第一章

6.若有两个基准测试程序 P1 和 P2 在机器 M1 和 M2 上运行，假定 M1 和 M2 的价格分别是 5000 元和 8000 元，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	200×10^6	10000	150×10^6	5000
P2	300×10^3	3	420×10^3	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？

（4）如果某个用户需要大量使用程序 P1，并且该用户主要关心系统的响应时间而不是吞吐率，那么，该用户需要大批购进机器时，应该选择 M1 还是 M2？为什么？（提示：从性价比上考虑）

（5）如果另一个用户也需要购进大批机器，但该用户使用 P1 和 P2 一样多，主要关心的也是响应时间，那么，应该选择 M1 还是 M2？为什么？

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。

（3）在 M1 上执行 P1 时的平均时钟周期数 CPI 为：$10\times800M/(200\times10^6)=40$。
在 M2 上执行 P1 时的平均时钟周期数 CPI 为：$5\times1.2G/(150\times10^6)=40$。

（4）考虑运行 P1 时 M1 和 M2 的性价比，因为该用户主要关心系统的响应时间，所以性价比中的性能应考虑执行时间，其性能为执行时间的倒数。故性价比 R 为：

R=1/(执行时间×价格)

R 越大说明性价比越高，也即，“执行时间×价格”的值越小，则性价比越高。
因为 10×5000＞5×8000，所以，M2 的性价比高。应选择 M2。

（5）P1 和 P2 需要同等考虑，性能有多种方式：执行时间总和、算术平均、几何平均。
若用算术平均方式，则：因为 (10+0.003)/2×5000＞(5+0.006)/2×8000，所以 M2 的性价比高，应选择 M2。
若用几何平均方式，则：因为 sqrt(10×0.003) ×5000＜sqrt(5×0.006) ×8000，所以 M1 的性价比高，应选择 M1。

7.若机器 M1 和 M2 具有相同的指令集，其时钟频率分别为 1GHz 和 1.5GHz。在指令集中有五种不同类型的指令 A~E。下表给出了在 M1 和 M2 上每类指令的平均时钟周期数 CPI。

机器	A	B	C	D	E
M1	1	2	2	3	4
M2	2	2	4	5	6

请回答下列问题：

（1）M1 和 M2 的峰值 MIPS 各是多少？

（2）假定某程序 P 的指令序列中，五类指令具有完全相同的指令条数，则在 M1 和 M2 上运行时，哪台机器更快？快多少？在 M1 和 M2 上执行程序 P 时的平均时钟周期数 CPI 各是多少？

参考答案：

（1）M1 上可以选择一段都是 A 类指令组成的程序，其峰值 MIPS 为 1000MIPS。

M2 上可以选择一段 A 和 B 类指令组成的程序，其峰值 MIPS 为 1500/2=750MIPS。

（2）5 类指令具有完全相同的指令条数，所以各占 20%。

在 M1 和 M2 上执行程序 P 时的平均时钟周期数 CPI 分别为：

M1：20%×(1+2+2+3+4)= 0.2×12 = 2.4

M2：20%×(2+2+4+5+6)= 0.2×19 = 3.8

假设程序 P 的指令条数为 N，则在 M1 和 M2 上的执行时间分别为：

M1：2.4× N×1/1G = 2.4N (ns)

M2：3.8×N×1/1.5G = 2.53 N (ns)

M1 执行 P 的速度更快，每条指令平均快 0.13ns，也即 M1 比 M2 快 0.13/2.53×100%≈5%。

（思考：如果说程序 P 在 M1 上执行比 M2 上快 (3.8–2.4)/3.8×100%= 36.8%，那么，这个结论显然是错误的。请问错在什么地方？）

8.假设同一套指令集用不同的方法设计了两种机器 M1 和 M2。机器 M1 的时钟周期为 0.8ns，机器 M2 的时钟周期为 1.2ns。某个程序 P 在机器 M1 上运行时的 CPI 为 4，在 M2 上的 CPI 为 2。对于程序 P 来说，哪台机器的执行速度更快？快多少？

参考答案：

假设程序 P 的指令条数为 N，则在 M1 和 M2 上的执行时间分别为：

M1：4 N×0.8 = 3.2N (ns)

M2：2 N×1.2 = 2.4 N (ns)

所以，M2 执行 P 的速度更快，每条指令平均快 0.8ns，比 M1 快 0.8/3.2×100%=25%。

9.假设某机器 M 的时钟频率为 4GHz，用户程序 P 在 M 上的指令条数为 8×10^9，其 CPI 为 1.25，则 P 在 M 上的执行时间是多少？若在机器 M 上从程序 P 开始启动到执行结束所需的时间是 4 秒，则 P 占用的 CPU 时间的百分比是多少？

参考答案：

程序 P 在 M 上的执行时间为：$1.25\times8\times10^9\times1/4G = 2.5$ s，从启动 P 执行开始到执行结束的总时间为 4 秒，其中 2.5 秒是 P 在 CPU 上真正的执行时间，其他时间可能执行操作系统程序或其他用户程序。

程序 P 占用的 CPU 时间的百分比为：2.5/4 = 62.5%。

10. 假定某编译器对某段高级语言程序编译生成两种不同的指令序列 S1 和 S2，在时钟频率为 500MHz 的机器 M 上运行，目标指令序列中用到的指令类型有 A、B、C 和 D 四类。四类指令在 M 上的 CPI 和两个指令序列所用的各类指令条数如下表所示。

	A	B	C	D
各指令的 CPI	1	2	3	4
S1 的指令条数	5	2	2	1

S2 的指令条数	1	1	1	5

请问：S1 和 S2 各有多少条指令？CPI 各为多少？所含的时钟周期数各为多少？执行时间各为多少？

参考答案：

S1 有 10 条指令，CPI 为 (5×1+2×2+2×3+1×4)/10=1.9，所含的时钟周期数为 10×1.9=19，执行时间为 19/500M = 38ns。

S2 有 8 条指令，CPI 为 (1×1+1×2+1×3+5×4)/8 =3.25，所含的时钟周期数为 8×3.25=26，执行时间为 26/500M = 52ns。

11.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。

7.假定一台 32 位字长的机器中带符号整数用补码表示，浮点数用 IEEE 754 标准表示，寄存器 R1 和 R2 的内容分别为 R1：0000108BH，R2：8080108BH。不同指令对寄存器进行不同的操作，因而，不同指令执行时寄存器内容对应的真值不同。假定执行下列运算指令时，操作数为寄存器 R1 和 R2 的内容，则 R1 和 R2 中操作数的真值分别为多少？

（1） 无符号数加法指令

（2） 带符号整数乘法指令

（3） 单精度浮点数减法指令

参考答案：

R1 ＝0000108BH＝0000 0000 0000 0000 0001 0000 1000 1011b

R2 ＝8080108BH＝1000 0000 1000 0000 0001 0000 1000 1011b

（1）对于无符号数加法指令，R1 和 R2 中是操作数的无符号数表示，因此，其真值分别为 R1：108BH, R2：8080108BH。

（2）对于带符号整数乘法指令，R1 和 R2 中是操作数的带符号整数补码表示，由最高位可知， R1 为正数， R2 为负数。R1 的真值为+108BH, R2 的真值为–(0111 1111 0111 1111 1110 1111 0111 0100b + 1b)＝–7F7FEF75H。

（3）对于单精度浮点数减法指令，R1 和 R2 中是操作数的 IEEE754 单精度浮点数表示。在 IEEE 754 标准中，单精度浮点数的位数为 32 位，其中包含 1 位符号位，8 位阶码，23 位尾数。

由 R1 中的内容可知，其符号位为 0，表示其为正数，阶码为 0000 0000，尾数部分为 000 0000 0001 0000 1000 1011，故其为非规格化浮点数，指数为–126，尾数中没有隐藏的 1，用十六进制表示尾数为+0.002116H，故 R1 表示的真值为+0.002116H × 10^{-126}。

由 R2 中的内容可知，其符号位为 1，表示其为负数，阶码为 0000 0001，尾数部分为 000 0000 0001 0000 1000 1011，故其为规格化浮点数，指数为 1－127＝–126，尾数中有隐藏的 1，用十六进制表示尾数为–1.002116H，故 R2 表示的真值为–1.002116H × 10^{-126}

8.假定机器 M 的字长为 32 位，用补码表示带符号整数。下表第一列给出了在机器 M 上执行的 C 语言程序中的关系表达式，请参照已有的表栏内容完成表中后三栏内容的填写。

关系表达式	运算类型	结果	说明
0 == 0U	无符号整数	1	00…0B = 00…0B
–1 < 0	有符号整数	1	11…1B (–1) < 00…0B (0)
–1 < 0U	无符号整数	0	11…1B (2^{32}–1) > 00…0B(0)
2147483647 > –2147483647 – 1	有符号整数	1	011…1B (2^{31}–1) > 100…0B (–2^{31})
2147483647U > –2147483647 – 1	无符号整数	0	011…1B (2^{31}–1) < 100…0B(2^{31})
2147483647 > (int) 2147483648U	有符号整数	1	011…1B (2^{31}–1) > 100…0B (–2^{31})
–1 > –2	有符号整数	1	11…1B (–1) > 11…10B (–2)
(unsigned) –1 > –2	无符号整数	1	11…1B (2^{32}–1) > 11…10B (2^{32}–2)

