

P4 Verilog CPU 设计文档

23373526 华家璇

一、设计草稿

(一) 部件设计

1. ALU

端口定义如下

端口名	方向	描述
A[31:0]	I	输入数据A
B[31:0]	I	输入数据B
ALUOp[2:0]	I	操作码
result[31:0]	O	输出

功能定义如下

ALUOp	O
000	A&B
001	A
010	A+B
011	A-B
100	A<B ? 1:0
101	sign(A)<sign(B) ? 1:0

2. GRF

端口定义如下

端口名	方向	描述
clk	I	时钟信号
reset	I	异步复位
wenable	I	写使能
waddr[4:0]	I	写地址
raddr2[4:0]	I	读地址1
raddr1[4:0]	I	读地址2

端口名	方向	描述
wdata[31:0]	I	写入数据
rdata1[31:0]	O	ra1寄存器数据
rdata2[31:0]	O	ra2寄存器数据

功能定义如下

- we信号为高电平时，写使能有效，在clk上升沿到来时写入数据WD到waddr对应的寄存器中
- 始终输出ra1寄存器数据到d1, ra2寄存器数据到d2
- 异步复位re有效时，将所有寄存器数据清零

3. EXT

端口定义如下

端口名	方向	描述
imm16[15:0]	I	输入的立即数
extOp[1:0]	I	操作码
out32[31:0]	O	扩展后的立即数

功能定义如下

extOp	out32
00	无符号扩展imm16到32位
01	符号扩展imm16到32位
10	将imm16加载到高16位

4. NEXTPC

端口定义如下

端口名	方向	描述
pcF[31:0]	I	输入F阶段程序计数器
pcD[31:0]	I	输入D阶段程序计数器
imm16[15:0]	I	输入16位立即数
imm26[25:0]	I	输入26位立即数
imm32[31:0]	I	输入32位立即数
branch	I	输入分支信号
jump	I	输入跳转信号
jr	I	输入跳转寄存器信号
zero	I	输入零标志信号

端口名	方向	描述
next_pc[31:0]	O	输出下一个程序计数器

功能定义如下

1. 每次时钟上升沿, $pc \leftarrow pc + 4$, 取出下一条指令
2. 当branch有效, 并且zero为1时, $pc \leftarrow pc + 4 + sign_ext(imm16 << 2)$
3. 当jump有效, $pc \leftarrow \{pc[31:28], instrIndex, 2'b0\}$
4. 当jr有效, $pc \leftarrow imm32$

5. Controller

端口定义如下

端口名	方向	描述
instr[31:0]	I	指令字
regDst[2:0]	O	寄存器目标选择信号
aluSrc	O	ALU操作数来源选择信号
memToReg	O	内存到寄存器选择信号
regWrite	O	寄存器写使能信号
memWrite	O	内存写使能信号
branch	O	分支信号
jump	O	跳转信号
jr	O	跳转寄存器信号
extOp[2:0]	O	立即数扩展操作信号
aluOp[2:0]	O	ALU操作选择信号
memByteen[2:0]	O	内存字节使能信号
cmpOp[1:0]	O	比较操作信号
mdop[2:0]	O	乘除法操作选择信号
hlsel	O	高低寄存器选择信号
hlread	O	高低寄存器读使能信号
hlwrite	O	高低寄存器写使能信号
mdstart	O	乘除法启动信号

功能定义如下

1. RegDst: 选择写入的寄存器, 0->rt, 1->rd, 2->ra
2. ALUSrc: 选择ALU数据来源, 0->rt, 1->ext
3. MemToReg: 选择GRF数据来源, 0->ALU, 1->DM
4. MemWrite: 选择是否写入存储器
5. Branch: 是否存在条件分支

6. Jump: 是否存在无条件跳转
7. RegWrite: 选择是否写入寄存器
8. ALUOp: 选择ALU操作码
9. extOp: 选择扩展操作码
10. jr: 是否发生寄存器跳转
11. cmpOp: 选择比较操作码
12. mdop: 选择乘除法操作码
13. hsel: 选择高低寄存器选择码
14. hlread: 选择高低寄存器读使能码
15. hlwrite: 选择高低寄存器写使能码
16. mdstart: 选择乘除法启动码

