流水线 CPU（Verilog 实现）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，支持的指令集包括{addu, subu, ori, beq, lw, sw, lui, j, jal, jr, nop}，为了实现这些功能，CPU 主要包括了 Fench, FD, Decode, DE, Excute，EM，Memory，MW。

（二）关键模块定义

1. Fench

内部包含 PC 与 IM 两个部件

表格 1 Fench 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号

2. PC

表格 2 PC 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
EnPC	I	写使能信号 1：可向 PC 中写入数据 0：不能向 PC 中写入数据
npc[31:0]	O	更新 PC 值
tonpc[31:0]	O	更新后 PC 值

3. IM

指令存储器

表格 3 IM 模块端口定义

信号名	方向	描述
tonpc[31:0]	I	当前 PC 值
Instr[31:0]	O	读出指令

4. FD

Fench 与 Decode 之间的流水线寄存器

表格 4　FD 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
EnFD	I	写使能信号 1：可向 FD 中写入数据 0：不能向 FD 中写入数据
PCF[31:0]	I	F 阶段当前 PC 值
InstrF[31:0]	I	F 阶段当前指令
PCD[31:0]	O	D 阶段当前 PC 值
InstrD[31:0]	O	D 阶段当前指令

5.　Decode

内部包含 GRF,EXT,CMP,NPC 四个部件。

表格 5　Decode 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
PCD[31:0]	I	D 阶段当前 PC 值
PCF[31:0]	I	F 阶段当前 PC 值
InstrD[31:0]	I	D 阶段当前指令
FwdD1[31:0]	I	CMP 原件端口转发而来的数据
FwdD2[31:0]	I	CMP 原件端口转发而来的数据
WDW[31:0]	I	由 MW 寄存器连接而来的 GRF 写入数据
A3W[4:0]	I	由 MW 寄存器连接而来的 GRF 写入地址
PCW[31:0]	I	由 MW 寄存器连接而来的 PC 值
RD1DE[31:0]	O	GRF 输出
RD2DE[31:0]	O	GRF 输出
Imm32DE[31:0]	O	EXT 输出
A3DE[4:0]	O	当前指令下的 GRF 写入地址

WDDE[31:0]	O	当前指令下的 GRF 写入输入
NPC[31:0]	O	待更新的 PC 值
D1Use	O	是否要使用 RD1
D2Use	O	是否要使用 RD2

6. GRF

表格 6 GRF 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
A1[4:0]	I	Instr[25:21]
A2[4:0]	I	Instr[20:16]
A3[4:0]	I	Instr[15:11]
WD[31:0]	I	GRF 写入数据
PC[31:0]	I	当前待输入 PC 值
RD1[31:0]	O	寄存器输出 1
RD2[31:0]	O	寄存器输出 2

7. EXT

表格 7 EXT 模块端口定义

信号名	方向	描述
Imm16	I	16 位输入
EXTop	I	零扩展或符号扩展选择信号
EXT_result[31:0]	O	扩展后结果输出

8. CMP

表格 8 CMP 模块端口定义

信号名	方向	描述
SrcA	I	输入 1
SrcB	I	输入 2
eq	O	两者相等
eqz	O	SrcA 等于 0
ltz	O	SrcA 小于零

9. NPC

表格 9 NPC 模块端口定义

信号名	方向	描述
PCF[31:0]	I	F 阶段当前 PC 值
PCD[31:0]	I	D 阶段当前 PC 值
Imm26[25:0]	I	D 阶段指令后 26 位
zero	I	比较两数相等
ra[31:0]	I	GRF 输出 1
npcop[1:0]	I	Npc 选择信号
npc[31:0]	O	下一个 PC 值

