

4.16

4.16.

4.16.1

流水线时钟周期: 350ps

非流水线时钟周期: 1250ps

4.16.2.

因为单条指令在流水线和非流水线的所消耗的时间相等, 所以都为 ~~1250ps~~ ~~1250ps~~ $350 \times 5 = 1750\text{ps}$

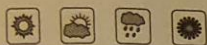
4.16.3.

应该选择延迟最大的ID级进行拆分, 拆分后时钟周期变为了 300ps

4.16.4

我们假设共有一百条指令, 所以总的时钟周期数为 $5 + 100 - 1 = 104$, 而只有lw和sw会使用数据存储器, 所以使用DM的时钟周期数为 $= 20 + 5 = 35$

$$\text{利用率} = \frac{35}{104} \times 100\% = 33.65\%$$


 Memo No. _____
 Date ____ / ____ / ____

Mo Tu We Th Fr Sa Su

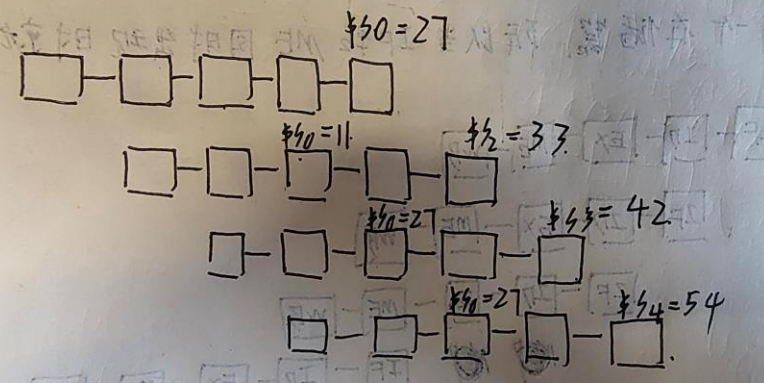
4.16.5

R型和lw要用到寄存器堆的端口, 所以使用
 周期数为 $45+20=65$.
 利用率为 $\frac{65}{104} \times 100\% = 62.5\%$

4.19

系别 _____ 班级 _____ 姓名 _____ 第 _____ 页

4.19

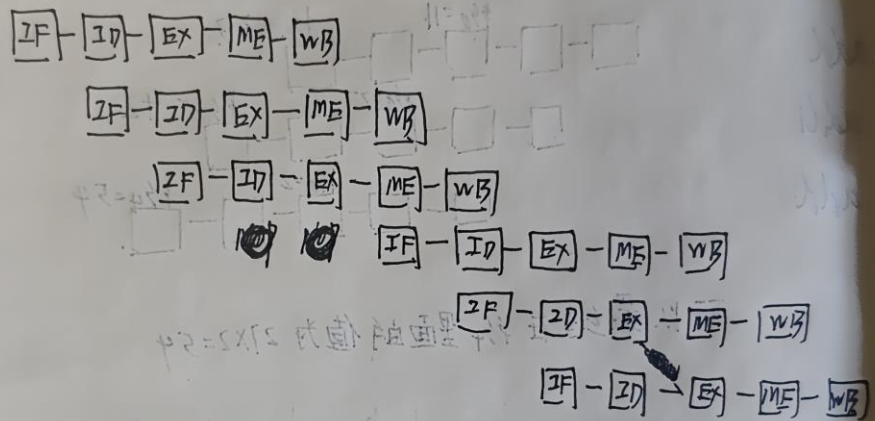
addi: 

所以最终在 s_4 里面的值为 $27 \times 2 = 54$

4.22

4.22.1

因为只有一个存储器，所以当 IF 和 ME 同时出现时就会产生阻塞。



4.22.2

重排代码并不会减少 NOP 的数量，因为每条指令都要经过五个阶段，重排代码只是会改变产生阻塞的指令对。当阻塞的类型是取数-使用型数据冒险时，重排代码是会减少 NOP，但显然本次代码并不是那种类型。

4.22.3

不可以使用 NOP 指令，因为 NOP 指令也是一个指令，我们要从存储器里面取入，所以这也会造成结构冒险。



南开大学

作业纸

系别_____ 班级_____ 姓名_____ 第_____ 页

4.22.4

因为所有指令都需要经过取指阶段,而并不是所有指令都需要访问数据,所以每有一个数据访问,就会产生一次阻塞,因为只有lw和sw会访问数据,所以阻塞的数量为 $25\% + 11\% = 36\%$

