ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HCM

BỘ MÔN ĐIỆN TỬ

EE3043: Computer Architecture

Laboratory Report

GVHD: Dr. Trần Hoàng Linh

TA: Cao Xuân Hải

Lớp: L01

Thành viên nhóm	MSSV
Trần Thế Nhân	2114274
Nguyễn Đắc Tâm	2114714
Nguyễn Đăng Khoa	2111527

Mục lục

Milestone 2

Design of a Single-Cycle Processor

1. Introduction

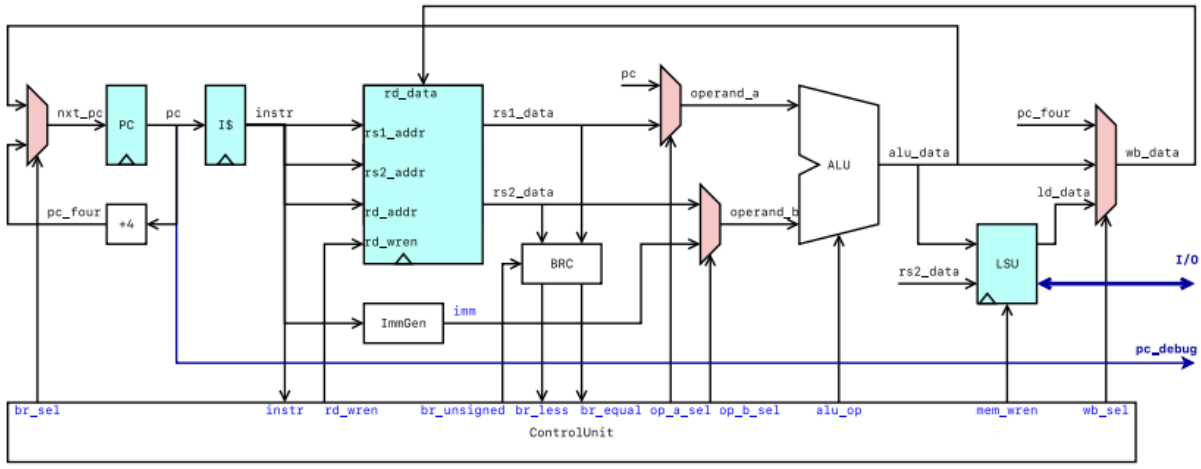
RISC-V instruction set architecture (ISA) là một kiến trúc có mã nguồn mở dùng để thiết kế vi xử lý cho máy tính và đã trở nên phổ biến cho những ứng dụng trong công nghiệp và học thuật. Nó dựa trên nguyên tắc Reduced Instruction Set Computer (RISC) với mục đích đơn giản hóa tập lệnh và cải thiện tính hiệu quả cũng như hiệu năng cho vi xử lý. RV32I là những tập lệnh xử lý số nguyên nằm trong kiến trúc RISC-V và được thiết kế để triển khai số 32 bit. Trong báo cáo, chúng em sẽ trình bày thiết kế của vi xử lý RV32I single-cycle. Trong thiết kế, hầu hết lệnh trong tập lệnh RV32I được thực hiện, ngoại trừ những lệnh như ecall, ebreak, lb, lh, lbu, lhu, sb, sh. Ngoài ra chúng em có thực hiện thêm lệnh mul để phục vụ cho một số ứng dụng cụ thể.

Trong báo cáo, những khối cơ bản trong vi xử lý single-cycle RV32I sẽ được trình bày chi tiết ở phần 2. Những cách kiểm tra các khối cơ bản và tập lệnh được nêu ra ở phần 3. Phần thiết kế mở rộng, đánh giá và kết luận sẽ được mô tả lần lượt ở phần 4, 5, 6.

Để hoàn thành được thiết kế, chúng em xin cảm ơn thầy Trần Hoàng Linh đã dạy những kiến thức về cấu trúc máy tính và xin cảm ơn anh Cao Xuân Hải đã hướng dẫn cụ thể cách thực hiện thiết kế cũng như những kiến thức liên quan.

2. Design Strategy

Những khối cơ bản để tạo nên vi xử lý được mô tả như hình dưới:



Hình 2.1. Vi xử lý single-cycle tiêu chuẩn

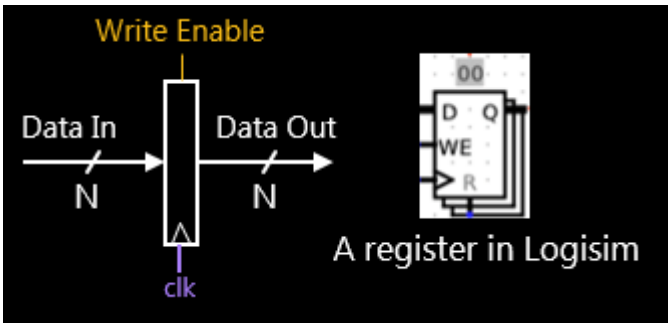
Thiết kế từng khối riêng lẻ bao gồm:

- Khối mux: từ sơ đồ khối cần thiết kế 2 loại mux là mux 2 to 1 và mux 3 to 1.
- Khối adder PC4: thiết kế bộ cộng 32 bit để có thể sử dụng cho bộ ALU.
- Khối PC: bộ đếm chương trình (program counter).
- Khối IMEM: lưu trữ những instruction, khởi tạo bộ nhớ 8KB và chứa lệnh $readmem để đọc dữ liệu từ file hex chứa mã của chương trình.
- Khối Regfile: bộ những thanh ghi của vi xử lý, thiết kế 32 thanh ghi 32 bit và có thể đọc ghi vào thanh ghi.
- Khối ImmGen: (immediate generator) khối tạo số từ instruction.
- Khối BRC: (branch comparator) bộ so sánh để thực hiện những lệnh rẽ nhánh.
- Khối ALU: bộ tính toán, thực hiện các phép toán như cộng, trừ, dịch, … Trước khi thiết kế bộ ALU, cần thiết kế bộ cộng 32 bit, trừ 32 bit, and 32 bit, or 32bit, xor 32 bit, shift left logical, shift right logical, shift right arithmetic, set less than, set less than unsigned.
- Khối LSU: bộ nhớ lưu trữ dữ liệu và giao tiếp với IO ngoại vi.
- Khối ControlUnit: bộ điều khiển các tín hiệu cho vi xử lý.

Sau khi thiết kế từng khối riêng lẻ, tổng hợp các khối như hình 2.1 tạo thành khối singlecycle. Sau đó mô phỏng RTL và kiểm tra lại các đường dây nối ở các khối với nhau.

2.1. Program Counter

Program counter là bộ đếm chương trình, về cơ bản thì program counter là một thanh ghi 32 bit. Ở mỗi cạnh lên xung clock, giá trị thanh ghi sẽ được cập nhật.



Hình 2.2. Program Counter

Ngõ vào:

- Bus dữ liệu ngõ vào 32 bit.
- Xung clock.

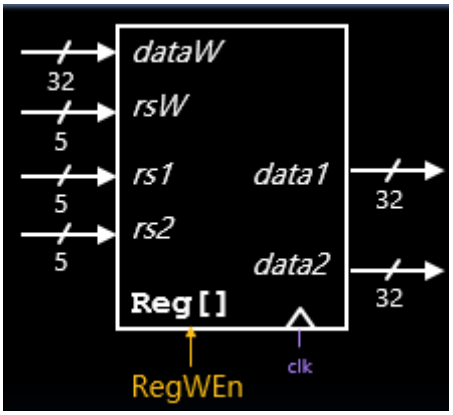
Ngõ ra:

- Bus dữ liệu ngõ ra 32 bit.

Hoạt động: ở mỗi cạnh lên xung clock, Data Out = Data In, ở những thời điểm khác Data Out sẽ không thay đổi và giữ nguyên giá trị trước đó.

2.2. Register File

Register file có 32 thanh ghi 32 bit.



Hình 2.3. Khối regfile

Ngõ vào:

- 1 bus ngõ vào 32 bit dataW chứa dữ liệu được ghi vào thanh ghi.
- 3 busses 5 bit rs1, rs2, rsW lần lượt chứa địa chỉ của 2 thanh ghi cần đọc và 1 thanh ghi cần ghi.

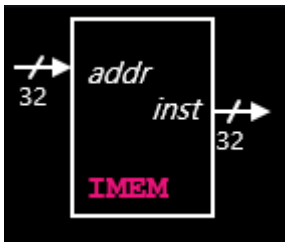
Ngõ ra:

- 2 busses ngõ ra 32 bit data1 và data2 chứa dữ liệu của 2 thanh ghi được đọc.

Hoạt động: Những thanh ghi sẽ được truy cập thông qua 5 bit địa chỉ, data1 = R[rs1], data2 = R[rs2], R[rsW] = dataW chỉ khi tín hiệu RegWEn = 1. Hoạt động ghi chỉ được thực hiện khi có cạnh lên xung clock, còn hoạt động đọc thanh ghi diễn ra như một hệ tổ hợp.

2.3. IMEM

Instruction memory là bộ nhớ chứa những lệnh (instruction). Trong thiết kế, imem có dung lượng 8KB.



