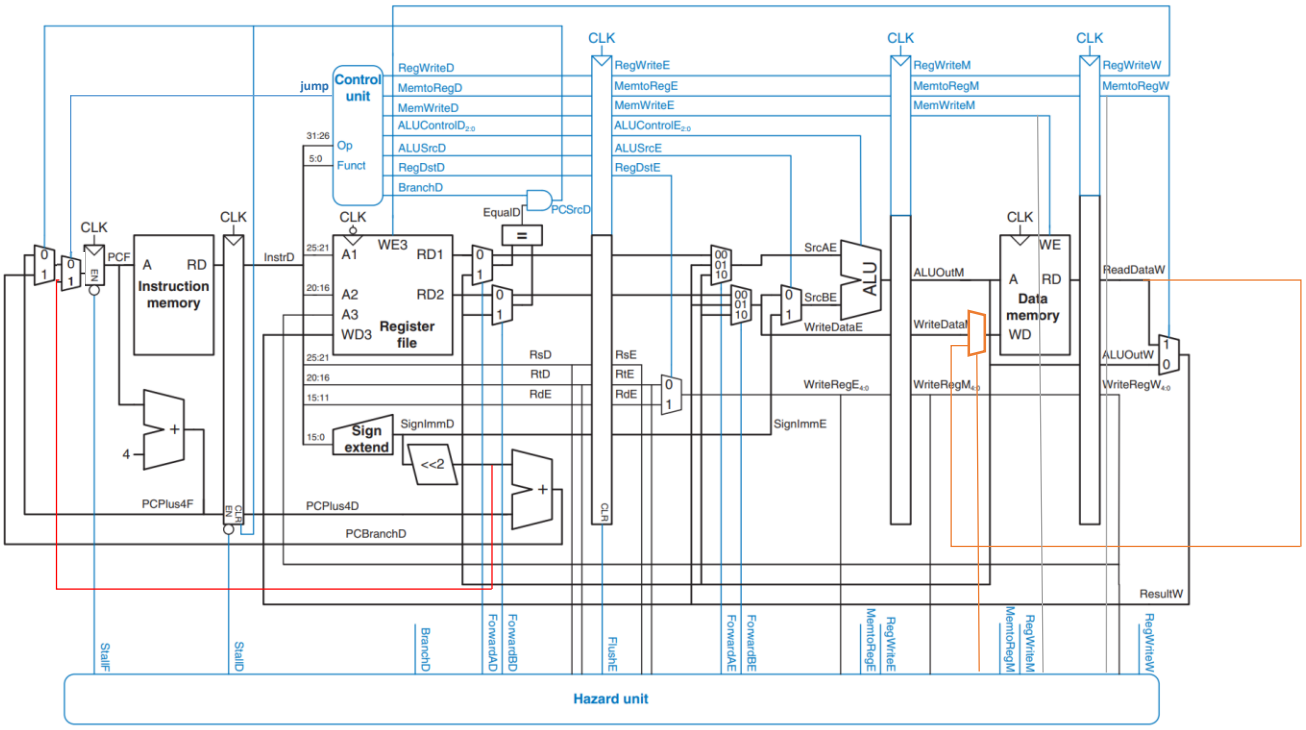
بسمه تعالی

محمدرضا عسگری شکوه
شماره دانشجویی : ۹۹۲۴۳۱۰۸

CA Project
۱۴۰۱/۰۴/۱۲

ابتدا لازم است datapath تغییر کند تا بتوانیم دستور sw که پس از lw میاید و data dependency دارند را هندل کنیم:



برای اینکه متوجه دستور sw پس از lw شویم لازم است شروط زیر توسط Hazard Unit شناسایی شود:

۱. سیگنال MemToReg برای دستوری که در WB قرار دارد باید فعال باشد.
۲. سیگنال MemWrite باید برای دستوری که در Mem قرار دارد فعال باشد.
۳. همچنین باید رجیستری که دستور lw میخواهد در آن بنویسد، با رجیستری که دستور sw میخواهد از آن بخواند یکسان باشد.

در صورت ارضا شدن این شروط، لازم است تا mux ای که در ورودی data memory قرار دارد، داده ای که دستور lw میخواهد در رجیستر مورد نظر بنویسد را در حافظه ذخیره کند.