4.26

4.26.1

① EX to 1st only

add \$s0 \$t1 \$t2

add \$s3 \$s0 \$t1

add \$t4 \$t1 \$t2

② Mem to 1st only

lw \$t1 0(\$t2)

add \$t3 \$t1 \$s0

add \$t4 \$s0 \$s5

③ EX to 2nd only

add \$s0 \$t1 \$t2

add \$s3 \$t1 \$t2

add \$t4 \$s0 \$t1

④ Mem to 2nd only

lw \$s0 0(\$t2)

add \$t1 \$t3 \$t4

add \$s5 \$s0 \$t3

⑤

EX to 1st and EX to 2nd

add \$s0 \$t1 \$t2

add \$t4 \$s0 \$t1

add \$t3 \$s0 \$t2

t. 26.2.

① Ex to 1st only

add \$s0 \$s1 \$s2

NOP

NOP

add \$s3 \$s0 \$s1

add \$s4 \$s1 \$s2

② Mem to 1st only

lw \$s1 0(\$s2)

NOP

NOP

add \$s3 \$s1 \$s0

add \$s4 \$s0 \$s5

③ Ex 2nd only

add \$s0 \$s1 \$s2

add \$s3 \$s1 \$s2

NOP

add \$s4 \$s0 \$s1

④ Mem to 2nd only

lw \$s0 0(\$s2)

add \$s1 \$s3 \$s4

NOP

add \$s5 \$s0 \$s3

⑤ Ex to 1st and to 2nd

add \$s0 \$s1 \$s2

NOP

NOP

add \$s4 \$s0 \$s1

add \$s3 \$s0 \$s2



南开大学

作业纸

系别

班级

姓名

第

页

4.26.3

因为要对每条指令单独分析, 所以我们可以与两条指令都跟下
“第三条指令相关的语句, 因为这样对每条指令单独分析时会重复计
算几个NOP指令。

 $S1 = S1 \quad lw \quad r3, 0(r3)$ $add \quad r3, r4, r5$ $add \quad r0, r3, r3$

对 lw 和 add 分析时, 需要 1 和 2 条 NOP 指令, 但实际上整个
代码需要 2 个 NOP 指令即可。

4.26.4

从表中的数据和对应的 NOP 数量, 我们可以计算出:

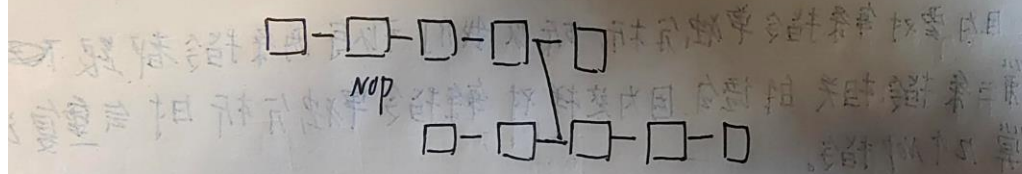
$$0.05 \times 2 + 0.2 \times 2 + 0.09 \times 1 + 0.1 \times 1 + 0.1 \times 2 = 0.85$$

也就是每条指令平均有 0.85 个 NOP 指令停顿, 所以 $CPI = 1 + 0.85 = 1.85$

$$\text{里面指令停顿的占比为 } \frac{0.85}{1.85} \times 100\% = 45.9\%$$

26.5

因为 (Ex to 1st only), (Ex to 2nd only), (Mem to 2nd only), 和 (Ex to 1st and 2nd) 都可以用旁路解决。而要想用旁路解决 (Mem to 1st only) 就要加一个 Nop, 如下图。



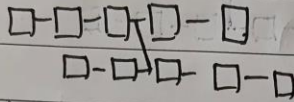
所以每条指令的平均停顿为 $0.2 \times 1 = 0.2$, $CPI = 1 + 0.2 = 1.2$
 占比为 $\frac{0.2}{1.2} \times 100\% = 16.6\%$

26.6

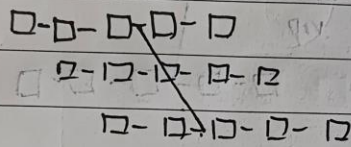
4.26.6

如果选择 ~~MEM~~ Ex/MEM.

