ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KỸ THUẬT MÁY TÍNH

----------ଓ★ଌ----------

BÁO CÁO LAB 4 + 5

THIẾT KẾ VÀ KIỂM TRA BỘ XỬ LÝ RISC-V 32I

(Single Cycle + Pipeline)

Lớp: CE409.O21.2

Giảng viên hướng dẫn: Phạm Thanh Hùng

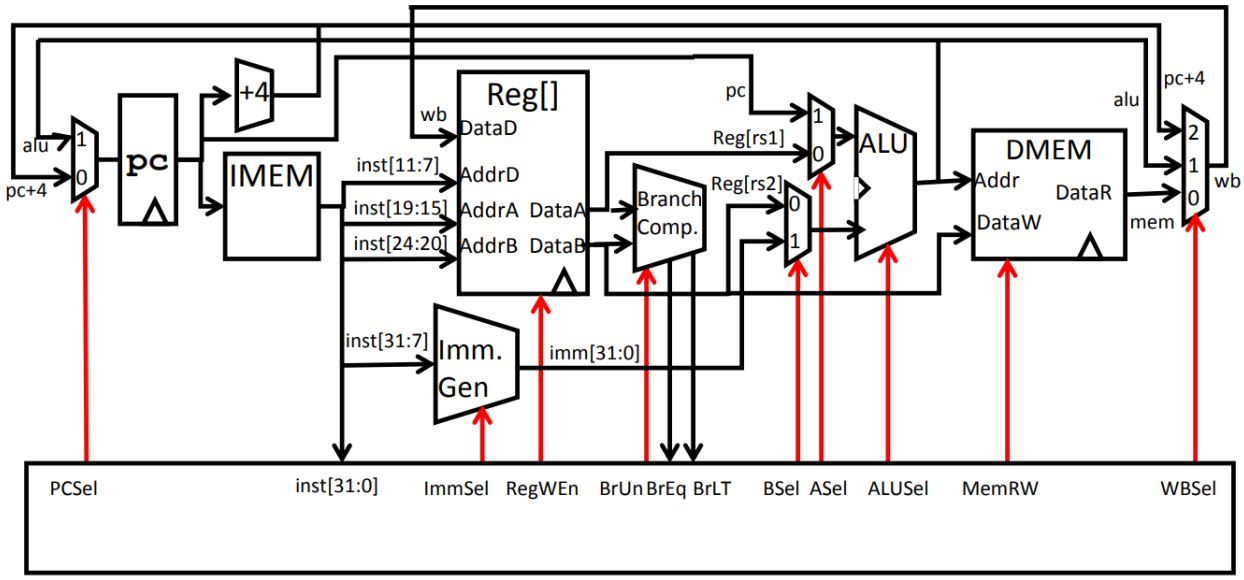
Sinh viên thực hiện:
1. Đào Phước Tài 21521391

TP.Hồ Chí Minh, ngày 11 tháng 6 năm 2024

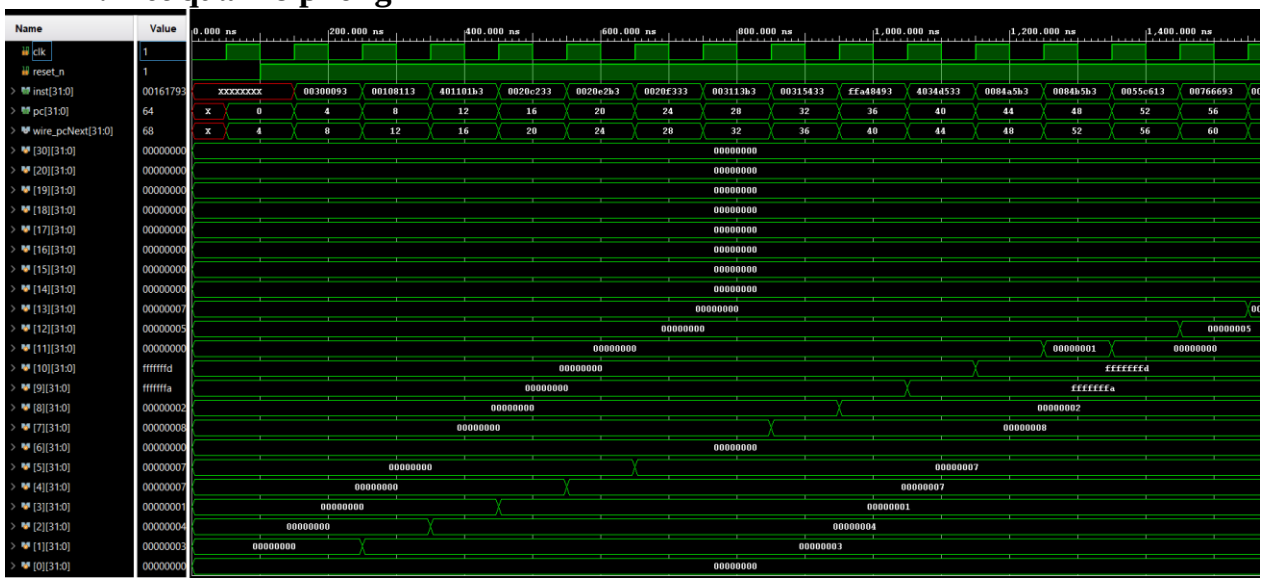
MỤC LỤC

I. SINGLE CYCLE RISC-V 32I

1. Mô hình thiết kế single cycle



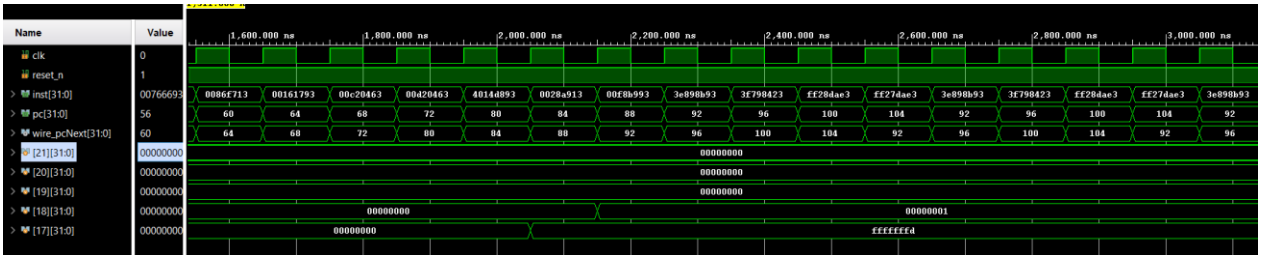
2. Kết quả mô phỏng



Các mã lệnh kiểm tra:

