计算机组成原理实验报告

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，支持的指令集包含：

MIPS-C3={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}

（二）关键模块定义

1.CU

CU 根据当前指令判断当前 CPU 需要进行什么操作，根据不同指令的需求通过输出信号控制 CPU 各部分的运行。同时在 CU 里得到读取、写入的寄存器地址，便于后面处理。端口定义如表 1 所示。

表 1 CU 端口描述

端口名称	方向	功能描述
S[31:0]	I	输入的指令
Stage[1:0]	I	当前在哪个阶段
NPCOp[2:0]	O	控制 NPC 的操作
GRF_D_Op[1:0]	O	选择 GRF 的写入数据
EXTSign	O	EXT 的扩展方式
ALUOp[2:0]	O	ALU 的计算类型
ALU_B_Op	O	ALU 的第二个计算数的来源
DMWrite	O	DM 的写入信号
A1[4:0]	O	需要读取的第一个寄存器

A2[4:0]	O	需要读取的第二个寄存器
A3[4:0]	O	需要写入的寄存器
Tnew[1:0]	O	见（三）3.
Tuse1[1:0]	O	A1 的 Tuse，见（三）3.
Tuse2[1:0]	O	A2 的 Tuse，见（三）3.

2. ALU

ALU 提供 32 位所有运算功能。其端口设置如表 2 所示。

表 2 ALU 端口描述

端口名称	方向	功能描述
Op[3:0]	I	代表需要进行的运算操作
A[31:0]	I	进行运算的第一个数
B[31:0]	I	进行运算的第二个数
C[31:0]	O	运算结果

3.MDU

MDU 负责实现乘除操作，内置了 HI、LO 两个寄存器。其端口设置如表 3 所示。

表 3 MDU 端口描述

端口名称	方向	功能描述
clk	I	时钟信号
reset	I	同步复位信号
Start	I	乘除运算的开始信号
Op[3:0]	I	代表需要进行的运算或存取操作
A[31:0]	I	进行运算的第一个数
B[31:0]	I	进行运算的第二个数
C[31:0]	O	运算结果
Busy	O	当前是否正在进行运算的信号

4. GRF

GRF 负责读写 32 个寄存器，编号为 0~31，其中 0 号寄存器的值始终保持为 0，其它寄存器的初始值为 0。GRF 可以随时读出 2 个不同寄存器的内容，并在信号的控制下在时钟上升沿将指定内容写入指定寄存器，具有同步复位的功能。其端口设置如表 4 所示。这里把写使能信号优化了，在不写入的情况下保证 A3=0。

表 4 GRF 端口描述

端口名称	方向	功能描述
clk	I	时钟信号
reset	I	同步复位信号
A1[4:0]	I	指定 32 个寄存器中的一个，将其中存储的数据读出到 DO1
A2[4:0]	I	指定 32 个寄存器中的一个，将其中存储的数据读出到 DO2
A3[4:0]	I	指定 32 个寄存器中的一个，作为写入的目标寄存器
DI[31:0]	I	需要写入 A3 指定的寄存器的数据
DO1[31:0]	O	32 位数据输出信号，表示 A1 指定的寄存器中的数据
DO2[31:0]	O	32 位数据输出信号，表示 A2 指定的寄存器中的数据

5. PC

PC 为寄存器，存储当前指令地址，具有同步复位功能，起始地址为 0x00003000。其端口设置如表 5 所示。

表 5 PC 端口描述

端口名称	方向	功能描述
clk	I	时钟信号
reset	I	同步复位信号
PCI[31:0]	I	下一步的指令地址
PCO[31:0]	I	当前输出的指令地址

6.NPC

NPC 根据当前指令的内容计算下一条指令的地址。其端口设置如表 6 所示。

表 6 PC 端口描述

端口名称	方向	功能描述
NPCOp[2:0]	I	控制信号
eq	I	ALU 计算结果是否为 0。用于判断 beq。
PC[31:0]	I	当前的指令地址（F 级）
Imm[25:0]	I	（可能）需要跳转的立即数
Ra[31:0]	I	（可能）需要跳转的寄存器值
NPC[31:0]	O	下一步指令地址

