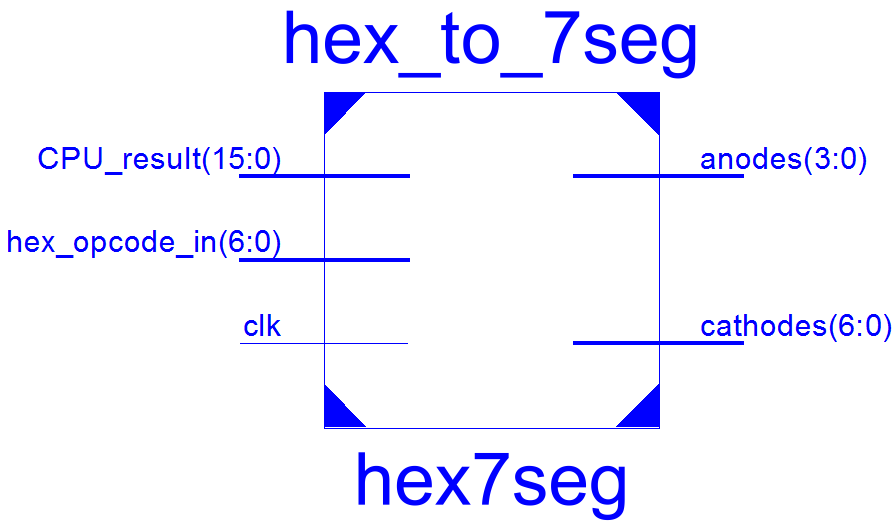


Figure 8 - Hex to Seven Segment Display Module Block Diagram

The Hex to Seven Segment Display module watches the writeback bus for an output opcode and takes the results from the bus and decodes them for use on the seven-segment display.

User inputted values are also taken and displayed during I/O operations. The anode and cathode lines are driven to pins on the Nexys2 board, each clock cycle a different anode is set low to display the hexadecimal value of each 4 bit digit.

# Simulation Result

## Format A Instructions

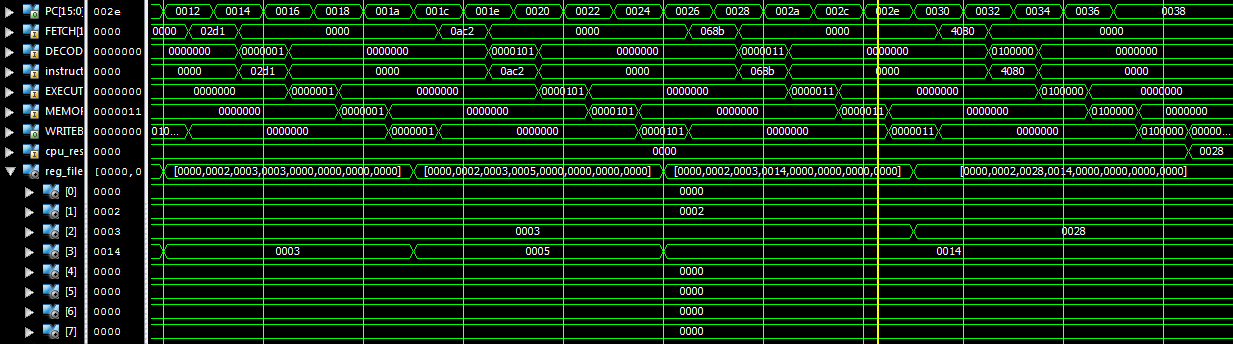


Figure 9 - Format A Simulation Results

Simulation results of the provided format A test instructions (Figure 9) produce the expected results. These are instructions that interact primarily with the ALU as well as the IN and OUT instructions.

## Format B Instructions

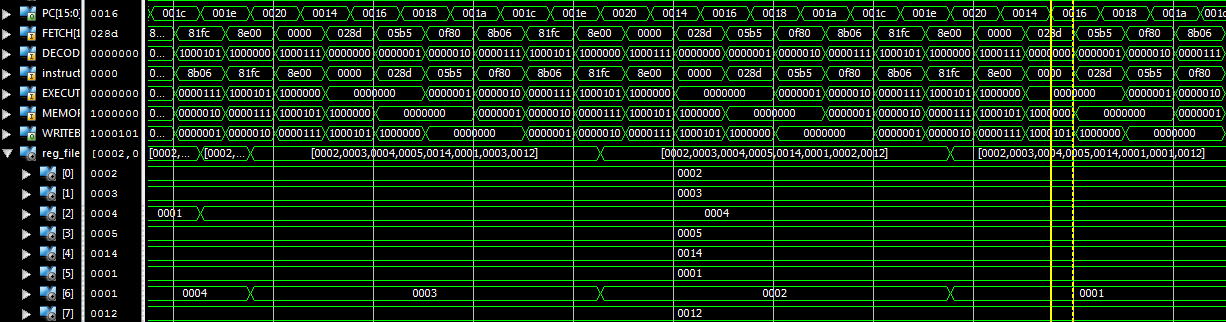


Figure 10 - Format B Simulation Results

Format B instructions were tested with a set of instructions from the lab website. The results are seen above in Figure 10 and show that expected results are obtained. Branches are predicted not taken, therefore, when a branch is taken the pipeline is flushed replacing the erroneous instructions with NOP instructions.

## Format L Instructions

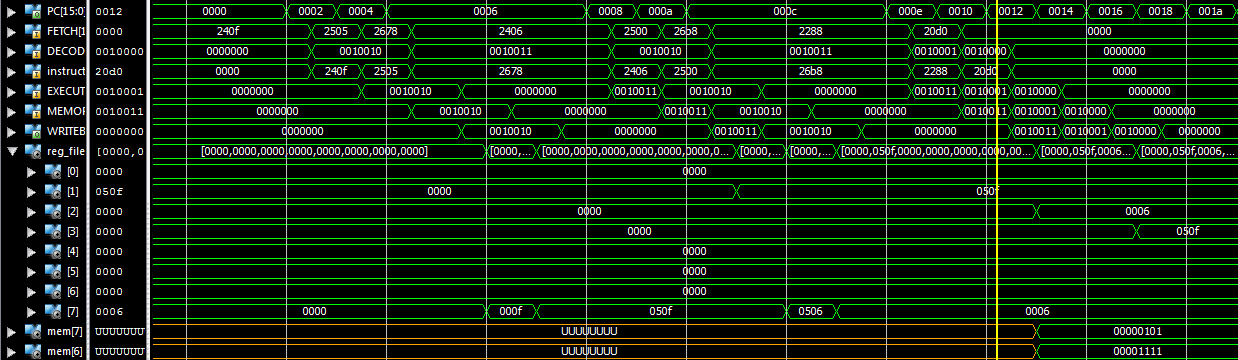


Figure 11 - Format L Simulation Results

The simulation results (Figure 11) of the format L instructions verify the correct operation of the LOAD, STORE, LOADIMM, and MOV instructions. Stalling needs to be implemented in some situations for instructions that have dependencies on these. This is due to the correct data not being available until on or after a pipeline stage. Forwarding can occur from later stages for instructions when applicable.