Hình 2.4. IMEM block

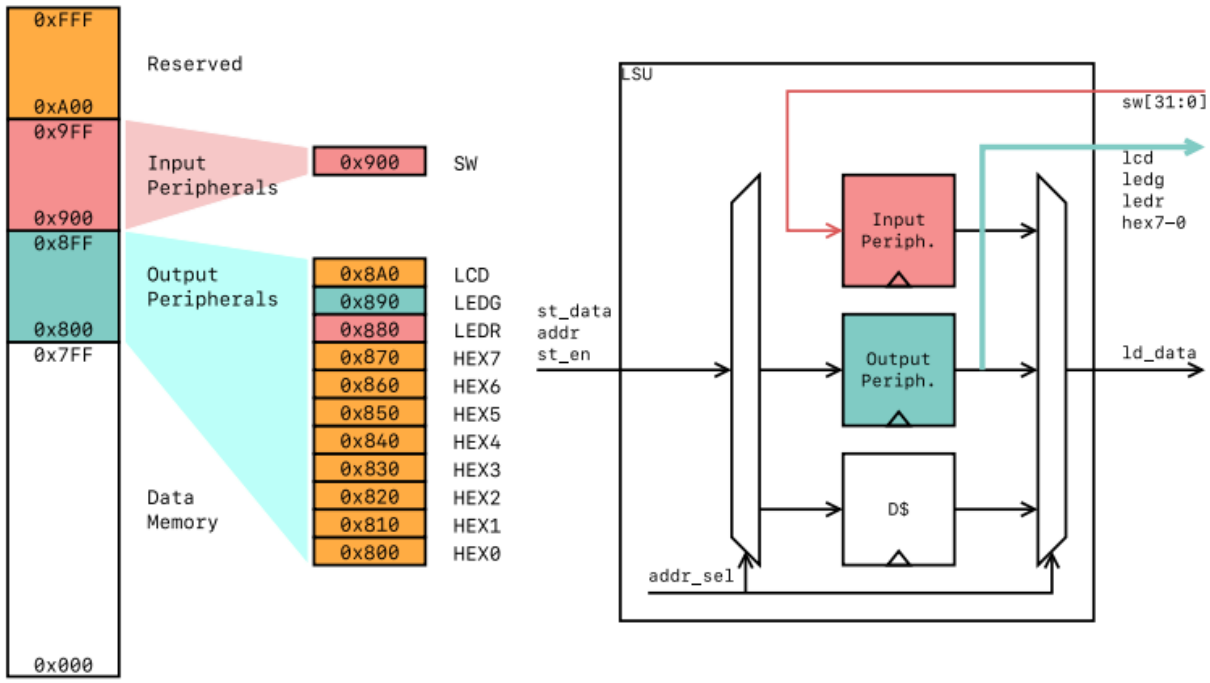
Ngõ vào: 32 bit addr, chứa địa chỉ của lệnh trong bộ nhớ.

Ngõ ra: 32 bit inst, chứa dữ liệu của lệnh.

Hoạt động: imem hoạt động như hệ tổ hợp, inst = mem[addr].

2.4. LSU

LSU (load – store unit) là bộ nhớ chứa dữ liệu được thiết kế có dung lượng 2KB, và được giao tiếp với ngoại vi bên ngoài.



Hình 2.5. Memory-mapping và LSU diagram

Ngõ vào:

- Xung clock: positive clock.
- rst_ni: tín hiệu low negative reset.
- 32 bit addr chứa địa chỉ đọc và ghi.
- 32 bit dataRW chứa dữ liệu đọc và ghi.
- MemRW tín hiệu chọn đọc hoặc ghi của bộ nhớ.
- 32 bit io_sw chứa 32 bit ngõ vào từ switch.

Ngõ ra:

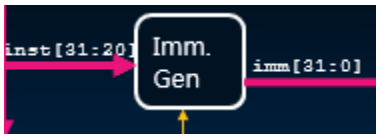
- 32 bit dataR chứa dữ liệu đọc từ bộ nhớ.
- 32 bit io_ledr chứa dữ liệu xuất ra ngoại vi red led.
- 32 bit io_ledg chứa dữ liệu xuất ra ngoại vi green led.
- 32 bit io_lcd chứa dữ liệu điều khiển ngoại vi lcd.

- 8 32-bit io_hex chứa 8 32-bit dữ liệu xuất ra ngoại vi led 7 đoạn.

Hoạt động: Ở cạnh lên của xung clock, bộ nhớ sẽ lưu giá trị mem[addr]=dataW khi tín hiệu MemRW được tích cực lên. Ở mọi thời điểm, dataR = mem[addr] và các giá trị ngoại vi sẽ được gán với từng vùng nhớ cụ thể.

2.5. ImmGen

Immediate generator là bộ tạo số immediate từ instruction.



Hình 2.6. ImmGen block

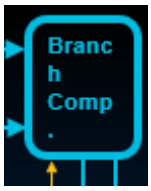
Ngõ vào: 25 bit inst[31:7] từ 32 bit instruction. 3 bit ImmSel là tín hiệu chọn loại lệnh (I, S, B, J, U).

Ngõ ra: 32 bit imm chứa dữ liệu số immediate.

Hoạt động: bộ ImmGen hoạt động như bộ tổ hợp, với mỗi tín hiệu ImmSel cụ thể thì sẽ tạo được immediate number từ ngõ vào instruction.

2.6. BRC

Branch comparator là bộ so sánh phục vụ cho những lệnh rẽ nhánh.



Hình 2.7. BRC block

Ngõ vào:

- 32 bit rst1 chứa dữ liệu của thanh ghi thứ nhất được lấy từ khối register file.
- 32 bit rst2 chứa dữ liệu của thanh ghi thứ hai được lấy từ khối register file.
- 1 bit BrUn là tín hiệu báo hiệu lệnh rẽ nhánh so sánh số không dấu.

Ngõ ra: 1 bit BrEq là tín hiệu báo hiệu 2 thanh ghi bằng nhau, 1 bit BrLt là tín hiệu báo hiệu thanh ghi thứ nhất bé hơn thanh ghi thứ hai.

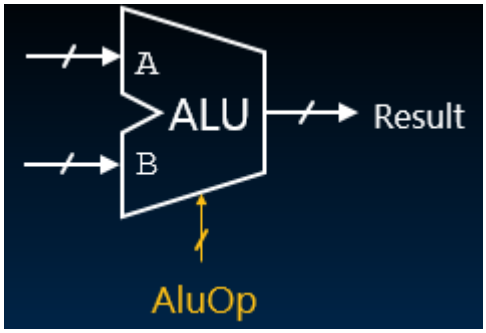
Hoạt động: Nếu BrUn = 1 thì BRC sẽ so sánh 2 thanh ghi dưới dạng số không dấu. Bộ BRC sẽ so sánh 2 thanh ghi, nếu rst1 = rst2 thì BrEq = 1 và BrLt = 0, nếu rst1 < rst2 thì BrEq = 0 và BrLt = 1, với các trường hợp còn lại thì BrLt và BrEq đều bằng 0.

2.7. ALU

Arithmetic logic unit là bộ tính toán các biểu thức +, -, <<, >>, >>>, ^, &, |.

alu_op	Description (R-type)	Description (I-type)
ADD	$rd = rs1 + rs2$	$rd = rs1 + imm$
SUB	$rd = rs1 - rs2$	n/a
SLT	$rd = (rs1 < rs2)?1 : 0$	$rd = (rs1 < imm)?1 : 0$
SLTU	$rd = (rs1 < rs2)?1 : 0$	$rd = (rs1 < imm)?1 : 0$
XOR	$rd = rs1 \oplus rs2$	$rd = rs1 \oplus imm$
OR	$rd = rs1 \vee rs2$	$rd = rs1 \vee imm$
AND	$rd = rs1 \wedge rs2$	$rd = rs1 \wedge imm$
SLL	$rd = rs1 << rs2[4:0]$	$rd = rs1 << imm[4:0]$
SRL	$rd = rs1 >> rs2[4:0]$	$rd = rs1 >> imm[4:0]$
SRA	$rd = rs1 >>> rs2[4:0]$	$rd = rs1 >>> imm[4:0]$

Hình 2.8. Các phép toán bộ ALU thực hiện



Hình 2.9. ALU block

Ngõ vào:

- 32 bit rst1 chứa dữ liệu thanh ghi thứ nhất từ regfile hoặc giá trị pc.
- 32 bit rst2 chứa dữ liệu thanh ghi thứ hai từ regfile hoặc giá trị từ ImmGen.

- 3 bit AluOp là tín hiệu lựa chọn phép tính cần thực hiện.

Ngõ ra:

- 32 bit Result chứa kết quả của phép toán được thực hiện.

Hoạt động: Với mỗi tín hiệu AluOp, các phép toán như cộng, trừ, dịch phải, dịch trái,… sẽ được thực hiện như hình 2.8 và kết quả được lưu vào Result.

2.8. CtrlUnit

Control Unit là khối điều khiển các tín hiệu lựa chọn cho các khối khác hoạt động.

Hình 2.10. Control Unit block

Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
sub	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
addi	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
lw	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
sw	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
beq	0	*	+4	B	*	PC	Imm	Add	Read	0	*
beq	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	1	*	+4	B	*	PC	Imm	Add	Read	0	*
blt	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
bltu	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
jalr	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
jal	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
auipc	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

Hình 2.11. Control Unit truth table

Ngõ vào: 32 bit inst chứa dữ liệu instruction từ IMEM. 1 bit BrEq là tín hiệu từ BRC, bằng 1 khi A = B. 1 bit BrLt là tín hiêu từ BRC, bằng 1 khi A < B.

Ngõ ra:

- 1 bit PCSel là tín hiệu chọn PC: bằng 0 if PC+4, 1 nếu được tính từ ALU.
- 1 bit BrUn là tín hiệu báo hiệu số không dấu, bằng 1 khi ở lệnh rẽ nhánh có 2 thông số so sánh là số không dấu.
- 1 bit WBSel là tín hiệu báo ghi dữ liệu vào regfile.
- 1 bit MemWEn là tín hiệu báo ghi dữ liệu vào LSU.
- 1 bit ASel là tín hiệu cho bộ mux, 0 nếu rs1, 1 nếu PC.
- 1 bit BSel là tín hiệu chọ bộ mux, 0 nếu rs2, 1 nếu imm.
- 4 bit ALUSel chọn phép tính cho ALU.
- 3 bit ImmSel chọn loại câu lệnh để tạo số immediate.

3. Verification Strategy

Verification sẽ được chia làm 2 loại: thứ nhất là viết testbench cho từng khối trong cpu và những module liên quan, thứ 2 là kiểm tra hoạt động của cpu với từng lệnh với các loại R, I, U, B, J, S.

Trước tiên cần test những khối tạo thành cpu như PC, ALU, Regfile,… Sau đó test từng câu lệnh theo từng loại.

Kiểm tra những khối tạo thành cpu:

Cần test những module adder_32bit, subtractor_32bit, or_32bit, and_32bit, xor_32bit, set_less_than, set_less_than_unsigned, shift_left_logical, shift_right_logical, shift_right_arithmetic. Sau đó mới test tới bộ ALU.