7.EXT

分别实现立即数的 0 扩展以及符号扩展。随后输出作为 ALU 的运算数。其端口设置如表 7 所示。

表 7 PC 端口描述

端口名称	方向	功能描述
EXTSign	I	扩展方式控制信号
DI[15:0]	I	需要进行扩展的数
DO[31:0]	O	扩展后的数

（三）重要机制实现方法

1. 跳转

CPU 支持 beq、jal、jr、j 跳转指令。

本质上就是在 NPC 模块中加入一个控制信号 NPCOp，然后判断下一步的地址。

对于 beq，需要在 ALU 中判断两个寄存器的数值是否相等，若相等则将立即数进行符号扩展、左移的处理，作为偏移量加到 PC 值上。

对于 jal、j 和 jr，就可以直接赋值成对应的地址。

2. 流水线延迟槽

当前指令为 beq、j、jal 或 jr 时，会产生跳转，而此时 F 级已经读出新的指令，这条指令也会被执行，到下一个周期才会跳转到指定地址。这就是延迟

槽。由于延迟槽，jal 需要写入 PC+8 至$ra。

为了增大效率和保持统一，把 beq 的比较前置到 D 级，在 GRF 读出数之后就进行比较。

在 NPC 计算时，我使用 F 级的 PC，D 级的控制信号进行计算。

3. 转发和暂停

首先明确转发数据的供给者和接收者。

供给者有：D/E 级寄存器（PC8 数据），E/M 级寄存器（PC8 或 ALU 计算结果），M/W 级寄存器（PC8 或 ALU 计算结果或 DM 读出数据）。

接收者有：D 级 GRF 的输出（CMP 的输入）、E 级 ALU 的输入，M 级 DM 的输入。

在整体架构中明确两部分，第一部分是需要用来计算、存储的数据，是接收者；第二部分是已经得到，将要写入寄存器的数据，是供给者。

先假设没有暂停的情况。我们采用暴力转发的方案。即：每个接收者会接收到所有在它之前得到的数据。判断时只要需要使用的数据来源地址和某个将要写入的地址相同即可转发。多个满足条件的取最新的数据。注意 0 号寄存器不被转发，也不接收。具体可以见思考题（三）。

如果需要暂停，也就是某个地址的数据还没有被得到，但是马上就要在下一轮被用到。这里用 AT 法判断是否暂停。每个指令对应着两个读取的寄存器的 Tuse。Tuse 指从 D 级开始，再过多少周期后，数据一定要被转发到当前阶段，才能进行下去。例如，addu 的两个 Tuse 均为 1，sw 的一个 Tuse 为 1，一个 Tuse 为 2。

同时还有 Tnew。Tnew 表示在当前阶段，再过多少个周期后，才能产生数据并传到下一级寄存器中。例如，lw 在 D 级的 Tnew 为 3，E 级为 2。注意 Tuse 和 Tnew 的意义都是建立在确定的寄存器地址上的，比较的时候也要比较相同地址的情况。

我们先把所有指令拦在在 D 级，然后进行比较。如果两个需要的寄存器都满足 Tuse>=Tnew，那么就说明可以通过转发解决。否则，就必须在这里停顿，让前面的指令再流一个周期。这就是暂停机制。暂停时，PC 和 F/D 级寄存器不变，D/E 级寄存器清空，后面的照样流下去。