# Analysis and Discussion

## Final Simulation and Performance

### Final Test 1 Results

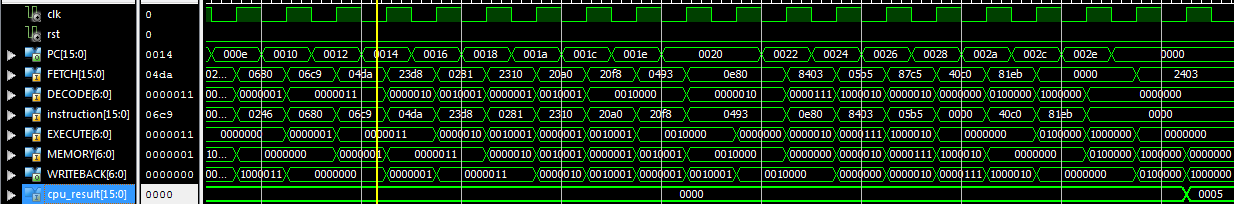


Figure 12 - Final Test 1 Simulation

Based on the simulation results (Figure 12), the performance is analyzed as in the above calculations. From this we have a CPI for the first set of test instructions at 1.157. The main source of bubbles being caused by taken branches, which require the pipeline to be flushed and inserting two NOP instructions.

### Final Test 2 Results



Figure 13 - Final Test 2 Simulation

The simulation results (Figure 13) were used to determine that a CPI of about 1.286 is achieved with the second set of test instructions. Like the first test, the primary source of bubbles is with branches being taken.

## Hardware Verification Procedures

For hardware testing, IN instructions take input from the set of 8 switches and OUT displays the result in hexadecimal on the 7-segment display. Different sets of instructions were tested to confirm the that the expected result is obtained.

To determine maximum operating speed the first set of final test instructions were used. It was found that correct operation occurs with a maximum clock frequency of 50 MHz. At 51 MHz and above, unexpected and erroneous results are observed.

# Conclusion

The result of this laboratory project was the implementation of a functional five stage pipelined processor using a RISC-like instruction set. The design was verified using the Spartan 3E FPGA Nexys2 development board. All test code produced the proper outputs, but the hardware realization of the second set of test code had unexpected behavior when producing an output on the seven-segment display. Aside from this drawback, the design was still realized on hardware performing all the operations required to realize every instruction given.

# Appendices

## VHDL Code

### All VHDL Code available at

### https://github.com/HansSanitizer/CENG-450-Project/tree/master/processor

### Control Unit

-- DECODE

opcode\_out **<=** opcode**;**

ra\_addr **<=**

"111" **when** opcode **=** "1000110" **else** -- BR.SUB

"111" **when** opcode **=** "0010010" **else** -- LOADIMM

operand\_ra**;**

rb\_addr **<=**

operand\_ra **when** opcode **=** "0000100" **else** -- NAND

operand\_ra **when** opcode **=** "0000101" **else** -- SHL

operand\_ra **when** opcode **=** "0000110" **else** -- SHR

operand\_ra **when** opcode **=** "0000111" **else** -- TEST

operand\_ra **when** opcode **=** "1000011" **else** -- BR

operand\_ra **when** opcode **=** "1000100" **else** -- BR.N

operand\_ra **when** opcode **=** "1000101" **else** -- BR.Z

operand\_ra **when** opcode **=** "1000110" **else** -- BR.SUB

operand\_ra **when** opcode **=** "0010001" **else** -- STORE

operand\_ra **when** opcode **=** "0100000" **else** -- OUT

"111" **when** opcode **=** "1000111" **else** -- RETURN

operand\_rb**;**

rc\_addr **<=**

operand\_rb **when** opcode **=** "0000100" **else** -- NAND

operand\_rb **when** opcode **=** "0010001" **else** -- STORE

operand\_rc**;**

op\_m1\_out **<=** operand\_m1**;**

imm\_data **<=**

imm **when** opcode **=** "0010010" **else** -- LOADIMM

**(**"0000" **&** operand\_c1**);**

disp\_data **<=**

**(**"000" **&** disp\_s**)** **when** **((**opcode **=** "1000011"**)** and **(**disp\_s**(**5**)** **=** '0'**))** **else** -- BR

**(**"111" **&** disp\_s**)** **when** **((**opcode **=** "1000011"**)** and **(**disp\_s**(**5**)** **=** '1'**))** **else**

**(**"000" **&** disp\_s**)** **when** **((**opcode **=** "1000100"**)** and **(**disp\_s**(**5**)** **=** '0'**))** **else** -- BR.N

**(**"111" **&** disp\_s**)** **when** **((**opcode **=** "1000100"**)** and **(**disp\_s**(**5**)** **=** '1'**))** **else**

**(**"000" **&** disp\_s**)** **when** **((**opcode **=** "1000101"**)** and **(**disp\_s**(**5**)** **=** '0'**))** **else** -- BR.Z

**(**"111" **&** disp\_s**)** **when** **((**opcode **=** "1000101"**)** and **(**disp\_s**(**5**)** **=** '1'**))** **else**

**(**"000" **&** disp\_s**)** **when** **((**opcode **=** "1000110"**)** and **(**disp\_s**(**5**)** **=** '0'**))** **else** -- BR.SUB

**(**"111" **&** disp\_s**)** **when** **((**opcode **=** "1000110"**)** and **(**disp\_s**(**5**)** **=** '1'**))** **else**

"000000000" **when** opcode **=** "1000111" **else** -- RETURN

disp\_l**;**

alu\_code **<=**

"001" **when** opcode **=** "0000001" **else** -- ADD

"010" **when** opcode **=** "0000010" **else** -- SUB

"011" **when** opcode **=** "0000011" **else** -- MUL

"100" **when** opcode **=** "0000100" **else** -- NAND

"101" **when** opcode **=** "0000101" **else** -- SHL

"110" **when** opcode **=** "0000110" **else** -- SHR

"111" **when** opcode **=** "0000111" **else** -- TEST

"001" **when** opcode **=** "1000000" **else** -- BRR

"001" **when** opcode **=** "1000001" **else** -- BRR.N

"001" **when** opcode **=** "1000010" **else** -- BRR.Z

"001" **when** opcode **=** "1000011" **else** -- BR

"001" **when** opcode **=** "1000100" **else** -- BR.N

"001" **when** opcode **=** "1000101" **else** -- BR.Z

"001" **when** opcode **=** "1000110" **else** -- BR.SUB

"001" **when** opcode **=** "1000111" **else** -- RETURN

"000"**;** -- NOP

-- possible data hazards

dataHazard**(**5 **downto** 4**)** **<=**

"01" **when** **((**operand\_ra **=** dest\_addr\_exe**)** and **(**opcode\_exe **/=** "0000000"**)** and **(**opcode **/=** "1000111"**))** **else**