ابتدا کد رجیستر های پایپ لاین را مشاهده میکنیم: در هر رجیستر لازم است تا در لبه های بالارونده هر کلاک محتوا های ورودی و به خروجی ها متصل شوند.البته اگر stall فعال نباشد یا flush نیاز نباشد.

رجیستر ۱

```
timescale 1ns / 1ps

module Reg_IF(clk,clr,stall,insF,pcp4,insD,out_pcp4
    );
input clk,clr,stall;
input[31:0] insF,pcp4;
output reg[31:0] insD,out_pcp4;

always@(posedge clk)begin
    if(clr)begin
        insD = 0;
        out_pcp4= 0;
        end
    else if(~stall)begin
        insD = insF;
        out_pcp4 = pcp4;
        end

end

endmodule
```

رجیستر ۲

```
`timescale 1ns / 1ps

module Reg_ID(clk,clr,regwriD,memtoregD,memwriD,aluconD,alusrcD,regdstD,r1D,r2D,rsD,rtD,rdD,immD,regwriE,memtoregE,memwriE,aluconE,alusrcE,regdstE,r1E,r2E,rsE,rtE,rdE,imm
    );
input clk,clr,regwriD,memtoregD,memwriD,alusrcD,regdstD;
input [2:0] aluconD;
input[31:0] r1D,r2D,immD;
input[4:0] rsD,rtD,rdD;
output reg regwriE,memtoregE,memwriE,alusrcE,regdstE;
output reg[2:0] aluconE;
output reg[31:0] r1E,r2E,immE;
output reg[4:0] rsE,rtE,rdE;

always@(posedge clk)begin
    if(clr)begin
        regwriE=0;memtoregE=0;memwriE=0;aluconE=0;alusrcE=0;regdstE=0;r1E=0;r2E=0;immE=0;rsE=0;rtE=0;rdE=0;
        end
    else begin
        regwriE=regwriD;memtoregE=memtoregD;memwriE=memwriD;aluconE=aluconD;alusrcE=alusrcD;regdstE=regdstD;r1E=r1D;r2E=r2D;immE=immD;rsE=rsD;rtE=rtD;rdE=rdD;
    end

end

endmodule
```

رجیستر ۳

```
`timescale 1ns / 1ps

module Reg_EX(clk,regwriE,memtoregE,memwriE,aluoutE,wridataE,wriregE,regwriM,memtoregM,memwriM,aluoutM,wridataM,wriregM
    );
input clk,regwriE,memtoregE,memwriE;
input[31:0] aluoutE,wridataE;
input[4:0] wriregE;
output reg regwriM,memtoregM,memwriM;
output reg[31:0] aluoutM,wridataM;
output reg[4:0] wriregM;

always@(posedge clk)begin
    regwriM=regwriE;memtoregM=memtoregE;memwriM=memwriE;aluoutM=aluoutE;wridataM=wridataE;wriregM=wriregE;
end

endmodule
```

رجیستر ۴

```
`timescale 1ns / 1ps

module Reg_M(clk,regwriM,memtoregM,readdataM,aluoutM,wriregM,regwriW,memtoregW,readdataW,aluoutW,wriregW
    );
input clk,regwriM,memtoregM;
input[31:0] readdataM,aluoutM;
input[4:0] wriregM;
output reg regwriW,memtoregW;
output reg[31:0] readdataW,aluoutW;
output reg[4:0] wriregW;

always@(posedge clk)begin
    regwriW=regwriM;memtoregW=memtoregM;readdataW=readdataM;aluoutW=aluoutM;wriregW=wriregM;
end


endmodule
```

سپس کد قسمت هایی که در کتاب موجود نبودند را میبینیم:

```
`timescale 1ns / 1ps
module ALU(a,b,control,out);
input[31:0] a,b;
input[2:0] control;
output reg[31:0] out;

always@(a or b)begin
    if(control==3'b10)
        out=a+b;
    else if(control==3'b11)
        out=b;
    else if(control==3'b110)
        out=a-b;
    else if(control==3'b0)
        out=a & b;
    else if(control==3'b1)
        out=a | b;
    else if(control==3'b111)
        out=(a<b)? 1:0;
end

endmodule
```