暂停机制确保了暴力转发的正确性。因为暴力转发的时候没有判断一个数据是否已经产生。在暂停机制下，如果数据没有产生就会被迫暂停，所以暴力转发是对的。

二、测试方案

（一）典型测试样例

主要测试转发和暂停的正确性。

```
.text
ori $1,$0,10
sw $1,0($0)
ori $1,$0,20
sw $1,4($0)
ori $1,$0,30
sw $1,8($0)
ori $1,$0,40
sw $1,12($0)
ori $1,$0,50
sw $1,16($0)
###Calc_r
lw $t2,0($0)
addu $t2,$t2,$t2
###Calc_i
lw $t2,0($0)
ori $t2,$t2,100
###beq
ori $s0,$0,1
addu $s1,$s0,$0
beq $s0,$s1,f1
nop
ori $1,$0,1
f1:
nop
#
ori $s0,$0,1
ori $s1,$s0,2
beq $s1,$s0,f2
nop
ori $1,$0,2
f2:
nop
```

```
#
ori $s0,$0,0
ori $s0,$0,10
lw $s1,0($0)
beq $s1,$s0,f3
nop
ori $1,$0,3
f3:
nop
#
ori $s0,$0,0
ori $s0,$0,10
lw $s1,0($0)
nop
beq $s1,$s0,f4
nop
ori $1,$0,4
f4:
addu $t0,$t0,$t0
###load
lw $t2,0($0)
lw $t3,0($t2)
###store
lw $t2,0($0)
sw $t3,0($t2)
###jr
ori $t3,$0,0x000030c8
addu $t2,$t2,$t3
jr $t2
nop
ori $1,$0,5
nop
#
ori $t3,$0,0x000030e0
ori $t2,$t3,0
jr $t2
nop
ori $1,$0,6
nop
#
ori $t3,$0,0x000030fc
sw $t3,0($0)
lw $t2,0($0)
jr $t2
```

```
nop
ori $1,$0,7
nop
#
ori $t3,$0,0x0000311c
sw $t3,0($0)
lw $t2,0($0)
nop
jr $t2
nop
ori $1,$0,8
nop
###
lw     $s1,0($a0)
addi   $a0,$a0,4
lw     $s2,0($a0)
addi   $a0,$a0,4
mult   $s1,$s2
mflo   $31
mfhi   $31
mult   $s2,$s1
mflo   $31
mfhi   $31
mult   $s1,$s1
mflo   $31
mfhi   $31
mult   $s2,$s2
mflo   $31
mfhi   $31
multu  $s1,$s2
mflo   $31
mfhi   $31
multu  $s2,$s1
mflo   $31
mfhi   $31
multu  $s1,$s1
mflo   $31
mfhi   $31
multu  $s2,$s2
mflo   $31
mfhi   $31
beq    $s1,$0,F1
addi   $31,$31,1
addi   $31,$31,1
```

```
div    $s2,$s1
mflo   $31
mfhi   $31
div    $s1,$s1
mflo   $31
mfhi   $31
divu   $s2,$s1
mflo   $31
mfhi   $31
divu   $s1,$s1
mflo   $31
mfhi   $31
F1:
beq    $s2,$0,F2
addi   $31,$31,1
addi   $31,$31,1
div    $s1,$s2
mflo   $31
mfhi   $31
div    $s2,$s2
mflo   $31
mfhi   $31
divu   $s1,$s2
mflo   $31
mfhi   $31
divu   $s2,$s2
mflo   $31
mfhi   $31
F2:
end:
j end
nop
```

（二）自动测试工具

1. 测试样例生成器

无

2. 自动执行脚本

用 python 自动运行仿真程序并比较结果。

```python
import os

vfiles_folder="vfiles"
testcase_folder="testcases\\7"
tb_filepath="test"
xilinx_path='C:\\Xilinx\\14.7\\ISE_DS\\ISE'


with open('mips.tcl','w') as f:
    f.write("run 200us;\nexit")

V=[]
for dirpath,dirnames,filenames in os.walk(vfiles_folder):
    for file in filenames:
        file_type=file.split('.')[-1]
        if file_type=="v":
            V.append(os.path.join(dirpath,file))

with open('mips.prj','w') as f:
    for i in range(len(V)):
        f.write('verilog work "'+V[i]+'"\n')

os.system("g++ check.cpp -o check.exe")
os.environ['XILINX'] = xilinx_path
os.system(xilinx_path + "\\bin\\nt64\\fuse -nodebug -prj mips.prj -o mips.exe "+tb_filepath+" >log.txt")


try:
    for dirpath,dirnames,filenames in os.walk(testcase_folder):
        for file in filenames:
            file_type=file.split('.')[-1]
            if file_type=="asm":
                mipscodedir=os.path.join(dirpath,file)
                os.system("java -jar Mars.jar "+mipscodedir+" nc mc CompactDataAtZero a dump .text HexText code.txt")
                os.system("check.exe < code.txt > out1.txt")
                os.system("mips.exe -nolog -tclbatch mips.tcl > out2.txt")

                A=[]
                B=[]
                with open('out1.txt','r') as f:
                    for s in f.readlines():
                        A.append(s.strip())

                with open('out2.txt','r') as f:
                    for s in f.readlines():
                        if s.find('@')==-1:
                            continue
                        s1=s.strip().split('@')
                        B.append('@'+s1[1])

                with open('out2.txt','w') as f:
                    for i in range(0,len(B)):
                        f.write(B[i]+"\n")

                h=min(min(len(A),len(B)),5000)
                for i in range(h):
                    if A[i]!=B[i]:
                        raise Exception(mipscodedir+" WA!\nExpected:"+A[i]+"\nGot      :"+B[i])

                if (len(A)!=len(B) and min(len(A),len(B))<5000):
                    raise Exception(mipscodedir+" WA!\nOutput length not match!")

                print(mipscodedir+" Accepted!")


except Exception as e:
    print(e)
else:
    print("Success!")
    pass
```