10. DE

表格 10 DE 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
reset	I	复位信号
FlushDE	I	DE 寄存器清空信号
EnDE	I	DE 寄存器使能信号
PCD[31:0]	I	D 阶段当前 PC 值
InstrD[1:0]	I	D 阶段当前指令
PCE[31:0]	O	E 阶段当前 PC 值
InstrE[31:0]	O	E 阶段当前指令
RD1D[31:0]	I	D 阶段 GRF 输出 1
RD2D[31:0]	I	D 阶段 GRF 输出 2
Imm32D[31:0]	I	D 阶段 EXT 输出
A3D[4:0]	I	D 阶段 A3 值
WDD[31:0]	I	D 阶段写入数据
RD1E[31:0]	O	E 阶段 GRF 输出 1
RD2E[31:0]	O	E 阶段 GRF 输出 2
Imm32E[31:0]	O	E 阶段 EXT 输出

A3E[4:0]	O	E 阶段 A3 值
WDE[31:0]	O	E 阶段写入数据

11. Excute

表格 11 Extcute 模块端口定义

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
InstrE[31:0]	I	E 阶段当前指令
Imm32E[31:0]	I	E 阶段 EXT 输出
WDE[31:0]	I	E 阶段写入数据
FwdE1[31:0]	I	ALU 端口转发而来的数据
FwdE2[31:0]	I	ALU 端口转发而来的数据
WDEM[31:0]	O	E 阶段后写入数据
ResEM[31:0]	O	E 阶段 ALU 输出数据
RD2EM[31:0]	O	ALU 端口 2 转发而来的数据
E1Use	O	ALU1 端口是否在使用
E2Use	O	ALU2 端口是否在使用

12. ALU

表格 12 ALU 模块端口定义

信号名	方向	描述
SrcA[31:0]	I	ALU 输入 1
SrcB[31:0]	I	ALU 输入 2
ALUop[3:0]	I	ALU 运算方式选择
Result[31:0]	O	ALU 输出

13. EM

表格 13 EM 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
FlushEM	I	清空信号

PCE[31:0]	I	E 阶段当前 PC 值
InstrE[4:0]	I	E 阶段当前指令
A3E[31:0]	I	E 阶段 A3 值
WDE[31:0]	I	E 阶段写入信号
ResE[31:0]	I	E 阶段 ALU 输出
RD2E[31:0]	I	E 阶段 ALU 端口 2
PCM[31:0]	O	M 阶段当前 PC 值
InstrM[31:0]	O	M 阶段当前指令
A3M[4:0]	O	M 阶段当前 A3 值
WDM[31:0]	O	M 阶段当前写入数据
ResM[31:0]	O	M 阶段 ALU 输出
RD2M[31:0]	O	M 阶段 ALU 端口 2

14. Memory

表格 14 Memory 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
PCM[31:0]	I	M 阶段当前 PC 值
InstrM[31:0]	I	M 阶段当前指令
WDM[31:0]	I	M 阶段写入数据
ResM[31:0]	I	M 阶段 ALU 输出
FwdM2[31:0]	I	M 写入端口转发而来的数据
WDMW[31:0]	O	MW 寄存器中写入数据
M2Use	O	M 写入端口是否使用

15. DM

表格 15 DM 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号

MemWrite	I	存储器写入信号
Addr[31:0]	I	写入地址
Data[31:0]	I	存储器写入数据
RD[31:0]	O	存储器输出数据
PC[31:0]	I	M 阶段当前 PC 值

16. MW

表格 16 MW 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
PCM[31:0]	I	M 阶段当前 PC 值
InstrM[31:0]	I	M 阶段当前指令
A3M[31:0]	I	M 阶段 A3 值
WDM[31:0]	I	M 阶段写入数据
PCW[31:0]	O	W 阶段当前 PC 值
InstrW[31:0]	O	W 阶段当前指令
A3W[31:0]	O	W 阶段 A3 值
WDW[31:0]	O	W 阶段写入数据

