32 bit five stage pipelined RISC V RV32I Core

About: This RISC – V Core executes 27 instructions from the RV32I Set. It is in behavioural description.

Languages used: System Verilog

Instructions Implemented

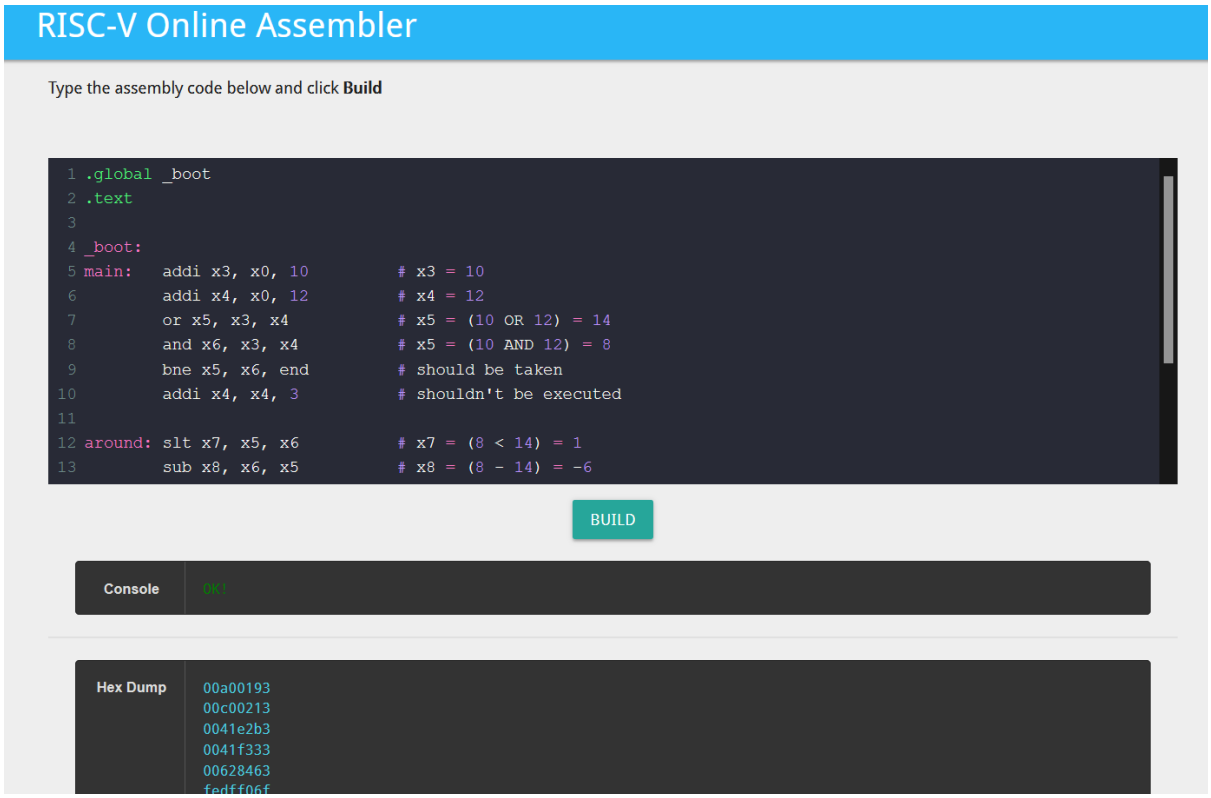
Instruction implemented as of now (**Highlighted)**

Instruction	Description
`lb rd, imm(rs1)`	load byte
`lh rd, imm(rs1)`	load half
`lw rd, imm(rs1)`	load word
`lbu rd, imm(rs1)`	load byte unsigned
`lhu rd, imm(rs1)`	load half unsigned
`addi rd, rs1, imm`	add immediate
`slli rd, rs1, uimm`	shift left logical immediate
`slti rd, rs1, imm`	set less than immediate
`sltiu rd, rs1, imm`	set less than imm. unsigned
`xori rd, rs1, imm`	xor immediate
`srli rd, rs1, uimm`	shift right logical immediate
`srai rd, rs1, uimm`	shift right arithmetic imm.
`ori rd, rs1, imm`	or immediate
`andi rd, rs1, imm`	and immediate
`auipc rd, upimm`	add upper immediate to PC
`sb rs2, imm(rs1)`	store byte
`sh rs2, imm(rs1)`	store half
`sw rs2, imm(rs1)`	store word
`add rd, rs1, rs2`	add
`sub rd, rs1, rs2`	sub
`sll rd, rs1, rs2`	shift left logical
`slt rd, rs1, rs2`	set less than
`sltu rd, rs1, rs2`	set less than unsigned
`xor rd, rs1, rs2`	xor
`srl rd, rs1, rs2`	shift right logical
`sra rd, rs1, rs2`	shift right arithmetic
`or rd, rs1, rs2`	or
`and rd, rs1, rs2`	and
`lui rd, upimm`	load upper immediate
`beq rs1, rs2, label`	branch if =
`bne rs1, rs2, label`	branch if ≠
`blt rs1, rs2, label`	branch if <
`bge rs1, rs2, label`	branch if ≥
`bltu rs1, rs2, label`	branch if < unsigned
`bgeu rs1, rs2, label`	branch if ≥ unsigned
`jalr rd, rs1, imm`	jump and link register
`jal rd, label`	jump and link