9. 在 32 位计算机中运行一个 C 语言程序， 在该程序中出现了以下变量的初值， 请写出它们对应的机器数（用十六进制表示）。

（1） int x=-32768 （2） short y=522

（3） unsigned z=65530

（4） char c=’ @’

（5） float a=-1.1

（6） double b=10.5

参考答案：

（1） FFFF8000H

（2） 020AH

（3） 0000FFFAH

（4） 40H

（5） BF8CCCCCH

（6） 40250000 00000000H

10.在 32 位计算机中运行一个 C 语言程序， 在该程序中出现了一些变量， 已知这些变量在某一时刻的机

器数（用十六进制表示） 如下， 请写出它们对应的真值。

（1） int x： FFFF0006H （2） short y： DFFCH

（3） unsigned z： FFFFFFFAH

（4） char c： 2AH

（5） float a： C4480000H （6） double b： C024800000000000H

参考答案：

（1） -65530

（2） -8196

（3） 4294967290（或 2 32 -6）

（4） 字符’ *’

（5） -800

（6） -10.25

16. 对于一个 n（n≥8）位的变量 x，请根据 C 语言中按位运算的定义，写出满足下列要求的 C 语言表达式。

(1) x 的最高有效字节不变，其余各位全变为 0。

(2) x 的最低有效字节不变，其余各位全变为 0。

(3) x 的最低有效字节全变为 0, 其余各位取反。

(4) % 的最低有效字节全变为 1，其余各位不变。

参考答案：

1. （x>>(n-8)<<(n-8)）
2. x&0xFF
3. ((x^^0Xff)>>8)<<8
4. x|0xFF

17. 以下是一个由反汇编器生成的一段针对某个小端方式处理器的机器级代码表示文本，其中， 最左边

是指令所在的存储单元地址， 冒号后面是指令的机器码， 最右边是指令的汇编语言表示，即汇编指

令。 已知反汇编输出中的机器数都采用补码表示， 请给出指令代码中划线部分表示的机器数对应的

真值。

80483d2: 81 ec b8 01 00 00 sub &0x1b8, %esp

80483d8: 8b 55 08

mov 0x8(%ebp) , %edx

80483db: 83 c2 14
add $0x14, %edx
80483de: 8b 85 58 fe ff ff mov 0xfffffe58(%ebp) , %eax
80483e4: 03 02
add (%edx) , %eax
80483e6: 89 85 74 fe ff ff mov %eax, 0xfffffe74(%ebp)
80483ec: 8b 55 08
mov 0x8(%ebp) , %edx
80483ef: 83 c2 44
add $0x44, %edx
80483f2: 8b 85 c8 fe ff ff mov 0xfffffec8(%ebp) , %eax
80483f8: 89 02
mov %eax, (%edx)
80483fa: 8b 45 10
mov 0x10(%ebp) , %eax
80483fd: 03 45 0c
add 0xc(%ebp) , %eax
8048400: 89 85 ec fe ff ff mov %eax, 0xfffffeec(%ebp)
8048406: 8b 45 08
mov 0x8(%ebp) , %eax
8048409: 83 c0 20
add $0x20, %eax

参考答案：

（1）440
（2）20
（3）-424
（4）-396
（5）68
（6）-312
（7）16
（8）12
（9）-276
（10）32

18.return strlen(str1) > strlen(str2);
19.

15.考虑以下C语言程序代码：

```
int func1 (unsigned word) { return (int) (( word <<24) >> 24); } int func2 (unsigned word) { return ( (int) word <<24 ) >> 24; }
```

假设在一个32位机器上执行这些函数，sizeof（int）=4。说明函数func1和func2的功能，填写表2.7，并给出对表中“异常”数据的说明。

参考答案：

函数func1的功能是把无符号数高24位清零（左移24位再逻辑右移24位），结果一定是正的带符号整数；而函数func2的功能是把无符号数的高24位都变成和第25位一样，因为左移24位后左边第一位变为原来的第25位，然后进行算术右移，高位补符号，即高24位都变成和原来第25位相同。

根据程序执行的结果填表（见表2.8），其中机器数用十六进制表示。

表2.8 题15中填入结果后的表

w		func1(w)		func2(w)	
机器数	值	机器数	值	机器数	值
0000007FH	127	0000007FH	+127	0000007FH	+127
00000080H	128	00000080H	+128	FFFFFF80H	-128
000000FFH	255	000000FFH	+255	FFFFFFFFH	
00000100H	256	00000000H	0	00000000H	

20.

14.填写表2.5，注意对比无符号数和带符号整数的乘法结果，以及截断操作前、后的结果。

表2.5 题14用表

模式	x		y		x*y（截断前）		x*y（截断后）	
	机器数	值	机器数	值	机器数	值	机器数	值
无符号数	110		010					
二进制补码	110		010					
无符号数	001		111					
二进制补码	001		111					
无符号数	111		111					
二进制补码	111		111					

参考答案：

表2.6 题14中填入结果后的表

模式	x		y		x·y（截断前）		x·y（截断后）	
	机器数	值	机器数	值	机器数	值	机器数	值
无符号数	110	6	010	2	**001100**	**12**	**100**	**4**
二进制补码	110	-2	010	+2	111100	-4	100	-4
无符号数	001	1	111	7	000111	7	111	7
二进制补码	001	+1	111	-1	111111	-1	111	-1
无符号数	111	7	111	7	**110001**	**49**	**001**	**1**
二进制补码	111	-1	111	-1	000001	+1	001	+1

