计算机组成课程设计 P6 实验报告

一、 数据通路设计

1、 F 级组合逻辑

（1）功能多路选择器 MUX_PC：根据当前指令的操作码，从 RFRD1、NPC、ADD4 中选择下一条指令的 PC 值

（2）PC 模块根据根据是否暂停，决定是否更新 PC

（3）IF 模块中包含 im 指令寄存器、可以根据 PC 值取出当前指令

PC 端口定义：

信号名	方向	描述
PC0[31：0]	I	下一条指令的 PC 值
stall_mult	I	因为计算而造成的暂停
en	I	使能信号，因为数据冲突而造成的暂停
clk	I	时钟信号
reset	I	复位信号
PC[31:0]	O	当前 PC 值

PC 功能定义：

序号	功能名称	功能描述
1	复位	当时钟上升沿到来时，如果复位信号有效，PC 被复位为 0x00003000
2	暂停	当 en 信号为 0 或 stall_mult 信号为 1 时，将 D 级信号暂停不变
3	输出下一个 PC 值	当时钟上升沿到来时，将 PC0 的值赋给 PC

IF 端口定义：

信号名	方向	描述
PC[31：0]	I	当前指令的 PC 值
IR[31:0]	O	当前指令的操作码
PC4[31:0]	O	当前 PC 加 4

IF 功能定义：

序号	功能名称	功能描述
1	读入指令	将 code.txt 中的指令读入到 im 寄存器中
2	输出指令操作码	根据当前 PC 值，从 im 寄存器中取出当前操作码
3	输出 PC4 的值	根据当前 PC 值，输出 PC+4 的值

2. ID 级组合逻辑

(1)其中包含 MFCMP1D、MFCMP2D 两个转发多路选择器，控制进入比较器的数据为正确的当前数据

（2）ID 模块中有 RF、EXT、CMP、NPC。RF 寄存器在 D 级为组合逻辑，其中包含 32 个临时寄存器 rege，根据当前指令码，输出 rs 和 rt 寄存器的值。在回写时，时钟上升沿到来时，如果写使能信号有效，将 WDdata 写入 A3 寄存器；CMP 为 b 指令的前移比较器，当两输入信号符合要求时，输出 1；EXT 单元进行立即数的扩展；NPC 单元根据当前 npcsel 信号和 PC 值,计算出跳转类指令的 NPC。

ID 端口定义：

信号名	方向	描述
IR_D[31:0]	I	D 级指令的操作码
PC4_W[4:0]	I	W 级的 PC4,用于输出回写地址
PC4_D[4:0]	I	D 级的 PC4，用于 NPC 单元计算下一级的 PC
A3_W[4:0]	I	回写的寄存器编号
MFCMP1D[31:0]	I	CMP 模块中第一个输入的转发结果
MFCMP2D[31:0]	I	CMP 模块中第二个输入的转发结果
RegWrite_W	I	写使能信号，用于回写单元
WD[31:0]	I	回写寄存器的值
clk	I	时钟信号
reset	I	复位信号
RFRD1[31:0]	I	RF 单元的 rs 寄存器的值
RFRD2[31:0]	I	RF 单元的 rt 寄存器的值
EXTout[31:0]	O	扩展单元的输出
NPC[31:0]	O	j 类和 b 类指令的 NPC

ID 功能定义：

序号	功能名称	功能描述
1	读取控制信号	根据当前 IR 值，取出相应的控制信号
2	读取操作数	根据当前 IR 值，取出 rs、rt 寄存器的值
3	回写寄存器	当 RegWrite 信号有效时，将 WD 的值写入 A3_W 寄存器
4	扩展立即数	根据 EXTop 信号，将 16 位立即数进行相应的扩展，并得出 b 类指令的 zero 信号
5	计算 NPC	计算 b 类和 j 类指令的 NPC

3. EX 级组合逻辑

（1）MFALUAE,MFALUBE 转发多选器，控制进入 ALU 的两路 rs、rt

数据为最新的正确的数据，MUX_ALUA,MUX_ALUB 功能多选器，根据当前指令计算出 ALU 单元的两个运算数

（2）ALU 模块，进行计算并产生结果

（3）MD 模块，内部包含 HI 和 LO 寄存器，进行乘除法的运算并输出 HI、LO 寄存器的结果。

ALU 端口定义：

信号名	方向	描述
ALUA[31:0]	I	第一个操作数
ALUB[31:0]	I	第二个操作数
IR_E[31:0]	I	E 级的 IR 值，用于取出控制信号
ALUout[31:0]	O	将两个操作数进行运算后的结果

