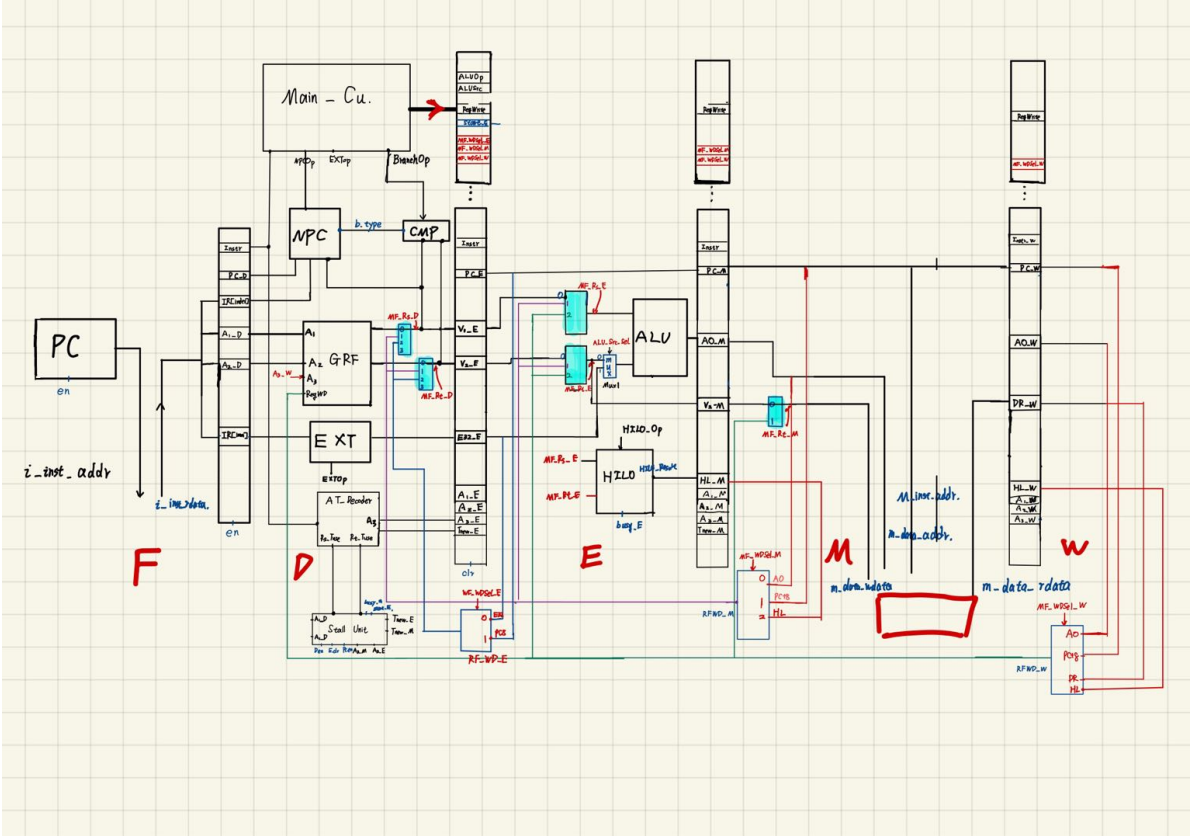
计算机组成P6实验报告

设计文档

一、整体结构

设计框架如图所示：



本流水线CPU支持的指令集为**{add, sub, and, or, slt, sltu, lui, addi, andi, ori, lb, lh, lw, sb, sh, sw, mult, multu, div, divu, mfhi, mflo, mthi,mtlo,beq, bne, jal, jr}**,相比于P5中的流水线CPU，主要新增了支持按字节、半字的存储指令、乘除法相关的指令已经部分cal_r,cal_类型的计算指令。

从整体架构而言，本流水线CPU设计为存储器外置，故取消了IFU模块与DM模块，对数据通路进行了一定的修改。同时，为支持新增的指令，本设计新增了HILO模块，BE模块以及DataExt模块，分别主要用于乘除法指令、store类指令以及load类指令。

