



中山大学 实验报告

SUN YAT-SEN UNIVERSITY

学号 21311274 姓名 林宇浩

【实验题目】流水线 CPU 设计(2)

【实验目的】用 FPGA 芯片实现可以处理数据冒险和控制冒险的流水线 CPU (MIPS)。

【实验工具】

BASYS3 实验板, Vivado 软件

【实验要求】

在“流水线 CPU 设计(1)”的基础上增加数据冒险与控制冒险的处理，使之能够正确运行程序“file2.asm”和“file3.asm”。

【实验建议步骤】

- 每一步都在原来的基础上拷贝一个新的 FPGA 工程，有时查不出错误时需要重做一遍。
- 新的 FPGA 工程先运行这个步骤的测试程序试一下并截屏，填写修改前的仿真图，然后再进行修改。
- 以下步骤从程序“file2a.asm”和“file3a.asm”开始修改，每一步都从前一步的测试程序进行修改。
- 本实验主要是分步骤完成数据冒险和控制冒险的处理，具体方法可以自己设计。

步骤 1、参考“fig-4.60.jpg”增加 ForwardUnit，处理数据冒险 (beq 和 bne 指令除外)，并正确运行删除了相关空语句之后的程序“file2b.asm”和“file3b.asm”(buzou 自己修改)。

* load-use 之间仍需要保留一条 nop 语句，load-beq/bne 仍需要保留两条 nop 语句。

* ForwardUnit 可以处理 ex 与 mem 和 wb 之间的数据冒险，beq 和 bne 的数据冒险属于 id 与 ex 和 mem 之间的数据冒险。

* Mux8 和 Mux9 加在 Mux7 之前,Mux10 和 Mux11 加在 Mux2 之前，Mux11 作为 ReadData2 接入 ExMemRegister (考虑 sw 和上一条指令的相关性)。

修改前的仿真图：

file2



file3



指出其中问题：

file2 中

第 3 条指令 `lw $s1,0($t0)` 原本在仿真的显示应该为 0806，即取出第 2 个数。但是因为没有旁路，取数地址值被错误计算为 0，在仿真中显示的是 0890，即错误取出了第一个数。

第 5 条指令 `addi $s1,$s1,1` 原本在仿真的显示应该为 1007。即使第三条指令执行错误，也应该为 1091。但是因为没有旁路，\$s1 中值为初始值 0，所以仿真的显示为 1001。

由于前面指令的取值和计算错误，导致 loop 循序执行次数错误。

file3 中

第 2 条指令 `addiu $s1,$s0,5` 原本在仿真的显示应该为 0402，由于没有旁路，\$s0 中值为初始值 0，所以仿真的显示为 0405。

第 9 条指令 `sll $s2,$s2,1` 原本在仿真的显示应该为 2002，由于没有旁路，\$s2 中值为初始值 0，所以仿真的显示为 2000。



中山大学 实验报告

第 10 条指令 addu \$s0,\$s0,\$s2 原本在仿真中的显示应该为 24ff，由于没有旁路，\$s2 中值为初始值 0，所以仿真中的显示为 20fd。

file3 中的 slt 和 stli 指令也存在数据冒险，但是恰巧比较结果是正确的。

修改成功后的仿真图：

file2



file3



步骤 2、参考“fig-4.60.jpg”增加 ForwardUnit2，处理 beq 和 bne 的数据冒险，并正确运行删除了相关空语句之后的程序“file2c.asm”和“file3c.asm”（自己修改）。

* 这样做主要是因为 beq 和 bne 的转移判断提前了，要单独处理。

* load-beq 和 load-bne 之间依需要保留两条 nop 语句。

修改前的仿真图：

file2



file3



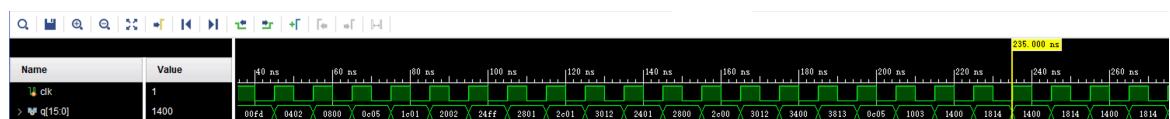
指出其中问题：

file2 没有相关空语句被删去，所以没有受到影响。

file3 从第 10 条指令 addu \$s0,\$s0,\$s2 开始进入循环，在仿真中对应 24ff。应循环两次后结束循环，但由于缺少对于 beq 和 bne 的旁路，仿真中只执行了一次循环。