"01" **when** **((**"111" **=** dest\_addr\_exe**)** and **(**opcode **=** "1000111"**))** **else** -- RETURN

"10" **when** **((**operand\_ra **=** dest\_addr\_mem**)** and **(**opcode\_mem **/=** "0000000"**)** and **(**opcode **/=** "1000111"**))** **else**

"10" **when** **((**"111" **=** dest\_addr\_mem**)** and **(**opcode **=** "1000111"**))** **else** -- RETURN

"11" **when** **((**operand\_ra **=** dest\_addr\_wb**)** and **(**opcode\_wb **/=** "0000000"**)** and **(**opcode **/=** "1000111"**))** **else**

"11" **when** **((**"111" **=** dest\_addr\_wb**)** and **(**opcode **=** "1000111"**))** **else** -- RETURN

"00"**;**

dataHazard**(**3 **downto** 2**)** **<=**

"01" **when** **((**operand\_rb **=** dest\_addr\_exe**)** and **(**opcode\_exe **/=** "0000000"**)** and **(**opcode **/=** "1000111"**))** **else**

"10" **when** **((**operand\_rb **=** dest\_addr\_mem**)** and **(**opcode\_mem **/=** "0000000"**)** and **(**opcode **/=** "1000111"**))** **else**

"11" **when** **((**operand\_rb **=** dest\_addr\_wb**)** and **(**opcode\_wb **/=** "0000000"**)** and **(**opcode **/=** "1000111"**))** **else**

"00"**;**

dataHazard**(**1 **downto** 0**)** **<=**

"01" **when** **((**operand\_rc **=** dest\_addr\_exe**)** and **(**opcode\_exe **/=** "0000000"**)** and **(**opcode **/=** "1000111"**))** **else**

"10" **when** **((**operand\_rc **=** dest\_addr\_mem**)** and **(**opcode\_mem **/=** "0000000"**)** and **(**opcode **/=** "1000111"**))** **else**

"11" **when** **((**operand\_rc **=** dest\_addr\_wb**)** and **(**opcode\_wb **/=** "0000000"**)** and **(**opcode **/=** "1000111"**))** **else**

"00"**;**

**when** "0000001" **=>** -- ADD

-- Check Operands for pending write

**case** dataHazard**(**3 **downto** 2**)** **is**

**when** "01" **=>**

**case** opcode\_exe **is**

**when** "0100001" **=>** -- IN

-- data not available until WB

stall **<=** '1'**;**

data1\_select **<=** "000"**;**

**when** "0010000" **=>** -- LOAD

-- data not avialable until MEM

stall **<=** '1'**;**

data1\_select **<=** "000"**;**

**when** **others** **=>**

-- Forward Op1 from EXE

data1\_select **<=** "101"**;**

led\_fwd\_exe **<=** '1'**;**

**end** **case;**

**when** "10" **=>**

**case** opcode\_mem **is**

**when** "0100001" **=>** -- IN

-- data not available until WB

stall **<=** '1'**;**

data1\_select **<=** "000"**;**

**when** **others** **=>**

-- Forward Op1 from MEM

data1\_select **<=** "110"**;**

led\_fwd\_mem **<=** '1'**;**

**end** **case;**

**when** "11" **=>**

-- Forward Op1 from WB

data1\_select **<=** "111"**;**

led\_fwd\_wb **<=** '1'**;**

**when** **others** **=>**

data1\_select **<=** "000"**;**

**end** **case;**

**case** dataHazard**(**1 **downto** 0**)** **is**

**when** "01" **=>**

**case** opcode\_exe **is**

**when** "0100001" **=>** -- IN

-- data not available until WB

stall **<=** '1'**;**

data2\_select **<=** "000"**;**

**when** "0010000" **=>** -- LOAD

stall **<=** '1'**;**

data2\_select **<=** "000"**;**

**when** **others** **=>**

-- Forward Op2 from EXE

data2\_select **<=** "101"**;**

led\_fwd\_exe **<=** '1'**;**

**end** **case;**

**when** "10" **=>**

**case** opcode\_mem **is**

**when** "0100001" **=>** -- IN

-- data not available until WB

stall **<=** '1'**;**

data2\_select **<=** "000"**;**

**when** **others** **=>**

-- Forward Op2 from MEM

data2\_select **<=** "110"**;**

led\_fwd\_mem **<=** '1'**;**

**end** **case;**

**when** "11" **=>**

-- Forward Op2 from WB

data2\_select **<=** "111"**;**

led\_fwd\_wb **<=** '1'**;**

**when** **others** **=>**

data2\_select **<=** "000"**;**

**end** **case;**

-- EXECUTE

pc\_write\_en **<=**

'1' **when** opcode\_exe **=** "1000000" **else** -- BRR

'1' **when** **((**opcode\_exe **=** "1000001"**)** and **(**n\_flag **=** '1'**))** **else** -- BRR.N

'1' **when** **((**opcode\_exe **=** "1000010"**)** and **(**z\_flag **=** '1'**))** **else** -- BRR.Z

'1' **when** opcode\_exe **=** "1000011" **else** -- BR

'1' **when** **((**opcode\_exe **=** "1000100"**)** and **(**n\_flag **=** '1'**))** **else** -- BR.N

'1' **when** **((**opcode\_exe **=** "1000101"**)** and **(**z\_flag **=** '1'**))** **else** -- BR.Z

'1' **when** opcode\_exe **=** "1000110" **else** -- BR.SUB

'1' **when** opcode\_exe **=** "1000111" **else** -- RETURN

'0'**;**

result\_select **<=**

"01" **when** opcode\_exe **=** "1000110" **else** -- BR.SUB

"10" **when** opcode\_exe **=** "0010000" **else** -- LOAD

"10" **when** opcode\_exe **=** "0010001" **else** -- STORE

"10" **when** opcode\_exe **=** "0010010" **else** -- LOADIMM

"10" **when** opcode\_exe **=** "0010011" **else** -- MOV

"10" **when** opcode\_exe **=** "0100000" **else** -- OUT

"00"**;**

-- MEMORY

mem\_write\_en **<=** '1' **when** opcode\_mem **=** "0010001" **else** '0'**;** -- STORE

mem\_data\_select **<=** '1' **when** opcode\_mem **=** "0010000" **else** '0'**;** -- LOAD

-- WRITE BACK

reg\_wen **<=**

'0' **when** opcode\_wb **=** "0000000" **else** -- NOP

'0' **when** opcode\_wb **=** "0000111" **else** -- TEST

'0' **when** opcode\_wb **=** "0100000" **else** -- OUT

'0' **when** opcode\_wb **=** "1000000" **else** -- BRR

'0' **when** opcode\_wb **=** "1000001" **else** -- BRR.N

'0' **when** opcode\_wb **=** "1000010" **else** -- BRR.Z

'0' **when** opcode\_wb **=** "1000011" **else** -- BR

'0' **when** opcode\_wb **=** "1000100" **else** -- BR.N

'0' **when** opcode\_wb **=** "1000101" **else** -- BR.Z

'0' **when** opcode\_wb **=** "1000111" **else** -- RETURN

'0' **when** opcode\_wb **=** "0010001" **else** -- STORE

'1'**;**

wr\_mode\_sel **<=**

"01" **when** **((**opcode\_wb **=** "0010010"**)** and **(**op\_m1\_wb **=** '0'**))** **else** -- LOADIMM lower

