BÁO CÁO ĐỒ ÁN CUỐI KÌ
INSTRUCTION OF MIPS

GVHD: Hồ Ngọc Diễm

Họ tên: Trần Triều Trung

MSSV: 21522727

MỤC LỤC

Bắt đầu

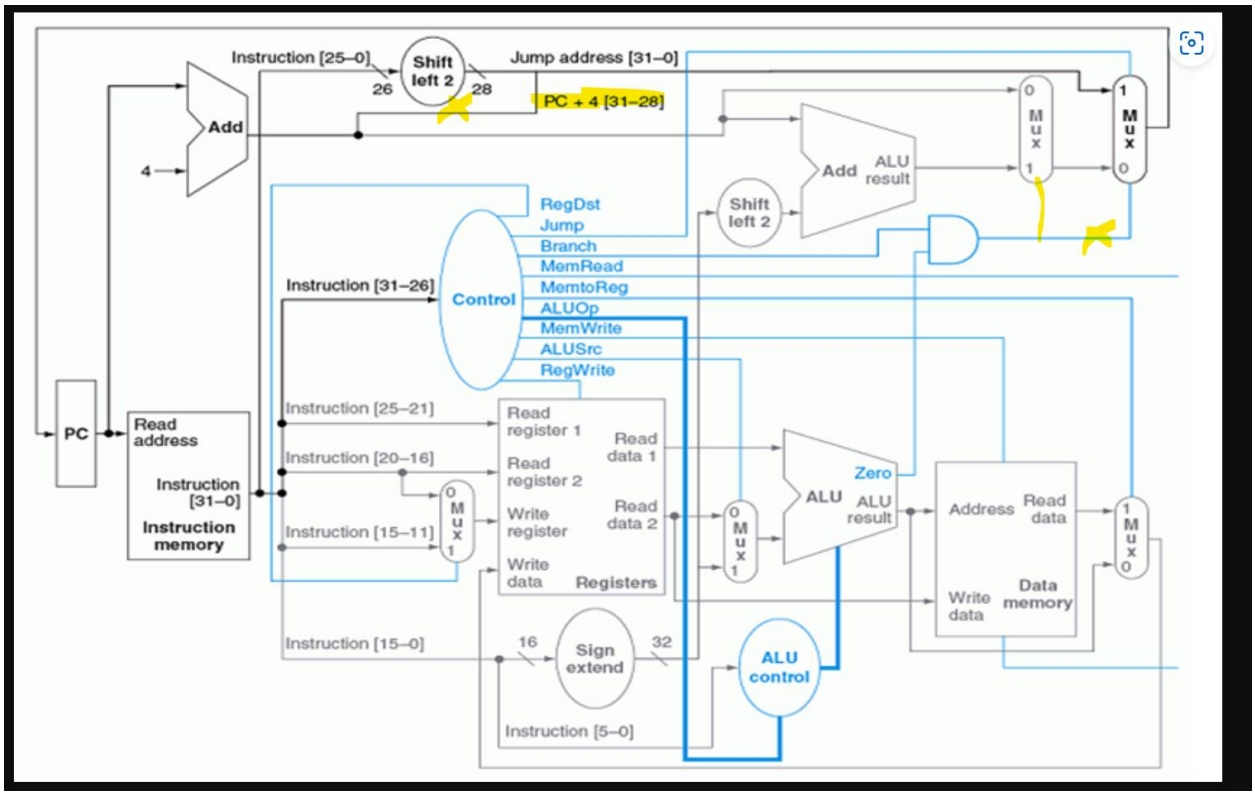
I. Giới thiệu về đề tài

Đề tài được sử dụng theo kiến trúc MIPS và tập lệnh của MIPS.

Sử dụng một số lệnh như sau:

Add	**Addi**	**Sub**	**Sll**
And	**Lw**	**Beq**	**Srl**
Or	**Sw**	**Slt**	**J**

Kiến trúc được mô tả như hình sau:



II. Các thanh ghi

1. Program Counter (PC)

Sẽ là 1 reg để lưu địa chỉ hiện tại của chương trình như sau:

```
module Pc(clk, PcNext, Pcn);
input clk;
input [31:0] PcNext;
output reg [31:0] Pcn;

always @(posedge clk) begin
  Pcn = PcNext;
end

initial begin
  Pcn = 0;
end

endmodule
```

