第四章　处理器

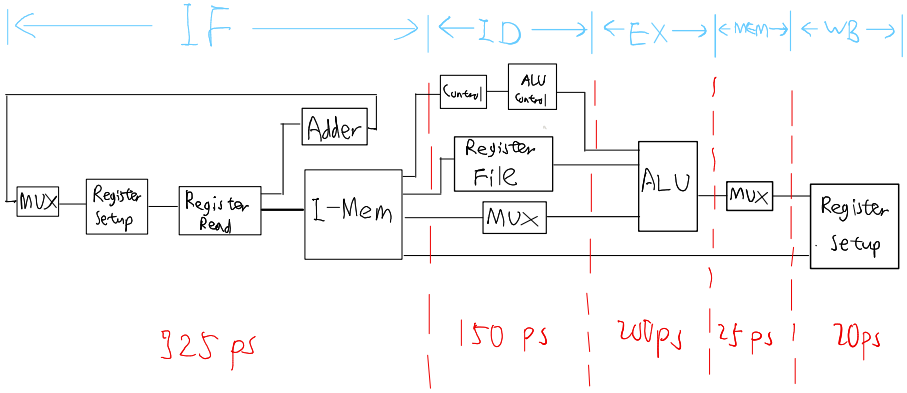
4.7 假设用来实现处理器数据通路的各功能模块延迟如下所示：

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

其中，寄存器读延迟指的是，时钟上升沿到寄存器输出端稳定输出新值所需的时间。该延迟仅针对 PC 寄存器。寄存器建立时间指的是，寄存器的输入数据稳定到时钟上升沿所需的时间。该数值针对 PC 寄存器和寄存器堆。

4.7.1 [5]<4.4>R 型指令的延迟是多少？比如，如果想让这类指令工作正确，时钟周期最少为多少？

R 型指令的步骤如下图所示。其中，在译码阶段，Control 的延迟为 50ps，Register File 的延迟为 150ps，Mux 的延迟为 25ps，由于这三个步骤可以同时执行，所以延迟取最大值，即 150ps。在访存（MEM）阶段，由于 R 型指令不需要访问内存，所以只需要通过一个多路选择器（MUX），所以延迟为 25ps。

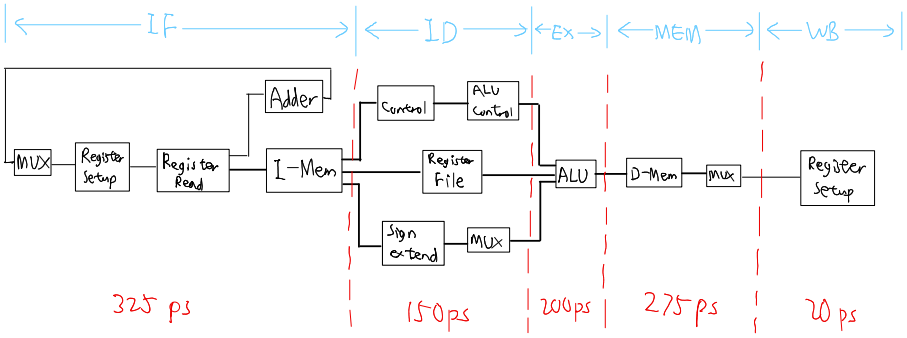


因此，延迟为 $325 + 150 + 200 + 25 + 20 = 720$ ps，即如果想让这类指令工作正确，时钟周期最少为 720 ps。

4.7.2 [10]<4.4>ld 指令的延迟是多少？仔细检查你的答案，许多学生会在关键路径上添加额外的寄存器。

ld 指令的步骤如下图所示，可以观察到在译码步骤中，虽然延迟的组成部分和上一

题不一样，但由于 Register File 的延迟较长，因此总的延迟还是由 Register File 决定，即 150ps。IF、ID、EX 步骤的延迟与上一题没有改变，但后面两个步骤有所改变。



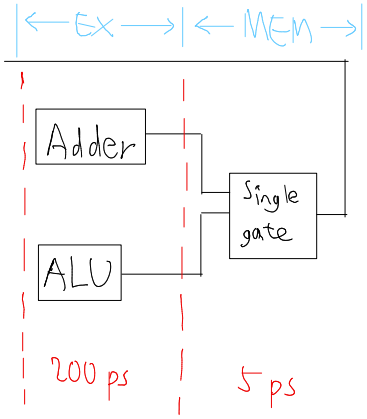
因此，延迟为 $325 + 150 + 200 + 275 + 20 = 970$ ps。

4.7.3 [10]<4.4>sd 指令的延迟是多少? 仔细检查你的答案，许多学生会在关键路径上添加额外的寄存器。

前三个步骤仍然与之前一样，在 MEM 中，只需要访问 D-Mem，即 250 ps，并且没有 WB 步骤，所以延迟为 $325 + 150 + 200 + 250 = 925$ ps。

4.7.4 [5]<4.4> beq 指令的延迟是多少?

