计算机组成原理第七次作业答案

4.21

4.21.1

不使用转发机制的流水线需要 1.4*n*250ps 的时间。使用转发机制的流水线需要 1.05*n*300ps 的时间。因此，加速比是(1.4*250)/(1.05*300) = 1.11。

4.21.2

我们的目标是让使用转发机制的流水线比不使用转发机制的流水线更快。设 y 为剩余停顿次数占“代码”指令的百分比，目标是让 300*(1+y)*n 小于 250*1.4*n。即 y 需要小于 16.7%。

4.21.3

目标是让 300(1 + y)*n 小于 250(1 + x)*n，即 y<(250x - 50)/300。

4.21.4

这是做不到的。在最佳情况下，转发机制消除了每个 NOP 的需求，程序将在使用转发机制的流水线上运行需要 300n 的时间。这比不使用转发机制的流水线所需的 250*1.075*n 还要慢。

4.21.5

当 4.21.3 的解小于 0 时，加速比是不可能的。解不等式 0 < (250x - 50)/300，得到 x 至少应为 0.2。

4.25

4.25.1

```
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

在特定的时钟周期内，如果一个流水线阶段处于停顿状态，或者通过该阶段的指令在那里没有执行任何有用的操作，那么该阶段就没有在做有用的工作。正如上面的图表所示，没有任何周期是每个流水线阶段都在进行有用工作的。

4.31.10
使用与 4.31.8 节中相同的代码，新数据通路并未提供净收益，因为不存在由于结构冲突导致的停顿。

4.32
4.32.1
两种设计的能耗是相同的：指令存储器（I-Mem）被读取，两个寄存器被读取，一个寄存器被写入。计算如下：140 皮焦耳 + 270 皮焦耳 + 60 皮焦耳 = 340 皮焦耳。
4.32.2
对于所有指令，都需要读取指令存储器。每条指令还会导致两次寄存器读取（即便只有一个值实际被使用）。加载指令会导致内存读取和寄存器写入；存储指令导致内存写入；所有其他指令最多导致一次寄存器写入。由于内存读取和寄存器写入的能耗之和大于内存写入的能耗，因此最坏情况下的指令是加载指令。加载指令的能耗为：140 皮焦耳 + 270 皮焦耳 + 60 皮焦耳 + 140 皮焦耳 = 480 皮焦耳。
4.32.3
每条指令都必须读取指令存储器。但是，我们可以避免读取那些值不会被使用的寄存器。为此，需要向寄存器单元添加 RegRead1 和 RegRead2 控制输入，以启用或禁用每个寄存器读取。必须快速生成这些控制信号以避免延长时钟周期时间。有了这些新的控制信号，加载指令只需进行一次寄存器读取（我们仍然需要读取用于生成地址的寄存器），因此每加载节省 70 皮焦耳（一个寄存器读取）。节省比例为 70/480 = 14.6%。
4.32.4
jal 指令将受益，因为它根本不需要读取任何寄存器。I 型指令也将受益，因为它们只需要读取一个寄存器。如果我们增加逻辑来检测 x0 作为源寄存器，那么像 beqz（即 beq x0, …）和 li（addi xn, x0, …）这样的指令也能受益。
4.32.5
改变之前，控制单元在进行寄存器读取的同时解码指令。改变之后，控制单元和寄存器读取的延迟不能重叠。这增加了 ID 阶段的延迟，如果 ID 阶段成为最长延迟阶段，则可能影响处理器的时钟周期时间。然而，寄存器读取（90 皮秒）和控制单元（150 皮秒）的延迟总和小于当前 250 皮秒的时钟周期时间。
4.32.6
如果每周期都读取内存，该值要么被需要（对于加载指令），要么不会通过 WB Mux（对于非加载指令且写入寄存器的情况），要么不会写入任何寄存器（包括分支和停顿在内的所有其他指令）。这个改动不影响时钟周期时间，因为时钟周期时间必须已经足够让 MEM 阶段读取内存。如果未使用的内存读取导致缓存未命中，这可能会影响整体性能。
该改动也影响能耗：每周期都进行内存读取，而不仅仅是在加载指令处于 MEM 阶段的周期。这在 250 皮秒时钟周期的 75%中增加了 140 皮焦耳的能耗。这相当于约 0.46 瓦特的消耗（不包括因缓存未命中而产生的任何能耗）。
英文原答案：
4.32.1
The energy for the two designs is the same: I-Mem is read, two registers

are read, and a register is written. We have: 140pJ + 2*70pJ + 60j = 340pJ.

4.32.2

The instruction memory is read for all instructions. Every instruction also results in two register reads (even if only one of those values is actually used). A load instruction results in a memory read and a register write; a store instruction results in a memory write; all other instructions result in at most a single register write. Because the sum of memory read and register write energy is larger than memory write energy, the worst-case instruction is a load instruction. For the energy spent by a load, we have: 140pJ + 2*70pJ + 60pJ + 140pJ = 480pJ.