Với khởi tạo ban đầu cho chương trình biết ở địa chỉ 0x00000000

2. Instruction Memory (IM)

Ở đây là nơi lưu địa chỉ lệnh nhập từ file txt ở bên ngoài đã chuyển sang mã máy là hex.

```
module InstructionMemory(RA, RD);
input [31:0] RA;
output [31:0] RD;

reg [31:0] mem[0:1023];

assign RD = mem[RA[31:2]];

initial begin
    $readmemh("memfile.txt",mem);
  end
endmodule
```

3. Register File (RF)

Thanh ghi này là nơi lưu giá trị của thanh ghi gồm 32 thanh ghi có độ dài 1 word.

```
module RegisterFile(
input clk,
    input [4:0] ReadAddress1,
    input [4:0] ReadAddress2,
    input [4:0] WriteAddress,
    input [31:0] WriteData,
    output [31:0] ReadData1,
    output [31:0] ReadData2,
    input ReadWriteEn
);

    reg [31:0] registers [0:31];

    assign ReadData1 = registers[ReadAddress1];
    assign ReadData2 = registers[ReadAddress2];

    always @(posedge clk) begin
        if (ReadWriteEn) begin
            registers[WriteAddress] = WriteData;
      end
  end

  initial begin
    registers[0] = 32'd0;
    registers[1] = 32'd0;
    registers[2] = 32'd0;
  end
endmodule
```

4. ALU

Tại đây gồm 2 giai đoạn:

- Decode ALUOp từ Control Unit và Inst[5:0] để ra được ALUcontrol
- Sử dụng ALUcontrol để giải quyết

Giai đoạn 1:

Định nghĩa các opcode của các lệnh dùng ở I

```
wire R;
assign R = (alu_control == 2'b10) ? 1'b1: 1'b0;

wire lw; // lw, sw, addi
assign lw = (alu_control == 2'b00) ? 1'b1: 1'b0;

parameter [5:0] And = 6'h24;
parameter [5:0] Add = 6'h20;
parameter [5:0] Sub = 6'h22;
parameter [5:0] Or = 6'h25;
parameter [5:0] Slt = 6'h2a;
parameter [5:0] Sll = 6'h00;
parameter [5:0] Srl = 6'h02;
```

Bảng sự thật:

add	0010
sub	0110
lw	0010
sw	0010
beq	0110
and	0000
or	0001

slt	0111
sll	0100
srl	0111

Code:

```
module alu_control_decode(inst , alu_control,control);
input [5:0] inst;
input [1:0] alu_control;
output [3:0] control;

wire R;
assign R = (alu_control == 2'b10) ? 1'b1: 1'b0;

wire lw; // lw, sw, addi
assign lw = (alu_control == 2'b00) ? 1'b1: 1'b0;

parameter [5:0] And = 6'h24;
parameter [5:0] Add = 6'h20;
parameter [5:0] Sub = 6'h22;
parameter [5:0] Or = 6'h25;
parameter [5:0] Slt = 6'h2a;
parameter [5:0] Sll = 6'h00;
parameter [5:0] Srl = 6'h02;

assign control = lw ? 4'b0010:
                 (alu_control == 2'b01) ? 4'b0110:
                 (R && inst == Add) ? 4'b0010:
                 (R && inst == Sub) ? 4'b0110:
                 (R && inst == And) ? 4'b0000:
                 (R && inst == Or) ? 4'b0001:
                 (R && inst == Slt) ? 4'b0111:
                 (R && inst == Sll) ? 4'b0100:
                 (R && inst == Srl) ? 4'b0111: 4'b1111;


endmodule
```

Giai đoạn 2:

Xử lí:

```
module alu_control_lab (a, b, control, is0, result);
input [31:0] a,b;
input [3:0] control;
output reg is0;
output [31:0] result;

  wire [31:0] ss;
  assign ss = (a<b) ? 32'd1: 32'd0;

  assign result = (control == 4'b0010) ? (a+b):
            (control == 4'b0110) ? (a-b):
            (control == 4'b0000) ? (a&b):
            (control == 4'b0001) ?  (a|b):
            (control == 4'b0111) ? ss :
            (control == 4'b0100) ? a << b:
            (control == 4'b0111) ? a >> b: 32'd0;

  always @(*) begin
    if(control == 4'b0110)
      if (result == 0)
        is0 = 1'b1;
      else is0 = 1'b0;
end

endmodule
```