Có những module cần test như sau:

- adder_32bit: cộng 2 số 32 bit
- subtractor_32bit: trừ 2 số 32 bit
- and_32bit: thực hiện and từng bit trong 2 số 32 bit
- or_32bit: thực hiện or từng bit trong 2 số 32 bit
- xor_32bit: thực hiện xor từng bit trong 2 số 32 bit
- mux2to1_32bit: chọn 1 trong 2 số 32 bit
- mux3to1_32bit: chọn 1 trong 3 số 32 bit
- set_less_than: ngõ ra bằng 1 khi A < B, bằng 0 khi A >= B, A và B là 2 số 32 bit có dấu
- set_less_than_unsign: ngõ ra bằng 1 khi A < B, bằng 0 khi A >= B, A và B là 2 số 32 bit không dấu
- shift_left_logical: dịch trái logical số thứ nhất n lần, với n bằng số thứ hai.
- shift_right_logical: dịch phải logical số thứ nhất n lần, với n bằng số thứ hai
- shift_right_arithmetic: dich phải arithmetic số thứ nhất n lần, với n bằng số thứ hai.
- pc: ngõ ra bằng ngõ vào khi xung clock có cạnh lên.
- imem: inst = mem[addr], với addr là địa chỉ ngõ vào 32 bit, inst là dữ liệu instruction 32 bit.
- regfile: ngõ ra 32 bit data1 = R[rs1], data2 = R[rs2], khi RegWEn = 1 thì R[rsW] = dataW ở canh lên xung clock. Khi rst = 0 thì tất cả thanh ghi được xóa về 0.
- imm_gen
    - Nếu ImmSel = 000, imm[31:0] = {{21{inst[31]}}, inst[30:20]}
    - Nếu ImmSel = 001, imm[31:0] = {{21{inst_i[31]}}, inst_i[30:25], inst_i[11:7]}
    - Nếu ImmSel = 010, imm[31:0] = {{20{inst_i[31]}}, inst_i[7], inst_i[30:25], inst_i[11:8], 1'b0}
    - Nếu ImmSel = 011, imm[31:0] = {{12{inst_i[31]}}, inst_i[19:12], inst_i[20], inst_i[30:21], 1'b0}
    - Nếu ImmSel = 100, imm[31:0] = {inst_i[31:12], {12{1'b0}}} : {32{1'b0}}

- brcomp: Nếu BrUn = 1 thì brcomp sẽ so sánh 2 thanh ghi dưới dạng số không dấu. Bộ brcomp sẽ so sánh 2 thanh ghi, nếu rst1 = rst2 thì BrEq = 1 và BrLt = 0, nếu rst1 < rst2 thì BrEq = 0 và BrLt = 1, với các trường hợp còn lại thì BrLt và BrEq đều bằng 0
- alu
    - Nếu AluSel = 0000, Result = rs1 + rs2
    - Nếu AluSel = 1000, Result = rs1 – rs2
    - Nếu AluSel = 0001, Result = rs1 << rs2
    - Nếu AluSel = 0010, Result = (rs1 < rs2) ? 1 : 0, số có dấu
    - Nếu AluSel = 0011, Result = (rs1 < rs2) ? 1 : 0, số không dấu
    - Nếu AluSel = 0100, Result = rs1 ^ rs2
    - Nếu AluSel = 0101, Result = rs1 >> rs2
    - Nếu AluSel = 1101, Result = rs1 >>> rs2
    - Nếu AluSel = 0110, Result = rs1 | rs2
    - Nếu AluSel = 0111, Result = rs1 & rs2
    - Nếu AluSel = 1111, Result = rs2
    - Các trường hợp còn lại Result = 0.
- ctrl_unit:
    - Ngõ vào: 32 bit inst chứa dữ liệu instruction từ IMEM. 1 bit BrEq là tín hiệu từ BRC, bằng 1 khi A = B. 1 bit BrLt là tín hiêu từ BRC, bằng 1 khi A < B.
    - Ngõ ra:
        - 1 bit PCSel là tín hiệu chọn PC: bằng 0 if PC+4, 1 nếu được tính từ ALU.
        - 1 bit BrUn là tín hiệu báo hiệu số không dấu, bằng 1 khi ở lệnh rẽ nhánh có 2 thông số so sánh là số không dấu.
        - 1 bit WBSel là tín hiệu báo ghi dữ liệu vào regfile.
        - 1 bit MemWEn là tín hiệu báo ghi dữ liệu vào LSU.
        - 1 bit ASel là tín hiệu cho bộ mux, 0 nếu rs1, 1 nếu PC.
        - 1 bit BSel là tín hiệu chọ bộ mux, 0 nếu rs2, 1 nếu imm.
        - 4 bit ALUSel chọn phép tính cho ALU.
        - 3 bit ImmSel chọn loại câu lệnh để tạo số immediate.
    - Các tín hiệu được mô tả như hình dưới

Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
sub	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
addi	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
lw	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
sw	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
beq	0	*	+4	B	*	PC	Imm	Add	Read	0	*
beq	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	1	*	+4	B	*	PC	Imm	Add	Read	0	*
blt	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
bltu	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
jalr	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
jal	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
auipc	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

Kiểm tra tất cả câu lệnh được thiết kế:

Có những lệnh cần được kiểm tra như sau:

- add
- sub
- xor
- or
- and
- sll
- srl
- sra
- slt
- sltu
- addi
- xori
- ori
- andi
- slli
- srli
- srai
- slti
- sltiu
- lw
- sw
- jal
- jalr
- lui

- auipc

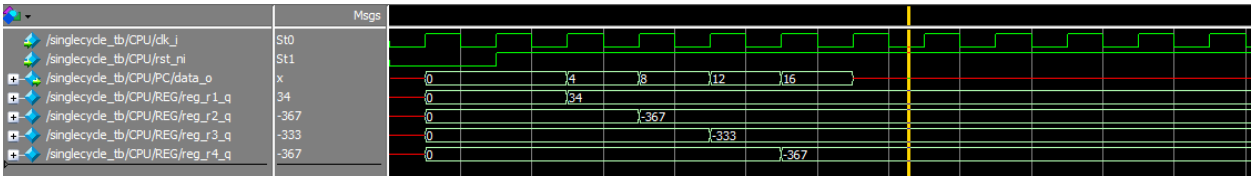
Thực hiện chạy waveform cho các lệnh ở trên:

Kiểm tra lệnh addi, and, sub

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x02200093	addi x1 x0 34	addi x1, x0, 34
0x4	0xE9100113	addi x2 x0 -367	addi x2, x0, -367
0x8	0x002081B3	add x3 x1 x2	add x3, x1, x2
0xc	0x40118233	sub x4 x3 x1	sub x4, x3, x1

Waveform:



Giải thích:

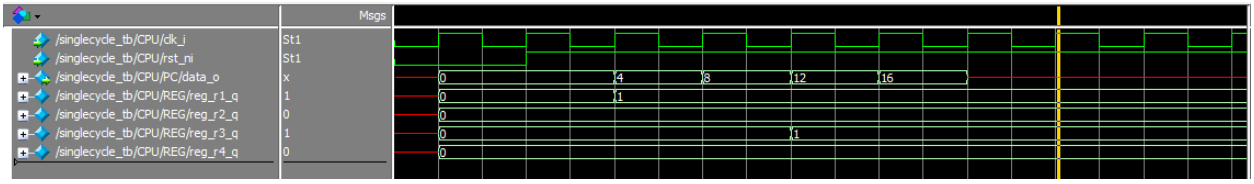
- Câu lệnh addi x1, x0, 34 thực hiện gán x1 = x0 + 34 =34
- Câu lệnh addi x2, x0, -367 thực hiện gán x2 =x0 - 367 = -367
- Câu lệnh addi x3, x1, x2 thực hiện gán x3 = x1 + x2 = 34 - 367 = -333
- Câu lệnh sub x4, x3, x1 thực hiện gán x4 = x3 – x1 = -333 – 34 = -367
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x1 được gán giá trị 34, ở cạnh lên xung clock thứ 3 x2 được gán giá trị - 367, ở cạnh lên thứ 4 x3 được gán giá trị -333, ở cạnh lên xung clock thứ 5 x4 được gán giá trị -367.

Kiểm tra lệnh xor:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00100093	addi x1 x0 1	addi x1, x0, 1
0x4	0x00000113	addi x2 x0 0	addi x2, x0, 0
0x8	0x0020C1B3	xor x3 x1 x2	xor x3, x1, x2
0xc	0x0011C233	xor x4 x3 x1	xor x4, x3, x1

Waveform:



Giải thích:

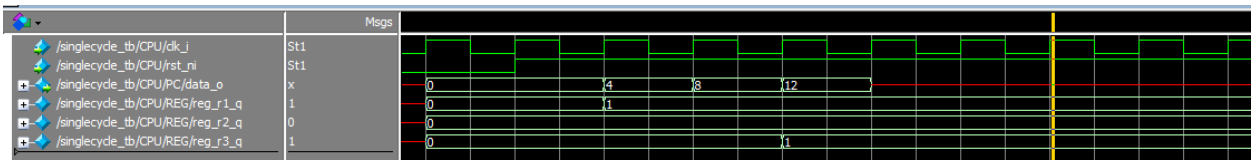
- Câu lệnh addi x1, x0, 1 thực hiện gán x1 = x0 + 1 = 1
- Câu lệnh addi x2, x0, 0 thực hiện gán x2 = x0 + 0 = 0
- Câu lệnh xor x3, x1, x2 thực hiện gán x3 = x2 ^ x1 = 0 ^ 1 = 1
- Câu lệnh xor x4, x3, x1 thực hiện gán x4 = x3 ^ x1 = 1 ^ 1 = 0
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x1 được gán giá trị 1, ở cạnh lên xung clock thứ 3 x2 được gán giá trị 0, ở cạnh lên thứ 4 x3 được gán giá trị 1, ở cạnh lên xung clock thứ 5 x4 được gán giá trị 0.
-

Kiểm tra lệnh or:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00100093	addi x1 x0 1	addi x1, x0, 1
0x4	0x00000113	addi x2 x0 0	addi x2, x0, 0
0x8	0x0020E1B3	or x3 x1 x2	or x3, x1, x2

Waveform:



Giải thích:

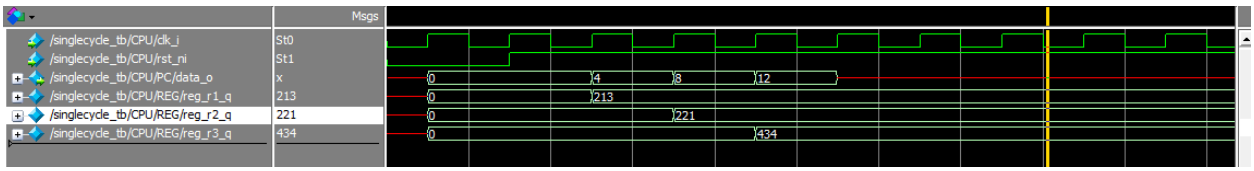
- Câu lệnh addi x1, x0, 1 thực hiện gán x1 = x0 + 1 = 1
- Câu lệnh addi x2, x0, 0 thực hiện gán x2 = x0 + 0 = 0
- Câu lệnh or x3, x1, x2 thực hiện gán x3 = x1 | x2
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x1 được gán giá trị 1, ở cạnh lên xung clock thứ 3 x2 được gán giá trị 0, ở cạnh lên thứ 4 x3 được gán giá trị 1.