4.32.3

Instruction memory must be read for every instruction. However, we can avoid reading registers whose values are not going to be used. To do this, we must add RegRead1 and RegRead2 control inputs to the Registers unit to enable or disable each register read. We must generate these control signals quickly to avoid lengthening the clock cycle time. With these new control signals, a load instruction results in only one register read (we still must read the register used to generate the address), so our change saves 70pJ (one register read) per load. This is a savings of 70/480 = 14.6%.

4.32.4

jal will benefit, because it need not read any registers at all. I-type instructions will also benefit because they need only read one register. If we add logic to detect x0 as a source register, then instructions such as beqz (i.e. beq x0, …) and li (addi xn, x0, …) could benefit as well.

4.32.5

Before the change, the Control unit decodes the instruction while register reads are happening. After the change, the latencies of Control and Register Read cannot be overlapped. This increases the latency of the ID stage and could affect the processor's clock cycle time if the ID stage becomes the longest-latency stage. However, the sum of the latencies for the register read (90ps) and control unit (150ps) are less than the current 250ps cycle time.

4.32.6

If memory is read in every cycle, the value is either needed (for a load instruction), or it does not get past the WB Mux (for a non-load instruction that writes to a register), or it does not get written to any register (all other instructions, including branches and stalls). This change does not affect clock cycle time because the clock cycle time must already allow enough time for memory to be read in the MEM stage. It can affect overall performance if the unused memory reads cause cache misses.

The change also affects energy: A memory read occurs in every cycle instead of only in cycles when a load instruction is in the MEM stage. This increases the energy consumption by 140pJ during 75% of the 250ps clock cycles. This corresponds to a consumption of approximately 0.46 Watts

(not counting any energy consumed as a result of cache misses).

4.31.1

