RV32I Single cycle processor

Introduction:

This is a RISC-V single cycle 32-bit core on logic simulator called as LOGISIM. The circuit contains 32-bit ALU, 32-bit data bus, 16-KB RAM/ROM, 12-bit address bus for RAM. The register file contains 32 registers with data width equals 32 bits.

Methodology:

At first, it is required to learn the basic instructions of the RV32I Instruction Set Architecture and learn their functionality. To learn the backend working use Venus Online RV32I Simulator. This Simulator helps grasp the working behind the instruction much faster. On the Logic Simulator software, first start with the program counter and memory address register. Then develop the circuit for the immediate generation which uses full instruction and PC to generate respective immediate. After that, create register file with 32 registers each having 32-bit data width. This register file takes 5-bit address to select one of the 32 registers and write data to it using register enable wire. Two 5-bit address RS1 and RS2 are to read one of the 32 registers simultaneously. Now make 32 Bit ALU with 4-bit ALU operation select which selects the operation to be performed according to the instruction provided. After the completion, create type decoder which uses 7-bit opcode to decode the type of the instruction. Then in control Decoder, depending upon the type of instruction, function3 and function7 different components are controlled. Integrate type decode and control decode, and this will become your control unit. Add RAM and configure its data bits to 32bit and address width to 12 bits. To handle branch instruction Branch circuit is required to be created using the simple comparators and depending upon the RS1 and RS2, conditional jump is done if branch is true. In the end add a 32-bit adder for jalr as this instruction requires two additions. One ALU cannot perform 2 operation on a single cycle. Connect the wiring using the splitters, multiplexers, constants, tunnels, and clocks. To troubleshoot the circuit, start with the simpler instructions e.g. add, addi and watch the circuit behavior using temporary register outputs. To load the machine code on the Instruction Register, simulate the code on Venus then using dump feature copy the machine code and create .mem extension file on notepad.

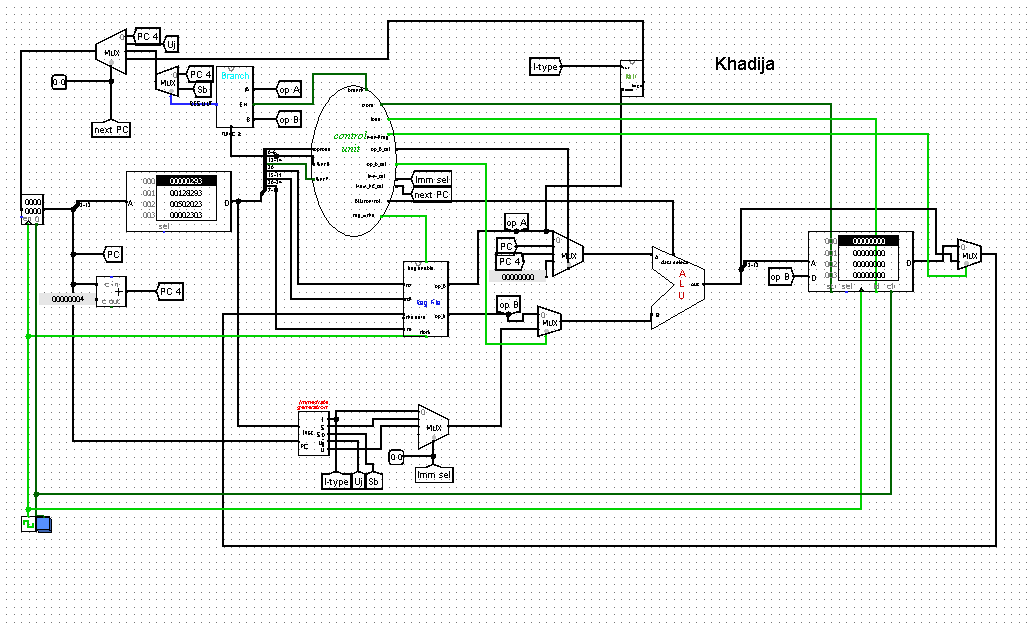
Implementation:

The circuit is implemented on a Logic Simulating software called as LOGISIM. All the different circuits/components are firstly developed and are then connected to design the top of the microprocessor. The circuit is required to be tested in order to see if it works properly or not, few codes are generated, may be simpler ones having two to three lines of instructions or may be a long code consisting of more than 9 to 10 lines of codes on online assembly language simulator called VENUS. The hex codes of these instructions are copied using the dump option on Venus. The same hex code is copied on a text file with header as v2.0 raw and saved in the format of filename.mem. The same file is loaded into the ROM by right clicking the ROM and using the load image option in there. It lets us load the hex codes of the instructions and allows us to see how the instruction is performing in the circuit.

Results:

The result that we can conclude after performing all this is that each instruction behaves differently from each other hence the entire circuitry would behave as the type of the instruction suggests. If, for instance, it is an add instruction which is type R having a 7 bit opcode, 5 bit destination register, function 3 of 3 bits, two registers each of 5 bits and a function 7 of 7 bits.

Verification:

******

This is the circuit, aligning all the sub circuits together leads us to this circuit which is called as the top of microprocessor.

Test program:

***Assembly Code:***

addi t3,zero,10

beq t0,t3,endt

addi t0,t0,1

addi t1,t1,9

jalr ra,x0,4

endt:

addi sp,sp,-4

sw ra,0x0(sp)

jal sub

lw ra,0x0(sp)

addi sp,sp,4

jal end

sub:

sub t3,t3,t1

ret

end:

***Hex*** ***Code:***

00a00e13

01c28863

00128293

00930313

004000e7

ffc10113

00112023

010000ef

00012083

00410113

00c000ef

406e0e33

00008067

Conclusion:

The circuit is made keeping in mind the instructions that we are supposed to execute using this microprocessor. Since, it is made on RISC-V open source environment, hence all the instructions are from the same environment. Quite randomly, the circuit is tested using the instruction codes and now it executes properly.

References:

Computer Organization and design By David A. Peterson, John H. Hennesy(RISC-V edition)

RISC-V instruction set manual(Volume I)