

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ



CẤU TRÚC MÁY TÍNH – EE3043

BÁO CÁO THÍ NGHIỆM – LỚP L01 – HK232

Milestone2: Design of a Single-Cycle Processor

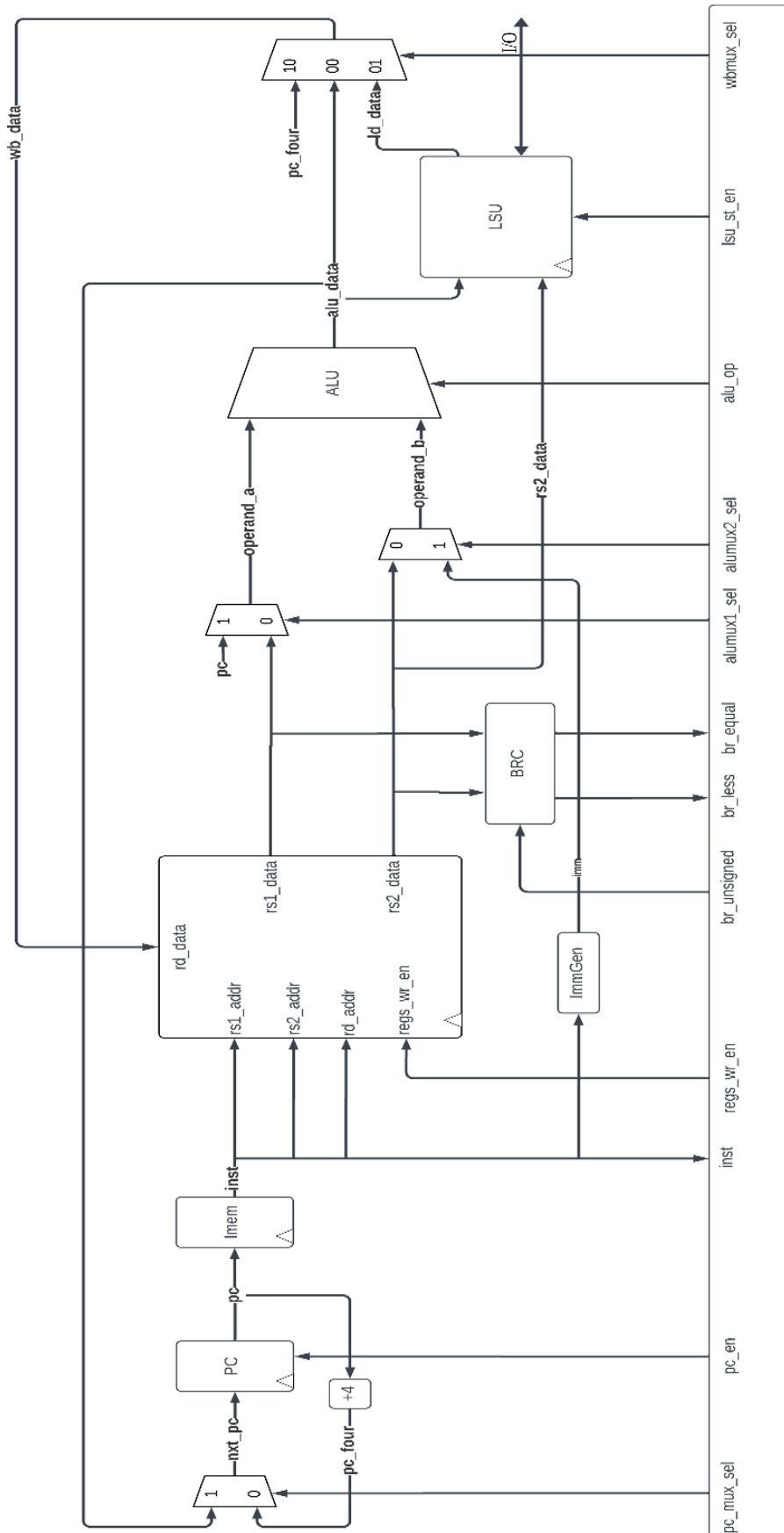
NHÓM 9: HUỖNH PHƯỚC TOÀN – 2012230
NGUYỄN ĐỨC CHÍNH – 2012739

