计算机组成原理实验报告

一、CPU 设计方案综述

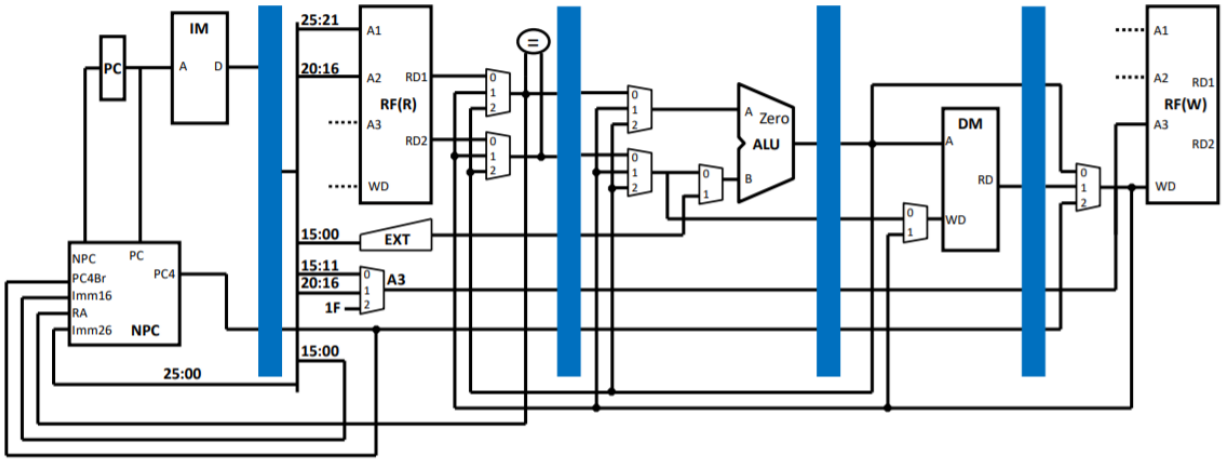
（一）总体设计概述

本 CPU 为 Verilog 实现的流水线 MIPS - CPU，支持的指令集包含{addu, subu, ori, lw, sw, beq, lui, jal, jr, nop}。为了实现这些功能，CPU 主要包含了 ALU、CU、DM、EXT、GRF、IM、NPC、PC 等模块。数据通路如下表所示。

表 1 数据通路表

部件	PC	NPC			IM	GRF				EXT	ALU		DM	
输入信号	PCI	PC	Imm	Ra	A	A1	A2	A3	DI	DI	A	B	A	DI
addu	NPC.NPC	PC.PCO			PC.PCO	IM.S[25:21]	IM.S[20:16]	IM.S[15:11]	ALU.C		GRF.DO1	GRF.DO2		
subu	NPC.NPC	PC.PCO			PC.PCO	IM.S[25:21]	IM.S[20:16]	IM.S[15:11]	ALU.C		GRF.DO1	GRF.DO2		
ori	NPC.NPC	PC.PCO			PC.PCO	IM.S[25:21]		IM.S[20:16]	ALU.C	IM.S[15:0]	GRF.DO1	EXT.DO		
lw	NPC.NPC	PC.PCO			PC.PCO	IM.S[25:21]		IM.S[20:16]	DM.DO	IM.S[15:0]	GRF.DO1	EXT.DO	ALU.C	
sw	NPC.NPC	PC.PCO			PC.PCO	IM.S[25:21]	IM.S[20:16]			IM.S[15:0]	GRF.DO1	EXT.DO	ALU.C	GRF.DO2
beq	NPC.NPC	PC.PCO	IM.S[15:0]		PC.PCO	IM.S[25:21]	IM.S[20:16]				GRF.DO1	GRF.DO2		
lui	NPC.NPC	PC.PCO			PC.PCO	IM.S[25:21]		IM.S[20:16]	ALU.C	IM.S[15:0]		EXT.DO		
jal	NPC.NPC	PC.PCO	IM.S[25:0]		PC.PCO			0x1f	NPC.PC4					
jr	NPC.NPC	PC.PCO		GRF.DO1	PC.PCO	IM.S[25:21]								
完整	NPC.NPC	PC.PCO	IM.S[15:0] IM.S[25:0]	GRF.DO1	PC.PCO	IM.S[25:21]	IM.S[20:16]	IM.S[15:11] IM.S[20:16] 0x1f	ALU.C DM.DO NPC.PC4	IM.S[15:0]	GRF.DO1	GRF.DO2 EXT.DO	ALU.C	GRF.DO2