"10" **when** **((**opcode\_wb **=** "0010010"**)** and **(**op\_m1\_wb **=** '1'**))** **else** -- LOADIMM upper

"00"**;**

wb\_mux\_sel **<=** '1' **when** opcode\_wb **=** "0100001" **else** '0'**;** -- IN

out\_reg\_en **<=** '1' **when** opcode\_wb **=** "0100000" **else** '0'**;** -- OUT

### ALU

result **<=**

STD\_LOGIC\_VECTOR**(**signed**(**in1**)** **+** signed**(**in2**))** **when(**alu\_mode **=** "001"**)** **else**

STD\_LOGIC\_VECTOR**(**signed**(**in1**)** **-** signed**(**in2**))** **when(**alu\_mode **=** "010"**)** **else**

STD\_LOGIC\_VECTOR**(**signed**(**in1**(**7 **downto** 0**))\***signed**(**in2**(**7 **downto** 0**)))** **when(**alu\_mode **=** "011"**)** **else**

STD\_LOGIC\_VECTOR**(**unsigned**(**in1**)** nand unsigned**(**in2**))** **when(**alu\_mode **=** "100"**)** **else**

STD\_LOGIC\_VECTOR**(**unsigned**(**in1**)** sll **to\_integer(**unsigned**(**in2**)))** **when(**alu\_mode**=**"101"**)** **else**

STD\_LOGIC\_VECTOR**(**unsigned**(**in1**)** srl **to\_integer(**unsigned**(**in2**)))** **when(**alu\_mode**=**"110"**)** **else**

STD\_LOGIC\_VECTOR**(**unsigned**(**in1**));**

z\_flag **<=**

'1' **when((**alu\_mode **=** "111"**)** and **(to\_integer(**unsigned**(**in1**))** **=** 0**))** **else**

'0' **when((**alu\_mode **=** "111"**)** and **(to\_integer(**unsigned**(**in1**))** **/=** 0**))** **else**

'0'**;**

n\_flag **<=**

'1' **when((**alu\_mode **=** "111"**)** and **(to\_integer(**signed**(**in1**))** **<** 0**))** **else**

'0' **when((**alu\_mode **=** "111"**)** and **(to\_integer(**signed**(**in1**))** **>=**0**))** **else**

'0'**;**

### Pipeline Registers

**process(**clk**)**

**begin**

**if(falling\_edge(**clk**))** **then**

-- Falling edge action latch new instruction

**if** **((**rst **=** '1'**)** or **(**mrst **=** '1'**))** **then**

-- Clear register with reset signal

instructionReg **<=** **(others** **=>** '0'**);**

**elsif(**rst **=** '0'**)** **then**

instruction **<=** instr\_in**;**

programCounter **<=** pc\_in**;**

**end** **if;**

**end** **if;**

**end** **process;**

**process(**clk**)**

**begin**

**if(rising\_edge(**clk**))** **then**

-- Send instruction out to be decoded

**if(**rst **=** '1'**)** **then**

instr\_out **<=** X"0000"**;**

pc\_out **<=** X"0000"**;**

**end** **if;**

**if** hold **=** '0' **then**

instr\_out **<=** instruction**;**

pc\_out **<=** programCounter**;**

**end** **if;**

**end** **if;**

**end** **process;**

**process(**clk**)**

**begin**

**if(**clk**=**'0' and clk'**event)** **then**

-- Falling edge action latch new data from previous stage

**if** **((**rst **=** '1'**)** or **(**flush **=** '1'**)** or **(**mrst **=** '1'**))** **then**

-- Clear register with reset signal

pipeRegister **<=** **(others** **=>** '0'**);**

**else**

nextPC **<=** next\_pc\_in**;**

opCode **<=** opcode\_in**;**

aluCode **<=** alu\_in**;**

destAddress **<=** dest\_addr\_in**;**

operandM1 **<=** op\_m1\_in**;**

operandData1 **<=** op1\_data\_in**;**

operandData2 **<=** op2\_data\_in**;**

**end** **if;**

**end** **if;**

**end** **process;**

**process(**clk**)**

**begin**

**if(**clk**=**'1' and clk'**event)** **then**

-- Send data out to next stage

next\_pc\_out **<=** nextPC**;**

opCode\_out **<=** opCode**;**

alu\_out **<=** aluCode**;**

dest\_addr\_out **<=** destAddress**;**

op\_m1\_out **<=** operandM1**;**

op1\_data\_out **<=** operandData1**;**

op2\_data\_out **<=** operandData2**;**

**end** **if;**

**end** **process;**

**process(**clk**)**

**begin**

**if(**clk**=**'0' and clk'**event)** **then**

-- Falling edge action latch new data from previous stage

**if(**rst **=** '1'**)** **then**

-- Clear register with reset signal

pipeRegister **<=** **(others** **=>** '0'**);**

**else**

operand2Data **<=** op2\_data\_in**;**

opcode **<=** opcode\_in**;**

destAddress **<=** dest\_addr\_in**;**

operandM1 **<=** op\_m1\_in**;**

aluResult **<=** result\_in**;**

zeroFlag **<=** z\_flag\_in**;**

negativeFlag **<=** n\_flag\_in**;**

**end** **if;**

**end** **if;**

**end** **process;**

**process(**clk**)**

**begin**

**if(**clk**=**'1' and clk'**event)** **then**

-- Send data out to next stage

op2\_data\_out **<=** operand2Data**;**

opcode\_out **<=** opcode**;**

dest\_addr\_out **<=** destAddress**;**

op\_M1\_out **<=** operandM1**;**

result\_out **<=** aluResult**;**

z\_flag\_out **<=** zeroFlag**;**

n\_flag\_out **<=** negativeFlag**;**

**end** **if;**

**end** **process;**

**process(**clk**)**

**begin**

**if(**clk**=**'0' and clk'**event)** **then**

-- Falling edge action latch new data from previous stage

**if(**rst **=** '1'**)** **then**

-- Clear register with reset signal

pipeRegister **<=** **(others** **=>** '0'**);**

**else**

opCode **<=** opcode\_in**;**

destAddress **<=** dest\_addr\_in**;**

operandM1 **<=** op\_m1\_in**;**

aluResult **<=** result\_in**;**

**end** **if;**

**end** **if;**

**end** **process;**

**process(**clk**)**

**begin**

**if(**clk**=**'1' and clk'**event)** **then**

-- Send data out to next stage

opcode\_out **<=** opcode**;**

dest\_addr\_out **<=** destAddress**;**

op\_m1\_out **<=** operandM1**;**

result\_out **<=** aluResult**;**

**end** **if;**

**end** **process;**

### Multiplexers

**architecture** Behavioral **of** op1\_data\_mux **is**

**begin**

data **<=**

pc\_value **when** data\_select **=** "001" **else**

**(**X"00" **&** immediate**)** **when** data\_select **=** "010" **else**

exe\_data **when** data\_select **=** "101" **else**

mem\_data **when** data\_select **=** "110" **else**

wb\_data **when** data\_select **=** "111" **else**

reg\_data**;**

**end** Behavioral**;**

**architecture** Behavioral **of** op2\_data\_mux **is**

--signal bigImmediate : STD\_LOGIC\_VECTOR(15 downto 0) := (others=>'0');