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 04/2024

MỤC LỤC

1. SƠ ĐỒ KHỞI TỔNG QUÁT	1
2. MỘT SỐ MODULE CHÍNH	4
a. IMEM – Instruction Memory.....	4
b. REGFILE - Register file	5
c. IMMGEN – Immediate Generator.....	6
d. ALU - Arithmetic Logic Unit	7
e. BRC - Branch Comparison	9
f. LSU - Load-Store Unit.....	9
3. MỘT SỐ VẤN ĐỀ GẶP PHẢI	11
4. TỔNG KẾT	14
a. Core.....	14
b. Application.....	14

1. SƠ ĐỒ KHỐI TỔNG QUÁT



- So với sơ đồ khối trong *Milestone2 Tutorial* và *Berkeley's Lecture*, thì nhóm em có thêm một tín hiệu là *pc_en* (dùng để làm tín hiệu enable cho module PC hoạt động).
- Và nhóm em có đặt tên lại cho đồng bộ và dễ quản lý:

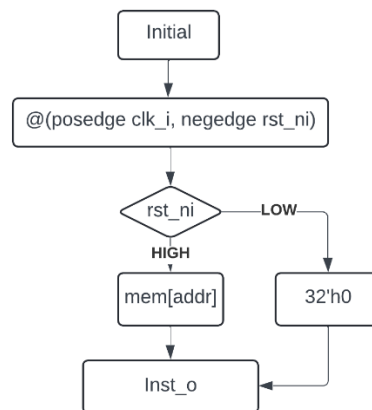
Berkeley's Lecture	Milestone2 Tutorial	In This Design	Meaning
PCSel	br_sel	pcmux_sel	Select signal: Source for PC
-	nxt_pc	nxt_pc	Next PC value
PC+4	pc_four	pc_four	PC+4
PC	pc	pc	Program Counter
-	-	pc_en	Enable signal: Program Counter
Inst	instr	inst	Instruction
DataD	rd_data	rd_data	Data: Destination Register
AddrA	rs1_addr	rs1_addr	Address: First Source Register
AddrB	rs2_addr	rs2_addr	Address: Second Source Register
AddrD	rd_addr	rd_addr	Address: Destination Register
RegWEn	rd_wren	regs_wr_en	Enable signal: Register File Write Access
R[rs1]	rs1_data	rs1_data	Data: First Source Register
R[rs2]	rs2_data	rs2_data	Data: Second Source Register
BrUn	br_unsigned	br_unsigned	Branch Unsigned
BrLT	br_less	br_less	Check Signal: Less Than
BrEq	br_equal	br_equal	Check Signal: Equal
Imm[31:0]	imm	imm	Immediate
ImmSel	-	-	Select the immediate value
-	operand_a	operand_a	ALU: First Operand
-	operand_b	operand_b	ALU: Second Operand
Asel	op_a_sel	alumux1_sel	Select signal: Source for operand_a
Bsel	op_b_sel	alumux2_sel	Select signal: Source for operand_b
ALUsel	alu_op	alu_op	ALU: Opcode

ALU	alu_data	alu_data	ALU: Result
Mem	ld_data	ld_data	Data: Load Instructions
MemRW	mem_wren	lsu_st_en	Enable Signal: LSU Stored Access
WBSel	wb_sel	wbmux_sel	Select signal: Source for wb_data
-	wb_data	wb_data	Data: Write-back

2. MỘT SỐ MODULE CHÍNH

a. IMEM – Instruction Memory

- Sơ đồ khối:



- Tóm gọn code (chi tiết xin mời xem tại github:

https://github.com/tlatonf/venmac/tree/20240413_milestone2/milestone2/00_sr_c).

