

# 2020年北航计组P6实验报告

冯张驰 19373573

## 一、CPU设计方案综述

### (一) 总体设计概述

本CPU为verilog实现的流水线MIPS - CPU，支持的MIPS-C3指令集为={LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、SUBU、MULT、MULTU、DIV、DIVU、SLL、SRL、SRA、SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO}

处理器为五级流水线设计。为了实现这些功能，CPU主要包含了IFU、GRF、ALU、DM、EXT、Controller、以及每级模块之间的流水寄存器模块、确定转发和暂停机制的处理冒险的模块单元等。

### (二) 关键模块定义

#### 1.IFU

本部分分为3个模块，PC,pcCalc,IM

##### PC模块

##### 端口定义

端口名	方向	描述
clk	I	时钟信号
reset	I	复位信号
next_pc [31:0]	I	下一时刻的PC数值
pc [31:0]	O	PC的当前值
en	I	使能信号

##### pcCalc模块

##### 端口定义

端口名	方向	描述
zero	I	cmp的比较结果
branch[3:0]	I	当前指令是否为b型指令，以及是哪一类b型指令
imm[31:0]	I	32位地址（beq指令的立即数符号扩展后的数据）
npc	O	next-pc值
jump	I	当前指令是否为j/jal指令
rsData [31:0]	I	GRF模块RD1输出端口的值
imm26 [25:0]	I	当前周期内指令的低26位立即数
pc_D4[31:0]	I	D级的PC+4
pc[31:0]	I	F级的PC值
pcSrc	I	PC的选择信号

## IM模块

### 端口定义

端口名	方向	描述
pc [31:0]	I	PC当前值
instr [31:0]	O	当前PC值对应的32位指令

### IFU功能

1. 同步复位 复位信号有效时，PC被设置为**0x00003000**。
2. 取指令 根据**PC当前值**从**IM模块**中取出指令，并输出。
3. 计算下一条指令地址。
  - 当前指令是jal(pcSrc==1)， $PC = PC[31:28] \parallel imm26 \parallel 0^2$
  - 如果当前指令是beq指令（branch==1）且ALU输出的zero信号为1，则 $PC = PC + 4 + sign\_extend(offset \parallel 0^2)$
  - 当前指令是jr指令， $PC = GPR[RS]$
  - 以上三种情况之外， $PC = PC + 4$ ；

## 2.F2D（流水寄存器1）

### 端口定义

端口名	方向	描述
PC_D[31:0]	O	D级PC值
PC_D4[31:0]	O	D级PC值+4
PC_D8[31:0]	O	D级PC值+8
instr_D[31:0]	O	D级指令
PC_F[31:0]	I	F级PC值
instr_F[31:0]	I	F级指令
clk	I	时钟信号
reset	I	异步复位信号
en	I	使能信号

功能定义

F/D级流水寄存器

3.GRF

端口定义

端口名	方向	描述
RA1[4:0]	I	RD1中数据的寄存器地址
RA2[4:0]	I	RD2中数据的寄存器地址
RD1[31:0]	O	RA1地址对应寄存器数值
RD2[31:0]	O	RA2地址对应寄存器数值
WA	I	写入数据的寄存器地址
WD	I	写入WA对应寄存器的数据
WE	I	写使能信号
clk	I	时钟信号
reset	I	异步复位信号

功能定义

- 1.复位。复位信号有效时所有寄存器的数值被设置为0x00000000。
- 2.读寄存器 根据当前输入的地址信号读出32位数据。
- 3.写寄存器。根据当前输入的地址信号，把输入的数据写入相应寄存器。

## 4.CMP

### CMP端口定义

端口名	方向	描述
d1[31:0]	I	32位数据d1
d2[31:0]	I	32位数据d2
branch[3:0]	I	D级控制单元输出的b类型指令类别
equal	O	是否满足对应指令下的跳转条件，满足则为1

### 功能

比较两个端口的数值的关系，确定是否满足跳转条件

## 5.EXT

### EXT端口定义

端口名	方向	描述
inm16[15:0]	I	16位立即数
extOp	I	扩展控制信号 0：符号扩展 1：无符号扩展
ext_D[31:0]	O	扩展后输出的数据值

### 功能

根据控制信号extOp进行相应的符号扩展操作

## 6.D2E（流水寄存器2）

端口名	方向	描述
instr_D[31:0]	I	D级指令
pc_D[31:0]	I	D级PC值
pc_D4[31:0]	I	D级PC值+4
pc_D8[31:0]	I	D级PC值+8
pc_E[31:0]	O	E级PC值
pc_E4[31:0]	O	E级PC值+4
pc_E8[31:0]	O	E级PC值+8
grf_RD1[31:0]	I	GRF端口RD1读的数值
grf_RD2[31:0]	I	GRF端口RD2读的数值
ext_D[31:0]	I	D级的32位扩展后立即数
instr_E[31:0]	O	E级指令
rs_E[31:0]	O	输出的rs的数值
rt_E[31:0]	O	输出的rt的数值
ext_E[31:0]	O	传给E级的32位扩展后立即数
clk	I	时钟信号
reset	I	异步复位信号

## 7.ALU

### 端口定义

端口名	方向	描述
A[31:0]	I	数据1
B[31:0]	I	数据2
aluCtrl[3:0]	I	控制信号0000：与 0001：或 0010：加 0011：减 0100：nor 0101：xor 0110：逻辑左移 0111：逻辑右移 1000：算数右移 1001：符号数相比小于置1 1010：无符号数相比，小于置1 1011：lui
result	O	输出的数据
zero	O	判断运算结果是否为0

### 功能定义

- 1.与：A&B
- 2.或：A|B
- 3.加：A+B

4.减：A-B

5.其他，如上表所示

8.E2M（流水寄存器3）

端口名	方向	描述
instr_E[31:0]	I	E级指令
pc_E[31:0]	I	E级PC值
pc_E4[31:0]	I	E级PC值+4
pc_E8[31:0]	I	E级PC值+8
rt_E[31:0]	I	E级保存的rt寄存器数值
aluRet_E[31:0]	I	ALU运算的结果
pc_M[31:0]	O	M级PC值
pc_M4[31:0]	O	M级PC值+4
pc_M8[31:0]	O	M级PC值+8
aluRet_M[31:0]	O	M级的ALU运算结果
instr_M[31:0]	O	M级指令
rt_M[31:0]	O	M级rt寄存器对应数值
clk	I	时钟信号
reset	I	同步复位信号

9.DM

端口定义

端口名	方向	描述
A[11:2]	I	数据的地址信号
WD[31:0]	I	当WE为高电平时，写入地址A的数据
RD[31:0]	O	当RE为高电平时，读出地址A的数据
RE	I	高电平时允许读数据
WE	I	高电平时允许写入数据
clk	I	时钟信号
reset	I	异步复位信号

功能定义

- 1. 复位：复位信号有效时，所有数据被设置为0x00000000。
- 2. 读：根据当前输入的寄存器地址读出数据
- 3. 写数据：根据输入地址，写入输入的数据。

store\_judge模块

端口名	方向	描述
op[1:0]	I	写内存控制信号
A_1_0	I	地址最低两位
BE[3:0]	O	写使能信号

data\_ext模块

端口名	方向	描述
A[1:0]	I	地址最低两位
op[2:0]	I	读内存的控制信号
Din[31:0]	I	内存中load的数据
Dout[31:0]	O	扩展的读取内存的输出数据