addi x1, x0, 3	x1 = 3	hexcode: 0x00300093
addi x2, x1, 1	x2 = 4	hexcode: 0x00108113
sub x3, x2, x1	x3 = 1	hexcode: 0x401101b3
xor x4, x1, x2	x4 = 7	hexcode: 0x0020c233
or x5, x1, x2	x5 = 7	hexcode: 0x0020e2b3
and x6, x1, x2	x6 = 0	hexcode: 0x0020f333
sll x7, x2, x3	x7 = 8	hexcode: 0x003113b3
srl x8, x2, x3	x8 = 2	hexcode: 0x00315433

addi x9, x9, -6 x9 = -6 hexcode: 0xffa48493
sra x10, x9, x3 x10 = -3 hexcode: 0x4034d533
slt x11, x9, x8 x11 = 1 hexcode: 0x0084a5b3
sltu x11, x9, x8 x11 = 0 hexcode: 0x0084b5b3
xori x12, x11, 5 x12 = 5 hexcode: 0x0055c613
ori x13, x12, 7 x13 = 7 hexcode: 0x00766693



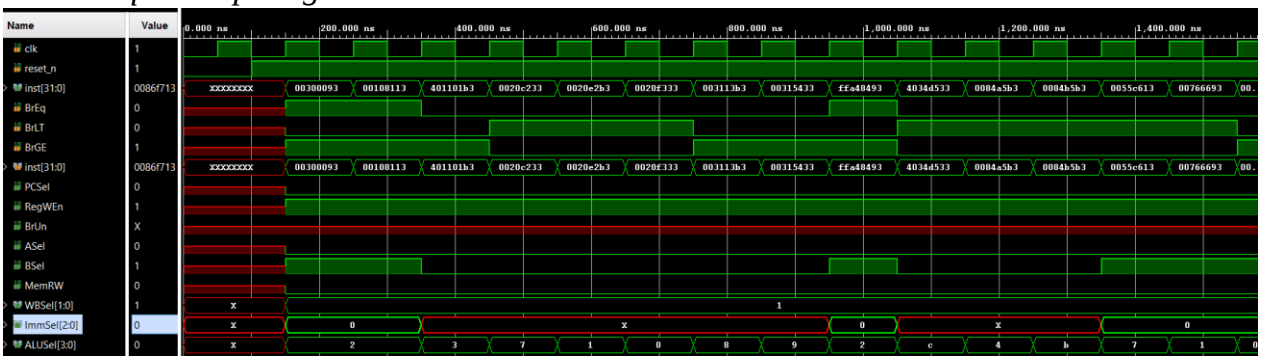
Các mã lệnh kiểm tra tiếp tục:

andi x14, x13, 8 x14 = 0 hexcode: 0x0086f713
slli x15, x12, 1 x15 = 10 hexcode: 0x00161793
beq x4, x12, 8 not jump hexcode: 0x00c20463
beq x4, x13, 8 jump hexcode: 0x00d20463

… tương tự với các lệnh nhảy khác như (beq, bne, bge, bgeu, ….)

3. Thiết kế khối CONTROLLER

3.1. Kết quả mô phỏng CONTROLLER



3.2. Bảng tín hiệu

3.2.1. Tín hiệu từ các khối ALUSel, ImmSel, WBSel

ALUSel	Instruction	ImmSel	Inst	WBSel	Instruction
0000	and	000	I	00	DMEM
0001	or	001	J	01	ALU
0010	add	010	S	10	pc + 4
0011	sub	011	U		
0100	slt	100	B	xx	Don't care
0101	nor	xxx	Don't care		
0110	sll (<< 12)				
0111	xor				
1000	A<<B				
1001	A>>B				
1010	A+(B<<12)				
1011	sltu				
1100	sra				

3.2.2. Bảng tín hiệu tổng quát

Type	Opcode	Inst	PCSel	ImmSel[2:0]	RegWEn	BrUn	ASel	BSel	ALUSel[3:0]	MemRW	WBSel[1:0]
R	0110011	add	0	xxx	1	x	0	0	0010	0	01
R	0110011	sub	0	xxx	1	x	0	0	0011	0	01
R	0110011	xor	0	xxx	1	x	0	0	0111	0	01
R	0110011	or	0	xxx	1	x	0	0	0001	0	01
R	0110011	and	0	xxx	1	x	0	0	0000	0	01
R	0110011	sll	0	xxx	1	x	0	0	1000	0	01
R	0110011	srl	0	xxx	1	x	0	0	1001	0	01
R	0110011	sra	0	xxx	1	x	0	0	1100	0	01
R	0110011	slt	0	xxx	1	x	0	0	0100	0	01
R	0110011	sltu	0	xxx	1	x	0	0	1011	0	01
I	0010011	addi	0	000	1	x	0	1	0010	0	01
I	0010011	xori	0	000	1	x	0	1	0111	0	01
I	0010011	ori	0	000	1	x	0	1	0001	0	01
I	0010011	andi	0	000	1	x	0	1	0000	0	01
I	0010011	slli	0	000	1	x	0	1	1000	0	01
I	0010011	srli	0	000	1	x	0	1	1001	0	01
I	0010011	srai	0	000	1	x	0	1	1100	0	01
I	0010011	slti	0	000	1	x	0	1	0100	0	01
I	0010011	sltiu	0	000	1	x	0	1	1011	0	01

