Verilog 流水线 CPU 设计文档

一、 CPU 设计方案综述

本 CPU 为 Verilog 实现的流水线 CPU（32 位），支持的指令集包含{addu, subu, ori, lw, sw, beq, lui, nop，j,jal，jr}，并进行了适当的指令扩展。

该流水线 CPU 采用五级流水结构，包括流水级寄存器、主要功能部件、功能控制器、冒险控制器等，采用分布式译码方式，对指令进行流水，并在各级进行译码处理。CPU 支持转发和必要的暂停，处理器顶层包含两个输入端口时钟信号 clk 和复位信号 reset。

二、 关键模块定义

1、 stageF

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
NPC[31:0]	I	输入跳转、分支指令下的下一指令地址
PC_en	I	PC 使能端，高电平有效
instr_F[31:0]	O	根据地址取到的当前指令
PC_F[31:0]	O	F 级当前指令地址

PC8_F[31:0]	O	F 级当前指令地址+8

2、 regD

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
D_en	I	D 级流水线寄存器使能信号
instr_F[31:0]	I	F 级指令
PC_F[31:0]	I	F 级当前指令地址
PC8_F[31:0]	I	当前指令地址+8
instr_D[31:0]	O	D 级指令
PC_D[31:0]	O	D 级当前指令地址
PC8_D[31:0]	O	D 级当前指令地址+8

3、 stageD

信号名	方向	描述
clk	I	时钟信号

reset	I	GRF 复位信号
A1_D[4:0]	I	地址输入信号，将对应地址寄存器的值输出至 RD1
A2_D[4:0]	I	地址输入信号，将对应地址寄存器的值输出至 RD2
A3_W[4:0]	I	地址输入信号，指定要进行写入的寄存器
PC_D[31:0]	I	D 级当前指令地址
PC_W[31:0]	I	W 级当前指令地址
imm16[15:0]	I	16 位立即数
addr26[25:0]	I	26 位地址
EXTOp	I	扩展的方式
RFWr	I	写使能信号
RFWD_W[31:0]	I	要写入寄存器的值
NPCOp[1:0]	I	控制 NPC 进行相应的操作： 00：当前为顺序执行指令，NPC 输出 PC+4 01：当前指令为 beq，作为决定是否跳转的条件之一 10：当前指令为 jal，NPC 输出 PC31..28||instr_index||02 11：当前指令为 jr，NPC 输出 GRF[rs]
MF_RD1_Sel[1:0]	I	RD1 端口转发 MUX 信号

MF_RD2_Sel[1:0]	I	RD2 端口转发 MUX 信号
RD1_D[31:0]	O	转发后的输出，输出 A1 地址对应的寄存器的值
RD2_D[31:0]	O	转发后的输出，输出 A2 地址对应的寄存器的值
imm32_D[31:0]	O	数据输出信号，输出 EXT 扩展后的 32 位立即数
Next_PC[31:0]	O	下一指令地址

4、 regE

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
E_clr	I	E 级单独的清空信号，用于暂停清空
instr_D[31:0]	I	D 级指令
PC_D[31:0]	I	D 级当前指令地址
PC8_D[31:0]	I	D 级当前指令地址+8
RD1_D[31:0]	I	数据输入信号，输出 A1 地址对应的寄存器的值
RD2_D[31:0]	I	数据输入信号，输出 A2 地址对应的寄存器的值
imm32_D[31:0]	I	数据输入信号，输出 EXT 扩展后的 32 位立即数

RD1_E[31:0]	O	数据输出信号，输出 A1 地址对应的寄存器的值
RD2_E[31:0]	O	数据输出信号，输出 A2 地址对应的寄存器的值
imm32_E[31:0]	O	数据输出信号，输出 EXT 扩展后的 32 位立即数
instr_E[31	O	E 级指令
PC_E[31:0]	O	E 级当前指令地址
PC8_E[31:0]	O	E 级当前指令地址+8

5、 stageE

信号名	方向	描述
RD1_E[31:0]	I	数据输入信号，输出 A1 地址对应的寄存器的值
RD2_E[31:0]	I	数据输入信号，输出 A2 地址对应的寄存器的值
imm32_E[31:0]	I	数据输入信号，输出 EXT 扩展后的 32 位立即数
RFWD_M[31:0]	I	M 级转发来源
RFWD_W[31:0]	I	W 级转发来源
MF_ALUA_Sel[31:0]	I	ALU.A 转发 MUX 控制信号
MF_ALUB_Sel[31:0]	I	ALU.B 转发 MUX 控制信号
ALUOp[1:0]	I	ALU 控制信号

