## 第一章 习题答案

5．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 | 200×106 | 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？
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。

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. 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。

1. 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。

6．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，则程序P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？）

7．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%。

8．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8×109，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：1.25×8×109×1/4G = 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%。

9．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。

（注：从上述结果来看，对于同一个高级语言源程序，在同一台机器上所生成的目标程序不同，其执行时间可能不同，而且，并不是指令条数少的目标程序执行时间就一定少。）

10．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**。**

## 第二章 习题答案

3．实现下列各数的转换。

1. (25.8125)10= (?)2= (?) 8= (?) 16
2. (101101.011)2 = (?)10= (?) 8= (?) 16= (?) 8421
3. (0101 1001 0110.0011)8421 = (?)10= (?) 2= (?) 16
4. (4E.C)16 = (?)10= (?) 2

参考答案：

（1） (25.8125)10 = (1 1001.1101)2 = (31.64) 8 = (19.D) 16

（2）(101101.011)2 = (45.375)10 = (55.3) 8 = (2D.6) 16 = (0100 0101.0011 0111 0101) 8421

（3）(0101 1001 0110.0011)8421 = (596.3)10 = (1001010100.01001100110011…) 2 = (254.4CCC…) 16

（4）(4E.C)16 = (78.75)10 = (0100 1110.11) 2

4． 假定机器数为8位（1位符号，7位数值），写出下列各二进制数的原码和补码表示。

+0.1001，–0.1001，+1.0，–1.0，+0.010100，–0.010100，+0，–0

参考答案：

原码 补码

+0.1001： 0.1001000 0.1001000

–0.1001： 1.1001000 1.0111000

+1.0： 溢出 溢出

–1.0： 溢出 1.0000000

+0.010100： 0.0101000 0.0101000

–0.010100： 1.0101000 1.1011000

+0： 0.0000000 0.0000000

–0： 1.0000000 0.0000000

5． 假定机器数为8位（1位符号，7位数值），写出下列各二进制数的补码和移码表示。

+1001，–1001，+1，–1，+10100，–10100，+0，–0

参考答案：

移码 补码

+1001： 10001001 00001001

–1001： 01110111 11110111

+1： 10000001 00000001

–1： 011111111 11111111

+10100： 10010100 00010100

–10100： 01101100 11101100

+0： 10000000 00000000

–0： 10000000 00000000

6． 已知 [x]补，求x

（1）[x]补=1.1100111 （2）[x]补=10000000

（3）[x]补=0.1010010 （4）[x]补=11010011

参考答案：

（1）[x]补=1.1100111 x = –0.0011001B

（2）[x]补=10000000 x = –10000000B = –128

（3）[x]补=0.1010010 x = +0.101001B

（4）[x]补=11010011 x = – 101101B = – 45

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  –1 < 0  –1 < 0U  2147483647 > –2147483647 – 1  2147483647U > –2147483647 – 1  2147483647 > (int) 2147483648U  –1 > –2  (unsigned) –1 > –2 | 无符号整数  有符号整数  无符号整数  有符号整数  无符号整数  有符号整数  有符号整数  无符号整数 | 1  1  0  1  0  1  1  1 | 00…0B = 00…0B  11…1B (–1) < 00…0B (0)  11…1B (232–1) > 00…0B(0)  011…1B (231–1) > 100…0B (–231)  011…1B (231–1) < 100…0B(231)  011…1B (231–1) > 100…0B (–231)  11…1B (–1) > 11…10B (–2)  11…1B (232–1) > 11…10B (232–2) |

9．以下是一个C语言程序，用来计算一个数组a中每个元素的和。当参数len为0时，返回值应该是0，但是在机器上执行时，却发生了存储器访问异常。请问这是什么原因造成的，并说明程序应该如何修改。

1 float sum\_elements(float a[], unsigned len)

2 {

3 int i;

4 float result = 0;

5

6 for (i = 0; i <= len–1; i++)

7 result += a[i];

8 return result;

9 }

参考答案：

参数len的类型是unsigned，所以，当len=0时，执行len-1的结果为11…1，是最大可表示的无符号数，因而，任何无符号数都比它小，使得循环体被不断执行，引起数组元素的访问越界，发生存储器访问异常。

只要将len声明为int型，或循环的测试条件改为i<len。

10. 设某浮点数格式为：

阶码

S1

尾数

S1

数符

S1

1位

S1

5位移码

S1

6位补码

S1

其中，移码的偏置常数为16，补码采用一位符号位，基数为4。

1. 用这种格式表示下列十进制数：+1.7，–0.12，+19，–1/8。
2. 写出该格式浮点数的表示范围，并与12位定点补码整数表示范围比较。

参考答案：（假定采用0舍1入法进行舍入）

（1） +1.7 = +1.1011001B = 0.011011B× 41, 故阶码为1 +16 = 17 = 10001B, 尾数为+0.011011的补码， 即0.011011，所以+1.7表示为0 10001 011011。

–0.12 = – 0.000111101B = – 0.011111B × 4–1, 故阶码为 –1 + 16 =15 = 01111B, 尾数为– 0.011111的补码，即1.100001, 所以–0.12表示为1 01111 100001。

+19 = +10011B = 0.010011B× 43，故阶码为3 + 16 = 19 = 10011B, 尾数为0.010011，所以+19表示为0 10011 010011。

–1/8 = – 0.125 = – 0.001B = – 0.100000 × 4–1，阶码为 –1 + 16 = 15 = 01111B，尾数为– 0.100000的补码，即1.100000，所以–1/8表示为1 01111 100000。

（2）该格式浮点数表示的范围如下。

正数最大值：0.111111B × 411111，即：0.333× 415 （≈230 ≈109）

正数最小值：0.000001B × 400000，即：0.001× 4–16 （≈2–34≈10–10）

负数最大值：–0.000001B × 400000，即：–0.001× 4–16

负数最小值：–1.000000B × 411111，即：–1.000× 415

因此，该格式浮点数的数量级在10–10～109之间。

12位定点补码整数的表示范围为：–211～+(211–1)，即：–2048～2047

由此可见，定点数和浮点数的表示范围相差非常大。

11. 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）

尾数

S1

阶码

S1

数符

S1

8位移码

S1

7位原码

S1

1位

S1

参考答案：

（1）无符号整数：0～216–1。

（2）原码定点小数：–(1–2–15) ～ + (1–2–15)。

（3）补码定点小数：–1 ～ + (1–2–15)。

（4）补码定点整数：–32768 ～ +32767。

（5）浮点数：负数：– (1–2–7)×2+127 ～ –2–7×2–128。

正数：+2–135 ～ (1–2–7) ×2+127。

12. 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 × 20, 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 × 24，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 × 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 × 28, 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。

13．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 **0000 0000 0010**B = +1. 0000 0000 001 × 212

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。

14．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 × 230

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

15．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10的幂次表示的值 |
| 0  1  最大规格化数  最小规格化数  最大非规格化数  最小非规格化数  +∞  NaN | 00000000  01111111  11111110  00000001  00000000  00000000  11111111  11111111 | 0….00  0….00  1…11  0….00  1…11  0…01  0….00  非全0 | 0  1  (2–2–23)×2127  1.0×2–126  (1–2–23)×2–126  2–23×2–126=2–149  –  – | 0  1  a×1038  a×10–38  a×10–38  a×10–44  –  – | 0  1  (2–2–52)×21023  1.0×2–1022  (1–2–52)×2–1022  2–52×2–1022  –  – | 0  1  a×10308  a×10–308  a×10–308  a×10–?  –  – |

16．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 = 1, 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。

17．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

18．假定某计算机的总线采用奇校验，每8位数据有一位校验位，若在32位数据线上传输的信息是8F 3C AB 96H，则对应的4个校验位应为什么？若接受方收到的数据信息和校验位分别为87 3C AB 96H和0101B，则说明发生了什么情况，并给出验证过程。

参考答案：

传输信息8F 3C AB 96H展开为1000 1111 0011 1100 1010 1011 1001 0110，每8位有一个奇校验位，因此，总线上发送方送出的4个校验位应该分别为0、1、0、1。

接受方的数据信息为87 3C AB 96H，展开后为1000 0111 0011 1100 1010 1011 1001 0110；接收到的校验位分别为0、1、0、1。在接受方进行校验判断如下：

根据接收到的数据信息计算出4个奇校验位分别为1、1、0、1，将该4位校验位分别和接收到的4位校验位进行异或，得到1、0、0、0，说明数据信息的第一个字节发生传输错误。对照传输前、后的数据信息，第一字节8FH变成了87H，说明确实发生了传输错误，验证正确。

19．写出16位数据的SEC码。假定数据为0101 0001 0100 0110，说明SEC码如何正确检测数据位5的错误。

**参考答案：**

对于16位数据， 可以如下插入校验位：

M16 M15 M14 M13 M12 P5 M11 M10 M9 M8 M7 M6 M5 P4 M4 M3 M2 P3 M1 P2 P1

其中Mi是原信息数据， Pi是加入的校验位， 对于各个校验位的值可以如下计算

P1 = M1⊕M2⊕M3⊕M4⊕M5⊕M7⊕M9⊕M11⊕M12⊕M14⊕M16 = 1

P2 = M1⊕M3⊕M4⊕M6⊕M7⊕M10⊕M11⊕M13⊕M14 = 1

P3 = M2⊕M3⊕M4⊕M8⊕M9⊕M10⊕M11⊕M15⊕M16 = 0

P4 = M5⊕M6⊕M7⊕M8⊕M9⊕M10⊕M11 = 0

P5 = M12⊕M13⊕M14⊕M15⊕M16 = 0

所以此时P5 P4 P3 P2 P1 = 00011，第五位数据出错时，数据字变为：0101 0001 0101 0110，P5’P4’P3’P2’P1’= 01010，故障字 = 00011⊕01010 = 01001，说明码字第9位出错，即M5出错。

20．假设要传送的数据信息为：100011，若约定的生成多项式为：G(x)= x3+1，则校验码为多少？假定在接收端接收到的数据信息为100010，说明如何正确检测其错误，写出检测过程。

参考答案：

原数据信息为100011，对应的报文多项式为M(x) = x5 + x + 1, 生成多项式的位数为4位， 所以在原数据信息后面添加3个0，变为M’(x) = x3M(x) = x8 + x4 + x3, 用M(x)去模2除G(x)，得到的余数为111， 所以得到CRC码为100011 111。

检测时， 用接收到的CRC码去模2除生成多项式1001，若得到的余数为0，则表明正确，否则说明传输时发生了错误。此题中接收到的CRC码为100010 111（即数据100010加检验位111），显然，用100010 111 模2除 1001，得到余数为001，不为0，说明传输时发生错误。

## 第三章 习题答案

**2（4）高级语言中的运算和机器语言（即指令）中的运算是什么关系？假定某一个高级语言源程序P中有乘、除运算，但机器M中不提供乘、除运算指令，则程序P能否在机器M上运行？为什么？**

**参考答案：（略）**

**3．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，该机器使用二进制补码表示带符号整数。无符号数采用逻辑移位，带符号整数采用算术移位。请填写下表，并说明函数func1和func2的功能。**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **W** | | **func1(w)** | | **func2(w)** | |
| **机器数** | **值** | **机器数** | **值** | **机器数** | **值** |
| **0000 007FH** | **127** | **0000 007FH** | **+127** | **0000 007FH** | **+127** |
| **0000 0080H** | **128** | **0000 0080H** | **+128** | **FFFF FF80H** | **–128** |
| **0000 00FFH** | **255** | **0000 00FFH** | **+255** | **FFFF FFFFH** | **–1** |
| **0000 0100H** | **256** | **0000 0000H** | **0** | **0000 0000H** | **0** |

**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），结果一定是正的有符号数；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位后进行算术右移，高24位补符号位（即第25位）。**

**4．填写下表，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。**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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** |

**5．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：**

**可以看出x\*M和“int t = x; x << = 4; x-=t;”三句对应，这些语句实现了x乘15的功能（左移4位相当于乘以16，然后再减1），因此，M等于15；**

**y/N与“if ( y < 0 ) y += 3; y>>2;”两句对应，功能主要由第二句“y右移2位”实现，它实现了y除以4的功能，因此N是4。而第一句“if ( y < 0 ) y += 3;”主要用于对y=–1时进行调整，若不调整，则–1>>2=–1而–1/4=0，两者不等；调整后 –1+3=2，2>>2=0，两者相等。**

**思考：能否把 if ( y < 0 ) y += 3; 改成 if ( y < 0 ) y += 2; ？**

**不能！因为y = - 4时不正确。**

**6．设A4~A1和B4~B1分别是四位加法器的两组输入，C0为低位来的进位。当加法器分别采用串行进位和先行进位时，写出四个进位C4 ~C1的逻辑表达式。**

**参考答案：**

**串行进位：**

**C1 = X1C0+Y1C0 + X1 Y1**

**C2 = X2C1+Y2C1 + X2 Y2**

**C3 = X3C2+Y3C2 + X3 Y3**

**C4 = X4C3+Y4C3 + X4 Y4**

**并行进位：**

**C1 = X1Y1 + (X1+Y1)C0**

**C2 = X2Y2 + (X2 +Y2) X1Y1 + (X2+Y2) (X1+Y1)C0**

**C3 = X3Y3 + (X3 + Y3) X2Y2 + (X3 + Y3) (X2 + Y2) X1Y1 + (X3 + Y3) (X2 + Y2)(X1 + Y1)C0**

**C4=X4Y4+(X4+Y4)X3Y3+(X4+Y4)(X3+Y3)X2Y2+(X4+Y4)(X3+Y3)(X2+Y2)X1Y1+(X4+Y4)(X3+Y3) (X2+Y2)(X1+Y1)C0**

**7．用SN74181和SN74182器件设计一个16位先行进位补码加/减运算器，画出运算器的逻辑框图，并给出零标志、进位标志、溢出标志、符号标志的生成电路。**

**参考答案（图略）：**

**逻辑框图参见教材中的图3.15和图3.16，将两个图结合起来即可，也即只要将图3.15中的B输入端的每一位Bi取反，得到Bi，和原码Bi一起送到一个二路选择器，由进位C0作为选择控制信号。当C0为1时做减法，此时，选择将Bi作为SN74181的B输入端；否则，当C0为1时，做加法。**