当前指令真值表如下

instruction	RegDst	ALUSrc	MemToReg	MemWrite	Branch	Jump	RegWrite	ALUOp	extOp	jr
add	001	0	0	0	0	0	1	010	000	0
sub	001	0	0	0	0	0	1	011	000	0
ori	000	1	0	0	0	0	1	001	000	0
beq	000	0	0	0	1	0	0	100	000	0
lui	000	1	0	0	0	0	1	010	010	0
lw	000	1	1	0	0	0	1	010	001	0
sw	000	1	0	1	0	0	0	010	001	0
jal	010	0	0	0	0	1	1	000	001	0
jr	000	0	0	0	0	0	0	010	001	1
jalr	001	0	0	0	0	0	1	010	001	1
j	000	0	0	0	0	1	0	000	001	0

6. cmp

端口定义如下

端口名	方向	描述
A[31:0]	I	操作数 A
B[31:0]	I	操作数 B
cmpOp[1:0]	I	比较操作选择信号
zero	O	比较结果标志

功能定义如下

1. 根据 cmpOp 的值进行比较操作。
2. 当 cmpOp 为 2'b00 时，判断 A 和 B 是否相等，若相等，标志信号设为 1；否则为 0。
3. 当 cmpOp 为 2'b01 时，判断 A 和 B 是否相等，若相等，标志信号设为 0；否则为 1。
4. 其余情况标志信号设为 0。

7. DE

端口定义如下

端口名	方向	描述
addr[1:0]	I	地址偏移
Din[31:0]	I	输入数据
Op[2:0]	I	数据扩展操作信号
Dout[31:0]	O	扩展后的数据

功能定义如下

- 根据 Op 的值执行不同的数据扩展操作。
- 当 Op 为 $3'b000$ 时，直接传递数据 Din 。
- 当 Op 为 $3'b001$ 时，选择 $addr$ 指定的字节，并对其进行零扩展为 32 位数据。
- 当 Op 为 $3'b010$ 时，选择 $addr$ 指定的字节，并对其进行符号扩展为 32 位数据。
- 当 Op 为 $3'b011$ 时，选择 $addr[1]$ 指定的半字，并对其进行零扩展为 32 位数据。
- 当 Op 为 $3'b100$ 时，选择 $addr[1]$ 指定的半字，并对其进行符号扩展为 32 位数据。
- 默认情况下，直接传递数据 Din 。

8. insExtract

提取指令中的各部分

端口定义如下

端口名	方向	描述
instr[31:0]	I	指令
imm16[15:0]	O	指令中的立即数
rs[4:0]	O	指令中rs地址
rt[4:0]	O	指令中rt地址
rd[4:0]	O	指令中rd地址
imm26[25:0]	O	J型指令中的地址

9. hazardcontrol

端口定义如下

端口名	方向	描述
rsD[4:0]	I	D 阶段源寄存器 rs
rtD[4:0]	I	D 阶段目标寄存器 rt
rsE[4:0]	I	E 阶段源寄存器 rs
rtE[4:0]	I	E 阶段目标寄存器 rt
raE[4:0]	I	E 阶段目标寄存器地址