17. Controller

表格 17 Controller 模块端口定义

信号名	方向	描述
Instr[31:0]	I	32 位指令
MemWrite	O	存储器写入信号
NPCop[1:0]	O	NPC 选择信号
ALUop	O	ALU 运算选择信号
ALUASel	O	ALU 端口 1 输入选择信号
ALUBSel	O	ALU 端口 2 输入选择信号
EXTop	O	EXT 运算选择信号
A3DE[31:0]	O	A3 选择结果

D1Use	O	是否使用 GRF 的 RD1 端口
D2Use	O	是否使用 GRF 的 RD2 端口
E1Use	O	是否使用 ALU 的输入端口 1
E2Use	O	是否使用 ALU 的输入端口 2
M2Use	O	是否使用 Memory 的写入端口
WDSelD	O	选择 D 阶段后的写入数据
WDSelE	O	选择 E 阶段后的写入数据
WDSelM	O	选择 M 阶段后的写入数据

18. Hazard

表格 18 Hazard 模块端口定义

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
A1D[4:0]	I	D 阶段当前 A1 值
A2D[4:0]	I	D 阶段当前 A2 值
RD1D[31:0]	I	D 阶段当前 GRF 输出端口 1
RD2D[1:0]	I	D 阶段当前 GRF 输出端口 2
D1Use	I	是否使用 GRF 的 RD1 端口
D2Use	I	是否使用 GRF 的 RD2 端口
A1E[4:0]	I	E 阶段当前 A1 值
A2E[4:0]	I	E 阶段当前 A2 值
RD1E[31:0]	I	E 阶段当前 GRF 输出端口 1
RD2E[31:0]	I	E 阶段当前 GRF 输入端口 2
E1Use	I	是否使用 ALU 输入端口 1
E2Use	I	是否使用 ALU 输入端口 2
A3E[4:0]	I	E 阶段当前 A3 值
WDE[31:0]	I	E 阶段当前写入数据
A2M[4:0]	I	M 阶段当前 A2 值
RD2M[31:0]	I	M 阶段 ALU 端口 2 转发而来的数据

M2Use	I	Memory 写入端口是否使用
A3M[4:0]	I	M 阶段当前 A3 值
WDM[31:0]	I	M 阶段写入数据
A3W[4:0]	I	W 阶段当前 A3 值
WDW[31:0]	I	W 阶段写入数据
FwdD1[31:0]	O	CMP 端口 1 转发而来的数据
FwdD2[31:0]	O	CMP 端口 2 转发而来的数据
FwdE1[31:0]	O	ALU 端口 1 转发而来的数据
FwdE2[31:0]	O	ALU 端口 2 转发而来的数据
FwdM2[31:0]	O	Memory 写入端口转发而来的数据
EnPC	O	PC 寄存器使能信号
EnDE	O	DE 寄存器使能信号
EnEM	O	EM 寄存器使能信号
FlushDE	O	DE 寄存器清空信号
FlushEM	O	EM 寄存器清空信号

二、CPU 测试

1. addu 测试

Mips 程序：

```
ori $t0, $t0, 123
addu $t1, $t0, $t0
subu $t2, $t1, $t0
ori $t3, $t2, 6666
lui $t4, 7777
lui $t4, 7854
addu $t5, $t3, $t4
subu $t6, $t5, $t1
```

期望输出：

```
@00003000: $ 8 <= 0000007b
@00003004: $ 9 <= 000000f6
@00003008: $10 <= 0000007b
@0000300c: $11 <= 00001a7b
@00003010: $12 <= 1e610000
@00003014: $12 <= 1eae0000
@00003018: $13 <= 1eae1a7b
```

@0000301c: $14 <= 1eae1985
实际输出：
118@00003000: $ 8 <= 0000007b
122@00003004: $ 9 <= 000000f6
126@00003008: $10 <= 0000007b
130@0000300c: $11 <= 00001a7b
134@00003010: $12 <= 1e610000
138@00003014: $12 <= 1eae0000
142@00003018: $13 <= 1eae1a7b
146@0000301c: $14 <= 1eae1985