10.M2W（流水寄存器3）

端口名	方向	描述
instr_M[31:0]	I	M级指令
pc_M[31:0]	I	M级PC值
pc_M4[31:0]	I	M级PC值+4
pc_M8[31:0]	I	M级PC值+8
rt_M[31:0]	I	M级保存的rt寄存器数值
aluRet_M[31:0]	I	M级ALU运算的结果
pc_W[31:0]	O	W级PC值
pc_W4[31:0]	O	W级PC值+4
pc_W8[31:0]	O	W级PC值+8
aluRet_W[31:0]	O	W级的ALU运算结果
instr_W[31:0]	O	W级指令
rt_W[31:0]	O	W级rt寄存器对应数值
clk	I	时钟信号
reset	I	同步复位信号
RD_M[31:0]	I	M级读出的数据
RD_W[31:0]	O	W保存的M级读出的数据

## 11.Controller

### 端口定义



端口名	方向	描述
op[5:0]	I	6位opcode字段
funcode[5:0]	I	6位funcode字段
regDst[1:0]	O	GRF写地址控制信号 <b>2'b00</b> : 选择rt端 <b>2'b01</b> : 选择rd端 <b>2'b10</b> : 选择31号寄存器
aluSrc	O	决定输入ALU模块的第二个输入端口数据是寄存器的输出值还是指令最低16位立即数扩展后的数值, <b>0, 是前者, 1, 是后者</b>
regWrite	O	GRF写控制
memRead	O	DM读控制
memWrite	O	DM写控制
memToReg[1:0]	O	GRF写数据选择: <b>2'b00</b> : 写入ALU的计算结果 <b>2'b01</b> : 写入DM的读出数据 <b>2'b10</b> : 写入lui计算所得的数据结果 <b>2'b11</b> : 写入当前PC加4后的值
extOp[1:0]	O	扩展控制信号 0: 符号扩展 1: 无符号扩展 2: 将16位立即数加载到高16位, 低16位补0
branch	O	是否为beq指令
aluCtrl	O	ALU的控制信号
jump	O	传递给pcCalc模块, 决定next_pc的值
pcSrc	O	传递给pcCalc模块, 选择是否为j类型的指令
shamt_or_rs	O	决定用于移位的运算符是shamt还是rs
mord	O	判断乘法还是除法
whiLo	O	判断是写高位寄存器还是低位
weMD	O	判断是否需要写乘除寄存器
mdStart	O	判断当前回合乘除单元是否需要开始计算
extRdOp[2:0]	O	判断读出的数据扩展信号
storeOp[1:0]	O	写入的数据的控制
signmd	O	乘除是否是无符号计算
rHiLo	O	需要读还是写高、低位寄存器

### (三) 控制器设计思路

设计方法

根据每个指令的opcode,funcode信号和对应的控制信号间的关系列出真值表，在 `always@(*)` 块中使用 `case` 语句块对译码和输出控制信号过程进行描述。详细代码在上一部分已展出。

主单元译码电路真值表

cal_r型指令	add	addu	sub	subu	AND	OR	NOR	XOR	sll	sliv	slt	sltu	sra	srav	srl	srlv
op[5:0]	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
func[5:0]	100000	100001	100010	100011	100100	100101	100111	100110	000000	000100	101010	101011	000011	000111	000010	000110
rt[5:0]	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx
regDst[1:0]	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
aluSrc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
regWrite	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
memRead	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
memWrite	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
memToReg[2:0]	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000
extOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
branch	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aluCtrl[3:0]	0010	0010	0011	0011	0000	0001	0100	0101	0110	0110	1001	1010	1000	1000	0111	0111
jump	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pcSrc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
extRdOp[2:0]	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000
storeOp[1:0]	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
mdStart	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mord	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shamt_or_rs	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0
whiLo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
weMD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

cal_r型指令	lui	ori	addi	addiu	andi	xori	slti	sltiu
op[5:0]	001111	001101	001000	001001	001100	001110	001010	001011
func[5:0]	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx
rt[5:0]	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx
regDst[1:0]	00	00	00	00	00	00	00	00
aluSrc	1	1	1	1	1	1	1	1
regWrite	1	1	1	1	1	1	1	1
memRead	0	0	0	0	0	0	0	0
memWrite	0	0	0	0	0	0	0	0
memToReg[2:0]	000	000	000	000	000	000	000	000
extOp	0	1	0	0	1	1	0	0
branch	0	0	0	0	0	0	0	0
aluCtrl[3:0]	1011	0001	0010	0010	0000	0101	1001	1001
jump	0	0	0	0	0	0	0	0
pcSrc	0	0	0	0	0	0	0	0
extRdOp[2:0]	000	000	000	000	000	000	000	000
storeOp[1:0]	00	00	00	00	00	00	00	00
mdStart	0	0	0	0	0	0	0	0
mord	0	0	0	0	0	0	0	0
shamt_or_rs	0	0	0	0	0	0	0	0
whiLo	0	0	0	0	0	0	0	0
weMD	0	0	0	0	0	0	0	0

md类型指令	mfhi	mflo	mthi	mtlo	mult	multu	div	divu
op[5:0]	000000	000000	000000	000000	000000	000000	000000	000000
func[5:0]	010000	010010	010001	010011	011000	011001	011010	011011
rt[5:0]	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx	xxxxx
regDst[1:0]	01	01	00	00	00	00	00	00
aluSrc	x	x	x	x	0	0	0	0
regWrite	1	1	0	0	0	0	0	0
memRead	0	0	0	0	0	0	0	0
memWrite	0	0	0	0	0	0	0	0
memToReg[2:0]	011	100	000	000	000	000	000	000
extOp	0	0	0	0	0	0	0	0
branch	0	0	0	0	0	0	0	0
aluCtrl[3:0]	0000	0000	0000	0000	0000	0000	0000	0000
jump	0	0	0	0	0	0	0	0
pcSrc	0	0	0	0	0	0	0	0
extRdOp[2:0]	000	000	000	000	000	000	000	000
storeOp[1:0]	00	00	00	00	00	00	00	00
mdStart	0	0	0	0	1	1	1	1
mord	0	0	0	0	0	0	1	1
shamt_or_rs	0	0	0	0	0	0	0	0
wHiLo	0	0	0	1	0	0	0	0
weMD	0	0	1	1	0	0	0	0

store类型	sw	sb	sh
op[5:0]	101011	101000	101001
func[5:0]	xxxxxxx	xxxxxxx	xxxxxxx
rt[5:0]	xxxxxx	xxxxxx	xxxxxx
regDst[1:0]	xx	xx	xx
aluSrc	1	1	1
regWrite	0	0	0
memRead	0	0	0
memWrite	1	1	1
memToReg[2:0]	xxx	xxx	xxx
extOp	0	0	0
branch	0	0	0
aluCtrl[3:0]	0010	0010	0010
jump	0	0	0
pcSrc	0	0	0
extRdOp[2:0]	xxx	xxx	xxx
storeOp[1:0]	00	10	01
mdStart	0	0	0
mord	0	0	0
shamt_or_rs	0	0	0
wHiLo	0	0	0
weMD	0	0	1

