

① 简要分析 CISC 和 RISC 架构各自的优势和劣势。

CISC: 优点: 对编译器和程序存储空间要求较低。

缺点: 硬件设计复杂, 测试验证难度高。

RISC: 优点: 硬件设计较为简单, 适合利用流水线提升性能

缺点: 对编译器设计要求较高, 程序代码密度较低

② RISC-V 中的基本指令集是什么? 列举五个常见的 RISC-V 标准扩展指令集并简要说明它们的作用和应用范围。

基础指令集包含: ① RV32I (32位整数指令集) ② RV32E (32位嵌入式指令集)

③ RV64I (64位整数指令集) ④ RV128I (128位整数指令集)

扩展指令集:

① "M" (乘除法指令集) 作用: 多周期乘除运算; 有无符号乘法; 有无符号带余除法; 有无符号取余。

应用范围: 适用于需要乘除的系统, 节省指令存储。

② "A" (原子指令集) 作用: 内存原子操作; 加锁保留条件存储; 执行时对指定数据上锁;

执行时不能切换线程。

应用范围: 与内存一致性模型相匹配, 多核系统中便于信息交流。

③ "F" (单精度浮点指令集) 作用: 浮点运算和转化; 算术和逻辑运算; 浮点乘除运算; 注入类型转化。

应用范围: 适用于需要浮点数系统, 需要32个通用浮点寄存器

④ "D" (双精度浮点指令集) 作用: 应用范围同③

⑤ "C" (16位压缩指令集) 作用: 部分32位指令的16位版本; 压缩写入源寄存器的指令; 压缩小或独立立即数指令。

应用范围: 节省指令存储, 对编译无影响。

4. 阅读 RISC-V 规范以回答以下问题:

- 1) RV32I 中的 add 指令和 RV64I 中的 addw 指令均为 32 位整型加法指令, 它们是否具有相同的指令操作数 (opcode)? 此外, RV32I 中的 add 指令和 RV64I 中的 add 指令是否具有相同的指令操作数 (opcode)? 试分析为什么采取这样的设计。
- 2) 在 RV64I 中, addw 和 addiw 指令的目标寄存器中存放的 32 位计算结果是否需要进行额外的符号扩展才能用于后续 64 位计算? 请说明理由。

1) 相同; 不同; 利用这样的设计来执行针对 32 位数值所额外加法, 以确保运行性能
2) 不需要, addw 与 addiw 指令本身就会把所得结果低 32 位扩展为 64 位。

5. 什么是 RISC-V 的 I 标准指令集中存在的 HINT 指令空间? 它有什么作用?

HINT 指令通常用于向微架构可达性能提示, 除了能向 PC 以及任何可用性能计数器外, 并不改变任何体系结构可见的状态, 具体实现可选择忽略这些编码。

6. 考虑如下指令序列:

```
div    a2,a0,a1
rem    a3,a0,a1
```

假设寄存器 a0 和 a1 的初始值分别为 16 和 -5, 则上述指令序列执行完成后 a2 和 a3 寄存器中的值分别是多少? 简要说明 RISC-V 的 M 标准指令集中对除法和余数指令的符号规定。

M 标准指令集中: 除法符号: DIV(U) 有 "U" 即为无符号除法
→ 进行 XLEN 位整数除以 XLEN 位整数除法操作

余数符号: REM(U) → 进行相应取余数功能

DIV rdq, rs1, rs2.
商 被除数 除数.

REM rdr, rs1, rs2.
余数 被除数 除数.

→ a2 中值为 -3; a3 中值为 1

11. 写出以下指令使用的寻址模式。

- 1) jal ra, 0x88
- 2) jalr x0, ra, 0
- 3) addi a0, a1, 4
- 4) mul a0, a1, a2
- 5) ld a4, 16(sp)

1) $Regs[ra] = PC + 4, PC = PC + 0x88$ 偏移量寻址

2) $Regs[x0] = PC + 4, PC = ra$ 偏移量寻址

3) $Regs[a0] = Regs[a1] + 4$ 立即数寻址

4) $Regs[a0] = Regs[a1] * Regs[a2]$ 寄存器直接寻址

5) $Regs[a4] = Mem[Regs[sp] + 16]$ 偏移量寻址