端口名	方向	描述
raM[4:0]		M 阶段目标寄存器地址
raW[4:0]		W 阶段目标寄存器地址
branchD		D 阶段分支信号
jrD		D 阶段跳转寄存器信号
zero		零标志信号
jumpD		D 阶段跳转信号
jumpM		M 阶段跳转信号
regWriteE		E 阶段寄存器写使能信号
regWriteM		M 阶段寄存器写使能信号
regWriteW		W 阶段寄存器写使能信号
memToRegE		E 阶段内存到寄存器信号
memToRegM		M 阶段内存到寄存器信号
busyE		E 阶段乘除法器忙碌信号
hlreadD		D 阶段高低寄存器读信号
mdstartE		E 阶段乘除法器启动信号
hlwriteD		D 阶段高低寄存器写信号
mdstartD		D 阶段乘除法器启动信号
clearDelaySlot		清除延迟槽信号
FowardA[2:0]	O	前递信号 A
FowardB[2:0]	O	前递信号 B
FowardAD[2:0]	O	D 阶段前递信号 A
FowardBD[2:0]	O	D 阶段前递信号 B
stallPC	O	停止程序计数器信号
stallF2D	O	停止 F 阶段到 D 阶段信号
stallD2E	O	停止 D 阶段到 E 阶段信号
stallE2M	O	停止 E 阶段到 M 阶段信号
stallM2W	O	停止 M 阶段到 W 阶段信号
ClrE2M	O	清除 E 阶段到 M 阶段信号
ClrD2E	O	清除 D 阶段到 E 阶段信号
ClrF2D	O	清除 F 阶段到 D 阶段信号

冲突共以下几种情况

- E级 rs rt 需要用到M、W级的结果，通过ForwardA、ForwardB控制
 - 2'b00 : 无需写回
 - 2'b01 : 写回W级
 - 2'b10 : 写回M级
 - 2'b11 : 0寄存器
- E级需要用到M级jal 结果，停顿一周期再转发
- load-use 冲突， D级rs 或 rt 需要用到E级load结果，停顿一周期再转发
- D级 beq 比较rs rt 需要用到后续的结果，通过ForwardAD、ForwardBD控制
 - 2'b00 : 无需写回
 - 2'b01 : 写回W级
 - 2'b10 : 写回M级
 - 2'b11 : 0寄存器

若 需要用到 E级结果， 停顿一周期， 再写回

若 需要用到 M级读内存结果， 停顿一周期， 再写回

若 需要用到 E读内存级结果， 停顿两周期， 再写回
- D级jr
 - 2'b00 : 无需写回
 - 2'b01 : 写回W级
 - 2'b10 : 写回M级
 - 2'b11 : 0寄存器

若 需要用到 E级结果， 停顿一周期， 再写回

若 需要用到 M级读内存结果， 停顿一周期， 再写回

若 需要用到 E读内存级结果， 停顿两周期， 再写回

10. FDreg

F/D 级寄存器

端口定义如下

端口名	方向	描述
clk	I	输入时钟信号
reset	I	输入复位信号
stall	I	输入停顿信号
instr[31:0]	I	输入指令
pc[31:0]	I	输入程序计数器
instr_out[31:0]	O	输出指令
pc_out[31:0]	O	输出程序计数器

11. DEreg

D/E 级寄存器

端口定义如下

端口名	方向	描述
clk	I	输入时钟信号
reset	I	输入复位信号
pc[31:0]	I	输入程序计数器
regaddr[4:0]	I	输入寄存器地址
rdata1[31:0]	I	输入寄存器1数据
rdata2[31:0]	I	输入寄存器2数据
imm32[31:0]	I	输入立即数
aluOp[2:0]	I	输入ALU操作码
rs[4:0]	I	输入寄存器rs
rt[4:0]	I	输入寄存器rt
aluSrc	I	输入ALU源选择信号
memToReg	I	输入写回数据选择信号
regWrite	I	输入写回使能信号
memWrite	I	输入内存写使能信号
branch	I	输入分支信号
jump	I	输入跳转信号
stall	I	输入停顿信号
memByteen[2:0]	I	内存字节使能信号
mdop[2:0]	I	乘除法操作信号
hlsel	I	高低寄存器选择信号
hlwrite	I	高低寄存器写信号
hlread	I	高低寄存器读信号
mdstart	I	乘除法器启动信号
memByteen_out[2:0]	O	内存字节使能输出信号
mdop_out[2:0]	O	乘除法操作输出信号
hlsel_out	O	高低寄存器选择输出信号
hlwrite_out	O	高低寄存器写输出信号
hlread_out	O	高低寄存器读输出信号
mdstart_out	O	乘除法器启动输出信号
pc_out[31:0]	O	输出程序计数器