کد ALU : که یک حالت جدید برای پشتیبانی از دستور addi در آن قرار داده ام که ورودی دوم که imm است را به خروجی ببرد.

```
timescale 1ns / 1ps

module ALU_control(Aluop,func,out);
input[1:0] Aluop;
input[5:0] func;
output reg[2:0] out;

always@(Aluop or func)begin
    if(Aluop==0)
        out=3'b010;
    else if(Aluop==1)
        out=3'b110;
    else if(Aluop==2'b11)
        out=3'b11;
    else begin
        if(func[3:0]==4'b0)
            out=3'b010;
        else if(func[3:0]==4'b10)
            out=3'b110;
        else if(func[3:0]==4'b100)
            out=3'b0;
        else if(func[3:0]==4'b101)
            out=3'b001;
        else if(func[3:0]==4'b1010)
            out=3'b111;
    end

end

endmodule
```

کد ALU control : همچنین این واحد برای کنترل ALU است که دو بیت از واحد کنترلی اصلی میگیرد و با استفاده از آن و func موجود در دستور، ALU را کنترل میکند.

کد واحد کنترلی اصلی:

برای دستور addi تنها لازم است ALUopc برای این دستور هر دو ۱ شوند.

```
timescale 1ns / 1ps
module Control(opcode,RegWr,MemtoReg,MemWr,ALUopc,ALUsrc,RegDst,Branch);
input[5:0] opcode;
output reg RegWr,MemtoReg,MemWr,ALUsrc,RegDst,Branch;
output reg[1:0] ALUopc;

always@(opcode)begin
    RegWr=(opcode==0)? 1:
            (opcode==6'b100011)? 1:
            (opcode==6'b1000)? 1:0;
    MemtoReg=(opcode==6'b100011)? 1:0;
    MemWr=(opcode==6'b101011)? 1:0;
    ALUsrc=(opcode==6'b100011)? 1:
            (opcode==6'b101011)? 1:
            (opcode==6'b1000)? 1:0;
    RegDst=(opcode==0)? 1:0;
    Branch=(opcode==6'b100)? 1:0;
    ALUopc[0]=(opcode==6'b100)? 1:
                (opcode==6'b1000)?1:0;
    ALUopc[1]=(opcode==0)? 1:
                (opcode==6'b1000)?1:0;
end

endmodule
```

کد Hazard unit :در ابتدا سیگنال های کنترلی رجیستر های پایپ لاین و select ماکس ها باید صفر باشند تا دستور ابتدایی به درستی اجرا شوند. سپس با توجه به ورودی ها، مقادیر forward ها تنظیم میشوند.

```
`timescale 1ns / 1ps

module Hazard_Unit(input regwriteW, memtoregW,  regwriteM,  memtoregM,  memwriteM,  regwriteE,  memtoregE,  branchD,
                   input[4:0] writeregW,   writeregM,  writeregE,  rtE,    rsE,    rtD,    rsD,
                   output  reg forwardM,   flushE, forwardAD,  forwardBD,  stallD, stallF,
                   output  reg [1:0] forwardAE,    forwardBE
    );

    initial begin
        forwardM = 0;
        flushE = 0;
        forwardAD = 0;
        forwardBD = 0;
        stallD = 0;
        stallF = 0;
        forwardAE = 0;
        forwardBE = 0;
        end

     always@(regwriteW or memtoregW or regwriteM or memtoregM or memwriteM or regwriteE or memtoregE or branchD or writeregW or writeregM or writeregE or rtE or rsE or rt
            forwardM=(writeregW==writeregM & memtoregW==1 & memwriteM==1)? 1:0;

            forwardAE=(regwriteM==1 & writeregM!=0 & writeregM==rsE)? 10:
                      (regwriteW==1 & writeregW!=0 & writeregM!=rsE & writeregW==rsE)? 1:0;
            forwardBE=(regwriteM==1 & writeregM!=0 & writeregM==rtE)? 10:
                      (regwriteW==1 & writeregW!=0 & writeregM!=rtE & writeregW==rtE)? 1:0;

            forwardAD=(regwriteM==1 & writeregM==rsD)?1:0;

            forwardBD=(regwriteM==1 & writeregM==rtD)?1:0;

            stallD=(memtoregE==1 & regwriteE==1 & (writeregE==rsD | writeregE==rtD) )? 1:0;
            stallF=(stallD==1)? 1:0;
            flushE=(stallD==1)? 1:0;
     end
endmodule
```

