P6 设计文档

——Verilog 流水线（50 指令）

一、指令集

add	addu	sub	subu	sll	srl	sra	sllv	srlv	srav
and	or	xor	nor	slt	sltu	addi	addiu	andi	ori
xori	lui	slti	sltiu	beq	bne	blez	bgtz	bltz	bgez
j	jal	jalr	jr	lb	lbu	lh	lhu	lw	sb
sh	sw	mult	multu	div	divu	mfhi	mflo	mthi	mtlo

注：所有运算类指令均不考虑因溢出而产生的异常。

二、模块规格

1. PC：指令地址寄存器（F 级部件）

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	同步复位信号
en		I	写使能信号
PCI	[31:0]	I	下一条指令的地址
PC	[31:0]	O	当前指令的地址

模块功能说明如下：

序号	功能名	描述
1	读指令地址	通过 PC 端口输出当前指令地址
2	写指令地址	时钟上升沿到来时若 en 信号为 1，则更新指令地址
3	同步复位	时钟上升沿到来时若 reset 信号为 1，则复位指令地址至初始状态 0x00003000

2. IM：指令存储器（F 级部件）

模块端口定义如下：

信号名	位数	方向	描述
IMI	[31:0]	I	当前指令地址信号
IM	[31:0]	O	当前指令信号

模块功能说明如下：

序号	功能名	描述
1	读指令	读出指令存储器中 IMI 地址对应的指令并通过 IM 端口输出
2	初始化写指令	初始时向指令存储器中读入所有指令

3. GRF： 寄存器堆（D 级部件，W 级部件）

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	同步复位信号
PC	[31:0]	I	当前指令的地址信号
A1	[4:0]	I	第一个读寄存器地址输入信号
A2	[4:0]	I	第二个读寄存器地址输入信号
A3	[4:0]	I	写寄存器地址输入信号
WD	[31:0]	I	写入数据信号
WE		I	写使能信号
RD1	[31:0]	O	A1 所对应的寄存器的数据信号
RD2	[31:0]	O	A2 所对应的寄存器的数据信号

模块功能说明如下：

序号	功能名	描述
1	读数据	读出 A1，A2 地址对应寄存器数据并通过 RD1，RD2 端口输出
2	写数据	时钟上升沿到来时，若 WE 信号为 1，则向 A3 地址对应寄存器写入数据 WD（0 号寄存器不能被写入）并输出一条与 PC 有关的信息
3	同步复位	时钟上升沿到来时，若 reset 信号为 1，则复位所有寄存器至初始状态 0x00000000

4. EXT：立即数拓展模块（D 级部件）

模块端口定义如下：

信号名	位数	方向	描述
EXTIMM	[15:0]	I	参与拓展的立即数信号
EXTOp	[1:0]	I	拓展方式的选择信号
EXT	[31:0]	O	拓展完成后的数据信号

模块功能说明如下：

序号	功能名	描述
1	符号拓展	若 EXTOp==00，将立即数加载至输出信号低位并用其最高位填充输出信号的高 16 位
2	零拓展	若 EXTOp==01，将立即数加载至输出信号低位并用 0 填充输出信号的高 16 位
3	低位零拓展	若 EXTOp==10，将立即数加载至输出信号高位并用 0 填充输出信号的低 16 位

5. NPC：指令地址计算器（D 级部件）

模块端口定义如下：

信号名	位数	方向	描述
ADD4	[31:0]	I	PC+4 信号
PC4_D	[31:0]	I	F/D 级流水线寄存器中的 PC4 信号
imm_index	[25:0]	I	计算跳转地址的 imm/index 数据信号
M_RD1_D	[31:0]	I	D 级读取 rs 寄存器时的转发信号
CMP		I	跳转时特殊条件的判断信号
Is_B		I	b 类指令的判断信号（beq/bne/blez/bgtz/bltz/bgez）
Is_J		I	j 类指令的判断信号（j / jal）
PCOp	[1:0]	I	计算下一条指令地址的选择信号
TURE_NPC	[31:0]	O	下一条指令地址