2. beq 测试
Mips 代码：
```
ori $t0, $t0, 123
ori $s0, $s0, 123
beq $t0, $s0, if
ori $t1, $t1, 2468
else:
ori $t2, $t2, 2468
beq $t2, $t1, end
nop
if:
ori $t3, $t3, 123
beq $t0, $t3, else
nop
end:
```
期望输出：
@00003000: $ 8 <= 0000007b
@00003004: $16 <= 0000007b
@0000300c: $ 9 <= 000009a4
@0000301c: $11 <= 0000007b
@00003010: $10 <= 000009a4
实际输出：
118@00003000: $ 8 <= 0000007b
122@00003004: $16 <= 0000007b
134@0000300c: $ 9 <= 000009a4
138@0000301c: $11 <= 0000007b
154@00003010: $10 <= 000009a4

3. lwsw 测试
Mips 代码：
```
ori $t0, $t0, 123
sw $t0, 0($zero)
ori $t1, $t0, 246
```

```
ori $t2, $t2, 4
sw $t1, 0($t2)
lw $t3, 0($zero)
addu $t4, $t2, $t3
subu $t5, $t4, $t3
ori $t6, $t5, 135
```

期望输出：

```
@00003000: $ 8 <= 0000007b
@00003004: *00000000 <= 0000007b
@00003008: $ 9 <= 000000ff
@0000300c: $10 <= 00000004
@00003010: *00000004 <= 000000ff
@00003014: $11 <= 0000007b
@00003018: $12 <= 0000007f
@0000301c: $13 <= 00000004
@00003020: $14 <= 00000087
```

实际输出：

```
118@00003000: $ 8 <= 0000007b
118@00003004: *00000000 <= 0000007b
126@00003008: $ 9 <= 000000ff
130@0000300c: $10 <= 00000004
130@00003010: *00000004 <= 000000ff
138@00003014: $11 <= 0000007b
146@00003018: $12 <= 0000007f
150@0000301c: $13 <= 00000004
154@00003020: $14 <= 00000087
```

4. jaljr 测试

Mips 代码：

```
ori $t0, $t0, 123
jal if
ori $t1, $t1, 123
addu $s0, $t1, $t0
beq $s0, $t0, end
ori $v0, $v0, 123
subu $s1, $s0, $t0
j end
if:
ori $t2, $t2, 246
jr $31
end:
ori $t3, $t3, 246
ori $t4, $t4, 123
```

期望输出：

```
@00003000: $ 8 <= 0000007b
@00003004: $31 <= 0000300c
@00003008: $ 9 <= 0000007b
@00003020: $10 <= 000000f6
@00003028: $11 <= 000000f6
@0000300c: $16 <= 000000f6
@00003014: $ 2 <= 0000007b
@00003018: $17 <= 0000007b
@00003020: $10 <= 000000f6
@00003028: $11 <= 000000f6
@0000302c: $12 <= 0000007b
```

实际输出：

```
118@00003000: $ 8 <= 0000007b
122@00003004: $31 <= 0000300c
126@00003008: $ 9 <= 0000007b
130@00003020: $10 <= 000000f6
138@00003028: $11 <= 000000f6
142@0000300c: $16 <= 000000f6
154@00003014: $ 2 <= 0000007b
158@00003018: $17 <= 0000007b
166@00003020: $10 <= 000000f6
170@00003028: $11 <= 000000f6
174@0000302c: $12 <= 0000007b
```

5. 综合测试

Mips 代码：

```
ori $28, $0, 0x0
ori $29, $0, 0x0
ori $1, $7, 0x1010
lw $10, 0x0($0)
sw $1, 0x0($0)
lui $2, 0x8723
ori $3, $0, 0x7856
lui $4, 0x85ff
ori $5, $0, 0x1
lui $6, 0xffff
ori $7, $7, 0xffff
addu $1, $1, $2
addu $9, $1, $3
subu $8, $1, $2
subu $0, $7, $0
nop
nop
```

```
nop
nop
nop
beq $28, $17, label1
nop
j label2
nop
label1: beq $1, $2, label2
nop
ori $2, $0, 0xc
nop
nop
nop
jal label3
sw $1, 0x0($2)
j label2
addu $1, $1, $2
label3:addu $1, $1, $2
addu $1, $1, $2
addu $1, $1, $2
sw $31, 0x0($2)
lw $1, 0x0($2)
nop
nop
nop
jr $1
sw $31, 0x0($2)
label2:
nop
```