21. 以下是两段 C 语言代码， 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( )， 可以推断函数 arith( )中 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;
}
```

参考答案：

M=15,N=4

22.下列几种情况所能表示的数的范围是什么？

（1） 16 位无符号整数

（2） 16 位原码定点小数

（3） 16 位补码定点小数

（4） 16 位补码定点整数

（5） 下述格式的浮点数（基数为 2，移码的偏置常数为 128）

数符	阶码	尾数
1 位	8 位移码	7 位原码

参考答案：

（1）无符号整数：$0 \sim 2^{16}-1$。

（2）原码定点小数：$-(1-2^{-15}) \sim +(1-2^{-15})$。

（3）补码定点小数：$-1 \sim +(1-2^{-15})$。

（4）补码定点整数：$-32768 \sim +32767$。

（5）浮点数：负数：$-(1-2^{-7})\times 2^{+127} \sim -2^{-7}\times 2^{-128}$。

正数：$+2^{-135} \sim (1-2^{-7})\times 2^{+127}$。

23. 以 IEEE 754 单精度浮点数格式表示下列十进制数。

+1.75，+19，–1/8，258

参考答案：

$+1.75 = +1.11B = 1.11B \times 2^0$，故阶码为 0+127=01111111B，数符为 0，尾数为 1.110…0，小数点前为隐藏位，所以+1.7 表示为 0 01111111 110 0000 0000 0000 0000 0000，用十六进制表示为 3FE00000H。

$+19 = +10011B = +1.0011B \times 2^4$，故阶码为 4+127 = 10000011B，数符为 0，尾数为 1.00110…0，所以+19 表示为 0 10000011 001 1000 0000 0000 0000 0000，用十六进制表示为 41980000H。

$-1/8 = -0.125 = -0.001B = -1.0 \times 2^{-3}$，阶码为–3+127 = 01111100B，数符为 1，尾数为 1.0…0，所以–1/8 表示为 1 01111100 000 0000 0000 0000 0000 0000，用十六进制表示为 BE000000H。

$258=100000010B=1.0000001B \times 2^8$，故阶码为 8+127=10000111B，数符为 0，尾数为 1.0000001，所以 258 表示为 0 10000111 000 0001 0000 0000 0000 0000，用十六进制表示为 43810000H。

24. 设一个变量的值为 4098，要求分别用 32 位补码整数和 IEEE 754 单精度浮点格式表示该变量（结果用十六进制表示），并说明哪段二进制序列在两种表示中完全相同，为什么会相同？

参考答案：

$4098 = +1\ \mathbf{0000\ 0000\ 0010}B = +1.0000\ 0000\ 001 \times 2^{12}$

32 位 2-补码形式为：0000 0000 0000 0000 0001 **0000 0000 0010** （00001002H）

IEEE754 单精度格式为：0 10001011 **0000 0000 0010** 0000 0000 000 （45801000H）

粗体部分为除隐藏位外的有效数字，因此，在两种表示中是相同的序列。

25. 设一个变量的值为–2147483647，要求分别用 32 位补码整数和 IEEE754 单精度浮点格式表示该变量（结果用十六进制表示），并说明哪种表示其值完全精确，哪种表示的是近似值。

参考答案：

$-2147483647 = -111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111B$

$= -1.11\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111 \times 2^{30}$

32 位 2-补码形式为：1000 0000 0000 0000 0000 0000 0000 0001 （80000001H）

IEEE 754 单精度格式为：1 10011101 1111 1111 1111 1111 1111 111 （CEFFFFFFH）

32 位 2-补码形式能表示精确的值，而浮点数表示的是近似值，低位被截断

26. 下表给出了有关 IEEE 754 浮点格式表示中一些重要数据的取值，表中已经有最大规格化

数的相应内容，要求填入其他浮点数的相应内容。（注：表中 a 代表一个在 1 到 10 之间的正纯小数）

项目	阶码	尾数	单精度		双精度	
			以 2 的幂次表示的值	以 10 的幂次表示的值	以 2 的幂次表示的值	以
0	00000000	0…00	0	0	0	
1	01111111	0…00	1	1	1	
最大规格化数	11111110	1…11	$(2-2^{-23})\times2^{127}$	$a\times10^{38}$	$(2-2^{-52})\times2^{1023}$	
最小规格化数	00000001	0…00	1.0×2^{-126}	$a\times10^{-38}$	1.0×2^{-1022}	
最大非规格化数	00000000	1…11	$(1-2^{-23})\times2^{-126}$	$a\times10^{-38}$	$(1-2^{-52})\times2^{-1022}$	
最小非规格化数	00000000	0…01	$2^{-23}\times2^{-126}=2^{-149}$	$a\times10^{-44}$	$2^{-52}\times2^{-1022}$	
+∞	11111111	0…00	–	–	–	
NaN	11111111	非全 0	–	–	–	

27. 已知下列字符编码：A=100 0001，a=110 0001，0=011 0000，求 E、e、f、7、G、Z、5 的 7 位 ACSII 码和第一位前加入奇校验位后的 8 位编码。

参考答案：

E 的 ASCII 码为 'A'+('E'－'A')＝100 0001＋100＝100 0101，奇校验位 P＝0，第一位前加入奇校验位后的 8 位编码是 0 100 0101。

e 的 ASCII 码为 'a'+('e'－'a')＝110 0001＋100＝110 0101，奇校验位 P＝1，第一位前加入奇校验位后的 8 位编码是 1 110 0101。

f 的 ASCII 码为 'a'+('f'－'a')＝110 0001＋101＝110 0110，奇校验位 P＝0，第一位前加入奇校验位后的 8 位编码是 1 110 0110。

7 的 ASCII 码为 '0'+(7－0)＝011 0000＋111＝011 0111，奇校验位 P＝0，第一位前加入奇校验位后的 8 位编码是 0 011 0111。

G 的 ASCII 码为 'A'+('G'－'A')＝100 0001＋0110＝100 0111，奇校验位 P＝1，第一位前加入奇校验位后的 8 位编码是 1 100 0111。

Z 的 ASCII 码为 'A'+('Z'－'A')＝100 0001＋11001＝101 1010，奇校验位 P＝1，第一位前加入奇校验位后的 8 位编码是 1 101 1010。

5 的 ASCII 码为 '0'+(5－0)＝011 0000＋101＝011 0101，奇校验位 P＝1，第一位前加入奇校验位后的 8 位编码是 1 011 0101。

28. 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 中每个字节在主存的存放位置。

参考答案：

解：

首先，将 x、y 和 i 换算成正确的机器数

（1）－0.125＝－0.001B＝－1.0 × 2-3，负浮点数，符号位为 1，阶码为-3（加偏置常数 127 后为 124），尾数为.0，所以，x 在机器内部的机器数为：1 01111100 00…0 (BE00

0000H)

（2） 7.5= +111.1B= +1.111 × 2 2 ， 正浮点数， 阶码为 2（加偏置常数 127 后为 129）尾数

为.111， 所以， y 在机器内部的机器数为： 0 10000001 11100…0 (40F0 0000H)

（3） 100=64+32+4=1100100B， 正整数， 直接转换， 所以 i 在 32 位的机器内部表示的机

器数为： 0000 0000 0110 0100（0064H）

所以 x、 y 和 i 在主存中的情况如下： （注意三个数的起始地址）

	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

29.

参考答案：

表 2.17 题 29 用表

表示	X	x	Y	y	X+Y	x+y	OF	SF	CF	X-Y	x-y	OF	SF	CF
无符号	0xB0	176	0x8C	140	0x3C	60	1	0	1	0x24	36	0	0	0
带符号	0xB0	-80	0x8C	-116	0x3C	60	1	0	1	0x24	36	0	0	0
无符号	0x7E	126	0x5D	93	0xDB	219	1	1	0	0x21	33	0	0	0
带符号	0x7E	126	0x5D	93	0xDB	-37	1	1	0	0x21	33	0	0	0

（1）无符号整数 176+140=316，无法用 8 位无符号数表示，即结果应有进位，CF 应为 1。减 256，得 60，验证正确。

（2）无符号整数 176-140=36，可用 8 位无符号数表示，即结果没有进位，CF 应为 0。验证正确。

（3）带符号整数-80+(-116)=-196，无法用 8 位有符号数表示，即结果溢出，OF 应为 1，加 256，得 60。验证正确。

（4）带符号整数-80-(-116)=36，可用 8 位有符号数表示，即结果不溢出，OF 应为 0。验证正确。

（5）无符号整数 126+93=219，可用 8 位无符号数表示，即结果没有进位，CF 应为 0。验证正确。

（6）无符号整数 126-93=33，可用 8 位无符号数表示，即结果没有进位，CF 应为 0。验证正确。

（7）带符号整数 126+93=219，无法用 8 位有符号数表示，即结果溢出，OF 应为 1。减 256，得-37，验证正确。

（8）带符号整数 126-93=33，可用 8 位有符号数表示，即结果不溢出，OF 应为 0。验证正确。

30. 某 32 位计算机上，有一个函数其原型声明为“int ch_mul_overflow(int x，int y)；”。。。。。

参考答案：

```
1 int ch_mul_overflow(int x,int y)
2 {
3 Long long prod_64 = (long long) x*y;
4 return prod_64 != (int)prod_64;
5 }
```

31. 对于第 2.7.5 节中例 2.31 存在的整数溢出漏洞，如果将其中的第 5 行改为以下两个语句：

```
unsigned long long arraysize=count*(unsigned long long)sizeof(int);
int *myarray = (int *) malloc(arraysize);
```

已知 C 语言标准库函数 malloc 的原型声明为“void *malloc(size_t size);”，其中，size_t 定义为 unsigned int 类型，则上述改动能否消除整数溢出漏洞？若能则说明理由；若不能则给出修改方案。

参考答案：

上述改动无法消除整数溢出漏洞。

分析： 这种改动方式虽然使得 arraysize 的表示范围扩大了， 避免了 arraysize 的溢出， 不过， 当调用
malloc 函数时， 若 arraysize 的值大于 32 位的 unsigned int 的最大可表示值， 则 malloc 函数还是只能按 32
位数给出的值去申请空间， 同样会发生整数溢出漏洞。
程序应该在调用 malloc 函数之前检测所申请的空间大小是否大于 32 位无符号整数的可表示范围， 若
是， 则返回-1， 表示不成功； 否则再申请空间并继续进行数组复制。
修改后的程序如下：