二、数据通路模块设计

1.F_PC模块

输入输出端口设计：

信号名	方向	描述
clk	I	时钟信号

信号名	方向	描述
reset	I	异步复位信号，将PC寄存器清零。1：复位；0：无效。
en	I	使能信号，用于阻塞
NPC[31:0]	I	32位输入信号，下一条指令的地址
PC[31:0]	O	32位输出信号，当前指令的地址

2.D_NPC模块

信号名	方向	描述
PC_F [31:0]	I	32位输入信号，F阶段指令的地址
PC_D[31:0]	I	32位输入信号，D阶段指令的地址
NPCOp[2:0]	I	下一个PC的选择信号
BranchSignal	I	1位输入信号，表示CMP模块比较结果
offset [31:0]	I	32位输入信号，代表地址偏移量
Instr_index [25:0]	I	26位输入信号，代表j/jal指令的26位地址索引
RegData [31:0]	I	32位输入信号，代表jr指令目标寄存器的值
NPC [31:0]	O	32位输出信号，下一个PC的地址

3.D_CMP模块

输入输出端口设计：

信号名	方向	作用
[31:0] MF_Rs_D	input	32位输入信号，代表branch类型指令需要的Rs数据
[31:0] MF_Rt_D	input	32位输入信号，代表branch类型指令需要的Rt数据
[2:0] BranchOp	input	3位输入信号，选择Branch类型中不同的指令
BranchSignal	output	1位输入信号，代表跳转条件是否成立

4.D_EXT模块

信号名	方向	描述
Imm [15:0]	I	16位输入信号，表示待进行拓展的立即数
EXTOp [2:0]	I	3位输入信号，代表EXT选择的操作

信号名	方向	描述
EXTOut [31:0]	O	32位输出信号，表示拓展后的立即数

5.D_GRF模块

信号名	方向	描述
clk	I	时钟信号
reset	I	异步复位信号，将32个寄存器的值全部清零
RegWrite	I	写使能信号，高电平时向A3寄存器写入数据WD3；低电平时无效
A1 [4:0]	I	5位输入信号，选择32个寄存器中的一个并将其数据输出至RD1端口
A2 [4:0]	I	5位输入信号，选择32个寄存器中的一个并将其数据输出至RD2端口
A3 [4:0]	I	5位输入信号，选择32个寄存器中的一个作为数据写入的目标寄存器
GRF_WD [31:0]	I	32位输入数据，当WE3有效时向GRF中写入数据
RF_RD1 [31:0]	O	32位输出信号，输出A1中选择的寄存器中的数据
RF_RD2 [31:0]	O	32位输出信号，输出A2中选择的寄存器中的数据

6.E_ALU模块

信号名	方向	描述
Src_A [31:0]	I	32位输入信号，待ALU处理的数据之一
Src_B [31:0]	I	32位输入信号，待ALU处理的数据之一
ALUOp [3:0]	I	3位输入选择信号，选择ALU需要进行的操作
ALUOut[31:0]	O	32位输出信号，ALU运算的输出结果

7.E_HILO模块：

信号名称	方向	功能
clk	I	时钟信号
reset	I	复位信号，高电平时有效
D1 [31:0]	I	32位输入信号，代表源操作数1
D2 [31:0]	I	32位输入信号，代表源操作数2
HILO_Op [3:0]	I	4位输入信号，选择不同的乘除法指令对应的操作

信号名称	方向	功能
HILO_Result [31:0]	O	32位输出信号，代表HILO需要从HI寄存器或LO寄存器输出的数据
HILO_Busy	O	1位输出信号，表示HILO是否正在执行乘除运算操作

8.M_BE模块：

信号名称	方向	功能
A [1:0]	I	2位输入信号，代表地址的后两位
DM_Op [3:0]	I	4位输入信号，代表执行的存取操作
m_data_byteen [3:0]	O	4位输出信号，用于写入内存的字节使

9.W_DataEXT模块：

信号名称	方向	功能
DM_Op_W [3:0]	I	4位输入信号， 代表执行的存取操作
Din [31:0]	I	32位输入信号，代表从DM中读出的数据
A [1:0]	I	2位输入信号，代表取址的地址后两位
Dout [31:0]	O	32位输出信号，代表数据拓展后的结果

