计算机组成课程设计P6实验报告

1. 数据通路设计
2. F级组合逻辑
3. 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
4. PC模块根据根据是否暂停，决定是否更新PC
5. 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

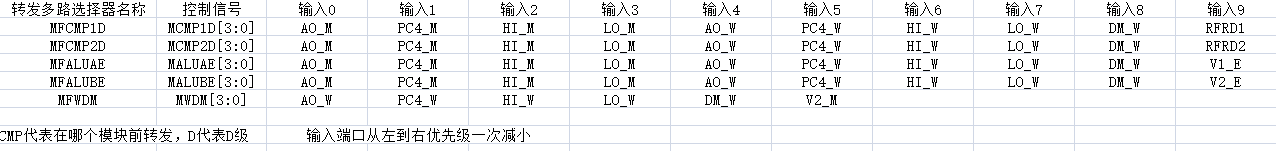
(1)功能多选器MUX\_WD，用于选择回写到RF单元的数据。

(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

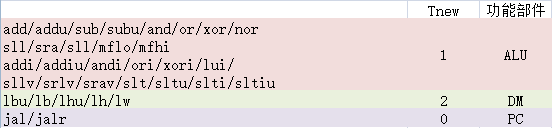
6.MUX（多路选择器）

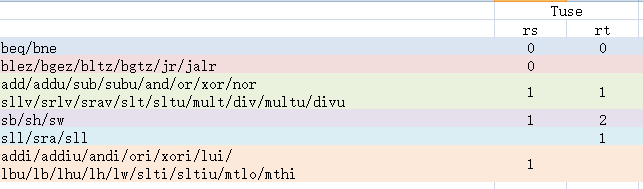
（1）功能多路选择器

|  |  |  |
| --- | --- | --- |
| 多路选择器名称 | 控制信号 | 描述 |
| MUX\_PC | npcsel | 当npcsel为`ADD4时，PC0=ADD4；  当npcsel为`NPC时，PC0=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`AO时WD=AO\_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; |

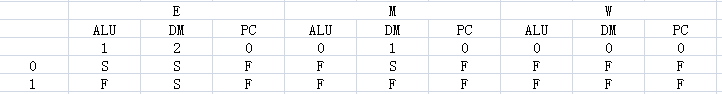
（2）转发多路选择器

7.暂停

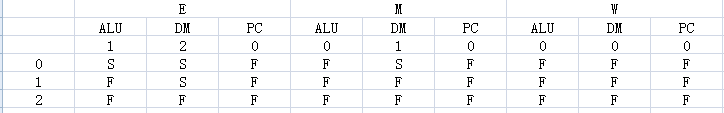




rs转发暂停矩阵



rt转发暂停矩阵



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。

1. 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。

1. 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。

1. 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。

1. 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？

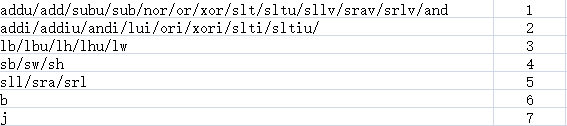
我的cpu属于规划者型，按照高老板的写法，可以很好的在控制器掌握全局，判断哪里会产生冲突，提前暂停，或是准备好转发。这种设计更符合工程化方法的设计要求，具有可移植性，能够全面的考虑转发暂停。出现错误之后也能够快速的查找出错点，需要的逻辑少，思考上较为简单，行动上可能需要更多一些的代码以及连接信号。

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。

1. 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。

1. 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。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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) |