1. 假定机器M的时钟频率为1.2GHz，某程序P在机器M上的执行时间为12秒钟。对P优化时，将其所有的乘4指令都换成了一条左移2位的指令，得到优化后的程序P’。已知在M上乘法指令的CPI为5，左移指令的CPI为2，P的执行时间是P’执行时间的1.2倍，则P中有多少条乘法指令被替换成了左移指令被执行？

参考答案：

显然，P’的执行时间为10秒，因此，P比P’多花了2秒钟，因此，执行时被换成左移指令的乘法指令的条数为1.2G×2/(5–2) = 800M**。**

1. 图形处理器中经常需要的一种转换是求平方根。浮点（FP）平方根的实现在性能方面有很大差异，特别是在为图形设计的处理器中，尤为明显。假设FP平方根（FPSQR）占用一项关键图形基准测试中20%的执行时间。有一项提议：升级FPSQR硬件，使这一运算速度提高到原来的10倍。另一项提议是让图形处理器中所有FP指令的运行速度提高到原来的1.6倍，FP指令占用该应用程序一半的执行时间。设计团队相信，他们使所有FP指令执行速度提高到1.6倍所需要的工作量与加快平方根运算的工作量相同。试比较这两种设计方案。

根据Amdahl‘s Law，两种方案的加速比分别为：

①1/((1-0.2)+0.2/10)=1/0.82=1.22

②1/((1-0.5)+0.5/1.6)=1/0.8125=1.23

因此，第二种方案的性能要稍好一些

1. 若有两个基准测试程序P1和P2在机器M1和M2上运行，下表给出了P1和P2在M1和M2上所花的时间和指令条数。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 程序 | M1 | | M2 | |
| 指令条数 | 执行时间(ms) | 指令条数 | 执行时间(ms) |
| P1 | 200×106 | 10000 | 150×106 | 5000 |
| P2 | 300×103 | 3 | 420×103 | 6 |

请回答下列问题：

1. 对于P1，哪台机器的速度快？快多少？对于P2呢？
2. 在M1上执行P1和P2的速度分别是多少MIPS？在M2上的执行速度又各是多少？从执行速度来看，对于P2，哪台机器的速度快？快多少？
3. 假定M1和M2的时钟频率各是800MHz和1.2GHz，则在M1和M2上执行P1时的平均时钟周期数CPI各是多少？

参考答案：

1. 对于P1，M2比M1快一倍；对于P2，M1比M2快一倍。
2. 对于M1，P1的速度为：200M/10=20MIPS；P2为300k/0.003=100MIPS。

对于M2，P1的速度为：150M/5=30MIPS；P2为420k/0.006=70MIPS。

从执行速度来看，对于P2，因为100/70=1.43倍，所以M1比M2快0.43倍。

1. 在M1上执行P1时的平均时钟周期数CPI为：10×800M/(200×106)=40。

在M2上执行P1时的平均时钟周期数CPI为：5×1.2G/(150×106)=40。

1. 假设我们在对有符号值使用补码运算的32位机器上运行代码。对于有符号值使用的是算术右移，对无符号值使用的是逻辑右移。变量的声明和初始化如下：

int x = foo(); //调用某某函数，给x赋值

int y = bar(); //调用某某函数，给y赋值

unsigned ux = x;

unsigned uy = y;

对于下面每个C表达式

证明对于所有的x和y 值，都为真（等于1）；或者（2）给出使得它为假的x和y值；

1. (x>0) || (x-1<0)
2. (x&7) != 7 || (x<<29<0)
3. (x \* x) >= 0
4. x<0 || -x<=0
5. x>0 || -x>=0
6. x+y == uy+ux
7. x\*~y + uy\*ux == -x
8. x\*4 + y\*8 == (x<<2)+(x<<3)
9. ((x>>2)<<2)<=x

A 假。设 x 等于-2147483 648（TMin32），那么，我们有 x-1 等于 2147483647（TMax32）

B 真。如果 (x&7) != 7 表达式的值为 0，那么我们必须有位 x2等于 1，当左移 29位时， 这个位将变成符号位。

C 假。当 x 为 65535（0xFFFF）时，x\*x 为-131071（0xFFFE0001）

D 真。如果 x 是非负数，则-x 是非正的

E 假。设 x 为-2 147 483 648（TMin32），那么 x 和-x 都为负数

F 真。补码和无符号数乘法有相同的位级行为，而且他们是可交换的

G 真。~y 等于-y-1。uy\*uy 等于 x\*y，因此等式左边等价于 x\*-y-x+x\*y

H 真。算术左移运算与乘法运算等价

I 真。因为右移总是向负无穷大方向取整

1. 以下是两段C语言程序代码，函数arith()是直接用C语言写的，而optarith()是对arith()函数以某个确定的M和N编译生成的机器代码反编译生成的。根据optarith()，推断函数arith()中的M和N的值各是多少？