10.D_REG寄存器

```
module D_REG(

input clk,

input reset,

input en,

input [31:0] Instr_F,

input [31:0] PC_F,



output reg [31:0] Instr_D,

output reg [31:0] PC_D,

output reg [25:0] Instr_index_D,
```

```
output reg [15:0] Imm_D,

output reg [4:0] A1_D,

output reg [4:0] A2_D

);
```

11.E_REG寄存器

```
module E_REG(

input clk,
input reset,
input clr,

//come from MainCU
input [3:0] ALUOp,
input ALUSrc_Sel,
input RegWrite,
input [1:0] Elevel_Sel,
input [1:0] Mlevel_Sel,
input [1:0] Wlevel_Sel,

 //come from D_stage
input [31:0] Instr_D,
input [31:0] PC_D,
input [31:0] EXTOut,
input [4:0] A1_D,
input [4:0] A2_D,

 //come from Decoder
input [4:0] A3,
input [2:0] Tnew,

 //come from forward-Mux
input [31:0] MF_Rs_D,
input [31:0] MF_Rt_D,

//control signal
output reg [3:0] ALUOp_E,
output reg [3:0] HILO_Op_E,
output reg [3:0] DM_Op_E,
output reg start_E,
output reg ALUSrc_Sel_E,
output reg RegWrite_E,
output reg [1:0] Elevel_Sel_E,    //E级选择数据转发
output reg [1:0] Mlevel_Sel_E,    //M级选择数据转发
output reg [1:0] Wlevel_Sel_E,    //W级选择数据转发

output reg [31:0] Instr_E,
output reg [31:0] PC_E,
```

```
output reg [31:0] V1_E,
output reg [31:0] V2_E,
output reg [31:0] E32_E,

output reg [4:0] A1_E,
output reg [4:0] A2_E,
output reg [4:0] A3_E,
output reg [2:0] Tnew_E
);
```

12.M_REG寄存器

12.M_REG寄存器

```
module M_REG(
input clk,
input reset,


input RegWrite_E,
input [1:0] Mlevel_Sel_E,
input [1:0] Wlevel_Sel_E,

input [31:0] Instr_E,
input [31:0] PC_E,
input [31:0] ALUOut,
input [31:0] V2_E,

input [4:0] A1_E,
input [4:0] A2_E,
input [4:0] A3_E,
input [2:0] Tnew_E,
input [31:0] HILO_Result,
input [3:0] DM_Op_E,

output reg MemWrite_M,
output reg RegWrite_M,
output reg [1:0] Mlevel_Sel_M,  //M阶段转发数据选择信号
output reg [1:0] Wlevel_Sel_M,

output reg [31:0] Instr_M,
output reg [31:0] PC_M,
output reg [31:0] AO_M,     //写回寄存器的WriteData
output reg [31:0] V2_M,     //输入到DM中的WriteData

output reg [4:0] A1_M,
output reg [4:0] A2_M,
output reg [4:0] A3_M,
output reg [2:0] Tnew_M,
output reg [31:0] HL_M,
output reg [3:0] DM_Op_M
);
```

13.W_REG寄存器

```
module W_REG(
input clk,
input reset,

input RegWrite_M,
input [3:0] DM_Op_M,
input [1:0] Wlevel_Sel_M,

input [31:0] PC_M,
input [31:0] Instr_M,
input [31:0] AO_M,
input [31:0] DM_RD,

input [4:0] A1_M,
input [4:0] A2_M,
input [4:0] A3_M,
input [31:0] HL_M,

output reg RegWrite_W,
output reg [3:0] DM_Op_W,
output reg [31:0] Instr_W,
output reg [1:0] Wlevel_Sel_W,       //W阶段转发数据选择信号

output reg [31:0] PC_W,
output reg [31:0] AO_W,
output reg [31:0] DR_W,

output reg [4:0] A1_W,
output reg [4:0] A2_W,
output reg [4:0] A3_W,
output reg [31:0] HL_W
);
```

三、控制器设计与冒险解决

1.主控制器

主控制器的控制信号列表：

信号名	功能
start	1'b1: 本指令为乘除法计算指令 1'b0：本指令不是乘除法计算指令
RegWrite	1'b1: 本指令需要将数据写入寄存器 1'b0：本指令不需要写入寄存器

