第 2 次作业

1、什么是 CPU 内的流水线？流水线的深度是指什么？可以采取哪些措施去提高流水线的效率？

CPU 内的流水线：是指 CPU 可以同时执行多条指令的不同阶段的操作。一条指令可以分解为多个阶段的操作，例如取指令码、译码、取操作数、执行、结果写回，等等。这些阶段的操作由 CPU 内不同硬件部件完成，因此第 1 条指令在译码时，取指部件处于空闲状态，所以在第 1 条指令在译码的同时，取指部件可以去读下一条指令的指令码。同理，第 1 条指令处于结果写回阶段时，CPU 可以同时执行第 2 条指令的执行操作、第 3 条指令的取操作数操作、第 4 条指令的译码、第 5 条指令的取指操作。这就是 CPU 的流水线操作。

流水线的深度：是指 CPU 能同时执行指令的不同阶段操作的指令数（多少条指令可以被同时执行）。

提高流水线的效率：(1)减少结构冒险。将流水线操作的不同阶段所需的数据分开，例如将指令码和操作数数据分开存贮。(2)减少数据冒险。例如第 2 条指令所需的操作数不能是第 1 条指令的执行结果。(3)减少转跳指令。转跳指令使得在流水线中转跳指令后面的指令操作失去作用。

2、Intel X86-64 CPU 中主要包含哪些部件？各个部件的作用是什么？并简述 CPU 执行一条指令的过程（包含 RIP 是如何变化的）。

CPU 中的主要部件：总线接口部件、存贮器管理部件（分段部件和分页部件）、指令预取部件、指令译码部件、执行部件。

各个部件的作用：总线接口部件是 CPU 与外部部件（如主存贮器）的缓冲器，存贮器管理部件用于确定所需要的主存贮单元的物理地址，指令预取部件用于从主存贮器中读取指令的机器码，指令译码部件用于分析预取部件中的指令码，执行部件根据译码部件的分析结果执行指令对应的操作。

CPU 执行一条指令的过程：CPU 根据 CS:RIP 的内容，从主存中读取一条指令的机器码，同时修改 RIP 的内容（下一条指令的偏移地址，即当前 RIP 的值加上下一条指令编码的字节数）。CPU 的译码部件对取到 CPU 的指令的机器码进行译码，然后交给执行部件执行该指令（如果是转移指令则会修正 RIP 的值）。最后 CPU 将执行结果写回主存或寄存器。

3、Intel X86-64 (即 x64) 位 CPU 中，有哪些通用的 64 位寄存器？通用的 32 位的寄存器？通用的 16 位寄存器？通用的 8 位寄存器？（只需要列出符号名即可，不用给出寄存器的中文名称）

64 位通用寄存器： rax、rbx、rcx、rdx、rbp、rsi、rdi、rsp、r8 ~ r15
32 位通用寄存器： eax、ebx、ecx、edx、ebp、esi、edi、esp、r8d ~ r15d
16 位通用寄存器： ax、 bx、cx、dx、bp、si、di、sp、r8w ~ r15w
8 位通用 寄存器： al、bl、cl、dl、bpl、sil、dil、spl、r8b ~ r15b

4、Intel X86-64 CPU 中，64 位的指令指示器 RIP 中存放的是什么？

RIP 存放的是下一条指令的偏移地址。

5、编译器在生成执行程序时，可以做哪些优化工作？为什么做相应的工作可以提高程序的执行速度？

将分支转移优化为顺序执行（提高 cache 的命中率）、尽量使用 SIMD 指令、用寄存器表示变量、尽量形成并行流水线操作。

6、已知 8 位二进制数 x1 和 x2 的值，请写出 [x1]补、[x2]补 各是多少？

[x1]补＋[x2]补 后的结果是多少，以及标志位 SF、ZF、CF、OF 各是多少？

(1) x1＝＋0110011B； x2＝＋1011010B

(2) x1＝－0101001B； x2＝－1011101B

(3) x1＝＋1100101B； x2＝－1011101B

(1) x1＝＋0110011B； x2＝＋1011010B

[x1]补 = x1 = 0011 0011

[x2]补 = x2 = 0101 1010

[x1]补 + [x1]补 = 1000 1101

SF = 1、ZF = 0、CF = 0、OF = 1

(2) x1＝－0101001B； x2＝－1011101B

x1 = －0010 1001 x2 = －0101 1101

[x1]补 = 1101 0111 [x2]补 = 1010 0011

[x1]补 + [x1]补 = 0111 1010

SF = 0、ZF = 0、CF = 1、OF = 1

(3) x1＝＋1100101B；　x2＝－1011101B

[x1]补　=　x1　=　0110 0101

x2　=　－0101 1101　　[x2]补　=　1010 0011

[x1]补 + [x1]补 = 0000 1000

SF = 0、ZF = 0、CF = 1、OF = 0

5、对如下 C语言程序，用VS2019（Intel CPU，x86-debug）编译、链接、调试运行。

```
int main( )
{
    int  a = 100;    //0x64
    int  b = 0x12345678;
    int  r = 0;
    char msg[6] = "abcde";  // 'a'的ASCII是 0x61
    return 0;
}
```

在 return 处设置断点调试时，在监视窗口中看到变量 a 的地址（即 &a）为 0x010ffe98；变量 b 的地址（即 &b）为 0x010ffe94；变量 r 的地址为 0x010ffe90，数组 msg 的起始地址为 0x010ffe88。

