计算机组成原理实验报告：Verilog 流水线 32 位 CPU 设计

一、模块规格

A.IFU

1. 基本描述

IFU（Instruction Fetch Unit，取指令单元），内部包括 PC（程序计数器）、IM（指令存储器）及相关逻辑，用于指令的有序获取和执行。

2. 端口说明

IFU 端口说明

信号名	方向	描述
MUX4PC_out[31:0]	I	下一条指令的 PC 值
Clk	I	时钟信号
Reset	I	复位信号 1：PC 被设置为 0x00003000 0：无效
PC4	O	PC+4
Instruction[31:0]	O	32 位 MIPS 指令

3. 功能描述

IFU 功能描述

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 被设置为 0x00003000
2	取指令	instruction=IM[PC]
3	顺序寻址	如果当前指令不是是分支指令或跳转指令，PC=PC+4
4	跳跃寻址	如果当前指令是分支指令或跳转指令，需要根据指令等改变 PC 的值

B.GRF

1. 基本描述

GPR（General Register File，通用寄存器组），是多个寄存器组成的阵列，用于在内存与 CPU 运算部件之间暂存数据。MIPS 模型机寄存器使用约定如下表所示。

MIPS 模型机寄存器使用约定

寄存器编号	寄存器名称	用途
0	$zero	常数 0
1	$at	保留给汇编器使用
2~3	$v0~ $v1	结果值和表达式求值
4~7	$a0~ $a3	参数
8~15	$t0~ $t7	临时变量
16~23	$s0~ $s7	数据寄存器
24~25	$t8 ~ $t9	其他临时变量
26~27	$k0 ~ $k1	保留给操作系统使用
28	$gp	全局指针
29	$sp	栈指针
30	$fp	帧指针
31	$ra	返回地址

2. 端口说明

GRF 端口说明

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号 1：所有寄存器中的值被清零 0：无效
WriteEnabled	I	写使能信号 1：可以向 GRF 写入数据 0：不可以向 GRF 写入数据
A1[4:0]	I	读寄存器地址 1，指定一个寄存器并将其数据读出到 RD1
A2[4:0]	I	读寄存器地址 2，指定一个寄存器并将其数据读出到 RD2

A3[4:0]	I	写寄存器地址，指定一个寄存器作为写入的目标寄存器
WriteData[31:0]	I	32 位数据输入信号
PC4[31:0]	I	PC+4 的值
RD1[31:0]	O	输出 A1 所指定寄存器中存储的数据
RD2[31:0]	O	输出 A2 所指定寄存器中存储的数据

3. 功能描述

GRF 功能描述

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有寄存器存储数值清零
2	写数据	当写使能信号有效且时钟上升沿来临时，将 WD 写入 A3 对应的寄存器
3	读数据	读出 A1、A2 地址对应寄存器存储的数据至 RD1、RD2

C.ALU

1. 基本描述

ALU（Arithmetic and Logic Unit，算术逻辑单元），是实现多组算术运算和逻辑运算的组合逻辑电路。

2. 端口说明

ALU 端口说明

信号名	方向	描述
A[31:0]	I	32 位输入数据 1
B[31:0]	I	32 位输入数据 2
ALUctr[3:0]	I	控制信号 0000：无符号加法 0001：无符号减法 0010：按位或运算 0011：按位和运算
C[31:0]	O	32 位输出数据

3. 功能描述

ALU 功能描述

序号	功能名称	功能描述
1	无符号加法	C=A+B，不考虑溢出
2	无符号减法	C=A-B，不考虑溢出
3	按位或运算	C=A|B
4	按位和运算	C=A&B
5	（其他功能）	（新增指令时，可能需要增加新的功能）

D.DM

1. 基本描述

DM（Data Memory，数据存储器），用于保存数据。

2. 端口说明

DM 端口说明

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号 1：所有寄存器中的值被清零 0：无效
In[31:0]	I	32 位输入数据
MemWrite	I	写入控制信号
MemAddr[31:0]	I	输入地址
PC4[31:0]	I	PC+4 的值
Out[31:0]	O	32 位输出数据