信号名	功能
ALUOp [3:0]	4'b0000: ADD 4‘b0001: SUB 4'b0010: OR 4'b0011: AND 4'b0100: SLT 4‘b0101: SLTU
EXTOp [1:0]	2’b00: Zero_Extend 2'b01: Sign_Extend 2'b10: Lui_Extend
NPCOp [2:0]	3'b000: PlusFour 3'b001: Branch 3'b010: Jump 3'b011:JumpReg
BranchOp [2:0]	3'b000:不是分支类指令 3'b001:BEQ 3'b010:BNE
HILO_Op [3:0]	4‘b0000:不是乘除类指令 4'b0001:MULT 4'b0010: MULTU 4'b0011:DIV 4'b0100:DIVU 4'b0101:MFHI 4'b0110:MFLO 4'b0111:MTHI 4'b1000:MTLO
DM_Op [3:0]	4‘b0000:不是存取类指令 4'b0001:SB 4'b0010: SH 4'b0011:SW 4'b0100:LB 4'b0101:LH 4'b0110:LW
ALUSrc_Sel	1‘b1: 本指令需要选择立即数 1‘b0：本指令不需要选择立即数
Elevel_Sel [1:0]	2’b00:E阶段转发Lui的结果 2'b01:E阶段转发PC+8
Mlevel_Sel [1:0]	2’b00:M阶段转发ALU的结果 2'b01:M阶段转发PC+8 2'b10:M阶段转发从乘除法模块读出的结果

信号名	功能
Wlevel_Sel [1:0]	2‘b00:W阶段转发ALU的结果 2'b01:W阶段转发PC+8 2 2'b10:W阶段转发DM的读取结果 2‘b11:W阶段转发从乘除法模块读出的结果

2.冲突处理单元

输入输出端口：

端口信号	方向	功能
[31:0] Instr_D	Input	32位输入信号，代表D级的指令
[2:0] Tnew_E	I	3位输入信号，代表E阶段指令的Tnew
[2:0] Tnew_M	I	3位输入信号，代表M阶段指令的Tnew
[4:0] A1_D	I	5位输入信号，代表D阶段指令的A1
[4:0] A2_D	I	5位输入信号，代表D阶段指令的A2
[4:0] A3_E	I	5位输入信号，代表E阶段指令的A3
[4:0] A3_M	I	5位输入信号，代表M阶段指令的A3
HILO_busy	I	1位输入信号，表示乘除模块是否在进行计算
Start_E	I	1位输入信号，表示E阶段指令是否为乘除运算
[2:0] Tnew	Output	3位输出信号，代表当前指令能产生新的写入值需要的周期数
[4:0] A3	Output	5位输出信号，代表当前指令的写入寄存器。若不需要写入寄存器，则A3的值为0
D_REG_en	O	1位输出信号，D级寄存器的使能信号
E_ERG_clr	O	1位输出信号，E级寄存器的清零信号
IFU_en	O	1位输出信号，IFU的使能信号

在P5阻塞分析的基础上，结合P6新增指令得出如下Tuse、Tnew表格：

	Tnew	功能部件
lui	0	EXT
jumplink	0	PC

	Tnew	功能部件
cal_r	1	ALU
cal_i	1	ALU
mf	1	HILO
load	2	DM

Tuse	Rs	Rt
branch	0	0
jumpreg	0	-
cal_r	1	1
cal_i	1	-
md	1	1
mt	1	-
store	1	2
load	1	-

阻塞机制：

```
wire StallRs_E=(Rs_Tuse<Tnew_E) && (A1_D == A3_E)&&(A1_D!=5'b0);

wire StallRt_E=(Rt_Tuse<Tnew_E) && (A2_D == A3_E)&&(A2_D!=5'b0);

wire StallRs_M=(Rs_Tuse<Tnew_M) && (A1_D == A3_M)&&(A1_D!=5'b0);

wire StallRt_M=(Rt_Tuse<Tnew_M) && (A2_D == A3_M)&&(A2_D!=5'b0);

wire Stall_HILO=(md_D||mf_D||mt_D) && (HILO_busy||start_E);

assign stall=StallRs_E||StallRt_E||StallRs_M||StallRt_M||Stall_HILO;
```