Ex to 1st only

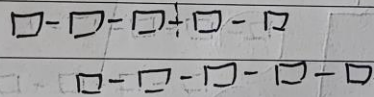


Ex to 2nd only



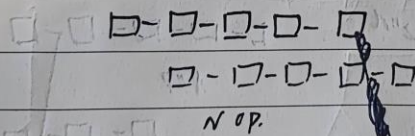
(因为只能下一时钟周期转发
所以需要 1 Nop)

Mem to 1st



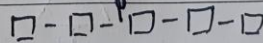
(不行, 需要 2 个 Nop)

Mem to 2nd

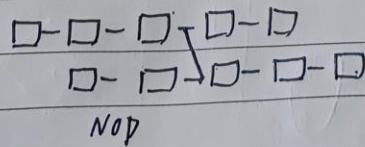


(需要 1 Nop)

Nop.

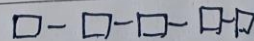


~~MEM~~ Ex to 1st and 2nd



(需要 1 Nop)

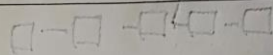
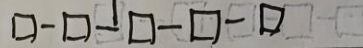
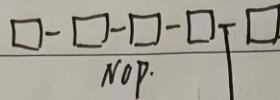
Nop



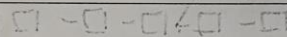
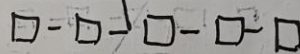
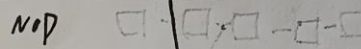
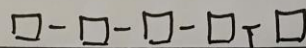
$$\therefore CPI = 1 + 0.2 \times 2 + 0.1 \times 1 + 0.05 \times 1 + 0.1 \times 1 = 1.65$$

同样地, 如果只使用 Mem/WB

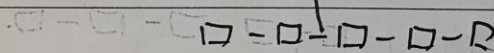
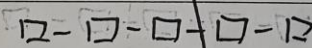
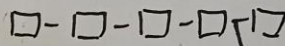
Ex to 1st



Mem to 1st



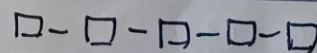
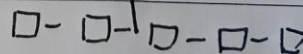
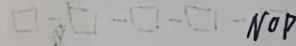
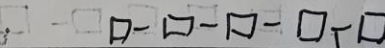
Ex to 2nd



Mem to 2nd

0 Nop

Ex to 1st and Ex to 2nd



$$CPI = 1 + 0.05 \times 1 + 0.2 \times 1 + 0.1 \times 1 = 1.35$$

4.26.7

由前面计算可得:

$$CPI(\text{无旁路}) = 1.85$$

$$CPI(\text{全旁路}) = 1.2$$

$$CPI(\text{只有 EX/MEM}) = 1.65$$

$$CPI(\text{只有 MEM/WB}) = 1.35$$

而时钟周期是取最大的那个:

$$T(\text{无旁路}) = 120\text{ps}$$

$$T(\text{全旁路}) = 130\text{ps}$$

$$T(\text{EX/MEM}) = 120\text{ps}$$

$$T(\text{MEM/WB}) = 120\text{ps}$$

因为指令数相等:

$$\text{加速比}(\text{全旁路}) = \frac{1.2 \times 130 \times n}{1.85 \times 120 \times n} = 1.42$$

$$\text{加速比}(\text{EX/MEM}) = \frac{1.85 \times 120 \times n}{1.65 \times 120 \times n} = 1.12$$