端口名	方向	描述
regaddr_out[4:0]	O	输出寄存器地址
rdata1_out[31:0]	O	输出寄存器1数据
rdata2_out[31:0]	O	输出寄存器2数据
imm32_out[31:0]	O	输出立即数
aluOp_out[2:0]	O	输出ALU操作码
rs_out[4:0]	O	输出寄存器rs
rt_out[4:0]	O	输出寄存器rt
aluSrc_out	O	输出ALU源选择信号
memToReg_out	O	输出写回数据选择信号
regWrite_out	O	输出写回使能信号
memWrite_out	O	输出内存写使能信号
branch_out	O	输出分支信号
jump_out	O	输出跳转信号

12. EMreg

E/M 级寄存器

端口定义如下

端口名	方向	描述
clk	I	输入时钟信号
reset	I	输入复位信号
pc[31:0]	I	输入程序计数器
regaddr[4:0]	I	输入寄存器地址
alures[31:0]	I	输入ALU运算结果
memToReg	I	输入写回数据选择信号
regWrite	I	输入写回使能信号
rdata2[31:0]	I	输入寄存器2数据
memWrite	I	输入内存写使能信号
branch	I	输入分支信号
jump	I	输入跳转信号
memByteen[2:0]	I	内存字节使能信号
memByteen_out[2:0]	O	内存字节使能输出信号

端口名	方向	描述
pc_out[31:0]	O	输出程序计数器
regaddr_out[4:0]	O	输出寄存器地址
alures_out[31:0]	O	输出ALU运算结果
memToReg_out	O	输出写回数据选择信号
regWrite_out	O	输出写回使能信号
rdata2_out[31:0]	O	输出寄存器2数据
memWrite_out	O	输出内存写使能信号
branch_out	O	输出分支信号
jump_out	O	输出跳转信号

13. WBreg

M/W 级寄存器

端口定义如下

端口名	方向	描述
clk	I	输入时钟信号
reset	I	输入复位信号
pc[31:0]	I	输入程序计数器
regaddr[4:0]	I	输入寄存器地址
alures[31:0]	I	输入ALU运算结果
memres[31:0]	I	输入内存读取结果
memToReg	I	输入写回数据选择信号
regWrite	I	输入写回使能信号
jump	I	输入跳转信号
pc_out[31:0]	O	输出程序计数器
regaddr_out[4:0]	O	输出寄存器地址
alures_out[31:0]	O	输出ALU运算结果
memres_out[31:0]	O	输出内存读取结果
memToReg_out	O	输出写回数据选择信号
regWrite_out	O	输出写回使能信号
jump_out	O	输出跳转信号

14. muldiv

乘除法器

端口定义如下

端口名	方向	描述
clk	I	时钟信号
reset	I	复位信号
we	I	写使能信号
op[2:0]	I	操作类型信号
a[31:0]	I	操作数 A
b[31:0]	I	操作数 B
sel	I	高低寄存器选择信号
start	I	开始信号
busy	O	运算忙碌标志信号
HI[31:0]	O	高位寄存器输出
LO[31:0]	O	低位寄存器输出

