作业 3:　RISC-V 处理器

1.　下图是一个实现了 RV32I 指令集的单周期处理器的数据通路，为使得不同的指令在同一个处理器中完成不同的功能，控制器对指令译码后，要发出不同的控制信号。表格中罗列了部分指令的控制信号。其中 * 表示控制信号可以任意取值；BrUn 为 1 时，表示比较指令的两个比较值是无符号数；Immsel 取不同的值，是因为不同的指令中的立即数在指令中不同的位置、以及立即数扩展的方法不同。

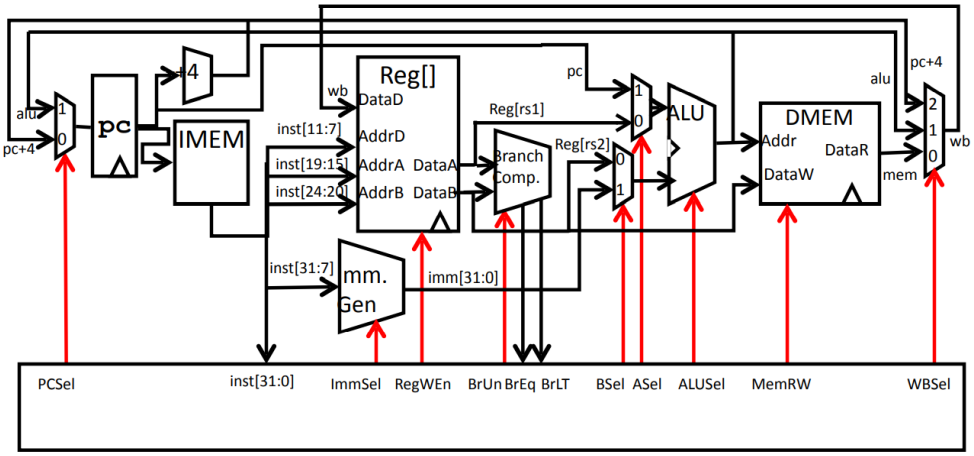


图 1. RV32I 单周期处理器数据通路

	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	*	*	0 (PC + 4)	*	*	0 (Reg)	0 (Reg)	add	0	1	1 (ALU)
ori	*	*	0	I	*	0 (Reg)	1 (Imm)	or	0	1	1 (ALU)
lw	*	*	0	I	*	0 (Reg)	1 (Imm)	add	0	1	0(MEM)
sw	*	*	0	S	*	0 (Reg)	1 (Imm)	add	1	0	*
beq	1/0	*	1/0	SB	*	1 (PC)	1 (Imm)	add	0	0	*
jal	*	*	1 (ALU)	UJ	*	1 (PC)	1 (Imm)	add	0	1	2(PC + 4)
bltu	*	1/0	1/0	SB	1	1 (PC)	1 (Imm)	add	0	0	*

表 1. 部分指令的控制信号

回答问题：参考上面的图表，分别写出 slli 指令和 AUIPC 指令的控制信号

答：

	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
slli	*	*	0	I	*	0(Reg)	1(Imm)	sll	0	1	1(ALU)
AUIPC	*	*	1	U	*	1	1	add	0	1	1

参考信息如下:

SLLI 用法举例: sllix11,x12,2 表示 x11 = x12<<2（Shift Left Logical Immediate）

SLLI 指令编码格式:

imm			funct3		opcode	
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI

AUIPC 指令用法举例:

```
auipc x1, <hi20bits>  # Adds upper immediate value to
                      #  and places result in x1
```

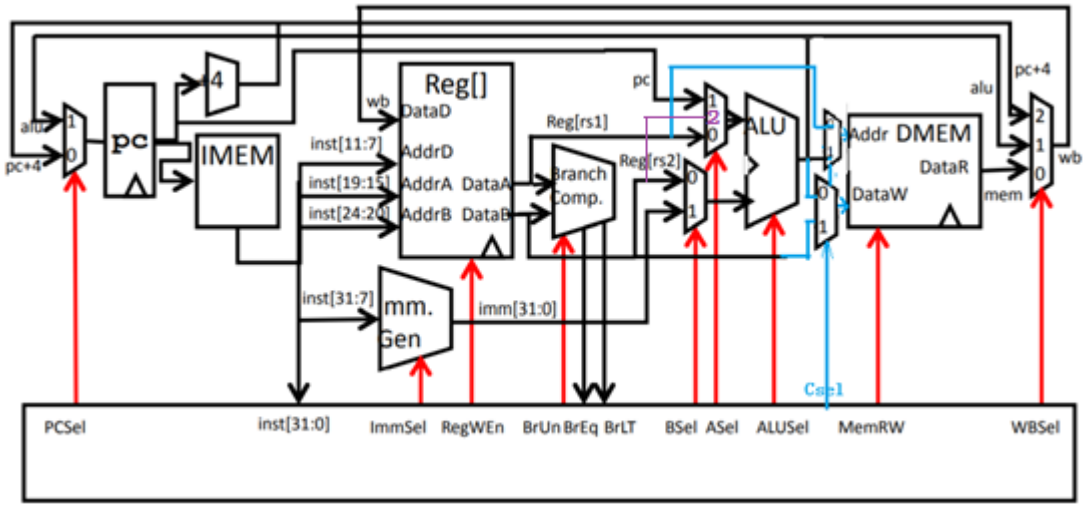
AUIPC 指令编码格式:

更多关于 RISC-V 指令的信息可参考:
http://inst.eecs.berkeley.edu/~cs61c/resources/RISCV_Green_Sheet.pdf

2. 尝试添加 RISC-V 指令： ss rs1, rs2, imm

指令的功能为： Mem[Reg[rs1]] = Reg [rs2] + immediate （存储两数之和）

回答问题：为了支持这条指令，对图 1 中的 RV32I 单周期处理器，需要添加的新功能部件是什么？现有的哪些部件需要改造？需要新添加的数据通路是什么？ 为控制单元新添加的控制信号有哪些?



答：

1、需要添加的新功能部件：
①DMEM 的 Addr 的 Mux 部件。
②DMEM 的 DataW 的 Mux 部件。
2、需要改造的部件：
①ALU 的第一个操作数的 Mux 部件，增加一个信号位 2。
3、需要新添加的数据通路：
①从 DataB 到 ALU 的第一个操作数的通路，ASel=2 时选通。
②从 DataA 到 DMEM 的 Addr 的通路,CSel=0 时选通。
③从 ALU 的输出结果到 DMEM 的 DataW 的通路，CSel=0 时选通。
4、新添加的控制信号：
①CSel 信号同时控制新添加的两个部件：CSel=1 时，Addr 为 ALU 的输出结果，DataW 为 Reg[rs2]；CSel=0 时【ss 指令对应的情况】，Addr 为 Reg[rs1]，DataW 为 ALU 的输出结果。

3. 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
- 建立时间（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）：从时钟边沿开始，到输出改变需要多长时间
- 周期（period）= 最大延迟= “CLK-to-Q”延迟+CL 延迟+建立时间
- 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？

提示：找到关键路径最长的那条指令，例如 add s1, t1, t2 指令的关键路径是：IMEMread + RegFileRead + MUX +ALU + MUX，哪条指令的关键路径更长？ 找出来

答：

根据表中数据，可看出 ALU、MemRead、MemWrite、RegFileRead 这几个元件的延时远大于其他元件，其中，ALU 与 RegFileRead 几乎在所有指令都会存在，而 MemRead 的延时大于 MemWrite，所以 lw 指令的关键路径更长。

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).

提示：tclk= PC 寄存器的 clktoQ + 关键路径（critical path）延迟 + RegFile_Setup

答：

$t_{clk} \geqslant 30+（250+150+25+200+250+25）+20 = 960$ ps

$f_{clk} \leqslant 1/960\ ps^{-1}$

故最小时钟周期为 960ps，最大时钟频率为 $1/960\ ps^{-1}$ 。

4. 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？

提示：

流水线处理器的最小时钟周期= max(clk-to-q + 某段的延迟 + 段寄存器 setup_time)

答：

t_{clk} = 30+（250）+20 = 300 ps

f_{clk} = 1/300 ps^{-1}

故最小时钟周期为 300ps，最大时钟频率为 1/300 ps^{-1} 。

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？

答：

speed up = 960/300 = 3.2

为什么小于 5：因为“某段的延迟”应该取最长的段延迟。

5. 图 2 给出了 RV32I 五阶段流水化处理器示意图：

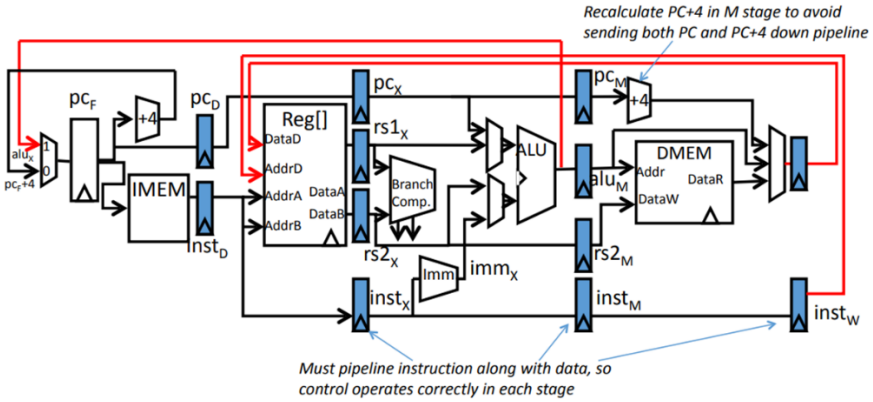


图 2. RV32I 五阶段流水处理器

在该处理器上执行以下指令序列：

```
add  x15,x12,x11
ld     x13, 4(x15)
ld     x12 , 0(x2)
or     x13, x15,x13
sd     x13, 0(x15)
```