Kiểm tra lệnh add:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x0D500093	addi x1 x0 213	addi x1, x0, 213
0x4	0x0DD00113	addi x2 x0 221	addi x2, x0, 221
0x8	0x002081B3	add x3 x1 x2	add x3, x1, x2

Waveform:



Giải thích:

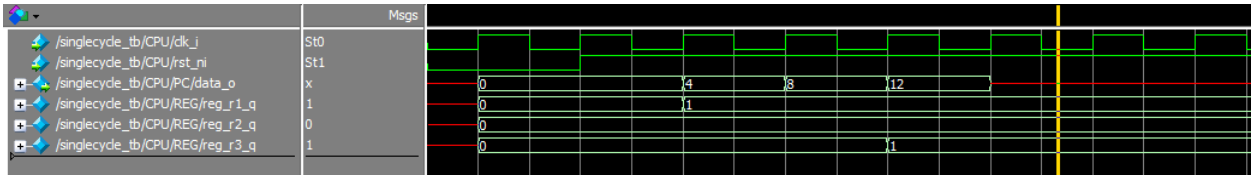
- Câu lệnh addi x1, x0, 213 gán giá trị x1 = x0 + 213
- Câu lệnh addi x2, x0, 211 gán giá trị x2= x0 + 211
- Câu lệnh add x3, x2, x1 gán giá trị x3 = x2 + x1 = 434
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x1 được gán giá trị 213, ở cạnh lên xung clock thứ 3 x2 được gán giá trị 221, ở cạnh lên thứ 4 x3 được gán giá trị 434.

Kiểm tra lệnh xori, ori, andi

Mã assembly

PC	Machine Code	Basic Code	Original Code
0x0	0x00104093	xori x1 x0 1	xori x1,x0,1
0x4	0x00006113	ori x2 x0 0	ori x2, x0,0
0x8	0x0010F193	andi x3 x1 1	andi x3, x1,1

Waveform:



Giải thích

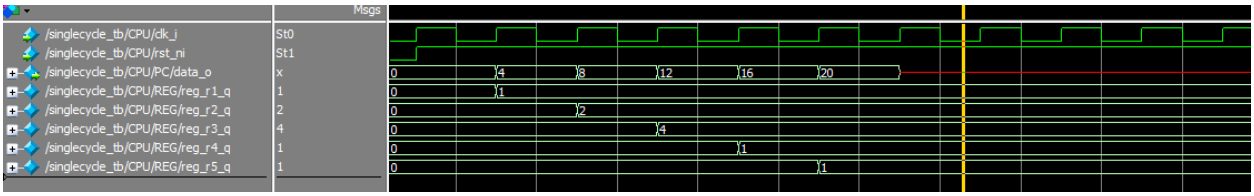
- Câu lệnh addi x1, x0, 213 gán giá trị x1 = 0 ^ 1 = 1
- Câu lệnh addi x2, x0, 211 gán giá trị x2= x0 | 0 = 0
- Câu lệnh add x3, x2, x1 gán giá trị x3 = x1 & 1 = 1
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x1 được gán giá trị 1, ở cạnh lên xung clock thứ 3 x2 được gán giá trị 0, ở cạnh lên thứ 4 x3 được gán giá trị 1.

Kiểm tra lệnh shift left logical, shight right logical, shift right arith:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00100093	addi x1 x0 1	addi x1, x0, 1
0x4	0x00200113	addi x2 x0 2	addi x2, x0, 2
0x8	0x002091B3	sll x3 x1 x2	sll x3, x1, x2
0xc	0x001151B3	srl x3 x2 x1	srl x3, x2, x1
0x10	0x40115233	sra x4 x2 x1	sra x4, x2 , x1

Waveform:



Giải thích:

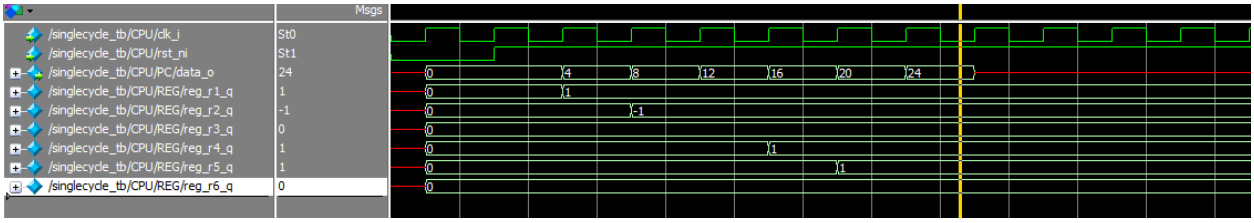
- Câu lệnh addi x1, x0, 1 gán giá trị x1 = x0 + 1 = 1
- Câu lệnh addi x2, x0, 2 gán giá trị x2 = x0 + 2 = 2
- Câu lệnh sll x3, x1, x2 gán giá trị x3 = x1 << x2 -> 1 dịch sang trái 2 thành 100bin = 4 dec, x3 = 4
- Câu lệnh srl x4, x2, x1 gán giá trị x4 = x2 >> x1 -> 2 dịch sang phải 1 thành 1bin = 1dec, x4 = 1
- Câu lệnh sra x5, x2, x1 gán giá trị x5 = x2 >> x1 -> 2 dịch sang phải thành 1bin = 1 dec, x5 = 1
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x1 được gán giá trị 1, ở cạnh lên xung clock thứ 3 x2 được gán giá trị 2, ở cạnh lên thứ 4 x3 được gán giá trị 4, ở cạnh lên xung clock thứ 5 x4 được gán giá trị 1, ở cạnh lên thứ 6 x5 được gán giá trị 1.

Kiểm tra câu lệnh set less than, set less than (u), set less than imm, set less than imm (u):

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00100093	addi x1 x0 1	addi x1, x0, 1
0x4	0xFFF00113	addi x2 x0 -1	addi x2, x0, -1
0x8	0x0020A1B3	slt x3 x1 x2	slt x3, x1, x2
0xc	0x0020B233	sltu x4 x1 x2	sltu x4, x1, x2
0x10	0x00112293	slti x5 x2 1	slti x5, x2 , 1
0x14	0x00113313	sltiu x6 x2 1	sltiu x6, x2, 1

Waveform:



Giải thích:

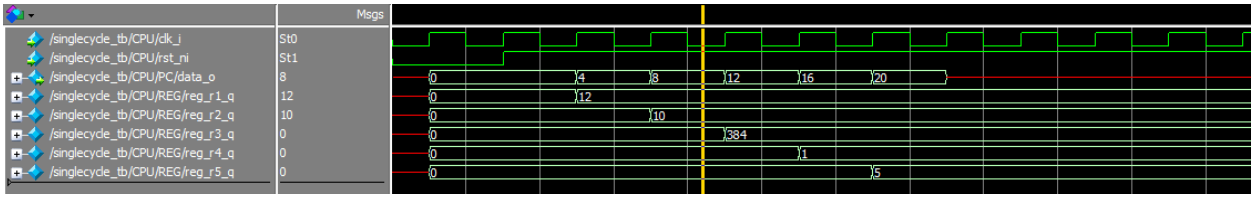
- Câu lệnh addi x1, x0, 1 gán giá trị x1 = x0 + 1 = 1
- Câu lệnh addi x2, x0, -1 gán giá trị x2 = x0 – 1 = -1
- Câu lệnh slt x3, x1, x2 gán giá trị x3 = (x1 < x2)?1:0 (so sánh giá trị có dấu trong thanh ghi r1 và r2, lưu lại kết quả vô r3, 1 nếu r1 < r2, 0 nếu r1 > r2)
- Câu lệnh sltu x4, x1, x2 gán giá trị x4 = (x1 < x2)?1:0 (so sánh giá trị không dấu trong thanh ghi r1 và r2, lưu lại kết quả vô r4, 1 nếu r1 < r2, 0 nếu r1 > r2)
- Câu lệnh slti x5, x2, 1 gán giá trị x5 = (x2 < 1)?1:0 (so sánh giá trị có dấu trong thanh ghi r2 và 1, lưu lại kết quả vô r5, 1 nếu r2 < 1, 0 nếu r2 > 1)
- Câu lệnh sltiu x5, x2, 1 gán giá trị x5 = (x2 < 1)?1:0 (so sánh giá trụ không dấu trong thanh ghi r2 và 1 lưu lại kết quả vô r5, 1 nếu r2 < 1, 0 nếu r2 > 1)
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x1 được gán giá trị 1, ở cạnh lên xung clock thứ 3 x2 được gán giá trị -1, ở cạnh lên thứ 4 x3 được gán giá trị 0, ở cạnh lên xung clock thứ 5 x4 được gán giá trị 1, ở cạnh lên thứ 6 x5 được gán giá trị 1, ở cạnh lên thứ 7 x6 được gán giá trị 0.