load类型	lw	lb	lbu	lh	lhu
op[5:0]	100011	100000	100100	100001	100101
func[5:0]	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx
rt[5:0]	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx
regDst[1:0]	00	00	00	00	00
aluSrc	1	1	1	1	1
regWrite	1	1	1	1	1
memRead	1	1	1	1	1
memWrite	0	0	0	0	0
memToReg[2:0]	001	001	001	001	001
extOp	0	0	0	0	0
branch	0	0	0	0	0
aluCtrl[3:0]	0010	0010	0010	0010	0010
jump	0	0	0	0	0
pcSrc	0	0	0	0	0
extRdOp[2:0]	000	010	001	100	011
storeOp[1:0]	xx	xx	xx	xx	xx
mdStart	0	0	0	0	0
mord	0	0	0	0	0
shamt_or_rs	0	0	0	0	0
wHiLo	0	0	0	1	0
weMD	0	0	1	1	0

<b>b类型</b>	<b>beq</b>	<b>bne</b>	<b>bgtz</b>	<b>blez</b>	<b>bgez</b>	<b>bltz</b>
op[5:0]	000100	000101	000111	000110	000001	000001
func[5:0]	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx
rt[4:0]	xxxxx	xxxxx	xxxxx	xxxxx	00001	00000
regDst[1:0]	xx	xx	xx	xx	xx	xx
aluSrc	x	x	x	x	x	x
regWrite	0	0	0	0	0	0
memRead	0	0	0	0	0	0
memWrite	0	0	0	0	0	0
memToReg[2:0]	xxx	xxx	xxx	xxx	xxx	xxx
extOp	x	x	x	x	x	x
branch[3:0]	0001	0010	0011	0100	0101	0110
aluCtrl[3:0]	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
jump	0	0	0	0	0	0
pcSrc	0	0	0	0	0	0
extRdOp[2:0]	xxx	xxx	xxx	xxx	xxx	xxx
storeOp[1:0]	xx	xx	xx	xx	xx	xx
mdStart	0	0	0	0	0	0
mord	0	0	0	0	0	0
shamt_or_rs	0	0	0	0	0	0
wHiLo	0	0	0	0	0	0
weMD	0	0	0	0	0	0

j类型	jal	j	jr	jalr
op[5:0]	000011	000010	000000	000000
func[5:0]	xxxxxx	xxxxxx	001000	001001
rt[4:0]	xxxxx	xxxxx	xxxxx	xxxxx
regDst[1:0]	10	xx	xx	01
aluSrc	x	x	x	x
regWrite	1	0	0	1
memRead	0	0	0	0
memWrite	0	0	0	0
memToReg[2:0]	010	xxx	xxx	010
extOp	x	x	x	x
branch[3:0]	0000	0000	0000	0000
aluCtrl[3:0]	xxxx	xxxx	xxxx	xxxx
jump	1	1	0	0
pcSrc	1	1	1	1
extRdOp[2:0]	xxx	xxx	xxx	xxx
storeOp[1:0]	xx	xx	xx	xx
mdStart	0	0	0	0
mord	0	0	0	0
shamt_or_rs	0	0	0	0
wHiLo	0	0	0	0
weMD	0	0	0	0

## (四) 冒险单元设计

### 端口定义

端口名	方向	描述
instr_D[31:0]	I	D级的31位指令
instr_E[31:0]	I	E级的31位指令
instr_M[31:0]	I	M级的31位指令
instr_W[31:0]	I	W级的31位指令
stall_F	O	F级暂停信号
stall_D	O	D级暂停信号
flush_E	O	E级暂停信号
selRsD[2:0]	O	forRsD选择信号
selRtD[2:0]	O	forRtD选择信号
selRsE[2:0]	O	forRsE选择信号
selRtE[2:0]	O	forRtE选择信号
selRtM[2:0]	O	forRtM选择信号
busy	I	乘除单元是否正在工作

冒险信号控制表格

IF/ID当前指令			ID/EX当前指令			EX/MEM
指令类型	源寄存器	Tuse	Tnew			
			cal_r/rd	cal_i/rt	load/rt	load/rt
			1	1	2	1
br	rs/rt	0	stall	stall	stall	stall
cal_r/cal_md	rs/rt	1			stall	
cal_i	rs	1			stall	
load	rs	1			stall	
store	rs	1			stall	
store	rt	2				
jr	rs	0	stall	stall	stall	stall
jalr	rs	0	stall	stall	stall	stall
w_md	rs	1			stall	
md&busy						
md&cal_md_E						

暂停控制信号表

需要转发的端口	寄存器	MUX	选择信号	默认输出	E			M				W						
					jal_31	jalr_rd	mflo/mfhi_rd	cal_r_rd	cal_i_rt	jal_31	jalr_rd	mflo_mfhi	cal_r_rd	cal_i_rt	load_rt	jal_31	jalr_rd	mflo_mfhi
D	rs	forRsD	selRsD	RD1	pc_ES	pc_ES	md_out	aluRet_M	aluRet_M	pc_MS	pc_MS	md_out_M	WD	WD	WD	pc_WS	pc_WS	WD
	rt	forRtD	selRtD	RD2	pc_ES	pc_ES	md_out	aluRet_M	aluRet_M	pc_MS	pc_MS	md_out_M	WD	WD	WD	pc_WS	pc_WS	WD
E	rs	forRsE	selRsE	grf_RD1				aluRet_M	aluRet_M	pc_MS	pc_MS	md_out_M	WD	WD	WD	pc_WS	pc_WS	WD
	rt	forRtE	selRtE	grf_RD2				aluRet_M	aluRet_M	pc_MS	pc_MS	md_out_M	WD	WD	WD	pc_WS	pc_WS	WD
M	rt	forRtM	selRtM	rt_M								WD	WD	WD	pc_WS	pc_WS	WD	

