The purpose of this third homework assignment is to provide an experience for designing a Multi-Cycle CPU with various instruction sets. The Datapath and Controller designs are given – all components are to be modeled in Verilog, and then verified through the functionality of a full testbench.

**Code Submission and Memory Files**

All Verilog code and memory files utilized in this CPU design, with detailed comments that go into detail regarding the definition of all lines of code, can be found in the root folder of the HW 3 ZIP file submitted for this assignment:

Verilog files:

*Controller.v, Datapath.v, CompArch\_CPU.v, CompArch\_CPU\_Tester.v*

Data (memory) files:

InstructionFile.mem, HexadecimalFile.mem

**Datapath and Controller Implementations**

The Datapath of the *CompArch\_CPU* is seen below, taken from the HW description:



The Datapath was developed all in one file, *Datapath.v*, with all of the different registers being instantiated and updated in *always* procedural statements and by their individual controller signals: *enACC*, *enIR*, *enPC*. The ALU, who is ultimately responsible for updating the *ACC* and *PR*, is updated whenever the *ALU\_op* changes or whenever *enACC* is asserted high. It is also responsible for multiplexing of signals on busses and for setting the *op\_code* coming in from the *IR*.

The Controller of the *CompArch\_CPU* is seen below, taken from the HW description:



Using these four stages, the Controller was realized using a Huffman style of coding, with one procedural block responsible for tracking the *reset/next\_state* transition, and the other for performing the operations within the states themselves. Below are quick descriptions of each of the states and which signals they interact with across the Datapath:

**Reset**:

* While *reset* *== 1*, stay in *`Reset* state. Begin fetching instructions once *reset* deasserts its current value (*reset ==* 0).

**Fetch**:

* Place the *PC* address on the *addr\_bus*. By setting *rd\_mem = 1*, place information from *mem[addr\_bus]* on *data\_bus*. Load the 16’b received into the *IR*, and *increment PC by 1*.

**Execute**:

* LDA – Place the *IR* information on the *addr\_bus*, then read *mem[addr\_bus]* into *data\_bus*. Place *data\_bus* on *MuxB*, and have it pass through the ALU, and then load it into the *ACC*.
* STA – Place current *ACC* data onto the *data\_bus*. Place *IR data* onto the *addr\_bus*. Write into *mem[addr\_bus]*.
* ADD (SUM) – Place the *IR* information on the *addr\_bus*, then read *mem[addr\_bus]* into the *data\_bus*. Place *data\_bus* on *MuxB*, and add it to the current contents of *ACC*. Then store results back into *ACC*.
* SUB – Place the *IR* information on the *addr\_bus*, then read *mem[addr\_bus]* into the *data\_bus*. Place *data\_bus* on *MuxB*, and subtract it from the current contents of *ACC*. Then store results back into *ACC*.
* JMP – Place the *IR* information on the *addr\_bus*. Place *addr\_bus* on *MuxB*, then pass it through and store it into PC.
* JEZ – If the ACC is 0 (based on zero signal), place the *IR* information on the *addr\_bus*. Place *addr\_bus* on *MuxB*, then pass it through and store it into *PC*.
* HLT – If this instruction is seen, the next state is the halt state.

**Halt**:

* Stay in this state until *reset = 1*, then go back to `*Reset* state.

A few notes about the Controller:

* A forced time delay is built in whenever the *rd\_mem* and *wr\_mem* signals are asserted. This is done so that there is sufficient time for the *data\_bus* to be populated based on force timing delays seen in thetestbench *Memory\_Read\_Write* procedural block.
* Another forced time delay is built in for when the ALU must perform an operation. This is done to allow sufficient amount of time for the calculation to be performed in the *Datapath*, and allows for accurate transmission of data. This will cause a delay in the *ACC* receiving and storing data.