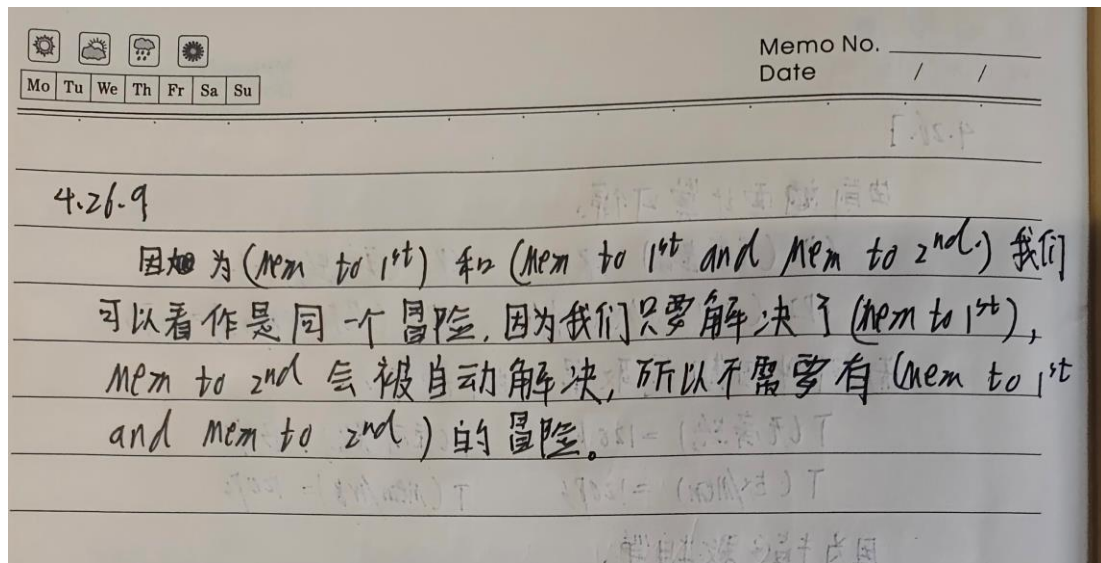
$$\text{加速比}(\text{MEM/WB}) = \frac{1.85 \times 120 \times n}{1.35 \times 120 \times n} = 1.37$$

4.26.8

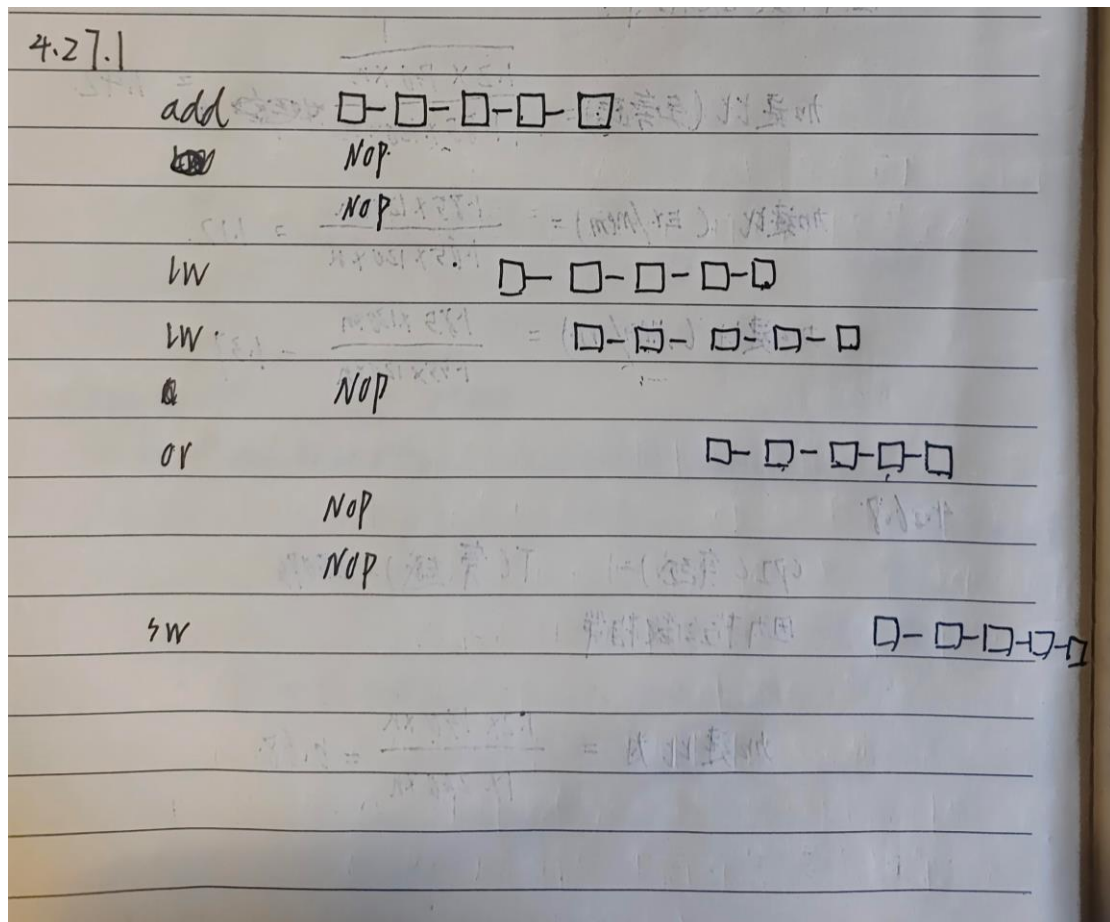
$$CPI(\text{旁路}) = 1 \quad T(\text{旁路}) = 230\text{ps}$$