3.转发机制

```
    //----Forward----//
    //--E_Level--//
wire [31:0] RF_WD_E= (Elevel_Sel_E==2'b00)? E32_E:
                                    (Elevel_Sel_E==2'b01)? PC_E+8:32'hffffffff;
    //--M_Level--//
wire [31:0] RF_WD_M= (Mlevel_Sel_M==2'b00)?AO_M:
```

```
                                    (Mlevel_Sel_M==2'b01)?PC_M+8:
                                    (Mlevel_Sel_M==2'b10)?HL_M:32'hffffffff;

    //--M_Level--//
wire [31:0] RF_WD_W= (Wlevel_Sel_W==2'b00)?AO_W:
                                        (Wlevel_Sel_W==2'b01)?PC_W+8:
                                        (Wlevel_Sel_W==2'b10)?DR_Ext:
                                        (Wlevel_Sel_W==2'b11)?
HL_W:32'hffffffff;

//-----Receive----//
//--D_Level--//
wire [31:0] MF_Rs_D=(A1_D==5'd0)?32'd0:
                                      (A1_D==A3_E)?RF_WD_E:
                                      (A1_D==A3_M)?RF_WD_M:
                                    RF_RD1;

wire [31:0] MF_Rt_D=(A2_D==5'd0)?32'd0:
                                    (A2_D==A3_E)?RF_WD_E:
                                    (A2_D==A3_M)?RF_WD_M:
                                    RF_RD2;

//--E_Level--//
wire [31:0] MF_Rs_E=(A1_E==5'd0)?32'd0:
                                    (A1_E==A3_M)?RF_WD_M:
                                    (A1_E==A3_W)?RF_WD_W:
                                    V1_E;

wire [31:0] MF_Rt_E=(A2_E==5'd0)?32'd0:
                                    (A2_E==A3_M)?RF_WD_M:
                                    (A2_E==A3_W)?RF_WD_W:
                                    V2_E;

//--M_Level--//
wire [31:0] MF_Rt_M=(A2_M==5'd0)?32'd0:
                                    (A2_M==A3_W)?RF_WD_W:V2_M;
```

测试数据

一、针对乘除法模块的测试

```
ori $t0,$t0,100
ori $t1,$t1,12313
ori $t2,$t2,32422
lui $t3,0x1111
ori $t3,$t3,0x1111

mult $t1,$t2
mfhi $s0
mflo $s1

multu $t2,$t3
mfhi $s0
```

```
mflo $s1

div $t1,$t2
mfhi $s0
mflo $s1

divu $t2,$t3
mfhi $s0
mflo $s1

ori $s3,$s3,100
mthi $s3
mtlo $s3
mult $t1,$t2
mfhi $s0
mflo $s1
```

测试结果：通过

```
C:\Users\HaojunYan\Desktop\autotest>fc out.txt cpuout.txt
正在比较文件 out.txt 和 CPUOUT.TXT
***** out.txt

***** CPUOUT.TXT
*****
```