转发控制信号表



## 冒险单元代码

```
`timescale 1ns / 1ps
`include "constants.v"
module hazard_Unit(
    input [31:0] instr_D,
    input [31:0] instr_E,
    input [31:0] instr_M,
    input [31:0] instr_W,
    output stall_F,
    output stall_D,
    output flush_E,
    output [2:0] selRsD,
    output [2:0] selRtD,
    output [2:0] selRsE,
    output [2:0] selRtE,
    output [2:0] selRtM,
    input busy
);

    wire
D_b,D_cal_r,D_cal_i,D_load,D_store,D_jr,D_jal,D_jalr,D_cal_md,D_md,D_r_md,D_w_md
;
    instr_class D_class (.instr(instr_D), .jal(D_jal), .cal_i(D_cal_i),
.cal_r(D_cal_r), .load(D_load), .b(D_b), .jr(D_jr),
.store(D_store),.jalr(D_jalr),.md(D_md),.cal_md(D_cal_md),.r_md(D_r_md),.w_md(D_w_md));
    wire
E_b,E_cal_r,E_cal_i,E_loaE,E_store,E_jr,E_jal,E_jalr,E_cal_md,E_md,E_r_md;
    instr_class E_class (.instr(instr_E), .jal(E_jal), .cal_i(E_cal_i),
.cal_r(E_cal_r), .load(E_load), .b(E_b), .jr(E_jr),
.store(E_store),.jalr(E_jalr),.md(E_md),.cal_md(E_cal_md),.r_md(E_r_md));
    wire
M_b,M_cal_r,M_cal_i,M_load,M_store,M_jr,M_jal,M_jalr,M_cal_md,M_md,M_r_md;
    instr_class M_class (.instr(instr_M), .jal(M_jal), .cal_i(M_cal_i),
.cal_r(M_cal_r), .load(M_load), .b(M_b), .jr(M_jr),
.store(M_store),.jalr(M_jalr),.md(M_md),.cal_md(M_cal_md),.r_md(M_r_md));
    wire
W_b,W_cal_r,W_cal_i,W_load,W_store,W_jr,W_jal,W_jalr,W_cal_md,W_md,W_r_md,W_w_md
;
    instr_class W_class (.instr(instr_W), .jal(W_jal), .cal_i(W_cal_i),
.cal_r(W_cal_r), .load(W_load), .b(W_b), .jr(W_jr),
.store(W_store),.jalr(W_jalr),.md(W_md),.cal_md(W_cal_md),.r_md(W_r_md),.w_md(W_w_md));

    wire stallRt,stallRs,stall,stallMd;

    wire [4:0] rs_D,rs_E,rt_D,rt_E,rs_M,rt_M,rd_M,rs_W,rt_W,rd_W,rd_E;
    assign rs_D = instr_D[`rs];
    assign rs_E = instr_E[`rs];
    assign rd_E = instr_E[`rd];
    assign rt_D = instr_D[`rt];
    assign rt_E = instr_E[`rt];
    assign rs_M = instr_M[`rs];
    assign rt_M = instr_M[`rt];
    assign rd_M = instr_M[`rd];
    assign rs_W = instr_W[`rs];
```

```

assign rt_w = instr_w[`rt];
assign rd_w = instr_w[`rd];

//stall logic
assign stallRt = (D_b&&E_load&&rt_E==rt_D&&rt_D)||
(D_b&&E_cal_i&&rt_E==rt_D&&rt_D)|| (D_b&&E_cal_r&&rd_E==rt_D&&rt_D)||
(D_b&&M_load&&rt_M==rt_D&&rt_D)||

(D_cal_r&&E_load&&rt_D==rt_E&&rt_D)||D_cal_md&&E_load&&rt_D==rt_E&&rt_D);
assign stallRs = (D_b&&E_load&&rt_E==rs_D&&rs_D)||
(D_b&&E_cal_i&&rt_E==rs_D&&rs_D)|| (D_b&&E_cal_r&&rd_E==rs_D&&rs_D)||
(D_b&&M_load&&rt_M==rs_D&&rs_D)||
(D_cal_r&&E_load&&rs_D==rt_E&&rs_D)||
(D_cal_i&&E_load&&rs_D==rt_E&&rs_D)|| (D_load&&E_load&&rs_D==rt_E&&rs_D)||
(D_store&&E_load&&rs_D==rt_E&&rs_D)|| (D_cal_md&&E_load&&rs_D==rt_E&&rs_D)||
(D_jr&&E_load&&rt_E==rs_D&&rs_D)||
(D_jr&&E_cal_i&&rt_E==rs_D&&rs_D)|| (D_jr&&E_cal_r&&rd_E==rs_D&&rs_D)||
(D_jr&&M_load&&rt_M==rs_D&&rs_D)||
((D_jalr||D_w_md)&&E_load&&rt_E==rs_D&&rs_D)||
(D_jalr&&E_cal_i&&rt_E==rs_D&&rs_D)|| (D_jalr&&E_cal_r&&rd_E==rs_D&&rs_D)||
(D_jalr&&M_load&&rt_M==rs_D&&rs_D);
assign stallMd=(D_md&&busy)|| (D_md&&E_cal_md);
assign stall = stallRs||stallRt||stallMd;
assign stall_F = ~stall;
assign stall_D = ~stall;
assign flush_E = stall;

//forward logic
assign selRsD = (rs_D==31&&E_jal&&rs_D||rs_D==rd_E&&E_jalr&&rs_D)?1:
(rs_D==rd_E&&E_r_md&&rs_D)?6:
(rs_D==31&&M_jal&&rs_D||rs_D==rd_M&&M_jalr&&rs_D)?2:
(rs_D==rt_M&&(M_cal_i||M_load)&&rs_D)|| (rs_D==rd_M&&M_cal_r&&rs_D)?3:
(rs_D==rd_M&&M_r_md&&rs_D)?7:
(rs_D==31&&W_jal&&rs_D||rs_D==rd_W&&W_jalr&&rs_D)?4:
(rs_D==rt_W&&(W_cal_i||W_load)&&rs_D)|| (rs_D==rd_W&&W_cal_r&&rs_D)||
(rs_D==rd_W&&W_r_md&&rs_D)?5:0;
assign selRtD = (rt_D==31&&E_jal&&rt_D||rt_D==rd_E&&E_jalr&&rt_D)?1:
(rt_D==rd_E&&E_r_md&&rt_D)?6:
(rt_D==31&&M_jal&&rt_D||rt_D==rd_M&&M_jalr&&rt_D)?2:
(rt_D==rt_M&&(M_cal_i||M_load)&&rt_D)|| (rt_D==rd_M&&M_cal_r&&rt_D)?3:
(rt_D==rd_M&&M_r_md&&rt_D)?7:
(rt_D==31&&W_jal&&rt_D||rt_D==rd_W&&W_jalr&&rt_D)?4:
(rt_D==rt_W&&(W_cal_i||W_load)&&rt_D)|| (rt_D==rd_W&&W_cal_r&&rt_D)||
(rt_D==rd_W&&W_r_md&&rt_D)?5:
0;
assign selRsE = (rs_E==31&&M_jal&&rs_E||rs_E==rd_M&&M_jalr&&rs_E)?1:
(rs_E==rt_M&&(M_cal_i||M_load)&&rs_E)|| (rs_E==rd_M&&M_cal_r&&rs_E)?2:
(rs_E==rd_M&&M_r_md&&rs_E)?5:
(rs_E==31&&W_jal&&rs_E||rs_E==rd_W&&W_jalr&&rs_E)?3:
(rs_E==rt_W&&(W_cal_i||W_load)&&rs_E)|| (rs_E==rd_W&&W_cal_r&&rs_E)||
(rs_E==rd_W&&W_r_md&&rs_E)?4:0;

assign selRtE = (rt_E==31&&M_jal&&rt_E||rt_E==rd_M&&M_jalr&&rt_E)?1:
(rt_E==rt_M&&(M_cal_i||M_load)&&rt_E)|| (rt_E==rd_M&&M_cal_r&&rt_E)?2:
(rt_E==rd_M&&M_r_md&&rt_E)?5:
(rt_E==31&&W_jal&&rt_E||rt_E==rd_W&&W_jalr&&rt_E)?3:
(rt_E==rt_W&&(W_cal_i||W_load)&&rt_E)|| (rt_E==rd_W&&W_cal_r&&rt_E)||
(rt_E==rd_W&&W_r_md&&rt_E)?4:0;