3. 正确性判定脚本

使用 cpp 模拟 cpu 的行为，和 ise 的结果对比。比较的时候去除了时间和版权信息，然后逐字符比较。见 2.

```cpp
#include<bits/stdc++.h>
using namespace std;
#define delay
unsigned int S,Ins[4096],Mem[4096],Reg[32],PC=0x3000,rs,rt,rd,Imm16,Imm26,Imm16s,HI,LO;
unsigned int Get(unsigned int x,int l,int r){
    return (x&((((1u<<r)-1)<<1)|1))>>l;
}
void RegWrite(unsigned int PC,unsigned int tar,unsigned int val){
    if (tar==0){
        return;
    }
    printf("@%08x: $%2d <= %08x\n",PC,tar,val);
    Reg[tar]=val;
}
void MemWrite(unsigned int PC,unsigned int tar,unsigned int val){
    printf("@%08x: *%08x <= %08x\n",PC,tar<<2,val);
    Mem[tar]=val;
}
long long Signed(unsigned int x){
    if (x>>31){
        return -(long long)((~x)+1);
    }else{
        return x;
    }
}
unsigned Sign_ext(unsigned int x,int i){
    if (x>>(i-1)){
        x|=0u-(1u<<i);
    }
    return x;
}
void exe(unsigned int &PC){
    S=Ins[(PC-0x3000)>>2];
    unsigned int Opcode=Get(S,26,31),Funct=Get(S,0,5),sa=Get(S,6,10);
    rs=Get(S,21,25);
    rt=Get(S,16,20);
    rd=Get(S,11,15);
    Imm16=Get(S,0,15);
    Imm16s=Sign_ext(Imm16,16);
    Imm26=Get(S,0,25);
    unsigned int p=Reg[rs]+Imm16s;

    unsigned int PC1=PC+4;
    if (Opcode==0b000000 && Funct==0b100000){//ADD
        RegWrite(PC,rd,Reg[rs]+Reg[rt]);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b100001){//ADDU
        RegWrite(PC,rd,Reg[rs]+Reg[rt]);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b100010){//SUB
        RegWrite(PC,rd,Reg[rs]-Reg[rt]);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b100011){//SUBU
        RegWrite(PC,rd,Reg[rs]-Reg[rt]);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b100100){//AND
        RegWrite(PC,rd,Reg[rs]&Reg[rt]);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b100101){//OR
        RegWrite(PC,rd,Reg[rs]|Reg[rt]);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b100110){//XOR
        RegWrite(PC,rd,Reg[rs]^Reg[rt]);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b100111){//NOR
        RegWrite(PC,rd,~(Reg[rs]|Reg[rt]));
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b101010){//SLT
        RegWrite(PC,rd,(unsigned int)((int)Reg[rs]<(int)Reg[rt]));
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b101011){//SLTU
        RegWrite(PC,rd,(unsigned int)(Reg[rs]<Reg[rt]));
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b000100){//SLLV
        RegWrite(PC,rd,Reg[rt]<<Get(Reg[rs],0,4));
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b000110){//SRLV
        RegWrite(PC,rd,Reg[rt]>>Get(Reg[rs],0,4));
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b000111){//SRAV
```

```
        RegWrite(PC,rd,(unsigned int)((int)Reg[rt]>>(int)Get(Reg[rs],0,4)));
        PC+=4;


    }else if (Opcode==0b001000){//ADDI
        RegWrite(PC,rt,Reg[rs]+Imm16s);
        PC+=4;
    }else if (Opcode==0b001001){//ADDIU
        RegWrite(PC,rt,Reg[rs]+Imm16s);
        PC+=4;
    }else if (Opcode==0b001100){//ANDI
        RegWrite(PC,rt,Reg[rs]&Imm16);
        PC+=4;
    }else if (Opcode==0b001101){//ORI
        RegWrite(PC,rt,Reg[rs]|Imm16);
        PC+=4;
    }else if (Opcode==0b001110){//XORI
        RegWrite(PC,rt,Reg[rs]^Imm16);
        PC+=4;
    }else if (Opcode==0b001111){//LUI
        RegWrite(PC,rt,Imm16<<16);
        PC+=4;
    }else if (Opcode==0b001010){//SLTI
        RegWrite(PC,rt,(unsigned int)((int)Reg[rs]<(int)Imm16s));
        PC+=4;
    }else if (Opcode==0b001011){//SLTIU
        RegWrite(PC,rt,(unsigned int)(Reg[rs]<Imm16s));
        PC+=4;


    }else if (Opcode==0b000000 && Funct==0b000000){//SLL
        RegWrite(PC,rd,Reg[rt]<<sa);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b000010){//SRL
        RegWrite(PC,rd,Reg[rt]>>sa);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b000011){//SRA
        RegWrite(PC,rd,(unsigned int)((int)Reg[rt]>>(int)sa));
        PC+=4;


    }else if (Opcode==0b000100){//BEQ
        PC+=4;
        if (Reg[rs]==Reg[rt]){
            PC+=Imm16s<<2;
            #ifdef delay