--attribute S: string;

--attribute S of bigImmediate: signal is "Yes";

**begin**

--bigImmediate(7 downto 0) <= immediate;

data **<=**

X"00" **&** immediate **when** data\_select **=** "001" **else**

"000000" **&** displacement **&** "0" **when** **((**data\_select **=** "010"**)** and **(**displacement**(**8**)** **=** '0'**))** **else** -- 2\*disp

"111111" **&** displacement **&** "0" **when** **((**data\_select **=** "010"**)** and **(**displacement**(**8**)** **=** '1'**))** **else** -- 2\*disp (negative)

exe\_data **when** data\_select **=** "101" **else**

mem\_data **when** data\_select **=** "110" **else**

wb\_data **when** data\_select **=** "111" **else**

reg\_data**;**

**end** Behavioral**;**

**architecture** Behavioral **of** mem\_data\_mux **is**

**begin**

data **<=** mem\_data **when** data\_select **=** '1' **else** result\_data**;**

**end** Behavioral**;**

**architecture** Behavioral **of** reg\_wrdata\_mux **is**

**begin**

data **<=** wb\_data **when** ext\_select **=** '0' **else** ext\_data**;**

**end** Behavioral**;**

**architecture** Behavioral **of** result\_data\_mux **is**

**begin**

data **<=**

pc\_value **when** data\_select **=** "01" **else**

op1\_data **when** data\_select **=** "10" **else**

alu\_data**;**

**end** Behavioral**;**

**architecture** Behavioral **of** wraddr\_mux **is**

**begin**

data **<=** "111" **when** data\_select **=** '1' **else** dest\_addr**;**

**end** Behavioral**;**

### RAM Module

**entity** RAM\_VHDL **is**

**generic(**N **:** integer **:=** 8**;** M **:** integer **:=** 8**);**

**port(**

clk**,** we **:** **in** STD\_LOGIC**;**

adr **:** **in** STD\_LOGIC\_VECTOR**(**15 **downto** 0**);**

din **:** **in** STD\_LOGIC\_VECTOR**(**15 **downto** 0**);**

dout **:** **out** STD\_LOGIC\_VECTOR**(**15 **downto** 0**));**

**end;**

**architecture** synth **of** RAM\_VHDL **is**

**type** mem\_array **is** **array** **((**2 **\*\*** N **-** 1**)** **downto** 0**)** **of** STD\_LOGIC\_VECTOR**(**M **-** 1 **downto** 0**);**

**signal** mem **:** mem\_array**;**

**begin**

**process(**clk**)**

**begin**

**if** **falling\_edge(**clk**)** **then**

**if** **(**we **=** '1'**)** **then**

mem**(to\_integer(**unsigned**(**adr**(**7 **downto** 0**))))** **<=** din**(**7 **downto** 0**);**

mem**(to\_integer(**unsigned**(**adr**(**7 **downto** 0**)))** **+** 1**)** **<=** din**(**15 **downto** 8**);**

**end** **if;**

**end** **if;**

**end** **process;**

-- Read Operations

dout**(**7 **downto** 0**)** **<=** mem**(to\_integer(**unsigned**(**adr**(**7 **downto** 0**))));**

dout**(**15 **downto** 8**)** **<=** mem**(to\_integer(**unsigned**(**adr**(**7 **downto** 0**))** **+** 1**));**

**end;**

### ROM Module

**entity** ROM\_VHDL **is**

**port(**

clk **:** **in** std\_logic**;**

addr **:** **in** std\_logic\_vector **(**15 **downto** 0**);**

data **:** **out** std\_logic\_vector **(**15 **downto** 0**)**

**);**

**end** ROM\_VHDL**;**

**architecture** BHV **of** ROM\_VHDL **is**

**type** ROM\_TYPE **is** **array** **(**0 **to** 255**)** **of** std\_logic\_vector **(**7 **downto** 0**);**

**constant** rom\_content **:** ROM\_TYPE **:=** **(**

-- Format L Test

"00100100"**,**"00001111"**,** -- LOADIMM.LOWER #15

"00100101"**,**"00000101"**,** -- LOADIMM.UPPER #5

"00100110"**,**"01111000"**,** -- MOV R1, R7

"00100100"**,**"00000110"**,** -- LOADIMM.LOWER #6

"00100101"**,**"00000000"**,** -- LOADIMM.UPPER #0

"00100110"**,**"10111000"**,** -- MOV R2, R7

"00100010"**,**"10001000"**,** -- STORE R2, R1

"00100000"**,**"11010000"**,** -- LOAD R3, R2

**others** **=>** "00000000"**);** -- NOP

**begin**

p1**:** **process** **(**clk**)**

**variable** add\_in **:** integer **:=** 0**;**

**begin**

**if** **rising\_edge(**clk**)** **then**

add\_in **:=** conv\_integer**(**unsigned**(**addr**));**

data**(**7 **downto** 0**)** **<=** rom\_content**(**add\_in**+**1**);**

data**(**15 **downto** 8**)** **<=** rom\_content**(**add\_in**);**

**end** **if;**

**end** **process;**

**end** BHV**;**

### Register File

**architecture** behavioural **of** register\_file **is**

**type** reg\_array **is** **array** **(**integer **range** 0 **to** 7**)** **of** std\_logic\_vector**(**15 **downto** 0**);**

--internals signals

**signal** reg\_file **:** reg\_array**;** **begin**

--write operation

**process(**clk**)**

**begin**

**if(**clk**=**'0' and clk'**event)** **then** **if(**rst**=**'1'**)** **then**

**for** i **in** 0 **to** 7 **loop**

reg\_file**(**i**)<=** **(others** **=>** '0'**);**

**end** **loop;**

**elsif(**wr\_enable**=**'1'**)** **then**

**case** wr\_index**(**2 **downto** 0**)** **is**

**when** "000" **=>** reg\_file**(**0**)** **<=** wr\_data**;**

**when** "001" **=>** reg\_file**(**1**)** **<=** wr\_data**;**

**when** "010" **=>** reg\_file**(**2**)** **<=** wr\_data**;**

**when** "011" **=>** reg\_file**(**3**)** **<=** wr\_data**;**

**when** "100" **=>** reg\_file**(**4**)** **<=** wr\_data**;**

**when** "101" **=>** reg\_file**(**5**)** **<=** wr\_data**;**

**when** "110" **=>** reg\_file**(**6**)** **<=** wr\_data**;**

**when** "111" **=>**

**case** wr\_mode**(**1 **downto** 0**)** **is**

**when** "01" **=>**

-- Lower Byte

reg\_file**(**7**)(**7 **downto** 0**)** **<=** wr\_data**(**7 **downto** 0**);**

