Mikhail Ivanov

Semestral Project Report

Simple CPU in Verilog

1. Goal

The goal of the semestral project is to develop a simple 32-bit processor in Verilog which has a separate instruction and data memory. The processor should use MIPS instruction set architecture and be able to execute following instructions: add, sub, or, slt, addi, lw, sw, beq, bne, sll, srl, jal and jr. Execution starts from the beginning of instruction memory (0x00000000).

2. Implementation

2.1 Source code

Source code is located in Github repository and available by following link: https://github.com/Misha91/PAP. The code is not public until the end of the semester.

2.2 Prerequisites

Prior to run the project following software should be installed:

-Python (tested on versions 2.7 and 3.6)

-make (tested on version 4.1)

-iverilog (tested on versions 10.3-1)

-gcc-mips-elf (tested on version 4.4.4-1_amd64)

-binutils-mips-elf (tested on version 2.20.51-1_amd64)

-GTKWave (tested on version 3.3.103)

2.3 Running routine

To run the processor it's supposed to call the python script in the root folder of the project by command "python runner.py" (python executable

can vary due to version). On call, the script will prompt to enter one of the modes:

0 – Test Mode

1 – GCD Execution Mode

2 – Exit

Any MIPS call will store the output dump in test.vcd file.

2.4 Execution modes

2.4.1 Test mode

In the test mode the script modifies code for the execution by walking through all defined test cases in "/code/test" directory. In every test case it modifies the assembly code for the execution by appropriate content based on *.S files, compile it, then it puts generated *.dat file into the root folder and calls MIPS processor for its execution. All the logs and outputs are stored in appropriate *.txt files which can be used for debug as well as for output correction checking.

Every test case is supposed to have expected results into *.res files to compare with program output. The script counts pass rate and prints it on the screen.

2.4.2 GCD Execution Mode

In Greatest Common Divisor (GCD) execution mode the script prompts user to enter two values for the GCD calculation. It can be any positive value that's width equal or less than 16 bits (<32767). The script will update source file for execution, compile and run the processor. It will take program output from expected register and will print the result on screen.

2.4.3 Exit

Entering this mode will terminate script execution. Alternatively, it could be terminated by Ctrl+C key combination.

3. Processor architecture

The processor is designed using classical MIPS architecture. The source of inspiration was book by Professor Emeritus David Harris and Sarah Harris "Digital Design and Computer Architecture", hence its structure and name convention correspond to the provided schematic (Fig.1).

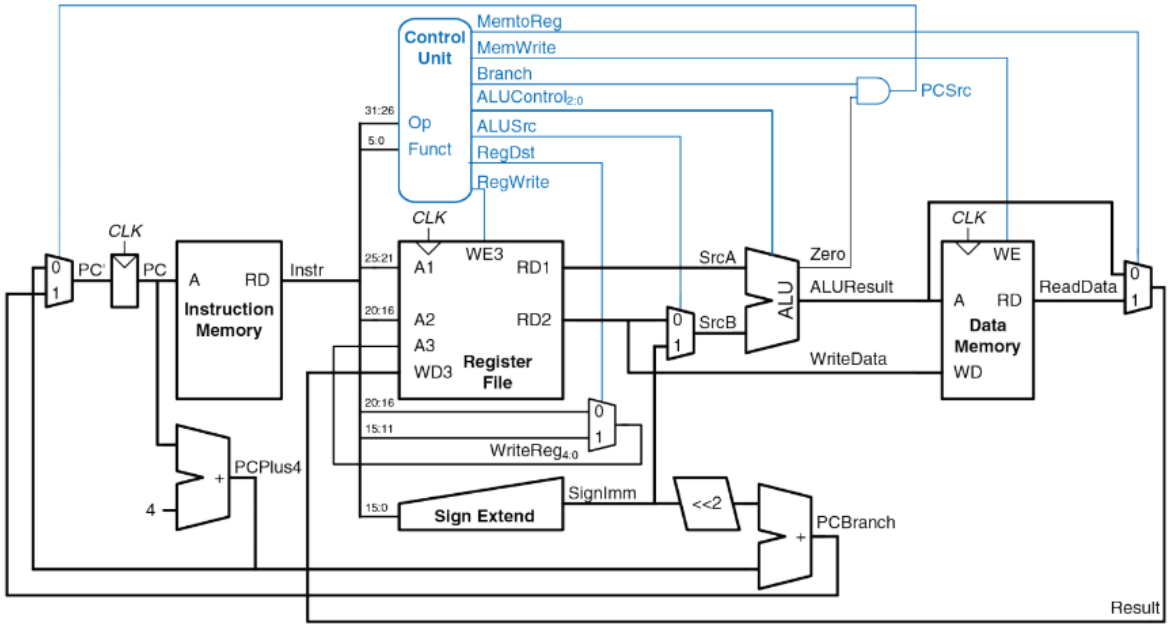

Fig.1. Processor schematic (source: "Digital Design and Computer Architecture", Professor Emeritus David Harris and Sarah Harris)

The final solution is consisted of following modules:

mips_tb – a testbench module for processor simulation. It toggles the clock (clk) and set initial program counter (PC) value.

mips – The processor module. Takes clk and PC as input. The module updates PC and uses imem and processor modules.

processor – Main module for control flow and execution. It has many sub-modules to pass and get variables, such as sign extension module sign_ext, data memory module dmem, register file reg_file, arithmetic-logical unit ALU, program counter update module pc_update and several multiplexers for flow control. On any event (always @(*)) processor sets control wires to initial state, then it goes through state machine (case-switch block) and set appropriate control wires depends on the input command.

ALU – the module performs all the arithmetic and logical commands and put its result on output line. The module can perform following commands – logical AND, logical OR, logical XOR, logic shift right and left, logical greater, arithmetic summation and arithmetic subtraction.

reg_file – the module stores register status. It has 32 registers of 32-bit width. Register #0 is always set to zero. Based on input lines it can write or

read data from the file. Address for reading is set by inputs A1 and A2 and output goes on output lines RD1 and RD2. Writing address is defined by A3, writing content – WD3, it's happen on positive edge of clk in case of input WE3 is set high.

sign_ext – module takes as input 16-bit value and extends it to 32-bit width.

dmem – module for storing data in special data memory. Data could be read or written.

imem – module to store program (instruction) memory. Data could be only read.

mux2 – 2:1 multiplexer (input and output are 32-bit width)

mux2_5 – 2:1 multiplexer (input and output are 5-bit width)

pc_update – module for program counter update. The new values depends on current instruction. In case of jr instruction (jump) it set by srcA input, in case of jal (relative jump) it's incremented by value set in offset plus 4, otherwise – incremented by 4.

m_and – helper module for PC source select.

Conclusions

In this work we learnt the principle of simple CPU implementation, its architecture and shortened MIPS instruction set. We also get familiar and practiced with Verilog, various aspects of the design for correct synthesis using HDLs and simulation. We also used GTKWave tool for debugging and processor state transition visualization.