```
1 /* 复制数组到堆中， count 为数组元素个数 */
2 int copy_array(int *array, int count) {
3 int i;
4 /* 在堆区申请一块内存 */
5 unsigned long long arraysize=count*(unsigned long long) sizeof(int) ;
6 size_t myarraysize=(size_t) arraysize; /*将 arraysize 强制转换成 size_t*/
7 if (myarraysize! =arraysize) /*然后比较转换后的 myarraysize 和 arraysize， 如果二
者不等， 这表示转换过程有数据丢失， 原数据 arraysize 在
size_t 型范围内溢出， 此时不能申请空间， 直接退出*/
8 return -1;
9 int *myarray = (int *) malloc(myarraysize) ;
10 if (myarray == NULL)
11 return -1;
12 for (i = 0; i < count; i++)
13 myarray[i] = array[i] ;
14 return count;
15 }
```


33.

48.假设一次整数加法、一次整数减法和一次移位操作都只需一个时钟周期，一次整数乘法操作需要10个时钟周期。若x为一个整型变量，现要计算55*x，请给出一种计算表达式，使得所用时钟周期数最少。

参考答案：

根据表达式55*x=（64-8-1）*x=64*x-8*x-x可知，完成55*x只要两次移位操作和两次减法操作，共4个时钟周期。若将55分解为32+16+4+2+1，则需要4次移位操作和4次加法操作，共8个时钟周期。

34. 无符号整数变量 ux 和 uy 的声明和初始化如下：

unsigned ux=x;
unsigned uy=y;
若 sizeof(int)=4， 则对于任意 int 型变量 x 和 y， 判断以下关系表达式是否永真。
若永真则给出证明；
若不永真则给出结果为假时 x 和 y 的取值。
（1） (x*x) >= 0
（2） (x-1<0) || x>0
（3） x<0 || -x<=0
（4） x>0 || -x>=0
（5） x&0xf!=15 || (x<<28)<0
（6） x>y==(-x<-y)
（7） ~x+~y==~(x+y)
（8） (int) (ux-uy) == -(y-x)
（9） ((x>>2)<<2) <= x
（10） x*4+y*8==(x<<2)+(y<<3)
（11） x/4+y/8==(x>>2)+(y>>3)
（12） x*y==ux*uy
（13） x+y==ux+uy
（14） x*~y+ux*uy==-x
参考答案：
（1） (x*x)>=0
非永真。 例如， x=65 534 时， 则 x*x=(2 16 -2)* (2 16 -2)= 2 32 -2*2*2 16 +4 (mod 2 32)=-(2 18 -4)=-262140。
x 的机器数为 0000FFFEH， x*x 的机器数为 FFFC0004H。
（2） (x-1<0) || x>0
非永真。 当 x=-2 147 483 648 时， 显然， x<0， 机器数为 80000000H， x-1 的机器数为 7FFFFFFFH，
符号位为 0， 因而 x-1>0。 此时， (x-1<0) 和 x>0 两者都不成立。
（3） x<0 || -x<=0
永真。 若 x>0， x 符号位为 0 且数值部分为非 0（至少有一位是 1）， 从而使-x 的符号位一定是 1，
即则-x<0； 若 x=0， 则-x=0。 综上， 只要 x<0 为假， 则-x<=0 一定为真， 因而是永真。
（4） x>0 || -x>=0
非永真。 当 x=-2 147 483 648 时， x<0， 且 x 和-x 的机器数都为 80000000H， 即-x<0。 此时， x>0
和-x>=0 两者都不成立。
（5） x&0xf!=15 || (x<<28)<0
非永真。 这里!=的优先级比&（按位与） 的优先级高。 因此， 若 x=0， 则 x&0xf!=15 为 0， (x<<28)<0
也为 0， 所以结果为假。
（6） x>y==(-x<-y)
非永真。 当 x=-2 147 483 648、 y 任意（除-2 147 483 648 外）， 或者 y=-2 147 483 648、 x 任意（除

-2 147 483 648 外）时不等。因为 int 型负数-2 147 483 648 是最小负数， 该数取负后结果仍为-2 147
483 648， 而不是 2 147 483 648。
（7） ~x+~y==~(x+y)
永假。 [-x] 补 =~[x] 补 +1， [-y] 补 =~[y] 补 +1， 故~[x] 补 +~[y] 补 =[-x] 补 +[-y] 补 -2。
[-(x+y)] 补 =~[x+y] 补 +1，故~[x+y] 补 =[-(x+y)] 补 -1=[-x] 补 +[-y] 补 -1。
由此可见， 左边比右边少 1。
（8） (int) (ux-uy) ==-(y-x)
永真。 (int) ux-uy=[x-y] 补 =[x] 补 +[-y] 补 =[-y+ x] 补 = [-(y-x)] 补
（9） ((x>>2)<<2) <= x
永真。 因为右移总是向负无穷大方向取整。
（10） x*4+y*8==(x<<2)+(y<<3)
永真。 因为带符号整数 x 乘以 2 k 完全等于 x 左移 k 位， 无论结果是否溢出。
（11） x/4+y/8==(x>>2)+(y>>3)
非永真。 当 x=-1 或 y=-1 时， x/4 或 y/8 等于 0， 但是， 因为-1 的机器数为全 1， 所以， x>>2 或
y>>3 还是等于-1。 此外， 当 x 或 y 为负数且 x 不能被 4 整除或 y 不能被 8 整除，则 x/4 不等于
x>>2， y/8 不等于 y>>3。
（12） x*y==ux*uy
永真。 根据第 2.7.5 节内容可知， x*y 的低 32 位和 ux*uy 的低 32 位是完全一样的位序列。
（13） x+y==ux+uy
永真。 根据第 2.7.4 节内容可知， 带符号整数和无符号整数都是在同一个整数加减运算部件中进
行运算的， x 和 ux 具有相同的机器数， y 和 uy 具有相同的机器数， 因而 x+y 和 ux+uy 具有完全
一样的位序列。
（14） x*~y+ux*uy==-x
永真。 -y=~y+1， 即~y=-y-1。 而 ux*uy=x*y， 因此， 等式左边为 x*(-y-1)+x*y=-x。

35. 变量 dx、 dy 和 dz 的声明和初始化如下：
```
double dx = (double) x;
double dy = (double) y;
double dz = (double) z;
```
若 float 和 double 分别采用 IEEE 754 单精度和双精度浮点数格式， sizeof(int)=4，则对于任意 int 型变
量 x、 y 和 z， 判断以下关系表达式是否永真。 若永真则给出证明； 若不永真则给出结果为假时 x 和 y
的取值。
（1） dx*dx >= 0
（2） (double)(float) x == dx
（3） dx+dy == (double) (x+y)

（4） (dx+dy)+dz == dx+(dy+dz)

（5） dx*dy*dz == dz*dy*dx

（6） dx/dx == dy/dy

参考答案：

(1)dx*dx>=0 永真。double 型数据用 IEEE 754 标准表示，尾数用原码小数表示，符号和数值部分分开运算。不管结果是否溢出，都不会影响乘积的符号。

(2)（double）（float）x==dx 非永真。当 int 型变量 x 的有效数比 float 型可表示的最大有效位数 24 更多时，x 强制转换为 float 型数据时有效位数丢失，而将 x 转换为都 double 型数据时没有有效位丢失。也即等式左边可能是近似值，而右边是精确值。

(3)dx+dy==（double）（x+y）非永真。因为 x+y 可能会溢出，而 dx+dy 不会溢出。

(4)（dx+dy）+dz==dx+（dy+dz）永真。因为 dx、dy、和 dz 是由 32 位 int 型数据转换得到的，而 double 类型可以精确表示 int 类型数据，并且对阶时尾数移位位数不会超过 52 位，因此位数不会舍入，因而不会发生大数吃小数的情况。

(5)dx*dy*dz==dz*dy*dx 非永真。相称的结果可能产生舍入。

(6)dx/dx==dy/dy 非永真。dx 和 dy 中只要有一个为 0、另一个不为 0 就不相等。

36. 在 IEEE 754 浮点数运算中， 当结果的尾数出现什么形式时需要进行左规， 什么形式时需要进行右规？

如何进行左规， 如何进行右规？

参考答案：

（1） 对于结果为±1x. xx…x 的情况， 需要进行右规。 右规时， 尾数右移一位， 阶码加 1。 右规操作可

以表示为： M b M b ×2

-1 ， E b E b +1。 右规时注意以下两点：

① 尾数右移时， 最高位“1” 被移到小数点前一位作为隐藏位， 最后一位移出时， 要考虑舍入。

② 阶码加 1 时， 直接在末位加 1。

（2） 对于结果为±0.00…01x…x 的情况， 需要进行左规。 左规时， 数值位逐次左移， 阶码逐次减 1，

直到将第一位“1”移到小数点左边。 假定 k 为结果中“±” 和最左边第一个 1 之间连续 0 的个数，

则左规操作可以表示为： M b M b ×2 k ， E b E b -k。 左规时注意以下两点：

① 尾数左移时数值部分最左边 k 个 0 被移出， 因此， 相对来说， 小数点右移了 k 位。 因为进行

尾数相加时， 默认小数点位置在第一个数值位（即： 隐藏位） 之后， 所以小数点右移 k 位后

被移到了第一位 1 后面， 这个 1 就是隐藏位。

② 执行 E b E b -k 时， 每次都在末位减 1， 一共减 k 次。

37. 在 IEEE 754 浮点数中，如何判断浮点运算的结果是否溢出？

参考答案：浮点运算结果是否溢出，并不以尾数溢出来判断，而主要看阶码是否溢出。尾数溢出时，可通过右规操作进行纠正。因为在进行规格化、尾数舍入和浮点数的乘/除运算过程中，都需要对阶码进行加、减运算，所以在这些操作过程中，可能会发生阶码上溢或阶码下溢。阶码上溢时，说明结果的数值太大，无法表示，是真正的溢出；阶码下溢时，说明结果数值太小，可以把结果近似为 0.

39. 采用 IEEE 754 单精度浮点数格式计算下列表达式的值。

（1） 0.75+(- 65.25)

（2） 0.75-(- 65.25)

参考答案：

（1） x = 0.75 = 0.110...0B = (1.10...0) 2 × 2 -1 ，

y = - 65.25 = - 1000001.01000...0B = (-1.00000101...0) 2 × 2 6

用 IEEE 754 标准单精度格式表示为：

[x] 浮

= 0 01111110 10...0 ： 阶码 E x = 01111110， 尾数 M x = 0(1). 1...0

[y] 浮

= 1 10000101 000001010...0 ： 阶码 E y = 10000101， 尾数 M y =1

(1).000001010...0

尾数 M x 和 M y 用原码表示， 其中小数点前写了 2 位， 第一位是符号位， 第二位是隐藏位

“1”， 加了括号。

然后， 详细描述计算机中进行浮点数加减运算的过程： （假定保留 2 位附加位： 保护位和舍入位）

（1） 0.75+ (- 65.25)

① 对阶： [Δ E] 补 =E x + [-E y] 补 (mod 2 n) = 0111 1110 + 0111 1011 = 1111 1001， 即Δ E = -7。

根据对阶规则可知需对 x 进行对阶， 结果为： E x = E y = 10000101， M x = 00.000000110...000

M x 右移 7 位， 符号不变， 数值高位补 0， 隐藏位右移到小数点后面， 最后移出的 2 位保留

② 尾数相加： M b = M x + M y = 0 0. 000000110...000+ 11. 000001010 ...000 （注意小数点在隐藏位后）

根据原码加/减法运算规则， 得： 00.000000110...000+ 11.000001010...000 = 11.000000100…000

上式尾数中最左边第一位是符号位， 其余都是数值部分， 尾数后面两位是附加位（加粗）。

③ 规格化： 尾数数值部分最高位为 1， 因此不需要进行规格化。

④ 舍入： 把结果的尾数 M b 中最后两位附加位舍入掉， 从本例来看， 不管采用什么舍入法， 结果

都一样， 都是把最后两个 0 去掉， 得： M b = 11.000000100…0

⑤ 溢出判断： 在上述阶码计算和调整过程中， 没有发生“阶码上溢” 和“阶码下溢”的问题。 因

此， 阶码 E b = 10000101。

最后结果为 E b = 10000101， M b = 1(1).00000010…0， 即： - 64.5。

（2） 0.75-(- 65.25)

① 对阶： [Δ E] 补

= E x + [-E y] 补 (mod 2 n) = 0111 1110 + 0111 1011 = 1111 1001， Δ E = -7。

根据对阶规则可知需要对 x 进行对阶， 结果为： E x = E y = 10000110， M x = 00.000000110...000

M x 右移一位， 符号不变， 数值高位补 0， 隐藏位右移到小数点后面， 最后移出的位保留

② 尾数相加： M b = M x - M y = 00. 000000110...000 - 11. 000001010...000（注意小数点在隐藏位后）

根据原码加/减法运算规则， 得： 00.000000110...000 - 11.000001010...000=01.00001000…000

上式尾数中最左边第一位是符号位， 其余都是数值部分， 尾数后面两位是附加位（加粗）。

③ 规格化： 尾数数值部分最高位为 1， 不需要进行规格化。

④ 舍入： 把结果的尾数 M b 中最后两位附加位舍入掉， 从本例来看， 不管采用什么舍入法， 结果

都一样， 都是把最后两个 0 去掉， 得： M b = 01.00001000…0

⑤ 溢出判断： 在上述阶码计算和调整过程中， 没有发生“阶码上溢” 和“阶码下溢”的问题。 因

此， 阶码 E b = 10000101。

最后结果为 E b = 10000101， M b = 0(1).00001000…0， 即： +66。

40.以下是函数 fpower2 的 C 语言源程序，它用于计算 2x 的浮点数表示，其中调用了函数 uf，u2f 用于将一个无符号整数表示的 0/1 序列作为 float 类型返回。请填写 fpower2 函数中的空包部分，以使其能正确计算结果。

参考答案：

-149

0

0

-126

0

0x400000>>(-x-127)

128

x+127

0

255

0

41.以下是一组关于浮点数按位级进行运算的编程题目，其中用到一个数据类型 float_bits，••••••

参考答案：

（1）