前三个步骤仍然与之前一样，只需要在 MEM 中加入一个 Single gate 的延迟，并且没有 WB 步骤。



因此，延迟为 $325 + 150 + 200 + 5 = 680$ ps。

4.7.5 [5]<4.4>I 型指令的延迟是多少?

与 R 型指令类似，I 型指令只是在 ID 阶段需要在 MUX 前加入 Sign extend 的延迟，但仍然没有 Register File 的延迟大，所以 ID 步骤仍然需要 150 ps 的延迟，所以总延迟仍然为 720 ps。

4.7.6 [5]<4.4> 该 CPU 的最小时钟周期是多少?

由于延迟最长的指令为 ld 指令，所以该 CPU 的最小时钟周期为 ld 指令的延迟，即 970 ps。

4.8 [10]<4.4> 假设你能设计一款处理器并让每条指令执行不同的周期数。给定指令比例如下表所示，相比图 4-21 中的处理器，这款新处理器的加速比是多少?

R-type/I-type (non-ld)	ld	sd	beq
52%	25%	11%	12%

R 型指令和 I 型指令没有 MEM 阶段，ld 指令 5 个阶段都有，sd 指令没有 WB 阶段，beq 指令没有 MEM 和 WB 阶段。可以认为一个阶段的执行需要一个时钟周期，如果所有指令执行相同的周期数，那么都需要 5 个时钟周期，而如果每条指令可以执行不同的周期数，那么各类指令的周期数之比为 R/I:4/5；ld:1；sd:4/5；beq:3/5。将各类指令的周期比按照指令比例加权平均即可得到周期比：

$$\frac{4}{5}\times 0.52+1\times 0.25+\frac{4}{5}\times 0.11+\frac{3}{5}\times 0.12=0.826$$

取倒数即可得到加速比：$\frac{1}{0.826}=\frac{500}{413}\approx 1.21065375302663$

4.16 在本题中将讨论流水线如何影响处理器的时钟周期。假设数据通路的各个流水级的延迟如下:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

同时，假设处理器执行的指令分布如下：

ALU/Logic	Jump/Branch	Load	Store
45%	20%	20%	15%

4.16.1 [5]<4.5> 在流水化和非流水的处理器中，时钟周期分别是多少?

流水化的处理器：按照最长的阶段，即 350ps；
非流水化的处理器：所有阶段延迟之和，即 $250+350+150+300+200=1250$ ps。

4.16.2 [10]<4.5 > 在流水化和非流水的处理器中，对于 ld 指令的延迟分别是多少?

ld 指令 5 个阶段都有，在流水化的处理器中，为 $350\times 5=1750$ ps；
非流水化的处理器中，为所有阶段延迟之和，即 $250+350+150+300+200=1250$ ps。

4.16.3 [10]<4.5> 如果我们将数据通路中的一个流水级拆成两个新流水级，每一个新流水级的延迟是原来的一半，那么我们将拆分哪一级？新处理器的时钟周期是多少？

应该拆分最长的一级，即 ID 阶段，拆分之后最长的阶段延迟为 300 ps，所以新处理器的时钟周期是 300 ps。

4.16.4 [10]<4.5> 假设没有停顿或冒险，数据存储的利用率如何？

既然题目中出现了“停顿”“冒险”这种只有在流水化处理器中才会出现的情况，那么说明这里只需要考虑流水化的情况。数据存储对应的是 MEM 阶段，MEM 阶段只需要 300ps，但是为了满足流水线，延迟到了 350ps，所以 MEM 阶段的利用率为 $\frac{300}{350}$；而在题目给出的指令分布中，只有 Load 和 Store 指令会用到数据存储，即 $20\%+15\%$，所以数据存储的利用率为

$$\frac{300}{350}\times(0.2+0.15)=0.3$$

4.16.5 [10]<4.5> 假设没有停顿或冒险，寄存器堆的写口利用率如何？

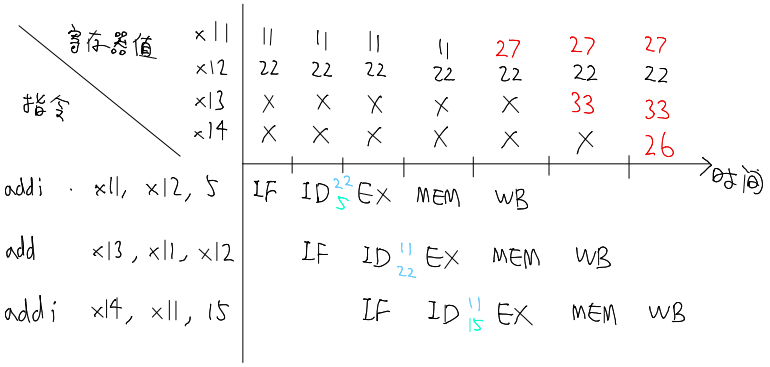
寄存器堆的写口对应的是 WB 阶段，WB 阶段的利用率为 $\frac{200}{350}$；在题目给出的指令分布中，使用到寄存器堆的写口的指令为 ALU/Logic 和 Load，即 $45\%+20\%$，所以寄存器堆写口的利用率为