修改成功后的仿真图：

file3





中山大学 实验报告

步骤3、参考“fig-4.60.jpg”增加HazardDetectUnit，处理load-use和load-beq/bne的use指令或beq和bne指令，并正确运行删除了相关空语句之后的程序“file2d.asm”和“file3d.asm”（自己修改）。

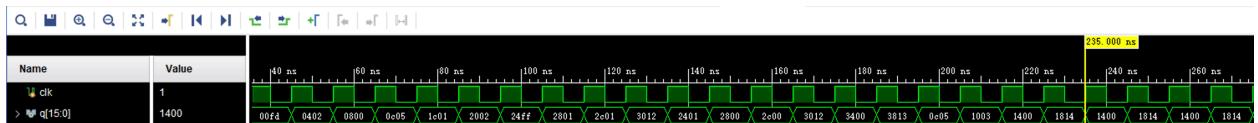
- * 如果是use指令，则删除之后的nop指令需要暂停(stall)一个时钟周期
- * 如果是beq/bne指令，则删除之后的nop指令需要暂停(stall)2个时钟周期。

修改前的仿真图：

file2



file3



指出其中问题：

file2中

第4条指令 addi \$s1,\$s1,1 原本在仿真中的显示应该为 0c07，由于没有进行阻塞，旁路的值不正确，错把取数地址4旁路到ALU进行加1，所以仿真中的显示为0c05。

同理，第6条指令 beq \$s1,\$s2,exit 原本在仿真中的显示应该为 14fe，由于没有进行阻塞，旁路的值不正确，错把取数地址8旁路到跳转比较单元进行比较，所以仿真中的显示为14fd。

file3中没有相关空语句被删去，所以没有受到影响。

修改成功后的仿真图：

file2



步骤5、删除转移指令之后的nop指令，并可以正确运行程序“file2.asm”和“file3.asm”。

参考“fig-4.60.jpg”，当转移指令(beq、j等)转移时，要清除其后的一条指令。

修改前的仿真图：

file2





中山大学 实验报告

file3



指出其中问题:

file2 中

第 10 条指令 sub \$s3,\$s3,\$s2 在仿真中的显示为 2400，由于第 9 条指令 j loop 后面没有进行阻塞，第 10 条指令被提前执行，导致 \$s3 中的内容被错误修改，造成后续循环中数值均错误。

最后一条指令 j 11 在仿真中的显示为 3000，由于最后一条指令后面没有进行阻塞，pc 在跳转前顺序增加，地址值超出指令取值范围，导致取到不确定态信号程序崩溃。

file3 中

第 4 条指令 addu \$s2,\$s1,\$s0 在仿真中的显示为 0c02，由于第 3 条指令 jal test 后面没有进行阻塞，第 10 条指令被提前执行，导致 \$s2 中的内容被错误修改。

第 10 条指令 bne \$s3,\$zero,loop 在仿真中的显示为 2401，由于其后面没有空指令阻塞，所以第 11 条 jr \$ra 被直接执行，程序被提前返回主函数。

修改成功后的仿真图:

file2



file3



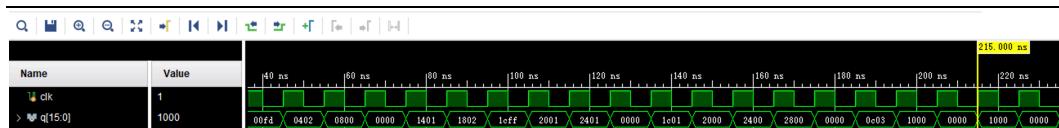
步骤 6 (选做)、分析一下，看是否合并 ForwardUnit 和 ForwardUnit2，如果认为可以，请完成实验，即可以正确运行程序“file2.asm”和“file3.asm”。