```
float_bits float_abs(float_bits)
{
    unsigned sign=f>>31;
    unsigned exp=f>>23&0xFF;
    unsigned frac=f&0x7FFFFF;
    if((exp==0xFF)&&(frac!=0)||(sign==0))
        return f;
    else
```

```
        return f&0x7FFFFFFF;
}
(2)
float_bits float_neg(float_bits f)
{
    unsigned exp=f>>23&0xFF;
    unsigned frac=f&0x7FFFFF;
    if ((exp==0xFF)&&(frac!=0))
        return f;
    else
        return f^0x80000000;
}
(3)
float_bits float_half(float_bits f)
{
    unsigned sign=f>>31;
    unsigned exp=f>>32&0xFF;
    unsigned frac=f&0x7FFFFF;
    if ((exp==0xFF)&&(frac!=0))
        return f;
    else if((exp==0)&&(frac!=0))
        return sign<<31|frac>>1;
    else
    {
        exp=exp+0xFF;
        if(exp!=0)
            return sign<<31|exp<<23|frac;
        else
            return sign<<31|(frac|0x800000)>>1;
    }
}
(4)
float_bits float_twice(float_bits f)
{
    unsigned sign=f>>31;
    unsigned exp=f>>23&0xFF;
    unsigned frac=f&0x7FFFFF;
    if((exp==0)||(exp==0xFF)&&(frac)!=0)
    {
        if(frac&0x400000)
            return sign<<31|1<<23|(frac&0x3FFFFF)<<1;
        else
            return sign<<31|frac<<1;
    }
```

```
    else
    {
        exp=exp+0x01;
        if(exp!=0xFF)
            return sign<<31|exp<<23|frac;
        else
            return sign<<31|exp<<23;
    }
}
(5)
float_bits loat_i2f(int t)
{
    unsigned pre_count=30;
    unsigned pos_count=31;
    unsigned sign=(unsigned)i>>31;
    unsigned neg_i;
    if(i==0)
        return i;
    if(sign==0)
    {
        while(i>>pre_count==0)
            pre_count--;
        return sign<<31|(127+pre_count)<<23|(unsigned)(i<<(32-pre_count))>>23;
    }
    else
    {
        while(i<<pos_count==0)
            pos_count--;
        neg_i=(~(i>>(32-pos_count))<<(32-pre_count))|(1<<(31-pos_count));
        while(neg_i>>pre_count==0)
            pre_count--;
        return sign<<31|(127+pre_count)<<23|neg_i<<(32-pre_count)>>23;
    }
}
(6)
int float_f2i(float_bits f)
{
    unsigned sign=f>>31;
    unsigned exp=f>>23&0xFF;
    unsigned frac=f&0x7FFFFF;
    unsigned exp_value=exp-127;
    unsigned neg_i;
    unsigned pos_count=31;
    if((exp==0Xff)||(exp_value>30))
```

```
        return 0x80000000;
    else if((exp==0)||(exp_value<0))
        return 0;
    else if(sign==0)
        return (1<<30|frac<<7)>>(30-exp_value);
    else
    {
        neg_i=(1<<30|frac<<7)>>(30-exp_value);
    while(neg_i<<pos_count==0)
        pos_count--;
    return (~(neg_i>>(32-pos_count))<<(32-pos_count))|(1<<(31-pos_count));
    }
}
```

第三章

3. 对于以下 AT&T 格式汇编指令，根据操作数的长度确定对应指令助记符中的长度后缀，并说明每个操作数的寻址方式。

（1）mov 8(%ebp, %ebx, 4), %ax
（2）mov %al, 12(%ebp)
（3）add (, %ebx,4), %ebx
（4）or (%ebx), %dh
（5）push $0xF8
（6）mov $0xFFF0, %eax
（7）test %cx, %cx
（8）lea 8(%ebx, %esi), %eax

参考答案：

（1）后缀：w，源：基址+比例变址+偏移，目：寄存器
（2）后缀：b，源：寄存器，目：基址+偏移
（3）后缀：l，源：比例变址，目：寄存器
（4）后缀：b，源：基址，目：寄存器
（5）后缀：l，源：立即数，目：栈
（6）后缀：l，源：立即数，目：寄存器
（7）后缀：w，源：寄存器，目：寄存器
（8）后缀：l，源：基址+变址+偏移，目：寄存器

4.

4. 使用汇编器处理以下各行AT&T格式代码时都会产生错误，请说明每一行存在什么错误。

（1）movl 0xFF, (%eax) （2）movb %ax, 12(%ebp)
（3）addl %ecx, $0xF0 （4）orw $0xFFFF0, (%ebx)
（5）addb $0xF8, (%dl) （6）movl %bx, %eax
（7）andl %esi, %esx （8）movw 8(%ebp, , 4), %ax

答：
（1）书写错误。因为源操作数是立即数 0xFF，所以需要在前面加上‘$’
（2）由于源操作数（%ax）是 16 位，而长度后缀是字节‘b’，所以不一致，应改为‘movw’
（3）目的操作数不能是立即数寻址
（4）操作数位数超过 16 位，而长度后缀为 16 位的‘w’，应改为‘orl’
（5）不能用 8 位寄存器作为目的操作数地址所在寄存器
（6）源操作数寄存器与目的操作数寄存器长度不一致
（7）不存在 ESX 寄存器

5. 假设变量 *x* 和 *ptr* 的类型声明如下：

```
src_type  x;
dst_type  *ptr;
```

这里，src_type 和 dst_type 是用 typedef 声明的数据类型。有以下一个 C 语言赋值语句：

```
*ptr=(dst_type) x;
```

若 *x* 存储在寄存器 EAX 或 AX 或 AL 中，*ptr* 存储在寄存器 EDX 中，则对于表 3.12 中给出的 src_type 和 dst_type 的类型组合，写出实现上述赋值语句的机器级代码。要求用 AT&T 格式汇编指令表示机器级代码。

表 3.12 题 5 用表

src_type	dst_type	机器级表示
char	int	
int	char	
int	unsigned	
short	int	
unsigned char	unsigned	
char	unsigned	
int	int	

表 3.12 题 5 用表

src_type	dst_type	机器级表示
char	int	movsbl %al, (%edx)
int	char	movb %al, (%edx)
int	unsigned	movl %eax, (%edx)
short	int	movswl %ax, (%edx)
unsigned char	unsigned	movzbl %al, (%edx)
char	unsigned	movsbl %al, (%edx)
int	int	movl %eax, (%edx)

6. 假设某个 C 语言函数 func 的原型声明如下：

```
void func(int *xptr, int *yptr, int *zptr);
```

函数 func 的过程体对应的机器级代码用 AT&T 汇编形式表示如下：