ALU 功能定义：

序号	功能名称	功能描述
1	将输入的两个 32 位操作数进行算数逻辑运算	当 ALUctr 分别为 add、sub、cor、cand、cxor、sign_right、zero_right、left、cnor、sign_less、zero_less 时，计算出相应的 ALUout

MD 端口定义：

信号名	方向	描述
IR [31:0]	I	当前指令操作码
clk	I	时钟信号
reset	I	复位信号
D1[31:0]	I	第一个操作数

D2[31:0]	I	第二个操作数
busy	O	表示计算单元正在计算的信号
start	O	启动计算的信号
HIout[31:0]	O	HI 寄存器输出
LOout[31:0]	O	LO 寄存器输出

MD 功能定义：

序号	功能名称	功能描述
1	进行乘除法运算	当根据当前操作码，计算出的控制信号是 start 时，开始进行计算，并将下一跳 M 类指令冻结在 D 级，清除 E 级信息，冻结 PC。并输出 busy 为 1 表示正在计算。
2	写 HI、LO 寄存器	当 DIVWrite 信号有效时，将 rs 的值写入 HI 或 LO
3	输出 HI、LO 寄存器的值	将 HI、LO 寄存器的值输出

4. MEM 级组合逻辑

（1）包含 MEM 模块，和 MFDMF 转发多选器

（2）MEM 模块中包含 dm 存储器，被当做内存存储；MFDMF 是 WD_mem 前的多路选择器，用于转发回到 WD_mem 端口的正确信息。

MEM 端口定义：

信号名	方向	描述
IR_M[31:0]	I	M 级的操作码，用于取出相应的控制信号
AO_M[31:0]	I	写入内存的数据（初步）

PC4_M[31:0]	I	M 级的 PC4，用于输出回写时的 PC
WD_mem[31:0]	I	回写内存的值
reset	I	复位信号
clk	I	时钟信号
RD[31:0]	O	从内存中读取的数据

MEM 功能定义：

序号	功能名称	功能描述
1	写入内存数据	当写使能信号有效时，将 WD 写入相应的 dm 中；
2	读取内存数据	将 AO_M 所代表地址的数据输出
3	复位功能	当时钟上升沿到来时，如果 reset 信号有效，将 dm 清零

5. WD 级

(1)功能多选器 MUX_WD，用于选择回写到 RF 单元的数据。

(2)扩展单元，LBEXT，将 MEM 输出的回写数据，根据 lb、lbu、lh、lhu、lw 进行分割和扩充，并将正确结果写在 DM_W 中

6. MUX（多路选择器）

（1）功能多路选择器

多路选择器名称	控制信号	描述
MUX_PC	npcsel	当 npcsel 为`ADD4 时，PCO=ADD4； 当 npcsel 为`NPC 时，PCO=NPC;

		当 npcsel 为`MFPCF 时，PC0=MFPCF; 否则 PC0=32’h00003000;
MUX_ALUB	ALUBctr	当 ALUBctr 为 0 时，将 V2_E 接到 ALUB; 当 ALUBctr 为 1 时，将 E32 接到 ALUB;
MUX_ALUA	ALUActr	当 ALUActr 为 0 时，将 V1_E 接到 ALUB; 当 ALUActr 为 1 时，将 s 的扩展接到 ALUB;
MUX_WD	MemtoReg	当 MemtoReg 为`A0 时 WD=A0_W;  当 MemtoReg 为`DR 时 WD=DM_W; 当 MemtoReg 为`PC4 时 WD=PC4_W;  当 MemtoReg 为`PC8 时 WD=PC4_W+4; 当 MemtoReg 为`HI 时 WDdata=HI_W;  当 MemtoReg 为`LO 时 WDdata=LO_W; 否则 WD=0;

（2）转发多路选择器

转发多路选择器名称	控制信号	输入0	输入1	输入2	输入3	输入4	输入5	输入6	输入7	输入8	输入9
MFCMP1D	MCMP1D[3:0]	AO_M	PC4_M	HI_M	LO_M	AO_W	PC4_W	HI_W	LO_W	DM_W	RFRD1
MFCMP2D	MCMP2D[3:0]	AO_M	PC4_M	HI_M	LO_M	AO_W	PC4_W	HI_W	LO_W	DM_W	RFRD2
MFALUAE	MALUAE[3:0]	AO_M	PC4_M	HI_M	LO_M	AO_W	PC4_W	HI_W	LO_W	DM_W	V1_E
MFALUBE	MALUBE[3:0]	AO_M	PC4_M	HI_M	LO_M	AO_W	PC4_W	HI_W	LO_W	DM_W	V2_E
MFWDM	MWDM[3:0]	AO_W	PC4_W	HI_W	LO_W	DM_W	V2_M				
CMP代表在哪个模块前转发，D代表D级		输入端口从左到右优先级一次减小									

