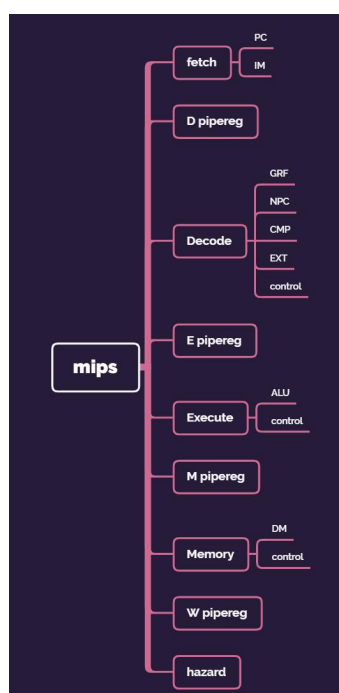


计算机组成原理实验报告参考模板

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, j, jal, jr, nop}。为了实现这些功能, CPU 主要包含了 IM、GRF 等模块, 这些模块按照以下顶层设计逐级展开。



（二）关键模块定义

1.PC

（1）端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	同步置零
3	NPC	I	下一条指令在 IM 的地址
4	en	I	阻塞信号

5	PC	O	当前指令在 IM 的地址
---	----	---	--------------

(2) 功能定义

序号	功能	描述
1	存储指令地址	保存当前执行指令在 IM 中的地址
2	阻塞	阻塞为 0 时，PC 不工作

2. IM

(1) 端口说明

序号	信号名	方向	描述
1	PC[31:0]	I	当前指令在 IM 的地址
2	Instr[31:0]	O	当前 PC 对应的指令

(2) 功能定义

序号	功能	描述
1	取指令	读取指令

3. D_pipereg

(1) 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	同步复位信号
3	en	I	阻塞信号
4	F_PC	I	F 级的 PC
5	F_instr	I	F 级的指令
6	D_PC	O	D 级的 PC
7	D_instr	O	D 级的指令

(2) 功能定义

序号	功能	描述
1	存储要流水的值	存储要流水的值

4. NPC

(1) 端口说明

序号	信号名	方向	描述
1	PCSrc	I	分支信号
2	PCj	I	j 型跳转信号
3	PCjr	I	jr 和 jalr 跳转信号
4	D_PC[31:0]	I	F 级 PC
5	F_PC[31:0]	I	F 级 PC
6	Regjr[31:0]	I	jr 和 jalr 跳转的地址
7	instr_index[25:0]	I	D 级 26 位立即数
8	NPC[31:0]	O	下一个 PC 的值

(2) 功能定义

序号	功能	描述
1	计算下一个 PC	计算下一个 PC

5. EXT

(1) 端口说明

序号	信号名	方向	描述
1	imm	I	16 位立即数
2	sign	I	符号扩展选择信号 0: 进行无符号扩展 1: 进行符号扩展
3	SignImm	O	扩展后的 32 位数

(2) 功能定义

序号	功能	描述
1	无符号扩展	当 sign 为 0 时, 进行无符号扩展
2	符号扩展	当 sign 为 1 时, 进行有符号扩展

6. GRF

(1) 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	异步复位信号， 1: 清零 0: 保持
3	PC[31:0]	I	D 级 PC 的值，用于 display
4	A1[4:0]	I	5 位地址输入，从外界接收[25: 21]， 代表 32 个寄存器中的一个，并将其中的 值输出到 RD1
5	A2[4:0]	I	5 位地址输入，从外界接收[20: 16]， 代表 32 个寄存器中的一个，并将其中的 值输出到 RD2
6	A3[4:0]	I	5 位地址输入，选择 32 个寄存器中的 一个，将 WD 输入的数据存储到其中
7	WD[31:0]	I	32 位写入数据
8	RD1[31:0]	O	输出 A1 代表的寄存器中的值
9	RD2[31:0]	O	输出 A2 代表的寄存器中的值

(2) 功能定义

序号	功能	描述
1	异步复位	Reset 为 1 时，所有寄存器被清零
2	读数据	将 A1 和 A2 对应的寄存器中的值输出到 RD1 和 RD2
3	写数据	如果 WE 为 1,时钟上升沿时将 WD 中的数据 写入到 A3 对应的寄存器中