```
1   movl    8(%ebp), %eax
2   movl    12(%ebp), %ebx
3   movl    16(%ebp), %ecx
4   movl    (%ebx), %edx
5   movl    (%ecx), %esi
6   movl    (%eax), %edi
7   movl    %edi, (%ebx)
8   movl    %edx, (%ecx)
9   movl    %esi, (%eax)
```

请回答下列问题或完成下列任务。

（1）在过程体开始时三个入口参数对应实参所存放的存储单元地址是什么？（提示：当前栈帧底部由帧指针寄存器 EBP 指示）

（2）根据上述机器级代码写出函数 func 的 C 语言代码。

6. （1）xptr、yptr 和 zptr 对应实参所存放的存储单元地址分别为：R[ebp]+8、R[ebp]+12、R[ebp]+16。

（2）函数 func 的 C 语言代码如下：

```
void func(int *xptr, int *yptr, int *zptr)
{
    int tempx=*xptr;
    int tempy=*yptr;
    int tempz=*zptr;

    *yptr=tempx;
    *zptr = tempy;
    *xptr = tempz;
}
```

7. 假设变量 x 和 y 分别存放在寄存器 EAX 和 ECX 中，请给出以下每条指令执行后寄存器 EDX 中的结果。

```
（1）leal   (%eax), %edx
（2）leal   4(%eax, %ecx), %edx
（3）leal   (%eax, %ecx, 8), %edx
（4）leal   0xC(%ecx, %eax, 2), %edx
（5）leal   ( , %eax, 4), %edx
（6）leal   (%eax, %ecx), %edx
```

答：

```
（1）R[edx]=x              //把 eax 的值放到 edx 中去
（2）R[edx]=x+y+4          //把 eax+ecx+4 的值放到 edx 中去
（3）R[edx]=x+8*y          //把 eax+8*ecx 的值放到 edx 中去
（4）R[edx]=y+2*x+12       //把 ecx+2*eax+12 的值放到 edx 中去
（5）R[edx]=4*x            //把 4*eax 的值放到 edx 中去
（6）R[edx]=x+y            //把 eax+ecx 的值放到 edx 中去
```

8. 假设以下地址以及寄存器中存放的机器数如表 3.13 所示。

表 3.13 题 8 用表

地址	机器数	寄存器	机器数
0x8049300	0xfffffff0	EAX	0x08049300
0x8049400	0x80000008	EBX	0x00000100
0x8049384	0x807fff00	ECX	0x00000010
0x8049380	0x905f12a8	EDX	0x00000080

分别说明执行以下指令后，哪些地址或寄存器中的内容会发生改变？改变后的内容是什么？条件标志 OF、SF、ZF 和 CF 会发生什么改变？

```
（1）addl   (%eax), %edx
（2）subl   (%eax, %ebx), %ecx
（3）orw    4(%eax, %ecx, 8), %bx
（4）testb  $0x80, %dl
（5）imull  $32, (%eax, %edx)
（6）decw   %cx
```

8. （1）指令功能为：R[edx]←R[edx]+M[R[eax]]=0x00000080+M[0x8049300]，寄存器 EDX 中内容改变。改变后的内容为以下运算的结果：00000080H+FFFFFF0H

```
  0000 0000 0000 0000 0000 0000 1000 0000
+ 1111 1111 1111 1111 1111 1111 1111 0000
-------------------------------------------
1 0000 0000 0000 0000 0000 0000 0111 0000
```

因此，EDX 中的内容改变为 0x00000070。根据表 3.5 可知，加法指令会影响 OF、SF、ZF 和 CF 标志。OF=0，ZF=0，SF=0，CF=1。

（2）指令功能为：R[ecx]←R[ecx]−M[R[eax]+R[ebx]]=0x00000010+M[0x8049400]，寄存器 ECX 中内容改变。改变后的内容为以下运算的结果：00000010H−80000008H

```
  0000 0000 0000 0000 0000 0000 0001 0000
+ 0111 1111 1111 1111 1111 1111 1111 1000
-------------------------------------------
0 1000 0000 0000 0000 0000 0000 0000 1000
```

因此，ECX 中的内容改为 0x80000008。根据表 3.5 可知，减法指令会影响 OF、SF、ZF 和 CF 标志。OF=1，ZF=0，SF=1，CF=1⊕0=1。

（3）指令功能为：R[bx]←R[bx] or M[R[eax]+R[ecx]*8+4]，寄存器 BX 中内容改变。改变后的内容为以下运算的结果：0x0100 or M[0x8049384]=0100H or FF00H

```
  0000 0001 0000 0000
or 1111 1111 0000 0000
-----------------------
  1111 1111 0000 0000
```

因此，BX 中的内容改为 0xFF00。由 3.3.3 节可知，OR 指令执行后 OF=CF=0；因为结果不为 0，故 ZF=0；因为最高位为 1，故 SF=1。

（4）test 指令不改变任何通用寄存器，但根据以下“与”操作改变标志：R[dl] and 0x80

```
  1000 0000
and 1000 0000
-------------
  1000 0000
```

（5）指令功能为：M[R[eax]+R[edx]]←M[R[eax]+R[edx]]*32，即存储单元 0x8049380 中的内容改变为以下运算的结果：M[0x8049380]*32=0x908f12a8*32，也即只要将 0x908f12a8 左移 5 位即可得到结果。

1001 0000 1000 1111 0001 0010 1010 1000<<5

=0001 0001 1110 0010 0101 0101 0000 0000

因此，指令执行后，单元 0x8049380 中的内容改变为 0x11e25500。显然，这个结果是溢出的。但是，根据表 3.5 可知，乘法指令不影响标志位，也即并不会使 OF=1。

（6）指令功能为：R[cx]←R[cx]−1，即 CX 寄存器的内容减一。

```
  0000 0000 0001 0000
+ 1111 1111 1111 1111
---------------------
1 0000 0000 0000 1111
```

因此，指令执行后 CX 中的内容从 0x0010 变为 0x000F。由表 3.5 可知，DEC 指令会影响 OF、ZF、SF，根据上述运算结果，得到 OF=0，ZF=0，SF=0。

10. 假设函数 product 的 C 语言代码如下，其中 num_type 是用 typedef 声明的数据类型。

```
1  void product(num_type *d, unsigned x, num_type y ) {
2      *d = x*y;
3  }
```

函数 product 的过程体对应的主要汇编代码如下：

```
3  imull   %eax, %ecx
4  mull    16(%ebp)
5  leal    (%ecx, %edx), %edx
6  movl    8(%ebp), %ecx
7  movl    %eax, (%ecx)
8  movl    %edx, 4(%ecx)
```

请给出上述每条汇编指令的注释，并说明 num_type 是什么类型。

10. 从汇编代码的第 2 行和第 4 行看，y 应该是占 8 个字节，R[ebp]+20 开始的 4 个字节为高 32 位字节，记为 y_h；R[ebp]+16 开始的 4 个字节为低 32 位字节，记为 y_l。根据第 4 行为无符号数乘法指令，得知 y 的数据类型 num_type 为 unsigned long long。

```
movl    12(%ebp), %eax    //R[eax]←M[R[ebp]+12]，将 x 送 EAX
movl    20(%ebp), %ecx    //R[ecx]←M[R[ebp]+20]，将 yh 送 ECX
imull   %eax, %ecx        //R[ecx]←R[ecx]*R[eax]，将 yl*x 的低 32 位送 ECX
mull    16(%ebp)          //R[edx][R[eax]←M[R[ebp]+16]*R[eax]，将 yl*x 送 EDX-EAX
leal    (%ecx, %edx), %edx
        // R[edx]←R[ecx]+R[edx]，将 yl*x 的高 32 位与 yh*x 的低 32 位相加后送 EDX
movl    8(%ebp), %ecx     //R[ecx]←M[R[ebp]+8]，将 d 送 ECX
movl    %eax, (%ecx)      //M[R[ecx]]←R[eax]，将 x*y 低 32 位送 d 指向的低 32 位
movl    %edx, 4(%ecx)     //M[R[ecx]+4]←R[edx]，将 x*y 高 32 位送 d 指向的高 32 位
```

11. 已知 IA-32 是小端方式处理器，根据给出的 IA-32 机器代码的反汇编结果（部分信息用 x 表示）回答问题。

（1）已知 je 指令的操作码为 01110100，je 指令的转移目标地址是什么？call 指令中的转移目标地址 0x80483b1 是如何反汇编出来的？

```
804838c:    74 08                   je      xxxxxxx
804838e:    e8 1e 00 00 00          call    80483b1<test>
```

（2）已知 jb 指令的操作码为 01110010，jb 指令的转移目标地址是什么？movl 指令中的目的地址如何反汇编出来的？

```
8048390:    72 f6                   jb      xxxxxxx
8048392:    c6 05 00 a8 04 08 01    movl    $0x1, 0x804a800
8048399:    00 00 00
```

（3）已知 jle 指令的操作码为 01111110，mov 指令的地址是什么？

```
xxxxxxx:    7e 16                   jle     80492e0
xxxxxxx:    89 d0                   mov     %edx, %eax
```

（4）已知 jmp 指令的转移目标地址采用相对寻址方式，jmp 指令操作码为 11101001，其转移目标地址是什么？