```
                     1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
li x12, 0           IF ID EX ME WB
jal ENT             IF ID .. EX ME WB

bne x12, x13, TOP      IF .. ID EX ME WB
slli x5, x12, 3        IF .. ID .. EX ME WB

add x6, x10, x5           IF .. ID EX ME WB
ld x7, 0(x6)              IF .. ID .. EX ME WB

ld x29, 8(x6)                IF .. ID EX ME WB
sub x30, x7, x29             IF .. ID .. .. EX ME WB

add x31, x11, x5                IF .. .. ID EX ME WB
sd x30, 0(x31)                  IF .. .. ID .. EX ME WB

addi x12, x12, 2                           IF .. ID EX ME WB
bne x12, x13, TOP                          IF .. ID .. EX ME WB

slli x5, x12, 3                                  IF .. ID EX ME WB
add x6, x10, x5                                  IF .. ID .. EX ME WB

ld x7, 0(x6)                                           IF .. ID EX ME WB
ld x29, 8(x6)                                          IF .. ID .. EX ME WB

sub x30, x7, x29                                             IF .. ID .. EX ME WB
add x31, x11, x5                                             IF .. ID .. .. EX ME WB

sd x30, 0(x31)                                                     IF .. .. ID EX ME WB
addi x12, x12, 2                                                   IF .. .. ID .. EX ME WB

bne x12, x13, TOP                                                            IF .. ID EX ME WB
slli x5, x12, 3                                                              IF .. ID .. EX ME WB
```

4.31.2 The original code requires 10 cycles/loop on a 1-issue machine (stalls shown below) and 10 cycles/loop on the 2-issue machine. This gives no net speedup. (That's a terrible result considering we nearly doubled the amount of hardware.) We know that the code takes 10 cycles/iteration on the 2-issue machine because the first instruction in loop 1 (The slli) begins execution in cycle 6 and the first instruction in iteration 3 begins execution in cycle 26, so (26–6)/2 = 10.

```
    li x12,0
    jal ENT
TOP:
    slli x5, x12, 3
    add x6, x10, x5
    ld x7, 0(x6)
    ld x29, 8(x6)
    <stall>
    sub x30, x7, x29
    add x31, x11, x5
    sd x30, 0(x31)
    addi x12, x12, 2
ENT:
    bne x12, x13, TOP
```

4.31.3 Here is one possibility:

```
    beqz x13, DONE
    li x12, 0
    jal ENT
TOP:
    slli x5, x12, 3
    add x6, x10, x5
    ld x7, 0(x6)
    ld x29, 8(x6)
    addi x12, x12, 2
    sub x30, x7, x29
    add x31, x11, x5
    sd x30, 0(x31)
ENT:
    bne x12, x13, TOP
DONE:
```

If we switch to a "pointer-based" approach, we can save one cycle/loop.

The code below does this:

```
for (i = 0; i ! = j; i+ = 2) {
  *b = *a - *(a+1);
  b+=2;
  a+=2;
}
```

```
  bez x13, DONE
  li x12, 0
  jal ENT
TOP:
  ld x7, 0(x10)
  ld x29, 8(x10)
  addi x12, x12, 2
  sub x30, x7, x29
  sd x30, 0(x11)
  addi x10, x10, 16
  addi x11, x11, 16
ENT:
  bne x12,x13,TOP
DONE:
```

4.31.4 Here is one possibility:

```
  beqz x13, DONE
  li x12, 0
TOP:
  slli x5, x12, 3
  add x6, x10, x5
  ld x7, 0(x6)
  add x31, x11, x5
  ld x29, 8(x6)
  addi x12, x12, 2
  sub x30, x7, x29
  sd x30, 0(x31)
  bne x12, x13, TOP
DONE:
```

If we switch to a “pointer-based” approach, we can save one cycle/loop.

The code below does this:

```
for (i = 0; i ! = j; i+ = 2) {
   *b = *a - *(a+1);
   b+=2;
   a+=2;
}
   beqz x13, DONE
   li x12, 0
TOP:
   ld x7, 0(x6)
   addi x12, x12, 2
   ld x29, 8(x6)
   addi x6, x6, 16
   sub x30, x7, x29
   sd x30, 0(x31)
   bne x12, x13, TOP
DONE:
```

4.31.5

```
                    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
beqz x13, DONE      IF ID EX ME WB
li x12, 0           IF ID .. EX ME WB

slli x5, x12, 3        IF .. ID EX ME WB
add x6, x10, x5        IF .. ID .. EX ME WB

ld x7, 0(x6)                 IF .. ID EX ME WB
add x31, x11, x5             IF .. ID EX ME WB

ld x29, 8(x6)                      IF ID EX ME WB
addi x12, x12, 2                   IF ID EX ME WB

sub x30, x7, x29                      IF ID .. EX ME WB
sd x30, 0(x31)                        IF ID .. .. EX ME WB

bne x12, x13, TOP                        IF .. .. ID EX ME WB
slli x5, x12, 3                          IF .. .. ID .. EX ME WB

add x6, x10, x5                                   IF .. ID EX ME WB
ld x7, 0(x6)                                      IF .. ID .. EX ME WB