$$\frac{200}{350}\times(0.45+0.2)=\frac{13}{35}\approx 0.371428571428571$$

4.18 [5]<4.5> 假设初始化寄存器 x11 为 11,x12 为 22，如果在 4.5 节中的流水线结构上执行下述代码，寄存器 x13 和 x14 中最终为何值？注：硬件不处理数据冒险，编程者需要在必要处插入 NOP 指令来解决数据冒险。

```
addi    x11, x12, 5
add     x13, x11, x12
addi    x14, x11, 15
```

显然在硬件不处理数据冒险情况下，执行上述代码会出现数据冒险，以下是示意图，在 ID 指令旁边标注了实际取出的操作数。



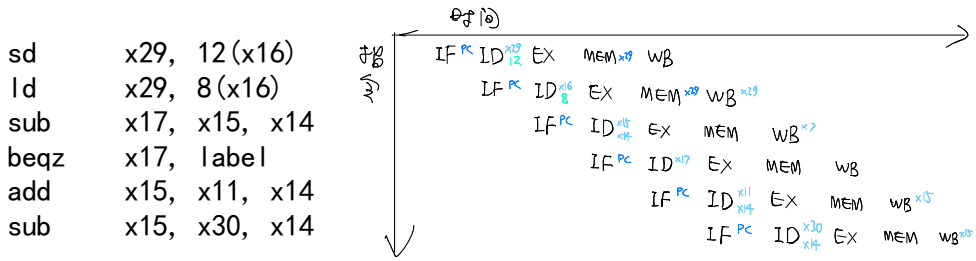
所以寄存器 x13 最终为 33，寄存器 x14 最终为 26。

4.22 [5]<4.5> 对于如下的 RISC-V 的汇编片段:

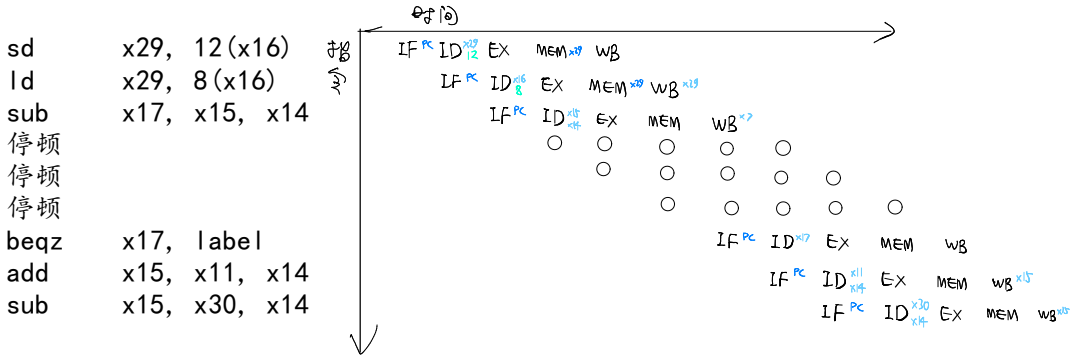
```
sd      x29, 12(x16)
ld      x29, 8(x16)
sub     x17, x15, x14
beqz    x17, label
add     x15, x11, x14
sub     x15, x30, x14
```

4.22.1 [5]<4.5> 请画出流水线图，说明以上代码会在何处停顿。

在加入停顿之前的流水线图是这样的，显然由于前面的指令的 MEM 阶段和后面的指令的 IF 阶段都需要访问存储器，会发生结构冒险。



加入停顿后流水线图变成了这样：



4.22.2 [5]<4.5> 是否可通过重排代码来减少因结构冒险而导致停顿的次数?