二、综合转发与阻塞测试

```
init:
    add $28,$0,$0
    add $29,$0,$0
    add $30,$0,$0
    add $31,$0,$0
    ori $28,$28,4

#D级Rs
block1:     #E jal
addi $t0,$0,10
jal block2
add $t0,$ra,$0

block2:     #E lui
addi $t0,$0,10
lui $t1,0x1111
add $t0,$t1,$0

block3:     #M_jal
addi $t0,$0,10
jal block4
nop
add $t0,$ra,$0
```

```
block4:     #M_lui
addi $t0,$0,10
lui $t1,0x1111
nop
add $t0,$t1,$0

block5:     #M_cal_r
addi $t0,$t0,10
addi $t1,$t1,20
nop
add $t2,$t1,$0

block6:     #M_cal_i
addi $t0,$t0,10
addi $t1,$t1,20
nop
ori $t2,$t1,10

block7:     #M_HILO
addi $t0,$t0,10
addi $t1,$t1,20
nop
mult $t0,$t1
mfhi $t2
mflo $t3

block8:     #W_jal
addi $t0,$t0,10
jal block9
nop
nop
add $t0,$ra,$0

block9:     #W_lui
addi $t0,$t0,10
lui $t1,0x1111
nop
nop
add $t0,$t1,$0

block10:    #W_cal_r
addi $t0,$t0,10
addi $t1,$t1,20
nop
nop
add $t2,$t1,$0

block11: #W_cali
addi $t0,$t0,10
addi $t1,$t1,20
nop
nop
ori $t2,$t1,10

block12: #W_HILO
```

```
addi $t0,$t0,10
addi $t1,$t1,20
nop
nop
mult $t0,$t1
mfhi $t2
mflo $t3

block13: #W_load
addi $t1,$t1,10
addi $t0,$0,0
lw $t1,($t0)
nop
nop
add $t2,$t1,$0

#D级Rt
block14:        #E jal
addi $t0,$0,10
jal block15
add $t0,$0,$ra

block15:        #E lui
addi $t0,$0,10
lui $t1,0x1111
add $t0,$0,$t1

block16:        #M_jal
addi $t0,$0,10
jal block17
nop
add $t0,$0,$ra

block17:        #M_lui
addi $t0,$0,10
lui $t1,0x1111
nop
add $t0,$0,$t1

block18:        #M_cal_r
addi $t0,$t0,10
addi $t1,$t1,20
nop
add $t2,$0,$t1

block19:        #M_HILO
addi $t0,$t0,10
addi $t1,$t1,20
nop
mult $t0,$t1
mfhi $t2
mflo $t3

block20:        #W_jal
addi $t0,$t0,10
```

```
jal block21
nop
nop
add $t0,$0,$ra

block21:        #W_lui
addi $t0,$t0,10
lui $t1,0x1111
nop
nop
add $t0,$0,$t1

block22:    #W_cal_r
addi $t0,$t0,10
addi $t1,$t1,20
nop
nop
add $t2,$0,$t1


block23: #W_HILO
addi $t0,$t0,10
addi $t1,$t1,20
nop
nop
mult $t0,$t1
mfhi $t2
mflo $t3

block24: #W_load
addi $t1,$0,10
addi $t0,$0,0
lw $t1,($t0)
nop
nop
add $t2,$0,$t1

#E_Rs
block25:    #M_jal
jal block26
add $t0,$ra,$0

block26:    #M_lui
addi $t0,$0,1
lui $t1,100
add $t2,$t1,$0
nop

block27:    #M_cal_r
addi $t0,$0,1
add $t1,$0,$t0
add $t2,$t1,$0
nop

block28:    #M_cal_i
```

```
ori $t0,$0,1
ori $t1,$0,100
add $t2,$t1,$0

block29:    #M_Hilo
addi $t0,$0,10
addi $t1,$0,20
mult $t0,$t1
mflo $t2
add $t3,$t2,$0

block30:
jal block31
nop
add $t0,$ra,$0

block31:
addi $t0,$0,1
lui $t1,100
nop
add $t2,$t1,$0

block32:
addi $t0,$0,1
add $t1,$0,$t0
nop
add $t2,$t1,$0

block33:
addi $t0,$0,1
ori $t1,$0,100
nop
add $t2,$t1,$0

block34:
ori $t1,$0,10
ori $t0,$0,0
lw $t1,0($t0)
nop
add $t2,$t1,$0

block35:
addi $t0,$0,10
addi $t1,$0,20
mult $t0,$t1
mflo $t2
nop
add $t3,$t2,$0

#E_Rt
block36:    #M_jal
jal block37
add $t0,$0,$ra

block37:    #M_lui
```

```
addi $t0,$0,1
lui $t1,100
add $t2,$0,$t1
nop

block38:    #M_cal_r
addi $t0,$0,1
add $t1,$0,$t0
add $t2,$0,$t1
nop

block39:    #M_cal_i
ori $t0,$0,1
ori $t1,$0,100
add $t2,$0,$t1

block40:    #M_Hilo
addi $t0,$0,10
addi $t1,$0,20
mult $t0,$t1
mflo $t2
add $t3,$0,$t2

block41:
jal block42
nop
add $t0,$0,$ra

block42:
addi $t0,$0,1
lui $t1,100
nop
add $t2,$0,$t1

block43:
addi $t0,$0,1
add $t1,$0,$t0
nop
add $t2,$0,$t1

block44:
addi $t0,$0,1
ori $t1,$0,100
nop
add $t2,$0,$t1

block45:
ori $t1,$t1,10
ori $t0,$0,0
lw $t1,0($t0)
nop
add $t2,$0,$t1

block46:
addi $t0,$0,10
```

```
addi $t1,$0,20
mult $t0,$t1
mflo $t2
nop
add $t3,$0,$t2

#M_Rt
block47:
jal block48
sw $ra,0($0)

block48:
addi $t0,$0,0
lui $t0,0x1111
sw $t0,0($0)

block49:
addi $t0,$t0,100
add $t1,$t0,$0
sw $t1,0($0)

block50:
addi $t0,$t0,100
add $t1,$t0,$0
sw $t1,0($0)

block51:
addi $t0,$t0,100
lw $t0,($0)
sw $t0,($0)

block52:    #W级HILO
addi $t0,$0,10
addi $t1,$0,10
mult $t0,$t1
mflo $t2
sw $t2,($0)

##STALL TEST

#branch
block53:
addi $t0,$t0,1
add $t1,$t0,$0
beq $t0,$t1,label1
nop
label1:
add $s0,$0,$0

block54:
addi $t0,$t0,1
addi $t1,$t0,10
beq $t0,$t1,label2
nop
label2:
```

```
add $s0,$0,$0

block55:
addi $t0,$t0,1
lw  $t1,($0)
beq $t0,$t1,label3
nop
label3:
add $s0,$0,$0

block56:
addi $s1,$0,100
addi $t0,$t0,10
addi $t1,$t1,10
mult $t0,$t1
mflo $t2
beq $t2,$s1,label4
nop
label4:
add $s0,$0,$0

block57:
addi $t0,$t0,1
lw  $t1,($0)
nop
beq $t0,$t1,label5
nop
label5:
add $s0,$0,$0

#cal_r
block58:
addi $t0,$0,10
nop
lw $t0,($0)
add $t1,$t0,$t0

#cal_i
block59:
addi $t0,$0,10
nop
lw $t0,($0)
ori $t0,$t0,100

#hilo
block60:
addi $t0,$t0,10
addi $t1,$t1,10
sw $t1,($0)
nop
lw $t2,($0)
mult $t1,$t2
mfhi $s0
mflo $s1
```

```
#load
block61:
sw $0,($0)
addi $t0,$0,4
lw $t0,($0)
lw $t1,($t0)

#store
block62:
addi $t0,$0,123
lw $t0,($0)
sw $t1,($t0)
```