7. 暂停

	Tnew	功能部件
add/addu/sub/subu/and/or/xor/nor sll/sra/sll/mflo/mfhi addi/addiu/andi/ori/xori/lui/ sllv/srlv/srav/slt/sltu/slti/sltiu	1	ALU
lbu/lb/lhu/lh/lw	2	DM
jal/jalr	0	PC

	Tuse	
	rs	rt
beq/bne	0	0
blez/bgez/bltz/bgtz/jr/jalr	0	
add/addu/sub/subu/and/or/xor/nor sllv/srlv/srav/slt/sltu/mult/div/multu/divu	1	1
sb/sh/sw	1	2
sll/sra/sll		1
addi/addiu/andi/ori/xori/lui/ lbu/lb/lhu/lh/lw/slti/sltiu/mtlo/mthi	1	

rs 转发暂停矩阵

	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F

rt 转发暂停矩阵

	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F

二、测试程序

```
lui $1,0xf0ff

ori $2,$1,0xabbd

sll $3,$1,3

sra $4,$1,3
```

```
srl $5,$1,3

addu $6,$4,$3

add $7,$4,$3

sub $8,$3,$4

nor $22,$3,$2

subu $8,$3,$4

or $9,$3,$4

nor $10,$9,$8

xor $11,$9,$8

mult $1,$2

mflo $12

mfhi $13

mtlo $5

mthi $6

mfhi $5

mflo $6

multu $1,$2

mflo $5

mfhi $6
```

```
div $1,$2

mflo $5

mfhi $6

divu $1,$2

mflo $5

mfhi $6

beq $5,$6,loop

nop

andi $9,$4,9

loop:jal loop1

nop

nop

nop

sll $0,$0,0

loop1:xori $10,$31,100

ori $9,$0,0x30a8

jalr $5,$9

nop

and $4,$3,$4
```

```
srav $19,$12,$9

srlv $20,$12,$9

sllv $21,$12,$9

sw $1,0($0)

sb $2,4($0)

sb $2,5($0)

sh $2,6($0)

lb $22,0($0)

lbu $22,1($0)

lh $23,2($0)

lhu $24,4($0)

lw $25,4($0)

addi $4,$4,4

addiu $5,$5,5

andi $6,$6,6

xori $7,$7,7

lui $4,3

ori $5,$0,4

slti $6,$5,4
```

```
sltiu $7,$5,4

slti $6,$5,5

sltiu $7,$5,5

bne $4,$5,loop2

nop

lui $4,4

loop3:

blez $0,loop4

nop

lui $4,3

loop2:blez $0,loop3

nop

loop4:

jal loop5

nop

lui $4,4

bgtz $31,loop6

nop

lui $4,4
```

```
loop5:
jr $31
nop
lui $4,4
loop6:
lui $4,0xf000
bltz $4,loop7
nop
lui $4,4
loop7:
slt $9,$4,$5
sltu $10,$5,$4
ori $16,0x316c
jalr $3,$16
nop
nop
add $4,$4,$4
j end
nop
```

```
sll $4,$4,3

end:
```