3. 功能描述

DM 功能描述

序号	功能名称	功能描述
1	复位	复位信号有效时，所有数据清零
2	读出数据	根据输入地址读出相应数据
3	写入数据	当写入控制信号有效时，将输入数据写入存储器相应地址

E.EXT

1. 基本描述

EXT（Extender，位扩展器），用于对数据进行某种扩展。

2. 端口说明

EXT 端口说明

信号名	方向	描述
In[15:0]	I	16 位输入数据
Extop[1:0]	I	控制信号 00：无符号扩展（零扩展） 01：符号扩展 10：低位补 0 扩展
Out[31:0]	O	32 位输出数据

3. 功能描述

EXT 功能描述

序号	功能名称	功能描述
1	无符号扩展	对输入数据进行无符号扩展，即零扩
2	符号扩展	对输入数据进行符号扩展
3	低位补 0 扩展	对输入数据进行低位补 0 扩展

二、控制器设计

A.端口说明

控制器端口说明

信号名	方向	描述
op[5:0]	I	当前指令的 op 字段
func[5:0]	I	当前指令的 func 字段
RegDst[1:0]	O	寄存器堆写地址控制 00：写入的目标寄存器地址来自 rt 字段 01：写入的目标寄存器地址来自 rd 字段 10：写入的目标寄存器是$ra，即 31
ALUsrc	O	ALU 第二个操作数选择信号 1：来自符号扩展后的立即数 0：来自寄存器堆第二个输出数据
Mem2Reg[1:0]	O	寄存器堆输入选择信号 00：来自 ALU 的输出 01：来自存储器的输出 10：来自 PC+4
RegWrite	O	寄存器堆写使能信号 1：寄存器堆可以写入数据 0：寄存器堆不能写入数据
MemWrite	O	存储器写使能信号 1：存储器可以写入数据 0：存储器不能写入数据
selPC[1:0]	O	当前指令是否是分支指令或跳转指令的标志 01：当前指令是分支指令 10：当前指令是 J 跳转指令 11：当前指令是 JR 跳转指令 00：其它

EXTop[2:0]	O	EXT 控制信号 000：无符号扩展 001：符号扩展 010：低位补 0 扩展 011：改为 PC+4
ALUop[3:0]	O	ALU 控制信号 0000：无符号加法 0001：无符号减法 0010：按位或运算 0101：输出 B

B.真值表（不含转发、暂停）

控制器真值表

func	100001	100011	n/a	n/a	n/a	n/a	n/a	000000
op	000000	000000	001101	100011	101011	000100	001111	000000
	addu	subu	ori	lw	sw	beq	lui	nop
RegDst	01	01	00	00	xx	xx	00	xx
ALUsrc	0	0	1	1	1	0	1	x
Mem2Reg	00	00	00	01	xx	xx	00	xx
RegWrite	1	1	1	1	0	0	1	0
MemWrite	0	0	0	0	1	0	0	0
selPC	00	00	00	00	00	01	00	00
EXTop	xxx	xxx	000	001	001	xxx	010	xxx
ALUop	0000	0001	0010	0000	0000	0001	0000	xxxx

控制器真值表（更多指令）

func	n/a	n/a	001000					
op	000010	000011	000000					

	j	jal	jr					
RegDst	xx	10	xx					
ALUsrc	x	1	x					
Mem2Reg	xx	10	xx					
RegWrite	0	1	0					
MemWrite	0	0	0					
selPC	10	10	11					
EXTop	xxx	011	xxx					
ALUop	xxxx	0101	xxxx					