功能定义如下

op	o
000	mult
001	multu
010	div
011	divu

15. PC

PC寄存器

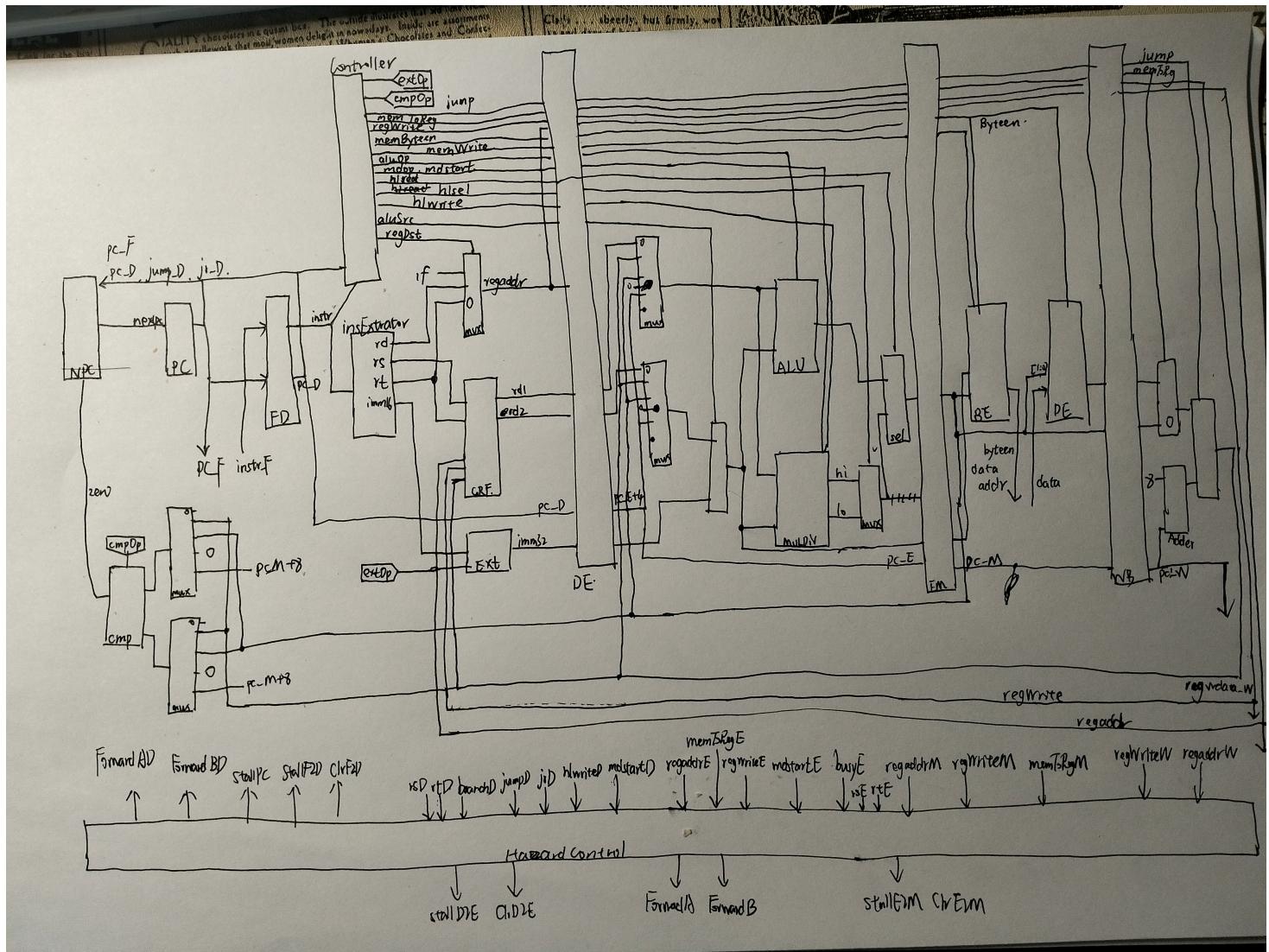
端口定义如下

端口名	方向	描述
clk	I	时钟信号
reset	I	复位信号
stall	I	停止信号
next_pc[31:0]	I	下一条指令地址
pc[31:0]	O	当前指令地址