7. CMP

(1) 端口说明

序号	信号名	方向	描述
1	A	I	操作数 A
2	B	I	操作数 B
3	D_equal	O	相等信号

(2) 功能定义

序号	功能	描述
1	判断 A、B 是否相等	若 A=B，则输出 1

8. E_pipereg

(1) 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	同步复位信号
3	en	I	阻塞信号
4	flush	I	清洗信号
5	D_PC[31:0]	I	D 级 PC 值
6	D_instr[31:0]	I	D 级指令
7	D_signimm[31:0]	I	D 级扩展后 32 位数
8	D_RD1[31:0]	I	D 级 GRF 输出 1
9	D_RD2[31:0]	I	D 级 GRF 输出 2
10	D_A3[4:0]	I	D 级 A3 地址
11	D_WD[31:0]	I	D 级 GRF 写入值
12	E_PC[31:0]	O	E 级 PC 值
13	E_instr[31:0]	O	E 级指令
14	E_signimm[31:0]	O	E 级扩展后 32 位数
15	E_RD1[31:0]	O	E 级 RD1

16	E_RD2[31:0]	O	E 级 RD2
17	E_A3[4:0]	O	E 级 A3 地址
18	E_WD[31:0]	O	E 级 WD

(2) 功能定义

序号	功能	描述
1	存储要流水的值	存储要流水的值

9. ALU

(1) 端口说明

序号	信号名	方向	描述
1	op[2:0]	I	选择执行哪一个操作 000: 加法 001: 减法 010: 与 011: 或 100: 左移 16 位
2	inA[31:0]	I	参与运算的第一个数
3	inB[31:0]	I	参与运算的第二个数
4	ALUResult[31:0]	O	输出 inA 和 inB 操作后的结果

(2) 功能定义

序号	功能	描述
1	加法	将两个输入加起来输出到 ALUResult
2	减法	将两个输入相减输出到 ALUResult
3	与运算	对两个输入进行与操作输出结果到 ALUResult
4	或运算	对两个输入进行或操作输出结果到 ALUResult
5	左移 16 位	将 SrcB 左移 16 位输出到 ALUResult

10. M_pipereg

(1) 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	同步复位信号
3	flush	I	清洗信号
4	E_PC[31:0]	I	E 级 PC
5	E_WD[31:0]	I	E 级 WD
6	E_A3[4:0]	I	E 级 A3
7	E_instr[31:0]	I	E 级 instr
8	E_ALUResult[31:0]	I	E 级 ALUResult
9	E_RD2[31:0]	I	E 级 RD2
10	M_PC[31:0]	O	M 级 pc
11	M_WD[31:0]	O	M 级 WD
12	M_A3[4:0]	O	M 级 A3
13	M_instr[31:0]	O	M 级 instr
14	M_ALUResult[31:0]	O	M 级 ALUResult
15	M_RD2[31:0]	O	M 级 RD2

(2) 功能定义

序号	功能	描述
1	存储要流水的值	存储要流水的值

11. DM

(1) 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	同步复位信号
3	DMaddr[31:0]	I	减去 0x3000, 择取其[13:2]位, 作为

			读取或写入信号的地址
4	WD[31:0]	I	32 位要写入的数据
5	WE	I	写使能信号，外界对应 MemWrite
6	RD[31:0]	O	32 位读出的数据
7	PC[31:0]	I	用于 display，满足课下需求

(2) 功能定义

序号	功能	描述
1	同步复位	Reset 为 1 且为时钟上升沿时，所有寄存器被清零
2	读数据	将 A 地址对应的数据输出到 RD（这个功能始终存在，不管 WE 是否为 1）
3	写数据	WE 为 1 且时钟上升沿到来时，将 WD 的数据写入到 A 对应的地址

12. W_pipereg

(1) 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	同步复位信号
3	M_PC[31:0]	I	M 级 PC
4	M_instr[31:0]	I	M 级 instr
5	M_A3[4:0]	I	M 级 A3
6	M_WD[31:0]	I	M 级 WD
7	W_PC[31:0]	O	W 级 PC
8	W_instr[31:0]	O	W 级 instr
9	W_A3[4:0]	O	W 级 A3
10	W_WD[31:0]	O	W 级 WD