5.Data Memory

Nơi lưu trữ gồm 1024 thanh ghi 32 bits lưu trữ

MemRead, MemWrite cho phép đọc, ghi

```
module DataMemory_mips(
    clk, Address, WriteData, ReadData, WriteEn, ReadEn
);
  input clk;
    input [31:0] Address;
    input [31:0] WriteData;
    output reg [31:0] ReadData;
    input WriteEn;
    input ReadEn;

    reg [31:0] memory[0:1023];

    always @(posedge clk) begin
        if (WriteEn)
            memory[Address] <= WriteData;
      if (ReadEn)
            ReadData <= memory[Address];
  end

endmodule
```

6. Control Unit

Đây là cần thiết cấp đầu ra cho các thành phần trong Datapath

Bảng sự thật:

	R-type	Lw	Sw	Beq	Addi	J
Jump	0	0	0	0	0	1
RegDst	1	1	0	0	0	0
ALUSrc	0	1	1	0	1	0
MemToReg	0	1	1	0	0	0
RegWrite	1	1	1	0	1	0
MemRead	0	0	1	0	0	0
MemWrite	0	0	0	0	0	0
Branch	0	0	0	1	0	0
ALUOp	10	0	0	1	0	0

Code:

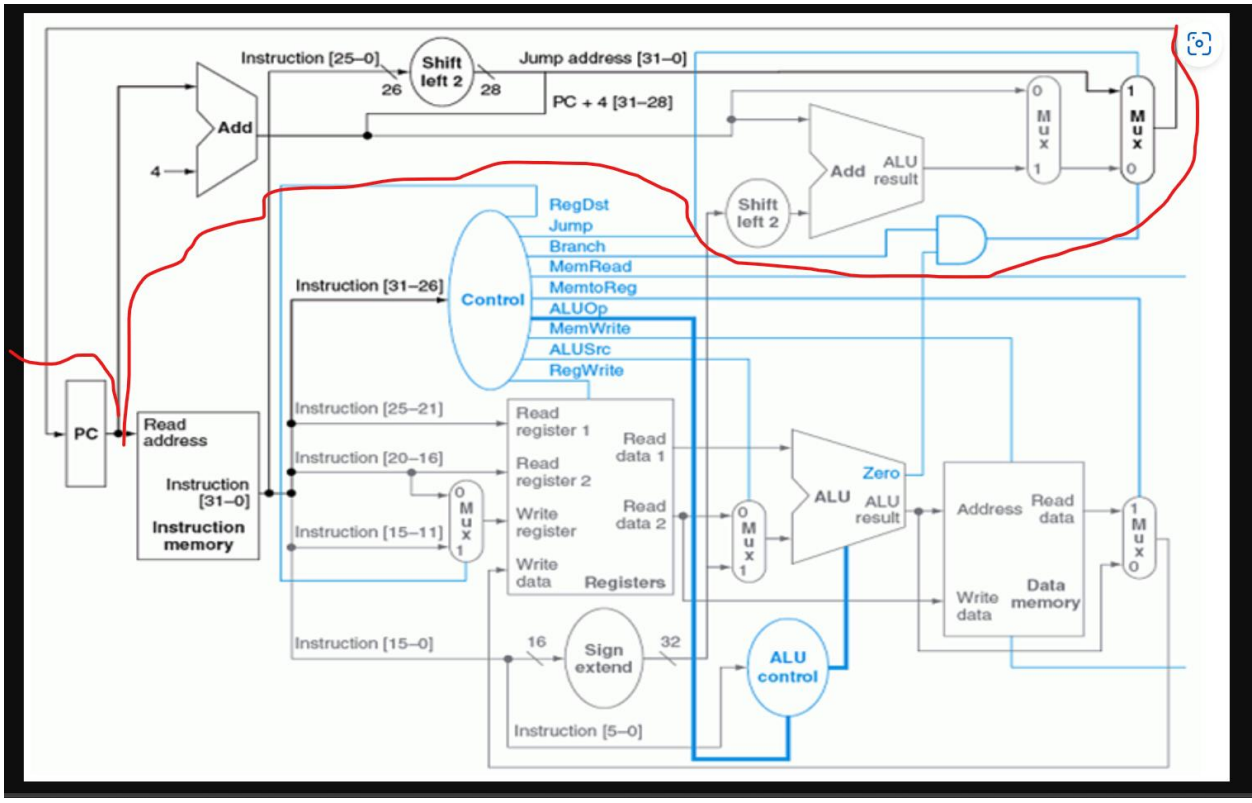
```
module control(opcode, RegDst, MemRead, MemWrite, MemToReg, ALUOp, ALUSrc, RegWrite, Branch, Jump);
input [5:0] opcode;
output RegDst, MemRead, MemWrite, MemToReg, ALUSrc, RegWrite, Branch, Jump;
output [1:0] ALUOp;

parameter R = 6'h00;
parameter lw = 6'h23;
parameter sw = 6'h2b;
parameter beq = 6'h04;
parameter addi = 6'h08;
parameter J  = 6'h02;

wire [9:0] c;
assign c = (opcode == R) ? 10'h122 :
           (opcode == lw) ? 10'h0f0 :
           (opcode == sw) ? 10'h1c8 :
           (opcode == beq) ? 10'h005 :
           (opcode == addi) ? 10'h0a0 :
           (opcode == J) ? 10'h200 : 10'd0;

assign Jump = c[9];
assign RegDst = c[8];
assign ALUSrc = c[7];
assign MemToReg = c[6];
assign RegWrite = c[5];
assign MemRead = c[4];
assign MemWrite = c[3];
assign Branch = c[2];
assign ALUOp = c[1:0];


endmodule
```

