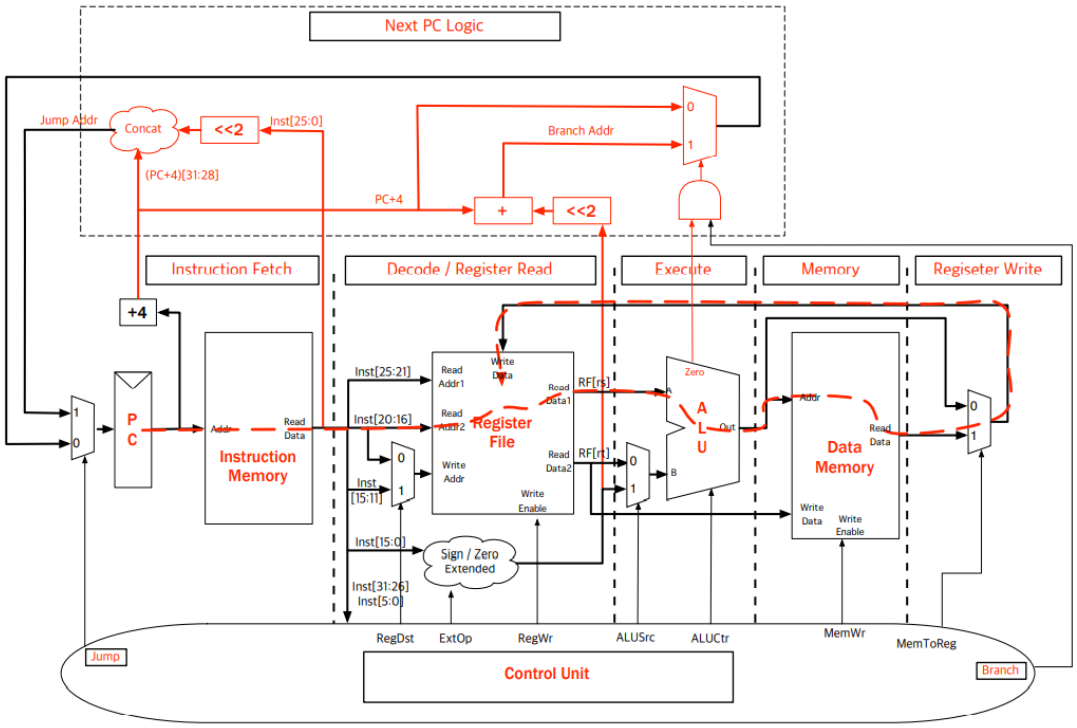
处理器设计（作业）

陈文迪 519021910071

1. 单周期处理器控制逻辑

这是一个单周期处理器的数据通路（single cycle CPU diagram）：



在下表中填写出上图中各个控制信号的数值：

Instrs.	Control Signals								
			RegDst	ExtOp	ALUSrc	ALUCtr	MemWr	MemtoReg	RegWr
add									
ori									
lw									
sw									
beq									
j									

Instrs.	Control Signals								
			DegDst	ExtOp	ALUSrc	ALUCtr	MemWr	MemtoReg	RegWr
add			1	x	0	0010	0	0	1
ori			0	unSn	1	0001	0	0	1
lw			0	Sn	1	0010	0	1	1
sw			x	Sn	1	0010	1	x	0
beq			x	Sn	0	0110	0	x	0
j			x	x	x	x	0	x	0

这个表格给出了算术逻辑单元每个操作的 ALUCtr 值：

Operation	AND	OR	ADD	SUB	SLT	NOR
ALUCtr	0000	0001	0010	0110	0111	1100

2. 单周期处理器的性能分析

时钟分析方法：

- 每个状态元件的输入信号必须在时上升沿之前稳定下来。
- 关键路径（critical path）：电路中状态元件之间最长的延迟路径。
- $t_{clk} \geqslant t_{clk\text{-}to\text{-}q} + t_{CL} + t_{setup}$，其中 t_{CL} 是组合逻辑中的关键路径
- 如果我们把寄存器放在关键路径上，我们可以通过减少寄存器之间的逻辑量来缩短周期。