因为指令数相等

$$\text{加速比为} = \frac{1.2 \times 130 \times n}{1 \times 230 \times n} = 0.68$$



4.27



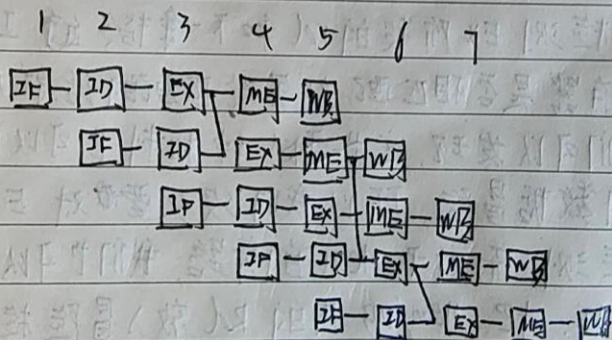
4.27.2.

试了几次后发现 没有方法去减少使用NOP的数量。

4.27.3.

指令能正常运行, 因为冒险检测单元是被用来检测 伪取数-使用数 冒险, 而在我们代码中并没有出现该冒险。

4.27.4



第一个时钟从第三个时钟周期开始讨论:

3: 因为ALU的输入都来自寄存器堆, forward A = forward B = 0

4: ALU的第一个输入来自上条的 EX/mem, 所以 forward A = 2 forward B = 0

5: 因为 ALU 的输入都来自当前寄存器堆

forward A = forward B = 0

6: 因为 ALU 的第二个输入来自上一条的运算结果

forward A = 0 forward B = 2

7: ~~因为~~ ALU 的输入都来自寄存器堆, 但是 SW 的数据

~~forward A = forward B~~

是来自上一条指令的。所以:

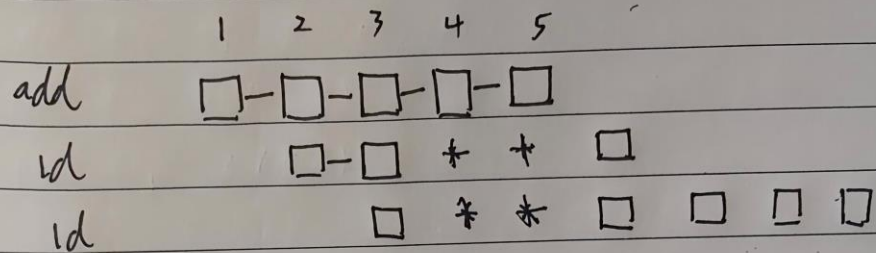
forward A = 0 forward B = 2

因为没有出现取数-使用型数据冒险, 所以冒险检测单元并不会输出到 PC 和 IF/ID 流水线寄存器, 而控制的多选器也将选择由控制单元输出的信号, 而不是全 0。

4.27.5

目前的冒险检测单元是能识别到取数-使用型冒险, 也就是检测 EX 阶段的 RD 和下一条指令 ID 阶段的某个源寄存器是否相匹配, 如果能匹配的话就加入一条 NOP。所以我们可以发现, 这也可以种机制也可以检测至 EX 旁路的数据冒险, 所以我们没必要对 EX 旁路设置新的检测, 而对于 Mem 的旁路, 我们也可以增加跟 EX 旁路一样, 把 Mem/DB 内的 RD 放入冒险检测单元, 判断跟下一条指令 ID 阶段的某个寄存器是否相等。而冒险检测器不需要增加新的输入。

4.27.6



(1) (2) (3) 都没有产生冒险, 所以

pc write = 1 IF/ID write = 1 control mux = 0

而 (4) (5) 时钟周期都产生了冒险, 所以

pc write = 0 IF/ID write = 0 control mux = 1