运行结果：

```
55@00003000: $ 1 <= f0ff0000
65@00003004: $ 2 <= f0ffabbd
75@00003008: $ 3 <= 87f80000
85@0000300c: $ 4 <= fe1fe000
95@00003010: $ 5 <= 1e1fe000
105@00003014: $ 6 <= 8617e000
115@00003018: $ 7 <= 8617e000
125@0000301c: $ 8 <= 89d82000
135@00003020: $22 <= 08005442
145@00003024: $ 8 <= 89d82000
155@00003028: $ 9 <= ffffe000
165@0000302c: $10 <= 00001fff
175@00003030: $11 <= 7627c000
255@00003038: $12 <= 41430000
265@0000303c: $13 <= 00e113f0
295@00003048: $ 5 <= 8617e000
305@0000304c: $ 6 <= 1e1fe000
385@00003054: $ 5 <= 41430000
395@00003058: $ 6 <= e2dfbfad
525@00003060: $ 5 <= 00000001
535@00003064: $ 6 <= ffff5443
665@0000306c: $ 5 <= 00000000
675@00003070: $ 6 <= f0ff0000
715@0000307c: $ 9 <= 00000000
725@00003080: $31 <= 00003088
745@00003094: $10 <= 000030ec
755@00003098: $ 9 <= 000030a8
775@0000309c: $ 5 <= 000030a4
795@000030a8: $19 <= 00414300
805@000030ac: $20 <= 00414300
815@000030b0: $21 <= 43000000
815@000030b4: *00000000 <= f0ff0000
825@000030b8: *00000004 <= 000000bd
835@000030bc: *00000004 <= 0000bdbd
845@000030c0: *00000004 <= abbdbdbd
865@000030c4: $22 <= 00000000
875@000030c8: $22 <= 00000000
885@000030cc: $23 <= fffff0ff
```

```
895@000030d0: $24 <= 0000bdbd
905@000030d4: $25 <= abbdbdbd
915@000030d8: $ 4 <= fe1fe004
925@000030dc: $ 5 <= 000030a9
935@000030e0: $ 6 <= 00000000
945@000030e4: $ 7 <= 8617e007
955@000030e8: $ 4 <= 00030000
965@000030ec: $ 5 <= 00000004
975@000030f0: $ 6 <= 00000000
985@000030f4: $ 7 <= 00000000
995@000030f8: $ 6 <= 00000001
1005@000030fc: $ 7 <= 00000001
1075@00003120: $31 <= 00003128
1115@00003128: $ 4 <= 00040000
1145@00003144: $ 4 <= f0000000
1185@00003154: $ 9 <= 00000001
1195@00003158: $10 <= 00000001
1205@0000315c: $16 <= 0000316c
1225@00003160: $ 3 <= 00003168
1245@0000316c: $ 4 <= e0000000
```

四、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？**

答：根据高内聚低耦合的设计原则，ALU 与乘除法模块的输入输出接口不同，内部运算的周期不尽相同，控制信号个数、位数也不尽相同。如果将其整合到 ALU 中的话，会影响 ALU 原有计算的效率，多了很多控制信号，ALU 内部设计会很复杂。

应该有确定的寄存器来存放结果，从而使得计算结果可以快速取得；而且乘除法的结果，不应该受到外部影响而改变，所以，要单独有两个寄存器。

2. **参照你对延迟槽的理解，试解释“乘除槽”。**

乘除法进入 EX 级之后，进行运算，5 个周期之后才能出结果，在这个过程中，后面一条指令进入 D 级，如果是不相干的指令就让他继续执行。如果相关，就要等 5 个周期结束之后才能继续运算。将和该指令计算结果姓关的指令暂停，不相干的指令放走。

3. 为何上文文末提到的 lb 等指令使用的数据扩展模块应在 MEM/WB 之后，而不能在 DM 之后?

将数据扩展单元放在 MEM/WB 之前，因为 MEM 级已经有 DM 模块了，所以会将 MEM 级的整体延迟时间增大，会降低流水线的整体效率。而 WD 级只有一个多路选择器，加上扩展单元后，该模块的延迟对整体延迟的影响不大。

4. 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）

一个英文字符占一个字节，在 C 语言中，当对字符串的处理是以字符为单位（非 4 的整数倍）的操作时，用字节访问内存效率会更高。

5. 如何概括你所设计的 CPU 的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段？

我的 cpu 属于规划者型，按照高老板的写法，可以很好的在控制器掌握全局，判断哪里会产生冲突，提前暂停，或是准备好转发。这种设计更符合工程化方法的设计要求，具有可移植性，能够全面的考虑转发暂停。出现错误之后也能够快速的查找出错点，需要的逻辑少，思考上较为简单，行动上可能需要更多一些的代码以及连接信号。

我利用了宏，让自己的代码可读性高，便于追错，将设计逻辑显示表达。将属于同一类的模块，按照高内聚低耦合原则进行封装。

6. 你对流水线 CPU 设计风格有何见解？

流水线的风格没有哪个更好，各有利弊，关键是看你能否根据自己的需求选择适合自己的方法。而且两种方法也不是完全对立的，可以相互融合，保证自己的设计准确、可读、添加指令方便。