(2) 功能定义

序号	功能	描述
----	----	----

1	存储要流水的值	存储要流水的值
---	---------	---------

13. Hazard

(1) 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	同步复位信号
3	D_A1[4:0]	I	D 级读取 rs 的序号
4	D_A2[4:0]	I	D 级读取 rt 的序号
5	D_RD1[31:0]	I	D 级读取 rs 的值
6	D_RD2[31:0]	I	D 级读取 rt 的值
7	D_rs_Tuse	I	D 级是否在使用 rs
8	D_rt_Tuse	I	D 级是否在使用 rt
9	E_A1[4:0]	I	E 级读取 rs 的序号
10	E_A2[4:0]	I	E 级读取 rt 的序号
11	E_RD1[31:0]	I	E 级读取 rs 的值
12	E_RD2[31:0]	I	E 级读取 rt 的值
13	E_A3[4:0]	I	E 级写入寄存器的序号
14	E_WD[31:0]	I	E 级写入寄存器的值
15	E_rs_Tuse	I	E 级是否使用 rs
16	E_rt_Tuse	I	E 级是否使用 rt
17	M_A2[4:0]	I	M 级读取 rt 的序号
18	M_RD2[31:0]	I	M 级读取到寄存器的值
19	M_A3[4:0]	I	M 级写入寄存器的序号
20	M_WD[31:0]	I	M 级写入寄存器的值
21	W_A3[4:0]	I	W 级写入寄存器的序号
22	W_WD[31:0]	I	W 级写入寄存器的值
23	D_Forward1[31:0]	O	转发给 D 级的值 1

24	D_Forward2[31:0]	O	转发给 D 级的值 2
25	E_Forward1[31:0]	O	转发给 E 级的值 1
26	E_Forward2[31:0]	O	转发给 E 级的值 2
27	M_Forward2[31:0]	O	转发给 M 级的值 2
28	F_en	O	PC 的使能信号
29	D_pipereg_en	O	D 级流水线寄存器的使能信号
30	E_pipereg_en	O	E 级流水线寄存器的使能信号
31	E_pipereg_flush	O	E 级流水线寄存器的清洗信号
32	M_pipereg_flush	O	M 级流水线寄存器的清洗信号

(2) 功能定义

序号	功能	描述
1	产生各转发值	产生各转发值
2	产生各阻塞信号	产生各阻塞信号
3	产生各清洗信号	产生各清洗信号

14. Control（分布式译码）

(1) 端口说明

序号	信号名	方向	描述
1	instr[31:0]	I	每一级的指令
2	D_equal	I	D 级 RD1 和 RD2 是否相等
3	sign	O	ext 是否进行有符号扩展
4	PCSrc	O	是否进行分支
5	PCj	O	是否为 J 型指令
6	PCjr	O	是否为 jr 或 jalr
7	D_WDsel	O	D 级中写入 GRF 值的选择信号
8	D_A3[4:0]	O	D 级中写入 GRF 地址的选择信号
9	ALUControl[3:0]	O	ALU 选择哪种计算方式
10	ALUASel	O	ALU 里面 inA 选择哪个作为输入

11	ALUBSel	O	ALU 里面 inB 选择哪个作为输入
12	E_WDSel[1:0]	O	E 级中写入 GRF 值的选择信号
13	MemWrite	O	E 级中写入 GRF 地址的选择信号
14	width[1:0]	O	写入 DM 的位宽
15	sign_1	O	写入 DM 的值是否进行有符号扩展
16	M_WDSel	O	M 级中写入 GRF 值的选择信号
17	D_rs_Tuse	O	D 级是否读取 rs
18	D_rt_Tuse	O	D 级是否读取 rt
19	E_rs_Tuse	O	E 级是否读取 rs
20	E_rt_Tuse	O	E 级是否读取 rt