模块功能说明如下：

序号	功能名	描述
1	计算指令地址	若 PCOp==00，输出 PC+4 信号 若 PCOp==01，输出 M_RD1_D 信号 若 PCOp==10 且 is_B==1 且 CMP==1，则计算 PC4_D+imm 拓展并输出 若 PCOp==10 且 is_J==1，则计算 PC4_D 与 index 拓展信号并输出 若 PCOp==10 且不属于以上情况，则输出 PC4_D+4 信号

6. CMP：数据比较器（D 级部件）

模块端口定义如下：

信号名	位数	方向	描述
D1	[31:0]	I	参与比较的第一个数据信号
D2	[31:0]	I	参与比较的第二个数据信号
CMPOp	[2:0]	I	比较方式的选择信号
CMP		O	比较结果信号

模块功能说明如下：

序号	功能名	描述
1	比较相等	若 CMPOp==000，二数据信号相等，则输出 1，否则输出 0
2	比较不等	若 CMPOp==001，若二数据信号不等，则输出 1，否则输出 0
3	比较小于 0	若 CMPOp==010，若 D1 小于 0，则输出 1，否则输出 0
4	比较小于等于 0	若 CMPOp==011，若 D1 小于等于 0，则输出 1，否则输出 0
5	比较大于 0	若 CMPOp==100，若 D1 大于 0，则输出 1，否则输出 0
6	比较大于等于 0	若 CMPOp==101，若 D1 大于等于 0，则输出 1，否则输出 0

7. ALU：计算模块（E 级部件）

模块端口定义如下：

信号名	位数	方向	描述
A	[31:0]	I	参与运算的第一个数据信号
B	[31:0]	I	参与运算的第二个数据信号
ALUOp	[3:0]	I	运算方式的选择信号
ALU	[31:0]	O	运算结果的数据信号

模块功能说明如下：

序号	功能名	描述
1	按位与运算	若 ALUOp==0000，计算 A&B 并通过 ALU 端口输出
2	按位或运算	若 ALUOp==0001，计算 A|B 并通过 ALU 端口输出
3	不支持溢出加法	若 ALUOp==0010，计算 A+B 并通过 ALU 端口输出
4	不支持溢出减法	若 ALUOp==0011，计算 A-B 并通过 ALU 端口输出
5	逻辑左移	若 ALUOp==0100，计算 B 逻辑左移 A 并通过 ALU 端口输出
6	逻辑右移	若 ALUOp==0101，计算 B 逻辑右移 A 并通过 ALU 端口输出
7	算术右移	若 ALUOp==0110，计算 B 算术右移 A 并通过 ALU 端口输出
8	异或	若 ALUOp==0111，计算 A xor B 并通过 ALU 端口输出
9	或非	若 ALUOp==1000，计算 A nor B 并通过 ALU 端口输出
10	小于置 1	若 ALUOp==1001，A < B 则输出 1，否则输出 0
11	无符号小于置 1	若 ALUOp==1010，A 无符号 < B 则输出 1，否则输出 0

8. XALU：乘除法计算模块（E 级部件）

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	同步复位信号
XALUA	[31:0]	I	参与运算的第一个数据信号
XALUB	[31:0]	I	参与运算的第二个数据信号
XALUOp	[2:0]	I	运算方式的选择信号
START		I	乘除法计算的开始信号
BUSY		O	正在运算乘除法的判断信号
HIGH	[31:0]	O	HI 寄存器的值
LOW	[31:0]	O	LO 寄存器的值

