Ch4

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

Pipelined: 350ps

Non-pipelined: 1250ps

4.16.2

Pipelined: 1250ps

Non-pipelined: 1250ps

4.16.3

Split the ID stage. This reduces the clock-cycle time to 300ps.

4.16.4

35%

4.16.5

65%

4.17

在一个k级流水的CPU上执行n条指令，最少需要多少周期？证明你的公式。

Answer: n+k-1

Let's look at when each instruction is in the WB stage. In a k-stage pipeline, the 1st instruction doesn’t enter the WB stage until cycle k. From that point on, at most one of the remaining n − 1 instructions is in the WB stage during every cycle.
Th is gives us a minimum of k + (n − 1) = n + k − 1 cycles.

4.21

假设软件优化后，n条指令的典型程序需要额外添加0.4n条nop指令才能正确处理数据冒险。

4.21.1

Pipeline without forwarding requires $1.4n \times 250ps$, while pipeline with forwarding requires $1.05n \times 300ps$, The speedup is therefore $\frac{1.4 \times 250}{1.05 \times 300} = 1.11$.

4.21.2

our goal is for the pipeline with forwarding to be faster than the pipeline without forwarding. Let y be the number of stalls remaining as a percentage of "code" instructions. Our goal is for $300 \times (1 + y) \times n < 250 \times 1.4 \times n$. Thus, y must be less than 16.7%.

4.21.3

This time, our goal is for $300 \times (1 + y) \times n < 250 \times (1 + x) \times n$. This happens when $y < (250x - 50)/300$.

4.21.4

It cannot. In the best case, where forwarding eliminates the need for every NOP, the program will take time $300n$ to run on the pipeline with forwarding. This is slower than the $250 \times 1.075n$ required on the pipeline with no forwarding.

4.21.5

Speedup is not possible when the solution to 4.21.3 is less than 0. Solving $0 < (250x - 50)/300$ for x gives that x must be at least 0.2.

4.25

考虑如下循环：

```
LOOP:
ld x10, 0(x13)
ld x11, 8(x13)
add x12, x10, x11
subi x13, x13, 16
bnez x12, LOOP
```

如果使用完美的分支预测（即没有控制冒险带来的流水线停顿），流水线中没有使用延迟槽，采用硬件前递解决数据冒险，分支指令在EX阶段判断是否跳转。

4.25.1

.. indicates a stall, ! indicates a stage that does not do useful work.

ld x10, 0(x13) IF ID EX ME | WB
ld x11, 8(x13) IF ID EX | ME WB
add x12, x10, x11 IF ID | .. EX ME! WB
addi x13, x13, 16 IF | .. ID EX ME! WB

```
bnez x12, LOOP          | ..   IF  ID  EX   ME! WB!
ld x10, 0(x13)                  IF  ID   EX   ME   WB
ld x11, 8(x13)                      IF   ID   EX   ME WB
add x12, x10, x11                        IF   ID   ..   EX | ME! WB
addi x13, x13, 16                             IF   ..   ID | EX   ME! WB
bnez x12, LOOP                                          IF | ID   EX   ME! WB!
Completely busy         | N    N  N   N    N   N    N    N |
```

4.25.2

In a particular clock cycle, a pipeline stage is not doing useful work if it is stalled or if the instruction going through that stage is not doing any useful work there. As the diagram above shows, there are not any cycles during which every pipeline stage is doing useful work.

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

```
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)
```

4.27.2

It is not possible to reduce the number of NOPs.

4.27.3

The code executes correctly. We need hazard detection only to insert a stall when the instruction following a load uses the result of the load. That does not happen in this case.

4.27.4

CYCLE	1	2	3	4	5	6	7	8	
add	IF	ID	EX	ME	WB				
ld		IF	ID	EX	ME	WB			
ld			IF	ID	EX	ME	WB		
or				IF	ID	EX	ME	WB	
sd					IF	ID	EX	ME	WB

Because there are no stalls in this code, PCWrite and IF/IDWrite are always 1 and the mux before ID/EX is always set to pass the control values through.
(1) ForwardA = X; ForwardB = X (no instruction in EX stage yet)
(2) ForwardA = X; ForwardB = X (no instruction in EX stage yet)
(3) ForwardA = 0; ForwardB = 0 (no forwarding; values taken from registers)
(4) ForwardA = 2; ForwardB = 0 (base register taken from result of previous instruction)
(5) ForwardA = 1; ForwardB = 1 (base reguster taken from result of two instructions previous)
(6) ForwardA = 0; ForwardB = 2 (rs1 = x15 taken from register; rs2 = x13 taken from result of 1st ld—two instructions ago)
(7) ForwardA = 0; ForwardB = 2 (base register taken from register file. Data to be written taken from previous instruction)

4.27.5

The hazard detection unit additionally needs the values of rd that comes out of the MEM/WB register. The instruction that is currently in the ID stage needs to be stalled if it depends on a value produced by (or forwarded from) the instruction in the EX or the instruction in the MEM stage. So we need to check the destination register of these two instructions. The Hazard unit already has the value of rd from the EX/MEM register as

inputs, so we need only add the value from the MEM/WB register. No additional outputs are needed. We can stall the pipeline using the three output signals that we already have. The value of rd from EX/MEM is needed to detect the data hazard between the add and the following ld . The value of rd form MEM/WB is needed to detect the data hazard between the first ld instruction and the or instruction.

4.27.6

CYCLE	1	2	3	4	5	6
add	IF	ID	EX	ME	WB	
ld		IF	ID	-	-	EX
ld			IF	-	-	ID

(1) PCWrite = 1; IF/IDWrite = 1; control mux = 0
(2) PCWrite = 1; IF/IDWrite = 1; control mux = 0
(3) PCWrite = 1; IF/IDWrite = 1; control mux = 0
(4) PCWrite = 0; IF/IDWrite = 0; control mux = 1
(5) PCWrite = 0; IF/IDWrite = 0; control mux = 1