更详细的真值表见压缩包内真值表文件内

三、测试 CPU

测试 CPU 的部分较为复杂，代码附在其他文件内，这里讲述我的测试方法。

首先，本人使用了 Python 写了部分测试程序和 Mars 指令生成程序以提高测试的效率，因为指令数量较多，想要尽可能全面。

对于每条指令，需要测试其功能、作为需求方产生数据冒险、作为供给方产生数据冒险，因此，第一个测试要求在后两者测试时也会覆盖到，因此只测试后两者。

A.作为需求方，即需求 R1、R2 时的数据冒险测试

测试方法，首先使用编写“被测指令”，然后使用程序生成测试用指令（自动生成指令），然后使用改造过的 Mars、本 CPU 分别运行，将结果放入相应 TXT，再使用测试程序测试。

测试使用程序、“被测指令”、自动生成指令都在压缩包内的其他文件内。

B.作为供给方的数据冒险测试

测试方法同上。

C.特殊指令的功能测试

例如 J、JAL、JR、JALR，Load 族、Save 族、B 族等指令，前后跳转、前后存储、不同

位存储等情况，需要手工构造专属测试指令。

四、设计思路详解

A.数据冒险的根本原因思考

数据冒险是由于某指令对寄存器进行写入操作是需要时间的，在这段时间内之后的指令如果对相同寄存器有读取操作，则会读到未更新的错误值。完全暂停会导致流水线性能大大下降，而正确值在写回阶段前就可能产生，于是可以通过转发来使得提前获得正确值，减少暂停。

从对寄存器的供给和需求角度出发，在暂停实现的基础上，采用暴力转发，即符合要求就转发，可能转发多次。暂停机制保障相关指令之间保持“距离”，即保证一定在正确值产生后转发，也就是转发对的值。本设计中采用 WhoNew 信号判断寄存器相关情况，保证转发给对的寄存器。

B.暂停的实现（真值表）

	addu	ori	lui	beq	sw	lw	jal	jr
WhoNew	r3	r2	r2	/	/	r2	31	/
TNew	2	2	2	/	/	3	3	/
r1_TUse	2	2	/	1	2	2	/	1
r2_TUse	2	/	/	1	3	/	/	/

更详细的真值表见压缩包内真值表文件内

上表列出了指令的 WhoNew、Tnew、Tuse 值，其中 WhoNew 由每个阶段的控制器发出，Tnew 由 D、E 阶段的控制器发出（因为 M 阶段一定以及产生了，而指令本身就在 F 阶段，故这两个不用考虑），Tuse 仅在 F 阶段的控制器内计算。

关于“/”的处理：WhoNew 中“/”改为 0，因为实现了 0 号寄存器不转发；Tnew 中改为 0，由于 WhoNew 存在，WhoNew 为 0 的指令的 Tnew 的值不关心；Tuse 中改为 5，视为不需要使用，因为 5 总能大于任何 Tnew。

C.转发的实现

在 D、E、M 阶段都增加 R1、R2 转发寄存器，候选数值为其后续流水线寄存器中的值（分别为 ALUout 的值和 WD2A3 多路选择器选择的值），当 WhoNew 与 r1 或 r2 相等且不为零时选择该值，就近优先。

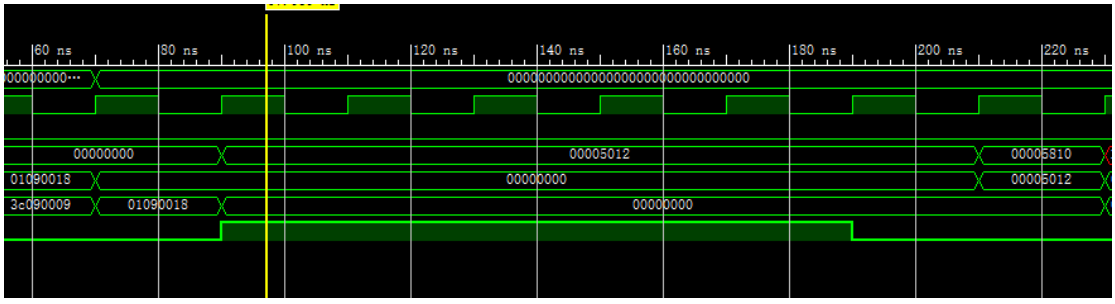
本设计采用的暴力转发不出错的要求之一是，对 Tnew<=x 的指令而言，在 x 阶段输出的流水线寄存器唯一。即 Tnew<=2 的指令在 E 阶段正确值的位置都是 ALUout，Tnew<=3 在 M 阶段正确值的位置都是 WD2A3。