#define M

#define N

int arith(int x, int y)

{

int result = 0;

result = x\*M+y/N;

return result;

}

int optarith(int x, int y)

{

int t = x;

x << = 4;

x -= t;

if(y < 0) y+=3;

y>>=2;

return x+y;

}

答案：

M=15;N=4

1. 假定在一个程序中定义了变量x、y和i，其中，x和y是float型变量（用IEEE754单精度浮点数表示），i是16位short型变量（用补码表示）。程序执行到某一时刻，x = –0.125、y=7.5、i=100，它们都被写到了主存（按字节编址），其地址分别是100，108和112。请分别画出在大端机器和小端机器上变量x、y和i在内存的存放位置。

参考答案：

–0.125 = –0.001B = –1.0 × 2-3

x在机器内部的机器数为：1 01111100 00…0 (BE00 0000H)

7.5= +111.1B= +1.111 × 22

y在机器内部的机器数为：0 10000001 11100…0 (40F0 0000H)

100=64+32+4=1100100B

i在机器内部表示的机器数为：0000 0000 0110 0100（0064H）

大端机 小端机

地址 内容 内容

100 BEH 00H

101 00H 00H

102 00H 00H

103 00H BEH

108 40H 00H

109 F0H 00H

110 00H F0H

111 00H 40H

112 00H 64H

113 64H 00H

1. We are running programs on a machine with the following characteristics:

* Values of type int are 32 bits. They are represented in two’s complement, and they are right shifted arithmetically. Values of type unsigned are 32 bits.
* Values of type float are represented using the 32-bit IEEE ﬂoating point format, while values of type double use the 64-bit IEEE ﬂoating point format.
* We generate arbitrary values x, y, and z, and convert them to other forms as follows:

/\* Create some arbitrary values \*/

int x = random();

int y = random();

int z = random();

/\* Convert to other forms \*/

unsigned ux = (unsigned) x;

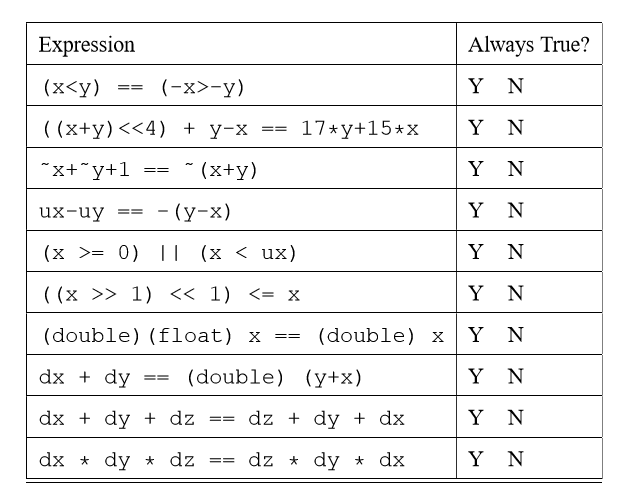
unsigned uy = (unsigned) y;

double dx = (double) x;

double dy = (double) y;

double dz = (double) z;

For each of the following C expressions, you are to indicate whether or not the expression always yields 1. If so, circle “Y”. If not, circle “N” and tell why.



Answers:

Expression Always True?

1. (x<y) == (-x>-y) No: Let x = Tmin, y = 0

2. ((x+y)<<4) + y-x == 17\*y+15\*x Yes: Associative, commutative, distributes

3. ~x+~y+1 == ~(x+y) Yes: (-x-1)+(-y-1)+1 == -(x+y)-1

4. ux-uy == -(y-x) Yes:

5. (x >= 0) || (x < ux) No: x = -1. Comparison x < ux is never true.

6. ((x >> 1) << 1) <= x Yes: x>>1 rounds toward minus infinity.

7. (double)(float) x == (double) x No: Try x = Tmax.

8. dx + dy == (double) (y+x) No: Try x=y=Tmin.

9. dx + dy + dz == dz + dy + dx Yes: Within range of exact representation by double's.

10. dx \* dy \* dz == dz \* dy \* dx No: Try dx=Tmax, dy=Tmax-1, dz=Tmax-2