```

```

    assign selRtM = (rt_M==31&&w_jal&&rt_M|rt_M==rd_w&&w_jalr&&rt_M)?1:
(rt_M==rt_w&&(w_cal_i|w_load)&&rt_M)|(rt_M==rd_w&&w_cal_r&&rt_M)|
(rt_M==rd_w&&w_r_md&&rt_M)?2:0;
/* wire test_ans;
    assign test_ans=(rs_E==rt_w&&w_cal_i&&rs_E)|(rs_E==rd_w&&w_cal_r&&rs_E);*/
endmodule

```

## 二、测试方案

### 典型测试样例

#### 暂停机制测试

```

#D instr and E instr
#beq cal_r
ori $1 $0 12
addu $1 $1 $0
beq $1 $0 next
ori $4 $0 1234
next:addu $1 $1 $1
#beq cal_i
ori $1 $0 12
beq $1 $0 next
ori $4 $0 1234
next:addu $1 $1 $1
#beq load
ori $1 $0 12
sw $1 0($0)
lw $2 0($0)
beq $2 $0 next
ori $4 $0 1234
next:addu $1 $1 $1
#cal_r load
ori $1 $0 12
sw $1 0($0)
lw $2 0($0)
subu $3 $2 $1
#cal_i load
ori $1 $0 12
sw $1 0($0)
lw $2 0($0)
ori $3 $2 0
#load load
ori $1 $0 12
sw $1 0($0)
lw $2 0($0)
lw $3 0($2)
#store load
ori $1 $0 12
sw $1 0($0)
lw $2 0($0)
sw $2 0($2)
#jr cal_r
ori $1 $0 0x300c

```

```

addu $1 $1 $0
jr $1
ori $4 $0 1234
next:addu $1 $1 $1
#jr cal_i
ori $1 $0 0x300c4
jr $1
ori $4 $0 1234
next:addu $1 $1 $1
#jr load
ori $1 $0 0x3014
sw $1 0($0)
lw $2 0($0)
jr $2
ori $4 $0 1234
next:addu $1 $1 $1

#D instr and M instr
#beq load
ori $1 $0 0x3018
sw $1 0($0)
lw $2 0($0)
ori $5 1234
beq $2 $0 next
ori $4 $0 1234
next:addu $1 $1 $1
#beq load
ori $1 $0 0x3018
sw $1 0($0)
lw $2 0($0)
ori $5 1234
jr $2
ori $4 $0 1234
next:addu $1 $1 $1

```

## 转发机制测试

```

#whole forward test
ori $1 $0 1244
ori $3 $0 1244
addu $4 $1 $3
addu $5 $3 $3
sw $5 0($3)
lw $7 0($3)
jal 0x3024
addu $2 $31 $0
j 0x302c
ori $3 $0 12222 #0x3024
jr $31
jal 0x3038
ori $9 $31 0
j 0x3040
ori $3 $0 12222 #0x3038
jr $31

```

## 综合测试

# 自动测试脚本

## 指令生成代码

```
import random
import os
testnum=3
def cal_i(instr,rt,rs,imm16):
    return "%s $%d $%d %d" % (instr,rt,rs,imm16)
def cal_r(instr,rd,rt,rs):
    return "%s $%d $%d $%d" % (instr,rd,rt,rs)
def jump(instr,label):
    return "%s %s" % (instr,label)
def lui(rt,imm16):
    return "lui $%d %d"%(rt,imm16)
def jr(rs):
    return "jr $%d"%rs
def load(instr,rt,rs,imm16):
    #rs=0
    rs=0
    return "%s $%d %d($%d)"%(instr,rt,imm16,rs)
def store(instr,rt,rs,imm16):
    #rs=0
    rs=0
    return "%s $%d %d($%d)"%(instr,rt,imm16,rs)
cal_r_list=["addu","subu"]
cal_i_list=["ori"]
jump_list=["j","jal"]
br_list=["branch"]
load_list=["lw"]
store_list=["sw"]
def instr(m,rs,rt,rd,imm16,f):
    if m==1:
        a=random.randint(0,len(cal_r_list)-1)
        #print(cal_r(cal_r_list[a],rd,rt,rs))
        f.write(cal_r(cal_r_list[a],rd,rt,rs)+"\n")
    elif m==2:
        a=random.randint(0,len(cal_i_list)-1)
        #print(cal_i(cal_i_list[a],rt,rs,imm16))
        f.write(cal_i(cal_i_list[a],rt,rs,imm16)+"\n")
    elif m==3:
        a=random.randint(0,len(load_list)-1)
        f.write(load(load_list[a],rt,rs,imm16)+"\n")
    elif m==4:
        a=random.randint(0,len(store_list)-1)
        f.write(store(store_list[a],rt,rs,imm16)+"\n")
    else:
        #print(lui(rt,imm16))
        f.write(lui(rt,imm16)+"\n")
def get_one_testpoint(file_name):
    path=os.path.dirname(os.path.realpath(__file__))
    os.chdir(path)
    with open(file_name,"w") as f:
        f.write("ori $gp $0 0"+"\\n")
        f.write("ori $sp $0 0"+"\\n")
```

```

for i in range(0,25):
    rt=random.randint(1,31)
    rs=random.randint(0,31)
    rd=random.randint(0,31)
    imm16=random.randint(0,0x1fff)*4
    imm26_jal=(i+12)*4+0x3000
    imm26_j=(i+19)*4+0x3000
    f.write("circle_%d_%x:"%(i,i*4+0x3000))
    for j in range(2):
        instr(1,rs,rt,rd,imm16,f)
        instr(2,rs,rd,rt,imm16,f)
        instr(3,rt,rd,rs,imm16,f)
    # 9:
    f.write(jump("jal","jal_%d"%i)+"\n")
    f.write("nop"+"\\n")
    f.write(jump("j","j_%d"%i)+"\\n")

    a=random.randint(1,4)
    b=random.randint(1,4)
    c=random.randint(1,4)
    f.write("jal_%d:"%(i))
    instr(a,rs,rt,rd,imm16,f)
    instr(b,rs,rd,rt,imm16,f)
    instr(c,rt,rd,rs,imm16,f)

    a=random.randint(1,4)
    b=random.randint(1,4)
    c=random.randint(1,4)
    instr(a,rs,rt,rd,imm16,f)
    instr(b,rs,rd,rt,imm16,f)
    instr(c,rt,rd,rs,imm16,f)

    for j in range(5):
        if(j==2):
            f.write("j_%d:"%(i))
            f.write("nop"+"\\n")
    f.close()
for k in range(testnum
testnum=3):
    file_name="testpoint%d.asm"%(k+1)
    get_one_testpoint(file_name)

```

### 生成的mips代码举例

```

ori $gp $0 0
ori $sp $0 0
circle_0_3000: addu $22 $7 $20
ori $22 $20 820
lw $22 820($0)
addu $22 $7 $20
ori $22 $20 820
lw $22 820($0)
jal jal_0
nop
j j_0
jal_0: sw $7 820($0)
ori $22 $20 820