可以合并。

file2



file3



最后成功仿真步骤的模块源码（名字可变，也可增减，字体和行距最小，能看清就行，也可截屏）：

[CPU]

```
module CPU(clk, clk2, sm_vci, sm_duan, start, btlb);
    input clk; // #cpo
    input clk2; // #mcg
    input start; // #tmc
    input btlb; // #tblb
    output wire [3:0] sm_vci;
    output wire [8:0] sm_duan;
    wire [15:0] smg;

//    output wire [16:0] q;
//    assign q[15:8] = memb_pc[7:0];
//    assign q[7:0] = mxc[7:0];

//#IF
    wire [27:0] jumpadr;
    wire [31:0] shiftLeft;
    wire branch;
    wire [21:0] pc, pchd4, ins;
    wire [31:0] mxc5, mxc6;
//#IF ID
    wire [31:0] ifid_pc, ifid_iaddr4, ifid_ins;
//#IF
    wire right, jump, branch, memRead, memToReg, aluSrc, regWrite, jal, jr;
    wire [20:0] shdp;
    wire [31:0] readData1, readData2;
    wire [31:0] ra;
//#IF IR
    wire idex_reqbit, idex_jump, idex_branch, idex_memRead, idex_memToReg, idex_memWrite, idex_aluSrc, idex_regWrite;
    wire [31:0] idex_aluOp;
    wire [31:0] idex_pc, idex_readData2, idex_data, idex_jins;
//#IF
    wire [31:0] alude;
    wire [31:0] data, aludeResult;
    wire zero;
    wire [31:0] mxc;
    wire [4:0] mxc1;
//#IF MEM
    wire enmem_memWrite, enmem_memRead, enmem_regWrite, enmem_memToReg;
    wire [31:0] enmem_pc, enmem_aluResult, enmem_readData2;
    wire [4:0] enmem_mxc1;
//#MEM
    wire [31:0] readData;
//#MEM TB
    wire memb_pc, memb_memToReg;
    wire [31:0] memb_pc, memb_readData, memb_aluResult;
    wire [4:0] memb_mxc1;
//#TB
    wire [31:0] mxc5;

//#IFB
    moduleregister MEMFB(clk,
        enmem_pc, enmem_pc,
        enmem_pc, enmem_pc, enmem_pc,
        enmem_pc, enmem_pc, enmem_pc,
        enmem_pc, enmem_pc, enmem_pc,
        enmem_pc, enmem_pc, enmem_pc);
//#IFB
    moduleregister MEMFB(clk,
        memb_pc, memb_pc,
        memb_pc, memb_pc, memb_pc,
        memb_pc, memb_pc, memb_pc,
        memb_pc, memb_pc, memb_pc,
        memb_pc, memb_pc, memb_pc);
//#IFB
    moduleregister EMMFB(clk,
        enmem_pc, enmem_pc,
        enmem_pc, enmem_pc,
        enmem_pc, enmem_pc,
        enmem_pc, enmem_pc,
        enmem_pc, enmem_pc);

    display single(clk, smg, sm_vci, sm_duan);
endmodule
```

[HazardDetectUnit]

```
module HazardDetectUnit(clk, idex_pc, idex_branch, idex_ifid_pc, ifid_iaddr4, ifid_ins, smg, mxc5, branch2);
    input clk;
    input idex_pc;
    input idex_branch;
    input [4:0] idex_ifid_pc, ifid_iaddr4, ifid_ins;
    input [31:0] ifid_ins;
    output smg, mxc5;
    input branch2;

    reg [1:0] record;
    initial begin
        record=0;
        smg=0;
        mxc5=0;
        branch2=0;
    end

    always@(*) begin
        if (idex_pc & ( (idex_ifid_pc == ifid_iaddr4) | (idex_ifid_pc == ifid_iaddr4) ) ) begin
            if (ifid_ins[31:26]==#b'000000 | (ifid_ins[31:26]==#b'000000) ) begin
                smg=1;
                record=1;
            end
            else begin
                smg=0;
                record=0;
            end
        end
        else if ( (ifid_ins[31:26]==#b'000000 | ifid_ins[31:26]==#b'000000) | (ifid_ins[31:26]==#b'000000 | (ifid_ins[31:26]==#b'000000 & ifid_ins[3:0]==#b'0000) ) begin
            smg=1;
            record=0;
        end
        else smg=0;
    end

    always@(posedge clk) begin
        if(record) begin
            smg=1;
            record=0;
        end
        else smg=0;
    end
endmodule
```