以字节为单位、用 16 进制数的形式填空，最左边是内存窗口显示的内存地址。

0x010ffe88　**_61_　_62_　_63_　_64_　_65_　_00_**　XX　XX

0x010ffe90　**_00_　_00_　_00_　_00_　_78_　_56_　_34_　_12_**

0x010ffe98　**_64_　_00_　_00_　_00_**　XX　XX　XX　XX

说明：若同学们要实验，看到变量比较紧凑的存放，可以设置编译开关：【项目属性-> C/C++ -> 代码生成 -> 基本运行时检查　设置为 默认值】【整个平台是 x86，是 32 位地址】

6、整数数据的表示

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

答案：不妨假设 x>=0。要使得 x = -x，必须 x = [-x]补，记 x 的 4 个 16 进制位为 X_1X_2 X_3X_4，即

X = X_1X_2 X_3X_4

那么，[-x]补 = (15 - X_1) (15 - X_2) (15 - X_3) (15 - X_4) + 1

即： X_1X_2 X_3X_4 = (15 - X_1) (15 - X_2) (15 - X_3) (15 - X_4) + 1

情况 1： (15 - X_4) + 1 不产生进位，那么 X_4 = (15 - X_4) + 1 得到 X_4 = 8，

这时必须 X_3 = 15 - X_3，即 X_3 = 7.5。显然这种情况是错误的。

情况 2： (15 - X_4) + 1 产生进位，那么(15 - X_4) + 1 一定为 0**（why?）**，

因此 (15 - X_4) = F，从而 X_4 = 0（X_4 = (15 - X_4) + 1 = 0）。

最低 4 位产生进位后，使得次低 4 位加 1，即 (15 - X_3) + 1，同理得到 X_3 = 0**（why?）。**

同样地，得到 X_2 = 0。

对于 X_1，必须满足 X_1 = (15 – X_1) + 1，即 X_1 = 8 或者 X_1 = 0。

于是，x = [-x]补的条件是 x = 0x8000 或者 x = 0x0000。

7、有符号数与无符号数

(1) 设有 short x = 0xf100; short y = 0x1234; 问 x > y 是否成立？说明理由。

(2) 设有 unsigned short u = 0xf100; unsigned short v = 0x1234; 问 u > v 是否成立？说明理由。

(3) 设有 unsigned short m = 0xf100; short n = 0x1234; 问 m > n 是否成立？说明理由。

(1) x > y 不成立，因为 x 是负数（将 0xf100 解释为有符号数时表示负数），y 是正数。

(2) u > v 成立，因为 0xf100 和 0x1234 都被解释为无符号数。

(3) m > n 成立，因为编译器会将低类型 short（有符号数）0x1234 强制转换为高类型（无符号数）。

8、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？

若有 x =(int) y; y ==(float) x； 是否（一定）成立，为什么？

x == (int)y 不一定成立，因为 y = (float)x 可能会产生精度损失，(int) y 也可能会产生精度损失。

x =(int) y; y ==(float) x 不一定成立，理由同上（float 转 int、int 转 float 都有可能产生损失）。

9、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)。

问 char s[] = "…"，引用中的字符串是什么？

"123good"

10、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。

11.25 D = 1011.01 B = 1.01101 * 2^3

阶码 = 3 + 127 = 130 D = 82 H

内存表示： 0 1000 0010 01101 00 0000 0000 0000 0000 B

11、为什么 float 数有+0 和-0？如何判断一个 float 变量的值是+0 还是-0？

IEEE754 表示 float 数时，由于精度问题，不能精确表示 0。所以 IEEE754 将无法表示的很小负数记为-0，将无法表示的很小正数记为+0。

12、编写一个 C 语言函数 int IsPNZeroNan(float x)，如果 x 是 NaN、+0、-0 时则返回 1、2、3，否则返回 0。

参考答案：

```
int IsPNZeroNan(float x)
{
        unsigned int *p = (unsigned int *)&x;
        if( (*p & 0b00000000011111111111111111111111) != 0 &&
 (*p & 0b01111111100000000000000000000000) == 0b01111111100000000000000000000000 )
            return 1;  //NaN
        else if(*p == 0B00000000000000000000000000000000) return 2;  //+0
        else if(*p == 0B10000000000000000000000000000000) return 3;  //-0
        else return 0;
}
```

测试程序如下：

```
int main()
{
        float x = 0.0f;
        float y = 0.0f;
        float z = x / y;
        int i = IsPNZeroNan(x / y);             // i = 1
```

```
    int j = IsPNZeroNan(1.0e-46f);        // j = 2
    int k = IsPNZeroNan(-1.0e-46f);       // k = 3
    int m = IsPNZeroNan(1.0e-40f);        // m = 0
}
```

13、假设：

float a = 65536; //0x10000

float b;

求满足 b>a 的条件下, IEEE754 能表示的最小 b。

a = 65536 = 0x10000 = 1 0000 0000 0000 0000 B = 1.0 * 2 ^ 16

阶码 = 16 + 127 = 143 D = 8F H

内存表示： 0 1000 1111 0000 0000 0000 0000 0000 000 B

大于 a 的最小数 b 的内存表示为： 0 1000 1111 0000 0000 0000 0000 0000 001 B

即 b = 1.00…001 * 2 ^ 16 (1.00…001, 小数点后面 22 个 0)