思考题

1. **为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？**

   因为在执行add、sub、or等运算时，均只需要在E级进行一个时钟周期。但是设计乘除法的运算较为复杂，往往需要多个时钟周期方能完成。将乘除法部件单独开来而非整合进ALU，可以有效提高流水线CPU的吞吐量，防止乘除法阻塞add、sub等仅需要在E级执行一个时钟周期的指令。

   设计独立的HI、LO寄存器，可以用于暂存乘除法的计算结果，再利用mfhi、mflo指令写入32个通用寄存器之中，减少数据冲突发生的复杂性。

2. **真实的流水线 CPU 是如何使用实现乘除法的？请查阅相关资料进行简单说明。**

   真实的流水线CPU是通过乘法器与除法器以实现乘除法的。本质上讲，乘法就是加法，除法就是减法。具体而言：乘法是利用乘数寄存器与被乘数寄存器，然后采取循环加的方式判断是是否已经计算得到结果。除法是利用余数寄存器与商寄存器，采取循环移位减并记录中间量的方式，判断循环是否已经结束从而得到运算结果。

3. **请结合自己的实现分析，你是如何处理 Busy 信号带来的周期阻塞的？**

   ```
   wire Stall_HILO=(md_D||mf_D||mt_D) && (HILO_busy||start_E);

   assign stall=StallRs_E||StallRt_E||StallRs_M||StallRt_M||Stall_HILO;
   ```