(2) 真值表

端口	addu	subu	ori	lw	sw	beq	lui
func	100001	100011	n/a	n/a	n/a	n/a	n/a
op	000000	000000	001101	100011	101011	000100	001111
sign	x	x	0	1	1	x	x
Branch	0	0	0	0	0	1	0
MemWrite	0	0	0	0	1	0	0
RegWrite	1	1	1	1	0	0	1
MemtoReg	0	0	0	1	x	x	0
ALUSrc	0	0	1	1	1	0	1
RegDst	1	1	0	0	x	x	0
ALUControl	000	001	011	000	000	001	100

(三) 重要机制实现方法

1. 跳转

CMP 模块在 D 级生成跳转信号

2. 流水线延迟槽

保证在 b 指令和 j 指令来临时，一定会执行后一条指令，根据判断结果，再

进行跳转

3. 转发

在 hazard 统一生成，再传到各个流水线级中

4. 进入中断处理程序

用 en 和 flush 进行终端和清洗

二、测试方案

(1) 测试代码

```
ori $a0,$0,0x100
ori $a1,$a0,0x123
lui $a2,456
lui $a3,0xffff
ori $a3,$a3,0xffff
addu $s0,$a0,$a2
addu $s1,$a0,$a3
addu $s4,$a3,$a3
subu $s2,$a0,$a2
subu $s3,$a0,$a3
sw $a0,0($0)
sw $a1,4($0)
sw $a2,8($0)
sw $a3,12($0)
sw $s0,16($0)
sw $s1,20($0)
sw $s2,24($0)
sw $s3,44($0)
sw $s4,48($0)
lw $a0,0($0)
lw $a1,12($0)
sw $a0,28($0)
sw $a1,32($0)
ori $a0,$0,1
ori $a1,$0,2
ori $a2,$0,1
beq $a0,$a1,loop1
beq $a0,$a2,loop2
loop1: sw $a0,36($t0)
loop2: sw $a1,40($t0)
jal loop3
```

```

jal loop3
sw $s5,64($t0)
ori $a1,$a1,4
jal loop4
loop3:sw $a1,56($t0)
sw $ra,60($t0)
ori $s5,$s5,5
jr $ra
loop4: sw $a1,68($t0)
sw $ra,72($t0)

```

(2) MARS 中结果

```

@00003000: $ 4 <= 00000100
@00003004: $ 5 <= 00000123
@00003008: $ 6 <= 01c80000
@0000300c: $ 7 <= ffff0000
@00003010: $ 7 <= ffffffff
@00003014: $16 <= 01c80100
@00003018: $17 <= 000000ff
@0000301c: $20 <= ffffffff
@00003020: $18 <= fe380100
@00003024: $19 <= 00000101
@00003028: *00000000 <= 00000100
@0000302c: *00000004 <= 00000123
@00003030: *00000008 <= 01c80000
@00003034: *0000000c <= ffffffff
@00003038: *00000010 <= 01c80100
@0000303c: *00000014 <= 000000ff
@00003040: *00000018 <= fe380100
@00003044: *0000002c <= 00000101
@00003048: *00000030 <= ffffffff
@0000304c: $ 4 <= 00000100
@00003050: $ 5 <= ffffffff
@00003054: *0000001c <= 00000100
@00003058: *00000020 <= ffffffff
@0000305c: $ 4 <= 00000001
@00003060: $ 5 <= 00000002
@00003064: $ 6 <= 00000001
@00003070: *00000024 <= 00000001
@00003074: *00000028 <= 00000002
@00003078: $31 <= 00003080
@0000307c: $31 <= 00003084
@00003080: *00000040 <= 00000000
@0000308c: *00000038 <= 00000002
@00003090: *0000003c <= 00003084
@00003094: $21 <= 00000005
@0000309c: *00000044 <= 00000002
@00003084: $ 5 <= 00000006
@00003088: $31 <= 00003090
@0000308c: *00000038 <= 00000006
@0000309c: *00000044 <= 00000006
@000030a0: *00000048 <= 00003090