Type	Opcode	Inst	PCSel	ImmSel[2:0]	RegWEn	BrUn	ASel	BSel	ALUSel[3:0]	MemRW	WBSel[1:0]
I_load	000011	lb	0	000	1	x	0	1	0010	0	00
I_load	000011	lh	0	000	1	x	0	1	0010	0	00
I_load	000011	lw	0	000	1	x	0	1	0010	0	00
I_load	000011	lbu	0	000	1	x	0	1	0010	0	00
I_load	000011	lhu	0	000	1	x	0	1	0010	0	00
S	0100011	sb	0	010	0	x	0	1	0010	1	xx
S	0100011	sh	0	010	0	x	0	1	0010	1	xx
S	0100011	sw	0	010	0	x	0	1	0010	1	xx
B	1100011	beq	BrEq	100	0	0	1	1	0010	0	xx
B	1100011	bne	!BrEq	100	0	0	1	1	0010	0	xx
B	1100011	blt	BrLT	100	0	0	1	1	0010	0	xx
B	1100011	bge	BrGE	100	0	0	1	1	0010	0	xx
B	1100011	bltu	BrLT	100	0	1	1	1	0010	0	xx
B	1100011	bgeu	BrLT	100	0	1	1	1	0010	0	xx
J	1101111	jal	1	001	1	x	1	1	0010	0	10
I	1100111	jalr	1	000	1	x	0	1	0010	0	10
U	0110111	lui	0	011	1	x	0	1	0110	0	01
U	0010111	auipc	0	011	1	x	1	1	1010	0	01

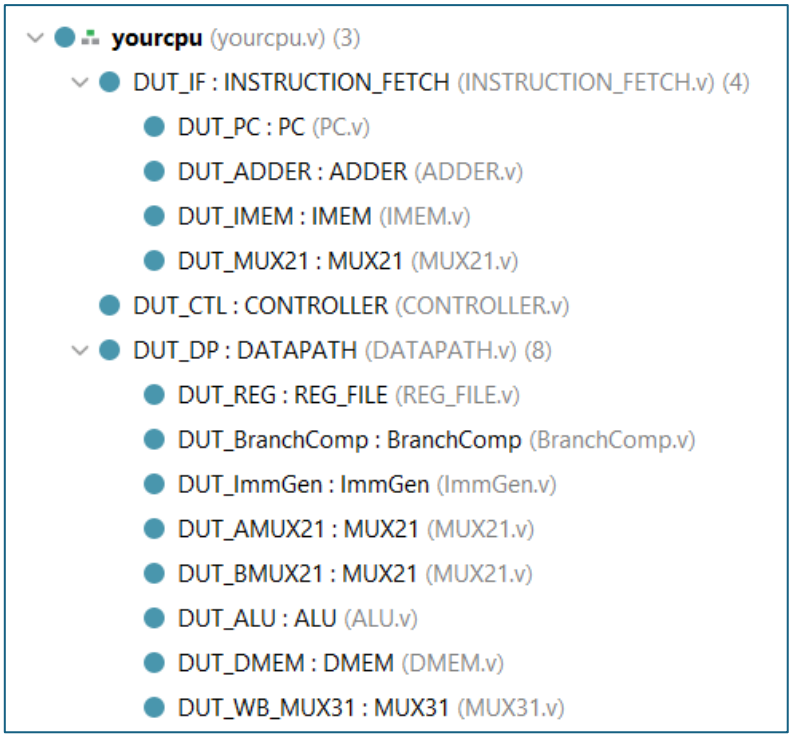
3.3. Mã nguồn thiết kế

```
module CONTROLLER(PCSel, inst, ImmSel, RegWEn, BrUn, BrEq, BrLT, BrGE, ASel, BSel, ALUSel, MemRW,
WBSel);
    parameter R_type        = 7'b0110011;
    parameter I_type        = 7'b0010011;
    parameter I_type_load   = 7'b0000011;
    parameter I_type_jalr   = 7'b1100111;
    parameter S_type        = 7'b0100011;
    parameter B_type        = 7'b1100011;
    parameter J_type        = 7'b1101111;
    parameter U_type_lui    = 7'b0110111;
    parameter U_type_auipc  = 7'b0010111;
    input          BrEq, BrLT, BrGE;
    input  [31:0]  inst;
    output         PCSel, RegWEn, BrUn, ASel, BSel, MemRW;
    output [1:0]   WBSel;
    output [2:0]   ImmSel;
    output [3:0]   ALUSel;
    wire [6:0]     opcode;
    wire [2:0]     funct3;
    wire [6:0]     funct7;
    reg  [14:0]    controlSignal;

    assign opcode = inst[6:0];
    assign funct3 = inst[14:12];
    assign funct7 = inst[31:25];

    always @(*) begin
        case(opcode)
                R_type: case(funct3)
                    3'b000: case(funct7)
                        7'b0000000: controlSignal <= 15'b0xxx1x000010001; //add
                        7'b0100000: controlSignal <= 15'b0xxx1x000011001; //sub
                            endcase
                    3'b001:         controlSignal <= 15'b0xxx1x001000001; //sll
                    3'b010:         controlSignal <= 15'b0xxx1x000100001; //slt
                    3'b011:         controlSignal <= 15'b0xxx1x001011001; //sltu
                    3'b100:         controlSignal <= 15'b0xxx1x000111001; //xor
                    3'b101: case(funct7)
                        7'b0000000: controlSignal <= 15'b0xxx1x001001001; //srl
                        7'b0100000: controlSignal <= 15'b0xxx1x001100001; //sra
                            endcase
                    3'b110:         controlSignal <= 15'b0xxx1x000001001; //or
                    3'b111:         controlSignal <= 15'b0xxx1x000000001; //and
                endcase
```

```
                I_type:     case(funct3)
                    3'b000:         controlSignal <= 15'b00001x010010001; //addi
                    3'b001:         controlSignal <= 15'b00001x011000001; //slli
                    3'b010:         controlSignal <= 15'b00001x010100001; //slti
                    3'b011:         controlSignal <= 15'b00001x011011001; //sltiu
                    3'b100:         controlSignal <= 15'b00001x010111001; //xori
                    3'b101: case(inst[31:25])
                        7'b0000000: controlSignal <= 15'b00001x011001001; //srli
                        7'b0100000: controlSignal <= 15'b00001x011100001; //srai
                            endcase
                    3'b110:         controlSignal <= 15'b00001x010001001; //ori
                    3'b111:         controlSignal <= 15'b00001x010000001; //andi
                endcase
                I_type_load: case(funct3)
                    3'b000:         controlSignal <= 15'b00001x010010000; //lb
                    3'b001:         controlSignal <= 15'b00001x010010000; //lh
                    3'b010:         controlSignal <= 15'b00001x010010000; //lw
                    3'b100:         controlSignal <= 15'b00001x010010000; //lbu
                    3'b101:         controlSignal <= 15'b00001x010010000; //lhu
                endcase
                I_type_jalr:        controlSignal <= 15'b10001x010010010; //jarl
                S_type:     case(funct3)
                    3'b000:         controlSignal <= 15'b00100x0100101xx; //sb
                    3'b001:         controlSignal <= 15'b00100x0100101xx; //sh
                    3'b010:         controlSignal <= 15'b00100x0100101xx; //sw
                endcase
                B_type:     case(funct3)
                    3'b000:         controlSignal <= { BrEq, 14'b100001100100xx}; //beq
                    3'b001:         controlSignal <= {!BrEq, 14'b100001100100xx}; //bne
                    3'b100:         controlSignal <= { BrLT, 14'b100001100100xx}; //blt
                    3'b101:         controlSignal <= { BrGE, 14'b100001100100xx}; //bge
                    3'b110:         controlSignal <= { BrLT, 14'b100011100100xx}; //bltu
                    3'b111:         controlSignal <= {!BrLT, 14'b100011100100xx}; //bgeu
                endcase
                J_type:             controlSignal <= 15'b10011x110010010; //jal
                U_type_lui:         controlSignal <= 15'b00111x010110001; //lui
                U_type_auipc:       controlSignal <= 15'b00111x111010001; //auipc

                default:
                    controlSignal <= 15'bxxxxxxxxxxxxxxx;
        endcase
    end
    assign {PCSel, ImmSel, RegWEn, BrUn, ASel, BSel, ALUSel, MemRW, WBSel} = controlSignal;
endmodule
```