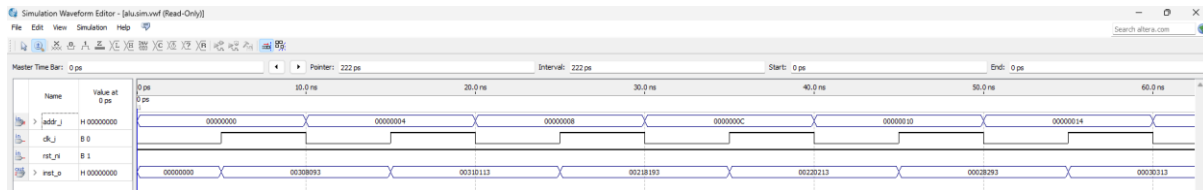
```

initial begin : readfile
    memory = '{`SIZE_IMEM{32'h0000_0000}}';
    $readmemh(`FILE_INST_HEX, memory);
end

always_ff @(posedge clk_i or negedge rst_ni) begin : imem_always
    if (!rst_ni) begin
        inst_o <= 32'h0000_0000;
    end else begin
        inst_o <= memory[f_srl(addr_i, 5'h02)];
    end
end
  
```

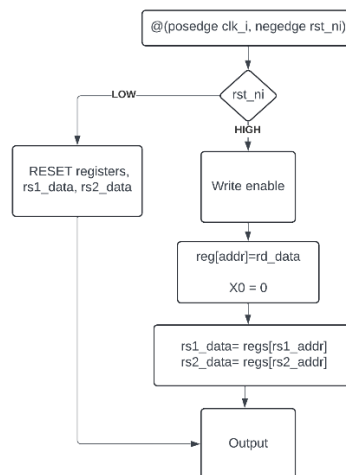
- Note: Function f_srl dùng để thực hiện dịch phải 2 đơn vị, mục đích là để thực hiện phép chia cho 4 (địa chỉ thanh ghi bằng giá trị thanh ghi PC chia cho 4).
- Verify: Dùng phương pháp waveform, truyền vào một file chương trình bất kỳ (.hex), sau đó kiểm tra thủ công xem module có thực hiện đúng yêu cầu thiết kế hay không. Thực hiện mô phỏng với đoạn chương trình bên dưới (ngẫu nhiên), ta thu được kết quả đúng theo yêu cầu.

```
00308093
00310113
00218193
00220213
00028293
00310113
...
```



b. REGFILE - Register file

- Sơ đồ khối:



- Tóm gọn code (chi tiết xin mời xem tại github:

https://github.com/tlatonf/venmac/tree/20240413_milestone2/milestone2/00_sr
c).

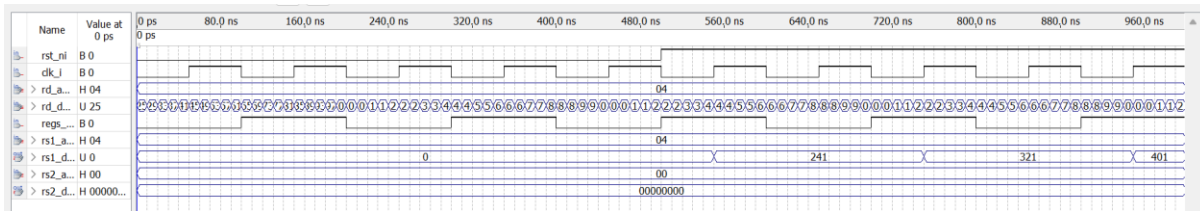
```
always @(posedge clk_i or negedge rst_ni) begin : regfile_always
  if (!rst_ni) begin
    registers = '{32{32'h0000_0000}};
    rs1_data_o = 32'h0000_0000;
    rs2_data_o = 32'h0000_0000;
  end else begin
    //Write
    if (regs wr en i) begin
```

```

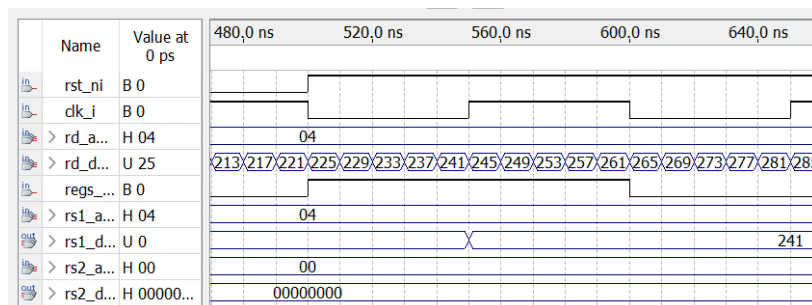
        registers[rd_addr_i] = (rd_addr_i == 5'b00000) ? 32'h0000_0000 :
rd_data_i;
    end
    //Read
    rs1_data_o = registers[rs1_addr_i];
    rs2_data_o = registers[rs2_addr_i];
end

```

- Verify: Dùng phương pháp waveform, cố định địa chỉ thanh ghi rd và rs1, cho giá trị rd_data chạy, sau đó kiểm tra thủ công xem module có thực hiện đúng yêu cầu thiết kế hay không.