模块功能说明如下：

序号	功能名	描述
1	同步复位	时钟上升沿到来时，若 reset 信号为 1，则复位 HI 和 LO 至 0x00000000，复位 BUSY 至 0
2	符号乘法	时钟上升沿到来时，若 START 信号为 1 且 XALUOp==000，则计算 A*B 并在 5 个时钟周期后通过 HIGH 和 LOW 输出
3	无符号乘法	时钟上升沿到来时，若 START 信号为 1 且 XALUOp==001，则计算无符号 A*B 并在 5 个时钟周期后通过 HIGH 和 LOW 输出
4	符号除法	时钟上升沿到来时，若 START 信号为 1 且 XALUOp==010，则计算 A/B 并在 10 个时钟周期后通过 HIGH 和 LOW 输出
5	无符号除法	时钟上升沿到来时，若 START 信号为 1 且 XALUOp==011，则计算无符号 A/B 并在 10 个时钟周期后通过 HIGH 和 LOW 输出
6	写 HI 寄存器	时钟上升沿到来时，若 XALUOp==100，则将 XALUA 值写入 HI
8	写 LO 寄存器	时钟上升沿到来时，若 XALUOp==100，则将 XALUA 值写入 LO

9. DM：数据存储器（M 级部件）

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	同步复位信号
PC	[31:0]	I	当前指令的地址信号
DMA	[31:0]	I	存取的地址信号
DMD	[31:0]	I	存取的数据信号
DMWE		I	存数据使能信号
DMOp	[1:0]	I	存数据方式的选择信号
DM	[31:0]	O	取出的数据信号

模块功能说明如下：

序号	功能名	描述
1	存数据	时钟上升沿到来时，若 DMWE 信号为 1，则根据 DMOp 的值向数据存储器 DMA 对应地址的某一段中写入 DMD 数据并输出一条与 PC 有关的信息
2	取数据	将数据存储器 DMA 地址对应的数据通过 DM 端口输出
3	同步复位	时钟上升沿到来时，若 reset 信号为 1，则复位数据存储器至初始状态 0x00000000

10. DMEXT：数据存储器拓展器（M 级部件）

模块端口定义如下：

信号名	位数	方向	描述
DM	[31:0]	I	所需要拓展的数据信号
DMA	[31:0]	I	读取数据存储器的地址信号
DMEXTOp	[2:0]	I	拓展方式选择信号
DMEXT	[31:0]	O	拓展后的数据信号

模块功能说明如下：

序号	功能名	描述
1	拓展数据	根据 DMA 末两位值与 DMEXTOp 值对 DM 数据进行拓展后通过 DMEXT 端口输出

三、流水线寄存器规格

1. FDreg：F/D 级流水线寄存器

模块端口定义如下：

控制信号	输入信号	输出信号
clk	PC	IR_D
reset	IM	PC_D
en		PC4_D
		PC8_D

2. DEreg：D/E 级流水线寄存器

模块端口定义如下：

控制信号	输入信号	输出信号
clk	IR_D	IR_E
reset	PC_D	PC_E
en	PC4_D	PC4_E
	PC8_D	PC8_E
	RD1	RD1_E
	RD2	RD2_E
	EXT	EXT_E
	GRFWE	GRFWE_E
	Tnew	Tnew_E

3. EMreg：E/M 级流水线寄存器

模块端口定义如下：

控制信号	输入信号	输出信号
clk	IR_E	IR_M
reset	PC_E	PC_M
en	PC4_E	PC4_M
	PC8_E	PC8_M
	ALU	ALU_M
	RD2_E	RD2_M
	GRFWE_E	GRFWE_M
	Tnew_E	Tnew_M

4. MWreg：M/W 级流水线寄存器

模块端口定义如下：

控制信号	输入信号	输出信号
clk	IR_M	IR_W
reset	PC_M	PC_W
en	PC4_M	PC4_W
	PC8_M	PC8_W
	ALU_M	ALU_W
	DM	DM_W
	GRFWE_M	GRFWE_W

四、控制器设计

1. 控制信号说明

	0	1	2	3	4	5	6	7	8	9	10
GRFA3 MUXOp	rt	rd	31								
ALUB MUXOp	RD2	EXT									
GRFWE	不写寄存器	写寄存器									
DMWE	不写DM	写DM									
GRFWD MUXOp	ALU	DM	PC8	HI	LO						
EXTOp	符号拓展	零拓展	加载高位								
is B		用imm算pc									
ALUOp	&	|	+	-	逻辑左移	逻辑右移	算术右移	xor	nor	小于置1	小于u置1
is J		用index算pc									
PCOp	ADD4	RD1	NPC								
Tuse rs	在npc用	在alu用	在dm用								
Tuse rt	在npc用	在alu用	在dm用								
Tnew		在npc产生	在alu产生	在dm产生							
A3 Op	rd	rt	31								
WD Op	pc8	alu	dm	HI	LO						
ALUA MUXOp	RD1	shamt									
CMPOp	==	!=	<0	<=0	>0	>=0					
DMOp	word	halfword	byte								
DMEXTOp	word	halfword	halfwordu	byte	byteu						
XALUOp	*	*u	/	/u	mthi	mtlo					
START		用XALU计算									