**Testbench Methodology and Planning**

Before going ahead and immediately programming the testbench, it was decided to implement an assembler to make it easier to load in programs to the CPU. The assembler would take in a specified instruction set, as provided by the HW description, and turn it into machine language that the CPU can understand. With a multi-cycle CPU and shared memory for both data and instructions, properly laying out a plan to validate the functionality of the CPU operation in the testbench is extremely important. The following layout is what was determined:

* Memory will contain 213 spaces, all initialized to 000016.
* The testbench will only utilize the first 30 memory spaces.
* Addresses in the memory 0 – 14 = Instructions
* Addresses in the memory 15 – 29 = Data (referenced by Instructions)
* The first 15 addresses will fully validate all functionality of the CPU.

Below is the layout of what the testbench scenario will look like:



Utilizing the *PC* (Program Counter), the testbench will navigate through the Instructions contained within Lines 00 – 14, accessing (reading and writing) data to Lines 15 – 29 as necessary. Based on the linear set of instructions, the following section details what should be expected based on each step:

Line 00: Load ACC with Data @ Line 25 > ACC = 10010

Line 01: Add value @ Line26 to ACC > ACC = 30010

Line 02: Subtract value @ Line 20 from ACC > ACC = 25010

Line 03: Add value @ Line 16 to ACC > ACC = 26010

Line 04: Store ACC in mem[001D] (Line 29) > mem[001D] = 010416

Line 05: Jump to instruction @ Line 08 > Skip HALT and SUB

Line 08: Since ACC != 0, ignore JEZ > Continue to Line 09

Line 09: Load ACC with Data @ Line 15 > ACC = 010

Line 10: Since ACC == 0, jump to Line 12 > Skip HLT

Line 12: Load ACC with Data @ Line 18 > ACC = 3010

Line 13: HALT the CompArch\_CPU > HALT until *reset = 1*

Line 14: Store ACC in mem[001C] (Line 28) > mem[001C] = 001E16

**Testbench Assembler (Machine Language)**

Utilizing the given instruction format, the following *InstructionFile.mem* file is created in order to populate the *HexadecimalFile.mem* file:

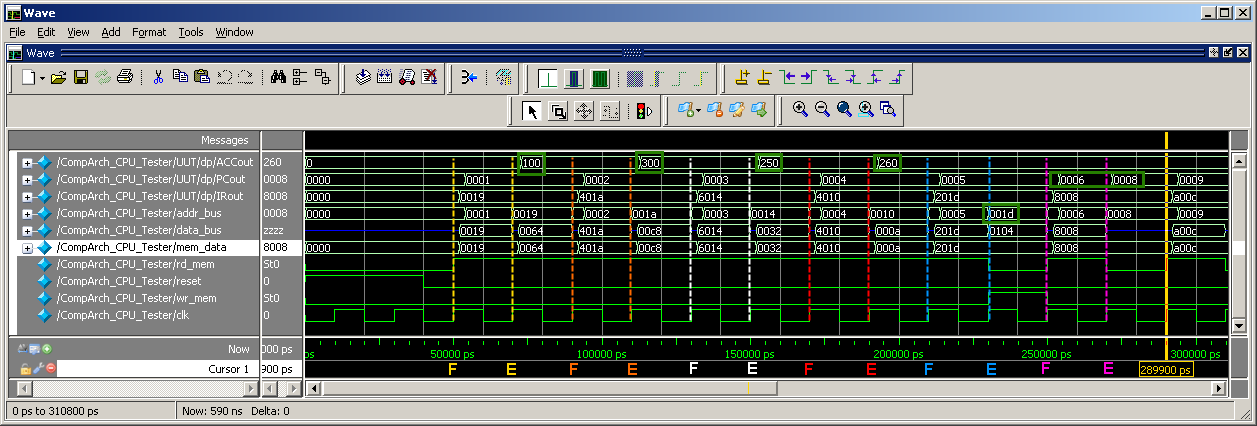
****

**Testbench Confirmation (Pt1)**

The two screenshots of the waveforms below showcase the output of the testbench created for the *CompArch\_ALU*. Each colored **F** and **E** pair represents one of the *`Fetch* and *`Execute* pairs of the Controller, and should map to the instructions listed in the *Testbench Methodology and Planning* section of this report. The green boxes in the output also explicitly highlight the items that should be changing at that point in time in the simulation.