**零标志ZF、进位标志CF、溢出标志OF、符号标志SF的逻辑电路根据以下逻辑表达式画出即可。**

**ZF=F15+F14+F13+F12+F11+F10+F9+F8+F7+F6+F5+F4+F3+F2+F1+F0**

**CF=C16**

**OF= C0（A15B15F15 + A15B15F15）+ C0（A15B15F15 + A15B15F15）**

**SF= F15**

1. **用SN74181和SN74182器件设计一个32位的ALU，要求采用两级先行进位结构。**
2. **写出所需的SN74181和SN74182芯片数。**
3. **画出32位ALU的逻辑结构图。**

**参考答案（图略）：**

**将如图3.15所示的两个16位ALU级联起来即可，级联时，低16位ALU的高位进位C16作为高16位ALU的低位进位C0，因此，只要用8片SN74181和2片SN74182。**

**9．已知x = 10，y = – 6，采用6位机器数表示。请按如下要求计算，并把结果还原成真值。**

1. **求[x+y]补，[x–y]补。**
2. **用原码一位乘法计算[x×y]原。**
3. **用MBA（基4布斯）乘法计算[x×y]补。**
4. **用不恢复余数法计算[x/y]原的商和余数。**
5. **用不恢复余数法计算[x/y]补的商和余数。**

**参考答案：**

**[10]补 = 001010 [–6]补 = 111010   [6]补 = 000110 [10]原 = 001010 [–6]原 = 100110**

**（1） [10+(– 6)]补= [10]补+[– 6]补= 001010+111010 = 000100 (+4)**

**[10–(–6)]补= [10]补+[– (–6)]补 = 001010+000110 = 010000 (+16)**

**（2） 先采用无符号数乘法计算001010× 000110的乘积，原码一位乘法过程（前面两个0省略）如下：**

**C P Y 说明**

**0 0 0 0 0 0 1 1 0 P0 = 0**

**+ 0 0 0 0 y4 = 0，+0**

**0 0 0 0 0 C, P 和Y同时右移一位**

**0 0 0 0 0 0 0 1 1 得P1**

**+ 1 0 1 0 y3 = 1，+X**

**0 1 0 1 0 C, P 和Y同时右移一位**

**0 0 1 0 1 0 0 0 1 得P2**

**+ 1 0 1 0 y2 = 1，+X**

**0 1 1 1 1 0 0 0 0 C, P 和Y同时右移一位**

**0 0 1 1 1 1 0 0 0 得P3**

**+ 0 0 0 0 y1 = 0，+0**

**0 0 1 1 1 C, P 和Y同时右移一位**

**0 0 0 1 1 1 1 0 0 得P4**

**若两个6位数相乘的话，则还要右移两次，得 000000 111100**

**符号位为：0 ⊕ 1 = 1，因此，[X×Y]原 = 1000 0011 1100**

**即X × Y = –11 1100B = – 60**

**（3） [–10]补 = 110110，布斯乘法过程如下：**

**P Y y-1 说明**

**0 0 0 0 0 0 1 1 1 0 1 0 0 设y-1 = 0，[P0]补 = 0**

**y0 y-1 = 00，P、Y直接右移一位**

**0 0 0 0 0 0 0 1 1 1 0 1 0 得[P1]补**

**+ 1 1 0 1 1 0 y1 y0 =10，+[–X]补**

**1 1 0 1 1 0 P、Y同时右移一位**

**1 1 1 0 1 1 0 0 1 1 1 0 1 得[P2]补**

**+ 0 0 1 0 1 0 y2 y1 =01，+[X]补**

**0 0 0 1 0 1 P、Y同时右移一位**

**0 0 0 0 1 0 1 0 0 1 1 1 0 得[P3]补**

**+ 1 1 0 1 1 0 1 0 0 1 1 1 0 y3 y2 = 10，+[–X]补**

**1 1 1 0 0 0 P、Y同时右移一位**

**1 1 1 1 0 0 0 1 0 0 1 1 1 得[P4]补**

**+ 0 0 0 0 0 0 0 1 0 0 1 1 1 y4 y3 = 11，+0**

**1 1 1 1 0 0 P、Y同时右移一位**

**1 1 1 1 1 0 0 0 1 0 0 1 1 得[P5]补**

**+ 0 0 0 0 0 0 0 0 1 0 0 1 1 y5 y4 = 11，+0**

**1 1 1 1 1 0 P、Y同时右移一位**

**1 1 1 1 1 1 0 0 0 1 0 0 1 得[P6]补**

**因此，[X × Y]补=1111 1100 0100，即X × Y = –11 1100B= – 60**

**（4） 因为除法计算是2n位数除n位数，所以[6]原=0110，[10]原=0000 1010，[–6]补=1010，**

**商的符号位：0 ⊕ 1 = 1，运算过程（前面两个0省略）如下：**

**余数寄存器R 余数/商寄存器Q 说 明**

**0 0 0 0 1 0 1 0 开始R0 = X**

**+ 1 0 1 0 R1 = X–Y**

**1 0 1 0 1 0 1 0 0 R1< 0，则q 4 = 0，没有溢出**

**0 1 0 1 0 1 0 0 2R1（R和Q同时左移，空出一位商）**

**+ 0 1 1 0 R2 = 2R1+Y**

**1 0 1 1 0 1 0 0 0 R2 < 0，则q 3 = 0**

**0 1 1 0 1 0 0 0 2R2 （R和Q同时左移，空出一位商）**

**+ 0 1 1 0 R3 = 2R2 +Y**

**1 1 0 0 1 0 0 0 0 R3 < 0，则q 2 = 0**

**1 0 0 1 0 0 0 0 2R3 （R和Q同时左移，空出一位商）**

**+ 0 1 1 0 R3 = 2R2 +Y**

**1 1 1 1 0 0 0 0 0 R4 < 0，则q1 = 0**

**1 1 1 0 0 0 0 0 2R4 （R和Q同时左移，空出一位商）**

**+ 0 1 1 0 R5 = 2R4 +Y**

**0 1 0 0 0 0 0 0 1 R5 > 0，则q 0 = 1**

**商的数值部分为：00001。所以，[X/Y]原=00001 (最高位为符号位)，余数为0100。**

**（5） 将10和–6分别表示成补码形式为：[10] 补 = 0 1010 , [–6] 补 = 1 1010，计算过程如下：**

**先对被除数进行符号扩展，[10] 补=00000 01010，[6] 补 = 0 0110**

**余数寄存器R 余数/商寄存器Q 说 明**

**0 0 0 0 0 0 1 0 1 0 开始R0 = [X]**

**+ 1 1 0 1 0 R1=[X] +[Y]**

**1 1 0 1 0 0 1 0 1 0 R1与[Y]同号，则q5 =1**

**1 0 1 0 0 1 0 1 0 1 2R1（R和Q同时左移，空出一位上商1）**

**+0 0 1 1 0 R2 = 2R1+[–Y]**

**1 1 0 1 0 1 0 1 0 1 R2与[Y]同号，则q4= 1，**

**1 0 1 0 1 0 1 0 1 1 2R2（R和Q同时左移，空出一位上商1）**

**+ 0 0 1 1 0 R3 = 2R2 +[-Y]**

**1 1 0 1 1 0 1 0 1 1 R3与[Y]同号，则q3 = 1**

**1 0 1 1 0 1 0 1 1 1 2R3（R和Q同时左移，空出一位上商1）**

**+ 0 0 1 1 0 R4 = 2R3 +[–Y]**

**1 1 1 0 0 1 0 1 1 1 R4与[Y]同号，则q 2 = 1**

**1 1 0 0 1 0 1 1 1 1 2R4 （R和Q同时左移，空出一位上商0）**

**+ 0 0 1 1 0 R5= 2R4 +[-Y]**

**1 1 1 1 1 0 1 1 1 1 R5与[Y]同号，则q1= 1，**

**1 1 1 1 0 1 1 1 1 1 2R5 （R和Q同时左移，空出一位上商1）**

**+ 0 0 1 1 0 R6= 2R5 +[–Y]**

**0 0 1 0 0 1 1 1 1 0 R6与[Y]异号，则q 0 = 0，Q左移，空出一位上商1**

**+ 0 0 0 0 0 + 1 商为负数，末位加1；余数不需要修正**

**0 0 1 0 0 1 1 1 1 1**

**所以，[X/Y] 补=11111，余数为00100。**

**即：X/Y= – 0001B = – 1，余数为 0100B = 4**

**将各数代入公式“除数×商+余数= 被除数”进行验证，得：(–6)×(–1) +4= 10。**

**10．若一次加法需要1ns，一次移位需要0.5ns。请分别计算用一位乘法、两位乘法、基于CRA的阵列乘法、基于CSA的阵列乘法四种方式计算两个8位无符号二进制数乘积时所需的时间。**

**参考答案：**

**一位乘法：8次右移，8次加法，共计12ns；**

**二位乘法：4次右移，4次加法，共计6ns；**

**基于CRA的阵列乘法：每一级部分积不仅依赖于上一级部分积，还依赖于上一级最终的进位，而每一级进位又是串行进行的，所以最长的路径总共经过了8+2×(8–1)=22次全加器，共计约22ns；**

**基于CSA的阵列乘法：本级进位和本级和同时传送到下一级，同级部分积之间不相互依赖，只进行O（N）次加法运算，因此，共计约8ns。**

**11．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，Eb←Eb+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 ×2k，Eb←Eb–k。左规时注意以下两点：**

**a) 尾数左移时数值部分最左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。**

**b) 执行Eb←Eb–k时，每次都在末位减1，一共减k次。**

**12．在IEEE 754浮点数运算中，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。阶码上溢时，说明结果的数值太大，无法表示；阶码下溢时，说明结果数值太小，可以把结果近似为0。**

**在进行对阶、规格化、舍入和浮点数的乘/除运算等过程中，都需要对阶码进行加、减运算，可能会发生阶码上溢或阶码下溢，因此，必须对阶码进行溢出判断。**

**（有关对阶码进行溢出判断的方法可参见教材中相关章节。）**

**13．假设浮点数格式为：阶码是4位移码，偏置常数为8，尾数是6位补码（采用双符号位），用浮点运算规则分别计算在不采用任何附加位和采用2位附加位（保护位、舍入位）两种情况下的值。（假定对阶和右规时采用就近舍入到偶数方式）**

**（1）(15/16) ×27 +(2/16) ×25 （2）(15/16) ×27–(2/16) ×25**

**（3）(15/16) ×25 +(2/16) ×27 （4）(15/16) ×25–(2/16) ×27**

**参考答案（假定采用隐藏位）：**

**X= (15/16) ×27 = 0.111100B ×27= (1.111000)2 × 26**

**Y1= (2/16) ×25 = 0.001000B ×25= (1.000000)2 × 22**

**Y2= (–2/16) ×25 = –0.001000B ×25= (–1.000000)2 × 22**

**K= (15/16) ×25 = 0.111100B ×25= (1.111000)2 × 24**

**J1= (2/16) ×27 = 0.001000B ×27= (1.000000)2 × 24**

**J2= (–2/16) ×27 = –0.001000B ×27= (–1.000000)2 × 24**

**根据题目所给的各种位数，可以得到在机器中表示为：**

**[X]浮 = 00 1110 (1)111000 [Y1]浮 = 00 1010 (1)000000 [Y2]浮 = 11 1010 (1)000000**

**[K]浮 = 00 1100 (1)111000 [J1]浮 = 00 1100 (1)000000 [J2]浮 = 11 1100 (1)000000**

**所以，E x = 1110，Mx = 00 (1). 111000 ，E y1 = 1010，My = 00(1).000000，E y2 = 1010，My = 11(1).000000**

**Ek = 1100，MK = 00 (1). 111000 ，E J1 = 1100，MJ1 = 00(1).000000，E J2 = 1100，MJ2 = 11(1).000000**

**尾数M中小数点前面有三位，前两位为数符，表示双符号，第三位加了括号，是隐藏位“1”。**

**没有附加位时的计算：**

* + 1. **X+Y1**

**[ΔE]补 = [E x]移 + [–[E y1]移]补 (mod 2n) = 1110 + 0110 = 0100**

**ΔE = 4，根据对阶规则可知需要对y1进行对阶，结果为：E y1 = E x = 1110，My 1= 000.000100**

**尾数相加：Mb = Mx + My1 = 001. 111000+ 000.000100 = 001.111100，两位符号相等，数值部分最高位为1，不需要进行规格化，所以最后结果为：E=1110，M=00(1).111100, 即(31/32) ×27**

* + 1. **X+Y2**

**[ΔE]补 = [E x]移 + [–[E y2]移]补 (mod 2n) = 1110 + 0110 = 0100;**

**ΔE = 4，根据对阶规则可知需要对y2进行对阶，结果为：E y2 = E x = 1110，My2= 111.111100**

**尾数相加：Mb = Mx + My2 = 001. 111000+ 111.111100=001.110100，两位符号相等，数值部分最高为1，不需要进行规格化，所以最后结果为：E=1110，M=00(1).110100, 即(29/32) ×27**

* + 1. **K+J1**

**[ΔE]补 = [E K]移 + [–[E J1]移]补 (mod 2n) = 1100 + 0100 = 0000;**

**ΔE = 0，根据对阶规则可知不需要进行对阶。**

**尾数相加：Mb = MK + MJ1 = 001. 111000+ 001.000000= 010.111000，两位符号不等，说明尾数溢出，需要进行右规，最后结果为：E=1101，M=00(1).011100, 即(23/32) ×26**

* + 1. **K+J2**

**[ΔE]补 = [E K]移 + [–[E J2]移]补 (mod 2n) = 1100 + 0100 = 0000;**

**ΔE = 0，根据对阶规则可知不需要进行对阶。**

**尾数相加：Mb = MK + MJ2 = 00 1. 111000+ 111.000000 = 000.111000，两位符号相等，数值部分最高位为0，需要进行左规，所以最后结果为：E=1011，M=00(1).110000, 即(7/8) ×24**

**如果有两位附加位精度上会有提高，在对阶的时候要注意小数点后就不是6位，而是8位，最后两位为保护位和舍入位。但是由于本题6位尾数已经足够，再加2位附加位，其结果是一样的。**

**14．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：**

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

**y = – 65.25 = – 1000001.01000...0B = (–1.00000101...0) 2 × 26**

**用IEEE 754标准单精度格式表示为：**

**[x]浮 = 0 01111110 10...0 [y]浮 = 1 10000101 000001010...0**

**所以，E x = 01111110，Mx = 0 (1). 1...0 ，E y = 10000101，My = 1(1).000001010...0**

**尾数Mx和My中小数点前面有两位，第一位为数符，第二位加了括号，是隐藏位“1”。**

**以下是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： [ΔE]补 = [E x]移 + [–[E y]移]补 (mod 2n) = 0111 1110 + 0111 1011 = 1111 1001**