2. 指令与控制信号真值表

	add	addu	sub	subu	sll	srl	sra	sllv	srlv	srav	and	or	xor	nor	slt	sltu
	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
	100000	100001	100010	100011	000000	000010	000011	000100	000110	000111	100100	100101	100110	100111	101010	101011
GRFA3 MUXOp	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ALUB MUXOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRFWE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
DMWE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRFWD MUXOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EXTOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
is B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ALUOp	2	2	3	3	4	5	6	4	5	6	0	1	7	8	9	10
is_T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PCOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Tuse_rs	1	1	1	1	5	5	5	1	1	1	1	1	1	1	1	1
Tuse_rt	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Tnew	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
A3 Op	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
WD Op	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ALUA MUXOp	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
DMOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DMEXTOp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	addi	addiu	andi	ori	xori	lui	slti	sltiu		lb	lbu	lh	lhu	lw
	001000	001001	001100	001101	001110	001111	001010	001011		100000	100100	100001	100101	100011
GRFA3 MUXOp	0	0	0	0	0	0	0	0		0	0	0	0	0
ALUB MUXOp	1	1	1	1	1	1	1	1		1	1	1	1	1
GRFWE	1	1	1	1	1	1	1	1		1	1	1	1	1
DMWE	0	0	0	0	0	0	0	0		0	0	0	0	0
GRFWD MUXOp	0	0	0	0	0	0	0	0		1	1	1	1	1
EXTOp	0	0	1	1	1	2	0	0		0	0	0	0	0
is B	0	0	0	0	0	0	0	0		0	0	0	0	0
ALUOp	2	2	0	1	7	2	9	10		2	2	2	2	2
is_T	0	0	0	0	0	0	0	0		0	0	0	0	0
PCOp	0	0	0	0	0	0	0	0		0	0	0	0	0
Tuse_rs	1	1	1	1	1	1	1	1		1	1	1	1	1
Tuse_rt	5	5	5	5	5	5	5	5		5	5	5	5	5
Tnew	2	2	2	2	2	2	2	2		3	3	3	3	3
A3 Op	1	1	1	1	1	1	1	1		1	1	1	1	1
WD Op	1	1	1	1	1	1	1	1		2	2	2	2	2
ALUA MUXOp	0	0	0	0	0	0	0	0		0	0	0	0	0
DMOp	0	0	0	0	0	0	0	0		0	0	0	0	0
DMEXTOp	0	0	0	0	0	0	0	0		3	4	1	2	0

	beq	bne	blez	bgtz	bltz	bgez	j	jal	jalr	jr		sb	sh	sw
	000100	000101	000110	000111	000001	000001	000010	000011	000000	000000		101000	101001	101011
rt					00000	00001			001001	001000				
GRFA3 MUXOp	0	0	0	0	0	0	0	2	1	0		0	0	0
ALUB MUXOp	0	0	0	0	0	0	0	0	0	0		1	1	1
GRFWE	0	0	0	0	0	0	0	1	1	0		0	0	0
DMWE	0	0	0	0	0	0	0	0	0	0		1	1	1
GRFWD MUXOp	0	0	0	0	0	0	0	2	2	0		0	0	0
EXTOp	0	0	0	0	0	0	0	0	0	0		0	0	0
is B	1	1	1	1	1	1	0	0	0	0		0	0	0
ALUOp	0	0	0	0	0	0	0	0	0	0		2	2	2
is_T	0	0	0	0	0	0	1	1	0	0		0	0	0
PCOp	2	2	2	2	2	2	2	2	1	1		0	0	0
Tuse_rs	0	0	0	0	0	0	5	5	0	0		1	1	1
Tuse_rt	0	0	5	5	5	5	5	5	5	5		2	2	2
Tnew	0	0	0	0	0	0	0	0	0	0		0	0	0
A3 Op	0	0	0	0	0	0	0	2	0	0		0	0	0
WD Op	0	0	0	0	0	0	0	0	0	0		0	0	0
ALUA MUXOp	0	0	0	0	0	0	0	0	0	0		0	0	0
CMPOp	0	1	3	4	2	5	0	0	0	0		0	0	0
DMOp	0	0	0	0	0	0	0	0	0	0		2	1	0
DMEXTOp	0	0	0	0	0	0	0	0	0	0		0	0	0