در انتها تمام قسمت های موجود در فایل **mips** به یکدیگر متصل میشوند.

```
`timescale 1ns / 1ps
module MIPS(input clk);

wire pcsrcD;
wire[31:0] pcplus4F,pcplus4D;
wire[31:0] pcBranchD;
wire[31:0] pcinput,pcF;

Mux2to1#(32) mux_pc(pcplus4D,   pcBranchD,  pcsrcD, pcinput);

wire stallF;
Program_counter pc(clk, stallF, pcinput,    pcF);

wire[31:0] insF;
Ins_mem instruction_memory(pcF, insF);

Adder addpc(pcF,    4,  pcplus4F);

wire stallD;
wire[31:0] insD;
Reg_IF reg_if_id(clk,   pcsrcD, stallD, insF,   pcplus4F,   insD,   pcplus4D);

wire regwriteW;
wire[4:0] writeregW;
wire[31:0] resultW,readdata1D,readdata2D;
regfile register_file(clk,  regwriteW,  insD[25:21],    insD[20:16],    writeregW,  resultW,    readdata1D,readdata2D);

wire[31:0] aluoutM,eq_srcA,eq_srcB;
wire ForwardAD,ForwardBD;
Mux2to1#(32) mux_forwardAD(readdata1D, aluoutM, ForwardAD,  eq_srcA);
Mux2to1#(32) mux_forwardBD(readdata2D, aluoutM, ForwardBD,  eq_srcB);

wire equalD,BranchD;
assign equalD=(eq_srcA==eq_srcB)? 1:0;
assign pcsrcD=(equalD & BranchD)? 1:0;
```

```
wire regwriteD,memtoregD,memwriteD,alusrcD,regdstD;
wire[1:0] aluopc;
Control control_unit(insD[31:26],   regwriteD,  memtoregD,  memwriteD,  aluopc, alusrcD,    regdstD,    BranchD);

wire[2:0] alucontrolD;
ALU_control alucontrol(aluopc,  insD[5:0],  alucontrolD);

wire[31:0] signimmD;
signextend se(insD[15:0],   signimmD);

wire[31:0] wirebranch_pc;
SL2 shiftlefttwo(signimmD,  wirebranch_pc);

Adder add_branch(wirebranch_pc, pcplus4D,   pcBranchD);


wire flushE;

wire regwriteE,memtoregE,memwriteE,alusrcE,regdstE;
wire[2:0] aluconE;
wire[4:0] rsE,rtE,rdE;
wire[31:0] immE,readdata1E,readdata2E;
Reg_ID register_id_ex(clk,  flushE, regwriteD,  memtoregD,  memwriteD,  alucontrolD,    alusrcD,    regdstD,    readdata1D, readdata2D,
    insD[25:21],    insD[20:16],    insD[15:11],    signimmD,   regwriteE,  memtoregE,  memwriteE,  aluconE,    alusrcE,    regdstE,
    readdata1E, readdata2E, rsE,    rtE,    rdE,    immE);

wire[4:0] writeregE;
Mux2to1#(5) mux_regdst(rtE, rdE,    regdstE,    writeregE);


wire[1:0] ForwardAE;
wire[31:0] srcAE;
Mux3to1#(32) mux_forwardAE(readdata1E,  resultW,    aluoutM,    ForwardAE,  srcAE);


```

```
Mux2to1#(32) mux_alusrc(writedataE, immE,   alusrcE,    srcBE);

wire[31:0] aluoutE;
ALU alu(srcAE,  srcBE,  aluconE,    aluoutE);

wire regwriteM,memtoregM,memwriteM;
wire[31:0] writedataM;
wire[4:0] writeregM;
Reg_EX register_ex_mem(clk, regwriteE,  memtoregE,  memwriteE,  aluoutE,    writedataE, writeregE,  regwriteM,  memtoregM,
        memwriteM,  aluoutM,    writedataM, writeregM);


wire[31:0] readdataW,output_mux_mem;
wire ForwardM;
Mux2to1#(32) mux_memory(writedataM, readdataW,  ForwardM,   output_mux_mem);

reg[31:0] save_reg;
always@(posedge clk)
    save_reg <= output_mux_mem;


wire[31:0] readdataM;
Data_mem data_memory(clk,   memwriteM,  aluoutM,    save_reg,   readdataM);


wire memtoregW;
wire[31:0] aluoutW;
Reg_M register_mem_wb(clk,  regwriteM,  memtoregM,  readdataM,  aluoutM,    writeregM,  regwriteW,  memtoregW,  readdataW,  aluoutW,    writeregW);


Mux2to1#(32) muxWB(aluoutW, readdataW,  memtoregW,  resultW);

Hazard_Unit hazard_unit(regwriteW,  memtoregW,  regwriteM,  memtoregM,  memwriteM,  regwriteE,  memtoregE,  BranchD,
                writeregW,  writeregM,  writeregE,  rtE,    rsE,    insD[20:16],    insD[25:21],
                ForwardM,   flushE, ForwardAD,  ForwardBD,  stallD, stallF,ForwardAE,   ForwardBE);

endmodule
```