**ΔE = –7，根据对阶规则可知需要对x进行对阶，结果为：Ex = E y = 10000101，Mx = 00.000000110...000**

**x的尾数Mx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：Mb = Mx + My = 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 = 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：根据所得尾数的形式，数值部分最高位为1，所以不需要进行规格化。**

**④ 舍入：把结果的尾数Mb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：Mb = 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Eb = 10000101。**

**最后结果为Eb = 10000101，Mb = 1(1).00000010…0，即：– 64.5。**

**（2） 0.75–(– 65.25）**

**① 对阶： [ΔE]补 = [E x]移 + [–[E y]移]补 (mod 2n) = 0111 1110 + 0111 1011 = 1111 1001**

**ΔE = -7，根据对阶规则可知需要对x进行对阶，结果为：Ex = E y = 10000110，Mx = 00.000000110...000**

**x的尾数Mx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：Mb = Mx – My = 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：根据所得尾数的形式，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Mb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：Mb = 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Eb = 10000101。**

**最后结果为Eb = 10000101，Mb = 0(1).00001000…0，即：+66。**

**思考题：对阶时发生什么情况，就可以不再继续进行计算？**

**15．假定十进制数用8421 NBCD码表示，采用十进制加法运算计算下列表达式的值，并讨论在十进制BCD码加法运算中如何判断溢出。**

**（1）234+567 （2）548+729**

**参考答案：**

**（1）234+567**

**0010 0011 0100**

**+ 0101 0110 0111**

**0111 1001 1011**

**+ 0110**

**0111 1010 0001**

**+ 0110 0000**

**1000 0000 0001 结果为： (801)10**

**（2）548+729**

**0000 0101 0100 1000**

**+ 0000 0111 0010 1001**

**0000 1100 0111 0001**

**+ 0000 0110 0000 0110**

**0001 0010 0111 0111**

**结果为： (1277)10**

**在第（2）题中，如果是采用12位数表示加数548和729，则能看出最后得到的答案是1100 0111 0111，这时就是BCD码加法溢出了。所以我们在判断的时候不能仅仅看BCD码最高位是不是丢失，而要看结果的最高4位是不是大于9，如果大于9，就可以认为是溢出了。**

**16．假定十进制数用8421 NBCD码表示，十进制运算673–356可以采用673加上(–356)的模10补码实现。画出实现上述操作的3位十进制数的BCD码减法运算线路，列出线路中所有的输入变量和输出变量。**

**参考答案：**

**[– (356) 10]补 = 0110 0100 0100**

**0110 0111 0011**

**+ 0110 0100 0100**

**1100 1011 0111**

**+ 0110 0110 0000**

**0011 0001 0111**

**最高位产生进位，因此，结果为正数：0011 0001 0111，故结果为：(+317)10**

**电路图分为两部分，一个是求出模10补码，另一个是计算以及判断输出结果的电路图（参见教材图3.33）。**

**求模10补码的电路图（RTL级）如下：**

****

**计算电路图（RTL级）如下：**

****

## 第四章 习题答案

1. 假定某计算机中有一条转移指令，采用相对寻址方式，共占两个字节，第一字节是操作码，第二字节是相对位移量（用补码表示），CPU每次从内存只能取一个字节。假设执行到某转移指令时PC的内容为200，执行该转移指令后要求转移到100开始的一段程序执行，则该转移指令第二字节的内容应该是多少？

参考答案：

因为执行到该转移指令时PC为200，所以说明该转移指令存放在200单元开始的两个字节中。因为CPU每次从内存只能取一个字节，所以每次取一个字节后PC应该加1。

该转移指令的执行过程为：取200单元中的指令操作码并译码→PC+1→取201单元的相对位移量→PC+1→计算转移目标地址。假设该转移指令第二字节为Offset，则100=200+2+Offset，即Offset = 100–202 = –102 = 10011010B

（注：没有说定长指令字，所以不一定是每条指令占2个字节。）

1. 假设地址为1200H的内存单元中的内容为12FCH，地址为12FCH的内存单元的内容为38B8H，而38B8H单元的内容为88F9H。说明以下各情况下操作数的有效地址和操作数各是多少？

（1）操作数采用变址寻址，变址寄存器的内容为12，指令中给出的形式地址为1200H。

（2）操作数采用一次间接寻址，指令中给出的地址码为1200H。

（3） 操作数采用寄存器间接寻址，指令中给出的寄存器编号为8，8号寄存器的内容为1200H。

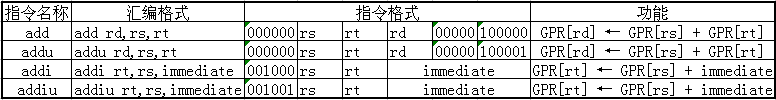
参考答案：

* 1. 有效地址EA=000CH+1200H=120CH，操作数未知。
  2. 有效地址EA=(1200H)=12FCH，操作数为38B8H。
  3. 有效地址EA=1200H，操作数为12FCH。

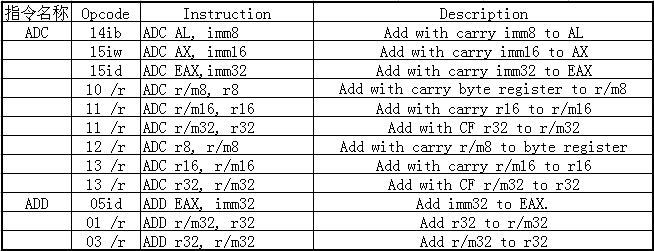
1. 通过查资料了解Intel 80x86微处理器和MIPS处理器中各自提供了哪些加法指令，说明每条加法指令的汇编形式、指令格式和功能，并比较加、减运算指令在这两种指令系统中不同的设计方式，包括不同的溢出处理方式。

参考答案（详细信息略）：

MIPS:



Intel 80x86:



1. 某计算机指令系统采用定长指令字格式，指令字长16位，每个操作数的地址码长6位。指令分二地址、单地址和零地址三类。若二地址指令有k2条，无地址指令有k0条，则单地址指令最多有多少条？

参考答案：

设单地址指令有k1条，则 ((16 – k2) ×26 – k1) ×26 = k0，所以 k1= (16 – k2) ×26 – k0/26

1. 某计算机字长16位，每次存储器访问宽度16位，CPU中有8个16位通用寄存器。现为该机设计指令系统，要求指令长度为字长的整数倍，至多支持64种不同操作，每个操作数都支持4种寻址方式：立即（I）、寄存器直接（R）、寄存器间接（S）和变址（X），存储器地址位数和立即数均为16位，任何一个通用寄存器都可作变址寄存器，支持以下7种二地址指令格式（R、I、S、X代表上述四种寻址方式）：RR型、RI型、RS型、RX型、XI型、SI型、SS型。请设计该指令系统的7种指令格式，给出每种格式的指令长度、各字段所占位数和含义，并说明每种格式指令需要几次存储器访问？

参考答案：

指令格式可以有很多种，只要满足以下的要求即可。

操作码字段：6位；寄存器编号：3位；直接地址和立即数：16位；变址寄存器编号：3位；总位数是8的倍数。

指令格式例1：

第1个部分只需要3位就可以满足7种指令格式的要求，取4位主要是满足总长度为字长的倍数。

**0000**

**OP (6位)**

**R t (3位)**

**Rs (3位)**

**RR型**

**0010**

**OP (6位)**

**Rt (3位)**

**Imm16 (16位)**

**RI型**

**0100**

**OP (6位)**

**R t (3位)**

**Rs (3位)**

**RS型**

**0110**

**OP (6位)**

**Rt (3位)**

**Offset16 (16位)**

**RX型**

**Rx (3位)**

**1000**

**OP (6位)**

**Rx (3位)**

**Imm16 (16位)**

**XI型**

**1010**

**OP (6位)**

**R t (3位)**

**000**

**SI型**

**1100**

**OP (6位)**

**Rt (3位)**

**SS型**

**Rs (3位)**

**Offset16 (16位)**

**000**

**000**

**Imm16 (16位)**

指令格式例2：分别在每个操作数前面用2位指定其寻址方式

**01**

**OP (6位)**

**R t (3位)**

**Rs (3位)**

**RR型**

**01**

**OP (6位)**

**Rt (3位)**

**Imm16 (16位)**

**RI型**

**000**

**OP (6位)**

**R t (3位)**

**Rs (3位)**

**RS型**

**OP (6位)**

**Offset16 (16位)**

**RX型**

**Rx (3位)**

**OP (6位)**

**Rx (3位)**

**Imm16 (16位)**

**XI型**

**OP (6位)**

**R t (3位)**

**000**

**SI型**

**OP (6位)**

**Rt (3位)**

**SS型**

**Rs (3位)**

**Offset16 (16位)**

**00**

**Imm16 (16位)**

**01**

**10**

**01**

**R t (3位)**

**01**

**11**

**11**

**000**

**00**

**10**

**00**

**10**

**10**

寻址方式字段(2位)----00：立即；01：寄直；10：寄间；11-变址

1. 有些计算机提供了专门的指令，能从32位寄存器中抽取其中任意一个位串置于一个寄存器的低位有效位上，并高位补0，如下图所示。MIPS指令系统中没有这样的指令，请写出最短的一个MIPS指令序列来实现这个功能，要求i=5, j=22, 操作前后的寄存器分别为$s0和$s2。

31

j

i

0

(31–j)位

(j–i)位

(i+1)位

31

0

(32–(j–i) )位

(j–i)位

0

000

参考答案：

可以先左移9位，然后右移15位，即：

sll $s2, $s0, 9

srl $s2, $s2, 15

思考：(1) 第二条用算术右移指令sra 行不行？

不行，因为不能保证高位补0！

(2) 若第一条指令中的$s2改成其他寄存器R，则会带来什么问题？

所用寄存器R的值被破坏！

1. 以下程序段是某个过程对应的指令序列。入口参数int a和int b分别置于$a0和$a1中，返回参数是该过程的结果，置于$v0中。要求为以下MIPS指令序列加注释，并简单说明该过程的功能。

add $t0, $zero, $zero

loop: beq $a1, $zero, finish

add $t0, $t0, $a0

sub $a1, $a1, 1

j loop

finish: addi $t0, $t0, 100

add $v0, $t0, $zero

参考答案：

1: 将t0寄存器置零

2: 如果a1的值等于零则程序转移到finish处

3: 将t0和a0的内容相加，结果存放于t0

4: 将a1的值减1

5: 无条件转移到loop处

6: 将t0的内容加上100，结果存放于t0

7: 将t0的值存放在v0

该程序的功能是计算“100+a×b”

1. 下列指令序列用来对两个数组进行处理，并产生结果存放在$v0中。假定每个数组有2500 个字，其数组下标为0到2499。两个数组的基地址分别存放在$a0和$a1中，数组长度分别存放在$a2和$a3中。要求为以下MIPS指令序列加注释，并简单说明该过程的功能。假定该指令序列运行在一个时钟频率为2GHz的处理器上，add、addi和sll指令的CPI为1；lw和bne指令的CPI为2，则最坏情况下运行所需时间是多少秒？

sll $a2, $a2, 2

sll $a3, $a3, 2

add $v0, $zero, $zero

add $t0, $zero, $zero

outer: add $t4, $a0, $t0

lw $t4, 0($t4)

add $t1, $zero, $zero

inner: add $t3, $a1, $t1

lw $t3, 0($t3)

bne $t3, $t4, skip

addi $v0, $v0, 1

skip: addi $t1, $t1, 4

bne $t1, $a3, inner

addi $t0, $t0, 4

bne $t0, $a2, outer

参考答案：

1: 将a2的内容左移2位，即乘4

2: 将a3的内容左移2位，即乘4

3: 将v0置零

4: 将t0置零

5: 将第一个数组的首地址存放在t4

6: 取第一个数组的第一个元素存放在t4

7: 将t1置零

8: 将第二个数组的首地址存放在t3

9: 取第二个数组的第一个元素存放在t3

10: 如果t3和t4不相等，则跳转到skip

11: 将v0的值加1，结果存于v0

12: 将t1的值加4，结果存于t1

13: 如果t1不等于a3，即还未取完数组中所有元素，则转移到inner

14: 将t0的值加4

15: 如果t0不等于a2，即还未取完数组中所有元素，则转移到outer

该程序的功能是统计两个数组中相同元素的个数。

程序最坏的情况是：两个数组所有元素都相等，这样每次循环都不会执行skip。因此，指令总条数为：5+2500×(3+2500×6+2)=37512505，

其中：add,addi和sll的指令条数为：4+2500×(2+2500×3+1)=18757504

lw和bne的指令条数为： 1+2500×(1+2500×3+1)=18755001

所以：程序执行的时间为：(2GHzclock的clock time=1/2G=0.5ns)

(18757504×1+18755001×2) × 0.5ns = 28133753ns≈0.028s

1. 用一条MIPS指令或最短的指令序列实现以下C语言语句：b=25|a。假定编译器将a和b分别分配到$t0和$t1中。如果把25换成65536，即b=65536|a，则用MIPS指令或指令序列如何实现？

参考答案：

只要用一条指令ori $t1, $t0, 25就可实现。

如果把25换成65536，则不能用一条指令ori $t1, $t0, 65536来实现，因为65536 （1 0000 0000 0000 0000B）不能用16位立即数表示。可用以下两条指令实现。

lui $t1, 1

or $t1, $t0, $t1

1. 以下程序段是某个过程对应的MIPS指令序列，其功能为复制一个存储块数据到另一个存储块中，存储块中每个数据的类型为float，源数据块和目的数据块的首地址分别存放在$a0和$a1中，复制的数据个数存放在$v0中，作为返回参数返回给调用过程。在复制过程中遇到0则停止，最后一个0也需要复制，但不被计数。已知程序段中有多个Bug，请找出它们并修改。

addi $v0, $zero, 0

loop: lw $v1, 0($a0)

sw $v1, 0($a1)

addi $a0, $a0, 4

addi $a1, $a1, 4

beq $v1, $zero, loop

参考答案：

修改后的代码如下：

addi $v0, $zero, 0

loop: lw $v1, 0($a0)

sw $v1, 0($a1)

beq $v1, $zero, exit

addi $a0, $a0, 4

addi $a1, $a1, 4

addi $v0, $v0, 1

j loop

exit:

1. 说明beq指令的含义，并解释为什么汇编程序在对下列汇编源程序中的beq指令进行汇编时会遇到问题，应该如何修改该程序段？

here: beq $s0, $s2, there

……

there: addi $s1, $s0, 4

参考答案：

beq是一个I型指令，可以跳转到当前指令前，也可以跳转到当前指令后。其转移目的地址的计算公式为：PC+4+offset×4，offset是16位带符号整数，用补码表示。因此，分支指令beq的相对转移范围如下。

其正跳范围为：0000 0000 0000 0000（4）~ 0111 1111 1111 1111（217=4+(215–1)×4）

负跳范围为：1000 0000 0000 0000（4–217=4+(–215)×4）~ 1111 1111 1111 1111（0=4+(–1)×4）

超过以上范围的跳转就不能用上述指令序列实现。

因此，上述指令序列应该改成以下指令序列：

here: bne $s0, $s2, skip

j there

skip: ……

……

there: add $s0, $s0, $s0

1. 以下C语言程序段中有两个函数sum\_array和compare，假定sum\_array函数第一个被调用，全局变量sum分配在寄存器$s0中。要求写出每个函数对应的MIPS汇编表示，并画出每个函数调用前、后栈中的状态、帧指针和栈指针的位置。

1 int sum=0;

2 int sum\_array(int array[], int num)

3 {

4 int i;

5 for (i = 0; i < num; i ++)

6 if compare (num, i+1) sum+=array[i] ;

7 return sum；

8 }

9 int compare (int a, int b)

10 {

11 if ( a >b)

12 return 1;

13 else

14 return 0;

15 }

参考答案（图略）：

程序由两个过程组成，全局静态变量sum分配给$s0。

为了尽量减少指令条数，并减少访问内存次数。在每个过程的过程体中总是先使用临时寄存器$t0~$t9，临时寄存器不够或者某个值在调用过程返回后还需要用，就使用保存寄存器$s0~$s7。

MIPS指令系统中没有寄存器传送指令，为了提高汇编表示的可读性，引入一条伪指令move来表示寄存器传送，汇编器将其转换为具有相同功能的机器指令。伪指令“move $t0, $s0”对应的机器指令为“add $t0,$zero,$s0”。

（1）过程set\_array：该过程和教材中例5.10中的不同，在例5.10中array数组是过程sum\_array的局部变量，应该在过程栈帧中给数组分配空间，但该题中的数组array是在其他过程中定义的，仅将其数组首地址作为参数传递给过程sum\_array（假定在在$a0中），因此，无需在其栈帧中给数组分配空间。此外，还有一个入口参数为num（假定在$a1中），有一个返回参数sum，被调用过程为compare。因此，其栈帧中除了保留所用的保存寄存器外，还必须保留返回地址，以免在调用过程compare时被覆盖。是否保存$fp要看具体情况，如果确保后面都不用到$fp，则可以不保存，但为了保证$fp的值不被后面的过程覆盖，通常情况下，应该保存$fp的值。

栈帧中要保存的信息只有返回地址$ra和帧指针$fp，其栈帧空间为4×2=8B。

汇编表示如下：

move $s0, $zero # sum=0

set-array: addi $sp, $sp, –8 # generate stack frame

sw $ra, 4($sp) # save $ra on stack

sw $fp, 0($sp) # save $fp on stack

addi $fp, $sp, 4 # set $fp

move $t2, $a0 # base address of array

move $t0, $a1 # $t0=num

move $t3, $zero # i=0

loop: slt $t1, $t3, $t0 # if i<num, $t1=1; if i>=num, $t1= 0

beq $t1, $zero, exit1 # if $t1= 0, jump to exit1

move $a0, $t0 # $a0=num

move $a1, $t3 # $a1=i

addi $a1, $a1, 1 # $a1=i+1

jal compare # call compare

beq $v0, $zero, else # if $v0 = 0, jump to else

sll $t1, $t3, 2 # i×4

add $t1, $t2, $t1 # $t1=array[i]

lw $t4, 0($t1) # load array[i]

add $s0, $s0, $t4 # sum+=array[i]

else： addi $t3, $t3, 1 # i=i+1

j loop

exit1: lw $ra, 4($sp) # restore $ra

lw $fp, 0($sp) # restore $fp

addi $sp, $sp, 8 # free stack frame

jr $ra # return to caller

（2）过程compare：入口参数为a和b，分别在$a0和$a1中。有一个返回参数，没有局部变量，是叶子过程，且过程体中没有用到任何保存寄存器，所以栈帧中不需要保留任何信息。

compare: move $v0, $zero # return 0

beq $a0, $a1, exit2 # if $a0=$a1, jump to exit2

slt $t1, $a0, $a1 # if $a0<$a1, $t1=1; if $a0>=$a1, $t1= 0

bne $t1, $zero, exit2 # if $a0<$a1, jump to exit2

ori $v0, $zero,1 # return 1

exit2: jr $ra

1. 以下是一个计算阶乘的C语言递归过程，请按照MIPS过程调用协议写出该递归过程对应的MIPS汇编语言程序，要求目标代码尽量短（提示：乘法运算可用乘法指令“mul rd, rs, rt”来实现，功能为“rd←(rs) ×(rt)”）。

int fact ( int n)

{

if (n < 1)

return (1) ;

else return (n\*fact (n-1) )；

}

参考答案：

Fact:

addi $sp,$sp,-8

sw $ra,4($sp)

sw $a0,0($sp)

slti $t0,$a0,1

beq $t0,$zero,L1

addi $v0,$zero,1

addi $sp,$sp,8

jr $ra

L1:

addi $a0,$a0,-1

jal Fact

lw $a0,0($sp)

lw $ra,4($sp)

addi $sp,$sp,8

mul $v0,$a0,$v0

jr $ra

## 第五章 习题答案

习题

1. 给出以下概念的解释说明。

指令周期（Instruction Cycle） 机器周期（Machine Cycle）

同步系统（Synchronous system） 时序信号（Timing signal）

控制单元（Control Unit, CU） 执行部件（Execute Unit，EU）

组合逻辑元件（Combinational logic element）或操作元件（Operate element）

时序逻辑元件（Sequential logic circuit）或状态元件（State element）

多路选择器（Multiplexor） 扩展器（Extension unit）

“零”扩展（0- extend） “符号”扩展（Sign extend）

算术逻辑部件ALU（Arithmetic Logic Unit） 加法器（Adder）

CPU总线（CPU Bus） 寄存器堆（Register file）

定时方式（Clocking methodology） 边沿触发（Edge-triggered）

寄存器写信号（Register Write） 指令存储器（Instruction Memory）

数据存储器（Data Memory） 程序计数器（Program Counter）

指令寄存器（Instruction Register） 指令译码器（Instruction Decoder）

时钟周期（Clock Cycle） 主频（CPU Clock Rate / Frequency

转移目标地址（Branch target address） 控制信号（Control signal）

微程序控制器（Microprogrammed control） 硬布线控制器（Hardwared control）

控制存储器（Control Storage，控存CS） 微代码（Microcode）

微指令（Microinstruction） 微程序（Microprogram）

固件（Firmware） 中断过程（Interrupt Processing）

异常（Exception） 故障（fault）

自陷(Trap) 终止（Abort）

中断（Interrupt） 中断服务程序（Interrupt Handler）

中断允许位（Interrupt Enable Bit） 关中断（Interrupt OFF）

开中断（Interrupt ON） 中断响应（Interrupt Response）

向量中断（Vector Interrupt） 中断向量（Interrupt vector）

中断向量表（Interrupt vector table） 向量地址（vector Address）

中断类型号（Interrupt number）

2. 简单回答下列问题。

（1）CPU的基本组成和基本功能各是什么？

（2）取指令部件的功能是什么？

（3）控制器的功能是什么？

（4）为什么对存储器按异步方式进行读写时需要WMFC信号？按同步方式访问存储器时，CPU如何实现存储器读写？

（5）单周期处理器的CPI是多少？时钟周期如何确定？为什么单周期处理器的性能差？元件在一个指令周期内能否被重复使用？为什么？

（6）多周期处理器的设计思想是什么？每条指令的CPI是否相同？为什么在一个指令周期内某个元件可被重复使用？

（7）单周期处理器和多周期处理器的控制逻辑设计的差别是什么？

（8）硬布线控制器和微程序控制器的特点各是什么？

（9）为什么CISC大多用微程序控制器实现，RISC大多用硬布线控制器实现？

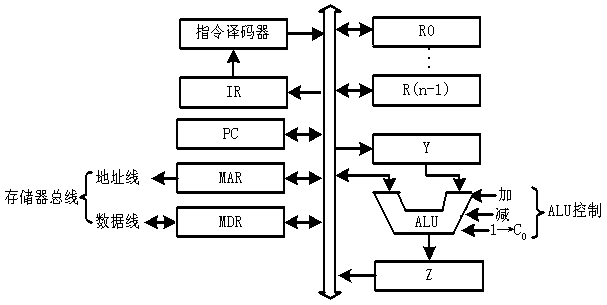
（10）水平型微指令和垂直型微指令的基本概念和优缺点是什么？

（11）CPU检测内部异常和外部中断的方法有什么不同？

3. 在书中图5.7中，假定总线传输延迟和ALU运算时间分别是20ps和200ps，寄存器建立时间为10ps，寄存器保持时间为5ps，寄存器的锁存延迟（Clk-to-Q time）为4ps，控制信号的生成延迟（Clk-to-signal time）为7ps，三态门接通时间为3ps，则从当前时钟到达开始算起，完成以下操作的最短时间是多少？各需要几个时钟周期？

（1）将数据从一个寄存器传送到另一个寄存器

（2）将程序计数器PC加1



接通三态

门时间

总线传输时间

建立时间

保持

时间

Riout

Rjin

t0 t1 t2 t3 t4

寄存器输出开始有效

Clk

Clk-to-Q

Clk-to-Signal

(a) 当前周期内不执行ALU运算

Clk-to-Signal

接通三态

门时间

总线传输时间

ALU时延

建立时间

保持

时间

Clk

Riout

Zin

t0 t1 t2 t3 t4 t5

Clk-to-Q

寄存器输出开始有效

(b) 当前周期内执行ALU运算

图6.10 单总线数据通路中主要路径的定时

参考答案：

（1）寄存器的锁存延迟Clk-to-Q time与控制信号的生成延迟的时间（Clk-to-signal time）重叠，

且Clk-to-signal time> Clk-to-Q time，所以完成寄存器传送的时间延迟为：7+3+20+10=40ps。

因为在这个过程中，只要最后保存一次信息到寄存器，所以只需要一个时钟周期。

（2）分两个阶段：

PC+1→Z ：7+3+20+200+10=240ps；

Z→PC：7+3+20+10==40ps

寄存器保持时间用来作为时间约束。

因为在这个过程中，需要经过两次总线传输，每次都将传输信息保存在某个寄存器中，所以需要两个时钟周期。

4. 右图30给出了某CPU内部结构的一部分，MAR和MDR直接连到存储器总线（图中省略）。在两个总线之间的所有数据传送都需经过算术逻辑部件ALU。ALU可实现的部分功能及其控制信号如下：

MOVa：F=A； MOVb：F=B；

a+1：F=A+1； b+1：F=B+1

a-1：F=A-1； b-1：F=B-1



其中A和B是ALU的输入，F是ALU的输出。假定JSR（转子指令）指令占两个字，第一个字是操作码，第二个字给出子程序的起始地址，返回地址保存在主存的栈中，用SP（栈指示器）指向栈顶，按字编址，每次从主存读取一个字。请写出读取并执行JSR指令所要求的控制信号序列（提示：当前指令地址在PC中）。

参考答案：

**假定采用同步方式**（**若为异步，则只需在read和Write后加一个等待信号WMFC）**

分三个阶段：

1. 取指令操作码：PCout, MOVb, MARin

Read, b+1, PCin（假设PC输出到总线后，下个时钟周期只要没有其他信号输出到总线，该信号一直有效）

MDRout, MOVb, IRin

2. 取子程序首址：PCout, MOVb, MARin

Read, b+1, Yin (返回地址送Y中)

MDRout, MOVb, **PCin**(子程序首址送PC中)

3. 保存返址至栈：SPout, MOVb, MARin

Yout, MOVb, MDRin

Write, SPout, b-1, SPin

(注：若按最长的存储访问时间作为CPU时钟周期，则上述每个阶段都需三个时钟周期)

能否用更少的时钟周期完成上述功能？不能！以下是另一种方式）

1. 取指令操作码：PCout, MOVb, MARin

Read, b+1, Yin

MDRout, MOVb, IRin

2. 取子程序首址：Yout, MOVb, MARin

Read, a+1, Yin （用b+1也行）

MDRout, MOVb, PCin

3. 保存返址至栈：SPout, MOVb, MARin

Yout, MOVb, MDRin

Write, SPout, b-1, Spin

5. 假定某计算机字长16位，CPU内部结构如书中图5.7所示，CPU和存储器之间采用同步方式通信，按字编址。采用定长指令字格式，指令由两个字组成，第一个字指明操作码和寻址方式，第二个字包含立即数Imm16。若一次存储访问所花时间为2个CPU时钟周期，每次存储访问存取一个字，取指令阶段第二次访存将Imm16取到MDR中，请写出下列指令在指令执行阶段的控制信号序列，并说明需要几个时钟周期。

（1）将立即数Imm16加到寄存器R1中，此时，Imm16为立即操作数。

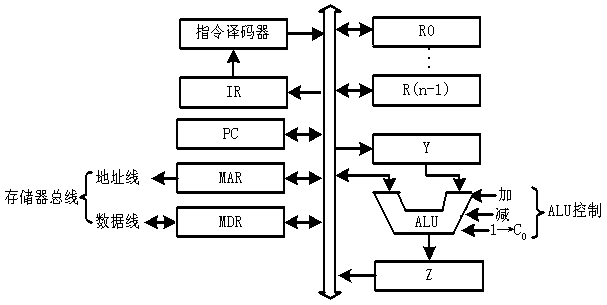
即：R[R1]←R[R1]+ Imm16

（2）将地址为Imm16的存储单元的内容加到寄存器R1中，此时，Imm16为直接地址。

即：R[R1]←R[R1]+ M[Imm16]

