EE 108B Processor Datapath Design - LAB2

Charles Guan

Nipun Agarwala

11th February, 2014

INTRODUCTION:

In the given project, we modified mipstop.v to add functionality to the instruction_fetch.v,alu.v and decode.v . The instruction_fetch.v module incremented the PC as the instruction needed. The decode.v module decodes the instruction passed in to it. It decides what kind of instruction it is and accordingly changes the appropriate signals connecting to the other modules. The alu.v module computes the results of the two operands according to the instruction passed into it. The irom.v contains the instructions to be executed.

DESIGN:

The decode.v module receives the instruction and decodes it. It dissembles the instruction into the opcode, operands, immediate, shamt and function code. Then, if the opcode is a branch instruction, the module assigns the appropriate wires with the correct values. if the opcode is a jump instruction, it updates the PC counter by the offset as specified in the instruction. Otherwise, using a big mux, as suggested by the always block, the module assigns the right alu instruction to the input to the ALU. According to the instruction, the decode.v module updates the memory write, memory read, register write and jump_with_link wires.

The outputs of the decode.v module go to the instruction_fetch.v and alu.v modules. The instruction_fetch.v module updates the PC depending on whether the jump_en and/or branch_en is high. Otherwise it just increments the PC. The alu.v receives the operand values from decode.v and computes the result depending on the function sent from the latter. It outputs the result along with the overflow, if any.

RESULTS:

Three major components of the processor were completed, along with a testbench instruction set. The Instruction Fetch, Instruction Decoder, and Arithmetic and Logic Unit were implemented, and test cases were placed in the Instruction ROM. The Instruction Fetch structure was in place, utilizing a flip-flop to latch an instruction from the memory at each cycle. The instruction fetch module read the next instruction and fed the next pc value, taking this from pc + 4 or the appropriate jump or branch instructions once the decoder decided whether to take these instructions.

The Instruction Decoder was in charge of re-directing the datapath correctly and assigning the appropriate control values. Additionally, the Instruction Decoder evaluates comparisons for branch expressions. The ALU was also available to evaluate comparisons but is less flexible in its curent options.

The ALU module implements common functions such as addition, subtraction, shifting, boolean operations, and comparisons. As this was implemented in Verilog, the circuit fundamentals of each operation could be abstracted through built in Verilog compilation. Because of timing constraints, multiplication and division will be left to a pipelined processor implementation in the future.

In order to test the complete processor, a sample program was written in the Instruction ROM using binary encoding of ISA assembly commands. The program tested out each instruction given, including all common arithmetic and logical operations, as well as several jump and branch instructions necessary to implement functions and loops in higher-level languages. We initially ran into problems with writing the instructions in terms of bits. The need to zero out fields such as the shift amount for several functions was unusual after programming in assembly. After each of the instructions were tested, the display was tested by drawing 1000 blocks across the display. This is a small step towards running our game of pong on the FPGA hardware.

CONCLUSIONS:

A single-cycle processor implementing a fundamental but basic subset of the MIPS ISA was created and tested on an FPGA. The implementation was well-decomposed into an instruction fetch, decoder, register file, ALU, and memory modules. The implentation's greatest strength is its simplicity, but lacks stronger functions like multiplication and division. Additionally, while the branch instructions could be implemented using the ALU, the readibility is greatly improved by placing the functions within the decoder.