[Mux4to1]

```
module Mux4to1(data00,data01,data10,data11,Forward,out);

    input [1:0] Forward;
    input [31:0] data00,data01,data10,data11;
    output [31:0] out;

    assign out = Forward==2'b01 ? data01 : (Forward == 2'b10 ? data10 : (Forward == 2'b11 ? data11 : data00));

endmodule
```

[PC]

```
module PC(clk,mux4,pc,pcAdd4,nop);
    input clk;
    input [31:0] mux4;
    output reg [31:0] pc;
    output [31:0] pcAdd4;
    input nop;

    initial begin
        pc=0;
    end

    assign pcAdd4 = pc+4;

    always@(posedge clk) begin
        if(nop==0)
            if(nop==0)
                pc=mux4;
        end
    end

endmodule
```

[IfIdRegister]

```
module IfIdRegister(clk,pc,pcAdd4,ins,ifid_pc,ifid_pcAdd4,ifid_ins,nop,nop2);
    input clk;
    input [31:0] pc,pcAdd4,ins;
    output reg [31:0] ifid_pc,ifid_pcAdd4,ifid_ins;

    input nop,nop2;

    initial begin
        {ifid_pc,ifid_pcAdd4,ifid_ins}<=0;
    end

    always @(posedge clk) begin
        if(nop==1)
            {ifid_pc,ifid_pcAdd4,ifid_ins}={ifid_pc,ifid_pcAdd4,ifid_ins};
        else if(nop2==1)
            {ifid_pc,ifid_pcAdd4,ifid_ins} = 0;
        else
            {ifid_pc,ifid_pcAdd4,ifid_ins} = {pc,pcAdd4,ins};
    end

endmodule
```

[Registers]

```
module Registers(clk,regWrite,readRegister1,readRegister2,writeRegister,writeData,readData,readData2,jal,pcAdd4,ra);
    input clk,regWrite,jal;
    input [4:0] readRegister1;
    input [4:0] readRegister2;
    input [4:0] writeRegister;
    input [31:0] writeData,pcAdd4;
    output [31:0] readData;
    output [31:0] readData2;
    output [31:0] ra;
    reg [31:0] regFiles[0:31];

    integer i;
    initial begin
        for(i=0;i<32;i=i+1)
            regFiles[i]<=0;
    end

    assign readData = regFiles[readRegister1];
    assign readData2 = regFiles[readRegister2];

    always @(posedge clk) begin
        if(jal==1)
            regFiles[31]=pcAdd4;//#if id#d
        if(regWrite)
            regFiles[writeRegister] = writeData;
    end

    assign ra=regFiles[31];
endmodule
```