```

(3) 该 CPU 中的结果

```

-----
45@00003000: $ 4 <= 00000100
55@00003004: $ 5 <= 00000123
65@00003008: $ 6 <= 01e80000
75@0000300c: $ 7 <= ffff0000
85@00003010: $ 7 <= ffffffff
95@00003014: $16 <= 01e80100
105@00003018: $17 <= 000000ff
115@0000301c: $20 <= ffffffff
125@00003020: $18 <= fe380100
135@00003024: $19 <= 00000101
135@00003028: *00000000 <= 00000100
145@0000302c: *00000004 <= 00000123
155@00003030: *00000008 <= 01e80000
165@00003034: *0000000c <= ffffffff
175@00003038: *00000010 <= 01e80100
185@0000303c: *00000014 <= 000000ff
195@00003040: *00000018 <= fe380100
205@00003044: *0000002c <= 00000101
215@00003048: *00000030 <= ffffffff
235@0000304c: $ 4 <= 00000100
245@00003050: $ 5 <= ffffffff
245@00003054: *0000001c <= 00000100
255@00003058: *00000020 <= ffffffff
275@0000305c: $ 4 <= 00000001
285@00003060: $ 5 <= 00000002
295@00003064: $ 6 <= 00000001
315@00003070: *00000024 <= 00000001
325@00003074: *00000028 <= 00000002
345@00003078: $31 <= 00003080
355@0000307c: $31 <= 00003084
355@0000308c: *00000038 <= 00000002
365@0000308c: *00000038 <= 00000002
375@00003090: *0000003c <= 00003084
395@00003094: $21 <= 00000005
405@0000309c: *00000044 <= 00000002
425@00003084: $ 5 <= 00000006
435@00003088: $31 <= 00003090
435@0000308c: *00000038 <= 00000006
445@0000309c: *00000044 <= 00000006
455@000030a0: *00000048 <= 00003090

```

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。

```
1 module NPC(  
2     input PCSrc,      // branch  
3     input PCj,  
4     input PCjr,  
5     input [31:0] D_PC,    // 贯彻命名风格,流水级_str  
6     input [31:0] F_PC,  
7     input [31:0] Regjr,  
8     input [25:0] instr_index,  
9     output reg [31:0] NPC  
10 );  
11 // wire [31:0] D_PC4;  
12 // assign D_PC4 = D_PC + 4;  
13 always @(*) begin  
14     if (PCSrc) begin  
15         NPC = D_PC + 4 + {{14{instr_index[15]}}, instr_index[15:0], 2'b00};  
16     end  
17     else if (PCj) begin  
18         NPC = {D_PC[31:28], instr_index, 2'b00};  
19     end  
20     else if (PCjr) begin  
21         NPC = Regjr;  
22     end  
23     else begin  
24         NPC = F_PC + 4;  
25     end  
26 end  
27 endmodule
```

数据通路: 通过 NPC 产生新的 PC, 再传入到 PC 中

控制信号: Control 解码出对应的 PC

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 ?

因为后面有延迟槽。

(二) 数据冒险的分析

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？

暂停-转发就是要算出来，在流水线寄存器中停一下，然后再转发。这样设计简单，而且是课程要求。

(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;

- i. 就是转发的优先级。以 jr 为例，在 D 级就要产生写入的 NPC 数据，此时若 E 级和 M 级要求写入同一个寄存器，导致数据冲突，则需要选择 E 级流水线的的数据。
- ii. 简化流水转发，不使用内部转发就要把每周周期 GRF 的使用分成前后两周期，前周期用于写入，后周期用于读取。
- iii. 因为不可能向 0 号寄存器写入东西，0 号寄存器保持为 0
- iv. 级数低的流水层产生的优先。

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？

如果不需要写寄存器，直接将 A 译码为为 0，这样甚至可以省略 we。

(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？

有 sw 和 r 型指令的冲突；定义相应的阻塞信号，以解决冲突。

2. 如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。

不是手动构造的

3. 此思考题请同学们结合自己测试 CPU 使用的具体手段，按照自己的实际情况进行回答。

- 画电路图，统一线的命名规范；
- 画暂停转发表，构造暂停条件
- 在电路图中找到要转发的位置，并根据命令情况将要转发的线连到需要转发的位置，以此构造转发命令。
- 使用讨论区的测试数据