4. Thiết kế khối DATAPATH

4.1. Mô hình thiết kế



4.2. Tổng hợp thiết kế (yourcpu.v)

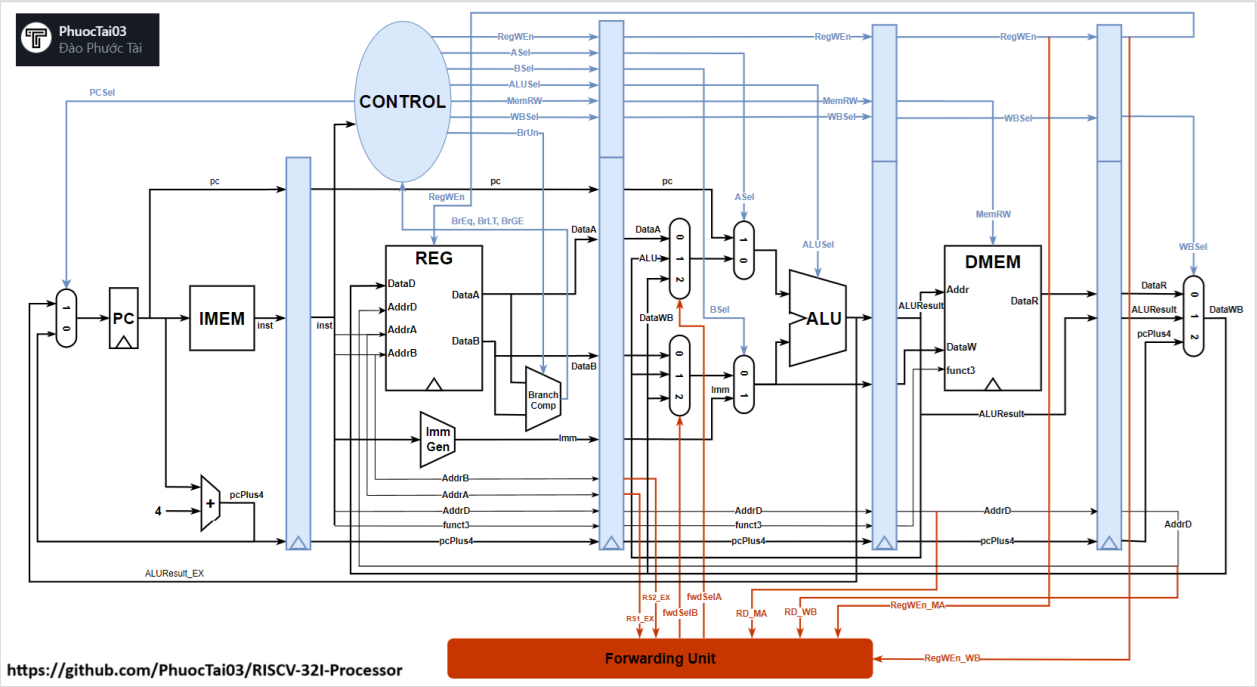
```
`timescale 1ns / 1ps
module yourcpu(clk, reset_n, tb_addr, tb_inst);
    input          clk, reset_n;
    input [31:0]   tb_addr, tb_inst;
    wire           PCSel, BrUn, BrEq, BrLT, BrGE, RegWEn, ASel, BSel, MemRW;
    wire [1:0]     WBSel;
    wire [2:0]     ImmSel;
    wire [3:0]     ALUSel;
    wire [31:0]    inst, alu, pc, pcPlus4;

    INSTRUCTION_FETCH  DUT_IF(clk, reset_n, PCSel, alu, inst, pc, pcPlus4, tb_addr, tb_inst);
    CONTROLLER         DUT_CTL(PCSel, inst, ImmSel, RegWEn, BrUn, BrEq, BrLT, BrGE, ASel, BSel,
ALUSel, MemRW, WBSel);
    DATAPATH           DUT_DP(ImmSel, RegWEn, BrUn, BrEq, BrLT, BrGE, ASel, BSel, ALUSel, MemRW,
WBSel, clk, inst, pc, pcPlus4, alu);
endmodule
```