可以，由于只考虑结构冒险（即两条指令在同一个阶段访问寄存器堆或存储器的冒险），可以把第一行的 sd 指令放到第二三行的 ld 和 sub 指令后面，这样就可以减少一个停顿。

4.22.3 [5]<4.5> 该结构冒险必须用硬件来解决吗? 我们可以通过在代码中插入 NOP 指令来消除数据冒险，对于结构冒险是否可以相同处理? 如果可以，请解释原因。否则，也请解释原因。

不一定要用硬件来解决，可以通过插入 NOP 指令来消除结构冒险，因为 NOP 指令相当于一个停顿，只需要把上述的停顿换成 NOP 指令即可。

4.22.4 [5]<4.5> 在典型程序中，大约需要为该结构冒险产生多少个周期的停顿?(使用习题 4.8 中的指令分布)。

仔细观察可以发现，停顿的产生是由于 sd 和 ld 有 MEM 阶段，会和后续指令的 IF 阶段冲突，ld 和 R 型指令有 WB 阶段，会和后续指令的 ID 阶段冲突，那么可以认为一个 ld 导致 2 个停顿，一个 sd 导致 1 个停顿，一个 R 型导致 1 个停顿。所以大概产生的停顿周期数为：

$$1 \times 0.52 + 2 \times 0.25 + 1 \times 0.11 = 1.13$$

即产生 1.13 × 原始时钟周期数 个周期的停顿。

4.27 讨论下述指令序列，假设在一个五级流水的数据通路中执行:

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

4.27.1 [5]<4.7> 如果没有前递逻辑或者冒险检测支持，请插入 NOP 指令保证程序正确执行。

```
add     x15, x12, x11       // 在第 5 个阶段写入 x15
nop
nop
ld      x13, 4(x15)         // 在第 2 个阶段读取 x15，在第 5 个阶段写入 x13
ld      x12, 0(x2)          // 在第 2 个阶段读取 x2，在第 5 个阶段写入 x12
nop
or      x13, x15, x13       // 在第 2 个阶段读取 x13 和 x15，在第 5 个阶段写入 x13
nop
nop
sd      x13, 0(x15)         // 在第 2 个阶段读取 x13 和 x15
```

4.27.2 [10]<4.7> 对代码进行重排，插入最少的 NOP 指令。假设寄存器 x17 可用来做临时寄存器。

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

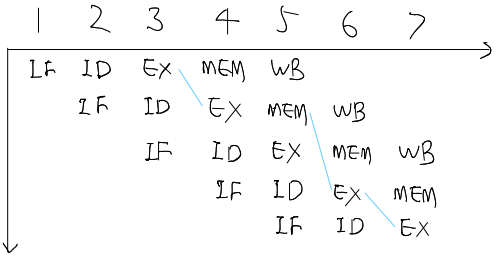
```
nop
or      x17, x15, x13
sd      x17, 0(x15)
```

4.27.3 [10]<4.7> 如果处理器中支持前递，但未实现冒险检测单元，上述代码段的执行将会发生什么?

不会发生数据冒险，因为在不存在加载后马上使用的情况，第二行加载到 x13 后在第 4 行才使用 x13，此时已经可以使用前递确保正确执行。

4.27.4 [20]<4.7> 以图 4-58 中的冒险检测和前递单元为例，如果执行上述代码，在前 7 个时钟周期中，每周期哪些信号会被它们置为有效?

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



前递信号如图所示。所以在前 7 个时钟周期中，有效信号表示如下表:

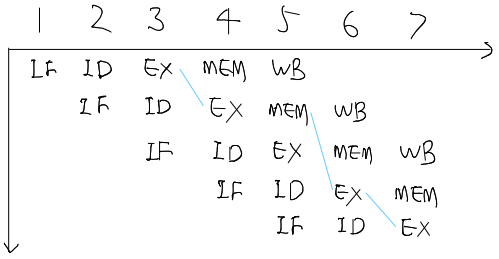
时钟周期	冒险检测	ForwardA	ForwardB
1	x	x	x
2	x	x	x
3	有效	有效	x
4	x	x	x
5	有效	x	有效
6	有效	有效	x
7	x	x	x

4.27.5 [10]<4.7> 如果没有前递单元，以图 4-58 中的冒险检测逻辑为例，需要为其增加哪些输入和输出信号?

第 3 个周期的前递，需要 IF/ID.Rs1 和 ID/EX.Rd 的输入信号，ForwardA 的输出信号；
第 5 个周期的前递，需要 IF/ID.Rs2 和 EX/MEM.Rd 的输入信号，ForwardB 的输出信号；
第 6 个周期的前递，需要 IF/ID.Rs1 和 ID/EX.Rd 的输入信号，ForwardA 的输出信号。
综上所述，需要增加 IF/ID.Rs1, IF/ID.Rs2, ID/EX.Rd, EX/MEM.Rd 的输入信号，ForwardA, ForwardB 的输出信号。

4.27.6 [20]<4.7> 如果使用习题 4.26.5 中的冒险检测单元，执行上述代码，在前 5 个时钟周期中, 每个周期哪些输出信号会有效?

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



前递信号如图所示。所以在前 5 个时钟周期中，有效信号表示如下表：

时钟周期	冒险检测	ForwardA	ForwardB
1	x	x	x
2	x	x	x
3	有效	有效	x
4	x	x	x
5	有效	x	有效