- Trong trường hợp đọc và ghi trong cùng một chu kỳ, thì module sẽ thực hiện theo thứ tự ghi dữ liệu vào regfile rồi mới thực hiện đọc dữ liệu đưa ra ngoài.
Ví dụ: Tại thời điểm $t=550ns$, $write_enable=1$, module sẽ thực hiện lưu giá trị (trước đó) $rd_data=241$ vào thanh ghi có địa chỉ $0x04$, sau đó mới thực hiện việc đọc dữ liệu đưa ra $rs1_data$.



c. IMMGEN – Immediate Generator

- Cơ sở thiết kế: Giải mã IMM dựa vào OPCODE, sau đó dựa vào FUNCT3 để xác định xem là *msb-extends* hay *zero-extends*.

R_TYPE	Don't care	
I_TYPE	imm[11:0]	inst[31:20]
S_TYPE	imm[11:5 4:0]	inst[31:25 11:7]

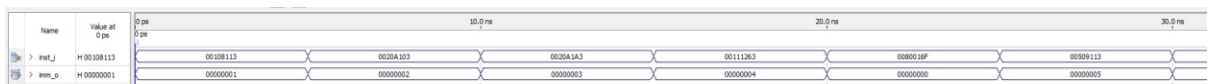
B_TYPE	imm[12 10:5 4:1 11]	inst [31 30:25 11:6 7]
U_TYPE	Imm[31:12]	inst[31:12]
J_TYPE	Imm[20 10:1 11 19:12]	inst[31:12]

- Code chi tiết:

https://github.com/tlatonf/venmac/tree/20240413_milestone2/milestone2/00_src

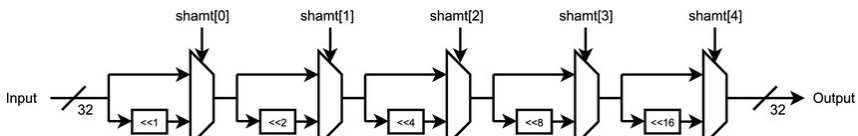
- Verify: Dùng phương pháp waveform, lựa chọn ngẫu nhiên các lệnh có imm ở các nhóm lệnh khác nhau, kiểm tra xem imm output có đúng yêu cầu hay không. Thực hiện mô phỏng với đoạn chương trình bên dưới (ngẫu nhiên), ta thu được kết quả đúng theo yêu cầu.

addi x2, x1, 1	0x00108113
lw x2, 2(x1)	0x0020a103
sw x2, 3(x1)	0x0020a1a3
bne x2, x1, 4	0x00111263
jal x2, 0	0x0080016f
addi x2, x1, 5	0x00508113
...	...



d. ALU - Arithmetic Logic Unit

- Cơ sở thiết kế: Do yêu cầu không được dùng **-, >, <, shift** nên em đã thiết kế ra các bộ tính toán như sau:

A - B	=> A + ~B + 1
A < B	=> A - B < 0 => A + ~B + 1 Overflow => function f_less, f_lessu
Shift	Dùng thuật toán Barrel Shift 

- Tóm gọn code (chi tiết xin mời xem tại github:

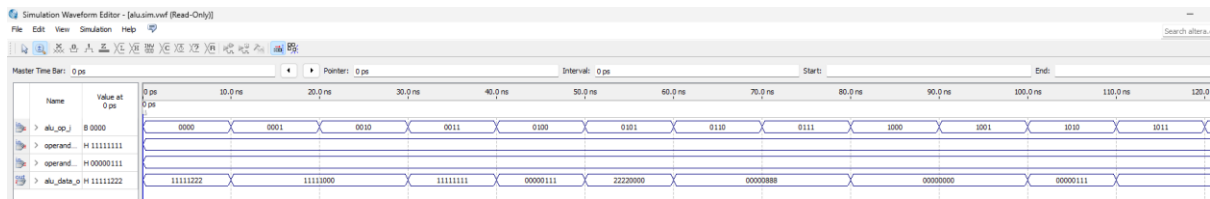
https://github.com/tlatonf/venmac/tree/20240413_milestone2/milestone2/00_sr_c).
c).

```
//function compare less than unsigned
function automatic logic f_lessu;
  input logic [31:0] a, b;
  logic [32:0] tmp;
  begin
    tmp = a + ~b + 32'h0000_0001;
    f_lessu = (tmp[32] == 1'b1) ? 1'b1 : 1'b0;
  end
endfunction
```

```
//function compare less than signed
function automatic logic f_less;
  input logic [31:0] a, b;
  begin
    case ({a[31], b[31]})
      //a is negative, b is negative
      2'b11: f_less = f_lessu(a, b) ? `FALSE : `TRUE;
      //a is negative, b is positive
      2'b10: f_less = `TRUE;
      //a is positive, b is negative
      2'b01: f_less = `FALSE;
      //a is positive, b is positive
      2'b00: f_less = f_lessu(a, b) ? `TRUE : `FALSE;
    endcase
  end
endfunction
```

```
//function shift left logical
function automatic logic [31:0] f_sll;
  input logic [31:0] num;
  input logic [4:0] shift;
  begin
    f_sll = (shift[0]) ? { num[30:0], 1'h0 } : num[31:0];
    f_sll = (shift[1]) ? { f_sll[29:0], 2'h0 } : f_sll[31:0];
    f_sll = (shift[2]) ? { f_sll[27:0], 4'h0 } : f_sll[31:0];
    f_sll = (shift[3]) ? { f_sll[23:0], 8'h00 } : f_sll[31:0];
    f_sll = (shift[4]) ? { f_sll[15:0], 16'h0000 } : f_sll[31:0];
  end
endfunction
```

- Verify: Dùng phương pháp waveform: Cho cố định giá trị của operand_a và operand_b, cho alu_op thay đổi sao cho bao quát tất cả các phép toán, ta thu được các giá trị của alu_data. Kiểm tra thủ công nhận thấy thiết kế đạt yêu cầu.



e. BRC - Branch Comparison

- Tóm gọn code (chi tiết xin mời xem tại github:

https://github.com/tlatonf/venmac/tree/20240413_milestone2/milestone2/00_sr_c).

```
always @ (rs1_data_i or rs2_data_i) begin : brcomp_always
    //check equal
    br_equal_o = (rs1_data_i == rs2_data_i) ? `TRUE : `FALSE;
    //check less than
    if (br_unsigned_i) begin : unsign_comp
        br_less_o = f_lessu(rs1_data_i, rs2_data_i) ? `TRUE : `FALSE;
    end else begin : sign_comp
        br_less_o = f_less(rs1_data_i, rs2_data_i) ? `TRUE : `FALSE;
    end
end
end
```

f. LSU - Load-Store Unit

- Tóm gọn code (chi tiết xin mời xem tại github:

https://github.com/tlatonf/venmac/tree/20240413_milestone2/milestone2/00_sr_c).

```
always_ff @(posedge clk_i or negedge rst_ni) begin : lsu_always
    if (!rst_ni) begin
        data_rd_o = 32'h0000_0000;
        io_lcd_o = 32'h0000_0000;
        io_ledg_o = 32'h0000_0000;
        io_ledr_o = 32'h0000_0000;
        io_hex7_o = 32'h0000_0000;
        io_hex6_o = 32'h0000_0000;
        io_hex5_o = 32'h0000_0000;
        io_hex4_o = 32'h0000_0000;
        io_hex3_o = 32'h0000_0000;
        io_hex2_o = 32'h0000_0000;
        io_hex1_o = 32'h0000_0000;
        io_hex0_o = 32'h0000_0000;
        memory = '{`SIZE_DMEM{32'h0000_0000}};
    end else begin
        //Read
        data_rd_o = memory[addr_i/4];
        io_lcd_o = memory[`IO_LCD];
        io_ledg_o = memory[`IO_LEDG];
        io_ledr_o = memory[`IO_LEDR];
    end
end
```

```
io_hex7_o = memory[`IO_HEX7];
io_hex6_o = memory[`IO_HEX6];
io_hex5_o = memory[`IO_HEX5];
io_hex4_o = memory[`IO_HEX4];
io_hex3_o = memory[`IO_HEX3];
io_hex2_o = memory[`IO_HEX2];
io_hex1_o = memory[`IO_HEX1];
io_hex0_o = memory[`IO_HEX0];

//Write
memory[`IO_SW] = io_sw_i[`VECTOR_RANGE];
memory[`IO_KEY] = io_key_i[`VECTOR_RANGE];
if (st_en_i) begin
    memory[addr_i/4] = st_data_i;
end
end

$writememh("./memory/dmem.hex", memory);

end
```