		00：A+B 01：A-B 10：A|B 11：B
Bsel	I	ALUB 端口功能 MUX 控制信号
C_E[31:0]	O	E 级 ALU 计算结果

6、 regM

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
instr_E[31:0]	I	E 级指令
PC_E[31:0]	I	E 级当前指令地址
PC8_E[31:0]	I	E 级 PC+8
C_E[31:0]	I	E 级 ALU 计算结果
RD2_E[31:0]	I	E 级 rt 寄存器的值
C_M[31:0]	O	M 级 ALU 计算结果

RD2_M[31:0]	O	M 级 rt 寄存器的值
instr_M[31:0]	O	M 级指令
PC_M[31:0]	O	M 级当前指令地址
PC8_M[31:0]	O	M 级 PC+8

7、 stageM

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
PC_M[31:0]	I	M 级当前指令地址
DMWr	I	DM 写使能
A_M[31:0]	I	DM 写入地址
WD_M[31:0]	I	DM 写入值
RFWD_W[31:0]	I	W 级转发来源
MF_DMWD_Sel	I	DM.WD 端口转发 MUX 控制信号
WDSel_M[1:0]	I	M 级写入值的控制信号
C_M[31:0]	I	M 级 ALU 计算结果

PC8_M[31:0]	I	M 级 PC+8
D_M[31:0]	O	M 级 DM 读出值
RFWD_M[31:0]	O	M 级写入值

8、 regW

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
instr_M[31:0]	I	M 级指令
PC8_M[31:0]	I	M 级 PC+8
D_M[31:0]	I	M 级 DM 读出值
C_M[31:0]	I	M 级 ALU 计算结果
instr_W[31:0]	O	W 级指令
D_W[31:0]	O	W 级 DM 读出值
C_W[31:0]	O	W 级 ALU 计算结果
PC8_W[31:0]	O	W 级 PC+8

9、stageW

信号名	方向	描述
D_W[31:0]	I	W 级 DM 读出值
C_W[31:0]	I	W 级 ALU 计算结果
PC8_W[31:0]	I	W 级 PC+8
WDSel_W[1:0]	I	W 级写入值选择信号
RFWD_W[31:0]	O	W 级写入值

10、controller

信号名	方向	描述
instr[31:0]	I	指令
NPCOp[1:0]	O	NPC 控制信号
RFWr	O	GRF 使能信号
EXTOp	O	EXT 控制信号
ALUOp[1:0]	O	ALU 控制信号
DMWr	O	DM 使能信号

A3Sel[1:0]	O	A3 选择信号
WDSel[1:0]	O	WD 端口选择信号
BSel	O	B 选择信号
imm16[15:0]	O	16 位立即数
addr26[25:0]	O	26 位立即数
rs[4:0]	O	rs 寄存器地址
rt[4:0]	O	rt 寄存器地址
rd[4:0]	O	rd 寄存器地址
A3[4:0]	O	当前 GRF 写入地址
Tuse_rs[1:0]	O	D 级 rs 的 Tuse
Tuse_rt[1:0]	O	D 级 rt 的 Tuse
Tnew_E[1:0]	O	rs 或 rt 的 Tnew_E
Tnew_M[1:0]	O	rs 或 rt 的 Tnew_M

11、ATcontroller

信号名	方向	描述
rs_D[4:0]	I	D 级 rs 地址

rt_D[4:0]	I	D 级 rt 地址
rs_E[4:0]	I	E 级 rs 地址
rt_E[4:0]	I	E 级 rt 地址
rt_M[4:0]	I	M 级 rt 地址
A3_E[4:0]	I	E 级 A3 地址
A3_M[4:0]	I	M 级 A3 地址
A3_W[4:0]	I	M 级 A3 地址
Tuse_rs[1:0]	I	D 级 rs 的 Tuse
Tuse_rt[1:0]	I	D 级 rt 的 Tuse
Tnew_E[1:0]	I	rs 或 rtE 级 Tnew
Tnew_M[1:0]	I	rs 或 rt 的 Tnew
MF_RD1_Sel[1:0]	O	CMPA 端口的转发 MUX 的选择信号
MF_RD2_Sel[1:0]	O	CMPB 端口的转发 MUX 的选择信号
MF_ALUA_Sel[1:0]	O	ALUA 端口的转发 MUX 的选择信号
MF_ALUB_Sel[1:0]	O	ALUB 端口的转发 MUX 的选择信号
MF_DMWD_Sel	O	DMWD 端口的转发 MUX 的选择信号
PC_en	O	PC 使能信号