Kiểm tra câu lệnh shift left logical imm, shift right logical imm, shift right arith imm:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00C00093	addi x1 x0 12	addi x1, x0, 12
0x4	0x00A00113	addi x2 x0 10	addi x2, x0, 10
0x8	0x00509193	slli x3 x1 5	slli x3, x1, 5
0xc	0x0030D213	srli x4 x1 3	srli x4, x1, 3
0x10	0x40115293	srai x5 x2 1	srai x5, x2 , 1

Waveform:



Giải thích:

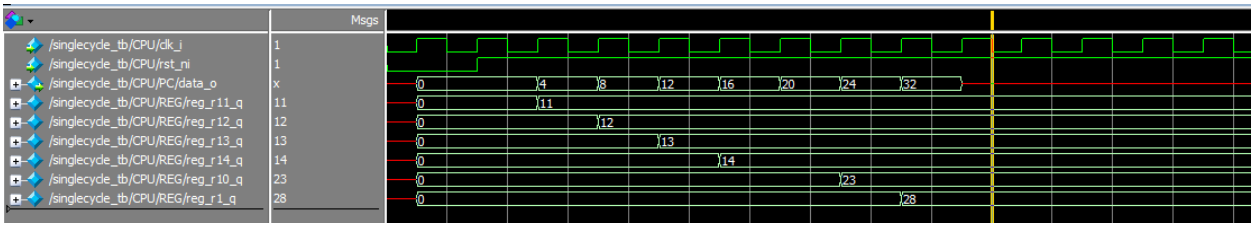
- Câu lệnh addi x1, x0, 1 gán giá trị x1 = x0 + 12 = 12
- Câu lệnh addi x2, x0, 2 gán giá trị x2 = x0 + 10 = 10
- Câu lệnh slli x3, x1, 5 gán giá trị x3 = x1 << 5 -> 1100 dịch sang trái 5 thành 11000000bin = 384 dec, x3 = 384
- Câu lệnh srli x4, x1, 3 gán giá trị x4 = x1 >> 3 -> 1100 dịch sang phải 3 thành 1bin = 1dec, x4 = 1
- Câu lệnh srai x5, x2, 1 gán giá trị x5 = x2 >> 1 -> 1100 dịch sang phải thành 110bin = 5dec, x5 = 5
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x1 được gán giá trị 12, ở cạnh lên xung clock thứ 3 x2 được gán giá trị 10, ở cạnh lên thứ 4 x3 được gán giá trị 384, ở cạnh lên xung clock thứ 5 x4 được gán giá trị 1, ở cạnh lên thứ 6 x5 được gán giá trị 5.

Kiểm tra lệnh branch equal, jump and link:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00B00593	addi x11 x0 11	addi x11, x0, 11
0x4	0x00C00613	addi x12 x0 12	addi x12, x0, 12
0x8	0x00D00693	addi x13 x0 13	addi x13, x0, 13
0xc	0x00E00713	addi x14 x0 14	addi x14, x0, 14
0x10	0x00E68663	beq x13 x14 12	beq x13,x14,Else
0x14	0x00C58533	add x10 x11 x12	add x10,x11,x12
0x18	0x008000EF	jal x1 8	jal x1, Exit
0x1c	0x40C58533	sub x10 x11 x12	Else:sub x10,x11,x12

Waveform:



Giải thích:

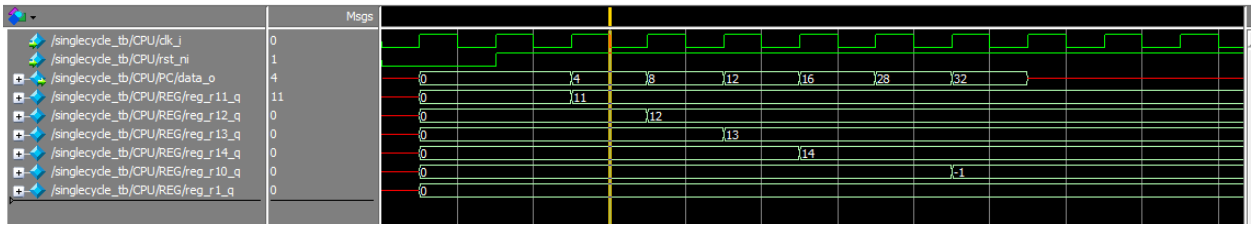
- Câu lệnh addi x11, x0, gán giá trị x11 = x0 + 11 = 11
- Câu lệnh addi x12, x0, gán giá trị x12 = x0 + 12 = 12
- Câu lệnh addi x11, x0, gán giá trị x13 = x0 + 13 = 13
- Câu lệnh addi x11, x0, gán giá trị x14 = x0 + 14 = 14
- Câu lệnh beq x13, x14, Else so sánh giá trị x13 và x14, nếu x13 != x14 thì thực hiên gán x10 = x11 + x12 rồi thanh ghi x1 = PC + 4 và nhảy tới nhãn Exit; nếu x13 = x14 thì nhảy tới nhãn Else thực hiện gán x10 = x11 – x12. Ở đây x13 != x14 nên x10 = x11 + x12 = 11 + 12 = 23 sau đó thanh ghi x1 = PC = PC + 4 = 24 + 4 = 28 rồi nhảy tới nhãn Exit.
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x11 được gán giá trị 11, ở cạnh lên xung clock thứ 3 x12 được gán giá trị 12, ở cạnh lên thứ 4 x13 được gán giá trị 13, ở cạnh lên xung clock thứ 5 x14 được gán giá trị 14, ở cạnh lên thứ 6 thực hiện so sánh, ở cạnh lên thứ 7 x10 được gán giá trị 23, ở cạnh lên thứ 8, x1 được gán giá trị 28.

Kiểm tra lệnh branch not equal:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00B00593	addi x11 x0 11	addi x11, x0, 11
0x4	0x00C00613	addi x12 x0 12	addi x12, x0, 12
0x8	0x00D00693	addi x13 x0 13	addi x13, x0, 13
0xc	0x00E00713	addi x14 x0 14	addi x14, x0, 14
0x10	0x00E69663	bne x13 x14 12	bne x13,x14,Else
0x14	0x00C58533	add x10 x11 x12	add x10,x11,x12
0x18	0x008000EF	jal x1 8	jal x1, Exit
0x1c	0x40C58533	sub x10 x11 x12	Else:sub x10,x11,x12

Waveform:



Giải thích:

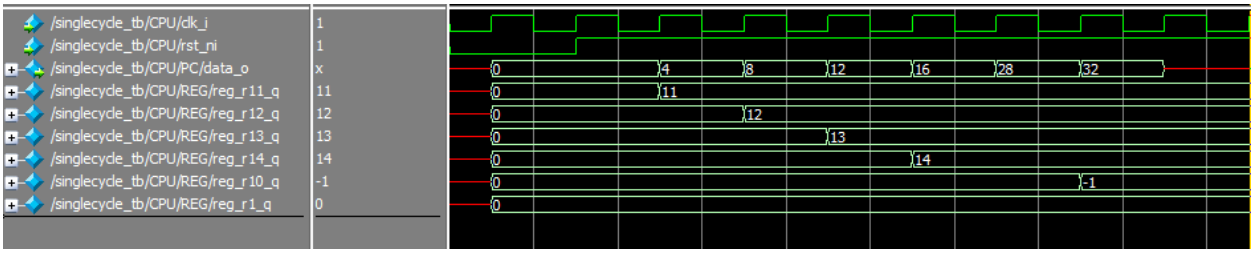
- Câu lệnh addi x11, x0, gán giá trị x11 = x0 + 11 = 11
- Câu lệnh addi x12, x0, gán giá trị x12 = x0 + 12 = 12
- Câu lệnh addi x11, x0, gán giá trị x13 = x0 + 13 = 13
- Câu lệnh addi x11, x0, gán giá trị x14 = x0 + 14 = 14
- Câu lệnh bne x13, x14, Else so sánh giá trị x13 và x14, nếu x13 != x14 thì thực hiện gán x10 = x11 + x12 rồi thanh ghi x1 = PC + 4 và nhảy tới nhãn Exit; nếu x13 = x14 thì nhảy tới nhãn Else thực hiện gán x10 = x11 – x12. Ở đây x13 != x14 nên x10 = x11 - x12 = 11 - 12 = -1.
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x11 được gán giá trị 11, ở cạnh lên xung clock thứ 3 x12 được gán giá trị 12, ở cạnh lên thứ 4 x13 được gán giá trị 13, ở cạnh lên xung clock thứ 5 x14 được gán giá trị 14, ở cạnh lên thứ 6 thực hiện so sánh, ở cạnh lên thứ 7 x10 được gán giá trị -1.

Kiểm tra lệnh branch less than:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00B00593	addi x11 x0 11	addi x11, x0, 11
0x4	0x00C00613	addi x12 x0 12	addi x12, x0, 12
0x8	0x00D00693	addi x13 x0 13	addi x13, x0, 13
0xc	0x00E00713	addi x14 x0 14	addi x14, x0, 14
0x10	0x00E6C663	blt x13 x14 12	blt x13,x14,Else
0x14	0x00C58533	add x10 x11 x12	add x10,x11,x12
0x18	0x008000EF	jal x1 8	jal x1, Exit
0x1c	0x40C58533	sub x10 x11 x12	Else:sub x10,x11,x12

Waveform:



Giải thích:

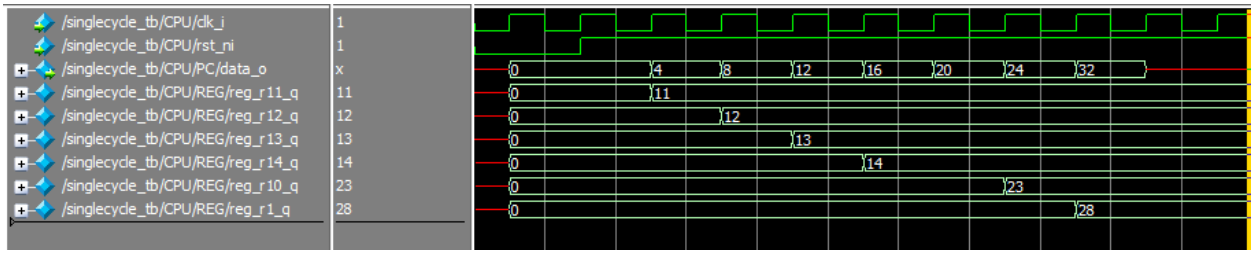
- Câu lệnh addi x11, x0, gán giá trị x11 = x0 + 11 = 11
- Câu lệnh addi x12, x0, gán giá trị x12 = x0 + 12 = 12
- Câu lệnh addi x11, x0, gán giá trị x13 = x0 + 13 = 13
- Câu lệnh addi x11, x0, gán giá trị x14 = x0 + 14 = 14
- Câu lệnh bne x13, x14, Else so sánh giá trị x13 và x14, nếu x13 > x14 thì thực hiện gán x10 = x11 + x12 rồi thanh ghi x1 = PC + 4 và nhảy tới nhãn Exit; nếu x13 < x14 thì nhảy tới nhãn Else thực hiện gán x10 = x11 – x12. Ở đây x13 < x14 nên x10 = x11 - x12 = 11 - 12 = -1.
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x11 được gán giá trị 11, ở cạnh lên xung clock thứ 3 x12 được gán giá trị 12, ở cạnh lên thứ 4 x13 được gán giá trị 13, ở cạnh lên xung clock thứ 5 x14 được gán giá trị 14, ở cạnh lên thứ 6 thực hiện so sánh, ở cạnh lên thứ 7 x10 được gán giá trị -1.