add x31, x11, x5                                        IF .. ID EX ME WB
ld x29, 8(x6)                                           IF .. ID EX ME WB

addi x12, x12, 2                                              IF ID EX ME WB
sub x30, x7, x29                                              IF ID .. EX ME WB

sd x30, 0(x31)                                                   IF .. ID EX ME WB
bne x12, x13, TOP                                                IF .. ID EX ME WB

slli x5, x12, 3                                                        IF ID EX ME WB
add x6, x10, x5                                                        IF ID .. EX ME WB
```

4.31.6 The code from 4.31.3 requires 9 cycles per iteration. The code from 4.31.5 requires 7.5 cycles per iteration. Thus, the speedup is 1.2.

4.31.7 Here is one possibility:

```
      beqz x13, DONE
      li x12, 0
TOP:
      slli x5, x12, 3
      add x6, x10, x5
      add x31, x11, x5
      ld x7, 0(x6)
      ld x29, 8(x6)
      ld x5, 16(x6)
      ld x15, 24(x6)
      addi x12, x12, 4
      sub x30, x7, x29
      sub x14, x5, x15
      sd x30, 0(x31)
      sd x14, 16(x31)
      bne x12, x13, TOP
DONE:
```

4.31.8 Here is one possibility:

```
      beqz x13, DONE
      li x12, 0
      addi x6, x10, 0
TOP:
      ld x7, 0(x6)
      add x31, x11, x5
      ld x29, 8(x6)
      addi x12, x12, 4
      ld x16, 16(x6)
      slli x5, x12, 3
      ld x15, 24(x6)
      sub x30, x7, x29
      sd x30, 0(x31)
      sub x14, x16, x15
      sd x14, 16(x31)
      add x6, x10, x5
      bne x12,x13,TOP
DONE:
```

4.31.9 The code from 4.31.7 requires 13 cycles per unrolled iteration. This is equivalent to 6.5 cycles per original iteration. The code from 4.30.4 requires 7.5 cycles per unrolled iteration. This is equivalent to 3.75 cycles per original iteration. Thus, the speedup is 1.73.

```
                          1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
beqz x13, DONE           IF ID EX ME WB
li x12, 0                IF ID .. EX ME WB

addi x6, x10, 0             IF .. ID EX ME WB
ld x7, 0(x6)                IF .. ID .. EX ME WB

add x31, x11, x5                  IF .. ID EX ME WB
ld x29, 8(x6)                     IF .. ID EX ME WB

addi x12, x12, 4                        IF ID EX ME WB
ld x16, 16(x6)                          IF ID EX ME WB

slli x5, x12, 3                            IF ID EX ME WB
ld x15, 24(x6)                             IF ID EX ME WB

sub x30, x7, x29                              IF ID EX ME WB
sd x30, 0(x31)                                IF ID .. EX ME WB

sub x14, x16, x15                                IF .. ID EX ME WB
sd x14, 16(x31)                                  IF .. ID EX ME WB

add x6, x10, x5                                        IF ID EX ME WB
bne x12,x13,TOP                                        IF ID EX ME WB

ld x7, 0(x6)                                              IF ID EX ME WB
add x31, x11, x5                                          IF ID EX ME WB

ld x29, 8(x6)                                                IF ID EX ME WB
addi x12, x12, 4                                             IF ID EX ME WB

ld x16, 16(x6)                                                  IF ID EX ME WB
slli x5, x12, 3                                                 IF ID EX ME WB

ld x15, 24(x6)                                                     IF ID EX ME WB
sub x30, x7, x29                                                   IF ID EX ME WB

sd x30, 0(x31)                                                        IF ID EX ME WB
sub x14, x16, x15                                                     IF ID .. EX ME WB

sd x14, 16(x31)                                                          IF .. ID EX ME WB
add x6, x10, x5                                                          IF .. ID EX ME WB

bne x12,x13,TOP                                                                IF ID EX ME WB
ld x7, 0(x6)                                                                   IF ID EX ME WB
```