	mult	multu	div	divu		mfhi	mflo	mthi	mtlo
	000000	000000	000000	000000		000000	000000	000000	000000
	011000	011001	011010	011011		010000	010010	010001	010011
GRFA3 MUXOp	0	0	0	0		1	1	0	0
ALUB MUXOp	0	0	0	0		0	0	0	0
GRFWE	0	0	0	0		1	1	0	0
DMWE	0	0	0	0		0	0	0	0
GRFWD MUXOp	0	0	0	0		3	4	0	0
EXTOp	0	0	0	0		0	0	0	0
is B	0	0	0	0		0	0	0	0
ALUOp	0	0	0	0		0	0	0	0
is_T	0	0	0	0		0	0	0	0
PCOp	0	0	0	0		0	0	0	0
Tuse_rs	1	1	1	1		0	0	1	1
Tuse_rt	1	1	1	1		0	0	5	5
Tnew	2	2	2	2		3	3	2	2
A3 Op	0	0	0	0		0	0	0	0
WD Op	0	0	0	0		3	4	0	0
ALUA MUXop	0	0	0	0		0	0	0	0
CMPOp	0	0	0	0		0	0	0	0
DMOp	0	0	0	0		0	0	0	0
DMEXTOp	0	0	0	0		0	0	0	0
XALUOp	0	1	2	3		0	0	4	5
START	1	1	1	1		0	0	0	0

五、冒险处理策略

在 P6 中共有 50 条指令，仍使用 P5 中所采取的分类表格法将十分繁杂且易于出错，因此，本设计采取暴力转发的方法。其主要实现方式为，在每一级流水线寄存器中都存入当前指令的 Tnew（距离产生结果所剩的时钟周期数）以及 A3（所将要写入的 5 位寄存器地址）和 WD（所将要写入的数据），同时 D 级控制器产生 D 级指令的 Tuse_rs（距需要使用 RD1 所剩的时钟周期数）和 Tuse_rt（距需要使用 RD2 所剩的时钟周期数）。每一次时钟上升沿到来时，若检测到 D 级指令需要用到的 RD1 或 RD2 仍未产生，则产生一次暂停，若检测到其他已产生结果的数据冒险，则进行转发。这种设计保证了每一条指令所使用的值均是正确的。此外，乘除法的暂停也由 BUSY 信号决定。

