**第3周作业 数据的表示**

1. 整型数据的表示

设有 short x; 除了 x=0外，x有无其他值使得 x == -x？该值是多少？

答：有， x = -32768 （即0x8000）。

2、整型数据的表示

设有C语句 int x=一个具体的数值(非变量,可含符号); 计算 x的机内表示的规则是什么（即如何得到4个字节的16进制值）？  
答：首先不看数值前的正负号（即看数值的绝对值）。将该值转换成二进制的表示形式（或者16进制的表示形式）。若不足32个二进制位，则高位补零至32位；若超过 32个二进制位，则高于32位的部分截断。

若数值前是+号，或者无符号，上面求得的结果即为 x的补码。若数值前是负号，则对前面的结果按位取反（0变1，1变0）；最后再加1，得到x的补码表示。

Intel CPU 对于整数采用小端存放模式。即 int 数（4个字节）在内存中存放在连续的4个字节单元中；数据的最高字节存放在地址最大的字节制单元中；数据的最低字节存放在地址最小的字节制单元中；（数据从高到低的4个字节依次存放在地址从大到小的4个内存字节单元中）

1. 有符号数与无符号数比较

设有 short x=0xf100; short y =0x1234; 问 x>y 是否成立？

设有 unsigned short u=0xf100; unsigned short v =0x1234; 问 u>v 是否成立？

答：

x > y 不成立。x的最高二进制位为1，对应的有符号数为负数。而y的最高二进制位为0，对应的有符号数为正数。

u > v 成立。u,v 是无符号数，高位的十六进制位大（从小到大的顺序0-9 A -F），对应的数值大。

说明：此题不需要转换成具体的十进制数；十六进制数一样比较大小。

4、字符串的表示  
 设有 char s[]=”…”; 在内存中观察数组s中存放的信息为 ：

31 32 33 67 6f 6f 64 00 (每个字节都是16进制数，31对应的字节地址最小，‘a’的ASCII是0x61)。  
 问 char s[]=”…”，引用中的字符串是什么？

答： 123good

5、浮点数的表示

给出 11.25 的单精度浮点表示（要分别给出符号位、指数部分、有效数部分的编码），以及该数在内存中的存放形式。

答；

11.25 = 1011.01B = 1.01101B\* 2^3

符号位（最高位）：正数 → 0

指数部分（8位）：实际指数 ( e = 3 )，偏移量 ( 127 ) → ( 3 + 127 = 130 ) → 二进制 10000010

有效数部分（23位）：取规格化后的尾数（去掉开头的1）并补零 → 01101000000000000000000

对应的浮点表示为 ： 0x41340000

在内存中是小端存放，十六进制形式：00 00 34 41

6、数据类型转换

设有 int x=…; float y; y = (float)x;  
 问 x == (int)y 是否（一定）成立，为什么？

答：等式 x == (int)y 不一定成立，具体原因如下：

单精度浮点数（float）的尾数部分为23位（隐含前导1，实际有效位24位）。因此，当整数x的绝对值超过224（即16777216）时，float无法精确表示该整数，会发生舍入误差。

7、数据类型转换

设有 int x; float y=…; x=(int) y;

问 y ==(float) x； 是否（一定）成立，为什么？

答：不一定成立。

| **情况** | **是否成立** | **原因** |
| --- | --- | --- |
| y为整数且在int范围内 | ✅ 成立 | 无精度丢失，转换可逆 |
| y有非零小数部分 | ❌ 不成立 | 小数截断导致值改变 |
| y超出int范围 | ❌ 不成立 | 溢出导致x为错误值 |
| y为NaN或无穷大 | ❌ 不成立 | 特殊值无法通过int转换恢复 |

8、数据类型转换

设有C语句 x=y; x是int类型的变量，y 是 unsigned short 类型的变量，计算x的机内表示的规则是什么？相应的机器指令操作符是什么？若 y 是short, 计算x的机内表示的规则又是什么？相应的机器指令操作符又是什么？   
 （指令操作符示例：mov、movsx、 movzx、 add、 sub 等等）