                exe(PC1);
            #endif
        }
    }else if (Opcode==0b000101){//BNE
        PC+=4;
        if (Reg[rs]!=Reg[rt]){
            PC+=Imm16s<<2;
            #ifdef delay
                exe(PC1);
            #endif
        }
    }else if (Opcode==0b000001 && Get(S,16,20)==0b00001){//BGEZ
        PC+=4;
        if (Signed(Reg[rs])>=0){
            PC+=Imm16s<<2;
            #ifdef delay
                exe(PC1);
            #endif
        }
    }else if (Opcode==0b000111){//BGTZ
        PC+=4;
        if (Signed(Reg[rs])>0){
            PC+=Imm16s<<2;
            #ifdef delay
                exe(PC1);
            #endif
        }
    }else if (Opcode==0b000110){//BLEZ
        PC+=4;
        if (Signed(Reg[rs])<=0){
            PC+=Imm16s<<2;
            #ifdef delay
                exe(PC1);
            #endif
```

```
        }
    }else if (Opcode==0b000001 && Get(S,16,20)==0b00000){//BLTZ
        PC+=4;
        if (Signed(Reg[rs])<0){
            PC+=Imm16s<<2;
            #ifdef delay
                exe(PC1);
            #endif
        }


    }else if (Opcode==0b000010){//J
        PC=(PC&(0u-(1u<<28)))|(Imm26<<2);
        #ifdef delay
            exe(PC1);
        #endif
    }else if (Opcode==0b000011){//JAL
        RegWrite(PC,31u,PC+8);
        PC=(PC&(0u-(1u<<28)))|(Imm26<<2);
        #ifdef delay
            exe(PC1);
        #endif
    }else if (Opcode==0b000000 && Funct==0b001000){//JR
        PC=Reg[rs];
        #ifdef delay
            exe(PC1);
        #endif
    }else if (Opcode==0b000000 && Funct==0b001001){//JALR
        RegWrite(PC,rd,PC+8);
        PC=Reg[rs];
        #ifdef delay
            exe(PC1);
        #endif


    }else if (Opcode==0b100000){//LB
        RegWrite(PC,rt,Sign_ext(Get(Mem[p>>2],(p&3)*8,(p&3)*8+7),8));
        PC+=4;
    }else if (Opcode==0b100100){//LBU
        RegWrite(PC,rt,Get(Mem[p>>2],(p&3)*8,(p&3)*8+7));
        PC+=4;
    }else if (Opcode==0b100001){//LH
        RegWrite(PC,rt,Sign_ext(Get(Mem[p>>2],(p&3)*8,(p&3)*8+15),16));
        PC+=4;
    }else if (Opcode==0b100101){//LHU
        RegWrite(PC,rt,Get(Mem[p>>2],(p&3)*8,(p&3)*8+15));
        PC+=4;
    }else if (Opcode==0b100011){//LW
        RegWrite(PC,rt,Mem[p>>2]);
        PC+=4;


    }else if (Opcode==0b101000){//SB
        MemWrite(PC,p>>2,
        Mem[p>>2]^((Get(Mem[p>>2],(p&3)*8,(p&3)*8+7)^Get(Reg[rt],0,7))<<((p&3)*8))
        );
        PC+=4;
    }else if (Opcode==0b101001){//SH
        MemWrite(PC,p>>2,
        Mem[p>>2]^((Get(Mem[p>>2],(p&3)*8,(p&3)*8+15)^Get(Reg[rt],0,15))<<((p&3)*8))
        );
        PC+=4;
    }else if (Opcode==0b101011){//SW
        MemWrite(PC,p>>2,Reg[rt]);
        PC+=4;

    }else if (Opcode==0b000000 && Funct==0b011000){//MULT
        unsigned long long res=(long long)(int)Reg[rs]*(int)Reg[rt];
        HI=res>>32;
        LO=res&(-1u);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b011001){//MULTU
        unsigned long long res=(unsigned long long)Reg[rs]*Reg[rt];
        HI=res>>32;
        LO=res&(-1u);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b011010){//DIV
        HI=(long long)(int)Reg[rs]%(int)Reg[rt];
        LO=(long long)(int)Reg[rs]/(int)Reg[rt];
        PC+=4;
```

```
    }else if (Opcode==0b000000 && Funct==0b011011){//DIVU
        HI=Reg[rs]%Reg[rt];
        LO=Reg[rs]/Reg[rt];
        PC+=4;

    }else if (Opcode==0b000000 && Funct==0b010000){//MFHI
        RegWrite(PC,rd,HI);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b010010){//MFLO
        RegWrite(PC,rd,LO);
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b010001){//MTHI
        HI=Reg[rs];
        PC+=4;
    }else if (Opcode==0b000000 && Funct==0b010011){//MTLO
        LO=Reg[rs];
        PC+=4;
    }


}
int main(){
    long T0=clock();
    int h=0;
    while (~scanf("%x",&S)){
        Ins[h++]=S;
    }
    while ((PC-0x3000)>>2<=(unsigned int)h){
        if (1.0*(clock()-T0)/CLOCKS_PER_SEC>0.05) break;
        exe(PC);
    }
    return 0;
}
```