   首先分析Busy信号带来的阻塞情况，我们约定乘除法模块遵守“当 Busy 信号或 Start 信号为 1 时，`mult, multu, div, divu, mfhi, mflo, mthi, mtlo`等乘除法相关的指令均被阻塞在 D 流水级。”，故需要将乘除法模块的busy信号与E寄存器输出的start信号均传输到冲突处理模块之中。同时，还需要当D级指令为乘除法相关的指令时才需要阻塞（否则不需要用到乘除法模块，也就没有冲突了）。故最终有 Stall_HILO=(md_D||mf_D||mt_D) && (HILO_busy||start_E)，并将该stall信号与其他阶段的阻塞信号相或即可。

4. **请问采用字节使能信号的方式处理写指令有什么好处？（提示：从清晰性、统一性等角度考虑）**

   从清晰度角度，采用四位字节使能信号可以直观对应一个字中的四个角度，针对不同写指令需要执行的操作，我们均可以直观地映射到字节使能信号之中。

   从统一性角度，字节使能信号将不同写指令的操作进行了整合。在新增指令的时候，我们无需再对新的写内存指令进行过多额外的分析，仅需对应好字节使能信号即可。

5. **请思考，我们在按字节读和按字节写时，实际从 DM 获得的数据和向 DM 写入的数据是否是一字节？在什么情况下我们按字节读和按字节写的效率会高于按字读和按字写呢？**

   实际上我们向DM中写入的数据以及从DM中获得的数据仍是一个字，而非一个字节。在写入DM的过程中，我们根据字节使能信号将写入DM的字进行替换，实际上我们写入的仍是字，只是表现的效果为仅修改了某个字节/半字。在读出DM时，我们也是将DM一个字的数据进行读出，然后再进行数据拓展已获得我们所需要的数据。

   在大量涉及对字节操作的程序中，lb、sb等指令占比较大，此时按字节读写的效率会高于按字读写。

6. **为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？**

   我采取了：利用宏定义代替操作、指令分类、加强模块化等手段对抗复杂性。

   特点与帮助：

   1）利用宏定义：代替数值可以在处理译码时进行一定的简化，防止出现bug发生的情况。同时，采用宏定义也能使得代码更加直观，便于理解。（例如以`ALU_add代替4'b0000）

   2）指令分类：在本流水线CPU中，我将指令分为以下几类：cal_r, cal_i, store, load, jumplink, jumpreg, branch, lui, md, mf, mt。对指令进行分类可以极大程度简化在新增指令时对转发与阻塞的处理，因为对于同一类指令，Tnew、Tuse、A3等均是大致相同的，在新增指令时我们只需要考虑分类即可，而无需单独考虑该指令可能涉及的冒险。

   3）加强模块化：为秉持高内聚低耦合的设计理念，本人权衡了模块化以及复杂度之间的处理。比如转发机制，其实无需单独设立一个模块对其进行判断，我们可以直接利用数据通路的数据结合MUX即可直接判断是否需要转发，这样即可减少了datapath与controller之间的联系，使得耦合度较低。

7. **在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？**

   指令类型组合产生的冲突：

   会产生写入寄存器结果的指令类型有

   1)cal_r 2)cal_i 3)jumplink 4)load 5)mf

   Rs与Rt需要从寄存器堆中读出结果的指令类型有

   1)cal_r 2)cal_i 3)jumpreg 4)store 5)load 6)mt 7)md 8)branch

   会产生写入的寄存器类型指令与需要从寄存器堆中读出数据的指令均有可能产生冲突，本人采用的解决方法为基于AT法的暴力转发+Tuse/Tnew阻塞。通过阻塞单元以及数据通路中的MUX进行实现。

相应的测试样例在测试数据中展示。

8. **如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。**

   本人测试采取了手动构造样例的方式，具体构造策略如下：

   在转发测试中，针对不同转发接收处，分别构造该处与不同类型指令之间的数据冒险，来检验此处转发数据的正确性。

   在阻塞测试中，根据Tnew与Tuse策略表，构造数据冒险检测阻塞机制是否能够工作。

   除此之外，针对新增的乘除法模块，本人额外构造了一份测试数据对其进行单独验证，判断是否能够正确工作。