[Control]



```
module Central(spCode, ins, regDst, jump, branch, memRead, memToReg, aluOp, memWrite,aluSrc,regWrite, jal,jr);
```

```
    input [5:0] spCode;
    input [31:0] ins;
    output reg regDst;
    output reg jump;
    output reg branch;
    output reg memRead;
    output reg memToReg;
    output reg [3:0] aluOp;
    output reg memWrite;
    output reg aluSrc;
    output reg regWrite;
    output reg jal;
    output reg jr;
```

```
    always@(*) begin
        if(ins==0) begin
            regDst = 0; aluSrc = 0; memToReg = 0;
            regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluOp = 4'b0000; jump = 0; jal=0; jr=0;
        end

        else if(spCode==4'b00000) begin
            if(ins[3:0]==4'b0000) begin//jr
                regDst = 0; aluSrc = 0; memToReg = 0;
                regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluOp = 4'b1000; jump = 0; jal=0; jr=0;
            end
            else begin//jal
                regDst = 1; aluSrc = 0; memToReg = 0;
                regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluOp = 4'b1111; jump = 0; jal=0; jr=0;
            end
        end
    end
```

```
    always@{posedge clk} begin
        if(spCode)
            if(ins[3:0]==4'b0000) begin
                regDst = 0; aluSrc = 0; memToReg = 0;
                regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluOp = 4'b0000; jump = 0; jal=0; jr=0;
            end
            else if(spCode==4'b00000) begin
                if(ins[3:0]==4'b0000) begin//jr
                    regDst = 0; aluSrc = 0; memToReg = 0;
                    regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluOp = 4'b1000; jump = 0; jal=0; jr=0;
                end
                else begin//jal
                    regDst = 1; aluSrc = 0; memToReg = 0;
                    regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluOp = 4'b1111; jump = 0; jal=0; jr=0;
                end
            end
        end
    end
```

```
    always@{posedge clk} begin
        if(spCode)
            if(ins[3:0]==4'b0000) begin
                regDst = 0; aluSrc = 0; memToReg = 0;
                regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluOp = 4'b0000; jump = 0; jal=0; jr=0;
            end
            else if(spCode==4'b00000) begin
                if(ins[3:0]==4'b0000) begin//jr
                    regDst = 0; aluSrc = 0; memToReg = 0;
                    regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluOp = 4'b1000; jump = 0; jal=0; jr=0;
                end
                else begin//jal
                    regDst = 1; aluSrc = 0; memToReg = 0;
                    regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluOp = 4'b1111; jump = 0; jal=0; jr=0;
                end
            end
        end
    end
```

[IdExRegister]

```
module IdExRegister(clk,sp,
jump,branch,memRead,memToReg,aluOp,memWrite,aluSrc,regWrite,
ifid_pc,readData1,readData2,idf_d,ins,
idex_regist,idex_jump,idex_branch,idex_memRead,idex_memToReg,idex_aluOp,idex_memWrite,idex_aluSrc,idex_regWrite,
idex_pc,idex_readData1,idex_data,idex_ins);

    input clk;
    input reg[jump,branch,memRead,memToReg,aluOp,memWrite,aluSrc,regWrite];
    input reg[idf_d];
    input reg[aluOp];
    input [31:0] ifid_pc;
    input [31:0] readData1,readData2;
    input [31:0] data;
    input [31:0] ifid_ins;

    input sp;

    output reg idex_regist,idex_jump,idex_branch,idex_memRead,idex_memToReg,idex_memWrite,idex_aluSrc,idex_regWrite;
    output reg [3:0] idex_aluOp;
    output reg [31:0] idex_pc;
    output reg [31:0] idex_readData1,idex_readData2;
    output reg [31:0] idex_data;
    output reg [31:0] idex_ins;

    initial begin
        idex_regist,idex_jump,idex_branch,idex_memRead,idex_memToReg,idex_memWrite,idex_aluSrc,idex_regWrite,
        idex_pc,idex_readData1,idex_readData2,idex_data,idex_ins)=0;
    end

    always @{posedge clk} begin
        if(sp==1)
            if(idex_pc==4'b0000)
                if(ex_regist,idex_jump,idex_branch,idex_memRead,idex_memToReg,idex_aluOp,idex_memWrite,idex_aluSrc,idex_regWrite,
                idex_pc,idex_readData1,idex_readData2,idex_data,idex_ins) ~= 0;
            else
                if(ex_regist,idex_jump,idex_branch,idex_memRead,idex_memToReg,idex_aluOp,idex_memWrite,idex_aluSrc,idex_regWrite,
                idex_pc,idex_readData1,idex_readData2,idex_data,idex_ins) ~= 0
                    reg[idf_d]=data;
                    ifid_pc.readData1=idex_pc.readData1;
                    ifid_pc.readData2=idex_pc.readData2;
                    ifid_pc.data=idex_pc.data;
                    ifid_pc.ifid_ins=idex_pc.ifid_ins;
    end
endmodule
```