7. 在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来

将指令分为 7 类，每一类的数据通路大致相同,从而按照类来进行冲突分析。

addu/add/subu/sub/nor/or/xor/slt/sltu/sllv/srav/srlv/and	1
addi/addiu/andi/lui/ori/xori/slti/sltiu/	2
lb/lbu/lh/lhu/lw	3
sb/sw/sh	4
sll/sra/srl	5
b	6
j	7

前序指令	后续指令	前序指令位置-后续指令位置	处理方式	样例

addu 类	addu	M-E
		W-E
			转发
			转发
				addu $5,$1,$2
				instr(0,1)
				addu $6,$5,$3

	ori	M-E W-E	转发 转发	addu $5,$1,$2 instr(0,1) ori $6,$5,0xcccc
	sll	M-E W-E	转发 转发	addu $5,$1,$2 instr(0,1) sll $6,$5,3
	beq	E-D M-D W-D	暂停 转发 转发	addu $5,$1,$2 instr(0,1, 2) beq $5,$0,loop@
	jr	E-D M-D W-D	暂停 转发 转发	addu $5,$1,$2 instr(0,1, 2) jr $5
	sw	W-M	转发	addu $4,$3,$0 sw $4,0($4)

	lw	M-E
		W-E
			转发
			转发
				addu $5,$4,$0
				instr(0,1)
				lw $6,0($5)

ori 类	addu/subu	M-E W-E	转发 转发	ori $5,$0,0x1111 instr(0,1) addu/subu $6,$5,$5
	ori	M-E W-E	转发 转发	ori $5,$0,0x1111 instr(0,1)

				ori $6,$5,0xcccc
	sll	M-E W-E	转发 转发	ori $5,$1,2 instr(0,1) sll $6,$5,3
	beq	E-D M-D W-D	暂停 转发 转发	ori $5,$0,0x1111 instr(0,1,2) beq $5,$1,loop@
	jr	E-D M-D W-D	暂停 转发 转发	ori $5,$0,0x30bc instr(0,1,2) jr $5
	sw	W-M	转发	ori $5,$0,0x1111 sw $5,0($0)
	lw	M-E W-E	转发 转发	ori $5,$4,1 instr(0,1) lw $4,0($5)
sll 类	addu/subu	M-E W-E	转发 转发	lui $5,0x1111 instr(0,1) addu/subu $6,$5,$5
	ori	M-E W-E	转发 转发	lui $5,0x1111 instr(0,1) ori $6,$5,0xcccc
	beq	E-D M-D W-D	暂停 转发 转发	lui $5, 0x1111 instr(0,1,2) beq $5,$1,loop@
	jr	E-D M-D W-D	暂停 转发 转发	lui $5,0x30bc instr(0,1,2) jr $5
	sw	W-M	转发	lui $5,0x1111

				sw $5,0($0)
	lw	M-E W-E	转发 转发	lui $5,0x1111 instr(0,1) lw $40($5)

lw 类	addu/subu	M-E
		W-E
			暂停
			转发
				lw $5,4($0)
				instr(0,1)
				addu $6,$5,$4@

	ori	M-E W-E	暂停 转发	lw $5,4($0) instr(0,1) ori $6,$5,$4
	sll	M-E W-E	转发 转发	lw $5,0($1) instr(0,1) sll $6,$5,3
	beq	E-D M-D W-D	暂停 暂停 转发	lw $5,4($0) instr(0,1,2) beq $5,$4,loop@
	jr	E-D M-D W-D	暂停 暂停 转发	lw $5,4($0) instr(0,1,2) jr $5
	sw	W-M	转发	lw $5,4($0) sw $5,0($5)

	lw	M-E
		W-E
			转发
			转发
				lw $5,0($0)
				instr(0,1)
				lw $6,0($5)

jal 类	addu/subu	M-E W-E	转发 转发	jal loop instr(0,1) addu $6,$31,$4@

	ori	M-E W-E	转发 转发	jal loop instr(0,1) ori $6,$31,$4
	sll	M-E W-E	转发 转发	jal loop instr(0,1) sll $6,$31,3
	beq	E-D M-D W-D	转发 转发 转发	jal loop instr(0,1,2) beq $5,$31,loop@
	jr	E-D M-D W-D	转发 转发 转发	jal loop instr(0,1,2) jr $31

	sw	W-M	转发	lw $5,4($0)
				
				sw $5,0($5)

	lw	M-E
		
		W-E
			转发
			
			转发
				lw $5,0($0)
				
				instr(0,1)
				
				lw $6,0($5)