**when** "10" **=>**

-- Upper Byte

reg\_file**(**7**)(**15 **downto** 8**)** **<=** wr\_data**(**7 **downto** 0**);**

**when** **others** **=>**

reg\_file**(**7**)** **<=** wr\_data**;**

**end** **case;**

**when** **others** **=>** **NULL;** **end** **case;**

**end** **if;**

**end** **if;**

**end** **process;**

--read operation

rd\_data1 **<=**

reg\_file**(**0**)** **when(**rd\_index1**=**"000"**)** **else**

reg\_file**(**1**)** **when(**rd\_index1**=**"001"**)** **else**

reg\_file**(**2**)** **when(**rd\_index1**=**"010"**)** **else**

reg\_file**(**3**)** **when(**rd\_index1**=**"011"**)** **else**

reg\_file**(**4**)** **when(**rd\_index1**=**"100"**)** **else**

reg\_file**(**5**)** **when(**rd\_index1**=**"101"**)** **else**

reg\_file**(**6**)** **when(**rd\_index1**=**"110"**)** **else** reg\_file**(**7**);**

rd\_data2 **<=**

reg\_file**(**0**)** **when(**rd\_index2**=**"000"**)** **else**

reg\_file**(**1**)** **when(**rd\_index2**=**"001"**)** **else**

reg\_file**(**2**)** **when(**rd\_index2**=**"010"**)** **else**

reg\_file**(**3**)** **when(**rd\_index2**=**"011"**)** **else**

reg\_file**(**4**)** **when(**rd\_index2**=**"100"**)** **else**

reg\_file**(**5**)** **when(**rd\_index2**=**"101"**)** **else**

reg\_file**(**6**)** **when(**rd\_index2**=**"110"**)** **else** reg\_file**(**7**);**

--fill this part

**end** behavioural**;**

### Program Counter

**architecture** Behavioral **of** program\_counter **is**

**signal** counterValue **:** STD\_LOGIC\_VECTOR**(**15 **downto** 0**)** **:=** **(others** **=>** '0'**);**

**attribute** S**:** string**;**

**attribute** S **of** counterValue**:** **signal** **is** "Yes"**;**

**begin**

**process(**clk**)**

**begin**

**if(**clk**=**'0' and clk'**event)** **then**

**if** **(**rst **=** '1'**)** **then**

counterValue **<=** X"0000"**;**

**elsif** **(**hold **=** '0'**)** **then**

**if** **(**write\_en **=** '1'**)** **then**

counterValue **<=** overwrite\_value**(**15 **downto** 1**)** **&** "0"**;** -- Mask LSB to ensure word alignment

**else**

counterValue **<=** next\_value**;**

**end** **if;**

**end** **if;**

**end** **if;**

**end** **process;**

current\_value **<=** counterValue**;**

**end** Behavioral**;**

**architecture** Behavioral **of** pc\_incrementor **is**

**begin**

output **<=** std\_logic\_vector**(**unsigned**(**input**)** **+**2 **);**

**end** Behavioral**;**

### Hex to 7-Segment

**architecture** Behavioral **of** hex\_to\_7seg **is**

**signal** temp**:** STD\_LOGIC\_VECTOR**(**3 **downto** 0**):=**"0000"**;**

**begin**

**process(**clk**)**

**begin**

**if(**clk**=**'1' and clk'**event)** **then**

**if** temp **=** "0000" **then**

temp **<=** "0001"**;**

**elsif** temp **=** "0001" **then**

temp **<=** "0010"**;**

**elsif** temp **=** "0010" **then**

temp **<=** "0100"**;**

**elsif** temp **=** "0100" **then**

temp **<=** "1000"**;**

**else** temp **<=** "0000"**;**

**end** **if;**

**end** **if;**

**end** **process;**

**process(**temp**,** CPU\_result**)**

**begin**

**if** temp **=** "0001" **then**

anodes **<=** not"0001"**;**

--LSB of Cathodes is CA

--EX : Display 0 -> "CG CF CE CD CC CB CA" = not "0111111"