(二) CPU主体设计

设计草案

如图



二、测试方案

采用自制测评机生成数据并对cpu输出和mars模拟输出进行对比

```
import subprocess
import re
import random

def gencode(num):
    strlist = []
    curlabel = 0
    lastisjump = False
    lastisMD = False
    for i in range(0,31):
        nu = random.randint(0,65535)
        strlist.append("ori $" + str(i) + ",$0," + str(nu))

    for i in range(0, num):
        cmdnum = random.randint(0, 28)
        cmd = ''
        lastisMD = False
        lastisjump = False
        if cmdnum == 0:
            cmd = "add"
            strlist.append(cmd + '$' + str(random.randint(0, 30)) + "$" + str(random.randint(0, 31)) + "$" + str(random.randint(0, 65535)))
        elif cmdnum == 1:
            cmd = "sub"
            strlist.append(cmd + '$' + str(random.randint(0, 30)) + "$" + str(random.randint(0, 31)) + "$" + str(random.randint(0, 65535)))
        elif cmdnum == 2:
            cmd = "lui"
            strlist.append(cmd + '$' + str(random.randint(0, 30)) + "," + str(random.randint(0, 65535)))
        elif cmdnum == 3:
            cmd = "lw"
            strlist.append(cmd + '$' + str(random.randint(0, 30)) + "," + str(random.randint(0, 3071)*4)))
        elif cmdnum == 4:
            cmd = "sw"
            strlist.append(cmd + '$' + str(random.randint(0, 31)) + "," + str(random.randint(0, 3071)*4)))
        elif cmdnum == 5:
            cmd = 'lb'
            strlist.append(cmd + '$' + str(random.randint(0, 31)) + "," + str(random.randint(0, 3071*4-1))))
        elif cmdnum == 6:
            cmd = 'sb'
            strlist.append(cmd + '$' + str(random.randint(0, 31)) + "," + str(random.randint(0, 3071*4-1))))
        elif cmdnum == 7:
            cmd = 'lh'
            strlist.append(cmd + '$' + str(random.randint(0, 31)) + "," + str(random.randint(0, 6142)*2)))
        elif cmdnum == 8:
            cmd = 'sh'
            strlist.append(cmd + '$' + str(random.randint(0, 31)) + "," + str(random.randint(0, 6142)*2)))
        elif cmdnum == 9:
            cmd = 'addi'
```

```

        strlist.append(cmd + '$' + str(random.randint(0, 30)) + "," + str(random.randint(0, 31)) + "," + str(random.randint(0, 31)))

    elif cmdnum == 10:
        cmd = 'andi'
        strlist.append(cmd + '$' + str(random.randint(0, 30)) + "," + str(random.randint(0, 31)) + "," + str(random.randint(0, 31)))

    elif cmdnum == 11:
        cmd = 'and'
        strlist.append(cmd + '$' + str(random.randint(0, 30)) + "," + str(random.randint(0, 31)) + "," + str(random.randint(0, 31)))

    elif cmdnum == 12:
        cmd = 'or'
        strlist.append(cmd + '$' + str(random.randint(0, 30)) + "," + str(random.randint(0, 31)) + "," + str(random.randint(0, 31)))

    elif cmdnum == 13:
        cmd = 'slt'
        strlist.append(cmd + '$' + str(random.randint(0, 30)) + "," + str(random.randint(0, 31)) + "," + str(random.randint(0, 31)))

    elif cmdnum == 14:
        cmd = 'sltu'
        strlist.append(cmd + '$' + str(random.randint(0, 30)) + "," + str(random.randint(0, 31)) + "," + str(random.randint(0, 31)))

    elif cmdnum == 15:
        if lastisMD == True:
            continue
        cmd = 'mult'
        strlist.append(cmd + '$' + str(random.randint(0, 31)) + "," + str(random.randint(0, 31)))
        lastisMD = True
    elif cmdnum == 16:
        if lastisMD == True:
            continue
        cmd = 'multu'
        strlist.append(cmd + '$' + str(random.randint(0, 31)) + "," + str(random.randint(0, 31)))
        lastisMD = True
    elif cmdnum == 17:
        if lastisMD == True:
            continue
        cmd = 'div'
        a = str(random.randint(1, 31))
        strlist.append('ori $' + a + "," + str(random.randint(1, 65535)))
        strlist.append(cmd + '$' + str(random.randint(0, 31)) + "," + a)
        lastisMD = True
    elif cmdnum == 18:
        if lastisMD == True:
            continue
        cmd = 'divu'
        a = str(random.randint(1, 31))
        strlist.append('ori $' + a + "," + str(random.randint(1, 65535)))
        strlist.append(cmd + '$' + str(random.randint(0, 31)) + "," + a)
        lastisMD = True
    # elif cmdnum == 19:
    #     if lastisjump:
    #         strlist.append('nop')
    #     cmd = "beq"
    #     label = 'L' + str(curlabel)