电路元件的延时如下所示：

Element	Register clk-to-q	Register Setup	MUX	ALU	Mem Read	Mem Write	RegFile Read	RegFile Setup
Parameter	$t_{clk\text{-}to\text{-}q}$	t_{setup}	t_{mux}	t_{ALU}	$t_{MEMread}$	$t_{MEMwrite}$	t_{RFread}	$T_{RFsetup}$
Delay(ps)	30	20	25	200	250	200	150	20

关于硬件中的时钟的一些术语说明：

- 时钟（CLK）：使系统同步的稳定方波
- 启动时间（setup time）：在时钟边沿之前，输入必须稳定的时间
- 保持时间（hold time）：在时钟边沿之后，输入必须稳定的时间
- “CLK-to-Q”延迟（“CLK-to-Q” delay）：从时钟边沿测量，改变输出需要多长时间
- 周期（period）=最大延迟=“CLK-to-Q”延迟+CL 延迟+启动时间
- 时钟频率=1/周期（即周期的倒数）

回答问题：

1) 用到关键路径（critical path）的指令是哪一条？

用到关键路径的指令为 load 指令。

2) 最小时钟周期 t_{clk}是多少？最大时钟频率 f_{clk}是什么？假设 $t_{clk\text{-}to\text{-}q}$ > 保持时间(hold time).

我们需要考虑关键路径的延迟。事实上，load 指令的延迟为

$$30 + 250 + 150 + (25 + 200) + 250 + 25 + 20 = 950\text{ps}$$

因此，最小时钟周期为 950ps。最大时钟频率为 1/(950ps) = 1.05GHz。

3. 流水线处理器设计（Pipelined CPU Design）

现在，我们将使用流水线方法来优化一个单周期处理器。流水线虽然增加了单个任务的延迟，但它可以减少时钟周期，提高吞吐量。 在流水线处理器中，多条指令重叠执行，体现了指令级并行性。

为了设计流水线，我们已经将单周期处理器分成五个阶段，在每两个阶段之间增加流水段寄存器。

接下来进行性能分析：

我们将使用与上一题相同的时钟参数：

Element	Register clk-to-q	Register Setup	MUX	ALU	Mem Read	Mem Write	RegFile Read	RegFile Setup
Parameter	$t_{clk\text{-}to\text{-}q}$	t_{setup}	t_{mux}	t_{ALU}	$t_{MEMread}$	$t_{MEMwrite}$	t_{RFread}	$T_{RFsetup}$
Delay(ps)	30	20	25	200	250	200	150	20

回答问题：

1) 这个五阶段流水线处理器的最小时钟周期长度和最大时钟频率分别是多少？
2) 相比于单周期处理器，性能加速比（speed up）是多少？为什么加速比会小于 5？

1）

我们可以根据如下式子计算。

$$t_{clkpipe} >= \max \begin{pmatrix} t_{clk\text{-}to\text{-}q} + t_{MEMread} + t_{setup}\ (\text{Fetch}) \\ t_{clk\text{-}to\text{-}q} + t_{RFread} + t_{setup}\ (\text{Decode}) \\ t_{clk\text{-}to\text{-}q} + t_{ALU} + t_{mux} + t_{setup}\ (\text{Execute}) \\ t_{clk\text{-}to\text{-}q} + t_{MEMread} + t_{setup}\ (\text{Memory}) \\ t_{clk\text{-}to\text{-}q} + t_{mux} + t_{RFsetup}\ (\text{Writeback}) \end{pmatrix}$$

由此我们得到最短时钟周期长度为 300ps，最大时钟频率为 3.3GHz。

2）加速比为 3.17。小于 5 的原因是因为每个阶段的延迟并不相同，周期长度取决于延迟最大的阶段，并且阶段之间的段寄存器增加了延迟。

4. 控制冒险（Control Hazard）

遇到 branch 和 jump 指令的时候会发生控制冒险。我们可以通过暂停流水线来解决。但是，由于分支条件是在执行阶段计算的，流水线需要停顿三个周期。我们可以在寄存器读取阶段增加一个分支比较器，并引入一个转移延迟槽（delayed slot），使分支语句(branch)后的指令总是会被执行。