Kiểm tra lệnh branch greater than:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00B00593	addi x11 x0 11	addi x11, x0, 11
0x4	0x00C00613	addi x12 x0 12	addi x12, x0, 12
0x8	0x00D00693	addi x13 x0 13	addi x13, x0, 13
0xc	0x00E00713	addi x14 x0 14	addi x14, x0, 14
0x10	0x00E6D663	bge x13 x14 12	bge x13,x14,Else
0x14	0x00C58533	add x10 x11 x12	add x10,x11,x12
0x18	0x008000EF	jal x1 8	jal x1, Exit
0x1c	0x40C58533	sub x10 x11 x12	Else:sub x10,x11,x12

Waveform:



Giải thích:

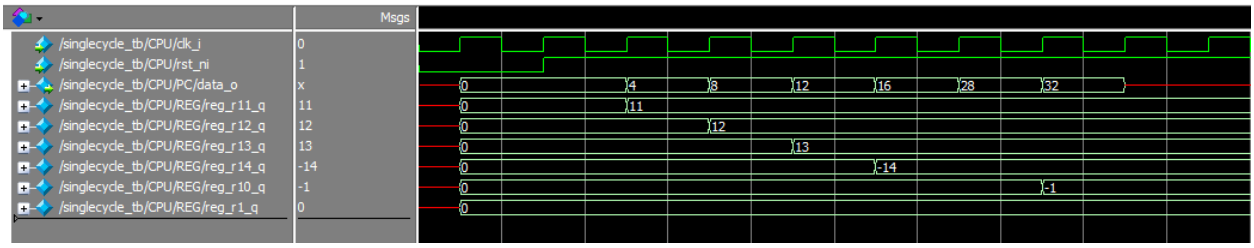
- Câu lệnh addi x11, x0, gán giá trị x11 = x0 + 11 = 11
- Câu lệnh addi x12, x0, gán giá trị x12 = x0 + 12 = 12
- Câu lệnh addi x11, x0, gán giá trị x13 = x0 + 13 = 13
- Câu lệnh addi x11, x0, gán giá trị x14 = x0 + 14 = 14
- Câu lệnh bne x13, x14, Else so sánh giá trị x13 và x14, nếu x13 < x14 thì thực hiện gán x10 = x11 + x12 rồi thanh ghi x1 = PC + 4 và nhảy tới nhãn Exit; nếu x13 >= x14 thì nhảy tới nhãn Else thực hiện gán x10 = x11 – x12. Ở đây x13 < x14 nên x10 = x11 + x12 = 11 + 12 = 23.
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x11 được gán giá trị 11, ở cạnh lên xung clock thứ 3 x12 được gán giá trị 12, ở cạnh lên thứ 4 x13 được gán giá trị 13, ở cạnh lên xung clock thứ 5 x14 được gán giá trị 14, ở cạnh lên thứ 6 thực hiện so sánh, ở cạnh lên thứ 7 x10 được gán giá trị 23, ở cạnh lên thứ 8 x1 được gán giá trị 28.

Kiểm tra lệnh branch less than usigned:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00B00593	addi x11 x0 11	addi x11, x0, 11
0x4	0x00C00613	addi x12 x0 12	addi x12, x0, 12
0x8	0x00D00693	addi x13 x0 13	addi x13, x0, 13
0xc	0xFF200713	addi x14 x0 -14	addi x14, x0, -14
0x10	0x00E6E663	bltu x13 x14 12	bltu x13,x14,Else
0x14	0x00C58533	add x10 x11 x12	add x10,x11,x12
0x18	0x008000EF	jal x1 8	jal x1, Exit
0x1c	0x40C58533	sub x10 x11 x12	Else:sub x10,x11,x12

Waveform:



Giải thích:

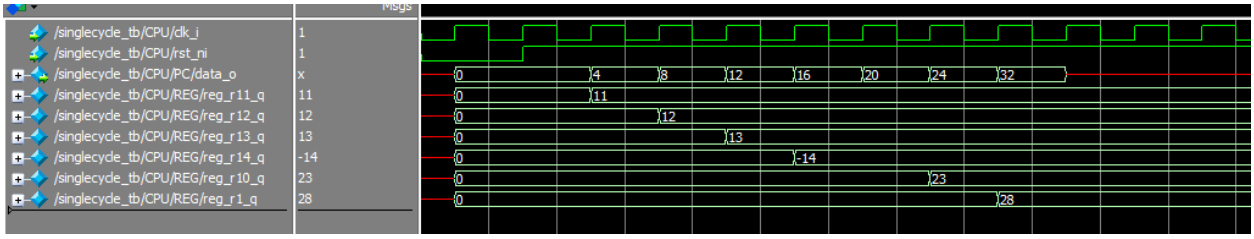
- Câu lệnh addi x11, x0, gán giá trị x11 = x0 + 11 = 11
- Câu lệnh addi x12, x0, gán giá trị x12 = x0 + 12 = 12
- Câu lệnh addi x11, x0, gán giá trị x13 = x0 + 13 = 13
- Câu lệnh addi x11, x0, gán giá trị x14 = x0 + 14 = -14
- Câu lệnh bne x13, x14, Else so sánh giá trị không dấu x13 và x14, nếu x13 > x14 thì thực hiện gán x10 = x11 + x12 rồi thanh ghi x1 = PC + 4 và nhảy tới nhãn Exit; nếu x13 < x14 thì nhảy tới nhãn Else thực hiện gán x10 = x11 – x12. Ở đây x13 < x14 nên x10 = x11 - x12 = 11 - 12 = -1.
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x11 được gán giá trị 11, ở cạnh lên xung clock thứ 3 x12 được gán giá trị 12, ở cạnh lên thứ 4 x13 được gán giá trị 13, ở cạnh lên xung clock thứ 5 x14 được gán giá trị 14, ở cạnh lên thứ 6 thực hiện so sánh, ở cạnh lên thứ 7 x10 được gán giá trị -1.

Kiểm tra lệnh branch greater than unsigned:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00B00593	addi x11 x0 11	addi x11, x0, 11
0x4	0x00C00613	addi x12 x0 12	addi x12, x0, 12
0x8	0x00D00693	addi x13 x0 13	addi x13, x0, 13
0xc	0xFF200713	addi x14 x0 -14	addi x14, x0, -14
0x10	0x00E6F663	bgeu x13 x14 12	bgeu x13,x14,Else
0x14	0x00C58533	add x10 x11 x12	add x10,x11,x12
0x18	0x008000EF	jal x1 8	jal x1, Exit
0x1c	0x40C58533	sub x10 x11 x12	Else:sub x10,x11,x12

Waveform:



Giải thích:

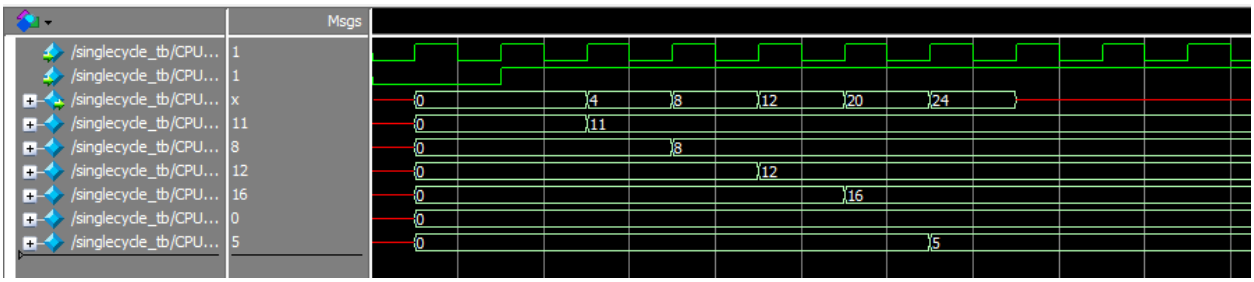
- Câu lệnh addi x11, x0, gán giá trị x11 = x0 + 11 = 11
- Câu lệnh addi x12, x0, gán giá trị x12 = x0 + 12 = 12
- Câu lệnh addi x11, x0, gán giá trị x13 = x0 + 13 = 13
- Câu lệnh addi x11, x0, gán giá trị x14 = x0 + 14 = -14
- Câu lệnh bne x13, x14, Else so sánh giá trị không dấu x13 và x14, nếu x13 < x14 thì thực hiện gán x10 = x11 + x12 rồi thanh ghi x1 = PC + 4 và nhảy tới nhãn Exit; nếu x13 >= x14 thì nhảy tới nhãn Else thực hiện gán x10 = x11 – x12. Ở đây x13 < x14 nên x10 = x11 + x12 = 11 + 12 = 23.
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x11 được gán giá trị 11, ở cạnh lên xung clock thứ 3 x12 được gán giá trị 12, ở cạnh lên thứ 4 x13 được gán giá trị 13, ở cạnh lên xung clock thứ 5 x14 được gán giá trị 14, ở cạnh lên thứ 6 thực hiện so sánh, ở cạnh lên thứ 7 x10 được gán giá trị 23, ở cạnh lên thứ 8 x1 được gán giá trị 28.