**case** CPU\_result**(**3 **downto** 0**)** **is**

**when** "0000" **=>** cathodes **<=** not "0111111"**;**

**when** "0001" **=>** cathodes **<=** not "0000110"**;**

**when** "0010" **=>** cathodes **<=** not "1011011"**;**

**when** "0011" **=>** cathodes **<=** not "1001111"**;**

**when** "0100" **=>** cathodes **<=** not "1100110"**;**

**when** "0101" **=>** cathodes **<=** not "1101101"**;**

**when** "0110" **=>** cathodes **<=** not "1111101"**;**

**when** "0111" **=>** cathodes **<=** not "0000111"**;**

**when** "1000" **=>** cathodes **<=** not "1111111"**;**

**when** "1001" **=>** cathodes **<=** not "1101111"**;**

**when** "1010" **=>** cathodes **<=** not "1110111"**;**

**when** "1011" **=>** cathodes **<=** not "1111100"**;**

**when** "1100" **=>** cathodes **<=** not "0111001"**;**

**when** "1101" **=>** cathodes **<=** not "1011110"**;**

**when** "1110" **=>** cathodes **<=** not "1111001"**;**

**when** "1111" **=>** cathodes **<=** not "1110001"**;**

**when** **others** **=>** cathodes **<=** not "0000000"**;**

**end** **case;**

**elsif** temp **=** "0010" **then**

anodes **<=** not"0010"**;**

**case** CPU\_result**(**7 **downto** 4**)** **is**

**when** "0000" **=>** cathodes **<=** not "0111111"**;**

**when** "0001" **=>** cathodes **<=** not "0000110"**;**

**when** "0010" **=>** cathodes **<=** not "1011011"**;**

**when** "0011" **=>** cathodes **<=** not "1001111"**;**

**when** "0100" **=>** cathodes **<=** not "1100110"**;**

**when** "0101" **=>** cathodes **<=** not "1101101"**;**

**when** "0110" **=>** cathodes **<=** not "1111101"**;**

**when** "0111" **=>** cathodes **<=** not "0000111"**;**

**when** "1000" **=>** cathodes **<=** not "1111111"**;**

**when** "1001" **=>** cathodes **<=** not "1101111"**;**

**when** "1010" **=>** cathodes **<=** not "1110111"**;**

**when** "1011" **=>** cathodes **<=** not "1111100"**;**

**when** "1100" **=>** cathodes **<=** not "0111001"**;**

**when** "1101" **=>** cathodes **<=** not "1011110"**;**

**when** "1110" **=>** cathodes **<=** not "1111001"**;**

**when** "1111" **=>** cathodes **<=** not "1110001"**;**

**when** **others** **=>** cathodes **<=** not "0000000"**;**

**end** **case;**

**elsif** temp **=** "0100" **then**

anodes **<=** not"0100"**;**

**case** CPU\_result**(**11 **downto** 8**)** **is**

**when** "0000" **=>** cathodes **<=** not "0111111"**;**

**when** "0001" **=>** cathodes **<=** not "0000110"**;**

**when** "0010" **=>** cathodes **<=** not "1011011"**;**

**when** "0011" **=>** cathodes **<=** not "1001111"**;**

**when** "0100" **=>** cathodes **<=** not "1100110"**;**

**when** "0101" **=>** cathodes **<=** not "1101101"**;**