32 bit five stage pipelined RISC V RV32I Core

Steps to execute the Instructions:

1. First, we write the instructions in assembly using the above table and convert it to hex file.
   Use this site: https://riscvasm.lucasteske.dev/#



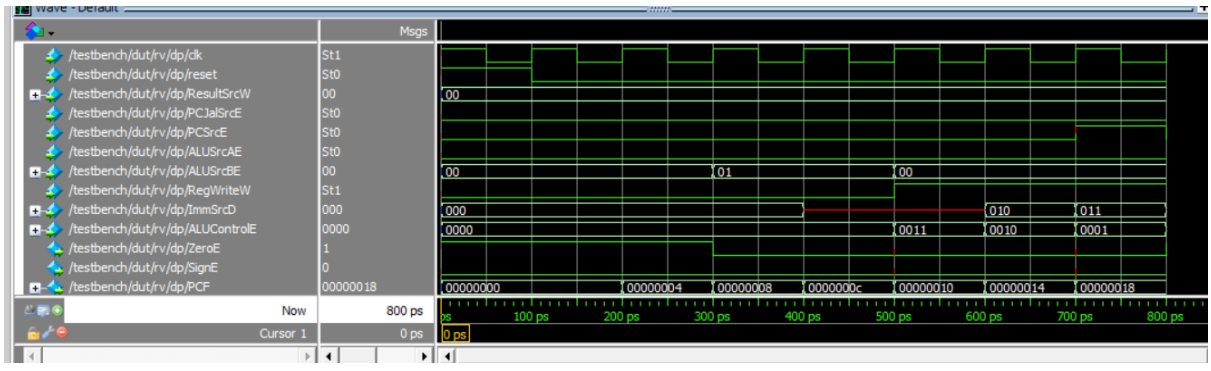
2. Convert the Hex dump into **byte - addressable format** using the python script. (Available in the folder)

```
// Instruction: 00a00193
RAM[0] = 8'h93;
RAM[1] = 8'h01;
RAM[2] = 8'ha0;
RAM[3] = 8'h00;
// Instruction: 00c00213
RAM[4] = 8'h13;
RAM[5] = 8'h02;
RAM[6] = 8'hc0;
RAM[7] = 8'h00;
// Instruction: 0041e2b3
RAM[8] = 8'hb3;
RAM[9] = 8'he2;
RAM[10] = 8'h41;
RAM[11] = 8'h00;
// Instruction: 0041f333
RAM[12] = 8'h33;
```

3. Copy the output into imem.sv – **instruction memory** file.

```
/*
   Name: Instruction Memory
*/


module imem(input logic [31:0] a, output logic [31:0] rd);

logic [7:0] RAM[128:0]; // 128 x 8 = byte addressable memory with 128 locations

assign rd = {RAM[a+3], RAM[a+2], RAM[a+1], RAM[a+0]};

// follow little-endian: LSB corresponds to lowest order memory address
initial
   begin
      // Instruction: 00a00193
      RAM[0] = 8'h93;
      RAM[1] = 8'h01;
      RAM[2] = 8'ha0;
      RAM[3] = 8'h00;
      // Instruction: 00c00213
      RAM[4] = 8'h13;
      RAM[5] = 8'h02;
      RAM[6] = 8'hc0;
      RAM[7] = 8'h00;
      // Instruction: 0041e2b3
      RAM[8] = 8'hb3;
      RAM[9] = 8'he2;
      RAM[10] = 8'h41;
      RAM[11] = 8'h00;
      // Instruction: 0041f333
      RAM[12] = 8'h33;
      RAM[13] = 8'hf3;
```

4. Set testbench as top module and start analysis and elaboration. Use ModelSim to verify the waveforms.



5. The value of ALU is shown in the transcript every clock cycle. Cross check this with the assembly file.