（3）将存储单元Imm16的内容作为地址所指的存储单元的内容加到寄存器R1中。此时，Imm16为间接地址。即：R[R1]←R[R1]+ M[M[Imm16]]

参考答案：

1. MDRout, Yin

R1out, add, Zin

Zout, R1in

需3个时钟周期

1. MDRout, MARin

Read1,（R1out, Yin也可以放在该控制信号所在的时钟周期中）

Read2, R1out, Yin

MDRout, add, Zin

Zout, R1in

需5个时钟周期

1. MDRout, MARin

Read1

Read2

MDRout, MARin

Read1,（R1out, Yin）

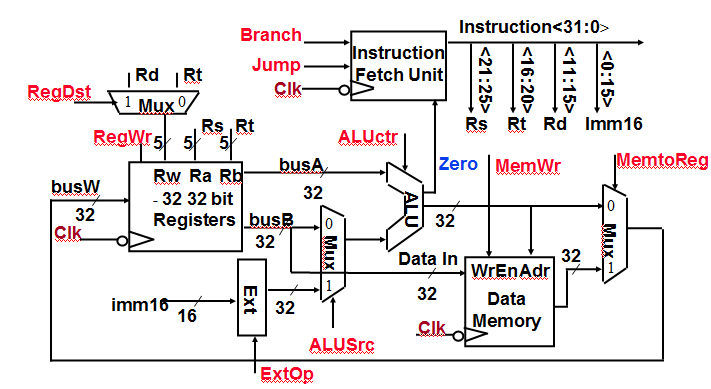
Read2, R1out, Yin

MDRout, add, Zin

Zout, R1in

需8个时钟周期

6. 假定图6.24单周期数据通路对应的控制逻辑发生错误，使得在任何情况下控制信号RegWr、RegDst、Branch、MemWr、ExtOp、R-type总是为0，则哪些指令不能正确执行？为什么？



参考答案：

|  |  |  |
| --- | --- | --- |
|  | 总是0 | 总是1 |
| RegWr | 则所有需写结果到寄存器的指令（如：R-Type指令、load指令等）都不能正确执行，因为寄存器不发生写操作 | 不需写结果到寄存器的指令可能会出错（如store,分支,转移指令等） |
| RegDst | 则所有R-Type指令都不能正确执行，因为目的寄存器指定错误 | 所有非R-Type指令都不能正确执行 |
| Branch | Branch指令可能出错，因为永远不会发生转移 | 非Branch指令都出错,因为下条指令的地址计算错误 |
| MemWr | Store指令不能正确执行，因为存储器不能写入所需数据 | 非Store指令都会出错,因为存储器内会写入错误数据 |
| ExtOp | 需要符号扩展的指令（如Beq、lw/sw,addiu等）发生错误 | 必须0扩展的指令会出错(比如ori) |

7. 假定图6.24单周期数据通路对应的控制逻辑发生错误，使得在任何情况下控制信号RegWr、RegDst、Branch、MemWr、ExtOp、R-type总是为1，则哪些指令不能正确执行？为什么？

参考答案：见第6题的表格.

8. 在MIPS指令集中需要增加一条swap指令，可以使用软件方式用若干条已有指令来实现伪指令，也可以通过改动硬件来实现。

（1）写出用伪指令方式实现“swap $rs, $rt”时的指令序列

（2）假定用硬件实现时会使一条指令的执行时间增加10%，则swap指令在程序中占多大的比例才值得用硬件方式来实现？

参考答案：

（1） swap指令可用以下三条指令实现。

xor $rs, $rs, $rt

xor $rt, $rs, $rt

xor $rs, $rs, $rt

（若使用额外寄存器$rtemp，则$rtemp内容会被破坏，所以伪指令一般不能用额外寄存器）

add $rtemp, $rs, $zero

add $rs, $rt, $zero

add $rt, $rtemp, $zero

（若使用加减法,，可能溢出。如使用无符号数加减法addu,subu也可以）

add $rs, $rs, $rt

sub $rt, $rs, $rt

add $rs, $rs, $rt

（2）假定该指令占x%，其他指令占(1-x)%

则用硬件实现该指令时, 程序执行时间为原来的1.1\*(x+1-x) =1.1 倍

用软件实现该指令时，程序执行时间为原来的3x+1-x =（2x+1）倍

当1.1 < 2x+1 时，硬件实现才有意义

由此可知，x > 5%

9. 假定图6.33多周期数据通路对应的控制逻辑发生错误，使得在任何情况下控制信号PCWr、IRWr、RegWr、BrWr、PCSource、MemWr、MemtoReg、PCWrCond、R-type总是为0，则哪些指令不能正确执行？为什么？

参考答案：

若PCWr=0，则所有指令都不正确，因为无法更新PC

若IRWr=0，则所有指令都不能正确执行，因为IR中不能写入指令

若RegWr=0，则所有需要写结果到寄存器的指令（如：R-Type指令、load指令等）都

不能正确执行，因为寄存器不发生写操作

若BrWr=0，则Branch指令不能正确执行，因为投机计算的分支地址无法送入寄存器

若PCSource=00，则除j之外的其他指令都不能正确得到下条指令地址

若MemWr=0，则Store指令不能正确执行，因为存储器不能写入数据

若MemtoReg=0，则所有Load指令执行错误，因为寄存器写入的是ALU输出

若PCWrCond=0，则Branch指令不能正确执行，因为不能写入转移目标地址到PC

若R-type=0，则所有R-type指令的执行可能出错

10. 假定P.185图6.32多周期数据通路对应的控制逻辑发生错误，使得在任何情况下控制信号PCWr、IRWr、RegWr、BrWr、PCSource、MemWr、MemtoReg、PCWrCond、R-type总是为1，则哪些指令不能正确执行？为什么？

参考答案：

若PCWr=1，则程序执行顺序失控，因为每个时钟都会更新PC

若IRWr=1，则所有指令都可能不能正确执行，因为写入IR的可能不是当前指令

若RegWr=1，则所有不需写结果到寄存器的指令（如：sw、beq等）都不能正确执行

若BrWr=1，则Branch指令不能正确执行，因为运算阶段的ALU输出也会放入寄存器，成为错误的分支转移目标地址。

若PCSource=01，则j和Branch指令不能正确得到下条指令地址

若MemWr=1，则除Store指令外的所有指令都不能正确执行

若MemtoReg=1，则除Load外的所有指令执行错误

若PCWrCond=1，则除Branch外的其他指令可能不能正确执行

若R-type=1，则所有非R-type指令的执行可能出错

12. 假定某计算机字长16位，标志寄存器Flag中的ZF、NF和VF分别是零、负和溢出标志，采用双字节定长指令字。假定Bgt (大于零转移) 指令的第一个字节指明操作码和寻址方式，第二个字节为偏移地址Imm8，其功能是：

若(ZF+(NF⊕VF)=0) 则 PC=PC+2+Imm8 否则 PC=PC+2

1. 该计算机的编址单位是什么？
2. 画出实现Bgt指令的数据通路。

参考答案：

（1）该计算机的编址单位是字节。

因为PC的增量是2，且每条指令占2个字节，所以编址单位是字节。

（2）实现Bgt指令的数据通路如下

根据“大于”条件判断表达式，可以看出该bgt指令实现的是带符号整数比较。因为无符号数比较时，其判断表达式中没有溢出标志OF。偏移地址Imm8为补码表示，转移目标地址可能在bgt指令之前，也可能在bgt指令之后。计算转移目标地址时，偏移量为Imm8, 范围为-128～127，故转移目标地址的范围是PC+2+(-128)～PC+2+127

如果偏移量为Imm8x2, 转移目标地址的范围是PC+2+(-128×2)～PC+2+127×2，其实意味着相对于bgt指令的前127条指令到后128条指令之间。

OP

MOD

Imm8

IR

PC

VF

NF

ZF

+2

Bgt

1 0

13. 对于多周期MIPS处理器，假定将访问数据的过程分成两个时钟周期可使时钟频率从4.8GHz提高到5.6GHz，但会使得lw和sw指令增加时钟周期数。已知基准程序CPUint 2000中各类指令的频率为：Load: 25%，Store: 10%，Branch: 11%，Jump: 2%，ALU: 52%。以基准程序CPUint 2000为标准计算时钟频率提高后处理器的性能提高了多少？若将取指令过程再分成两个时钟周期，则可进一步使时钟频率提高到6.4GHz，此时，时钟频率的提高是否也能带来处理器性能的提高？为什么？

参考答案：

三种频率的机器上，各类指令的百分比和 CPI

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Load | Store | Branch | Jump | ALU |
|  | 25% | 10% | 11% | 2% | 52% |
| M1 4.8GHz | 5 | 4 | 3 | 3 | 4 |
| M2 5.6GHz | 6 | 5 | 3 | 3 | 4 |
| M3 6.4GHz | 7 | 6 | 4 | 4 | 5 |

三种机器的平均CPI和MIPS

CPIofM1=25%x5+10%x4+11%x3+2%x3+52%x4=4.12 CPIofM2=25%x6+10%x5+11%x3+2%x3+52%x4=4.47

CPIofM3=25%x7+10%x6+11%x4+2%x4+52%x5=5.47

MIPSofM1=4.8G / 4.12 =1165

MIPSofM2=5.6G / 4.47 = 1253

MIPSofM3=6.4 G / 5.47 = 1170

由此可见，数据存取改为双周期的做法效果较好。进一步把取指令改为双周期的做法反而使MIPS数变小了，所以不可取。**因为数据存取只涉及到load/Store指令，而指令存取涉及到所有指令**，使得CPI显著提高。

15. 微程序控制器容量为1024×48位，微程序可在整个控存内实现转移，反映所有指令执行状态转换的有限状态机中有4个分支点，微指令采用水平格式，微地址由专门的下地址字段确定。请设计微指令的格式，说明各字段有多少位？为什么？

参考答案：

微程序控制器容量为1024×48位，说明微地址占10位，微指令字共48位，其中10位下地址字段用来给出下条微地址；转移控制字段需要对5种情况进行控制，需3位。例如，

000：取指令微指令首地址

100：根据分支1处的条件选择下条微地址

101：根据分支2处的条件选择下条微地址

110：根据分支3处的条件选择下条微地址

111：根据分支4处的条件选择下条微地址

剩下的48-10-3=35位用来表示微操作码字段。

（如果采用计数器法，则转移控制字段需要对6种情况进行控制，比上述5种情况多一种：即顺序执行下条微指令，此时，也要3位。）

也可以用5位作为转移控制字段, 33位作为微操作码字段

00001，00010,00100,01000,10000

------------------其它

16. 对于多周期CPU的异常和中断处理，回答以下问题：

（1）对于除数为0、溢出、无效指令操作码、无效指令地址、无效数据地址、缺页、访问越权和外部中断， CPU在哪些指令的哪个时钟周期能分别检测到这些异常或中断？

（2）在检测到某个异常或中断后，CPU通常要完成哪些工作？简要说明CPU如何完成这些工作？

（3）TLB缺失和cache缺失各在哪个指令的哪个时钟周期被检测到？如果检测到发生了TLB缺失和cache缺失，那么，CPU各要完成哪些工作？简要说明CPU如何完成这些工作？（提示：TLB缺失可以有软件和硬件两种处理方式。）

部分参考答案：

a. “除数为0”异常在取数/译码(ID/Reg)周期进行检测

b. “溢出”异常在R-Type指令的执行(Exe)周期进行检测

c. “无效指令”异常在取数/译码(ID/Reg)周期进行检测

d. “无效指令地址”、“缺页”和“访问越权”异常在取指令(IF)周期检测

e. “无效数据地址”、“缺页”和“访问越权”异常在存储器访问（Mem）周期检测

f. “中断”可在每条指令的最后一个周期(WB)的最后进行检测

11. 假定有一条MIPS伪指令“Bcmp $t1, $t2, $t3”，其功能是实现对两个主存块数据的比较，$t1和$t2中分别存放两个主存块的首地址，$t3中存放数据块的长度，每个数据占四个字节，若所有数据都相等，则将0置入$t1；否则，将第一次出现不相等时的地址分别置入$t1和$t2并结束比较。若$t4和$t5是两个空闲寄存器，请给出实现该伪指令的指令序列，并说明在类似于P.185图6.32的多周期数据通路中执行该伪指令时要用多少时钟周期。

参考答案：

（1）实现伪指令“bcmp $t1, $t2, $t3”的指令序列如下。

beq $t3, $zero, done # 若数据块长度为0，则结束

**compare: lw $t4, 0($t1) # 块1的当前数据取到$t4**

**lw $t5, 0($t2) # 块2的当前数据取到$t5**

**bne $t4, $t5, done # $t4和$t5的内容不等，则结束**

**addi $t1, $t1, 4 # 块1中的当前数据指向下一个**

**addi $t2, $t2, 4 # 块2中的当前数据指向下一个**

**addi $t3, $t3, -1 # 比较次数减1**

**bne $t3, $zero, compare # 若没有全部比较完，则继续比较**

addi $t1, $zero, 0 # 若全部都相等，则将$t1置0

done:

（2）在多周期数据通路执行时，上述程序段中用到的指令beq、lw、bne和addi的时钟周期数分别为3、5、3和4。若比较的数据块大小为50个字，则上述指令序列中的循环（粗体部分）最多被执行50次，因而所需的指令数最多为1+50×7+1=352。其中，load指令为50×2=100条，时钟周期数为5×100=500；branch指令数为1+2×50=101，时钟周期数为3×101=303；addi指令数为1+3×50=151，时钟周期数为4×151=604。所以，总时钟周期数最多为500+303+604=1407。