参考的结构示意图如下。



（二）关键模块定义

1. 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 2 所示。（见思考题，这里把写使能信号优化了，在不写入的情况下保证 A3=0）

表 2 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

2. DM

DM 负责在 32bit*3072 大小的空间中读写数据，所有数据初始值为 0，起始地址为 0x00000000。具有同步复位功能。其端口设置如表 3 所示。

表 3 DM 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
DMWrite	I	写使能信号
A[31:0]	I	要读入或输出的数据地址（以字节为单位）
DI[31:0]	I	需要写入的数据
DO[31:0]	O	A 指定的地址的数据（字）

3. ALU

ALU 提供 32 位加、减、或、左移 16 位（lui）运算功能。目前没有支持判断溢出的功能。其端口设置如表 4 所示。

表 4 ALU 端口描述

端口名称	方向	功能描述
ALUOp[2:0]	I	代表需要进行的运算操作 0：不运算，直接输出 A 1：A+B 2：A-B 3：A|B 4：B<<16
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

4.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 5 所示。每个指令对应控制信号的值如表 6 所示。

表 5 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.

表 6 CU 中每个指令对应的控制信号

输出信号	指令								
	addu	subu	ori	lw	sw	beq	lui	jal	jr
NPCOp[2:0]	000	000	000	000	000	001	000	010	011
GRF_D_Op[1:0]	00	00	00	01	XX	XX	00	10	XX
EXTOp	X	X	0	1	1	X	0	X	X
ALUOp[2:0]	001	010	011	001	001	010	100	XXX	XXX
ALU_B_Op	0	0	1	1	1	0	1	X	X
DMWrite	0	0	0	0	1	0	0	0	0

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 7 所示。

表 7 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 8 所示。

表 8 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. IM

IM 为一个 32bit*4096 大小的存储空间，根据当前 PC 读出指令。其端口设置如表 9 所示。

表 9 PC 端口描述

端口名称	方向	功能描述
A[31:0]	I	指令地址
S[31:0]	O	读出的指令

8.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 10 所示。

表 10 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
#
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;

unsigned int S,Ins[4096],Mem[4096],Reg[32],PC=0x3000,rs,rt,rd,Imm16,Imm26,Imm16s;
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,val);
    Mem[tar>>2]=val;
}
void exe(unsigned int &PC){
    S=Ins[(PC-0x3000)>>2];
    unsigned int Opcode=Get(S,26,31),Funct=Get(S,0,5);
    rs=Get(S,21,25);
    rt=Get(S,16,20);
    rd=Get(S,11,15);
    Imm16=Imm16s=Get(S,0,15);
    if (Imm16>>15){
        Imm16s|=0u-(1u<<16);
    }
    Imm26=Get(S,0,25);
    unsigned int PC1=PC+4;
    if (Opcode==0){
        if (Funct==0b100001){//addu
            RegWrite(PC,rd,Reg[rs]+Reg[rt]);
            PC+=4;
        }else if (Funct==0b100011){//subu
            RegWrite(PC,rd,Reg[rs]-Reg[rt]);
            PC+=4;
        }else if (Funct==0b001000){//jr
            PC=Reg[rs];
            exe(PC1);
        }else{//nop
            PC+=4;
        }
    }else if (Opcode==0b001101){//ori
        RegWrite(PC,rt,Reg[rs]|Imm16);
        PC+=4;
    }else if (Opcode==0b100011){//lw
        RegWrite(PC,rt,Mem[(Reg[rs]+Imm16s)>>2]);
        PC+=4;
    }else if (Opcode==0b101011){//sw
        MemWrite(PC,Reg[rs]+Imm16s,Reg[rt]);
        PC+=4;
    }else if (Opcode==0b000100){//beq
        PC+=4;
        if (Reg[rs]==Reg[rt]){
            PC+=Imm16s<<2;
            exe(PC1);
        }
    }else if (Opcode==0b001111){//lui
        RegWrite(PC,rt,Imm16<<16);
        PC+=4;
    }else if (Opcode==0b000011){//jal
        RegWrite(PC,31u,PC+8);
        PC=(PC&(0u-(1u<<28)))|(Imm26<<2);
        exe(PC1);
    }else if (Opcode==0b000010){//j
        PC=(PC&(0u-(1u<<28)))|(Imm26<<2);
        exe(PC1);
    }
}
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>0.01) break;
        exe(PC);
    }
    return 0;
}

```

三、思考题

（一）流水线冒险

1.在采用本节所述的控制冒险处理方式下，PC 的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

答：