Kiểm tra lệnh jump and link register:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x00B00593	addi x11 x0 11	addi x11, x0, 11
0x4	0x00800613	addi x12 x0 8	addi x12, x0, 8
0x8	0x00C00113	addi x2 x0 12	addi x2, x0, 12
0xc	0x008100E7	jalr x1 x2 8	jalr x1, x2, 8
0x10	0xFF200713	addi x14 x0 -14	addi x14, x0, -14
0x14	0x00500193	addi x3 x0 5	addi x3, x0, 5

Waveform:



Giải thích:

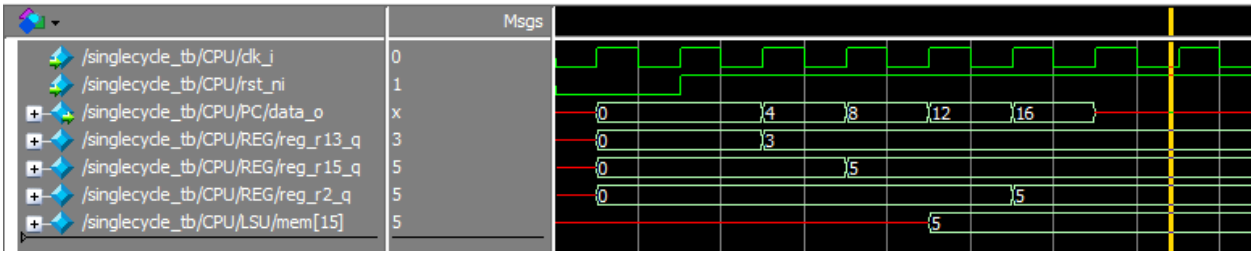
- Câu lệnh addi x11, x0, 11 gán x11 = x0 + 11 = 11
- Câu lệnh addi x12, x0, 8 gán x12 = x0 + 8 = 8
- Câu lệnh addi x2, x0, 12 gán x2 = x0 + 12 = 12
- Câu lệnh jalr x1, x2, 8 thực hiện gán x1 = PC + 4 = 14 + 4 = 16 và nhảy đến địa chỉ PC = x2 + 8 = 12 + 12 = 20 và sau đó thực hiện câu lệnh ở PC = 20 là addi x3, x0, 5 gán x3 = 5 và bỏ qua câu lệnh addi x14, x0, -14
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x11 được gán giá trị 11, ở cạnh lên xung clock thứ 3 x12 được gán giá trị 8, ở cạnh lên thứ 4 x2 được gán giá trị 12, ở cạnh lên xung clock thứ 5 x1 được gán giá trị 16, ở cạnh lên thứ 6 x3 được gán giá trị 5.

Kiểm tra lệnh load word, store word:

Mã assembly:

```
addi x13, x0, 3
addi x15, x0, 5
sw x15, 12(x13)
lw x2, 12(x13)
```

Waveform:



Giải thích:

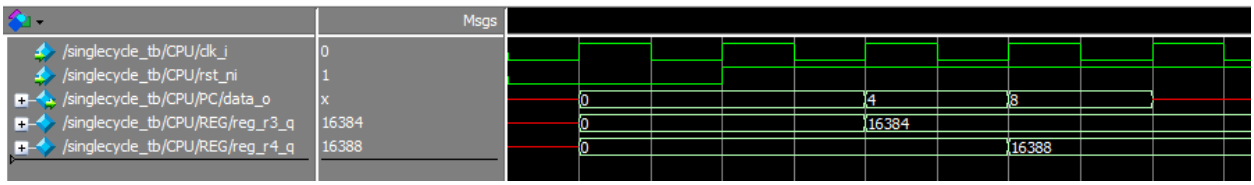
- Câu lệnh addi x13, x0, 3 gán x10 = x0 + 3 = 3
- Câu lệnh addi x15, x0, 5 gán x15 = x0 + 5
- Câu lệnh sw x15, 12(x13) lưu giá trị địa chỉ 12 + x13 = 12 + 3 = 15 vào x15 = 5
- Câu lệnh lw x2, 12(x13) lấy giá trị từ địa chỉ 12 + x13 = 12 + 3 = 15 vào x2 = 5
- Từ waveform, sau mỗi lần tăng pc, mỗi câu lệnh được thực hiện lần lượt ở cạnh lên xung clock thứ 2 x13 được gán giá trị 3, ở cạnh lên xung clock thứ 3, x15 được gán giá trị 5, ở cạnh lên thứ 4 địa chỉ 15 được gán giá trị 5, ở cạnh lên xung clock thứ 5, x2 được gán giá trị 5 từ địa chỉ 15.

Kiểm tra lệnh add upper imm to PC, load upper imm to PC:

Mã assembly:

PC	Machine Code	Basic Code	Original Code
0x0	0x000041B7	lui x3 4	lui x3, 4
0x4	0x00004217	auipc x4 4	auipc x4, 4

Waveform:



Giải thích:

- Câu lệnh lui x3, 4 thực hiện gán giá trị x3 = 10bin << 12= 16384
- Câu lệnh auipc x4, 4 thực hiện gán giá trị x4 = PC + 10bin << 12 = 4dec + 16384dec = 16388dec.

4. Alternative Design

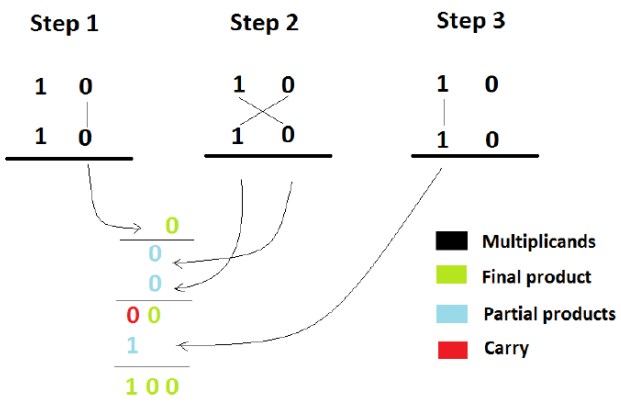
Trong thiết kế như hình 2.1, nhóm em có thiết kế thêm bộ nhân và thêm lệnh mul vào thiết kế trên với mục đích thực hiện tính toán trong những ứng dụng được dễ dàng hơn. Ưu điểm của phương pháp nhân Vedic là tăng tốc tính toán so với phương pháp tính toán thông thường là Array Multiplier. Dù có thể thực hiện được phép nhân bằng lệnh ở phần mềm nhưng việc thực hiện phép nhân bằng phần cứng giúp giảm thời gian tính toán đi rất đáng kể.

Lệnh mul: nhân 2 số có dấu 16 bit.

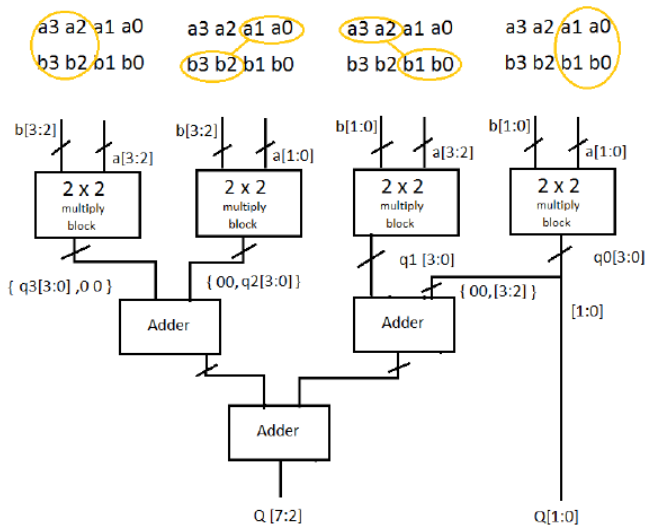
Phép nhân được thực hiện theo phương pháp Vedic.

Phép nhân 16 bit được kế thừa từ phép nhân 2, 4, 8 bit.

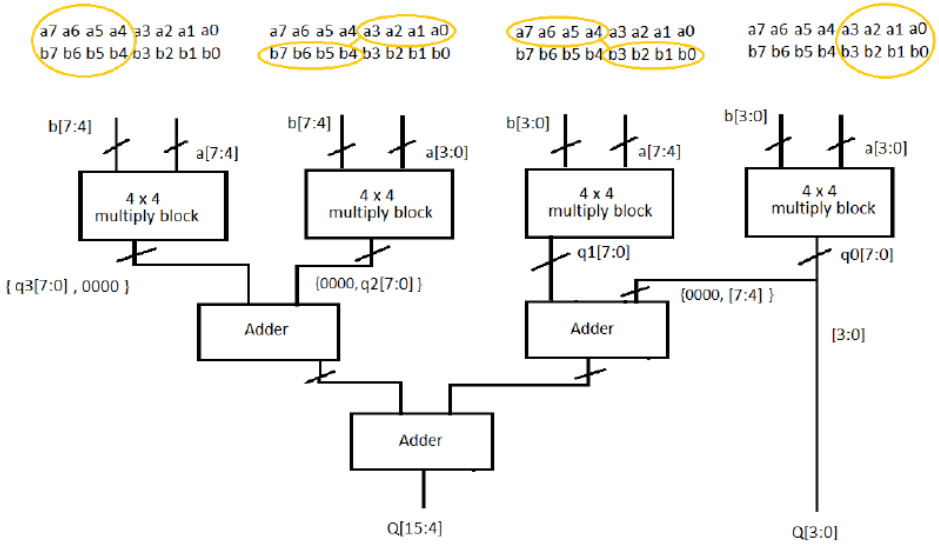
Phép nhân 2x2 bit như sau:



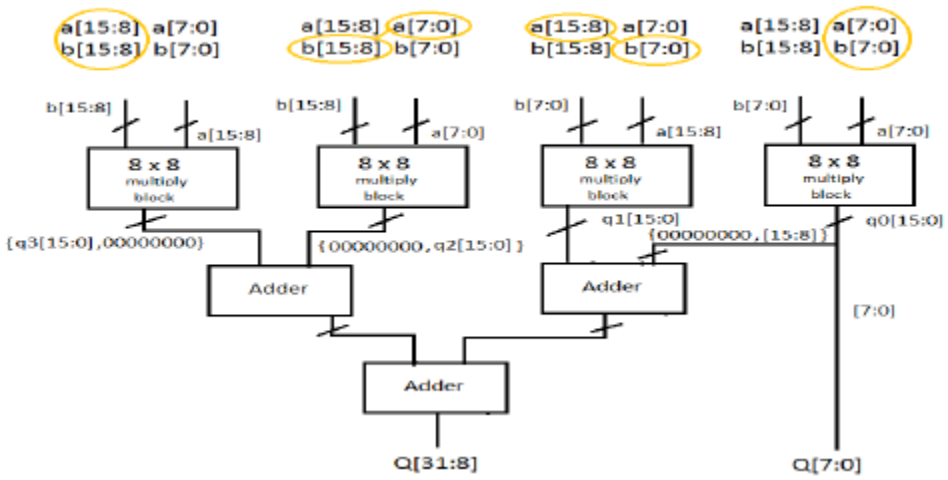
Phép nhân 4x4 bit như sau:



Phép nhân 8x8 bit:

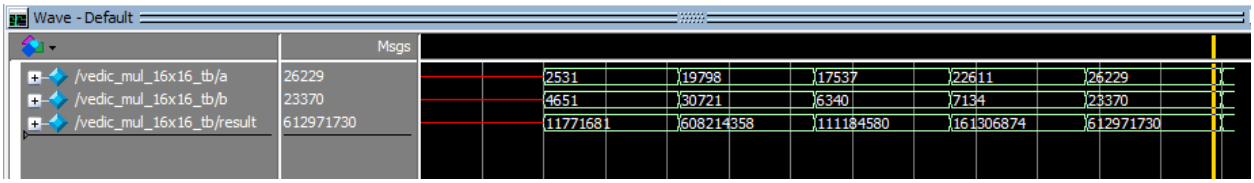


Phép nhân 16x16 bit:



Từ những block diagram trên thiết kế được các module vedic_mul_2x2, vedic_mul_4x4, vedic_mul_8x8, vedic_mul_16x16.