## 第六章 习题答案

1. 给出以下概念的解释说明。

指令流水线（Instruction pipelining） 流水线深度（Pipeline Depth）

指令吞吐量（Instruction throughput） 流水线冒险（Hazard）

结构冒险（Structural hazard） 控制冒险（Control hazard）

数据冒险（Data hazard） 流水线阻塞（Pipeline stall）

气泡（Bubble） 空操作（nop）

分支条件满足（Branch taken） 分支预测（Branch predict）

静态分支预测（Static predict） 动态分支预测（Dynamic predict）

延迟分支（Delayed branch） 分支延迟槽（Delayed branch slot）

转发（Forwarding） 旁路（Bypassing）

流水段寄存器（Pipeline register） IPC（Instructions Per Cycle）

静态多发射（Static multiple issue） 动态多发射（Dynamic multiple issue）

超流水线（Superpipelining） 超长指令字VLIW

超标量流水线（Superscalar） 动态流水线（Dynamic pipelining）

指令预取（Instruction prefetch） 指令分发（Instruction dispatch）

按序发射（in-order issue） 无序发射（out-of-order issue）

存储站（Reservation station） 重排序缓冲（Reorder buffer）

指令提交单元（Instruction commit unit） 乱序执行（out-of-order execution）

按序完成（in-order completion） 无序完成（out-of-order completion）

2. 简单回答下列问题。

（1）流水线方式下，一条指令的执行时间缩短了还是加长了？程序的执行时间缩短了还是加长了？为什么？

（2）具有什么特征的指令集易于实现指令流水线？

（3）流水线处理器中时钟周期如何确定？单条流水线处理器的CPI为多少？每个时钟周期一定有一条指令完成吗？为什么？

（4）流水线处理器的控制器实现方式更类似于单周期控制器还是多周期控制器？

（5）为什么要在各流水段之间加寄存器？各流水段寄存器的宽度是否都一样？为什么？

（6）你能列出哪几种流水线被阻塞的情况？你知道硬件和软件是如何处理它们的吗？

（7）超流水线和多发射流水线的主要区别是什么？

（8）静态多发射流水线和动态多发射流水线的主要区别是什么？

（9）为什么说Pentium 4是“CISC壳、RISC核”的体系结构？

3. 假定在一个五级流水线（如图7.5所示）处理器中，各主要功能单元的操作时间为：存储单元：200ps；ALU和加法器：150ps；寄存器堆读口或写口：50ps。请问：

（1）若执行阶段EX所用的ALU操作时间缩短20%，则能否加快流水线执行速度？如果能的话，能加快多少？如果不能的话，为什么？

（2）若ALU操作时间增加20%，对流水线的性能有何影响？

（3）若ALU操作时间增加40%，对流水线的性能有何影响？

参考答案：

a. ALU操作时间缩短20%不能加快流水线指令速度。因为存储单元的时间为200ps，所以流水线的时钟周期不会因为ALU操作时间的缩短而变短。

b. ALU操作时间延长20%时，变为180ps，比200ps小，对流水线性能没有影响；

c. ALU操作时间延长40%时，变为210ps，比200ps大，所以，流水线的时钟周期将变为210，其效率降低了（210-200)/200=5%。

4. 假定某计算机工程师想设计一个新CPU，一个典型程序的核心模块有一百万条指令，每条指令执行时间为100ps。请问：

（1）在非流水线处理器上执行该程序需要花多长时间？

（2）若新CPU是一个20级流水线处理器，执行上述同样的程序，理想情况下，它比非流水线处理器快多少？

（3）实际流水线并不是理想的，流水段间数据传送会有额外开销。这些开销是否会影响指令执行时间（Instruction latency）和指令吞吐率（Instruction throughput）？

参考答案：

（1）非流水线处理器上执行该程序的时间为：100ps x 106=100µs.

（2）若在一个20级流水线的处理器上执行，理想情况下，每个时钟周期为：100/20=5ps，所以，程序执行时间约为5 x 106=5µs. 快100/5=20倍。

（3）流水线段之间数据的传递产生的额外开销，使得一条指令的执行时间被延长，即影响 Instruction latency；同时也拉长了每个流水段的执行时间，即影响 Instruction throughput。

还有什么不理想的因素？

——时钟周期不会是5us

——可能发生阻塞等

5. 假定最复杂的一条指令所用的组合逻辑分成6块，依次为A~F，其延迟分别为80ps、30ps、60ps、50ps、70ps、10ps。在这些组合逻辑块之间插入必要的流水段寄存器就可实现相应的指令流水线，寄存器延迟为20ps。理想情况下，以下各种方式所得到的时钟周期、指令吞吐率和指令执行时间各是多少？应该在哪里插入流水线寄存器？

（1）插入一个流水段寄存器，得到一个两级流水线

（2）插入两个流水段寄存器，得到一个三级流水线

（3）插入三个流水段寄存器，得到一个四级流水线

（4）吞吐量最大的流水线

**参考答案：**

（1）两级流水线的平衡点在C和D之间，其前面一个流水段的组合逻辑延时为80+30+60=170ps，后面一个流水段的组合逻辑延时为50+70+10=130ps。这样每个流水段都以最长延时调整为170+20=190ps，故时钟周期为190ps，指令吞吐率为1/190ps=5.26GOPS，每条指令的执行时间为2x190=380ps。

（2）两个流水段寄存器分别插在B和C、D和E之间，这样第一个流水段的组合逻辑延时为80+30=110ps，中间第二段的时延为60+50=110ps，最后一个段延时为70+10=80ps。这样每个流水段都以最长延时调整为110+20=130ps，故时钟周期为130ps，指令吞吐率为1/130ps=7.69GOPS，每条指令的执行时间为3x130=390ps。

（3）三个流水段寄存器分别插在A和B、C和D、D和E之间，这样第一个流水段的组合逻辑延时为80ps，第二段时延为30+60=90ps，第三段时延为50ps，最后一段延时为70+10=80ps。这样每个流水段都以最长延时调整为90+20=110ps，故时钟周期为110ps，指令吞吐率为1/110ps=9.09GOPS，每条指令的执行时间为4x110=440ps。

（4）因为所有组合逻辑块中最长延时为80ps，所以，达到最大可能吞吐率的划分应该是以一个流水段延时为80ps+20ps来进行，因此，至少按五段来划分，分别把流水段寄存器插在A和B、B和C、C和D、D和E之间，这样第一段的组合逻辑延时为80ps，第二段为30ps，第三段为60ps，第四段为50ps，最后一段为70+10=80ps。这样每个流水段都以最长延时调整为80+20=100ps，故时钟周期为100ps，指令吞吐率为1/100ps=10GOPS，每条指令的执行时间为5x100=500ps。

——吞吐率的提高，单条指令执行时间的延长

6. 以下指令序列中，哪些指令对发生数据相关？假定采用“取指、译码/取数、执行、访存、写回”五段流水线方式，那么不用“转发”技术的话，需要在发生数据相关的指令前加入几条nop指令才能使这段程序避免数据冒险？如果采用“转发”是否可以完全解决数据冒险？不行的话，需要在发生数据相关的指令前加入几条nop指令才能使这段程序不发生数据冒险？

add $s3, $s1, $s0

sub $t2, $s0, $s3

lw $t1, 0($t2)

add $t1, $t1, $t2

**参考答案：**

发生数据相关的有：第1和2间关于$s3、第2和3间关于$t2、第2和4间关于$t2、第3和4间关于$t1。

不进行“转发”处理的话，需要分别在第2、3、4条指令前加三条nop指令才能避免数据冒险。而通过“转发”可以避免1和2、2和3、2和4间的数据相关；但第3和4间是load-use数据相关，所以无法用“转发”消除冒险，因此，需在第4条指令前加入一条nop指令。

寄存器写口和寄存器读口分别安排在一个时钟周期的前、后半个周期内独立工作呢？

——2、3、4条之前分别插入2条nop就可以

7. 假定以下MIPS指令序列在图7.18所示的流水线数据通路中执行：

addu $s3, $s1, $s0

subu $t2, $s0, $s3

lw $t1, 0($t2)

add $t3, $t1, $t2

add $t1, $s4, $s5

请问：（1）上述指令序列中，哪些指令的哪个寄存器需要转发，转发到何处？

（2）上述指令序列中，是否存在load-use数据冒险？

（3）第5周期结束时，各指令执行状态是什么？哪些寄存器的数据正被读出？哪些寄存器将被写入？

**参考答案：**

（1）发生数据相关的有：第1和2间关于$s3、第2和3间关于$t2、第2和4间关于$t2、第3和4间关于$t1。通过“转发”可以避免1和2、2和3、2和4间的数据相关；

（2）第3和4间是load-use数据相关，所以无法用“转发”消除冒险。

（3）第五个时钟内各条指令的执行情况如下：

指令1在“WB”阶段，控制信息等在MEM/WB.Reg中，$s3正在被写，结束时写完

指令2在“MEM”阶段，控制信息等在EX/MEM.Reg中。sub指令在该阶段进行的是空操作；在转发检测单元中，因为流水段寄存器Ex/Mem中的目的寄存器RegRd为$t2，流水段寄存器ID/Ex中的源寄存器Rs也为$t2，同时，流水段寄存器Ex/Mem中的RegWr控制信号为1，所以检测到转发条件满足，因而，此时，sub指令在上一个时钟周期中的执行结果（在流水段寄存器Ex/Mem中的ALU输出结果）正被回送到ALU的输入端；结束时转发完成

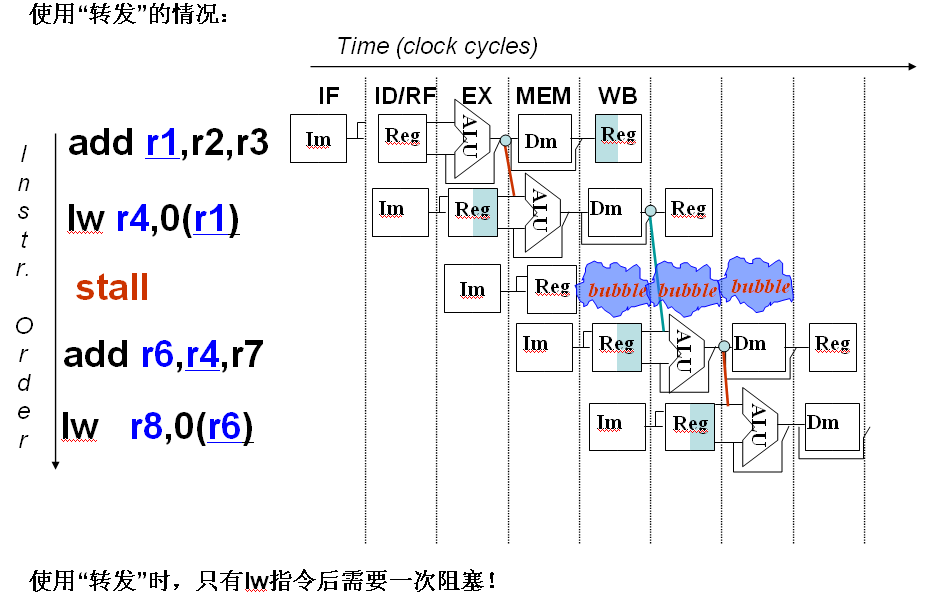
指令3在“EXE”阶段，ALU正在执行“add”操作，进行地址运算，ALU输出结果将被写入流水段寄存器Ex/Mem中；结束时运算完成。控制信息等在ID/EX.Reg中,正在检测是否loaduse冒险

指令4在“ID/REG”阶段，指令在IF/ID.Reg中，$t1和$t2正在被读出。在**load-use冒险检测单元**中，因为流水段寄存器IF/ID中源操作数寄存器Rs为$t1，流水段寄存器ID/Ex中目的操作数寄存器Rt也为$t1，同时，因为上条指令是lw，故流水段寄存器ID/Ex中的MemRead控制信号为1，所以在该阶段**检测到load-use冒险条件满足**，此时，需要进行load-use冒险处理，在流水线中插入一个“气泡”，**将指令的执行阻塞一个时钟周期**。包括以下三个步骤：① 将流水段寄存器ID/Ex中的控制信号全部清0，以保证第4条指令被阻塞一个时钟周期执行；② 将流水段寄存器IF/ID中的指令维持不变，以保证第4条指令重新译码后执行；③ 将PC的值维持不变，以保证根据PC的值重新取出第5条指令。结束时完成上述工作。

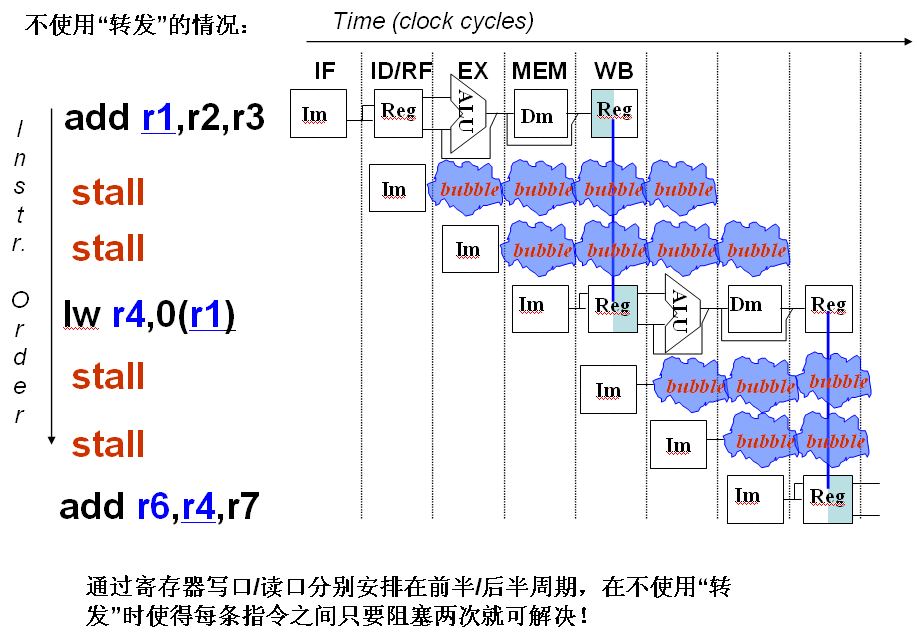
指令5在“IF”阶段，指令正被读出。结束时已送到流水段寄存器IF/ID的输入端。因为之前发生了load-use数据冒险，所以该指令将在随后的第6个时钟周期内重新被读出。8. 假定有一个程序的指令序列为“lw, add, lw, add, …”。add指令仅依赖它前面的lw指令，而lw指令也仅依赖它前面的add指令，寄存器写口和寄存器读口分别在一个时钟周期的前、后半个周期内独立工作。请问：（1）在带转发的五段流水线中执行该程序，其CPI为多少？