```

```

#     curlabel += 1
#     #pos = str(random.randint(0, len(strlist)))
#     #strlist.insert(int(pos), label + ':')
#     a =random.randint(0, 30)
#     b =random.randint(0, 31)
#     strlist.append('add $'+ str(a) + ",$" + str(a) + ",$1")
#     strlist.append(cmd +' $'+ str(a) + ",$" + str(b) + "," + label)
#     lastisjump = True
# elif cmdnum == 20:
#     cmd = 'bne'
#     label = 'L' + str(curlabel)
#     curlabel += 1
#     #pos = str(random.randint(0, len(strlist)))
#     #strlist.insert(int(pos), label + ':')
#     a =random.randint(0, 30)
#     b =random.randint(0, 31)
#     strlist.append('add $'+ str(a) + ",$" + str(a) + ",$1")
#     strlist.append(cmd +' $'+ str(a) + ",$" + str(b) + "," + label)
#     lastisjump = True
elif cmdnum == 21:
    cmd = 'nop'
    strlist.append(cmd)

elif cmdnum == 22:
    cmd = 'ori'
    strlist.append(cmd +' $'+ str(random.randint(0, 30)) + ",$" + str(random.randint(0, 31)) + "," + str(random.ra

# elif cmdnum == 23:
#     cmd = 'jal'
#     if lastisjump:
#         strlist.append('nop')
#     label = 'L' + str(curlabel)
#     curlabel += 1
#     #pos = str(random.randint(0, len(strlist)))
#     #strlist.insert(int(pos), label + ':')
#     strlist.append(cmd + ' ' + label)
#     lastisjump = True
# elif cmdnum == 24:
#     if lastisjump:
#         strlist.append('nop')
#     a = random.randint(0, 31)
#     cmd = 'jr'
#     strlist.append(cmd +' $'+ str(a))
#     lastisjump = True
elif cmdnum == 25:
    amd = 'mtlo'
    strlist.append(amd +' $'+ str(random.randint(0, 31)))

elif cmdnum == 26:
    amd = 'mthi'
    strlist.append(amd +' $'+ str(random.randint(0, 31)))

elif cmdnum == 27:
    amd = 'mflo'
    strlist.append(amd +' $'+ str(random.randint(0, 31)))

```

```

    elif cmdnum == 28:
        amd = 'mfhi'
        strlist.append(amd + '$'+ str(random.randint(0, 31)))

for i in range(0, curlabel):
    pos = str(random.randint(0, len(strlist)))
    strlist.insert(int(pos), 'L' + str(i) + ':')
if lastisjump:
    strlist.append('nop')
return strlist

def generate(count):
    file = open('randommips.txt', 'w')
    strl = gencode(count)
    for s in strl:
        file.write(s + '\n')
    file.close()
    exportHex('randommips.txt')

def exportHex(src):
    subprocess.run("java -jar mars.jar db nc mc CompactDataAtZero dump .text HexText code.txt 10000 " + src ,shell=True)
    file = open('./code.txt')
    return file.read()

def simulate(src):
    ret = subprocess.run("java -jar mars.jar db nc mc CompactLargeText ig col1 " + src ,shell=True,stdout=subprocess.PIPE)
    strlist = ret.stdout.splitlines(False)
    for s in strlist:
        if s != b'':
            info.append(s.decode("utf-8"))

def genandsim(count):
    generate(count)
    simulate('randommips.txt')

for j in range(200):

    info = []
    genandsim(700)
    res = subprocess.run('vvp mips.vvp',stdout=subprocess.PIPE,shell=True)
    res = str(res.stdout).replace('\\n','\n').replace('\\t','\t').replace('\\r','\r')

    liswithtime = re.findall(r'[0-9]+@[0-9a-z]{8}: [\$*][0-9 a-zA-Z]+ <= [0-9a-zA-Z]{8}', res)
    reslist = []
    timelist = []

    for line in liswithtime:
        line = line.split('@')
        timelist.append(line[0])
        reslist.append('@'+line[1])

    i=0
    flag = 0