D.全速的实现

lui 实际在 D 阶段就产生结果，jal 也是，改进 cpu 结构的话，可以缩短它们的 Tuse，从而减少暂停的需要，以达到全速。本设计中改进了这两种指令，使得 cpu 达到全速。

E.乘除模块的补充说明

由于本设计中，暂停设计在取指阶段内部，和其他设计不同，因此乘除模块的 busy 信号等设计略有不同。如果 busy 信号对乘除分别是 5、10 的话，波形如下图，由于本设计中**暂停的阶段比较早**，实际上这样做会在下一条冲突指令到达 E 阶段前延迟 7、12 个周期，因此本设计中把 busy 信号设为 3 和 8，而在实际的运行中，乘除模块有 5、10 个周期来计算。



如果设为 busy 分别设 5、10 的情况

本设计中，不是用 start 和 busy 取或来暂停的，还是由于，本设计中把会 start 置 1 的四个乘除指令单独增加一个用于判断 stock 的信号。设计本信号还有一个理由，是为了课上测试时应对需要暂停的、本设计中不能通过其他信号实

现暂停的特殊信号，如可能会写给多个寄存器或者写给哪个寄存器不确定的指令。

五、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：因为现实中乘除法消耗的时间大大超过其他 ALU 功能消耗的世界，如果整合进 ALU，整个执行阶段的时钟周期就会变长，从而使得流水线整体的指令周期变长，使得 CPU 的效率大大降低。一般而言，做完乘除法很可能紧接着使用 MFHI、MFLO，如果不用独立的寄存器，就会增加数据冲突，使得转发的设计复杂化。

2. 参照你对延迟槽的理解，试解释“乘除槽”。

答：乘除槽内的指令总会执行，不像乘除法相关指令都被阻塞。乘除槽也是为了尽量减小因乘除指令的数据冒险而被迫暂停所带来的损失，从而提高流水线 CPU 的效率。乘除槽内的指令应当与乘除法相关指令没有任何数据冒险。

3. 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？

答：设计要求转发是从寄存器转发，防止以功能部件作为转发源可能会导致的冲突级的延迟会加上转发过后的组合逻辑延迟。lw 指令本身就在 MEM/WB 之后转发，lb 等指令的数据扩展模块就也应该与 lw 保持一致。

4. 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。

答：例如读取按字节存储的多个等长度且不为 4byte 整数倍的字符串，例如”ab\0”、”bc\0”等，如果按字访问内存需要花费时间查找，而按字节访问可以快速计算出地址。并且按字节存储比按字存储更节省内存空间。

5. CPU 风格介绍。

答：本人使用 Detector 风格设计，具体介绍见设计思路详解部分，除了统一的命名规

范、使用宏、模块化编程等，还使用了以下小技巧/思想。

1.暴力转发：使用的前提是转发给不需要该寄存器的值的指令也不会造成不良影响，以及每个值在每个阶段的在唯一确定的位置，D 为 immediate 寄存器，E 为 ALUout 寄存器，M 为 M4 多路选择器（本设计中称为 WD2A3）。

2.对指令的供需分析精准到寄存器（Detector 风格），因此没有分 cal_r、cal_i 等等类别，当然设计的时候还是了解这些类别的，添加时同一类型的有相似之处。

3.采用“就阶段分析”原则，即每个指令只由它所在阶段的控制器分析其类型、供给需求等等，避免将指令到处传递。

4.为每条指令计算 R1Tuse、R2Tuse、TNew、WhoNew 四个属性，基于这些来实现暂停、转发（乘除相关指令还需考虑 busy）。

5.lui、带 link 的跳转指令将结果提前至 D 阶段计算完成，并增加相应转发，以加速。