**University Of Nevada Las Vegas.**

Class: CPE 300

Semester: Spring 2016

Document author: Carlos Lemus

Author's email: [lemus@unlv.nevada.edu](mailto:lemus@unlv.nevada.edu)

Document topic: Processor Report

1. **Description of Assignment**

The purpose of this project was to create a multi-cycle processor that could perform a number of operations given, including some instructions such as: addition, subtraction, storing memory, and jump instruction. The memory RAM would hold the instructions and additional memory that get latched on to the Instruction Register. These instructions are then shared with the controler to read an op-code and output instructions for the rest of the datapath.

1. **Describe the control unit design.**

The control unit is a three bit FSM with 7 states. The main three states are Fetch, Execute, Latch.

* Fetch increments PC and latches the instruction to the Instruction Register.
* Execute reads the opcode (last 4 bytes from the instruction register) and executes the desired instruction. If the first 4 bytes are used to determine the rest of the operation (ALU, Jumps, Miscellaneous), there is a case statement inside of the opcode case statement.
* Latch is used as a cycle to ensure the Instruction is latched on to regulate from one and two byte instructions.

Other states include Load Memory, Store Memory, Halt, Jump Allocate.

* Load Memory allows loading the memory into the accumulator.
* Store Memory allows storing the accumulator into the memory address.
* Halt performs no operation and remains in this state indefinitely. This was a design choice because the processor requires constant high start state to continue through the cycles. If there was a start state Halt could return to Start until start is high again.
* Jump Allocate allows ram to access the jump address and output before latching.

**Fetch**

**Store Memory**

**Halt**

**Load Memory**

**Execute**

**Latch**

Start/~Reset

**Jump**

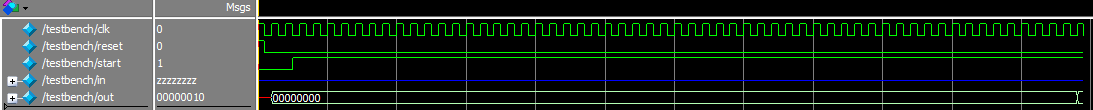
**Allocate**

1. **Hierarchy of Verilog Modules**

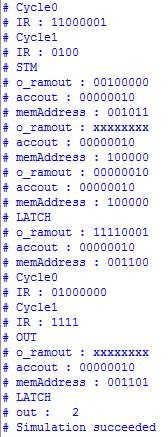
* processor.v
  + datapath.v
  + controller.v
* datapath.v
  + Alu.v
    - Perform addition, subtraction, increment, decrement, AND, OR, NOT
  + Shifter.v
  + Reg8.v
    - 8 bit register
  + Reg6.v
    - 6 bit register
  + Mux4.v
    - 4 choice mux 8 bits
  + Mux2.v
    - 2 choice mux
  + Addrel.v
    - Add pc plus relational jump
  + Subrel.v
    - Subtract pc minus relational jump
  + Regfile.v
    - Register file. R[0] is reserved for zero.
  + Ram.v
    - Read and write memory
  + Mux46
    - 4 choice mux 6 bits

1. **Waveforms: justification of selected inputs/outputs for verification. Screen shots.**

|  |  |  |
| --- | --- | --- |
| **Instruction address** | **Instruction** | **Hex** |
| 000000 | LDI A, 5 | 50 |
| 000001 |  | 05 |
| 000010 | DEC A | E2 |
| 000011 | JNZ 000010 | 80 |
| 000100 |  | 02 |
| 000101 | INC AJ | E1 |
| 000110 | STM 001111,A | 40 |
| 000111 |  | 0F |
| 001000 | STA 0,A | 21 |
| 001001 | ADD A,0 | C1 |
| 001010 | STM 010000, A | 40 |
| 001011 |  | 20 |
| 001100 | OUT | F1 |
| ...001111 | A = 1 from instruction 6 |  |
| ...100000 | A = 2 from instruction 10 |  |



Testbench clock is stopped when output changes.



The o\_ramout, memAddress, accout, have been adjusted to display whenever they change. This shows a small process from the fetch cycle through the cycles until the last instruction.

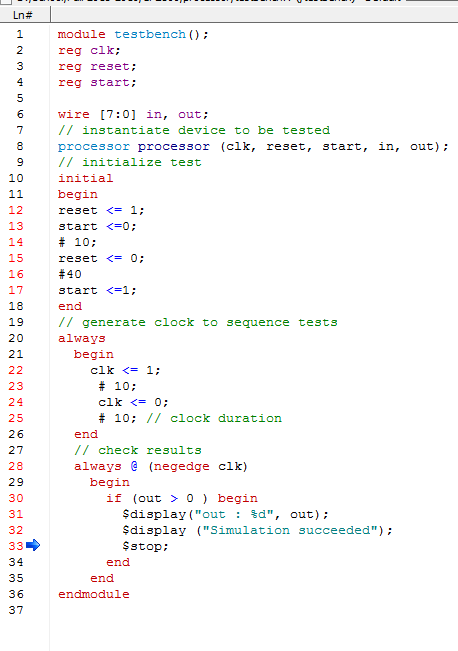
You can see from the memAddress: 100000 after STM instruction that the accumulator is being stored in that address.

1. **Explain the testbench. Proof you were able to verify operation in a comprehensive way. Screen shots.**

**\*** The testbench file has been attached for convenience.

The testbench begins by resetting all the registers in the first state by setting reset to high. You can see this by looking at the first clock cycle in the first wavelength screenshot of this report. After that clock cycle, Start is set to high. The clock starts high and rotates every 10 ps.

The way the testbench stops the process is if the out is greater than zero, this made it really easy to test by always having a value greater than zero in the accumulator. Storing and loading memory was tested by loading memory into the accumulator and also walking step by step through the instructions using the display outputs shown above.



1. **Concluding remarks. Analysis of the instruction set.**

There were many attempts to make this into a two cycle instruction only having the fetch and execute instructions, however, having to latch the instructions and having memory being read from the same place as the instructions cause problems for the memory and jump instructions. As mentioned, I could have added another state so that the start does not have to be held down to continue the program but this makes it easier to test, thus the halt does not go anywhere but end the processor indefinitely. Overall, though I was able to complete all the functions I feel there was some redundancy in the cycles I wish I had been able to delete.

1. **Appendix**
   1. Verilog files have been saved onto a shared drive