有关模块：

1. Analysis_MUX（A3 及 WD 分析模块）

功能：根据每一级控制器产生的 A3_MUX 和 WD_MUX 信号产生每一级指令的 A3 与 WD

2. StopUnit（暂停分析与产生模块）

功能：检测到 D 级指令所需数据不能及时得出时进行一次暂停，D 级为乘除法相关指令且 XALU 忙碌时进行一次暂停

3. ForwardUnit（转发来源分析与产生模块）

功能：在 D 级 rs、D 级 rt、E 级 rs、E 级 rt、M 级 rt 五个位点检测到数据冒险时根据各个流水线寄存器的 WD 和 A3 值进行转发

六、测试样例

首先对指令本身行为进行测试，测试程序与期望结果如下

1. add addu sub subu and or xor nor slt sltu

```
lui $1, 0x1234                  @00003000: $ 1 <= 12340000
ori $1, $1, 0x8765              @00003004: $ 1 <= 12348765
ori $2, $0, 0x5678              @00003008: $ 2 <= 00005678
add $4, $1, $2                  @0000300c: $ 4 <= 1234dddd
addu $5, $1, $2                 @00003010: $ 5 <= 1234dddd
sub $6, $1, $2                  @00003014: $ 6 <= 123430ed
subu $7, $1, $2                 @00003018: $ 7 <= 123430ed
and $8, $1, $2                  @0000301c: $ 8 <= 00000660
or $9, $1, $2                   @00003020: $ 9 <= 1234d77d
xor $10, $1, $2                 @00003024: $10 <= 1234d11d
nor $11, $1, $2                 @00003028: $11 <= edcb2882
slt $12, $1, $2                 @0000302c: $12 <= 00000000
sltu $13, $1, $2                @00003030: $13 <= 00000000
```

2. Sll srl sra sllv srlv srav

```
lui $1, 0x8234
ori $1, $1, 0x8765
ori $2, $0, 10
sll $3, $1, 10
srl $4, $1, 10
sra $5, $1, 10
sllv $6, $1, $2
srlv $6, $1, $2
srav $6, $1, $2
```

```
@00003000: $ 1 <= 82340000
@00003004: $ 1 <= 82348765
@00003008: $ 2 <= 0000000a
@0000300c: $ 3 <= d21d9400
@00003010: $ 4 <= 00208d21
@00003014: $ 5 <= ffe08d21
@00003018: $ 6 <= d21d9400
@0000301c: $ 6 <= 00208d21
@00003020: $ 6 <= ffe08d21
```

3. Addi addiu andi ori xori lui slti sltiu

```
lui $1, 0x8234
ori $1, $1, 0x8765
addi $3, $1, 0x5678
addiu $4, $1, 0x5678
andi $5, $1, 0x5678
ori $6, $1, 0x5678
xori $7, $1, 0x5678
slti $8, $1, 0x5678
sltiu $9, $1, 0x5678
```

```
@00003000: $ 1 <= 82340000
@00003004: $ 1 <= 82348765
@00003008: $ 3 <= 8234dddd
@0000300c: $ 4 <= 8234dddd
@00003010: $ 5 <= 00000660
@00003014: $ 6 <= 8234d77d
@00003018: $ 7 <= 8234d11d
@0000301c: $ 8 <= 00000001
@00003020: $ 9 <= 00000000
```

4. Lb lbu lh lhu lw

```
lui $1, 0x8765
ori $1, $1, 0x4321
sw $1, 0($0)
lw $2, 0($0)
lb $3, 0($0)
lb $4, 1($0)
lb $5, 2($0)
lb $6, 3($0)
lbu $7, 0($0)
lbu $8, 1($0)
lbu $9, 2($0)
lbu $10, 3($0)
lh $11, 0($0)
lh $12, 2($0)
lhu $11, 0($0)
lhu $12, 2($0)
```

```
@00003000: $ 1 <= 87650000
@00003004: $ 1 <= 87654321
@00003008: *00000000 <= 87654321
@0000300c: $ 2 <= 87654321
@00003010: $ 3 <= 00000021
@00003014: $ 4 <= 00000043
@00003018: $ 5 <= 00000065
@0000301c: $ 6 <= ffffff87
@00003020: $ 7 <= 00000021
@00003024: $ 8 <= 00000043
@00003028: $ 9 <= 00000065
@0000302c: $10 <= 00000087
@00003030: $11 <= 00004321
@00003034: $12 <= ffff8765
@00003038: $11 <= 00004321
@0000303c: $12 <= 00008765
```

5. Sb sh sw

```
lui $1, 0x8765
ori $1, $1, 0x4321
sw $1, 0($0)
sh $1, 4($0)
sh $1, 6($0)
sb $1, 8($0)
sb $1, 9($0)
sb $1, 10($0)
sb $1, 11($0)
```

```
145@00003000: $ 1 <= 87650000
155@00003004: $ 1 <= 87654321
155@00003008: *00000000 <= 87654321
165@0000300c: *00000004 <= 00004321
175@00003010: *00000004 <= 43214321
185@00003014: *00000008 <= 00000021
195@00003018: *00000008 <= 00002121
205@0000301c: *00000008 <= 00212121
215@00003020: *00000008 <= 21212121
```