7.Jump Address, PCNext

Ở lệnh Jump trong MIPS sẽ cần 26 bit [25:0] để đẩy lên địa chỉ PC

```
wire [31:0] JAdd;
assign JAdd = ins << 2;
```

PCNext là nơi ta xử lí phần khoanh đỏ:




```
mux_32bits mux8(Addr, JAdd, jump, Pc);

module PcNe(RA, sf, Branch, is0, PcNext);
input [31:0] RA, sf;
input Branch, is0;
output [31:0] PcNext;

wire [31:0] se;
assign se = sf << 2;

wire Br;
assign Br = is0 & Branch;

wire [31:0] addRA, addRA2;
assign addRA = RA + 32'd4;

assign addRA2 = addRA + se;

mux_32bits mux6(addRA, addRA2, Br, PcNext);


endmodule
```

III. Run

Đầu tiên cần viết mã code bằng assembly

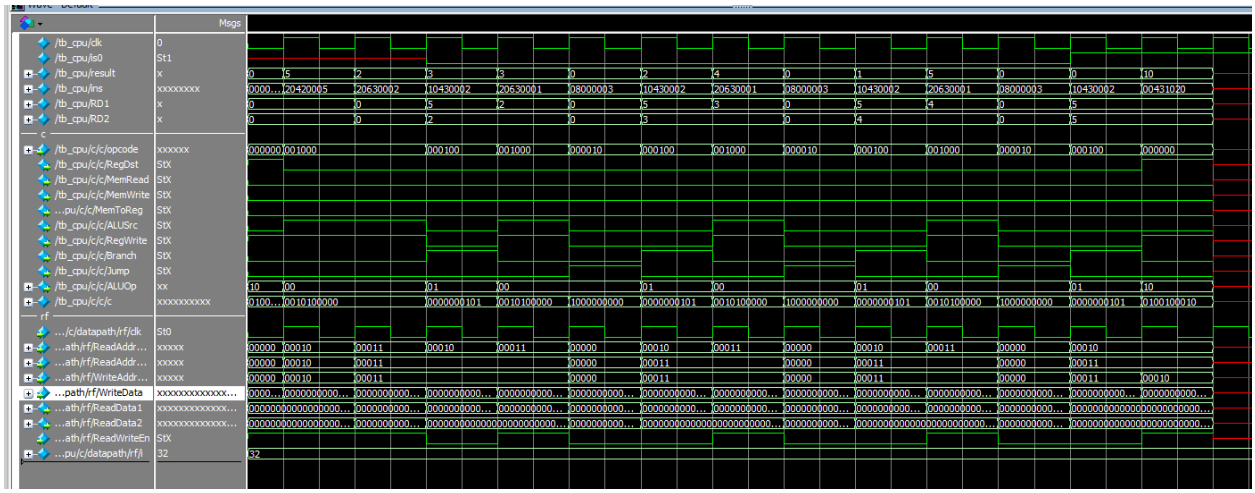
Ví dụ:

```
        addi $2, $2, 5
        addi $3, $3, 2
M:
        beq $2, $3, X
        addi $3, $3, 1
        j M
X:
       add $2, $2, $3
```

Sau đó chuyển qua mã máy bằng MARS

```
        00000000
        20420005
        20630002
        10430002
        20630001
        08000003 (Lưu ý khác MARS)
       00431020
```

Waveform:



----------HẾT----------