Sau đó khối vedic_mul_16x16 được kiểm tra trên ModelSim và thu được waveform như sau:



Hình 4.1. Waveform của phép nhân vedic 16x16

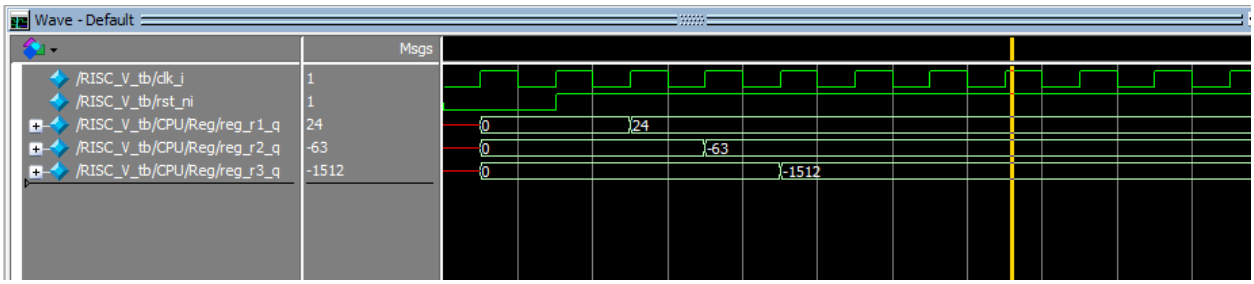
Vì chỉ thêm 1 lệnh mul nên tín hiệu Mul_ext được thêm vào bộ control unit để điều khiển bộ alu thực hiện phép nhân.

Cuối cùng lệnh mul được kiểm tra trên toàn bộ CPU.

Các lệnh được sử dụng như hình:

PC	Machine Code	Basic Code	Original Code
0x0	0x01800093	addi x1 x0 24	addi x1, x0, 24
0x4	0xFC100113	addi x2 x0 -63	addi x2, x0, -63
0x8	0x022081B3	mul x3 x1 x2	mul x3, x1, x2

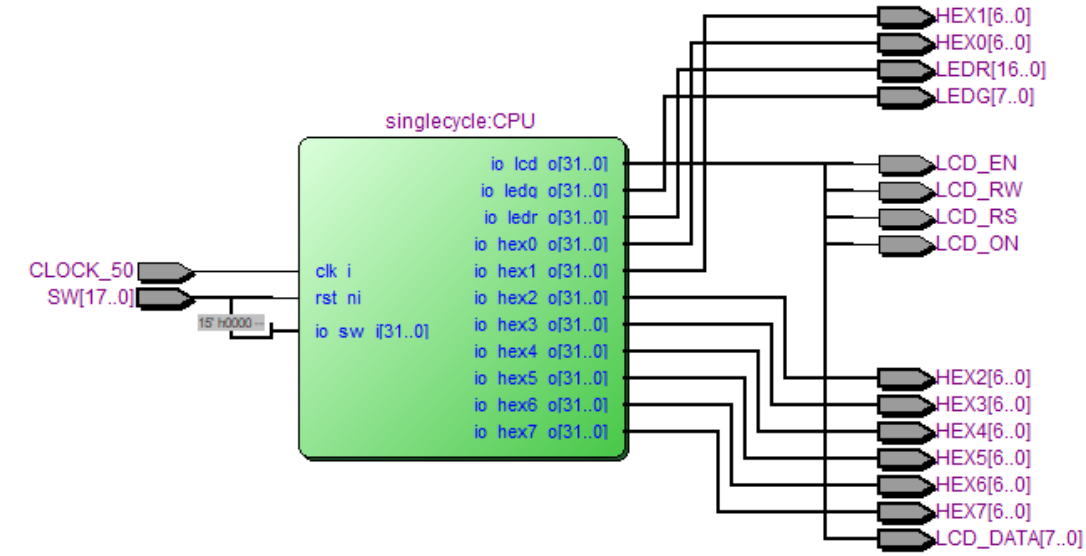
Waveform biểu diễn những lệnh trên:

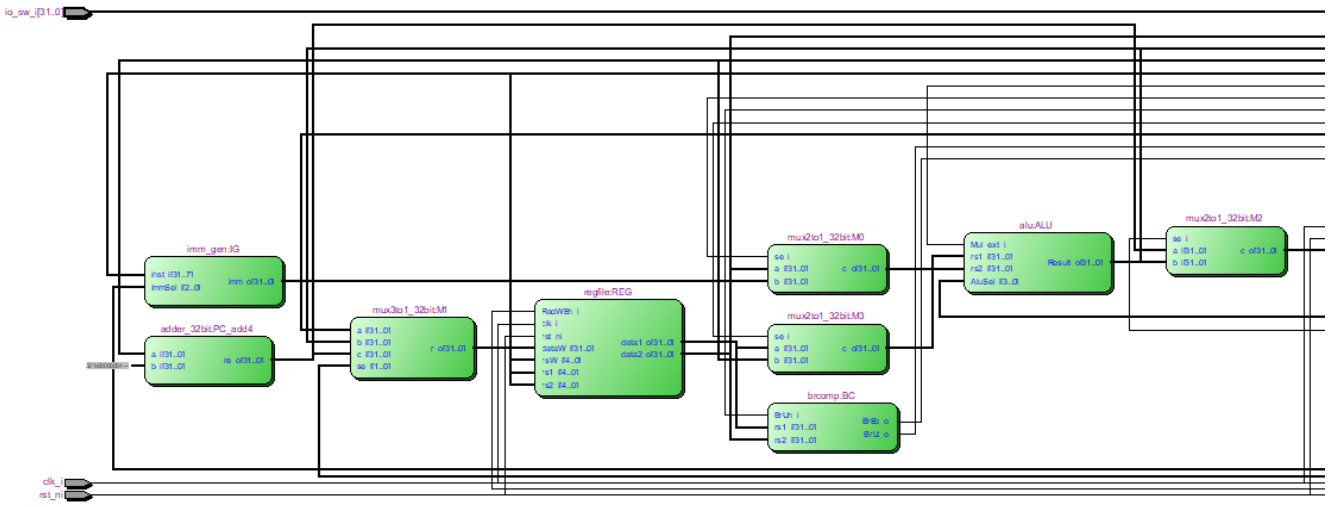


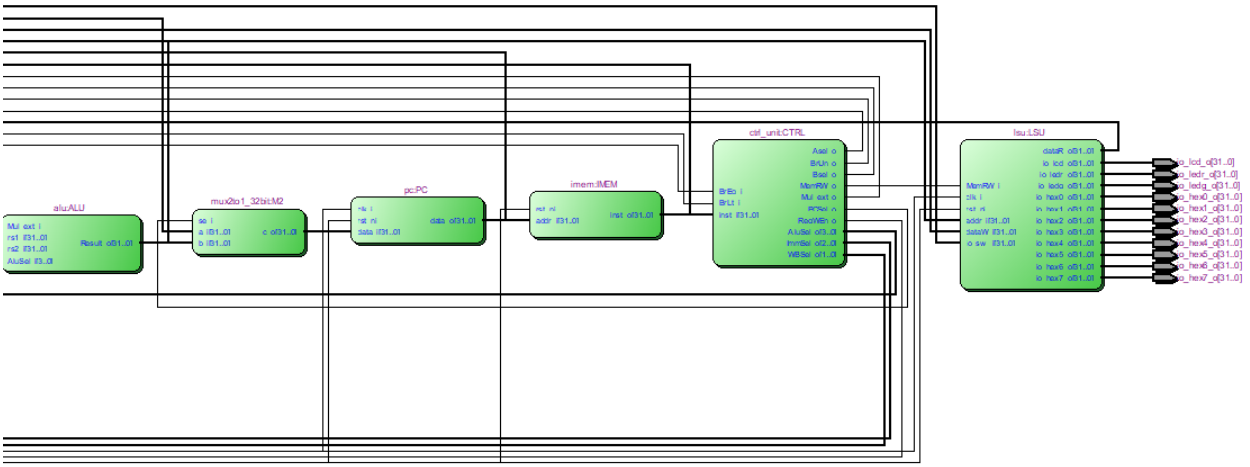
Từ đó, lệnh mul được thêm vào thiết kế để có thể tăng tốc việc thực hiện phép tính nhân và đơn giản hóa code hợp ngữ khi viết một số ứng dụng cụ thể.

5. Evaluation

Sau khi thiết kế hoàn chỉnh, mô phỏng được RTL của mạch như hình:







Các kết nối đã được kiểm tra và nhận thấy các khối được mô tả giống trong hình 2.1

Sau khi thực hiện chạy những lệnh trong bảng tập lệnh RV32I, kiểm tra những đường tín hiệu bằng phần mềm ModelSim nhận thấy giá trị đúng ở từng khối được mô tả.

Tuy nhiên, vi xử lý RV32I có thể được cải thiện tốc độ xử lý bằng cách áp dụng kiến trúc pipeline, thêm bộ nhớ cache. Hoặc mở rộng thêm multi-core hoặc đổi thành cấu trúc 6-7 tầng.

6. Conclusion

Vi xử lý single-cycle RV32I được thiết kế hoàn chỉnh và có thể thực hiện được những lệnh cơ bản nhưng mô tả ở phần 3. Một số ứng dụng cơ bản được viết bằng hợp ngữ và biên dịch ra mã hex, sau đó cho cpu chạy thử và thực hiện đúng chức năng. Chi tiết source code thiết kế, testbench và ứng dụng được đăng trên github:

https://github.com/Stork1323/Single_Cycle_Processor_RISC_V_32I.git