```
8048296:    e9 00 ff ff ff          jmp     xxxxxxx
804829b:    29 c2                   sub     %eax, %edx
```

11. 根据第 3.3.4 节得知，条件转移指令都采用相对转移方式在段内直接转移，即条件转移指令的转移目标地址为：（PC）+偏移量。

（1）因为 je 指令的操作码为 01110100，所以机器代码 7408H 中的 08H 是偏移量，故转移目标地址为：0x804838c+2+0x8=0x8048396。

call 指令中的转移目标地址 0x80483b1=0x804838e+5+0x1e，由此，可以看出，call 指令机器代码中后面的 4 个字节是偏移量，因 IA-32 采用小端方式，故偏移量为 0000001EH。call 指令机器代码共占 5 个字节，因此，下条指令的地址为当前指令地址 0x804838e 加 5。

（2）jb 指令中 F6H 是偏移量，故其转移目标地址为：0x8048390+2+0xf6=0x8048488。

movl 指令的机器代码有 10 个字节，前两个字节是操作码等，后面 8 个字节为两个立即数，因为是小端方式，所以，第一个立即数为 0804A800H，即汇编指令中的目的地址 0x804a800，最后 4 个字节为立即数 00000001H，即汇编指令中的常数 0x1。

（3）jle 指令中的 7EH 为操作码，16H 为偏移量，其汇编形式中的 0x80492e0 是转移目的地址，因此，假定后面的 mov 指令的地址为 x，则 x 满足以下公式：0x80492e0=x+0x16，故 x=0x80492e0−0x16=0x80492ca。

（4）jmp 指令中的 E9H 为操作码，后面 4 个字节为偏移量，因为是小端方式，故偏移量为 FFFFFF00H，即 −100H=−256，后面的 sub 指令的地址为 0x804829b，故 jmp 指令的转移目标地址为 0x804829b+0xffffff00=0x804829b−0x100=0x804819b。

14. 已知函数 do_loop 的 C 语言代码如下：

```
1   short do_loop(short x, short y, short k) {
2       do {
3           x*=(y%k) ;
4           k--;
5       } while ((k>0) && (y>k));
6       return x;
7   }
```

函数 do_loop 的过程体对应的汇编代码如下：

```
1    movw   8(%ebp), %bx
2    movw   12(%ebp), %si
3    movw   16(%ebp), %cx
4  .L1:
5    movw    %si, %dx
6    movw    %dx, %ax
7    sarw    $15, %dx
8    idiv    %cx
9    imulw   %dx, %bx
10   decw    %cx
11   testw   %cx, %cx
12   jle     .L2
13   cmpw    %cx, %si
14   jg      .L1
15 .L2:
16   movswl  %bx, %eax
```

请回答下列问题或完成下列任务：

（1）给每条汇编指令添加注释，并说明每条指令执行后，目的寄存器中存放的是什么信息？

（2）上述函数过程体中用到了哪些被调用者保存寄存器和哪些调用者保存寄存器？在该函数过程体前面的准备阶段哪些寄存器必须保存到栈中？

（3）为什么第 7 行中的 DX 寄存器需要算术右移 15 位？

14. （1）每个入口参数都要按 4 字节边界对齐，因此，参数 x、y 和 k 入栈时都占 4 个字节。

```
1    movw   8(%ebp), %bx      //R[bx]←M[R[ebp]+8]，将 x 送 BX
2    movw   12(%ebp), %si     //R[si]←M[R[ebp]+12]，将 y 送 SI
3    movw   16(%ebp), %cx     //R[cx]←M[R[ebp]+16]，将 k 送 CX
4  .L1:
5    movw   %si, %dx          //R[dx]←R[si]，将 y 送 DX
6    movw   %dx, %ax          //R[ax]←R[dx]，将 y 送 AX
7    sarw   $15, %dx          //R[dx]←R[dx]>>15，将 y 的符号扩展 16 位送 DX
8    idiv   %cx               //R[dx]←R[dx-ax]÷R[cx]的余数，将 y%k 送 DX
                              //R[ax]←R[dx-ax]÷R[cx]的商，将 y/k 送 AX
9    imulw  %dx, %bx          //R[bx]←R[bx]*R[dx]，将 x*(y%k) 送 BX
10   decw   %cx               //R[cx]←R[cx]-1，将 k-1 送 CX
11   testw  %cx, %cx          //R[cx] and R[cx]，得 OF=CF=0，负数则 SF=1,零则 ZF=1
12   jle    .L2               //若 k 小于等于 0，则转.L2
13   cmpw   %cx, %si          //R[si] - R[cx]，将 y 与 k 相减得到各标志
14   jg     .L1               //若 y 大于 k，则转.L1
15 .L2:
16   movswl  %bx, %eax        // R[eax]←R[bx]，将 x*(y%k) 送 AX
```

（2）被调用者保存寄存器有 BX、SI，调用者保存寄存器有 AX、CX 和 DX。

在该函数过程体前面的准备阶段，被调用者保存的寄存器 EBX 和 ESI 必须保存到栈中。

（3）因为执行第 8 行除法指令前必须先将被除数扩展为 32 位，而这里是带符号数除法，因此，采用算术右移以扩展 16 位符号，放在高 16 位的 DX 中，低 16 位在 AX 中。

17. 已知函数 test 的入口参数有 a、b、c 和 p，C 语言过程体代码如下：

```
*p = a;
return b*c;
```

函数 test 过程体对应的汇编代码如下：

```
1   movl    20(%ebp), %edx
2   movsbw  8(%ebp), %ax
3   movw    %ax, (%edx)
4   movzwl  12(%ebp), %eax
5   movzwl  16(%ebp), %ecx
6   mull    %ecx, %eax
```

写出函数 test 的原型，给出返回参数的类型以及入口参数 a、b、c 和 p 的类型和顺序。

17. 根据第 2、3 行指令可知，参数 a 是 char 型，参数 p 是指向 short 型变量的指针；根据第 4、5 行指令可知，参数 b 和 c 都是 unsigned short 型，根据第 6 行指令可知，test 的返回参数类型为 unsigned int。因此，test 的原型为：unsigned int test(char a, unsigned short b, unsigned short c, short *p);

19. 已知递归函数 refunc 的 C 语言代码框架如下：

```
1   int refunc(unsigned x) {
2       if ( _____ )
3           return _____ ;
4       unsigned nx =_____ ;
5       int rv = refunc(nx) ;
6       return _____ ;
7   }
```

上述递归函数过程体对应的汇编代码如下：

```
1   movl    8(%ebp), %ebx
```

```
2    movl    S0, %eax
3    testl   %ebx, %ebx
4    je      .L2
5    movl    %ebx, %eax
6    shrl    S1, %eax
7    movl    %eax, (%esp)
8    call    refunc
9    movl    %ebx, %edx
10   andl    S1, %edx
11   leal    (%edx, %eax), %eax
12 .L2:
     ……
     ret
```

根据对应的汇编代码填写 C 代码中缺失部分，并说明函数的功能。

19. 第 1 行汇编指令说明参数 x 存放在 EBX 中，根据第 4 行判断 x=0 则转.L2，否则继续执行第 5~10 行指令。根据第 5、6、7 行指令可知，入栈参数 nx 的计算公式为 x>>1；根据第 9、10、11 行指令可知，返回值为(x&1)+rv。由此推断出 C 缺失部分如下：

```
1   int refunc(unsigned x) {
2       if ( ___x==0___ )
3           return ____0____ ;
4       unsigned nx = ___x>>1___ ;
5       int rv = refunc(nx) ;
6       return ___(x & 0x1) + rv___ ;
7   }
```

该函数的功能为计算 x 的各个数位中 1 的个数。

20. 填写表 3.14，说明每个数组的元素大小、整个数组的大小以及第 i 个元素的地址。

表 3.14 题 20 用表

数组	元素大小（B）	数组大小（B）	起始地址	元素 i 的地址
char A[10]			&A[0]	
int B[100]			&B[0]	
short *C[5]			&C[0]	
short **D[6]			&D[0]	
long double E[10]			&E[0]	
long double *F[10]			&F[0]	

21. 假设 short 型数组 S 的首地址 A_S 和数组下标（索引）变量 i（int 型）分别存放在寄存器 EDX 和 ECX 中，下列给出的表达式的结果存放在 EAX 或 AX 中，仿照例子填写表 3.15，说明表达式的类型、值和相应的汇编代码。

表 3.15 题 21 用表

表达式	类型	值	汇编代码
S			
S+i			
S[i]	short	M[A_S+2*i]	movw (%edx, %ecx, 2), %ax
&S[10]			
&S[i+2]	short *	A_S+2*i+4	leal 4(%edx, %ecx, 2), %eax
&S[i]-S			
S[4*i+4]			
*(S+i-2)			

21.

表达式	类型	值	汇编代码
S	short *	A_S	leal (%edx), %eax
S+i	short *	A_S+2*i	leal (%edx, %ecx, 2), %eax
S[i]	short	M[A_S+2*i]	movw (%edx, %ecx, 2), %ax
&S[10]	short *	A_S+20	leal 20(%edx), %eax
&S[i+2]	short *	A_S+2*i+4	leal 4(%edx, %ecx, 2), %eax
&S[i]-S	int	(A_S+2*i-A_S)/2=i	movl %ecx, %eax
S[4*i+4]	short	M[A_S+2*(4*i+4)]	movw 8(%edx, %ecx, 8), %ax
*(S+i-2)	short	M[A_S+2*(i-2)]	movw -4(%edx, %ecx, 2), %ax

22. 假设函数 sumij 的 C 代码如下，其中，*M* 和 *N* 是用#define 声明的常数。

```
1   int a[M][N], b[N][M];
2
3   int sumij(int i, int j) {
4       return a[i][j] + b[j][i];
5   }
```

已知函数 sumij 的过程体对应的汇编代码如下：