[ALUcontrol]

```
module ALUcontrol(aluOp,funct,aluCtr);
    input [3:0] aluOp;
    input [5:0] funct;
    output reg [3:0] aluCtr;
    always @{aluOp or funct}
        casex(aluOp,funct)
            10'b0000xxxx: aluCtr = 4'b0010; // lw, sw, addi, addiu(addi)
            10'b0001xxxx: aluCtr = 4'b0110; // beq
            10'b1111100000: aluCtr = 4'b0010; // add
            10'b1111111101: aluCtr = 4'b0010; // addu(add)
            10'b1111110010: aluCtr = 4'b0110; // sub
            10'b11111x0100: aluCtr = 4'b0000; // and
            10'b11111100101: aluCtr = 4'b0001; // or
            10'b1111100000: aluCtr = 4'b0011; // sll
            10'b1111101010: aluCtr = 4'b0111; // slt
            10'b0010xxxxxx: aluCtr = 4'b0001; // ori
            10'b0011xxxxxx: aluCtr = 4'b0111; // slti
            10'b10110xxxxx: aluCtr = 4'b1110; // lne
            10'b10000xxxxx: aluCtr = 4'b1000; // j
            default: aluCtr = 4'b0010;
        endcase
endmodule
```

[ForwardUnit]



```
module ForwardDB(index_register, memb_register, idex_Rd, enem_Rd, memb_Rd, ForwardA, ForwardB, index_regWrite, ifid_Rt, ifid_Bd, idex_Bd, ForwardI, ForwardD);  
    input index_register, memb_register;  
    input [4:0] idex_Rs, idex_Bt;  
    input [4:0] enem_Rd, memb_Bd;  
    output reg [1:0] ForwardA, ForwardB;  
  
    initial begin  
        ForwardA=0;  
        ForwardB=0;  
    end  
  
    always@(*) begin  
        if ((index_register & (index_Rd == 0) & (index_Bd == ifid_Rt)) ForwardA = 2'b11;  
        else if ((enem_register & (enem_Rd == 0) & (enem_Bd == ifid_Rt)) ForwardA = 2'b10;  
        else if ((memb_register & (memb_Rd == 0) & (memb_Bd == ifid_Rt)) ForwardA = 2'b01;  
        else ForwardA = 2'b00;  
  
        if ((index_register & (index_Rd == 0) & (index_Bd == ifid_Bt)) ForwardB = 2'b11;  
        else if ((enem_register & (enem_Rd == 0) & (enem_Bd == ifid_Bt)) ForwardB = 2'b10;  
        else if ((memb_register & (memb_Rd == 0) & (memb_Bd == ifid_Bt)) ForwardB = 2'b01;  
        else ForwardB = 2'b00;  
    end  
endmodule
```

[ExMemRegister]

```
module ExMemRegister(clk,  
    index_memRead, index_regWrite, index_memToReg,  
    index_pc,aluResult, index_readData2, mux1,  
    enem_memWrite, enem_memRead, enem_regWrite, enem_memToReg,  
    enem_pc, enem_aluResult, enem_readData2, enem_mux1);  
  
    input clk;  
    input index_memWrite, index_memRead, index_regWrite, index_memToReg;  
    input [31:0] index_pc;  
    input [31:0] aluResult;  
    input [31:0] index_readData2;  
    input [4:0] mux1;  
  
    output reg enem_memWrite, enem_memRead, enem_regWrite, enem_memToReg;  
    output reg [31:0] enem_pc;  
    output reg [31:0] enem_aluResult;  
    output reg [31:0] enem_readData2;  
    output reg [4:0] enem_mux1;  
  
    initial begin  
        {enem_memWrite, enem_memRead, enem_regWrite, enem_memToReg,  
        enem_pc, enem_aluResult, enem_readData2, enem_mux1}<=0;  
    end  
  
    always @ (posedge clk) begin  
        {enem_memWrite, enem_memRead, enem_regWrite, enem_memToReg,  
        enem_pc, enem_aluResult, enem_readData2, enem_mux1} <=>  
        {index_memWrite, index_memRead, index_regWrite, index_memToReg,  
        index_pc, aluResult, index_readData2, mux1};  
    end  
endmodule
```

