|  |  |
| --- | --- |
|  | 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](#_Toc480291630)

[II. Table of Figures 4](#_Toc480291631)

[III. Objective 5](#_Toc480291632)

[Abstract 5](#_Toc480291633)

[Pipelining 5](#_Toc480291634)

[RISC-Like Instruction Set 5](#_Toc480291635)

[Project Objective 5](#_Toc480291636)

[IV. Design Requirement 6](#_Toc480291637)

[General Design Requirements 6](#_Toc480291638)

[Instruction Set 6](#_Toc480291639)

[Registers and Memory 6](#_Toc480291640)

[Pipelining 6](#_Toc480291641)

[Program Execution 7](#_Toc480291642)

[General Implementation Choices 7](#_Toc480291643)

[Data Forwarding and Hazard Detection 7](#_Toc480291644)

[Branch Prediction 7](#_Toc480291645)

[Design Process Guidelines Followed 7](#_Toc480291646)

[V. System 8](#_Toc480291647)

[High Level System Hierarchy Breakdown 8](#_Toc480291648)

[CPU Block Diagram 10](#_Toc480291649)

[VI. Individual Module Design 10](#_Toc480291650)

[Top Level Processor Unit 10](#_Toc480291651)

[CPU 11](#_Toc480291652)

[ALU 11](#_Toc480291653)

[Multiplexers 11](#_Toc480291654)

[Program Counter 12](#_Toc480291655)

[Pipeline Registers 12](#_Toc480291656)

[RAM Module 12](#_Toc480291657)

[ROM Module 13](#_Toc480291658)

[Register File 14](#_Toc480291659)

[Control Unit 14](#_Toc480291660)

[Data Forwarding and Hazard Detection 14](#_Toc480291661)

[Hex to Seven Segment Display 16](#_Toc480291662)

[VII. Simulation Result 16](#_Toc480291663)

[Format A Instructions 16](#_Toc480291664)

[Format B Instructions 16](#_Toc480291665)

[Format L Instructions 17](#_Toc480291666)

[VIII. Analysis and Discussion 17](#_Toc480291667)

[Hardware Verification Procedures 17](#_Toc480291668)

[IX. Conclusion 17](#_Toc480291669)

[X. Appendices 18](#_Toc480291670)

[VHDL Code 18](#_Toc480291671)

[All VHDL Code available at 18](#_Toc480291672)

[https://github.com/HansSanitizer/CENG-450-Project/tree/master/processor 18](#_Toc480291673)

[RTL Schematics 19](#_Toc480291674)

[Processor Top Level Schematic 19](#_Toc480291675)

[Processor Internal Connections 20](#_Toc480291676)

[CPU File Top Level Schematic 21](#_Toc480291677)

[CPU Internal Connections 22](#_Toc480291678)

[CPU Pipeline Stages 24](#_Toc480291679)

[Control Unit Top Level Schematic 26](#_Toc480291680)

# 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

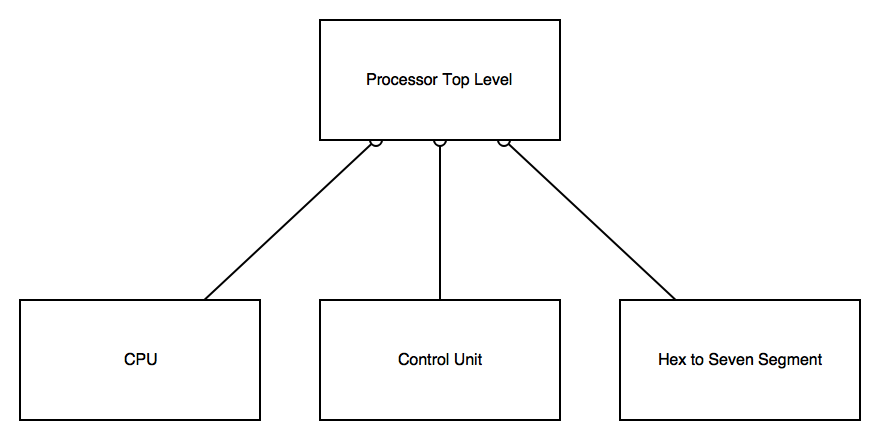


Figure - System Hierarchy Breakdown

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.