答：

y是unsigned short时，转换为int：零扩展，高位补0，指令如MOVZX。

y是short时，转换为int：符号扩展，高位补符号位，指令如MOVSX。

9、数据类型转换

设有C语句 x=y; x是长度长一些的整型变量，y 是长度短一些的整型变量，计算x的机内表示的规则是什么？   
答：

当将长度较短的整型变量（y）赋值给长度较长的整型变量（x）时，x的机内表示规则取决于y的符号性：

若y是无符号类型（如 unsigned short），采用零扩展，高位补零至目标类型的长度。

若y是有符号类型（如 short），符号扩展，高位填充y的最高二进制位至目标类型的长度。

10、指令流水线（中央处理器中的知识）

指令流水线与一般的工业流水线有何异同点？在使用指令流水线，存在哪些因素影响指令流水线的性能？可以采用哪些优化手段，提高指令流水线的利用率？  
  
Deepseek问答  
问：指令流水线 与工业流水线有何异同？  
答：指令流水线与工业流水线在概念上有相似之处，但也存在显著差异。以下是它们的异同点：

**相同点**

1. **分阶段处理**：
   * 两者都将任务分解为多个阶段，每个阶段专注于特定操作。
2. **并行处理**：
   * 通过流水线，多个任务可以同时在不同阶段进行处理，提升效率。
3. **提高效率**：
   * 流水线通过重叠操作减少了整体处理时间。

**不同点**

1. **应用领域**：
   * **指令流水线**：用于计算机处理器，处理指令执行。
   * **工业流水线**：用于制造业，生产物理产品。
2. **处理对象**：
   * **指令流水线**：处理指令和数据。
   * **工业流水线**：处理原材料和零部件。
3. **阶段划分**：
   * **指令流水线**：阶段包括取指、译码、执行、访存、写回等。
   * **工业流水线**：阶段根据具体产品设计，如装配、检测、包装等。
4. **错误处理**：
   * **指令流水线**：需处理数据依赖、分支预测等复杂问题。
   * **工业流水线**：需处理设备故障、质量控制等问题。
5. **灵活性**：
   * **指令流水线**：可通过软件调整以适应不同指令集。
   * **工业流水线**：调整通常需要物理改造，灵活性较低。

**总结**

指令流水线和工业流水线都通过分阶段和并行处理提升效率，但前者用于计算机指令处理，后者用于产品制造，处理对象、阶段划分、错误处理和灵活性等方面存在显著差异。

问：如何提高指令流水线的利用效率？  
答：提高指令流水线的利用效率是优化计算机处理器性能的关键。以下是一些常见的方法：

1. 减少流水线停顿（Pipeline Stalls）

数据转发（Forwarding/Bypassing）：通过直接将前一指令的结果传递给后续指令，减少因数据依赖导致的停顿。

分支预测（Branch Prediction）：通过预测分支指令的结果，提前取指和执行，减少因分支指令导致的流水线清空。

乱序执行（Out-of-Order Execution）：允许指令在不违反数据依赖的情况下乱序执行，充分利用执行单元。

2. 增加流水线深度

增加流水线的阶段数，使每个阶段的任务更简单，从而提高时钟频率。但过深的流水线可能导致更多的停顿和更高的功耗。

3. 指令调度（Instruction Scheduling）

静态调度：编译器在编译时重新排列指令，以减少数据依赖和资源冲突。

动态调度：硬件在运行时动态调整指令执行顺序，以充分利用流水线资源。

4. 超标量架构（Superscalar Architecture）

在每个时钟周期内发射多条指令到不同的执行单元，并行执行多条指令，提高吞吐量。

5. 循环展开（Loop Unrolling）

通过增加循环体内的指令数，减少循环控制指令的开销，提高流水线利用率。

6. 预取技术（Prefetching）

提前将可能需要的数据或指令加载到缓存中，减少因缓存未命中导致的停顿。

7. 多线程技术（Multithreading）

同时多线程（Simultaneous Multithreading, SMT）：通过在一个处理器核心上同时执行多个线程，隐藏单个线程的停顿时间，提高流水线利用率。

8. 优化缓存（Cache Optimization）

通过优化缓存结构和替换策略，减少因缓存未命中导致的流水线停顿。

9. 减少指令集复杂度

使用精简指令集（RISC）架构，减少指令的复杂性和执行时间，提高流水线效率。

10. 硬件支持

专用硬件单元：如专用的浮点运算单元、向量处理单元等，加速特定类型的指令执行。

多核处理器：通过多个核心并行处理任务，提高整体系统的吞吐量。

11. 编译器优化

指令选择：编译器选择更高效的指令序列。

寄存器分配：优化寄存器的使用，减少内存访问。

总结

提高指令流水线利用效率需要从硬件和软件多个层面进行优化。通过减少停顿、增加并行度、优化缓存和编译器支持等方法，可以显著提升流水线的性能和效率。