**when** "0110" **=>** cathodes **<=** not "1111101"**;**

**when** "0111" **=>** cathodes **<=** not "0000111"**;**

**when** "1000" **=>** cathodes **<=** not "1111111"**;**

**when** "1001" **=>** cathodes **<=** not "1101111"**;**

**when** "1010" **=>** cathodes **<=** not "1110111"**;**

**when** "1011" **=>** cathodes **<=** not "1111100"**;**

**when** "1100" **=>** cathodes **<=** not "0111001"**;**

**when** "1101" **=>** cathodes **<=** not "1011110"**;**

**when** "1110" **=>** cathodes **<=** not "1111001"**;**

**when** "1111" **=>** cathodes **<=** not "1110001"**;**

**when** **others** **=>** cathodes **<=** not "0000000"**;**

**end** **case;**

**elsif** temp **=** "1000" **then**

--

anodes **<=** not"1000"**;**

**case** CPU\_result**(**15 **downto** 12**)** **is**

**when** "0000" **=>** cathodes **<=** not "0111111"**;**

**when** "0001" **=>** cathodes **<=** not "0000110"**;**

**when** "0010" **=>** cathodes **<=** not "1011011"**;**

**when** "0011" **=>** cathodes **<=** not "1001111"**;**

**when** "0100" **=>** cathodes **<=** not "1100110"**;**

**when** "0101" **=>** cathodes **<=** not "1101101"**;**

**when** "0110" **=>** cathodes **<=** not "1111101"**;**

**when** "0111" **=>** cathodes **<=** not "0000111"**;**

**when** "1000" **=>** cathodes **<=** not "1111111"**;**

**when** "1001" **=>** cathodes **<=** not "1101111"**;**

**when** "1010" **=>** cathodes **<=** not "1110111"**;**

**when** "1011" **=>** cathodes **<=** not "1111100"**;**

**when** "1100" **=>** cathodes **<=** not "0111001"**;**

**when** "1101" **=>** cathodes **<=** not "1011110"**;**

**when** "1110" **=>** cathodes **<=** not "1111001"**;**

**when** "1111" **=>** cathodes **<=** not "1110001"**;**

**when** **others** **=>** cathodes **<=** not "0000000"**;**

**end** **case;**

**else**

cathodes **<=** not "0000000"**;**

anodes **<=** "0000"**;**

**end** **if;**

**end** **process;**

**end** Behavioral**;**

## RTL Schematics

### Processor Top Level Schematic

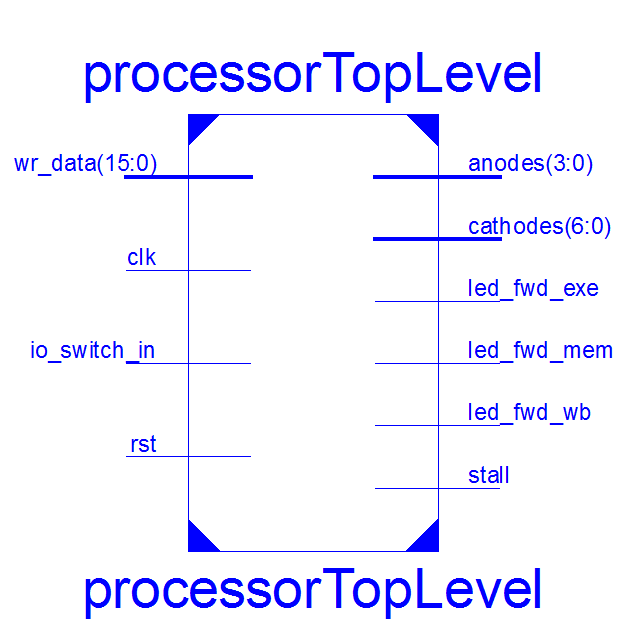


Figure 14 - Processor Top Level Schematic

### Processor Internal Connections

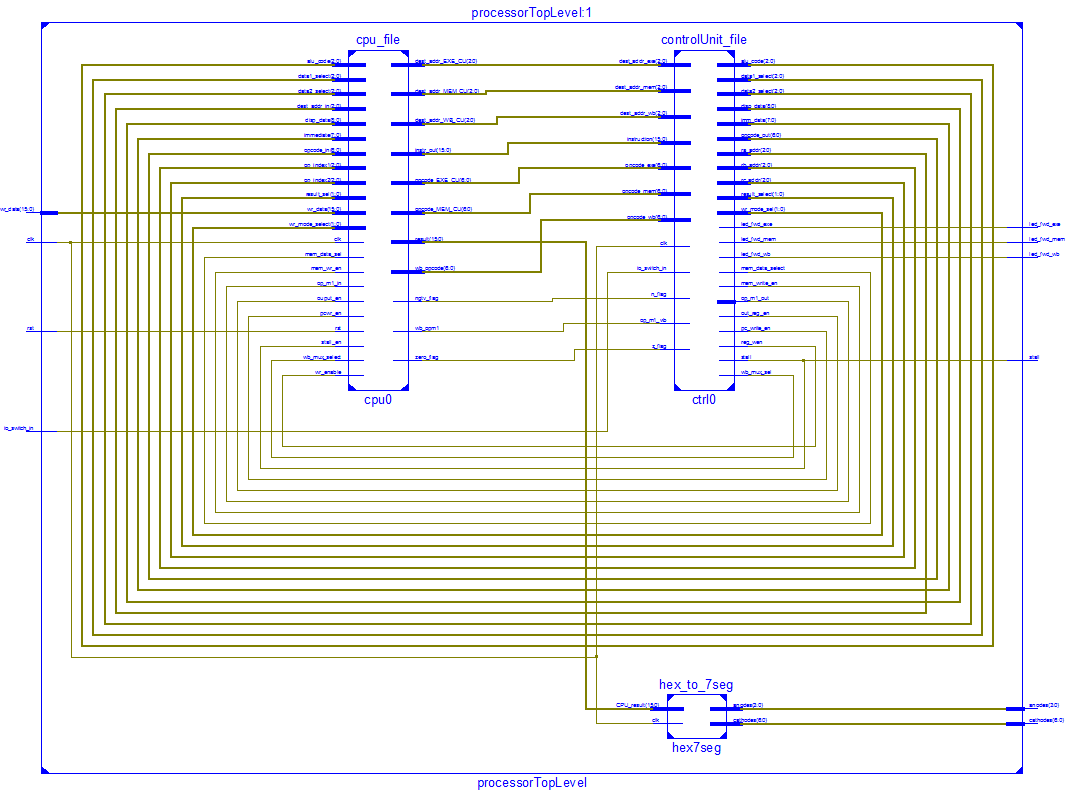


Figure 15 - Processor Internal Connections

### CPU File Top Level Schematic

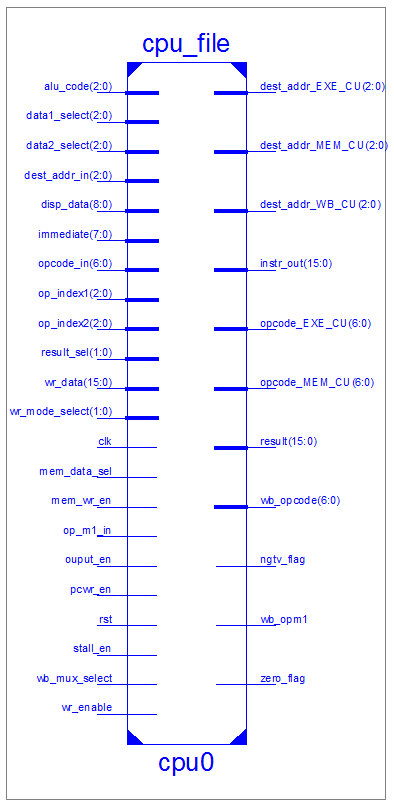


Figure 16 - CPU Top Level Schematic

### CPU Internal Connections

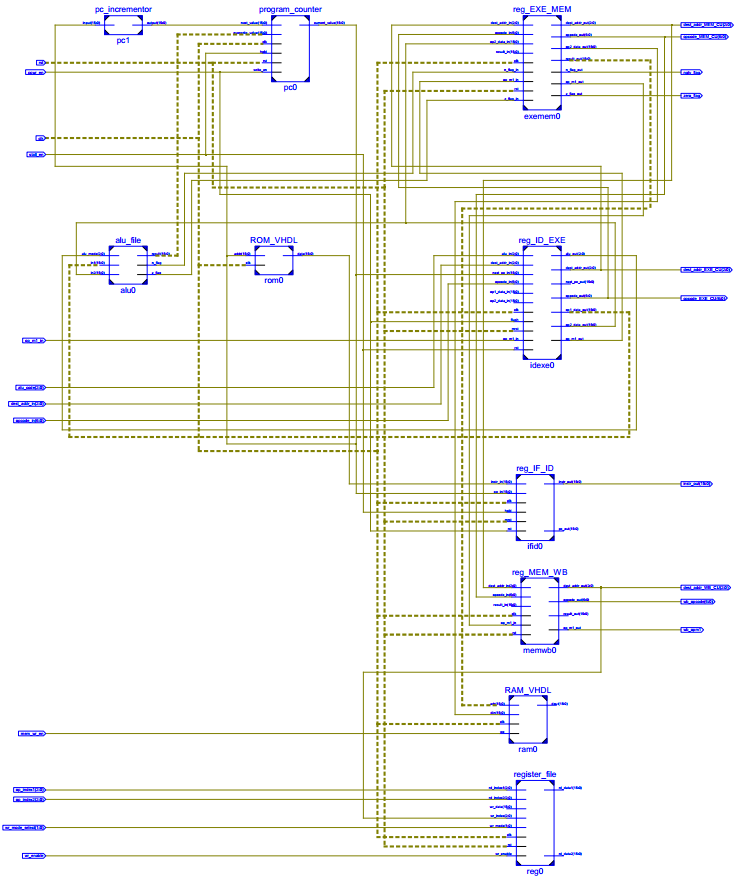


Figure 17 - CPU Internal Connections (MUX's Omitted)

### CPU Pipeline Stages

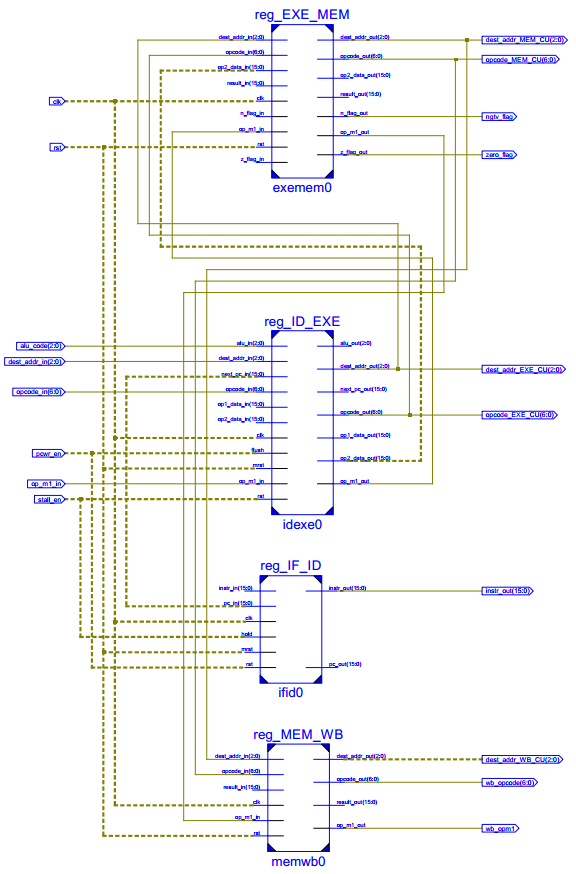


Figure 18 - Pipeline Stages Connection Schematic

### Control Unit Top Level Schematic

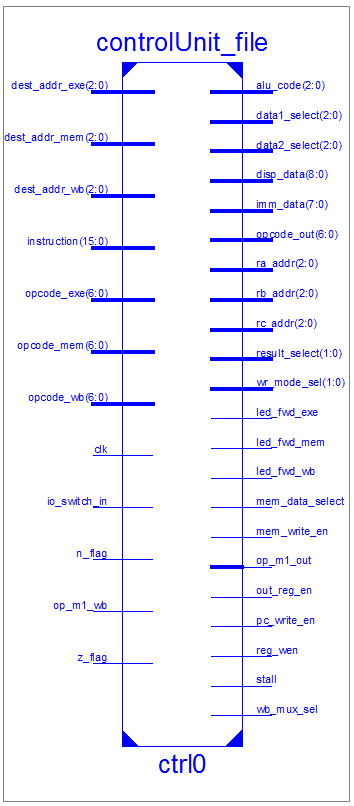


Figure 19 - Control Unit Top Level Schematic