[DataMem]

```
module DataMem(clk,memWrite,memRead,address,writeData,readData,btnl,mg):  
    input clk,btnl;  
    input memWrite,memRead;  
    input [31:0] address;  
    input [31:0] writeData;  
    output [31:0] readData;  
    output [19:0] mg;  
    reg [31:0] mem[300:0];  
    initial begin  
        //      mem[0] = 32'h00000011;  
        //      mem[1] = 8;  
        //      mem[2] = 9;  
        mem[0] = 32'h00000090;  
        mem[1] = 6;  
        mem[2] = 9;  
        mem[3] = 32'h00000018;  
        mem[4] = 32'h00000098;  
        mem[5] = 32'h00000079;  
        mem[6] = 32'h00000011;  
        mem[7] = 32'h00000025;  
    end  
    always@(posedge clk) begin  
        if ((memWrite) & (mem[address]>>2)==writeData);  
    end  
    assign readData=mem[address]>>2;  
  
    reg [3:0] count;  
    initial begin  
        count = 0;  
    end  
    always@(posedge btnl) begin  
        count = count+1;  
        if(count>7)  
            count = 0;  
    end  
    assign mg=[count,mem[count][11:0]];  
endmodule
```



中山大学 实验报告

[MemWbRegister]

```
module MemWbRegister(clk,
    exmem_regWrite, exmem_memToReg,
    exmem_pc_readData, exmem_aluResult, exmem_mux1,
    memwb_regWrite, memwb_memToReg,
    memwb_pc, memwb_readData, memwb_aluResult, memwb_mux1);

    input clk;
    input exmem_regWrite, exmem_memToReg;
    input [31:0] exmem_pc;
    input [31:0] readData;
    input [31:0] exmem_aluResult;
    input [5:0] exmem_mux1;

    output reg memwb_regWrite, memwb_memToReg;
    output reg [31:0] memwb_pc;
    output reg [31:0] memwb_readData;
    output reg [31:0] memwb_aluResult;
    output reg [5:0] memwb_mux1;

    initial begin
        {memwb_pc, memwb_pc, memwb_pc, memwb_pc} = 0;
        {memwb_pc, memwb_pc, memwb_pc, memwb_pc} <= {exmem_pc, exmem_pc, exmem_pc, exmem_pc};
    end

    always @ (posedge clk) begin
        {memwb_pc, memwb_pc, memwb_pc, memwb_pc} <=
            {exmem_pc, exmem_pc, exmem_pc, exmem_pc};
    end

endmodule
```

[cpu_sim]

```
module cpu_sim_withoutmsg();
    reg clk;
    wire [15:0] q;
    reg clk2, start, btl;
    wire [3:0] sm_rst;
    wire [6:0] sm_dian;
    wire [6:0] sm_vci;

    CPU uCPU(clk, clk2, sm_rst, sm_dian, start, btl, q);

    initial begin
        clk = 0;
        clk2 = 0;
        start = 0;
        sm_rst = 1;
        btl = 0;
    end

    always #2 clk = ~clk;
endmodule
```

【完成情况】

是否完成? (√ 完成 × 未完成)

步骤 1 [√] 步骤 2 [√] 步骤 3 [√] 步骤 4 [√] 步骤 5 [√] 步骤 6 [√]

【实验体会】

写出实验过程中遇到的问题，解决方法和自己的思考；并简述实验体会（如果有的话）。

本次实验收获良多。在冒险检测单元遇到了信号传递不及时的问题，通过结合使用 always@(*) 和 always@(posedge clk) 保证了信号正常传输。在新的四路选择器中通过连续使用三目运算符在 assign 语句中实现了条件判断。在旁路单元通过调整 if 和 else if 中条件的顺序保证了旁路的数据来自最新的指令。在 CPU 模块中，之前由于模块排序没有顺序在 debug 的时候减低了效率，这次将模块和相关参数都按照顺序书写并标注了注释，通过双击代码中的变量软件会将同名变量进行高亮标注的功能（不分大小写），使得一些关于变量的错误能够及时发现。

【交实验报告】

每位同学单独完成本实验内容并填写实验报告。

交作业地点：<http://172.18.187.251/netdisk/default.aspx?vm=21org>

FPGA 实验/08、流水线 CPU 设计(2)

截止日期：2022 年 12 月 15 日 17:30 (周四)

上传文件：学号_姓名_流水线 CPU 设计 2. doc

学号_姓名_流水线 CPU 设计 2. rar （包含源代码的工程）