II. PIPELINE RISC-V 32I

1. Kết quả thực hiện

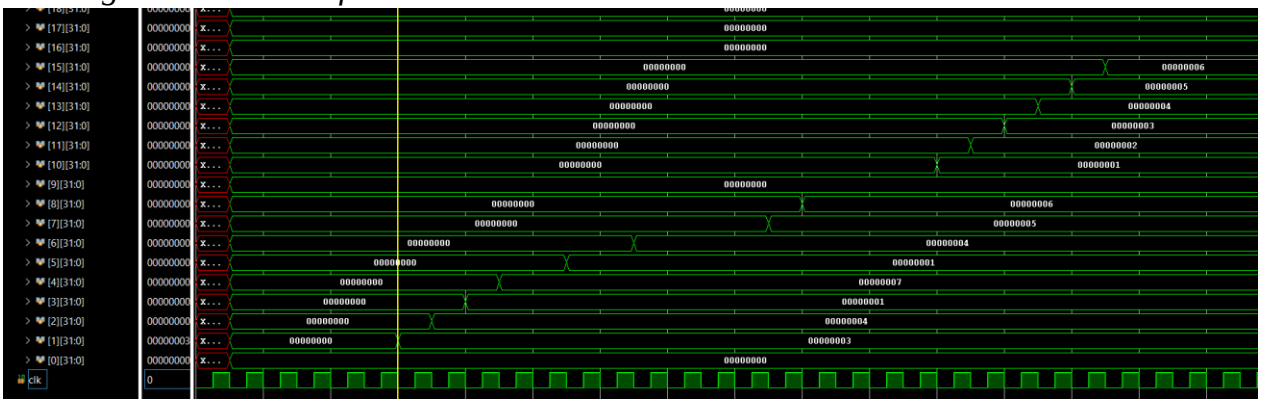
- Đã xử lý được
  - Xây dựng kiến trúc tập lệnh single/pipeline RISCV full lệnh cơ bản (37 lệnh).
  - Xây dựng testbench để kiểm tra thiết kế.
  - Xử lý được Data Hazard - Forwarding.
- Chưa xử lý được
  - Chưa xử lý được Data Hazard - Stalling
  - Chưa xử lý được Control Hazard (dùng lệnh nop trong testbench để thay thế).

2. Mô hình thiết kế pipeline



3. Kết quả mô phỏng (lệnh tự test)

3.1. Register Files waveform



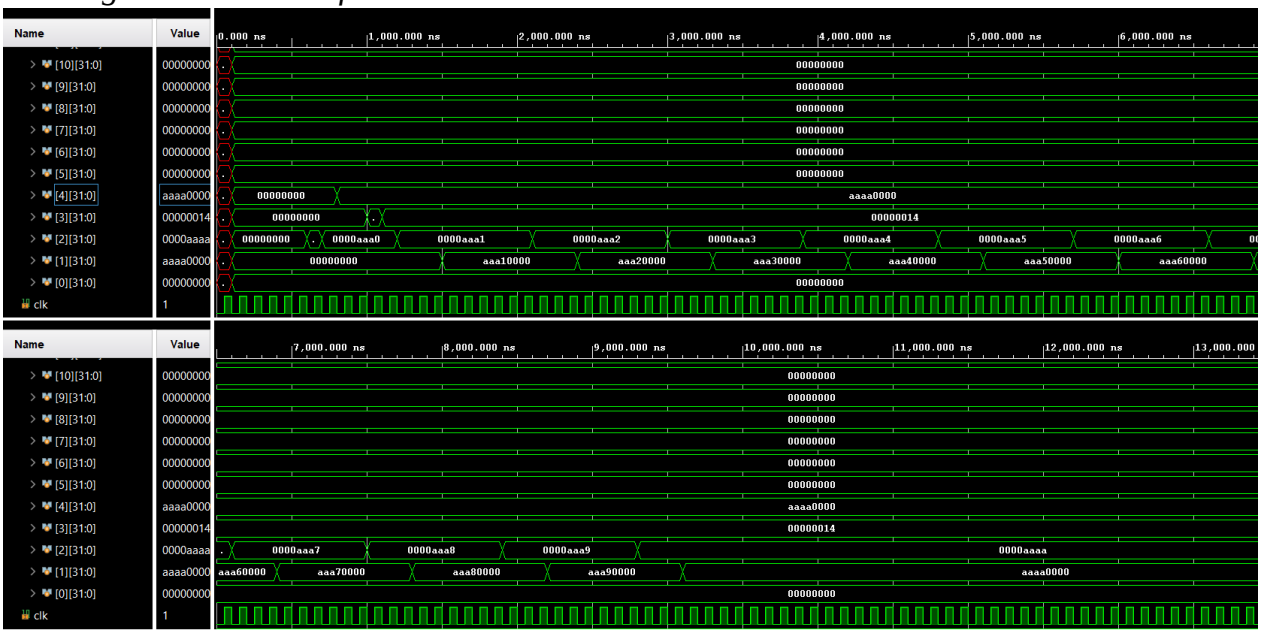
Với thiết kế register files ghi giá trị theo cạnh xuống xung clock để tránh việc bị lỗi khi ghi giá trị ngay vào cạnh lênh xung clock thì xung clock sau mới có giá trị bên trong register files (sẽ gây ra lỗi khi pipeline lệnh thứ n không thể truy cập được thanh ghi rd của lệnh thứ n - 3 nếu trùng).

3.2. Tập lệnh mô phỏng

```
addi x1, x0, 3      x1 = 3
addi x2, x1, 1      x2 = 4
sub x3, x2, x1      x3 = 1
xor x4, x1, x2      x4 = 7
sw   x3, 20(x1)     M[23] = 1
lw   x5, 20(x1)     x5 = 1
addi x0, x0, 0      nop
add  x6, x5, x1     x6 = 4
beq x1, x2, 16      not jump
addi x0, x0, 0      nop
addi x0, x0, 0      nop
addi x7, x2,  1     x7 = 5
addi x8, x7, 1      x8 = 6 x7 = 5, x8 = 6 => correct
beq x3, x5, 16      jump
addi x0, x0, 0      nop
addi x0, x0, 0      nop
addi x9, x8, 1      skip
addi x10, x9, 1     x10 = 1 x9 = 0, x10 = 1 => correct
addi x11, x10, 1    x11 = 2
addi x12, x11, 1    x12 = 3
addi x13, x12, 1    x13 = 4
addi x14, x13, 1    x14 = 5
addi x15, x14, 1    x15 = 6
```