期望输出：

```
@00003000: $28 <= 00000000
@00003004: $29 <= 00000000
@00003008: $ 1 <= 00001010
@0000300c: $10 <= 00000000
@00003010: *00000000 <= 00001010
@00003014: $ 2 <= 87230000
@00003018: $ 3 <= 00007856
@0000301c: $ 4 <= 85ff0000
@00003020: $ 5 <= 00000001
@00003024: $ 6 <= ffff0000
@00003028: $ 7 <= 0000ffff
@0000302c: $ 1 <= 87231010
@00003030: $ 9 <= 87238866
@00003034: $ 8 <= 00001010
```

@00003068: $ 2 <= 0000000c
@00003078: $31 <= 00003080
@0000307c: *0000000c <= 87231010
@00003088: $ 1 <= 8723101c
@0000308c: $ 1 <= 87231028
@00003090: $ 1 <= 87231034
@00003094: *0000000c <= 00003080
@00003098: $ 1 <= 00003080
@000030ac: *0000000c <= 00003080
@00003084: $ 1 <= 0000308c
实际输出：
118@00003000: $28 <= 00000000
122@00003004: $29 <= 00000000
126@00003008: $ 1 <= 00001010
130@0000300c: $10 <= 00000000
130@00003010: *00000000 <= 00001010
138@00003014: $ 2 <= 87230000
142@00003018: $ 3 <= 00007856
146@0000301c: $ 4 <= 85ff0000
150@00003020: $ 5 <= 00000001
154@00003024: $ 6 <= ffff0000
158@00003028: $ 7 <= 0000ffff
162@0000302c: $ 1 <= 87231010
166@00003030: $ 9 <= 87238866
170@00003034: $ 8 <= 00001010
214@00003068: $ 2 <= 0000000c
230@00003078: $31 <= 00003080
230@0000307c: *0000000c <= 87231010
238@00003088: $ 1 <= 8723101c
242@0000308c: $ 1 <= 87231028
246@00003090: $ 1 <= 87231034
246@00003094: *0000000c <= 00003080
254@00003098: $ 1 <= 00003080
270@000030ac: *0000000c <= 00003080
282@00003084: $ 1 <= 0000308c

三、思考题

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。
PC 值应由 D 阶段的 NPC 根据信号判断计算下一 PC 值后传回 PC 寄存器中；如果 pc 的使能信号有效，将 NPC 值写入 PC 中，否则不写入。

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？

因为采用了“延迟槽”，在执行 jal 后也执行了 jal 后一条指令，故应写入 jal 的下下条指令地址，即 PC+8。

3. 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？
供给者为 ALU 或 DM 的话，由于其是组合逻辑部件，运算后立即转发，可能导致逻辑混乱,难以判断究竟是哪个时间进行的转发,而流水寄存器可以避免该问题。

4. 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。
会导致运算错误。

5. 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？
既可以解决问题，又不用过多添加修改。在外部对 GRF 的输出端处添加一个 MUX 选择原输出与待转发数据。

6. 为什么 0 号寄存器需要特殊处理？
0 号寄存器不得修改，值始终为 0，如果不特殊处理可能会在内部转发时出错。

7. 什么是“最新产生的数据”？
在相对最靠前的阶段产生的转发数据，如 E 级转发数据比 M 级更新。

8. 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？

9. 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？
连续对同一个寄存器存取，需要使用转发；在 beq 判断时，需要将 cmp 提前。