```

```

sw $22 820($0)
ori $7 $20 820
sw $22 820($0)
addu $20 $22 $7
nop
nop
j_0:nop
nop
nop
circle_1_3004:addu $6 $22 $15
ori $6 $15 404
lw $6 404($0)
addu $6 $22 $15
ori $6 $15 404
lw $6 404($0)
jal jal_1
nop
j j_1
jal_1:ori $22 $15 404
sw $6 404($0)
addu $15 $6 $22
addu $6 $22 $15
lw $6 404($0)
lw $6 404($0)
nop
nop
j_1:nop
nop
nop
circle_2_3008:addu $14 $14 $1
ori $14 $1 1428
lw $14 1428($0)
addu $14 $14 $1
ori $14 $1 1428
lw $14 1428($0)
jal jal_2
nop
j j_2
jal_2:sw $14 1428($0)
lw $14 1428($0)
ori $14 $14 1428
addu $14 $14 $1
ori $14 $1 1428
addu $1 $14 $14
nop
nop
j_2:nop
nop
nop
circle_3_300c:addu $21 $26 $25
ori $21 $25 1900
lw $21 1900($0)
addu $21 $26 $25
ori $21 $25 1900
lw $21 1900($0)
jal jal_3
nop
j j_3

```

```

jal_3:ori $26 $25 1900
sw $21 1900($0)
ori $21 $26 1900
addu $21 $26 $25
ori $21 $25 1900
ori $21 $26 1900
nop
nop
j_3:nop
nop
nop
circle_4_3010:addu $12 $18 $31
ori $12 $31 140
lw $12 140($0)
addu $12 $18 $31
ori $12 $31 140
lw $12 140($0)
jal jal_4
nop
j j_4
jal_4:lw $18 140($0)
lw $12 140($0)
sw $12 140($0)
addu $12 $18 $31
addu $18 $12 $31
addu $31 $12 $18
nop
nop
j_4:nop
nop
nop
circle_5_3014:addu $15 $1 $11
ori $15 $11 892
lw $15 892($0)
addu $15 $1 $11
ori $15 $11 892
lw $15 892($0)
jal jal_5
nop
j j_5
jal_5:ori $1 $11 892
ori $15 $11 892
ori $15 $1 892
ori $1 $11 892
sw $15 892($0)
ori $15 $1 892
nop
nop
j_5:nop
nop
nop
circle_6_3018:addu $17 $27 $7
ori $17 $7 20
lw $17 20($0)
addu $17 $27 $7
ori $17 $7 20
lw $17 20($0)
jal jal_6

```



```
nop
j j_6
jal_6:addu $17 $27 $7
ori $17 $7 20
addu $7 $17 $27
addu $17 $27 $7
addu $27 $17 $7
lw $17 20($0)
nop
nop
j_6:nop
nop
nop
circle_7_301c:addu $8 $22 $27
ori $8 $27 440
lw $8 440($0)
addu $8 $22 $27
ori $8 $27 440
lw $8 440($0)
jal jal_7
nop
j j_7
jal_7:addu $8 $22 $27
sw $8 440($0)
ori $8 $22 440
ori $22 $27 440
ori $8 $27 440
addu $27 $8 $22
nop
nop
j_7:nop
nop
nop
circle_8_3020:addu $30 $5 $7
ori $30 $7 776
lw $30 776($0)
addu $30 $5 $7
ori $30 $7 776
lw $30 776($0)
jal jal_8
nop
j j_8
jal_8:lw $5 776($0)
sw $30 776($0)
ori $30 $5 776
ori $5 $7 776
ori $30 $7 776
addu $7 $30 $5
nop
nop
j_8:nop
nop
nop
circle_9_3024:addu $3 $24 $24
ori $3 $24 1540
lw $3 1540($0)
addu $3 $24 $24
ori $3 $24 1540
```

```

lw $3 1540($0)
jal jal_9
nop
j j_9
jal_9:ori $24 $24 1540
addu $24 $3 $24
addu $24 $3 $24
lw $24 1540($0)
addu $24 $3 $24
sw $3 1540($0)
nop
nop
j_9:nop
nop
nop
circle_10_3028:addu $27 $1 $31
ori $27 $31 668
lw $27 668($0)
addu $27 $1 $31
ori $27 $31 668
lw $27 668($0)
jal jal_10
nop
j j_10
jal_10:lw $1 668($0)
lw $27 668($0)
sw $27 668($0)
lw $1 668($0)
ori $27 $31 668
addu $31 $27 $1
nop
nop
j_10:nop
nop
nop
circle_11_302c:addu $8 $19 $25
ori $8 $25 832
lw $8 832($0)
addu $8 $19 $25
ori $8 $25 832
lw $8 832($0)
jal jal_11
nop
j j_11
jal_11:ori $19 $25 832
sw $8 832($0)
addu $25 $8 $19
addu $8 $19 $25
sw $8 832($0)
ori $8 $19 832
nop
nop
j_11:nop
nop
nop
circle_12_3030:addu $23 $13 $10
ori $23 $10 1776
lw $23 1776($0)