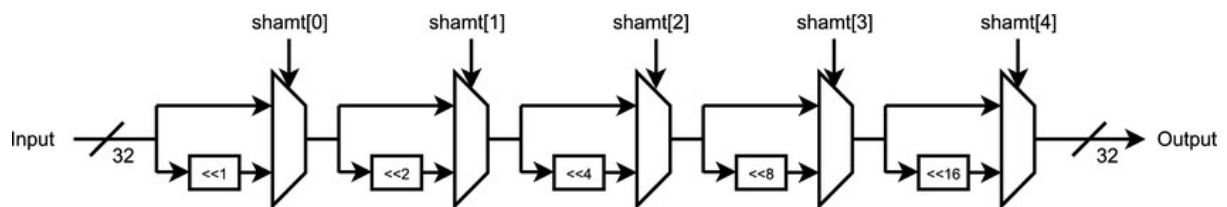
- LSU dùng phương pháp verify tương tự REGFILE.

3. MỘT SỐ VẤN ĐỀ GẶP PHẢI

Vấn đề 1: Trong function ShiftRight: Không thể thao tác gán giá trị bit ($num[31-shift:0]$) khi trong phép gán chứa tham số chưa xác định ($shift$): *Error (10734): Verilog HDL error at alu.sv(64): shift is not a constant*

```
function [31:0] ShiftRightLogical;
input logic [31:0] num;
input logic [31:0] shift;
begin
    if (CompLessThanUnsigned(shift, 32'd32)) begin
        ShiftRightLogical = {{shift{1'b0}}, num[31 - shift:0]};
    end else begin
        ShiftRightLogical = 32'h0;
    end
end
endfunction
```

- Giải pháp: Dùng thuật toán *Barrel Shift*



```
//function shift left logical
function automatic logic [31:0] f_sll;
input logic [31:0] num;
input logic [4:0] shift;
begin
    f_sll = (shift[0]) ? { num[30:0], 1'h0 } : num[31:0];
    f_sll = (shift[1]) ? { f_sll[29:0], 2'h0 } : f_sll[31:0];
    f_sll = (shift[2]) ? { f_sll[27:0], 4'h0 } : f_sll[31:0];
    f_sll = (shift[3]) ? { f_sll[23:0], 8'h00 } : f_sll[31:0];
    f_sll = (shift[4]) ? { f_sll[15:0], 16'h0000 } : f_sll[31:0];
end
endfunction
```