4. Kết quả mô phỏng (lệnh trên lớp)

4.1. Register Files waveform



4.2. Tập lệnh mô phỏng

Mã lệnh	Mã hex	Ý nghĩa
lui x2, 0xa (+1)	0x0000b137	x2 = 0x0000b000
addi x2, x2, 0xaa0 (-4096)	0xaa010113	x2 = 0x0000aaa0
lui x4, 0xaaaa0	0xaaaa0237	x4 = 0xaaaa0000
lui x3, 0x0	0x000001b7	x3 = 0x0
ori x3, x3, 0x10	0x0101e193	x3 = 0x10
loop:		
addi x3, x3, 0x4	0x00418193	x3 = x3 + 4
addi x2, x2, 0x1	0x00110113	x2 = x2 + 1
sw x0, 0(x3)	0x0000a023	M[x3] = 0
sh x2, 2(x3)	0x00219123	M[x3 + 2] = x2
lw x1, 0(x3)	0x0001a083	x1 = M[x3]
nop	0x00000013	no operation
nop	0x00000013	no operation
bne x1, x4, loop (imm = -24)	0xfe4094e3	jump to loop if x1 != x4
nop	0x00000013	no operation
nop	0x00000013	no operation
done:		
jal x4, done (imm = 0)	0x0000026f	jump to 'done' label (endless loop)

5. Tổng hợp thiết kế

5.1. Sơ đồ Design Sources

5.2. File thiết kế tổng hợp

```
`timescale 1ns / 1ps
module yourcpu_pipeline(clk, reset_n, tb_addr, tb_inst);
    input           clk, reset_n;
    input [31:0]    tb_addr, tb_inst;
    //IF wire
    wire [31:0]     inst_ID, pc_ID, pcPlus4_ID;
    //ID wire
    wire            PCSel_IF, RegWEn_EX, ASel_EX, BSel_EX, MemRW_EX;
    wire [1:0]      WBSel_EX;
    wire [2:0]      funct3_EX;
    wire [3:0]      ALUSel_EX;
    wire [4:0]      AddrA_EX, AddrB_EX, AddrD_EX;
    wire [31:0]     pc_EX, pcPlus4_EX, DataA_EX, DataB_EX, imm_EX;
    //EX wire
    wire            RegWEn_MA, MemRW_MA;
    wire [1:0]      WBSel_MA;
    wire [2:0]      funct3_MA;
    wire [4:0]      AddrD_MA;
    wire [31:0]     ALU_Result_EX, ALU_Result_MA, ALU_DataB_MA, pcPlus4_MA;
    //MA wire
    wire            RegWEn_WB;
    wire [1:0]      WBSel_WB;
    wire [4:0]      AddrD_WB;
    wire [31:0]     DataR_WB;
    wire [31:0]     ALU_Result_WB;
    wire [31:0]     pcPlus4_WB;
    wire [31:0]     DataWB_WB;
    //forwarding wire
    wire [1:0]      fwdSelA, fwdSelB;
    IF_PIPELINE DUT_IF  (
                            .clk            (clk),
                            .reset_n        (reset_n),
                            .tb_addr_in     (tb_addr),
                            .tb_inst_in     (tb_inst),
                            .PCSel_in       (PCSel_IF),
                            .alu_in         (ALU_Result_EX),

                            .inst_out       (inst_ID),
                            .pc_out         (pc_ID),
                            .pcPlus4_out    (pcPlus4_ID)
                        );
```

```
    ID_PIPELINE DUT_ID  (
                            .clk            (clk),
                            .reset_n        (reset_n),
                            .inst           (inst_ID),
                            .pc_in          (pc_ID),
                            .pcPlus4_in     (pcPlus4_ID),
                            .RegWEn_in      (RegWEn_WB),
                            .AddrD_in       (AddrD_WB),
                            .DataD_in       (DataWB_WB),

                            .pc_out         (pc_EX),
                            .pcPlus4_out    (pcPlus4_EX),
                            .AddrA_out      (AddrA_EX),
                            .DataA_out      (DataA_EX),
                            .AddrB_out      (AddrB_EX),
                            .DataB_out      (DataB_EX),
                            .AddrD_out      (AddrD_EX),
                            .imm_out        (imm_EX),
                            .PCSel_out      (PCSel_IF),
                            .RegWEn_out     (RegWEn_EX),
                            .ASel_out       (ASel_EX),
                            .BSel_out       (BSel_EX),
                            .ALUSel_out     (ALUSel_EX),
                            .MemRW_out      (MemRW_EX),
                            .WBSel_out      (WBSel_EX),
                            .funct3_out     (funct3_EX)
                        );
    EX_PIPELINE DUT_EX  (
                            .clk            (clk),
                            .reset_n        (reset_n),
                            .pc_in          (pc_EX),
                            .pcPlus4_in     (pcPlus4_EX),
                            .DataA_in       (DataA_EX),
                            .DataB_in       (DataB_EX),
                            .imm_in         (imm_EX),
                            .ALU_Result_in  (ALU_Result_MA),
                            .WB_Result_in   (DataWB_WB),
                            .AddrD_in       (AddrD_EX),
                            .RegWEn_in      (RegWEn_EX),
                            .ASel_in        (ASel_EX),
                            .BSel_in        (BSel_EX),
                            .ALUSel_in      (ALUSel_EX),
                            .MemRW_in       (MemRW_EX),
                            .WBSel_in       (WBSel_EX),
                            .funct3_in      (funct3_EX),
```

```
                        .fwdSelA        (fwdSelA),
                        .fwdSelB        (fwdSelB),

                        .ALU_Result_EX  (ALU_Result_EX),

                        .RegWEn_out     (RegWEn_MA),
                        .MemRW_out      (MemRW_MA),
                        .WBSel_out      (WBSel_MA),
                        .funct3_out     (funct3_MA),
                        .ALU_Result_out (ALU_Result_MA),
                        .DataB_out      (ALU_DataB_MA),
                        .pcPlus4_out    (pcPlus4_MA),
                        .AddrD_out      (AddrD_MA)
                    );
    MA_PIPELINE DUT_MA  (
                        .clk            (clk),
                        .reset_n        (reset_n),
                        .RegWEn_in      (RegWEn_MA),
                        .MemRW_in       (MemRW_MA),
                        .WBSel_in       (WBSel_MA),
                        .funct3_in      (funct3_MA),
                        .ALU_Result_in  (ALU_Result_MA),
                        .DataW_in       (ALU_DataB_MA),
                        .pcPlus4_in     (pcPlus4_MA),
                        .AddrD_in       (AddrD_MA),

                        .RegWEn_out     (RegWEn_WB),
                        .WBSel_out      (WBSel_WB),
                        .DataR_out      (DataR_WB),
                        .ALU_Result_out (ALU_Result_WB),
                        .pcPlus4_out    (pcPlus4_WB),
                        .AddrD_out      (AddrD_WB)
                    );
    WB_PIPELINE DUT_WB(
                        .WBSel_in       (WBSel_WB),
                        .DataR_in       (DataR_WB),
                        .ALU_Result_in  (ALU_Result_WB),
                        .pcPlus4_in     (pcPlus4_WB),

                        .DataWB         (DataWB_WB)
                    );



FowardingUnit DUT_FowardingUnit(
```

```
                            .reset_n        (reset_n),
                            .RS1_EX         (AddrA_EX),
                            .RS2_EX         (AddrB_EX),
                            .RD_MA          (AddrD_MA),
                            .RD_WB          (AddrD_WB),
                            .RegWEn_MA      (RegWEn_MA),
                            .RegWEn_WB      (RegWEn_WB),

                            .fwdSelA        (fwdSelA),
                            .fwdSelB        (fwdSelB)
                        );
endmodule
```