```

```
addu $23 $13 $10
ori $23 $10 1776
lw $23 1776($0)
jal jal_12
nop
j j_12
jal_12:addu $23 $13 $10
ori $23 $10 1776
ori $23 $13 1776
ori $13 $10 1776
addu $13 $23 $10
lw $23 1776($0)
nop
nop
j_12:nop
nop
nop
circle_13_3034:addu $8 $4 $31
ori $8 $31 1440
lw $8 1440($0)
addu $8 $4 $31
ori $8 $31 1440
lw $8 1440($0)
jal jal_13
nop
j j_13
jal_13:lw $4 1440($0)
addu $4 $8 $31
addu $31 $8 $4
ori $4 $31 1440
sw $8 1440($0)
addu $31 $8 $4
nop
nop
j_13:nop
nop
nop
circle_14_3038:addu $19 $17 $5
ori $19 $5 572
lw $19 572($0)
addu $19 $17 $5
ori $19 $5 572
lw $19 572($0)
jal jal_14
nop
j j_14
jal_14:ori $17 $5 572
lw $19 572($0)
sw $19 572($0)
ori $17 $5 572
lw $19 572($0)
lw $19 572($0)
nop
nop
j_14:nop
nop
nop
circle_15_303c:addu $4 $9 $2
```

```
ori $4 $2 1868
lw $4 1868($0)
addu $4 $9 $2
ori $4 $2 1868
lw $4 1868($0)
jal jal_15
nop
j j_15
jal_15:addu $4 $9 $2
sw $4 1868($0)
ori $4 $9 1868
addu $4 $9 $2
ori $4 $2 1868
addu $2 $4 $9
nop
nop
j_15:nop
nop
nop
circle_16_3040:addu $4 $8 $8
ori $4 $8 1988
lw $4 1988($0)
addu $4 $8 $8
ori $4 $8 1988
lw $4 1988($0)
jal jal_16
nop
j j_16
jal_16:sw $8 1988($0)
addu $8 $4 $8
sw $4 1988($0)
lw $8 1988($0)
sw $4 1988($0)
lw $4 1988($0)
nop
nop
j_16:nop
nop
nop
circle_17_3044:addu $31 $4 $30
ori $31 $30 1632
lw $31 1632($0)
addu $31 $4 $30
ori $31 $30 1632
lw $31 1632($0)
jal jal_17
nop
j j_17
jal_17:ori $4 $30 1632
sw $31 1632($0)
ori $31 $4 1632
ori $4 $30 1632
sw $31 1632($0)
lw $31 1632($0)
nop
nop
j_17:nop
nop
```

```
nop
circle_18_3048:addu $19 $20 $22
ori $19 $22 2036
lw $19 2036($0)
addu $19 $20 $22
ori $19 $22 2036
lw $19 2036($0)
jal jal_18
nop
j j_18
jal_18:sw $20 2036($0)
sw $19 2036($0)
lw $19 2036($0)
addu $19 $20 $22
addu $20 $19 $22
ori $19 $20 2036
nop
nop
j_18:nop
nop
nop
circle_19_304c:addu $10 $24 $10
ori $10 $10 280
lw $10 280($0)
addu $10 $24 $10
ori $10 $10 280
lw $10 280($0)
jal jal_19
nop
j j_19
jal_19:addu $10 $24 $10
addu $24 $10 $10
ori $10 $24 280
addu $10 $24 $10
lw $10 280($0)
ori $10 $24 280
nop
nop
j_19:nop
nop
nop
circle_20_3050:addu $10 $7 $28
ori $10 $28 12
lw $10 12($0)
addu $10 $7 $28
ori $10 $28 12
lw $10 12($0)
jal jal_20
nop
j j_20
jal_20:addu $10 $7 $28
addu $7 $10 $28
ori $10 $7 12
sw $7 12($0)
ori $10 $28 12
ori $10 $7 12
nop
nop
```

```
j_20:nop
nop
nop
circle_21_3054:addu $22 $27 $12
ori $22 $12 484
lw $22 484($0)
addu $22 $27 $12
ori $22 $12 484
lw $22 484($0)
jal jal_21
nop
j j_21
jal_21:addu $22 $27 $12
lw $22 484($0)
sw $22 484($0)
sw $27 484($0)
ori $22 $12 484
ori $22 $27 484
nop
nop
j_21:nop
nop
nop
circle_22_3058:addu $28 $20 $25
ori $28 $25 1844
lw $28 1844($0)
addu $28 $20 $25
ori $28 $25 1844
lw $28 1844($0)
jal jal_22
nop
j j_22
jal_22:lw $20 1844($0)
ori $28 $25 1844
addu $25 $28 $20
addu $28 $20 $25
addu $20 $28 $25
addu $25 $28 $20
nop
nop
j_22:nop
nop
nop
circle_23_305c:addu $28 $20 $22
ori $28 $22 828
lw $28 828($0)
addu $28 $20 $22
ori $28 $22 828
lw $28 828($0)
jal jal_23
nop
j j_23
jal_23:sw $20 828($0)
ori $28 $22 828
addu $22 $28 $20
sw $20 828($0)
ori $28 $22 828
lw $28 828($0)
```

```

nop
nop
j_23:nop
nop
nop
circle_24_3060:addu $30 $28 $25
ori $30 $25 1296
lw $30 1296($0)
addu $30 $28 $25
ori $30 $25 1296
lw $30 1296($0)
jal jal_24
nop
j j_24
jal_24:lw $28 1296($0)
lw $30 1296($0)
addu $25 $30 $28
lw $28 1296($0)
addu $28 $30 $25
lw $30 1296($0)
nop
nop
j_24:nop
nop
nop

```

## 自动测试

需要环境: python3环境, 且装好os,re,random库

```

import os
import re
import random
data_source=input("describe the test data:")
#data_source=""
case=input("Your test files num >1?(y/n):")==="y"
#case=
#xlinx=input("your xilinx src is:")
xlinx="D:\\Xilinx\\14.7\\ISE_DS\\ISE"
if(case):
    file_src=input("input asm test files folder src:(must like this
D:\\P5\\test\\)")
    #file_src=""
    file_list=os.listdir(file_src)
    k=1
    for name in file_list:
        namek="testpoint"+str(k)+".asm"
        try:
            os.rename(file_src+name,file_src+namek)
        except:
            pass
        k=k+1
    file_num=k-1
    error_list=[]
    wrong_cnt=0
    for j in range(5,file_num+1):
        path=os.path.dirname(os.path.realpath(__file__))

```

```

os.chdir(path)
asmfilename=file_src+"testpoint"+str(j)+".asm"
time="10us"
filelist=os.walk(path)
os.environ['XILINX']=xlinx
with open("mips.prj","w") as prj:
    for folder in filelist:
        for file in folder[2]:
            if(len(file.split("."))>1 and file.split(".")[1]=="v"):
                prj.write("verilog work \"\""+folder[0]+"\\\""+file+"\\\"\\n")
with open("mips.tcl","w") as tcl:
    tcl.write("run "+time+";\nexit;\n")
os.system("java -jar Mars.jar db a nc mc CompactDataAtZero dump .text
HexText code.txt 1000000 "+asmfilename)
os.system("java -jar Mars.jar db nc mc CompactDataAtZero dump .text
HexText code.txt >ans.txt 1000000 "+asmfilename)
os.system(xlinx+"\\bin\\nt64\\fuse "+ "--nodebug --prj mips.prj -o
mips.exe mipsAutoTest >log.txt")
os.system("mips.exe -nolog -tclbatch mips.tcl >my.txt")

process=0
with open("my.txt","r") as my:
    lines=my.readlines()
    if(len(lines)==0):
        print("fail to simulate")
        os._exit(1)
    if(lines[0][0]=='I'):
        process=1
my.close()
n=0
while(1):
    if(lines[n][0]=="@"):
        break
    else:
        n=n+1
if(process):
    with open("my.txt","w") as my:
        my.writelines(lines[n:])
i=0
biao=0
with open("my.txt","r") as my:
    with open("ans.txt","r") as ans:
        while(1):
            i+=1
            l1=my.readline().strip()
            l2=ans.readline().strip()
            if((l1== "" or l1==None)and(l2=="" or l2==None)):
                break
            elif l1!=l2 and not "$ 0"in l2 and not "$ 0" in l1:
                biao=1
                error_list.append(asmfilename)
            try:
                code_line=int(l1[6:9],16)/4+1
            except:
                code_line=0
            if l2=="" or l2 == None:

```



```

        print("wrong answer occur in line %d of your ans and
in line %d of test code:"%(i,code_line)+"we got "+l1+" when we expected
Nothing")

        else:
            print("wrong answer occur in line %d of your ans and
in line %d of test code:"%(i,code_line)+"we got "+l1+" when we expected "+l2)
            break

        if biao==0:
            print("Accepted on the point %d"%j)
        else:
            print("wrong on the point %d"%j)
            wrong_cnt=wrong_cnt+1

    my.close()
    ans.close()
    if(wrong_cnt):
        print("-----\nYou have %d wrong answer\n
test data description:%s\nthey are:"%(wrong_cnt,data_source))
        for x in error_list:
            print(x)
    else:
        print("-----\ncongratulations!You
accepted %d point\ntest data description:%s"%(file_num,data_source))
        os._exit(1)
else:
    file_name=input("input asm test file name(can have src):")
    path=os.path.dirname(os.path.realpath(__file__))
    os.chdir(path)
    asmfilename=file_name
    time="10us"
    filelist=os.walk(path)
    os.environ['XILINX']=xlinx
    with open("mips.prj","w") as prj:
        for folder in filelist:
            for file in folder[2]:
                if(len(file.split("."))>1 and file.split(".")[1]=="v"):
                    prj.write("verilog work \"\""+folder[0]+"\""+file+"\""\n")
    with open("mips.tcl","w") as tcl:
        tcl.write("run "+time+";\nexit;\n")
    os.system("java -jar Mars.jar db a nc mc CompactDataAtZero dump .text
HexText code.txt 1000000 "+asmfilename)
    os.system("java -jar Mars.jar db nc mc CompactDataAtZero dump .text HexText
code.txt >ans.txt 1000000 "+asmfilename)
    os.system(xlinx+"\\bin\\nt64\\fuse "+"--nodebug --prj mips.prj -o mips.exe
mipsAutoTest >log.txt")
    os.system("mips.exe -nolog -tclbatch mips.tcl >my.txt")

    process=0
    with open("my.txt","r") as my:
        lines=my.readlines()
        if(len(lines)==0):
            print("fail to simulate")
            os._exit(1)
        if(lines[0][0]=='I'):
            process=1
    my.close()
    n=0
    while(1):

```

```






























        if(lines[n][0]=="@"):
            break
        else:
            n=n+1
    if(process):
        with open("my.txt","w") as my:
            my.writelines(lines[n:])
    i=0
    biao=0
    wrong_cnt=0
    with open("my.txt","r") as my:
        with open("ans.txt","r") as ans:
            while(1):
                i+=1
                l1=my.readline().strip()
                l2=ans.readline().strip()
                if((l1== "" or l1==None)and(l2=="" or l2==None)):
                    break
                elif l1!=l2 and not "$ 0"in l2 and not "$ 0" in l1:
                    biao=1
                    try:
                        code_line=int("0x"+l1[6:9],16)/4+1
                    except:
                        code_line=0
                    if l2=="" or l2 == None:
                        print("Wrong answer occur in line %d of your ans and in
line %d of test code:"%(i,code_line)+"we got "+l1+" when we expected Nothing")
                    else:
                        print("Wrong answer occur in line %d of your ans and in
line %d of test code:"%(i,code_line)+"we got "+l1+" when we expected "+l2)
                    break

            if biao==0:
                print("Accepted on this point")
            else:
                print("Wrong on the point")
                wrong_cnt=wrong_cnt+1
    my.close()
    ans.close()
    os._exit(1)

```

此脚本实现了批量命名MIPS代码文件为：testpoint*i*.asm ( $1 \leq i \leq n$ ,  $n$ 为文件名)，以及通过增加文件路径的输入适配不同电脑不同地址的情况

以下为自动改名的情况：

 testpoint1.asm	2019/11/23 20:54	ASM 文件	1 KB
 testpoint2.asm	2019/11/23 15:00	ASM 文件	1 KB
 testpoint3.asm	2019/11/24 14:47	ASM 文件	1 KB
 testpoint4.asm	2019/11/21 10:46	ASM 文件	1 KB
 testpoint5.asm	2019/11/23 16:11	ASM 文件	1 KB
 testpoint6.asm	2019/11/23 15:51	ASM 文件	1 KB
 testpoint7.asm	2019/11/21 1:49	ASM 文件	1 KB
 testpoint8.asm	2019/11/23 21:48	ASM 文件	1 KB
 testpoint9.asm	2019/11/23 14:09	ASM 文件	1 KB
 testpoint10.asm	2019/11/21 11:01	ASM 文件	1 KB
 testpoint11.asm	2019/11/21 11:03	ASM 文件	1 KB
 testpoint12.asm	2019/11/21 0:57	ASM 文件	1 KB
 testpoint13.asm	2019/11/24 15:04	ASM 文件	1 KB
 testpoint14.asm	2019/11/21 10:59	ASM 文件	1 KB
 testpoint15.asm	2019/11/23 14:23	ASM 文件	1 KB
 testpoint16.asm	2019/11/21 2:17	ASM 文件	5 KB
 testpoint17.asm	2019/11/24 14:52	ASM 文件	1 KB
 testpoint18.asm	2019/11/21 11:14	ASM 文件	1 KB
 testpoint19.asm	2019/11/21 11:33	ASM 文件	1 KB
 testpoint20.asm	2019/11/24 14:56	ASM 文件	1 KB
 testpoint21.asm	2019/11/24 14:54	ASM 文件	1 KB
 testpoint22.asm	2019/11/23 16:18	ASM 文件	1 KB
 testpoint23.asm	2019/11/23 21:24	ASM 文件	1 KB
 testpoint24.asm	2019/11/24 15:12	ASM 文件	1 KB
 testpoint25.asm	2019/11/24 15:17	ASM 文件	1 KB
 testpoint26.asm	2019/11/24 15:20	ASM 文件	1 KB
 testpoint27.asm	2019/11/24 15:26	ASM 文件	1 KB
 testpoint28.asm	2019/11/24 15:31	ASM 文件	1 KB
 testpoint29.asm	2019/11/23 16:53	ASM 文件	1 KB

以下为多个测试用例时的情况：



以下为单个用例的测试情况演示：

```
PS C:\Users\Administrator> & D:/Anaconda3/python.exe g:/P5_test/test.py
describe the test data:mips3.asm
Your test files num >1?(y/n):n
input asm test file name(can have src):mips3.asm
WARNING:HDLCompiler:1016 - "G:\P5_test\mips.v" Line 27: Port aluSrc is not connected to this instance
WARNING:HDLCompiler:1016 - "G:\P5_test\mips.v" Line 42: Port regDst is not connected to this instance
WARNING:HDLCompiler:1016 - "G:\P5_test\mips.v" Line 56: Port regDst is not connected to this instance
WARNING:HDLCompiler:1016 - "G:\P5_test\mips.v" Line 66: Port aluSrc is not connected to this instance
Accepted on this point
```

### 三、思考题

**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？**

乘除法运算慢，需要单独模拟。在E级单独使用，从而不影响GRF。

**参照你对延迟槽的理解，试解释“乘除槽”。**

当乘除法正在进行运算时，将与乘除有关的指令阻塞在D级，即进入槽内但被阻塞，和延迟槽类似。

**举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint：考虑C语言中字符串的情况）**

当字节访问多的时候更有优势（如字符串操作）

**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？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。此思考题请同学们结合自己测试CPU使用的具体手段，按照自己的实际情况进行回答**

和P5几乎一样。列表，在stall里判断是否必须暂停，在forward中判断在哪里可以转发、转发的选择信号是什么。

测试样例主要是自动生成以及结合往年学长以及评论区的测试数据。

**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？**

每一级的相同信号命名采用 公用名字\_流水级 这样的形式，比如：instr\_M，regWrite\_W等

采用指令分类模块，简化了译码时的复杂程度