Parth Parth

CS232L

Fall 2023

**Project 7 Report**

**Setup**

A new project named 'cpu' was initiated in Quartus, involving the creation of ProgramRom and DataRAM entities using the MegaWizard Plugin Manager, and initializing them with .mif files.

**CPU Entity Definition**

A top-level VHDL file, **cpu.vhd**, was created, featuring a cpu entity with various inputs and outputs, including a clock, reset, input port, and output port.

**ALU Design**

An ALU was designed in VHDL (**alu.vhd**) with four condition bits (Zero, Arithmetic overflow, Negative, and Carry out) based on the operation result.

**CPU State Machine**

A 9-state machine was established for the CPU, starting from a startup state to a halt state, with a 3-bit internal counter for the pause in the start state.

**Internal Signals**

Numerous internal signals were defined, including signals for each register, ALU buses, condition flags, and various other components.

**Port Mapping**

Components like ProgramROM, DataRAM, and ALU were port-mapped into the CPU, ensuring correct connections for clocks, addresses, and data wires.

**CPU Implementation**

* **Reset Case**: Resets all registers and state to initial values.
* **Start State**: Increments a counter until moving to the fetch state.
* **Fetch State**: Transitions to execute-setup state after loading instructions.
* **Execute-Setup State**: Sets up instructions based on the type.
* **Execute-ALU State**: Engages RAM write operations for certain instructions.
* **Execute-MemWait State**: A passive state for memory read instructions.
* **Execute-Write State**: Finalizes instruction execution, updating registers and flags.
* **Execute-Return States**: Special states for handling RETURN instructions.

**Testing**

The CPU was tested using **cpubench.vhd** and **ghdl**, simulating the given program and additional tests for stack operations, function calls, and generating the Fibonacci sequence.

**Screenshots:**

alutestbench.vcd:

A screenshot of a computer

Description automatically generated

**cpubench.vcd**

program.mif:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

testpush.mif:

A screenshot of a computer

Description automatically generated

testcall.mif:

A screenshot of a computer

Description automatically generated

fibonnaci.mif:

A screenshot of a computer

Description automatically generated

I conducted thorough hardware testing to validate the functionality and accuracy of the designed circuit. This included scrutinizing the interactions among the ALU, CPU, memory (ROM and RAM), and the testbench setup.

**ALU Testing**

**Setup**: With alu.vhd, I initially confirmed elementary operations such as addition, subtraction, and logical functions (AND, OR, XOR, shifting, rotation). I fed specific values into srcA and srcB, then monitored the resulting output at dest and the condition flags in cr.

**Example Test - Addition**: I assigned "0000000000000101" (5 in decimal) to srcA and "0000000000000011" (3 in decimal) to srcB, setting the operation to "000" (addition). The outcome at dest was expected to be "0000000000001000" (8 in decimal), with cr(0) not indicating zero. The ALU successfully performed the addition, validating its functionality.

**CPU Testing**

**Setup**: Utilizing cpu.vhd, I examined the CPU's execution of instructions. This involved populating the program ROM with instructions, initializing the CPU's state, and tracking changes in registers and memory.

**Example Test - Load and Store**: I loaded a ROM instruction for a load operation, moving data from a RAM address to a register, followed by a store operation, transferring data from a register to another RAM address. I verified that the data in the targeted register and RAM address matched expected values.

**Integration Testing**

**Setup**: With cpubench.vhd, I combined the ALU, CPU, and memory for simulation, emulating the execution of multiple instructions in sequence.

**Example Test - Instruction Sequence**: I programmed a series of instructions in programROM.vhd, including arithmetic operations, data transfers between registers, and memory access. After computing expected outcomes, I compared the actual results with these predictions to affirm the integrated system's accuracy.

**Memory Testing**

**Setup**: I tested DataRAM.vhd and ProgramROM.vhd to confirm accurate data storage and retrieval.

**Example Test - Memory Read/Write**: I wrote and then read specific data to and from a RAM address. Similarly, I verified that data read from certain ROM addresses matched the contents of a loaded .mif file.

Testing with .mif Files

These files, including program.mif, testpush.mif, testcall.mif, and fibonacci.mif, enabled specific scenario testing, observing the CPU's interaction with the ALU and memory.

**Testing with program.mif**:

* **Setup**: Contained basic operations like data transfers and arithmetic functions.
* **Test Run**: After loading program.mif into the ROM, initial instructions set values in registers, followed by additions, subtractions, and a loop through a jump instruction.
* **Observations**: Register values aligned with expectations post-execution. For example, after addition at address 05, RA displayed the correct sum of RD and RE. The loop confirmed the CPU's jump instruction handling.

**Testing with testpush.mif**:

* **Setup**: Focused on testing stack operations (push and pop).
* **Test Execution**: Running testpush.mif in the ROM, instructions pushed values onto the stack and then popped them into registers.
* **Results**: Register values post-pop matched those initially pushed, confirming the stack pointer's functionality and the CPU's stack operation handling.

**Testing with testcall.mif**:

* **Setup**: Targeted testing of CALL and RETURN instructions.
* **Procedure**: After loading testcall.mif, the program executed a function call, performed operations, and returned to the main program.
* **Observations**: The CALL instruction properly shifted the program counter to the function's start. The RETURN instruction restored the counter after the function, validating the CPU's function call and return capabilities, including stack management.

**Testing with fibonacci.mif**:

* **Setup**: Created to generate the Fibonacci sequence, testing looping and arithmetic functions.
* **Execution**: Observing the CPU's loop execution post-loading fibonacci.mif.
* **Results**: The output port displayed the Fibonacci numbers correctly, demonstrating the CPU's proficiency in managing loops, conditional branches, and arithmetic operations.

These tests collectively assessed the CPU's capabilities in arithmetic, stack handling, function calls, looping, and branching. The outcomes, as seen in the output port, registers, and memory states, confirmed the integrated success of the CPU, ALU, and memory, adhering to the project's design specifications. Each test reinforced the design's reliability and compliance.