6. Beq bne blez bgtz bltz bgez

```
lui $9, 0xffff
ori $9, $9, 0xffff
ori $10, $0, 1
ori $11, $11, 1
ori $1, $0, 1
ori $2, $0, 1
beq $1, $2, b1
addu $10, $10, $10
addu $11, $11, $11
b1:
bne $10, $11, b2
addu $10, $10, $10
addu $11, $11, $11
b2:
blez $9, b3
addu $10, $10, $10
addu $11, $11, $11
b3:
bgtz $10, b4
addu $10, $10, $10
addu $11, $11, $11
b4:
```

```
@00003000: $ 9 <= ffff0000
@00003004: $ 9 <= ffffffff
@00003008: $10 <= 00000001
@0000300c: $11 <= 00000001
@00003010: $ 1 <= 00000001
@00003014: $ 2 <= 00000001
@0000301c: $10 <= 00000002
@00003028: $10 <= 00000004
@00003034: $10 <= 00000008
@00003040: $10 <= 00000010
@0000304c: $10 <= 00000020
@00003058: $10 <= 00000040
@00003060: $10 <= 00000080
@00003064: $11 <= 00000002
```

```
bltz $9, b5
addu $10, $10, $10
addu $11, $11, $11
b5:
bgez $10, b6
addu $10, $10, $10
addu $11, $11, $11
b6:
addu $10, $10, $10
addu $11, $11, $11
nop
```

7. J jal jalr jr

```
ori $9, $0, 16
ori $1, $0, 1
ori $10, $0, 1
ori $11, $0, 1
j j1
addu $10, $10, $10
addu $11, $11, $11
j1:
beq $10, $9, end
nop
jal j2
addu $10, $10, $10
addu $11, $11, $11
j2:
beq $10, $9, end
nop
jalr $30, $ra
addu $10, $10, $10
addu $11, $11, $11
j3:
jr $ra
addu $10, $10, $10
addu $11, $11, $11
end:
nop
```

```
@00003000: $ 9 <= 00000010
@00003004: $ 1 <= 00000001
@00003008: $10 <= 00000001
@0000300c: $11 <= 00000001
@00003014: $10 <= 00000002
@00003024: $31 <= 0000302c
@00003028: $10 <= 00000004
@00003038: $30 <= 00003040
@0000303c: $10 <= 00000008
@0000302c: $11 <= 00000002
@00003038: $30 <= 00003040
@0000303c: $10 <= 00000010
@0000302c: $11 <= 00000004
```

8. Mult multu div divu（测试此类指令时打开写入 HILO 时的输出信息）

```
ori $1, $0, 2
ori $2, , $0, 3
mult $1, $2
nop
```

```
145@00003000: $ 1 <= 00000002
155@00003004: $ 2 <= 00000003
195@: $         32 <= 00000000
195@: $         33 <= 00000006
```

9. Mfhi mflo mthi mtlo（测试此类指令时打开写入 HILO 时的输出信息）

```
ori $10, $0, 0xffff
mthi $10
mtlo $10
ori $1, $0, 2
ori $2, , $0, 3
mult $1, $2
nop
mflo $11
mfhi $12
```

```
135@: $         32 <= 0000ffff
145@00003000: $10 <= 0000ffff
145@: $         33 <= 0000ffff
175@0000300c: $ 1 <= 00000002
185@00003010: $ 2 <= 00000003
225@: $         32 <= 00000000
225@: $         33 <= 00000006
265@0000301c: $11 <= 00000006
275@00003020: $12 <= 00000000
```

确认指令本身行为无误后，进行暂停与转发相关的测试，由于使用暴力转发的方法且 P5 所使用指令集已确保正确，因此在 P6 中加入 calr、cali、b、j、load、store 类型指令时均可认为暂停与转发正确，因此测试的重点为乘除法指令之间的冲突与乘除法指令和其他类型指令之间的冲突。