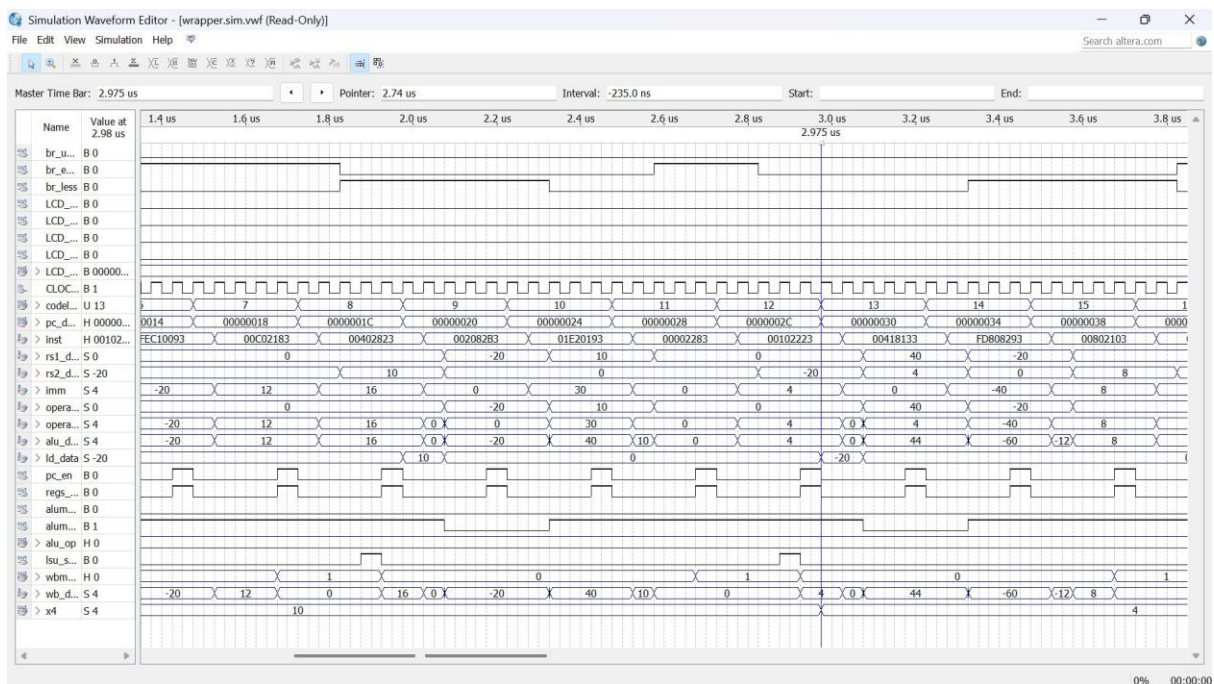
Vấn đề 2: Gặp lỗi DEFINE LANG_MAP DLHSYN

```
comp00u09@miranda:~/venma  x  comp00u09@miranda:~/venm  x  Windows PowerShell  x  +  v
10 xrun(64): 23.09-s003: (c) Copyright 1995-2023 Cadence Design Systems, Inc.
9 TOOL: xrun(64) 23.09-s003: Started on Apr 02, 2024 at 11:33:34 +07
8 xrun
7 -hal
6 -sv
5 ../00_src/define.sv
4 ../00_src/mux2input.sv
3 DEFINE LANG_MAP (`M
2 |
1 xrun: *F,DLHSYN (./xcelium.d/run.lnx8664.23.09.d/hdLrun.var,1): hdl.var syntax error '(`'.
11 TOOL: xrun(64) 23.09-s003: Exiting on Apr 02, 2024 at 11:33:35 +07 (total: 00:00:01)
~
~
~
```

- Nhóm tụi em chưa tìm hiểu được đây là lỗi gì, nó không thường xuyên xảy ra, tuy nhiên tụi em mất rất nhiều thời gian để xác định nguyên nhân của nó.
- Giải pháp: Tụi em xoá tất cả file output từ Xcelium, sau đó chạy lại thì thành công.

Vấn đề 3: Code sai do sơ suất: Luôn luôn enable write regfile khi vào stage Writeback (sai với các lệnh S_TYPE và B_TYPE)

- Hậu quả là nhóm phải bung toàn bộ source ra để debug:



Vấn đề 4: Gặp hơn 500 cảnh báo do gặp kí tự đặt biệt trong file

- Lí giải: Do nhóm em code bên môi trường windows, sau đó mới tiến hành chạy mô phỏng ở linux. Mà kí tự kết thúc một dòng ở windows là **CR LF**, còn bên linux chỉ là **LF**. Đây là một kiến thức khá cơ bản, nhưng phải rất lâu sau thì nhóm mới nhận ra sai lầm của mình.
- Giải pháp: Thực hiện việc chuyển đổi từ DOS sang UNIX khi copy file từ windows sang linux. Thực hiện bằng lệnh có sẵn trong linux:

```
find . -type f -print0 | xargs -0 dos2unix
```

```

7 =====
6
5 Analysis summary :
4
3 Errors : (33)
2 CBPAHI (33)
1
266 Warnings : (878)
1 ATGLGC (1) BADSYS (1) BBXSIG (1) BITUNS (10)
2 BITUSD (1) BLKSQB (20) CDEFVC (1) CSTEP (2)
3 CTLCHR (531) DALIAS (1) DFDRVS (1) EXTFSM (1)
4 FFWSNR (11) IMPDTC (15) INDXOP (1) INIUSP (1)
5 IPRTEX (2) IPSFOU (13) IPUFSE (1) LATBAS (1)
6 LATINF (9) LATMLG (1) LCVARN (7) LENCPI (9)
7 LRGOPR (6) MAXLEN (15) MULOPR (2) MXUANS (1)
8 NAUTOF (7) NOBLKN (5) NOLOCL (2) OBMEMI (2)
9 OLDALW (1) POOBID (2) PRMFSM (11) RDBFAS (3)
10 RDOPND (1) STYVAL (80) SYNASN (1) SYNPRT (70)
11 TRNMBT (2) UELCIT (2) URDREG (2) USEPRT (17)
12 VERPDR (1) VLGWEM (2)

```

Vấn đề 5: Không tìm được nguyên nhân và cách sửa lỗi *halstruct: CBPAHI*

Combinatorial path crossing multiple units drives.

- Khi checksyntax bằng Xcelium thì nhóm em vẫn còn 33 lỗi chưa được giải quyết (không báo lỗi khi compile bằng quartus). Tất cả đều báo là Combinatorial path crossing multiple units drives (bao gồm 1 lỗi do pc, và 32 lỗi do 32 thanh ghi regfile).
- Tuy em đã thử nhiều cách nhưng vẫn không sửa được lỗi này.

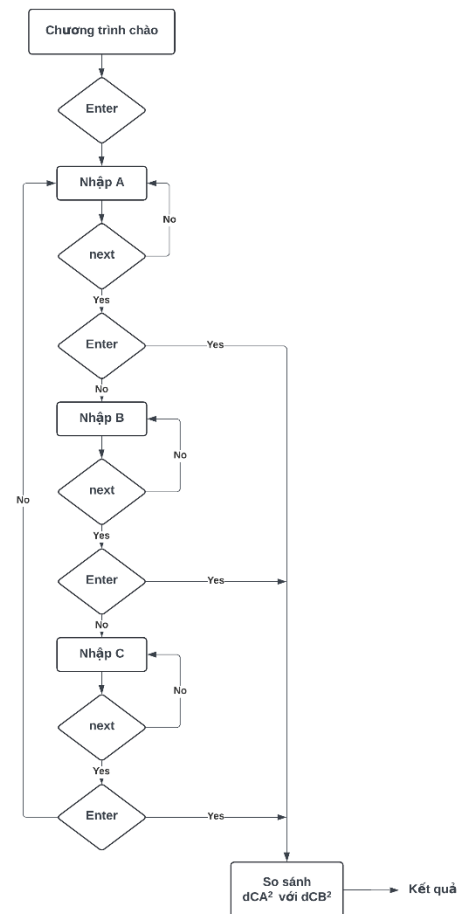
4. TỔNG KẾT

a. Core

- Nhóm đã hoàn thành phần core, đánh giá chất lượng đạt 100%. Đánh giá nhờ vào độ tin cậy và kết quả thực hiện một lượng lớn các lệnh ngẫu nhiên (*đa phần lệnh sau lấy kết quả của lệnh liền trước nó*) giữa thiết kế của nhóm và trình mô phỏng của [Cornell University](https://www.cornell.edu).

b. Application

- Program: *Input 3 2-D coordinates of A, B, and C. Determine which point, A or B, is closer to C using LCD as the display.*
- Sơ đồ giải thuật:
- Mức độ hoàn thành tại thời điểm nộp báo cáo: 75% (*đánh giá dựa số task đã hoàn thành, minh chứng: [link code](#), nhóm em đang gặp bugs ở phần nhập tọa độ từ KEY và SW – task cuối cùng*).
- Link video thuyết trình thiết kế của nhóm (*cập nhật sau khi nộp bài*): [video thuyết trình](#).



HẾT !