问题：

考虑填充转移延迟槽，我们需要重新排列以下几组指令，如果实在找不到指令填充延迟槽，你可能需要插入一条 nop 指令。

Set 1	Reordered set 1	Set 2	Reordered Set 2
addiu $t0, $t1, 5		addiu $t0, $t1, 5	
ori $t2, $t3, 0xff		ori $t2, $t3, 0xff	
beq $t0, $s0, label		beq $t0, $t2, label	
lw $t4, 0($t0)		lw $t4, 0($t0)	

Set 1	Reordered Set 1	Set 2	Reordered Set 2

addiu $t0,$t1,5	addiu $t0,$t1,5	addiu $t0,$t1,5	addiu $t0,$t1,5
ori $t2,$t3,0xff	beq $t0,$s0,labe	ori $t2,$t3,0xff	ori $t2,$t3,0xff
beq $t0,$s0,label	ori $t2,$t3,0xff	beq $t0,$t2,label	beq $t0,$t2,label
lw $t4,0($t0)	lw $t4,0($t0)	lw $t4,0($t0)	nop
			lw $t4,0($t0)

5. 转移延迟槽

考虑以下两种设计:

- 第一种设计为每一条 branch 指令设计两个转移延迟槽（delay slots)，但不使用转移预测（branch prediction),而是在编译时调度可用的指令填充转移延迟槽。假设其中30%的 branch 指令在编译时能找到指令将两个延迟槽填满，60%的 branch 指令在编译时只能找到指令填充一个延迟槽，剩下 10%的 branch 指令的两个延迟槽在编译时无法填充。
- 第二种设计不采用转移延迟槽 (delay slots)，而是采用转移预测 (Branch Prediction)。转移预测错误的开销（mis-prediction penalty)是 3 个周期。Branch 指令本身需要花一周期执行，但如果转移预测错误，就会增加 3 个额外周期的开销。

如果需要第二种设计能达到第一种设计的性能，转移预测的准确度应该为多少?

我们假设预测的准确度为 x，则我们比较二者产生的额外开销。

$$(1-x)\times 3 \le 0.6\times 1 + 0.1\times 2$$

解上述不等式可得预测的准确度应该至少达到 73.3%。

6. 指令调度

假定在一个有转发（forwarding）功能的五段流水线中执行以下程序段，则可以怎样调整以下指令序列使其性能达到最好?

```
1    lw    $2, 100($6)
2    add   $2, $2, $3
3    lw    $3, 200($7)
4    add   $6, $4, $7
5    sub   $3, $4, $6
6    lw    $2, 300($8)
7    beq   $2, $8, Loop
```

我们可以按照如下次序排列指令序列：1、4、2、6、3、5、7。

7. 中断

- 当 MIPS 处理器在执行一条除法指令时，发生了除数为 0 异常（exception）。那么此时处理器就要进行中断处理。在中断处理过程中，最开始的一部分工作由硬件完成，描述一下:

1）中断处理开始的阶段，硬件需要完成哪些工作，保存哪些状态?

2）如果中断处理程序（interrupt handler）需要读寄存器 R5, R6, R7， 写寄存器 R5, R8, R10, 那么中断处理程序应该在一开始保留哪几个寄存器的值？
3）ERET（中断返回）指令会触发硬件完成哪些动作？
4）如果一条指令在执行阶段，即发生了“指令地址错误”异常，又发生了“ALU 运算溢出”异常，那么这条指令被中断时，原因寄存器（cause register）中记录的中断原因，应该是哪一个？

1）在中断处理阶段，首先需要设置 EPC 为发生异常的指令的地址；然后设置 STATUS 寄存器中的一个控制位 SR（EXL）,强迫 CPU 进入 kernel 态，禁用中断响应来“关中断”；接着设置 Cause 寄存器，使软件可以得到异常的类型信息；最后 CPU 开始从一个统一入口取指令，剩下的操作交由软件处理。
2）应该保留 R5, R8, R10 这些寄存器的值。
3）ERET 会将 EPC 中的内容移入 PC，并将 SR（EXL）清零来“开中断”，允许新的中断响应，CPU 进入用户态。接下来开始重新执行被异常事件中断的那条指令。
4)依据精准中断，对于单个指令“指令地址错误”异常会覆盖“ALU 运算溢出”异常。因此，原因寄存器应该记录“指令地址错误”异常。