5.3. Xử lý data hazard (forwarding)

Khi một lệnh không thể thực thi do các thanh ghi rs1 hoặc rs2 phụ thuộc vào data của thanh ghi rd của một vài lệnh trước đó và giá trị cần lấy chưa được lưu vào register files theo đúng chu kì của lệnh. Chúng ta cần phải lấy trước các giá trị này ở các tầng EX-MA (ví dụ ở loại lệnh R, I) hoặc MA-WB (ví dụ ở các lệnh loadword, loadbyte, ...). Và chúng ta sẽ tạo thêm một khối để kiểm tra địa chỉ của thanh ghi rd ở các tầng này có trùng với thanh ghi rs1 hoặc rs2 của lệnh hiện tại hay không. Nếu có, ta sẽ kiểm tra tiếp tín hiệu RegWEn ở các tầng này xem dữ liệu có được ghi không. Nếu có thì đây sẽ là giá trị mới mà chúng ta cần lấy cho lệnh cần thực thi hiện tại.

Ngõ ra của khối này sẽ là tín hiệu chọn cho bộ MUX3 được minh họa ở sơ đồ thiết kế được đề cập trước đó, với mục đích chọn được liệt kê kèm theo mã nguồn bên dưới.

```
`timescale 1ns / 1ps
//-----------SELECTION TABLE-------------
//  fwdSel(A/B)     *      ALU_in(A/B)  *
//      00          *       regfile     *       //default
//      01          *       alu_ma      *
//      10          *       ma_wb       *
//      11          *       32'bx       *
//---------------------------------------
module FowardingUnit(
    input           reset_n,
    input [4:0]     RS1_EX,
                    RS2_EX,
                    RD_MA,
                    RD_WB,
    input           RegWEn_MA,
                    RegWEn_WB,
    output reg[1:0] fwdSelA,
                    fwdSelB
    );
    always @(*) begin
        fwdSelA = 2'b00;
        fwdSelB = 2'b00;
        if(!reset_n) begin
            fwdSelA = 2'b00;
            fwdSelB = 2'b00;
        end
        else begin
            if((RegWEn_MA == 1'b1) && (RD_MA == RS1_EX) && (RD_MA != 0))
                fwdSelA = 2'b01;
            else if((RegWEn_WB == 1'b1) && (RD_WB == RS1_EX) && (RD_WB != 0))
                fwdSelA = 2'b10;
            if((RegWEn_MA == 1'b1) && (RD_MA == RS2_EX) && (RD_MA != 0))
```

```
                fwdSelB = 2'b01;
            else if((RegWEn_WB == 1'b1) && (RD_WB == RS2_EX) && (RD_WB != 0))
                fwdSelB = 2'b10;
        end
    end
endmodule
```

6. Thiết kế các chương trình mô phỏng thiết kế

6.1. Thiết kế top testbench (yourcpu_test_top_pipeline.sv)

Thông qua program test, chúng ta sẽ nạp lệnh vào DUT thông qua interface top_io, các lệnh sẽ được đưa vào bên trong imem ở tầng IF. Sau đó theo sự thay đổi của từng cạnh lênh xung clock, các lệnh sẽ được thực thi dần dần dựa trên địa chỉ của thanh ghi PC.

```
`timescale 1ns / 1ps
module yourcpu_test_top_pipeline();
    parameter simulation_cycle = 100;
    bit SystemClock;
    yourcpu_io  top_io  (SystemClock);
    test        t       (top_io.tb);

    yourcpu_pipeline dut    (
                                .reset_n(top_io.reset_n),
                                .clk    (SystemClock),
                                .tb_addr(top_io.addr),
                                .tb_inst(top_io.data)
                            );
    initial begin
        SystemClock = 0;
        forever begin
            #(simulation_cycle/2)
            SystemClock = !SystemClock;
        end
    end
endmodule
```