三、思考题

（一）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？

答：因为乘除法不能在一个周期内得到结果，而且结果存储在 HI/LO 寄存器中，和普通的运算指令有本质区别。而且如果整合进 ALU，所有的运算指令都会被阻塞，效率低下。单独设计乘除法部件后，在乘除法的同时也可以执行非乘除的指令。整合也会使得 ALU 端口增加，不符合高内聚低耦合的原则。

乘除法会产生两个寄存器的内容，如果不用独立的 HI、LO 寄存器，而是放到 GRF 内，那么就要同时写入 2 个寄存器，在当前的基础上需要很大改动才能实现。而且在 MIPS 架构下，HI、LO 两个寄存器相对于其他 32 个寄存器来说是独立的，不会直接用于计算，因此可以直接独立出来放到乘除部件里。

（二）参照你对延迟槽的理解，试解释 “乘除槽”。

答：乘除的延迟是模拟了真实情况下乘除法需要的周期数，在乘除法正在进行的时候，如果 D 级是存取 HI、LO 或乘除的指令，就被阻塞，直到正在进行的乘除法结束。而如果 D 级不是乘除法，那么可以继续进行下去，没有影响。

（三）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）

答：假设有一个字符串 ABCD，每个字符占一个字节，四个字符才占了一个字。如果只需要取出某个字符，按字节访问就可以直接取出，而按字访问则还需要从取出的字里挑出需要的字符。

（四）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？

答：除了 P5 中的情况，还有乘除法所产生的冲突：MFHI/MFLO/MTHI/MTLO 和四个乘除法指令之间的冲突。

D级			E级			M级
指令类型	寄存器	Tuse	Tnew			
			calc_r:rd/1	calc_i:rt/1	load:rt/2	load:rt/2
calc_r	rs/rt	1			Stall	
calc_I	rs	1			Stall	
beq	rs/rt	0	Stall	Stall	Stall	Stall
load	rs	1			Stall	
store	rs	1			Stall	
	rt	2				
jr	rs	0	Stall	Stall	Stall	Stall

（五）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？

答：把所有指令分类处理，相同特性的归为一类，它们的 instr 格式以及 Tnew、Tuse 都相同，在产生控制信号的时候比较直观、方便，在数据冲突的时候也可以比较方便地判断。