D_en	0	D 级使能
E_clr	0	E 级流水线寄存器清除信号

各级流水寄存器储存值如下

D	E	M	W
instr_D	instr_E	instr_M	instr_W
	RD1_D	C_M	C_W
	RD2_D	RD2_M	D_W
	imm32_D		
PC_D	PC_E	PC_M	
PC8_D	PC8_E	PC8_M	PC8_W
	(A3_E)	(A3_M)	(A3_W)
	(WDSel_E)	(WDSel_M)	(WDSel_W)

三、 转发与暂停处理

转发

转发点共有五个：CMP.A,CMP.B,ALU.A,ALU.B,DM.WD

最后一级 W 级对 GRF 采用内部转发

对其他为 RFWD_W:C_W,D_W,PC8_W

M 级为 RFWD_M：PC8_M,C_E

E 级为 PC8_E

转发点	0	1	2
GRF.RD1	RD1_D_raw	PC8_E	RFWD_M
GRF.RD2	RD2_D_raw	PC8_E	RFWD_M
ALU.A	RD1_E	RFWD_M	RFWD_W
ALU.B	RD2_E	RFWD_M	RFWD_W
DM.WD	RD2_M	RFWD_W	

转发的判断条件是：当前使用的寄存器地址与转发来源寄存器地址相同，且不为 0，则按照优先等级进行转发。

如果转发来源不进行写入的话，令转发来源寄存器地址为 0（从而实现不转发）。

暂停

在 D 级进行 Tuse 和各级 Tnew 的比较如果 Tuse<Tnew，则进行暂停：

PC_en 无效，保持原值不变，D 级流水线寄存器使能信号无效，保持原值不变。E_clr 有效，清除原流水线寄存器的值，实现暂停。

T 矩阵：

指令	Tuse_rs	Tuse_rt	Tnew_E	Tnew_M
addu	01	01	01	00
subu	01	01	01	00
ori	01	11	01	00
lui	11	11	01	00
lw	01	11	10	01
sw	01	10	00	00
beq	00	00	00	00
jal	11	11	00	00
j	11	11	00	00
jr	00	11	00	00
nop	不处理即可			

四、 控制信号取值表

指令	NPCOp[1:0]	RFWr	EXTOp	ALUOp[1:0]	DMWr	A3Sel[1:0]	WDSel[1:0]	BSel

addu (000000/100001)	00	1	x	00	0	00	00	0
subu (000000/100011)	00	1	x	01	0	00	00	0
ori (001101)	00	1	0	10	0	01	00	1
lui (001111)	00	1	0	11	0	01	00	1
lw (100011)	00	1	1	00	0	01	01	1
sw (101011)	00	0	1	00	1	xx	xx	1
beq (000100)	01	0	x	01	0	xx	xx	0
jal (000011)	10	1	x	xx	0	10	10	x
j	10	0	x	xx	0	xx	xx	x

（000010）								
jr (000000/001000)	11	0	x	xx	0	xx	xx	x

五、 测试方案

```
ori $1,$1,1

ori $2,$2,4

subu $3,$1,$2

addu $1,$3,$2

lui $1,1234

sw $1,0($zero)

lw $4,0($zero)

addu $4,$4,$1

ori $5,$5,5

ori $6,$6,5

beq $5,$6,beqt1

addu $2,$2,$1
```

```
subu $2,$2,$1

beqt1:

addu $5,$5,$1

beq $5,$6,beqt2

addu $2,$2,$1

subu $2,$2,$1



beqt2:

j jt

addu $5,$5,$1

subu $5,$5,$1



jt:

jal jalt

addu $5,$5,$1

subu $5,$5,$1

addu $5,$5,$1

subu $5,$5,$1
```

```
    jalt:

    ori $7,$7,4

    addu $31,$31,$7

    sw $31,0($zero)

    lw $31,0($zero)

    jr $ra

    addu $1,$1,$1
```