6.2. Thiết kế program test (test.sv)

Có thể load lệnh bằng cách tạo random thông qua task gen() và có thể gán lệnh trực tiếp thông qua task load_inst() vào DUT bằng cách truyền qua interface. Các giá trị này sẽ được khi vào REG trong imem.mem bên trong khối IF.

```
`timescale 1ns / 1ps
`ifndef TEST_SV
`define TEST_SV
`include "Packet.sv"

program automatic test (yourcpu_io.tb yourcpu_io);
    integer file_descriptor;
    logic [31:0] inst_arr[1023:0];
    static int pkts_generated = 0;

    initial begin
        reset();
        //gen();
        load_inst();
        driver1();
        #4000 $finish;
    end
    task load_inst();
        pkts_generated  = 16;
        //reset
        inst_arr[0]     = 32'hxxxxxxxx;
        inst_arr[1]     = 32'h0000b137;     //lui x2, 0xa (+1)
        inst_arr[2]     = 32'haa010113;     //addi x2, x2, 0xaa0 (-4096)
        inst_arr[3]     = 32'haaaa0237;     //lui x4, 0xaaaa0
        inst_arr[4]     = 32'h000001b7;     //lui x3, 0x0
        inst_arr[5]     = 32'h0101e193;     //ori x3, x3, 0x10

        //loop:
        inst_arr[6]     = 32'h00418193;     //addi x3, x3, 0x4
        inst_arr[7]     = 32'h00110113;     //addi x2, x2, 0x1
        inst_arr[8]     = 32'h0000a023;     //sw x0, 0(x3)
        inst_arr[9]     = 32'h00219123;     //sh x2, 2(x3)
        inst_arr[10]    = 32'h0001a083;     //lw x1, 0(x3)
        inst_arr[11]    = 32'h00000013;     //nop
        inst_arr[12]    = 32'h00000013;     //nop
        inst_arr[13]    = 32'hfe4094e3;     //bne x1, x4, -24
        inst_arr[14]    = 32'h00000013;     //nop
        inst_arr[15]    = 32'h00000013;     //nop
        //done:
        inst_arr[16]     = 32'h0000026f;     //jal x4, done
```

```
    endtask

    task reset();
        yourcpu_io.reset_n = 1'b0;
        #100
        yourcpu_io.reset_n <= 1'b1;
    endtask

    task gen();
        Packet pkt2send;
        repeat (5) begin
            pkt2send = new($sformatf("Packet[%0d]", pkts_generated));
            if(!pkt2send.randomize()) begin
                $display("Error: Randomization failed");
                $finish;
            end
            pkt2send.gen();
            pkt2send.display();
            inst_arr[pkts_generated] = pkt2send.inst;
            pkts_generated = pkts_generated + 1;
        end
    endtask

    task driver1();
        integer i;
        for (i = 0; i < pkts_generated; i = i + 1) begin
            $display("driver[%0d]", i);
            #1
            yourcpu_io.addr = i;
            yourcpu_io.data = inst_arr[i];
        end
    endtask
    task driver2();
        $writememh("D:\\Vivado\\CE409_Labs\\Lab45\\CE409_Lab45_21521391_DaoPhuocTai\\CE409_L
ab45_21521391_DaoPhuocTai.srcs\\sources_1\\new\\imem.mem", inst_arr);
    endtask
endprogram: test
`endif
```

6.3. Thiết kế interface (yourcpu_io.sv)

```
`timescale 1ns / 1ps
interface yourcpu_io(input bit clk);
    logic           reset_n;
    logic [31:0]    addr;
    logic [31:0]    data;
    clocking cb @(posedge clk);
        output reset_n;
        output addr;
        output data;
    endclocking
    modport tb(clocking cb, output reset_n, addr, data);
endinterface
```

6.4. Thiết kế class tạo lệnh random và load lệnh(Packet.sv)

Một packet sẽ tương ứng với 1 lệnh 32 được tạo ra, bao gồm các method:

- New(): Tạo ra 1 đối tượng mới với tên được tham chiếu.
- Gen(): Tạo ra lệnh bằng cách random có ràng buộc (tập lệnh I, R).
- setInst(): Tạo ra lệnh bằng cách tham chiếu mã lệnh 32bit từ bên ngoài.

```
`timescale 1ns / 1ps
`ifndef PACKET
`define PACKET
class Packet;
    rand bit [6:0]  opcode;
    rand bit [4:0]  rs1, rs2, rd;
    rand bit [11:0] imm;
    rand bit [2:0]  funct3;
    rand bit [6:0]  funct7;
    string          name;
    bit [31:0]      inst;
    constraint opcode_constraint {opcode inside {7'b0010011, 7'b0110011};}
    constraint funct7_constraint1 {
        if(opcode == 7'b0110011)
            if((funct3 != 3'b000) && (funct3 != 3'b101))
                funct7 == 7'b0000000;
            else
                funct7 inside {7'b0100000, 7'b0000000};
        else
            if(funct3 == 3'b001)
                imm inside {[0:31]}; //slli
            else if(funct3 == 3'b101)
                imm[11:5] inside {7'b0000000, 7'b0100000};
    }
    extern function new(string name);
```

```
    extern function gen();
    extern function void display(string prefix = "NOTE");
    extern function void setInst(bit [31:0] newInst);
endclass

function Packet::new(string name);
    this.name <= name;
endfunction

function Packet::gen();
    if(this.opcode == 7'b0010011)
        this.inst <= {this.imm, this.rs1, this.funct3, this.rd, this.opcode};
    else
        this.inst <= {this.funct7, this.rs2, this.rs1, this.funct3, this.rd, this.opcode};
endfunction

function void Packet::display(string prefix = "NOTE");
    //$display("%s: Packet displayed", prefix);
    if(this.opcode == 7'b0010011) begin
        $display("Type: I - Name: %s", this.name);
        $display("Instruction: %h", this.inst);
    end
    else begin
        $display("Type: R - Name: %s", this.name);
        $display("Instruction: %h", this.inst);
    end
endfunction

function void Packet::setInst(bit [31:0] newInst);
    this.inst = newInst;
endfunction
`endif
```

7. Mô phỏng thiết kế pipeline chạy lệnh random

```
# run 2000ns
Type: R - Name: Packet[0]
Instruction: 01b795b3
Type: R - Name: Packet[1]
Instruction: 006bc033
Type: I - Name: Packet[2]
Instruction: f908bb13
Type: R - Name: Packet[3]
Instruction: 0194b733
Type: I - Name: Packet[4]
Instruction: c4d4c093
driver[0]
driver[1]
driver[2]
driver[3]
driver[4]
```

Các lệnh sau khi được giải mã:

0x01b795b3: sll x11, x15, x27 => x11 = 0 | do x15 ban đầu = 0

0x006bc033: xor x0, x23, x6 => x0 = 0 | tương tự

0xf908bb13: sltiu x22, x17, -112 => x22 = 1 | do x17 = 0 < -112(số không dấu)

0x0194b733: sltu x14, x9, x25 => x14 = 0 | do x9 = x25 = 0

0x c4d4c093: xori x1, x9, -947 => x1 = -947 | -947 ^ 0 = -947