乘除法指令之间的冲突：（需要观察波形图以确定暂停周期数）

```
ori $1, $0, 2
ori $2, $0, 3
mult $1, $2
ori $3, $0, 4
ori $4, $0, 5
mult $3, $4
nop
```

```
145@00003000: $ 1 <= 00000002
155@00003004: $ 2 <= 00000003
175@0000300c: $ 3 <= 00000004
185@00003010: $ 4 <= 00000005
195@: $         32 <= 00000000
195@: $         33 <= 00000006
265@: $         32 <= 00000000
265@: $         33 <= 00000014
```

```
ori $1, $0, 2
ori $2, $0, 3
mult $1, $2
ori $3, $0, 4
ori $4, $0, 5
mult $3, $4
mflo $5
nop
```

```
145@00003000: $ 1 <= 00000002
155@00003004: $ 2 <= 00000003
175@0000300c: $ 3 <= 00000004
185@00003010: $ 4 <= 00000005
195@: $         32 <= 00000000
195@: $         33 <= 00000006
265@: $         32 <= 00000000
265@: $         33 <= 00000014
305@00003018: $ 5 <= 00000014
```

乘除法指令与其他指令之间的冲突：

至少应在每类指令中均选取一个进行冲突测试，以下为例：

```
ori $1, $0, 2
ori $2, $0, 3
mult $1, $2
```

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？

每条乘法指令需要 5 个周期，每条除法指令需要 10 个周期，若将乘除法也整合进 ALU，则每次出现乘除法指令时整条流水线都需要暂停 5 或 10 个周期（因不需要使用 ALU 的指令极少），会造成效率的巨大损失，而单独使用 XALU 后乘除法指令后的非乘除法指令不会被阻塞，对流水线效率影响较小。乘除法结果为 64 位而其他一般为 32 位，故需要独立的两个寄存器用于存储结果，这样能减少流水线寄存器等的连线和复杂程度。

2. 参照你对延迟槽的理解，试解释“乘除槽”。

当乘除法正在计算时 XALU 的 BUSY 端口输出为 1，此时非乘除法指令仍能正确通过流水线传递而不受影响，此即为在“乘除槽”内。正是因为有“乘除槽”的存在，流水线才能够在 E 级才开始分析指令类型而不需要在 D 级便分析完全。

3. 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）

当汇编语言中连续出现 lb 指令而 lw 指令较少时（如 C 语言中在字符串中取字符的情况）按字节访问内存相对于按字访问性能上更有优势，因每个 char 类型变量仅占据 1byte 内存，按字节访问后不需对所取数据进行额外的拓展处理。

4. 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？

我的暂停与转发模块为 DETECTOR 型，它接收各流水线寄存器上指令的 WD 和 A3 和 Tnew 信息，在发生数据冒险时及时进行暂停或转发。为了对抗复杂性，我的暂停与转发模块由多个小模块构成，Analysis_MUX 用于由各级指令本身分析出各级 WD 与 A3，StopUnit 用于由各级 A3 与 TnewTuse 产生暂停信号，ForwardUnit 用于数据转发，并一并连接到数据通路中，这样每个模块的代码量不会过多，同时功能简洁，在出现 bug 时易于定位错误位置与调试。此外，规范变量名，对齐代码等方法也能够对抗其复杂性。

5. 你对流水线 CPU 设计风格有何见解？

流水线 CPU 的各功能部件（PC、ALU、EXT、GRF 等）由于其功能的高度特定化而大同小异，因此区分 CPU 设计风格的主要要点在于控制器的设计和转发暂停的方式。其中，控制器既可以将指令用与或门进行连接并产生控制信号，也可以使用 case 语句将每一个指令的控制信号单独列出，前者优点在于代码量小，但是难以在调试时看清每一条指令的控制信号，后者代码量大，但可以复制粘贴得出且较清晰，加指令时也较为方便。转发暂停的方式主要有表格法、暴力转发与标记转发，三者思考难度上逐渐递增，但代码量与修改的难易程度逐渐递减。

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？

见测试样例部分。