```
1   movl    8(%ebp), %ecx
2   movl    12(%ebp), %edx
3   leal    (,%ecx, 8), %eax
4   subl    %ecx, %eax
5   addl    %edx, %eax
6   leal    (%edx, %edx, 4), %edx
7   addl    %ecx, %edx
8   movl    a(, %eax, 4), %eax
9   addl    b(,%edx, 4), %eax
10  movl    20(%ebp), %edx
11  movl    %eax, (%edx)
12  movl    $4536, %eax
```

根据上述汇编代码，确定 *L*、*M* 和 *N* 的值。

22. 根据汇编指令功能可以推断最终在 EAX 中返回的值为：

M[a+28*i+4*j]+M[b+20*j+4*i]，因为数组 a 和 b 都是 int 型，每个数组元素占 4B，因此，M=5, N=7。

23. 执行第 11 行指令后，a[i][j][k]的地址为 a+4*(63*i+9*j+k)，所以，可以推断出 M=9，N=63/9=7。根据第 12 行指令，可知数组 a 的大小为 4536 字节，故 L=4536/(4*L*M)=18。

28. 以下是结构 test 的声明：

```
struct {
    char        c;
    double      d;
    int         i;
    short       s;
    char        *p;
    long        l;
    long long   g;
    void        *v;
} test;
```

假设在 Windows 平台上编译，则这个结构中每个成员的偏移量是多少？结构总大小为多少字节？如何调整成员的先后顺序使得结构所占空间最小？

28. Windows 平台要求不同的基本类型按照其数据长度进行对齐，每个成员的偏移量如下：

c	d	i	s	p	l	g	v
0	8	16	20	24	28	32	40

结构总大小为 48 字节，因为其中的 d 和 g 必须是按 8 字节边界对齐，所以，必须在末尾再加上 4 个字节，即 44+4=48 字节。变量长度按照从大到小顺序排列，可以使得结构所占空间最小，因此调整顺序后的结构定义如下：

```
struct {
        double      d;
        long long   g;
        int         i;
        char        *p;
        long        l;
```

简答题

1 冯.诺依曼计算机由哪几部分组成？各部分的功能是什么？
（5.0分）

正确答案：
计算机应由运算器、控制器、存储器、输入设备和输出设备五个基本部件组成。
各基本部件的功能是：
存储器不仅能存放数据，而且也能存放指令，形式上两者没有区别，但计算机应能区分数据还是指令；
控制器应能自动取出指令来执行；
运算器应能进行加/减/乘/除四种基本算术运算，并且也能进行一些逻辑运算和附加运算；
操作人员可以通过输入设备、输出设备和主机进行通信。
答案解析：

2 一条指令的执行过程包含哪几个阶段？
（5.0分）

正确答案：
第一步：根据PC取指令
第二步：指令译码
第三步：PC增量
第四步：取操作数，指令执行
第五步：送结果
答案解析：

3 指令和数据在形式上没有差别，且都存于存储器中，计算机如何区分它们？
（5.0分）

正确答案：
指令和数据在计算机内部都是用二进制表示的，都是0、1序列，在形式上没有差别。
在指令和数据取到CPU之前，它们都存在存储器中，CPU并不是把信息从主存读出后靠某种判断方法来识别读出的信息是数据还是指令的，而是在读出之前就知道要读的信息是数据还是指令了。
因为执行指令的过程分为“取指令、指令译码、取操作数、运算、送结果”等，所以在取指令阶段，根据PC的值从主存取来的一定是当做指令，而在取操作数阶段，取来的一定是当做数据。
答案解析：

4 计算机系统的层次结构如何划分？用户可以分哪几类？每类用户工作在哪个层次？
（5.0分）

正确答案：
计算机系统为软件和硬件两部分，软件和硬件的界面是指令集体系结构（ISA）。
计算机系统从高到低相分为应用软件、系统软件和硬件三个层次；
不同计算机用户工作在不同的层次，从高到低相分为应用程序级（最终用户）、高级语言虚拟机级（高级语言程序员或应用程序员）、汇编语言虚拟机级（汇编语言程序员）、操作系统虚拟机级（系统管理员）、机器语言机器级（机器语言程序员）。
答案解析：

5 微操作
（5.0分）

正确答案：
控制部件通过控制线向执行部件发出各种控制命令，通常这种控制命令叫做微命令。而执行部件接受微命令后所执行的操作就叫做微操作。
答案解析：

6 语言处理系统
（5.0分）

正确答案:
在进行高级语言程序设计时，需要有相应的应用程序开发支撑环境。提供程序编辑器和各类翻译转换软件的工具包统称为语言处理系统。
答案解析:

7 指令集体系结构（ISA）
(5.0分)

正确答案:
ISA是计算机硬件与系统软件之间的接口。指机器语言程序员或操作系统、编译器、解释器设计人员所看到的计算机功能特性和概念性结构。
其核心部分是指令系统，同时还包含数据类型和数据格式定义、寄存器组织、I/O空间的编址和数据传输方式、中断结构、计算机状态的定义和切换、存储保护等。
ISA设计得好坏直接决定了计算机的性能和成本。

答案解析:

8 透明
(5.0分)

正确答案:
由于计算机系统采用了层次化结构进行设计和组织，因此面向不同的硬件或软件层面进行工作的人员或用户所“看到”的计算机是不一样的。也就是说，计算机组织方式或系统结构中的一部分对某些用户而言是“看不到”的或称为“透明”的。
答案解析:

9 响应时间
(5.0分)

正确答案:
也称为执行时间或等待时间，是指从作业提交开始到作业完成的时间。
一般一个程序的响应时间除了CPU执行程序包含的指令执行时间外，还包括I/O的时间、系统运行其他用户程序所用的时间以及操作系统运行的时间等。
答案解析:

10 吞吐率
(5.0分)

正确答案:
也称为带宽，是指在一定的时间内所完成的工作量
答案解析:

11 CPI
(5.0分)

正确答案:
衡量CPU性能的一种基本参数，它表示执行一条指令所需的平均时钟周期个数
答案解析:

12 MIPS
(5.0分)

正确答案:
用来衡量单位时间内执行指令的条数，具体是指每秒执行多少百万条指令
答案解析:

13 图1.1所示模型机（采用图1.2所示指令格式）的指令系统中，除了有mov（op=0000）、add（op=0001）、load（op=1110）和store（op=1111）指令外，R型指令还有减（sub，op=0010）和乘（mul，op=0011）等指令，请依照图1.3给出求解表达式“z=(x-y)*y;”所对应的指令序列（包括机器代码和对应的汇编指令）以及在主存中的存放内容，并仿照图1.5给出每条指令的执行过程以及所包含的微操作。

(20.0分)

正确答案:

3 数值数据表示的三要素?
(3.0分)

正确答案:
进位记数制
定、浮点表示
数的编码表示

答案解析:

---

4 为什么现代计算机都用补码来表示整数?
(4.0分)

正确答案:
和原码、反码相比，用补码表示有四个好处:
(1) 符号位可以和数值位一起参加运算
(2) 补码可以实现模运算，即可用加法方便地实现减法运算
(3) 零的表示唯一
(4) 可以多表示一个最小负数
答案解析:

---

5 为什么要引入浮点数表示?为什么浮点数的阶(指数)要用移码表示?现代计算机中采用什么标准来表示浮点数?
(3.0分)

正确答案:
(1) 因为定点数不能表示实数，而且表数范围小，所以要引入浮点数表示
(2) 在浮点数的加减运算中，要进行对阶操作，要比较两个阶的大小，引入移码可以简化阶的比较过程
(3) IEEE754标准
答案解析:

6 机器数与真值
(5.0分)

正确答案:
通常将数值数据在计算机内部编码表示的数称为机器数。机器数中只有0和1两种符号。
机器数真正的值(即原来带有正负号的数)称为机器数的真值。
答案解析:

---

7 数值数据与非数值数据
(5.0分)

正确答案:
数值数据是指有确定的值的数据，在数轴上能找到其对应的点，可以比较其大小。
非数值数据是指在数轴上没有确定的值的数据。
答案解析:

---

8 定点数与浮点数
(5.0分)

正确答案:
定点数：是计算机中小数点固定在最左边或最右边的数，有定点整数和定点小数两种。
浮点数：是计算机中可以指定小数点在不同位置的数。
答案解析:

---

9 算术移位与逻辑移位
(5.0分)

正确答案:
算术移位是对带符号整数进行的，移位前后符号位不变。移位时，符号位不动，只是数值部分进行移位。左移时，高位移出，末位补0，移出非符时，发生溢出；右移时，高位补符，低位移出。移出时，进行舍入操作。

逻辑移位是对无符号数进行的移位，左移时，高位移出，低位补0；右移时，低位移出，高位补0。
答案解析：

10 零扩展与符号扩展
（5.0分）

正确答案：
对无符号整数进行高位补0的操作称为零扩展。对补码整数在高位直接补符的操作称为符号扩展。
答案解析：

11 溢出与舍入
（5.0分）

正确答案：
溢出是指一个数比给定的格式所能表示的最大值还要大，或比最小值还要小的现象。
舍入是指数值数据右部的低位数据需要丢弃时，为保证丢弃后数值误差尽量小而考虑的一种操作。
答案解析：