（2）在不带转发的五段流水线中执行该程序，其CPI为多少？

参考答案：（1）因为lw指令和add指令之间存在一个load- use数据冒险，所以每个lw指令和add指令之间要有一次流水线阻塞。而add指令和lw指令之间的数据冒险可通过数据转发解决。即：CPI为1.5



（2）如果没有转发，而寄存器写口和寄存器读口分别在一个时钟周期的前、后半个周期内工作，则在每条lw指令和add指令之间将会有两个阻塞，这样每条指令相当于都要有三个时钟才能完成。即：CPI为3



9. 假定在一个带转发功能的五段流水线中执行以下程序段，则可以怎样调整以下指令序列使其性能达到最好？

参考答案：

1 lw $2, 100($6)

2 add $2, $2, $3

3 lw $3, 200($7)

**4 add $6, $4, $7**

**5 sub $3, $4, $6**

6 lw $2, 300($8)

7 beq $2, $8, Loop

因为采用“转发”技术，所以，只要对load-use数据冒险进行指令序列调整。从上述指令序列来看，第1和第2条指令、第6和第7条指令之间存在load-use数据冒险，所以，可将与第2和第3条指令无关的第4条指令插入第2条指令之前；将无关的第5条指令插入第7条指令之前。调整顺序后的指令序列如下（粗体部分为变换了位置的指令）。

lw $2, 100($6)

add $6, $4, $7

add $2, $2, $3

lw $3, 200($7)

lw $2, 300($8)

sub $3, $4, $6

beq $2, $8, Loop

10. 在一个采用“取指、译码/取数、执行、访存、写回”的五段流水线中，若检测结果是否为“零”的操作在执行阶段进行，则分支延迟损失时间片（即分支延迟槽）为多少？以下一段MIPS指令序列中，在考虑数据转发的情况下，哪些指令执行时会发生流水线阻塞？各需要阻塞几个时钟周期？

1 loop: add $t1, $s3, $s3

2 add $t1, $t1, $t1

3 add $t1, $t1, $s6

4 lw $t0, 0($t1)

5 bne $t0, $s5, exit

6 add $s3, $s3, $s4

7 j loop

8 exit:

参考答案：

若检测操作在执行阶段进行，则分支延迟损失时间片（即分支延迟槽）为2。

分析：发生数据相关的是：第1和第2条指令之间关于$t1，第2和第3条指令之间关于$t1，第3和第4条指令之间关于$t1，第4和第5条指令之间关于$t0，以及第6和第1条指令之间关于$s3。此外，第5和第7条指令的执行都会发生控制相关。

对于数据冒险，如果不采用“转发”，而是简单地通过加入nop指令来避免冒险的话，那么应该在第2、3、4、5条指令前各加两条nop指令，以消除数据相关；对于第6条和第1条指令之间的数据相关，则可通过在第7条“j loop”指令后面加一条或两条nop指令消除（这样同时还能解决第7条“j loop”指令的控制冒险）；

此处，第2、3、4条指令所需的操作数可通过“转发”得到，无需加nop指令。第5条bne指令所需的操作数$t0是load-use冒险，不能用“转发”解决问题，需要在第5条指令前加一条nop指令，或通过硬件将第5条指令的执行阻塞一个时钟周期。

j指令如果在**译码阶段**就根据译码结果计算跳转目标地址，那么j指令后面指令会被阻塞1个时钟周期，若在**执行阶段**计算，则要阻塞2个时钟周期。

------------------------- 其它

11. 假设数据通路中各主要功能单元的操作时间为：存储单元：200ps；ALU和加法器：100ps；

寄存器堆读口或写口：50ps。程序中指令的组成比例为：取数25%、存数10%、ALU52%、分支11%、跳转2%。假设时钟周期取存储器存取时间的一半，MUX、控制单元、PC、扩展器和传输线路等的延迟都忽略不计，则下面的实现方式中，哪个更快？快多少？

（1）单周期方式：每条指令在一个固定长度的时钟周期内完成；

（2）多周期方式：每类指令时钟数：取数-7，存数-6，ALU-5，分支-4，跳转-4；

（3）流水线方式：取指1、取指2、取数/译码、执行、存取1、存取2、写回7段流水线；没有结构冒险；数据冒险采用“转发”技术处理；load指令与后续各指令之间存在依赖关系的概率分别1/2、1/4、1/8、…；分支延迟损失时间片为2，预测准确率为75%；不考虑异常、中断和访问失效引起的流水线冒险。

参考答案：

单周期：存储器操作变为两个时钟周期后，其数据通路的时钟周期不变，为**600ps**

多周期：CPI=0.25x7+0.10x6+0.52x5+0.11x4+0.02x4 = 5.47

存储器操作变为两个时钟周期后，多周期数据通路的时钟周期为100ps，

故一条指令的执行时间为100x5.47=**547ps**

流水线：存储器操作变为两个时钟周期后，其流水线包含了7个阶段.

对于ALU指令，随后的数据相关指令都可通过转发解决，故CPI=1

对于Store指令，不会发生数据冒险，故CPI=1

对于Jump指令，总要等到译码结束才能确定转移地址，故CPI=3（取指1,2，译码）

对于beq，若预测正确，则为1个周期，若预测错误，则为3个周期，故CPI=1/4x3+3/4x1=1.5

对于load，随后第一条则为3个（阻塞2个）周期；随后第二条则为2个（阻塞1个）周期，以后的指令都不需要阻塞，故CPI=1/2x3+1/4x2+2/8x1=2.25

**平均CPI为：2.25x25%+1x10%+1x52%+1.5x11%+3x2%=1.41**

所以， 1条指令的执行时间为1.41x100=**141(ps)**

12.假设有一段程序的核心模块中有五条分支指令，该模块将会被执行成千上万次，在其中一次执行过程中，五条分支指令的实际执行情况如下（T：Taken；N：not Taken）。

分支指令1（B1）：T–T–T。

分支指令2（B2）：N–N–N–N。

分支指令3（B3）：T–N–T–N–T–N。

分支指令4（B4）：T–T–T–N–T。

分支指令5（B5）：T–T–N–T–T–N–T。

假定各个分支指令在每次模块执行过程中实际执行情况都一样，并且动态预测时，每个分支指令都有各自的预测表项，每次执行时的初始预测位都相同。请给出以下几种预测方案的预测准确率。

（1）静态预测，总是预测转移 （Taken）。

（2）静态预测，总是预测不转移（not Taken）。

（3）一位动态预测，初始预测转移 （Taken）。

（4）二位动态预测，初始预测弱转移（Taken）。

**【分析解答】**

预测准确率=预测正确次数/总预测次数×100%。以下R表示正确预测次数，W表示错误预测次数。

（1）B1: R-3, W-0； B2: R-0, W-4； B3: R-3, W-3； B4: R-4, W-1； B5: R-5, W-2；60%

（2）B1: R-0, W-3； B2: R-4, W-0； B3: R-3, W-3； B4: R-1, W-4； B5: R-2, W-5；40%

（3）B1: R-3, W-0； B2: R-3, W-1； B3: R-1, W-5； B4: R-3, W-2； B5: R-3, W-4；52%

（4）B1: R-3, W-0； B2: R-3, W-1； B3: R-3, W-3； B4: R-4, W-1； B5: R-5, W-2；72%

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| B4 | T | T | T | N | T |
| 静态预测，总是预测转移 | 对 | 对 | 对 | 错 | 对 |
| 总是预测不转移 | 错 | 错 | 错 | 对 | 错 |
| 一位动态预测，初始预测转移 | 对 | 对 | 对 | 错(N) | 错 |
| 二位动态预测，初始预测弱转移 | 对(强转移) | 对(强转移) | 对(强转移) | 错(弱转移) | 对 |

## 第七章 习题答案

**2. 已知某机主存空间大小为64KB，按字节编址。要求：**

**（1）若用1K×4位的SRAM芯片构成该主存储器，需要多少个芯片？**

**（2）主存地址共多少位？几位用于选片？几位用于片内选址？**

**（3）画出该存储器的逻辑框图。**

**参考答案：**

**（1）64KB / 1K×4位 = 64×2 = 128片。**

**（2）因为是按字节编址，所以主存地址共16位，6位选片，10位片内选址。**

**（3）显然，位方向上扩展了2倍，字方向扩展了64倍。下图中片选信号CS为高电平有效。**

****

**3. 已知某主机主存的最大寻址空间为4GB,按字节编址。假定用位的具有8个位平面的DRAM芯片构成容量位512MB、传输宽度为64位的内存条。问：**



(1)每个内存条需要多少个这样的DRAM芯片？

(2)构建容量为2GB的主存时，需要几个内存条？

(3)主存地址共多少位?其中几位用于DRAM芯片内地址?哪几位时片内行地址?哪几位是片内列地址?哪几位用于选择芯片?

参考答案：

(1)每个内存条需要多少个这样的DRAM芯片？

因为DRAM芯片容量为



所以每个内存条需要



很多同学这里计算结果为64，把内存条的容量算作了512位了



(2)构建容量为2GB的主存时，需要几个内存条？

因为每个内存条容量位512MB,所以需要



(3)主存地址共多少位?其中几位用于DRAM芯片内地址?哪几位是片内行地址?哪几位是片内列地址?哪几位用于选择芯片?

因为,按字节编址则地址需要32位



芯片内总共64MB=226B=213\*213B 所以芯片内部需要26位，其中低13位为行地址，高13位为列地址。即第0-12位行地址，第13-25位为列地址。



（这里很多同学计算芯片内部的时候是按照内存条容量512MB来算的，用14位选择行，15位选择列）

主存总共有32位地址，所以需要32-26=6位，即第26-31位用于选择芯片。



（因为前面的计算错误，所以导致这里的结果也有误。）

这是用高6位做片选，存储单元地址在一个芯片内连续分配。如果用最低6位做片选，就要用多体交叉分配地址，这种结构较适合总线传输带宽是多字节的情况，比如64位宽度，可以由8个芯片同时读出放到总线上，而这8个芯片读出的8个字节单元应该形成一个连续的地址范围。

**4. 用64K×1位的DRAM芯片构成256K×8位的存储器。要求：**

**（1） 计算所需芯片数，并画出该存储器的逻辑框图。**

**（2） 若采用异步刷新方式，每单元刷新间隔不超过2ms，则产生刷新信号的间隔是多少时间？若采用集中刷新方式，则存储器刷新一遍最少用多少读写周期？**

**参考答案：**

**（1）256KB / 64K×1位 = 4×8 = 32片。存储器逻辑框图见下页（图中片选信号CS为高电平有效）。**

**（2）因为每个单元的刷新间隔为2ms，所以，采用异步刷新时，在2ms内每行必须被刷新一次，且仅被刷新一次。因为DRAM芯片存储阵列为64K=256×256，所以一共有256行。因此，存储器控制器必须每隔2ms/256=7.8µs产生一次刷新信号。采用集中刷新方式时，整个存储器刷新一遍需要256个存储（读写）周期，在这个过程中，存储器不能进行读写操作。**

****

**5. 用8K×8位的EPROM芯片组成32K×16位的只读存储器，试问：**

**（1）数据寄存器最少应有多少位？ （2） 地址寄存器最少应有多少位？**

**（3） 共需多少个EPROM芯片？ （4） 画出该只读存储器的逻辑框图。**

**参考答案：**

**（1）数据寄存器最少有16位。**

**（2）地址寄存器最少有：15位（若按16位的字编址）；16位（若按字节编址）。**

**（3）共需要 32K×16位 / 8K×8位= 4×2 = 8片。**

**（4）该只读存储器的逻辑框图如下（假定按字编址，图中片选信号CS为高电平有效）。**

****

1. **某计算机中已配有0000H～7FFFH的ROM区域，现在再用8K×4位的RAM芯片形成32K×8位的存储区域，CPU地址总线为A0-A15，数据总线为D0-D7，控制信号为R/W#（读/写）、MREQ#（访存）。要求说明地址译码方案，并画出ROM芯片、RAM芯片与CPU之间的连接图。假定上述其他条件不变，只是CPU地址线改为24根，地址范围000000H～007FFFH为ROM区，剩下的所有地址空间都用8K×4位的RAM芯片配置，则需要多少个这样的RAM芯片？**

**参考答案：**

**CPU地址线共16位，故存储器地址空间为0000H～FFFFH，其中，8000H～FFFFH为RAM区，共215=32K个单元，其空间大小为32KB，故需8K×4位的芯片数为32KB/8K×4位= 4×2 = 8片。**

**因为ROM区在0000H～7FFFH，RAM区在8000H～FFFFH，所以可通过最高位地址A15来区分，当A15为0时选中ROM芯片；为1时选中RAM芯片，此时，根据A14和A13进行译码，得到4个译码信号，分别用于4组字扩展芯片的片选信号。（图略，可参照图4.15）**

**若CPU地址线为24位，ROM区为000000H～007FFFH，则ROM区大小为32KB，总大小为16MB=214KB=512×32KB，所以RAM区大小为511×32KB，共需使用RAM芯片数为511×32KB/8K×4位=511×4×2个芯片。**

1. **假定一个存储器系统支持4体交叉存取，某程序执行过程中访问地址序列为3, 9, 17, 2, 51, 37, 13, 4, 8, 41, 67, 10，则哪些地址访问会发生体冲突？**

**参考答案：**

**对于4体交叉访问的存储系统，每个存储模块的地址分布为：**

**Bank0: 0、4、8、12、16 … …**

**Bank1: 1、5、9、13、17 …37 …41…**

**Bank2: 2、6、10、14、18 … …**

**Bank3: 3、7、11、15、19…51…67**

**如果给定的访存地址在相邻的4次访问中出现在同一个Bank内，就会发生访存冲突。所以，17和9、37和17、13和37、8和4发生冲突。**

1. **现代计算机中，SRAM一般用于实现快速小容量的cache，而DRAM用于实现慢速大容量的主存。以前超级计算机通常不提供cache，而是用SRAM来实现主存（如，Cray巨型机），请问：如果不考虑成本，你还这样设计高性能计算机吗？为什么？**