注：寄存器堆先写后读，ID 阶段的指令可以在同一个周期内得到 WB 阶段写回的数据；

回答问题：

（1） 假设硬件不检查和处理冒险，数据通路没有前向传递（Forwarding），在指令序列中

插入空指令 NOP，使得上述指令序列得到正确的执行结果。

答：

```
add   x15,x12,x11
NOP
NOP
ld    x13, 4(x15)
ld    x12 , 0(x2)
NOP
or    x13, x15,x13
NOP
NOP
sd    x13, 0(x15)
```

（2） 假设对硬件不改变，是否可以编译优化：对代码的次序重排、寄存器换名，使得插入的空指令减少？

答：

无法减少总的插入空指令数。

（3） 假设进行硬件优化：数据通路中增加了前向传递（forwarding）， 并增加了冒险检测单元。 哪些指令之间还是需要停顿？停顿几个周期？

答：

不需要停顿

6. 在一个采用“取指、译码/取数（ID）、执行、访存、写回”的五段流水线中，bne 指令检测结果是否为“零”的操作在执行阶段进行。 j 指令以下指令序列中，在有数据转发（forwarding）、并将转移方向和转移地址计算提前到 ID 段进行（后面提供了文字和图片说明）的情况下，哪些指令执行时会发生流水线阻塞？各需要阻塞几个时钟周期？

```
1   loop:    add   t1, s3, s3
2            add   t1, t1, t1
3            add   t1, t1, s6
4            lw    t0, 0(t1)
5            bne   t0, s5, exit
6            add   s3, s3, s4
7            j       loop    //这是一条伪指令，是 jal 指令的变体
8    exit:
```

答：

执行 bne 时会发生流水线阻塞，需要阻塞 1 个时钟周期。
当 bne 指令跳转时，需要阻塞 2 个周期。
当 j loop 跳转时，需要阻塞 1 个时钟周期。

对“**转移方向和转移地址计算提前到 ID 段**”提供一些补充信息供参考：

如果将跳转地址的计算放在 ALU 段，因此无论是 beq 这样的条件转移指令还是 Jal 这样的无条件转移指令,都要在 ALU 段才计算出转移地址，所以如下图 3，当 beq 指令跳转时，要浪费两个周期（该指令的 CPI=3）。同理，Jal 指令也一样。

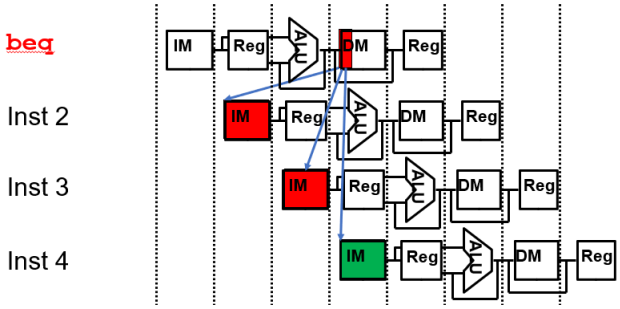


图 3. 控制冒险导致的周期停顿

为了提高转移指令的性能，设计处理器时，可以将转移条件计算和转移地址计算都提前到流水线的 ID 段进行。如下图中红色圈出部分所示：

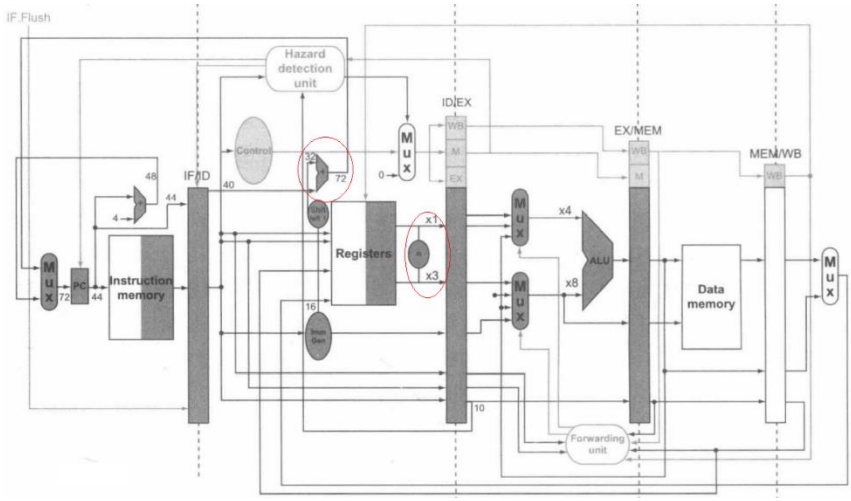


图 4. 改进后的流水线数据通路（图片来源： 教材图 4-60）

改进后，jal 和 beq 指令发生转移时，只需停顿一个时钟周期（CPI=2）。