A.3 [20] <A.9>使用表 A-14 计算 MIPS 的实际 CPI。假定我们已经对各指令类型的平均 CPI 进行了以下测量：

指　令	时钟周期
所有ALU指令	1.0
载入–存储	1.4
条件分支	
选中	2.0
未选中	1.5
跳转	1.2
浮点乘	6.0
浮点加	4.0
浮点除	20.0
载入–存储浮点	1.5
其他浮点	2.0

假定有 60%的条件分支被选中，表 A-14 中“其他”类别的所有指令都是 ALU 指令。对 lucas 和 swim 的指令频率求平均值，以获得指令比例。

表A-14　SPECfp2000中5个程序的MIPS动态指令比例

指　令	applu	art	equake	lucas	swim	浮点均值
载入	13.8%	18.1%	22.3%	10.6%	9.1%	15%
存储	2.9%		0.8%	3.4%	1.3%	2%
加	30.4%	30.1%	17.4%	11.1%	24.4%	23%
减	2.5%		0.1%	2.1%	3.8%	2%
乘	2.3%			1.2%		1%
比较		7.4%	2.1%			2%
载入立即数	13.7%		1.0%	1.8%	9.4%	5%
条件分支	2.5%	11.5%	2.9%	0.6%	1.3%	4%
条件移动		0.3%	0.1%			0%
跳转			0.1%			0%
调用			0.7%			0%
返回			0.7%			0%
移位	0.7%		0.2%	1.9%		1%
与			0.2%	1.8%		0%
或	0.8%	1.1%	2.3%	1.0%	7.2%	2%
异或		3.2%	0.1%			1%
其他逻辑			0.1%			0%
载入浮点	11.4%	12.0%	19.7%	16.2%	16.8%	15%
存储浮点	4.2%	4.5%	2.7%	18.2%	5.0%	7%
加浮点	2.3%	4.5%	9.8%	8.2%	9.0%	7%
减浮点	2.9%		1.3%	7.6%	4.7%	3%
乘浮点	8.6%	4.1%	12.9%	9.4%	6.9%	8%
除浮点	0.3%	0.6%	0.5%		0.3%	0%
移动寄存器–寄存器浮点	0.7%	0.9%	1.2%	1.8%	0.9%	1%
比较浮点		0.9%	0.6%	0.8%		0%
条件移动浮点		0.6%		0.8%		0%
其他浮点				1.6%		0%

* 注意整数寄存器–寄存器移动指令包含在“或”指令中。空白项表示取值为 0.0%。

对表 A-14 中 lucas 和 swim 包含指令进行分类：

使用 ALU 指令的是：加、减、乘、载入立即数、移位、与、或，所占比例为：

$$\frac{11.1\%+24.4\%}{2}+\frac{2.1\%+3.8\%}{2}+\frac{1.2\%+0}{2}+\frac{1.8\%+9.4\%}{2}+\frac{1.9\%+0}{2}+\frac{1.8\%+0}{2}+\frac{1.0\%+7.2\%}{2}=32.85\%$$

使用载入-存储指令的是：载入和存储，所占比例为：

$$\frac{10.6\%+9.1\%}{2}+\frac{3.4\%+1.3\%}{2}=12.2\%$$

使用条件分支的是：条件分支，所占比例为：$\frac{0.6\%+1.3\%}{2}=0.95\%$

使用浮点乘的是：乘浮点，所占比例为 $\frac{9.4\%+6.9\%}{2}=8.15\%$

使用浮点加的是：加浮点、减浮点，所占比例为 $\frac{8.2\%+9.0\%}{2}+\frac{7.6\%+4.7\%}{2}=14.75\%$

使用浮点除的是：除浮点，所占比例为 $\frac{0+0.3\%}{2}=0.15\%$

使用载入-存储浮点的是：载入浮点、存储浮点，所占比例为 $\frac{16.2\%+16.8\%}{2}+\frac{18.2\%+5.0\%}{2}=28.1\%$

使用其它浮点的是：移动寄存器-寄存器浮点、比较浮点、条件移动浮点、其它浮点，所占比例为：

$$\frac{1.8\%+0.9\%}{2}+\frac{0.8\%+0}{2}+\frac{0.8\%+0}{2}+\frac{1.6\%+0}{2}=2.95\%$$

根据题干所给各指令时钟周期，

$$CPI=1.0\times 32.85\%+1.4\times 12.2\%+2.0\times 0.95\%\times 60\%+1.5\times 0.95\%\times 40\%+6.0\times 8.15\%$$
$$+4.0\times 14.75\%+20.0\times 0.15\%+1.5\times 28.1\%+2.0\times 2.95\%=2.1059\simeq 2.11$$

A.7 [20/20] <A.2、A.9>考虑以下 C 代码段：

```
for (i = 0; i <= 100; i++)
{ A[i] = B[i] + C; }
```

假定 A 和 B 是 64 位整数的数组，C 和 i 是 64 位整数。假定所有数据值及其地址都保存在存储器中（A、B、C、i 分别位于地址 1000、3000、5000、7000 处），但在对其进行操作时例外。假定寄存器中的值在该循环的各次迭代之间丢失。

a. [20] <A.2、A.9>写出 MIPS 的代码。动态需要多少条指令？将执行多少次存储器数据引用？代码大小为多少字节？

b. [20] <A.2>写出 x86 的代码。动态需要多少条指令？将执行多少次存储器数据引用？代码大小为多少字节？

MIPS代码：

```
      DADDI  R1, R0, R0     # i=0
      SW     R1, 7000(R0)   # 将i存入7000位置
      # 因为迭代间会丢失值，所以每次都要lw
Loop: LW     R1, 7000(R0)
      DSLL   R2, R1, 3      # 转换为字地址
      DADDI  R3, R2, 3000   # B[i]的地址
      LW     R4, 0(R3)      # B[i]
      LW     R5, 5000(R0)   # C
      DADD   R6, R5, R4     # B[i]+C
      DADDI  R7, R2, 1000   # A[i]的地址
      SW     R6, 0(R7)      # A[i]=B[i]+C
      DADDI  R1, R1, 1
      SW     R1, 7000(R0)
      SLE    R8, R1, 100
      BEQ    R8, 1, Loop
```

a. 动态指令：

2 + 12 × 101 = 1214

数据引用：

0 + 6 × 101 = 606

代码大小：

14 × 4 byte/条 = 56 byte

A.9 [10/15] <A.2>对于以下练习，假定 A、B、C、D、E 和 F 驻存在存储器中。另外假定指令操作码以 8 位表示，存储器地址为 64 位，寄存器地址为 6 位。

a. [10] <A.2>对于表 A-1 所示的每个指令集体系结构，对于计算 C=A+B 的代码，每条指令中出现多少个地址或名称？总代码大小为多少？

b. [15] <A.2>表 A-1 中的一些指令集体系结构会在计算过程中销毁操作数。这种在处理器器内部存储中丢失数据值的情况会造成性能影响。对于表 A-1 中的每种体系结构，编写代码序列，以计算：

```
C = A + B
D = A – E
F = C + D
```

在代码中，标出所有将在执行期间被销毁的操作数，有些指令的存在只是为了应对处理器内部存储器的数据丢失，也请标出所有这些“开销”指令。对于每段代码序列，总代码大小、向（自）存储器移动的指令与数据的字节数、开销指令的数量、开销数据字节的数目各为多少？ load/store

表A-1 四类指令集中C=A+B的代码序列

栈	累加器	寄存器（寄存器–存储器）	寄存器（载入–存储）
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R3, R1, B	Load R2, B
Add	Store C	Store R3, C	Add R3, R1, R2
Pop C			Store R3, C

* 注意，对于栈和累加器体系结构，Add 指令拥有隐式操作数，对于寄存器体系结构拥有显式操作数。假定 A、B 和 C 都属于存储器，A 和 B 的值不能被销毁。图 A-1 显示了针对每类体系结构的 Add 运算。

a. 栈出现3个地址或名称，总代码：(8+64)+(8+64)+8+(8+64) = 224 bit

累加器出现3个地址或名称，总代码：(8+64)+(8+64)+(8+64) = 216 bit

寄存器-存储器出现了个地址或名称，总代码：(8+6+64)+(8+6+6+64)+(8+6+64) = 240 bit

载入-存储出现 9个地址或名称，总代码：(8+6+64)+(8+6+64)+(8+6+6+6)+(8+6+64) = 260 bit

b. C=A+B; D=A-E; F=C+D

#销毁 和
#数据丢失
总代码大小
向存储器移动指令与数据

①栈：

```
Push A;
Push B;
Add;  #销毁A和B
Pop C;
Push A; #数据丢失(开销指令)
Push E;
Sub;  #销毁A和E
Pop D;
Push C; #数据丢失(开销指令)
Push D; #数据丢失(开销指令)
Add;  #销毁C和D
Pop F;
```

总代码大小：(8+64)×9+8×3=672 bit = 84byte

向存储器移动数据：64×9=576 bit = 72 byte

3条开销指令

开销指令数据大小：64×3=192 bit = 24 byte

② 累加器

```
Load A;
Add  B;    # 销毁A
Store C;
Load A;    # 销毁C且数据丢失(开销指令)
Sub  E;    # 销毁A
Store D;
Add  C;    # 销毁D
Store F;
```

总代码大小：(8+64)×8 = 576 bit = 72 byte

向存储器移动数据：64×8 = 512 bit = 64 byte

1条开销指令

开销指令数据大小：64 bit = 8 byte

③ 寄存器-存储器

（2个操作数 一个在寄存器 一个在存储器）

```
load  R1, A;
Add   R2, R1, B;
Store  R2, C;
Sub   R3, R1, E;
Store  R3, D;
Add   R4, R3, C;
Store  R4, F;
```

无数据销毁和开销指令

总代码大小：(8+64+6)×4+(8+64+6+6)×3 = 564 bit ≈ 71 byte

向存储器移动数据：64×7 = 448 bit = 56 byte

0条开销指令

开销指令数据大小：0 byte

④ 载入-存储

```
Load R1, A;
Load R2, B;
Add R3, R1, R2;
Store R3, C
Load R4, E
Sub R5, R1, R4
Store R5, D
Add R6, R3, R5
Store R6, F
```

无数据销毁和开销指令

总代码大小：$(8+6+64)\times 6+(8+6+6+6)\times 3=546$ bit $=69$ byte

向存储器移动数据：$64\times 6=384$ bit $=48$ byte

0 条开销指令

开销指令数据大小：0 byte