```
module npc(
      input [2:0] NPCOp,
      input eq,
      input [31:0] PC,//F_PC
      input [25:0] Imm,
      input [31:0] Ra,
      output [31:0] NPC
   );
   reg [31:0] res;
   assign NPC=res;
   always @(*)
   begin
      case (NPCOp)
         3'b000: res<=PC+32'd4;
         3'b001: res<=PC+(eq?{{14{Imm[15]}},Imm[15:0],2'b0}:32'd4);
         3'b010: res<={PC[31:28],Imm[25:0],2'b0};
         3'b011: res<=Ra;
         default: res<=PC+32'd4;
      endcase
   end
endmodule
```

从 D 级的 CU 获取控制信号和需要跳转的地址，CMP 获取是否相等，从 F 级获取当前 PC 值，然后对不同情况讨论。

NPCOp=0 表示正常，1 为 beq，2 为 jal 或 j，3 为 jr。

2.对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8？

答： 因为考虑了延时槽。

（二）数据冒险的分析

1. 为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由 ALU 或者 DM 等部件来提供数据？

答：因为如果直接使用 ALU 或 DM 转发提供数据，虽然可以减少一个周期，但是因为有两个部件需要依次工作，周期可能需要增加一倍左右，得不偿失。而且会造成数据不稳定的情况。

（三）AT 法处理流水线数据冒险

1. “转发（旁路）机制的构造”中的 Thinking 1-4

Thinking 1：如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题。

答：采用原始数据显然是错误的，会导致没有用到最新的数据。

```
lw $1,A
nop
nop
ori $2,$1,1
```

当 lw 位于 W 级时，ori 位于 D 级，此时可以转发。但如果 E 级不使用转发后的数据，那么就是没有更改的数据，是错误的，而此时 lw 已经完成，不会再进行转发了。

Thinking 2：我们为什么要对 GPR 采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？

答：采用内部转发机制可以将转发放在 GPR 内部，不需要在外面额外处理，比较方便。如果不采用，则需要和其他转发一样，把 M/W 级寄存器里的数据转发到 D 级。我认为后者结构上更加统一、清晰，于是采用了后者。

Thinking 3：为什么 0 号寄存器需要特殊处理？

答：因为 0 号寄存器的值始终为 0，在这一条件下，如果需要使用的是 0 号寄存器的值，可以直接当做 0，就不需要解决冲突，能够提高效率。如果考虑了要写入 0 号寄存器的值反而是错的。

Thinking 4：什么是“最新产生的数据”？

答：也就是离当前指令最近的一个指令产生的数据，显然指令是根据顺序执行的，如果前面有多个指令都产生了同一个寄存器的数据，那么最近的一个指令产生的才是我们想要的。

2. 在 AT 方法讨论转发条件的时候，只提到了“供给者需求者的 A 相同，且不为 0”，但在 CPU 写入 GRF 的时候，是有一个 we 信号来控制是否要写入的。为何在 AT 方法中不需要特判 we 呢？为了用且仅用 A 和 T 完成转发，在翻译出 A 的时候，要结合 we 做什么操作呢？

答：可以令不写入寄存器的时候 A 为 0，这样不用判断 we 就可以知道不

能转发，并且使逻辑更加简单。

（四）在线测试相关说明。

1. 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：可能产生阻塞的情况如下图所示。测试样例见二。

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

构造样例时，先把每种情况造出来，然后进行拼接，保证覆盖了所有暂停情况。