When zoomed in closely, the values of the output can be seen. These screenshots are also saved in the root directory of the HW 3 folder.

**Waveform Analysis for 1st Screenshot**



**F & E –** The *ACC* should be **100**. **CORRECT**

**F & E –** The *ACC* should be **300**. **CORRECT**

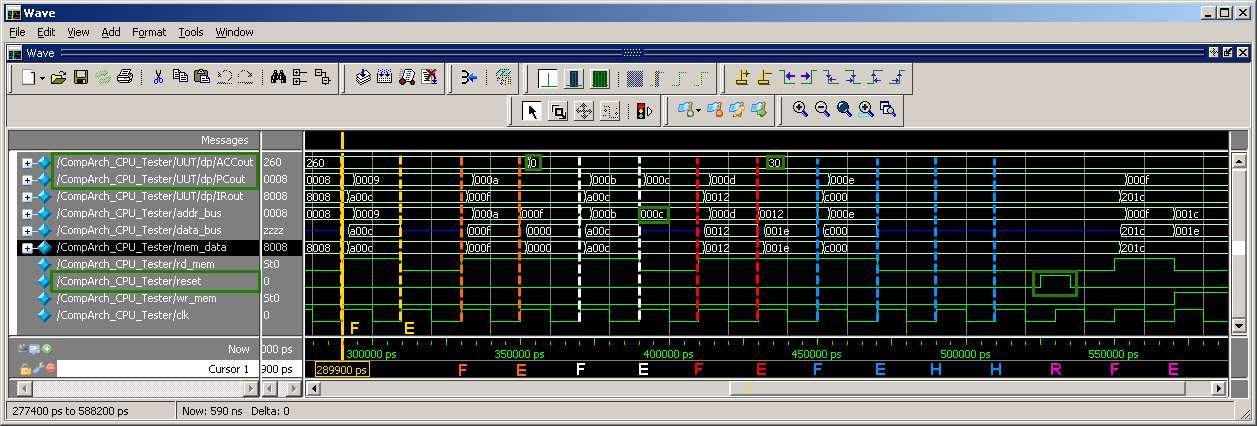
**F & E –** The *ACC* should be **250**. **CORRECT**

**F & E –** The *ACC* should be **260**. **CORRECT**

**F & E –** Store *ACC* in mem[**001d**]. **NEED TO CONFIRM**

**F & E –** Jump to address @ **0008**. **CORRECT**

**Waveform Analysis for 2nd Screenshot**



**F & E – Do nothing**; JEZ will not happen since ACC != 0. **CORRECT**

**F & E –** The *ACC* should be **0**. **CORRECT**

**F & E –** Jump to address @ **000c**, since JEZ is now valid. **CORRECT**

**F & E –** The *ACC* should be **30**. **CORRECT**

**F & E –** *Halt* the machine until *reset = 1* is seen. **CORRECT**

**F & E –** Reset machine, store *ACC* in mem[**001c**]. **NEED TO CONFIRM**

According to all waveform outputs, all operations occurred as intended. The last thing that is needs to be verified is that data was properly written when the write instructions were issued. This can be verified by making a comparison between the *HexadecimalFile.mem* before the CompArch\_CPU operation was started, and after once all computations were performed in the simulation.

The comparison from both files can be seen on the following page:



Line 04: Store ACC in mem[001D] (Line 29) > mem[001D] = 010416

Line 14: Store ACC in mem[001C] (Line 28) > mem[001C] = 001E16

**F & E –** Store *ACC* in mem[**001d**]. **CORRECT**

**F & E –** Reset machine, store *ACC* in mem[**001c**]. **CORRECT**

Thus, all functionality and capabilities for the CompArch\_CPU have been verified.

This concludes the analysis for Homework 03.