预期结果见 mars：

结果：

```
55@00003000: $ 1 <= 00000001

65@00003004: $ 2 <= 00000004
75@00003008: $ 3 <= fffffffd
85@0000300c: $ 1 <= 00000001
95@00003010: $ 1 <= 04d20000
95@00003014: *00000000 <= 04d20000
115@00003018: $ 4 <= 04d20000
135@0000301c: $ 4 <= 09a40000
145@00003020: $ 5 <= 00000005
155@00003024: $ 6 <= 00000005
185@0000302c: $ 2 <= 04d20004
195@00003034: $ 5 <= 04d20005
225@0000303c: $ 2 <= 09a40004
235@00003040: $ 2 <= 04d20004
255@00003048: $ 5 <= 09a40005
265@00003050: $31 <= 00003058
275@00003054: $ 5 <= 0e760005
285@00003064: $ 7 <= 00000004
295@00003068: $31 <= 0000305c
295@0000306c: *00000000 <= 0000305c
```

```
315@00003070: $31 <= 0000305c
355@00003078: $ 1 <= 09a40000
365@0000305c: $ 5 <= 181a0005
375@00003060: $ 5 <= 0e760005
385@00003064: $ 7 <= 00000004
395@00003068: $31 <= 00003060
395@0000306c: *00000000 <= 00003060
415@00003070: $31 <= 00003060
455@00003078: $ 1 <= 13480000
465@00003060: $ 5 <= fb2e0005
475@00003064: $ 7 <= 00000004
485@00003068: $31 <= 00003064
485@0000306c: *00000000 <= 00003064
505@00003070: $31 <= 00003064
545@00003078: $ 1 <= 26900000
555@00003064: $ 7 <= 00000004
565@00003068: $31 <= 00003068
565@0000306c: *00000000 <= 00003068
585@00003070: $31 <= 00003068
625@00003078: $ 1 <= 4d200000
635@00003068: $31 <= 0000306c
635@0000306c: *00000000 <= 0000306c
655@00003070: $31 <= 0000306c
695@00003078: $ 1 <= 9a400000
695@0000306c: *00000000 <= 0000306c
715@00003070: $31 <= 0000306c
755@00003078: $ 1 <= 34800000
755@0000306c: *00000000 <= 0000306c
775@00003070: $31 <= 0000306c
815@00003078: $ 1 <= 69000000
815@0000306c: *00000000 <= 0000306c
835@00003070: $31 <= 0000306c
875@00003078: $ 1 <= d2000000
875@0000306c: *00000000 <= 0000306c
895@00003070: $31 <= 0000306c
935@00003078: $ 1 <= a4000000
935@0000306c: *00000000 <= 0000306c
955@00003070: $31 <= 0000306c
995@00003078: $ 1 <= 48000000

995@0000306c: *00000000 <= 0000306c
```

六、 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 根据 NPCOp 进行更新，该控制信号来源于 D 级。

顺序执行的指令 PC 更新为 PC_F+4；分支类需要进行判断，如果条件满足更新为 PC_D+4+offset，否则更新为 PC_F+4；对于 j/jal 按照绝对地址进行更新；对于 jr 按照 rs 寄存器中的值进行更新。

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？

实验支持延迟槽，会执行 jal 的下一条指令，应该将下下条指令的地址存入 rs 寄存器。

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？

如果由 ALU 或者 DM 等部件提供数据进行转发，相比流水级寄存器提供数据进行转发，要经过的组合逻辑更多，需要的时间更多。

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。

会产生数据冒险。

addu $1,$1,$2

subu $3,$1,$2

addu 到 M 级的时候，subu 在 E 级，但此时 addu 还没有对$1 进行写入操作，如果采用原始数据，$1 就不是最新值，于是产生了数据冒险。

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？

确保读出的数是 W 级最新写入的数，避免数据冒险。

把要写入的数据与读出的数据接入 MUX，根据两寄存器的号数关系进行选择（0 号进行特判），相等就选择写入值，反之原值。

Thinking 3：为什么 0 号寄存器需要特殊处理？

如果当前读 0 号寄存器，之前写入 0 号寄存器，不对 0 号进行特判就会产生从 0 号寄存器中读出非零数的现象。

Thinking 4：什么是"最新产生的数据"？

距离该条指令最近的指令对 GRF 写入的数据为最新产生的数据。

5、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 WE 无效，则将 A 置为 0，从而转发 MUX 中选择前一级流水线寄存器中的值，完成了 WE 无效的情况下不转发的行为。

6、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？

写寄存器的指令：addu，subu,ori,lui,lw,jal

读寄存器的指令:addu,subu,ori,lw,sw,beq,jr

以上两种指令类型相互间会发生冲突，通过暂停和转发解决

在 D 级比较 Tuse 和 Tnew 的关系，如果 Tuse<Tnew_? 且读出寄存器地址和写入寄存器地址相同且不为 0 则暂停，PC_en,D_en 无效,E_clr 有效。（如果不写入则将写入地址置为 0，转发 MUX 选择为原值，相当于没转发）

否则可以通过转发解决，如果读取地址和写入地址相同，则按“近”转发，使用、传递最新值。

测试样例见上面代码。

7、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。

手动测试样例。

写寄存器的指令：addu，subu,ori,lui,lw,jal

读寄存器的指令:addu,subu,ori,lw,sw,beq,jr

不读不写的指令：nop（不用管）

只有可能在这两类之间发生冲突，并且要求读取和写入寄存器的地址相同，共有 6*7 种组合，即可覆盖。