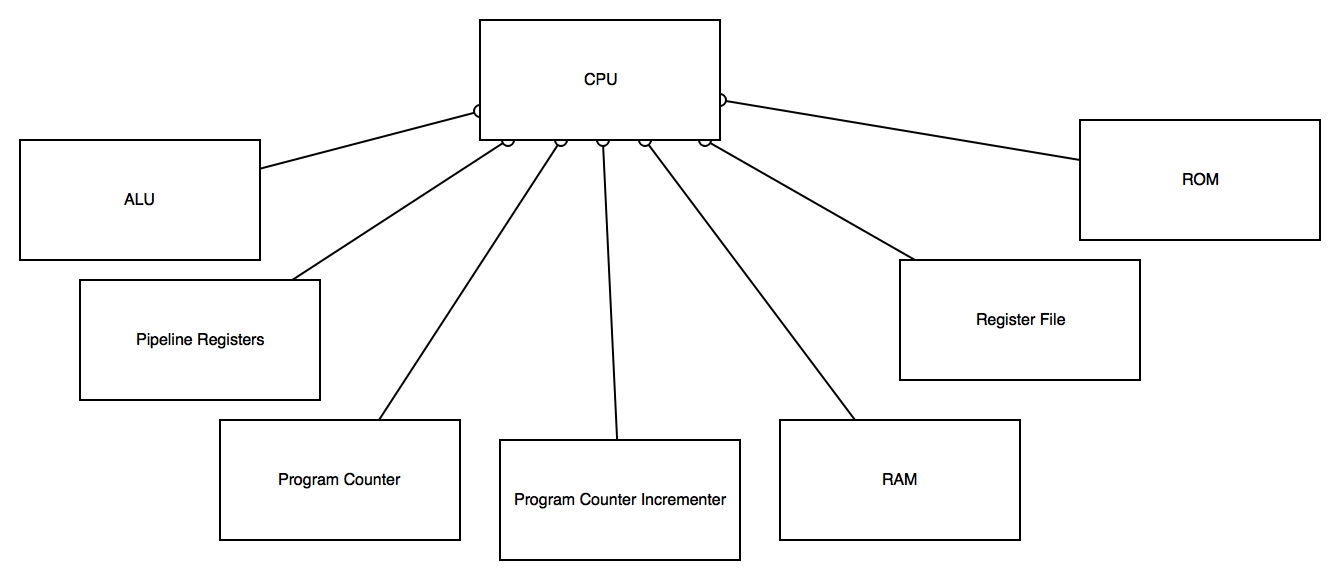


Figure - CPU Hierarchy Breakdown

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

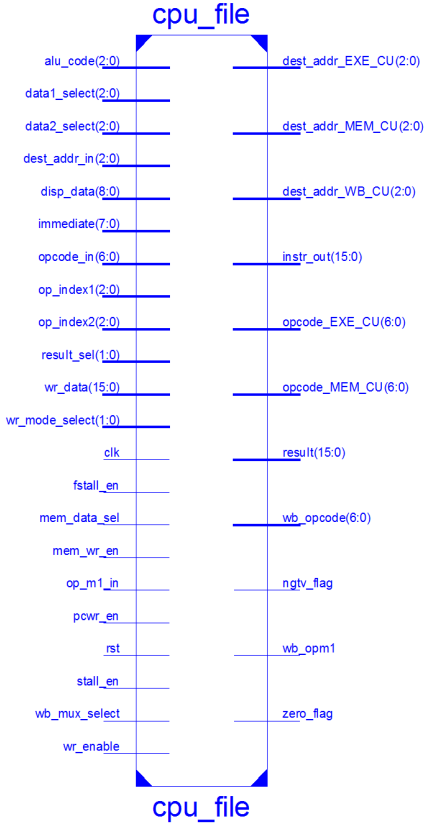


Figure - CPU File RTL Block Schematic

# 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

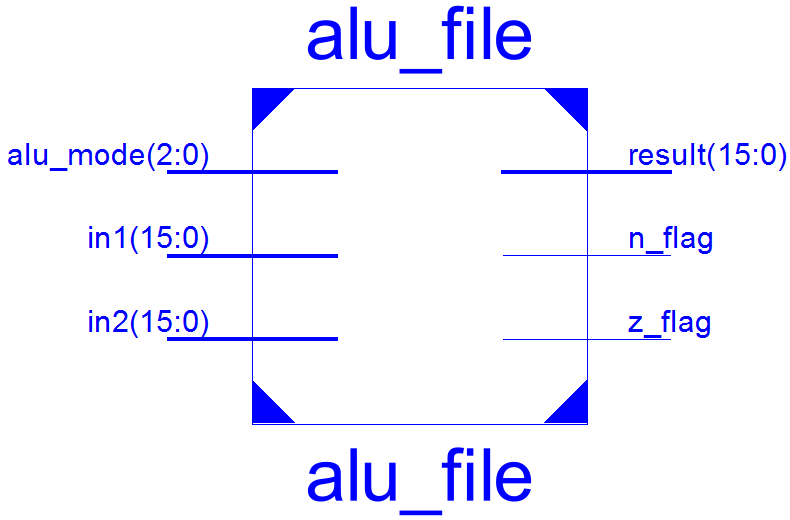


Figure - ALU Block Diagram

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

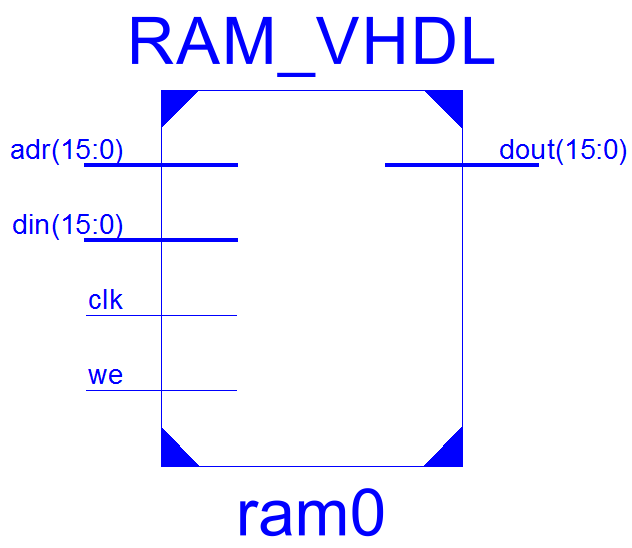


Figure - RAM Module Block Diagram

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

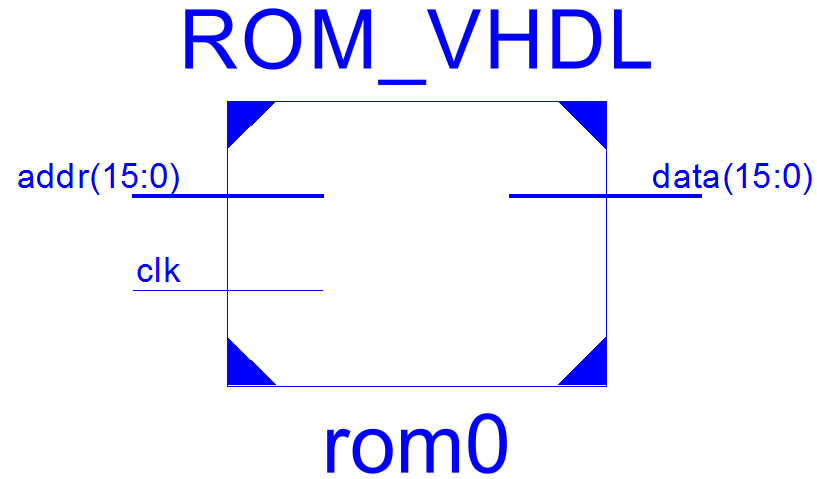


Figure - ROM Module Block Diagram

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

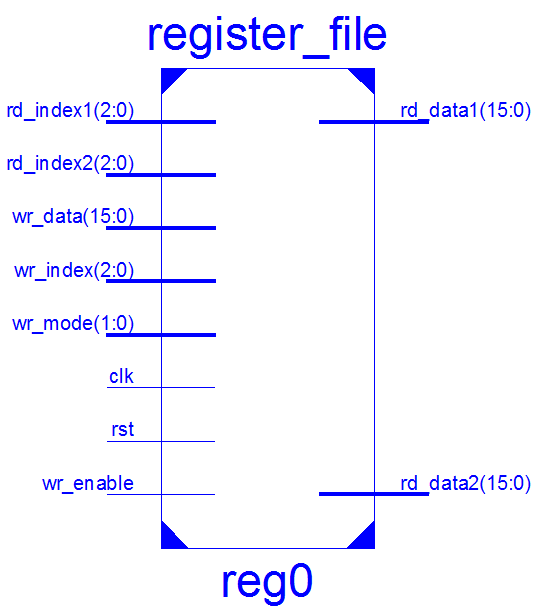


Figure – Register File Module Block Diagram

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 13* 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 | |

|  |  |
| --- | --- |
| 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

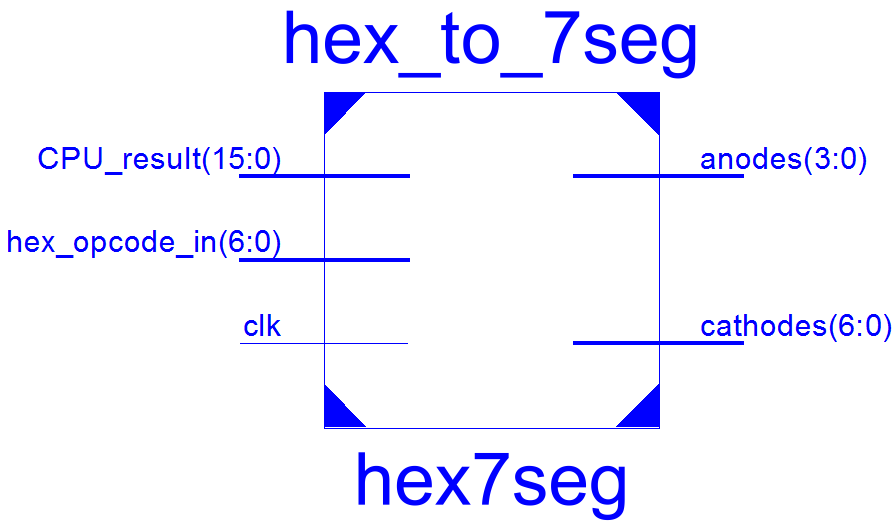


Figure - Hex to Seven Segment Display Module Block Diagram

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

## Format B Instructions

## Format L Instructions

# Analysis and Discussion

## Hardware Verification Procedures

For our on hardware testing we obtained 50 MHz operation speed

# 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

## RTL Schematics

### Processor Top Level Schematic

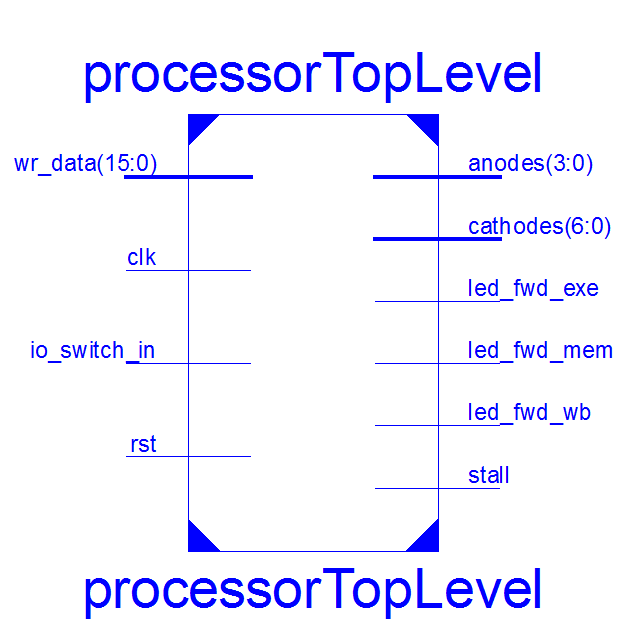


Figure - Processor Top Level Schematic

### Processor Internal Connections

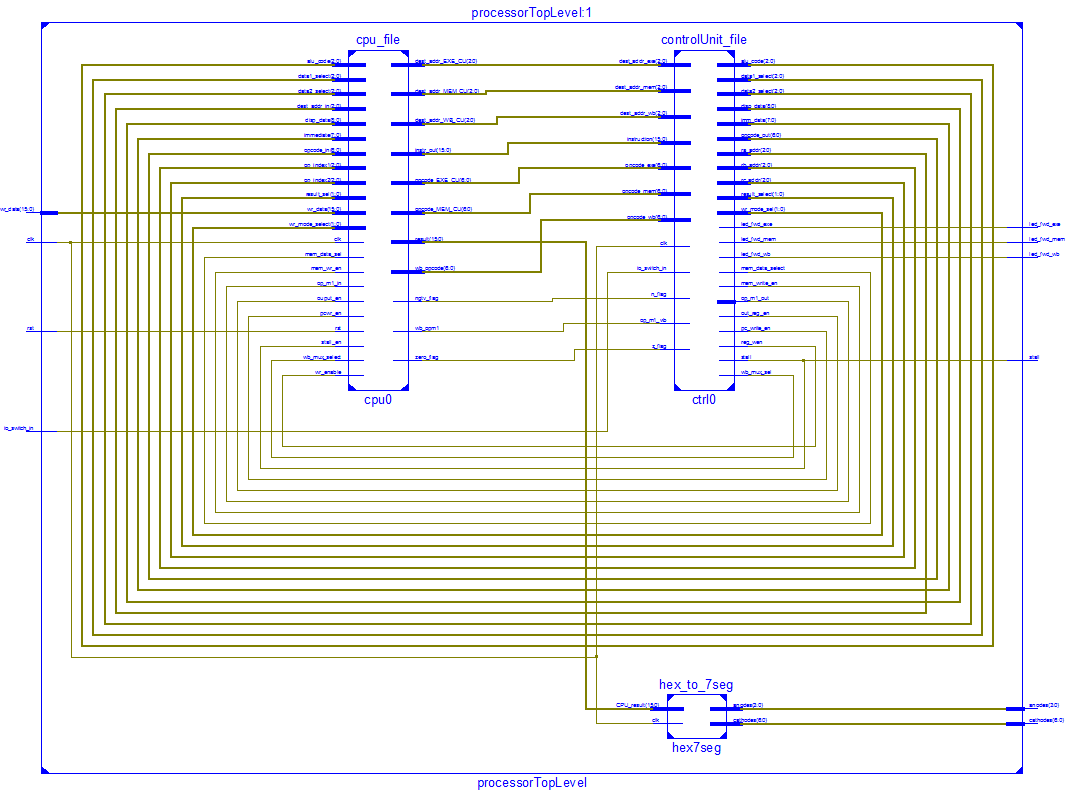


Figure - Processor Internal Connections

### CPU File Top Level Schematic

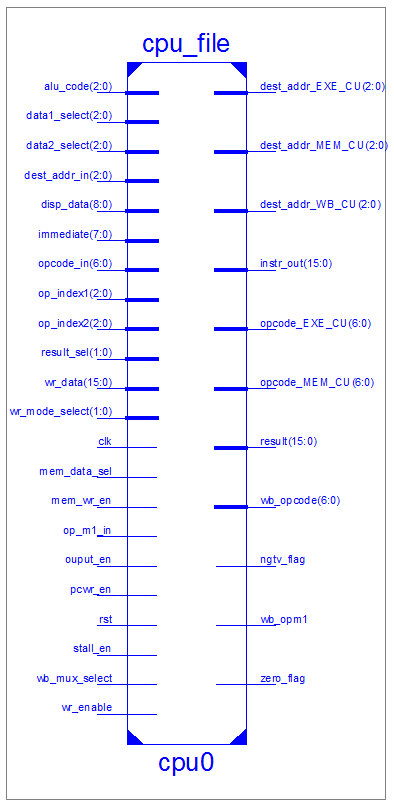


Figure - CPU Top Level Schematic

### CPU Internal Connections

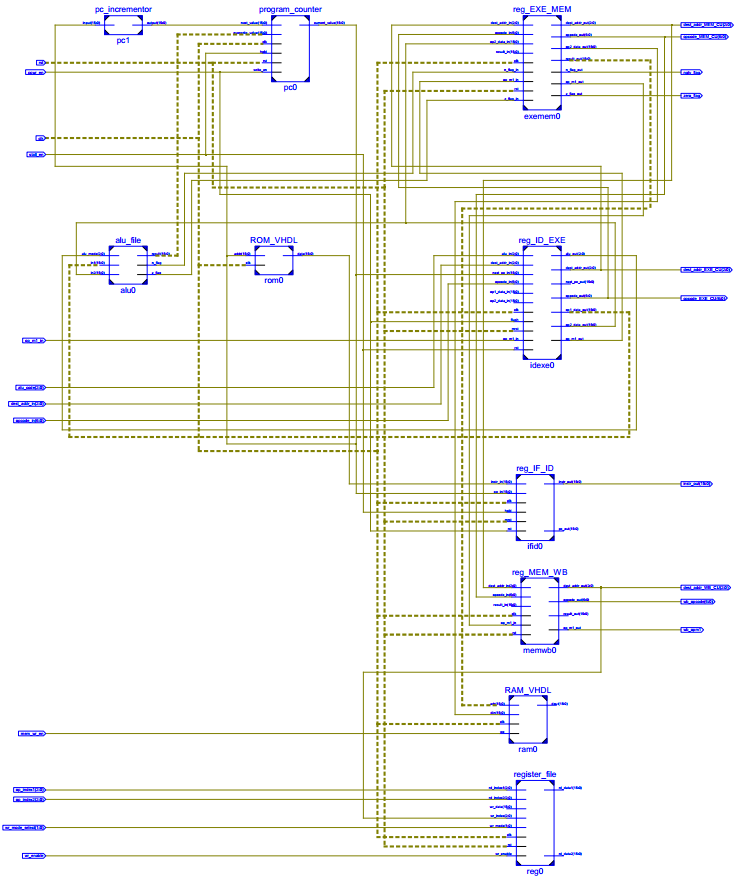


Figure - CPU Internal Connections (MUX's Omitted)

### CPU Pipeline Stages

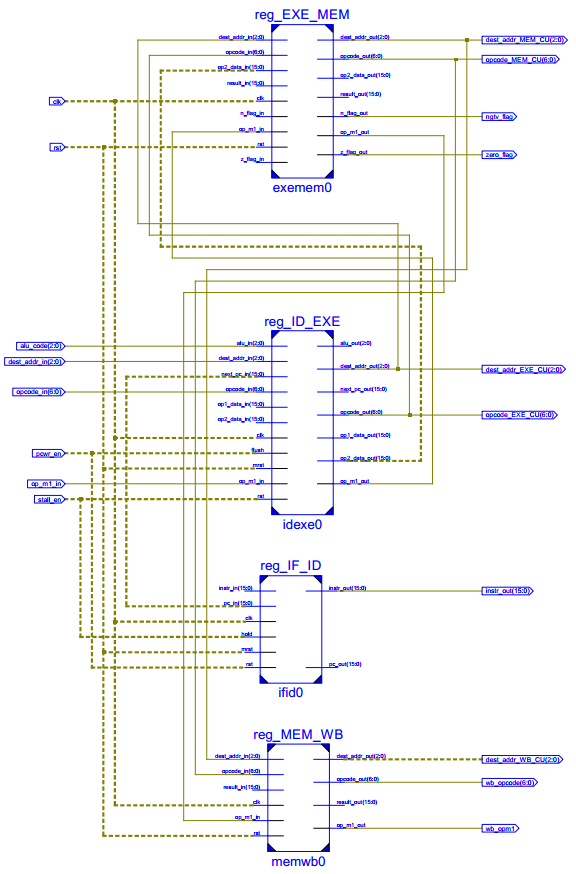


Figure - Pipeline Stages Connection Schematic

### Control Unit Top Level Schematic

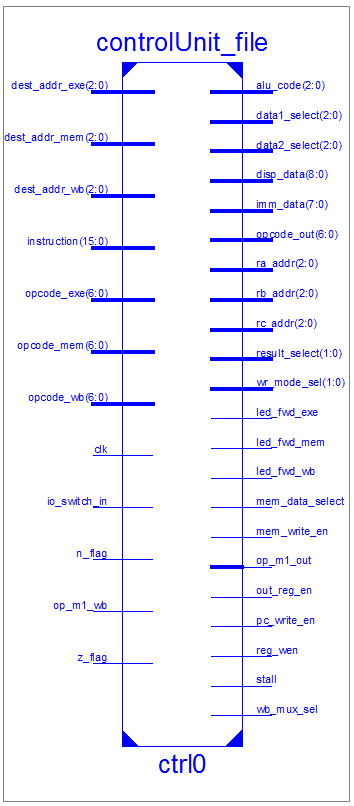


Figure - Control Unit Top Level Schematic