در انتها برای فایل یک تست بنچ مینویسیم که تنها ورودی آن clk است.

```
`timescale 1ns / 1ps

module TB;
    reg clk;

    MIPS uut (
        .clk(clk)
    );

    initial begin
        clk = 0;
        forever #5 clk=~clk;
    end


endmodule
```

برای بررسی ، باید یک سری دستورات را در ins_mem ذخیره کنیم.

```
1   `timescale 1ns / 1ps
2   module Ins_mem(input [31:0] a,  output [31:0] rd);
3       reg [31:0] RAM[127:0];
4       assign rd = RAM[a];
5
6       initial RAM[0]=32'b00100000000000010000000000000101;    //addi $v0, $zero, 0x0005      #$v0=0x5
7       initial RAM[4]=32'b00100000000000011000000000001100;    //addi $v1, $zero, 0x000C      #$v1=0xc
8       initial RAM[8]=32'b00100000011001111111111111110111;    //addi $a3, $v1, 0xFFF7        #$a3=0x10003
9       initial RAM[12]=32'b00000000111000100010000000100101;   //or   $a0, $a3, $v0               #$a0=0x10007
10      initial RAM[16]=32'b00000000011001000010100000100100;   //and $a1, $v1, $a0            #$a1=0x4
11      initial RAM[20]=32'b00000000101001000010100000100000;   //add $a1, $a1, $a0            #$a1=0x1000B
12      initial RAM[24]=32'b00010000101001110000000000001010;   //beq $a1, $a3, 0x000A         #$a1=0x1000B != $a3=0x10003 => not taken
13      initial RAM[28]=32'b00000000011001000010000000101010;   //slt $a0, $v1, $a0            #$a0=1
14      initial RAM[32]=32'b00010000100000000000000000000001;   //beq $a0, $zero, 0x0001       #not taken
15      initial RAM[36]=32'b00100000000001010000000000000000;   //addi $a1, $zero, 0x0000      #$a1=0x0
16      initial RAM[40]=32'b00000000111000100010000000101010;   //slt $a0, $a3, $v0            #$a0=0
17      initial RAM[44]=32'b00000000100001010011100000100000;   //add $a3, $a0, $a1            #$a3=0x0
18      initial RAM[48]=32'b00000000111000100011100000100010;   //sub $a3, $a3, $v0            #$a3=0xFFFFFFFB
19      initial RAM[52]=32'b10101100011001110000000001000100;   //sw $a3, 0x0044, $v1          #mem[80]=0xFFFFFFFB
20      initial RAM[56]=32'b10001100000000100000000001010000;   //lw $v0, 0x0050, $zero        #$v0=mem[80]=0xFFFFFFFB
21      initial RAM[60]=32'b00001000000000000000000000010001;   //j 0x0011                         #JUMP 68
22      initial RAM[64]=32'b00100000000000100000000000000001;   //addi $v0, $zero, 0x0001      #doesnt run
23      initial RAM[68]=32'b10001100000010010000000001010000;   //lw $t1, 0x0050, $zero        #$t1=mem[80]=0xFFFFFFFB
24      initial RAM[72]=32'b10101100000010010000000001011000;   //sw $t1, 0x0058, $zero        #mem[88]=$t1=0xFFFFFFFB
25      initial RAM[76]=32'b10101100000000010000000001010100;   //sw $v0, 0x0054, $zero        #mem[84]=$v0=0xFFFFFFFB
26
27  endmodule
```

برای آنکه دستور Sw بعد از lw را بررسی کنیم ابتدا لازم است یک رجیستر را مقدار دهی کنیم و محتویات آن را در جایی از حافظه ذخیره کنیم. سپس عدد ذخیره شده در حافظه را ابتدا در یک رجیستر load میکنیم و سپس همان محتویات را در جای دیگر حافظه ذخیره میکنیم.

محتویات رجیستر ها و حافظه اصلی پس از اجرای دستورات به این شکل است: