第一次习题课

1.12 在 1.10 节中提到使用性能公式的子集来作为性能评价指标的陷阱。下面的习题将对其进行说明。考虑下面两种处理器。P1 的时钟频率为 4GHz，平均 CPI 为 0.9，需要执行 5.0×10^9 条指令；P2 的时钟频率为 3GHz，平均 CPI 为 0.75，需要执行 1.0×10^9 条指令。

1.12.1 [5] <1.6, 1.10> 一个常见的错误是，认为时钟频率最高的计算机具有最高的性能。这种说法正确吗？请用 P1 和 P2 来验证这一说法是否正确。

1.12.2 [10] <1.6, 1.10> 另一个错误是，认为执行指令最多的处理器需要更多的 CPU 时间。考虑 P1 执行 1.0×10^9 条指令序列所需的时间，假定 P1 和 P2 的 CPI 不变，计算一下 P2 用同样的时间可以执行多少条指令。

1.12.3 [10] <1.6, 1.10> 一个常见的错误是用 MIPS（每秒百万条指令数）来比较两台不同的处理器的性能，并认为 MIPS 数值大的处理器具有最高的性能。这种说法正确吗？请用 P1 和 P2 验证这一说法是否正确。

1.12.4 [10] <1.10> 另一个常见的性能标志是 MFLOPS（每秒百万条浮点指令），其定义为

$$\text{MFLOPS} = \frac{\text{浮点操作的数目}}{\text{执行时间} \times 10^6}$$

但此指标与 MIPS 有同样的问题。假定 P1 和 P2 上执行的指令有 40% 的浮点指令，求出各处理器的 MFLOPS 数值。

1.12.1 该说法错误。
性能主要取决于 CPU 时间其计算公式为： CPU 时间 = (指令数 × CPI) / 时钟频率
计算 P1 和 P2 的执行时间： P1 的时间 = (5.0 × 10^9 × 0.9) / (4 × 10^9 Hz) = 1.125 秒
P2 的时间 = (1.0 × 10^9 × 0.75) / (3 × 10^9 Hz) = 0.25 秒
P1 的时钟频率（4GHz）比 P2（3GHz）高，但 P2 的执行时间更短，性能更好。

1.12.2
首先计算 P1 执行 1.0 × 10^9 条指令的时间： P1 时间 = (1.0 × 10^9 × 0.9) / (4 × 10^9) = 0.225 秒
设 P2 在 0.225 秒内执行的指令数为 I： (I × 0.75) / (3 × 10^9) = 0.225 秒 I × 0.75 = 0.225 × 3 × 10^9 I × 0.75 = 0.675 × 10^9 I = 0.9 × 10^9
所以 P2 可以执行 0.9 × 10^9 条指令。

1.12.3 该说法错误。

计算 P1 和 P2 的 MIPS： P1 的 MIPS = (4 × 10^9) / (0.9 × 10^6) ≈ 4444.44
P2 的 MIPS = (3 × 10^9) / (0.75 × 10^6) = 4000
结论：P2MIPS更低，性能更强

1.12.4
计算 P1： 浮点操作数 = 5.0 × 10^9 × 40% = 2.0 × 10^9 P1 的 MFLOPS = (2.0 × 10^9) / (1.125 × 10^6) ≈ 1777.78
计算 P2： 浮点操作数 = 1.0 × 10^9 × 40% = 0.4 × 10^9 P2 的 MFLOPS = (0.4 × 10^9) / (0.25 × 10^6) = 1600
P2MFLOPS更低，但是性能更强。

作业1

✓ 简答题（选一）

- 冯机架构中指令和数据都存储于存储器中，系统执行时如何区分？
- “计算机组成”与“计算机系统结构”的关系？
- 比较Amdahl’s Law和古斯塔夫森定律

1.冯架构中，指令和数据的存储地位是平等的，区分是通过执行流程进行的。
取指阶段，取出的视为指令。执行阶段，取出的视为数据。（或者通过PC指向地址区分等…）

2.用几个例子可以很好的说明这两者的区别。
计算机系统结构关心的是：指令集、寻址方式、寄存器结构、数据表示等。
计算机组成关系的是：具体功能的逻辑设计、控制信号、总线结构、存储器实现等。
可见，同样的指令集，可以用不同的硬件架构实现（流水线，非流水线）。计算机系统结构是更抽象的概念，运行在计算机组成之上。

3.开放问题。结合两个定律的基本内容作答即可。
Amdahl‘s Law聚焦于对部件的优化对于整体性能的优化。
Gustafson‘s Law 认为核的数量可以线性地增长加速比（对于可并行的部分）。

2.9 [20] <2.2, 2.5> 对于练习 2.8 中的每条 RISC-V 指令，写出操作码（op）、源寄存器（rs1）和目标寄存器（rd）字段的值。对于 I 型指令，写出立即数字段的值，对于 R 型指令，写出第二个源寄存器（rs2）的值。对于非 U 型和 UJ 型指令，写出 funct3 字段，对于 R 型和 S 型指令，写出 funct7 字段。

作业2

查表即可，这是助教查表的结果。

指令	指令类型	opcode	rd	rs1	rs2	imm	funct3	funct7
addi x30, x10, 8	I	0010011	30	10	-	8	000	-
addi x31, x10, 0	I	0010011	31	10	-	0	000	-
sd x31, 0(x30)	S	0100011	-	30	31	0	011	-
ld x30, 0(x30)	I	0000011	30	30	-	0	011	-
add x5, x30, x31	R	0110011	5	30	31	-	000	0000000

2.24 考虑以下 RISC-V 循环：

```
LOOP:  beq  x6, x0, DONE
       addi x6, x6, -1
       addi x5, x5, 2
       jal  x0, LOOP
DONE:
```

2.24.1 [5] <2.7> 假设寄存器 x6 初始化为 10。寄存器 x5 的最终值是多少（假设 x5 初始值为零）？

2.24.2 [5] <2.7> 对于上面的循环，编写等效的 C 代码。假设寄存器 x5 和 x6 分别是整型 acc 和 i。

2.24.3 [5] <2.7> 对于上面用 RISC-V 汇编语言编写的循环，假设寄存器 x6 初始化为 N。总共执行了多少条 RISC-V 指令？

2.24.4 [5] <2.7> 对于上面用 RISC-V 汇编语言编写的循环，将指令“beq x6, x0, DONE”替换为“blt x6, x0, DONE”指令并写出等效的 C 代码。

1.10*2=20。

2.
```
while (i != 0) {
        i = i - 1;
        acc = acc + 2;
}
```

3.**指令总数：**$4N + 1$

循环体执行： 循环内的 4 条指令（beq, addi, addi, jal）共执行 N 次，计 $4N$ 条。
最后一次判定： 当 x6 减为 0 时，执行最后一次 beq 分支跳转到 DONE，计 1 条。
总计： $4N + 1$ 条。

4.
```
while (i >= 0) {
 i = i - 1;
 acc = acc + 2;
}
```

2.35 考虑以下代码：

```
lb x6, 0(x7)
sd x6, 8(x7)
```

假设寄存器 x7 包含地址 0x10000000，且地址中的数据是 0x1122334455667788。

2.35.1 [5] <2.3, 2.9> 在大端对齐的机器上 0×10000008 中存储的是什么值？

2.35.2 [5] <2.3, 2.9> 在小端对齐的机器上 0×10000008 中存储的是什么值？

2.35.1 大端序情况
在大端模式下，地址 0x10000000 存储的第一个字节是最高有效位，即 0x11。
lb 指令将该字节加载到寄存器 x6。因为 0x11 是正数，符号扩展后寄存器 x6 的值为 0x0000000000000011。
sd 指令将寄存器 x6 的 64 位值存入地址 0x10000008。
所以，在大端机器上 0x10000008 中存储的值是 0x0000000000000011。
2.35.2 小端序情况
在小端模式下，地址 0x10000000 存储的第一个字节是最低有效位，即 0x88。
lb 指令将该字节加载到寄存器 x6。因为 0x88 的最高位是 1，代表负数，符号扩展后寄存器 x6 的高位全部补 1，
值为 0xFFFFFFFFFFFFFF88。
sd 指令将寄存器 x6 的 64 位值存入地址 0x10000008。
所以，在小端机器上 0x10000008 中存储的值是 0xFFFFFFFFFFFFFF88。

2.40 假设对于一个给定程序，70% 的执行指令是算术指令，10% 是加载 / 存储指令，20% 是分支指令。

2.40.1 [5] <1.6, 2.13> 假设算术指令需要 2 周期，加载 / 存储指令需要 6 周期，而一条分支指令需要 3 周期，求平均 CPI。

2.40.2 [5] <1.6, 2.13> 对于性能提高 25%，如果加载 / 存储和分支指令都没有改进，一条算术指令平均需要多少周期？

2.40.3 [5] <1.6, 2.13> 对于性能提高 50%，如果加载 / 存储和分支指令都没有改进，一条算术指令平均需要多少周期？

2.40.1
算术指令贡献为 70% 乘以 2 周期， 1.4。
加载存储指令贡献为 10% 乘以 6 周期， 0.6。
分支指令贡献为 20% 乘以 3 周期， 0.6。
全部相加得到平均 CPI 等于 2.6。 、

2.40.2 新的 CPI 等于 2.6 乘以 0.8 等于 2.08。
设新的算术指令周期为 X。 计算公式为： 70% X + 10% *6 + 20% *3 等于 2.08
得到一条算术指令平均需要约 1.26 周期。
2.40.3
性能提高 50% 意味着执行时间缩短为原来的 1 / 1.5 ≈ 0.667 倍。
新的平均 CPI = 2.6 / 1.5 ≈ 1.733 设新的算术指令周期为 Y:
(70% × Y) + (10% × 6) + (20% × 3) = 1.733
Y = 0.533 / 0.7 ≈ 0.761 所以一条算术指令平均需要约 0.76 周期。

3-27 课后思考

- 思考题（不交）：
- 以GCC为例，代码是如何转换为指令的。
- 调研：GCC的不同优化之间的区别在哪些方面
- 自行实现代码，生成指令序列，计算代码执行时间

- 源代码 (.c) → 预处理 → 编译 → 汇编 → 链接 → 可执行文件
- 几种常见的编译优化级别：
  - O0: 不进行优化
  - O1: 进行基本的优化
  - O2: 更激进的优化
  - O3: 最激进的优化

3-27 课后思考

- O0: 不进行优化
- O1: 进行基本的优化
  - 常量折叠：int x = 3 * 4; → int x = 12;
  - 基本块内指令重排
  - 删除死代码
- O2: 更激进的优化
  - 函数内联：小函数直接展开
  - 循环优化：循环展开、循环不变代码外提
  - 指令调度：重排指令减少流水线停顿
  - 公共子表达式消除
- O3: 最激进的优化
  - 自动向量化：使用SIMD指令（SSE/AVX）
  - 更激进的函数内联
  - 循环变换：循环交换、分块

3-27 课后思考

```c
int main(void) {
    int sum = 0;
    for (int i = 1; i <= 10; i++) {
        sum += i;
    }
    return 0;
}
```

```
gcc -S -O0 example.c -o example.s
```

```
  movl  $0, -8(%rbp)
  movl  $1, -4(%rbp)
  jmp .L2
.L3:
  movl  -4(%rbp), %eax
  addl  %eax, -8(%rbp)
  addl  $1, -4(%rbp)
.L2:
  cmpl  $10, -4(%rbp)
  jle .L3
  movl  $0, %eax
```

4-8 其他内容

主控制器真值表

Input or output	Signal name	R-format	ld	sd	beq
Inputs	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

$$R = \overline{I_6} \cdot I_5 \cdot I_4 \cdot \overline{I_3} \cdot \overline{I_2} \cdot I_1 \cdot I_0$$
$$ld = \overline{I_6} \cdot \overline{I_5} \cdot \overline{I_4} \cdot \overline{I_3} \cdot \overline{I_2} \cdot I_1 \cdot I_0$$
$$sd = \overline{I_6} \cdot I_5 \cdot \overline{I_4} \cdot \overline{I_3} \cdot \overline{I_2} \cdot I_1 \cdot I_0$$
$$beq = I_6 \cdot I_5 \cdot \overline{I_4} \cdot \overline{I_3} \cdot \overline{I_2} \cdot I_1 \cdot I_0$$
$$ALUSrc = ld + sd$$
$$MemtoReg = ld$$
$$RegWrite = R + ld$$
$$MemRead = ld$$
$$MemWrite = sd$$
$$Branch = beq$$
$$ALUOp1 = R$$
$$ALUOp0 = beq$$

4-8 其他内容

调研不同 FSM 控制器实现方式的特点

- Moore
  - 输出只由状态决定
  - 设计和调试更简单
  - 通常状态数更多
  - 响应相对慢一些
- Mealy
  - 输出由状态和输入共同决定
  - 响应更快
  - 通常状态数更少
  - 更容易受输入变化影响

4-8 其他内容

调研水平微指令和垂直微指令的特点

	水平微指令	垂直微指令
字长	较长	较短
编码特点	控制字段展开较多，微命令可较直接表示	控制字段压缩较多，通常需译码后产生微命令
并行度	高：一条微指令可同时发出多个互不冲突的微命令	低：微命令编码较紧凑，一条微指令能直接表示的并行操作较少
控存（CM）容量	大（位宽大，需更多存储单元）	小（位宽窄，节省控存空间）
译码电路	较简单	较复杂，需要将编码字段译成具体控制信号
执行效率	高：并行微操作多，所需微指令条数较少	较低：并行性较弱，完成同一功能通常需更多微指令