```

```

print('step:{0}'.format(j))
while i < len(reslist) and i < len(info):
    if (reslist[i] != info[i]):
        if (i < len(reslist) - 1 and i < len(info) - 1):
            if (timelist[i] == timelist[i+1]):
                if (info[i+1] == reslist[i]):
                    i = i + 1
                    continue
            if (i > 0):
                if (timelist[i] == timelist[i-1]):
                    if (info[i-1] == reslist[i]):
                        i = i + 1
                        continue
        print('miss match at line {0} : we expect {1} but got {2}'.format(i, info[i], reslist[i]))
        flag = 1
        i = i + 1

    if flag:
        print("Wrong answer at point {0}".format(j))
        break
    else:
        if len(reslist) != len(info):
            print('Your answer is accepted within your output! ')
        else:
            print('Accepted! ')

```

三、思考题

- 乘除法运算复杂度较高，占用较多资源，单独设计有助于提高性能和模块化。ALU 处理加减法逻辑较为简单，整合乘除法会增加延迟和硬件复杂性。独立的 HI/LO 寄存器有助于明确操作结果的存储，提高操作和数据管理的清晰性。
- 通常使用独立的硬件乘法器和除法器单元，与主流水线并行工作。乘除法可能需要多个时钟周期，常通过流水线或分时复用实现高效计算。使用乘除法指令时，流水线可能暂停或设置忙信号，避免数据错误。
- 如果ID阶段存在需要乘除法结果的指令，但busy信号存在则阻塞D级，等待乘除法结果。如果不存在执行的指令则直接并行执行。通过状态机管理乘除法单元，确保多周期操作与流水线的同步。
- 清晰性：通过字节使能信号明确地指定写入范围，避免不必要的数据写入。
统一性：适应不同指令和数据宽度（如字节、半字、字）的存储需求。
高效性：避免无关数据被覆盖，降低错误风险。
- 实际上从 DM 获得或写入的数据可能是一个字（32 位），但通过字节使能信号精确控制某个字节的读写。
读写字时效率更高。
- 将功能分解成独立模块（如控制单元、ALU、寄存器文件、存储器等），以减少单个模块的复杂性。
使用流水线寄存器分割各阶段，确保数据传递的规范性和时序同步。
为每条指令定义明确的控制信号及操作时序。
抽象使译码阶段逻辑更加清晰，规范化控制信号减少设计错误。
处理数据冲突时易于插入冒险检测单元，并用流水线暂停或转发解决问题。
- 除了P5涉及的冒险外，本次新加入的乘除法也有冒险，如D级指令需要乘除法运算结果；E级指令为乘除，D级指令修改hi, lo寄存器的情况。
解决方法：D级指令需要乘除法运算结果时阻塞流水线，D级为无关乘除结果指令时并行执行，D级指令修改hi, lo寄存器时阻塞至乘除法结束。
测试样例：

```
#1
ori $1, $0, 0x1234
ori $2, $0, 0x4567
mult $1, $2
ori $3, $0, 0x1234
ori $4, $0, 0x4567
ori $5, $0, 0x1234
ori $6, $0, 0x4567
ori $7, $0, 0x1234
ori $8, $0, 0x4567
```

```
#2
ori $1, $0, 0x1234
ori $2, $0, 0x4567
mult $1, $2
add $3, $1, $2
```

```
#3
ori $1, $0, 0x1234
ori $2, $0, 0x4567
mult $1, $2
mfhi $3
mflo $4
```

```
#4
ori $1, $0, 0x1234
ori $2, $0, 0x4567
mult $1, $2
mthi $2
add $3, $1, $2
mfhi $3
mflo $4
```

8. 采用计算类指令随机生成，跳转类指令手动构造的方式。由于随机性太高，可能需要非常多的测试数据才能命中cpu缺陷。手动构造主要对jal jr beq bne 与 乘除法相关操作的指令进行测试。