**参考答案：**

**不这样做的理由主要有以下两个方面：**

**① 主存越大越好，主存大，缺页率降低，因而减少了访问磁盘所需的时间。显然用DRAM芯片比用SRAM芯片构成的主存容量大的多。**

**② 程序访问的局部性特点使得cache的命中率很高，因而，即使主存没有用快速的SRAM芯片而是用DRAM芯片，也不会影响到访问速度。**

1. **分别给出具有下列要求的程序或程序段的示例：**

**（1）对于数据的访问，几乎没有时间局部性和空间局部性。**

**（2）对于数据的访问，有很好的时间局部性，但几乎没有空间局部性。**

**（3）对于数据的访问，有很好的空间局部性，但几乎没有时间局部性。**

**（4）对于数据的访问，空间局部性和时间局部性都好。**

**参考答案（略）：**

**可以给出许多类似的示例。例如，对于按行优先存放在内存的多维数组，如果按列优先访问数组元素，则空间局部性就差，如果在一个循环体中某个数组元素只被访问一次，则时间局部性就差。**

1. **假定某机主存空间大小1GB，按字节编址。cache的数据区（即不包括标记、有效位等存储区）有64KB，块大小为128字节，采用直接映射和全写（write-through）方式。请问：**

**（1）主存地址如何划分？要求说明每个字段的含义、位数和在主存地址中的位置。**

**（2）cache的总容量为多少位？**

**参考答案：**

**（1）主存空间大小为1GB，按字节编址，说明主存地址为30位。cache共有64KB/128B=512行，因此，行索引（行号）为9位；块大小128字节，说明块内地址为7位。因此，30位主存地址中，高14位为标志（Tag）；中间9位为行索引；低7位为块内地址。**

**（2）因为采用直接映射，所以cache中无需替换算法所需控制位，全写方式下也无需修改（dirty）位，而标志位和有效位总是必须有的，所以，cache总容量为512×(128×8+14+1)=519.5K位。**

1. **假定某计算机的cache共16行，开始为空，块大小为1个字，采用直接映射方式。CPU执行某程序时，依次访问以下地址序列：2，3，11，16，21，13，64，48，19，11，3，22，4，27，6和11。要求：**

**（1）说明每次访问是命中还是缺失，试计算访问上述地址序列的命中率。**

**（2）若cache数据区容量不变，而块大小改为4个字，则上述地址序列的命中情况又如何？**

**参考答案**

1. **cache采用直接映射方式，其数据区容量为16行×1字/行=16字；主存被划分成1字/块，所以，主存块号 = 字号。因此，映射公式为：cache行号 = 主存块号 mod 16 = 字号 mod 16。**

**开始cache为空，所以第一次都是miss，以下是映射关系（字号-cache行号）和命中情况。**

**2-2: miss，3-3: miss，11-11: miss，16-0: miss, 21-5: miss，13-13: miss，64-0: miss、replace，**

**48-0: miss、replace，19-3: miss、replace，11-11: hit, 3-3: miss、replace，22-6: miss，**

**4-4: miss，27-11: miss、replace，6-6: miss、replace，11-11: miss、replace。**

**只有一次命中！**

**（2）cache采用直接映射方式，数据区容量不变，为16个字，每块大小为4个字，所以，cache共有4行；主存被划分为4个字/块，所以，主存块号 = [字号/4]。因此，映射公式为：cache行号 = 主存块号 mod 4 = [字号/4] mod 4。**

**以下是映射关系（字号-主存块号-cache行号）和命中情况。**

**2-0-0: miss，3-0-0: hit，11-2-2: miss，16-4-0: miss、replace，21-5-1、13-3-3: miss，**

**64-16-0、48-12-0、19-4-0: miss, replace，11-2-2: hit，3-0-0: miss、replace，**

**22-5-1: hit，4-1-1: miss、replace，27-6-2: miss、replace，6-1-1: hit，11-2-2: miss、replace。**

**命中4次。**

**由此可见，块变大后，能有效利用访问的空间局部性，从而使命中率提高！**

1. **假定数组元素在主存按从左到右的下标顺序存放。试改变下列函数中循环的顺序，使得其数组元素的访问与排列顺序一致，并说明为什么修改后的程序比原来的程序执行时间短。**

**int sum\_array ( int a[N][N][N])**

**{**

**int i, j, k, sum=0;**

**for (i=0; i < N; i++)**

**for (j=0; j < N; j++)**

**for (k=0; k < N; k++) sum+=a[k][i][j];**

**return sum;**

**}**

**参考答案：**

**int sum\_array ( int a[N][N][N])**

**{**

**int i, j, k, sum=0;**

**for (k=0; k < N; k++)**

**for (i=0; i < N; i++)**

**for (j=0; j < N; j++) sum+=a[k][i][j];**

**return sum;**

**}**

**修改后程序的数组元素的访问与排列顺序一致，使得空间局部性比原程序好，故执行时间更短。**

1. **分析比较以下三个函数的空间局部性，并指出哪个最好，哪个最差？**

# define N 1000

typedef struct {

int vel[3];

int acc[3];

} point;

point p[N];

void clear3(point \*p, int n)

{

int i, j;

for (j=0; j<3; j++) {

for (i=0; i<n; i++)

p[i].vel[j] = 0;

for (i=0; i<n; i++)

p[i].acc[j] = 0;

}

}

# define N 1000

typedef struct {

int vel[3];

int acc[3];

} point;

point p[N];

void clear1(point \*p, int n)

{

int i, j;

for (i = 0; i < n; i++) {

for (j = 0; j<3; j++)

p[i].vel[j] = 0;

for (j = 0; i<3; j++)

p[i].acc[j] = 0;

}

}

# define N 1000

typedef struct {

int vel[3];

int acc[3];

} point;

point p[N];

void clear2(point \*p, int n)

{

int i, j;

for (i=0; i<n; i++) {

for (j=0; j<3; j++) {

p[i].vel[j] = 0;

p[i].acc[j] = 0;

}

}

}

**参考答案：**

**对于函数clear1，其数组访问顺序与在内存的存放顺序完全一致，因此，空间局部性最好。**

**对于函数clear2，其数组访问顺序在每个数组元素内跳越式访问，相邻两次访问的单元最大相差3个int型变量（假定sizeof(int)=4，则相当于12B），因此空间局部性比clear1差。若主存块大小比12B小的话，则大大影响命中率。**

**对于函数clear3，其数组访问顺序与在内存的存放顺序不一致，相邻两次访问的单元都相差6个int型变量（假定sizeof(int)=4，则相当于24B）因此，空间局部性比clear2还差。若主存块大小比24B小的话，则大大影响命中率。**

1. **以下是计算两个向量点积的程序段：**

**float dotproduct (float x[8], float y[8])**

**{**

**float sum = 0.0;**

**int i,;**

**for (i = 0; i < 8; i++) sum += x[i] \* y[i];**

**return sum;**

**}**

**要求：**

**（1）试分析该段代码中数组x和y的时间局部性和空间局部性，并推断命中率的高低。**

**（2）假定该段程序运行的计算机的数据cache采用直接映射方式，其数据区容量为32字节，每个主存块大小为16字节。假定编译程序将变量sum和i分配给寄存器，数组x存放在00000040H开始的32字节的连续存储区中，数组y紧跟在x后进行存放。试计算该程序数据访问的命中率，要求说明每次访问的cache命中情况。**

**（3）将上述（2）中的数据cache改用2-路组相联映射方式，块大小改为8字节，其他条件不变，则该程序数据访问的命中率是多少？**

**（4）在上述（2）中条件不变的情况下，如果将数组x定义为float[12]，则数据访问的命中率是多少？**

**参考答案：**

**（1）数组x和y都按存放顺序访问，不考虑映射的情况下，空间局部性都较好，但都只被访问一次，故没有时间局部性。命中率的高低与块大小、映射方式等都有关，所以，无法推断命中率的高低。**

**（2）cache采用直接映射方式，块大小为16字节，数据区大小为32字节，故cache共有2行。数组x的8个元素（共32B）分别存放在主存40H开始的32个单元中，共有2个主存块，其中x[0] ~ x[3]在第4块，x[4] ~ x[7]在第5块中；数组y的8个元素（共32B）分别在主存第6块和第7块中。所以，x[0] ~ x[3]和y[0] ~ y[3]都映射到cache第0行；**

**x[4] ~ x[7]和y[4] ~ y[7]都映射到cache第1行。**

|  |  |  |
| --- | --- | --- |
| **cache** | **第0-3次循环** | **第4-7次循环** |
| **第0行** | **x[0-3]，y[0-3]** |  |
| **第1行** |  | **x[4-7]，y[4-7]** |

**每调入一块，装入4个数组元素，因为x[i]和y[i]总是映射到同一行，相互淘汰对方，故每次都不命中，命中率为0.**

**（3）改用2路组相联，块大小为8B，则cache共有4行，每组两行，共两组。**

**数组x有4个主存块，x[0] ~ x[1]、x[2] ~ x[3]，x[4] ~ x[5]，x[6] ~ x[7]分别在第8~11块中；**

**数组y有4个主存块，y[0] ~ y[1]、y[2] ~ y[3]，y[4] ~ y[5]，y[6] ~ y[7]分别在第12~15块中；**

|  |  |  |
| --- | --- | --- |
| **cache** | **第0行** | **第1行** |
| **第0组** | **x[0-1],x[4-5]** | **y[0-1],y[4-5]** |
| **第1组** | **x[2-3],x[6-7]** | **y[2-3],y[6-7]** |

**每调入一块，装入两个数组元素，第二个数组元素的访问总是命中，故命中率为50%。**

**（4）若（2）中条件不变，数组x定义了12个元素，共有48B，使得y从第7块开始，因而，x[i]和y[i]就不会映射到同一个cache行中，即：x[0] ~ x[3]在第4块，x[4] ~ x[7]在第5块，x[8] ~ x[11]在第6块中，y[0] ~ y[3]在第7块，y[4] ~ x[7]在第8块。**

|  |  |  |
| --- | --- | --- |
| **cache** | **第0-3次循环** | **第4-7次循环** |
| **第0行** | **x[0-3]** | **y[4-7]** |
| **第1行** | **y[0-3]** | **x[4-7]** |

**每调入一块，装入4个数组元素，第一个元素不命中，后面3个总命中，故命中率为75%。**

1. **以下是对矩阵进行转置的程序段：**

**typedef int array[4][4];**

**void transpose(array dst, array src)**

**{**

**int i, j;**

**for (i = 0; i < 4; i++)**

**for (j = 0; j < 4; j++)**

**dst[j][i] = src[i][j];**

**}**

**假设该段程序运行的计算机中sizeof(int)=4，且只有一级cache，其中L1 data cache的数据区大小为32B，采用直接映射、写回方式，块大小为16B，初始为空。数组dst从地址0000C000H开始存放，数组src从地址0000C040H开始存放。填写下表，说明数组元素src[row][col]和dst[row][col]映射到cache的哪一行，其访问是命中（hit）还是失效（miss）。若L1 data cache的数据区容量改为128B时，重新填写表中内容。**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **src数组** | | | | **dst数组** | | | |
| **32B** | **col=0** | **col=1** | **col=2** | **col=3** | **col=0** | **col=1** | **col=2** | **col=3** |
| **row=0** | **0/miss** | **0/miss** | **0/hit** | **0/miss** | **0/miss** | **0/miss** | **0/miss** | **0/miss** |
| **row=1** | **1/miss** | **1/hit** | **1/miss** | **1/hit** | **1/miss** | **1/miss** | **1/miss** | **1/miss** |
| **row=2** | **0/miss** | **0/miss** | **0/hit** | **0/miss** | **0/miss** | **0/miss** | **0/miss** | **0/miss** |
| **row=3** | **1/miss** | **1/hit** | **1/miss** | **1/hit** | **1/miss** | **1/miss** | **1/miss** | **1/miss** |
|  | **src数组** | | | | **dst数组** | | | |
| **128B** | **col=0** | **col=1** | **col=2** | **col=3** | **col=0** | **col=1** | **col=2** | **col=3** |
| **row=0** | **4/miss** | **4/hit** | **4/hit** | **4/ hit** | **0/miss** | **0/hit** | **0/hit** | **0/hit** |
| **row=1** | **5/ miss** | **5/hit** | **5/hit** | **5/hit** | **1/miss** | **1/hit** | **1/hit** | **1/hit** |
| **row=2** | **6/ miss** | **6/hit** | **6/hit** | **6/hit** | **2/miss** | **2/hit** | **2/hit** | **2/hit** |
| **row=3** | **7/ miss** | **7/hit** | **7/hit** | **7/hit** | **3/miss** | **3/hit** | **3/hit** | **3/hit** |

**参考答案：**

**从程序来看，数组访问过程如下：**

**src[0] [0]、dst[0] [0]、src[0] [1]、dst[1] [0]、src[0] [2]、dst[2] [0]、src[0] [3]、dst[3] [0]**

**src[1] [0]、dst[0] [1]、src[1] [1]、dst[1] [1]、src[1] [2]、dst[2] [1]、src[1] [3]、dst[3] [1]**

**src[2] [0]、dst[0] [2]、src[2] [1]、dst[1] [2]、src[2] [2]、dst[2] [2]、src[2] [3]、dst[3] [2]**

**src[3] [0]、dst[0] [3]、src[3] [1]、dst[1] [3]、src[3] [2]、dst[2] [3]、src[3] [3]、dst[3] [3]**

**因为块大小为16B，每个数组元素有4个字节，所以4个数组元素占一个主存块，因此每次总是调入4个数组元素到cache的一行。**

**当数据区容量为32B时，L1 data cache中共有2行。数组元素dst[0][i]、dst[2][i] 、src[0][i]、src[2][i] (i=0～3)都映射到cache第0行，数组元素dst[1][i]、dst[3][i] 、src[1][i]、src[3][i] (i=0～3)都映射到cache第1行。因此，从上述访问过程来看，src[0][0]所在的一个主存块（即存放src[0][i] (i=0～3)四个数组元素）刚调入cache后，dst[0][0]所在主存块又把src[0][0]替换掉了。……**

**当数据区容量为128B时，L1 data cache中共有8行。数组元素dst[0][i]、dst[1][i] 、dst[2][i]、dst[3][i]、src[0][i]、src[1][i]、src[2][i]、src[3][i] (i=0～3) 分别映射到cache第0、1、2、3、4、5、6、7行。因此，不会发生数组元素的替换。每次总是第一个数组元素不命中，后面三个数组元素都命中。**

1. **通过对方格中每个点设置相应的CMYK值就可以将方格图上相应的颜色。以下三个程序段都可实现对一个8×8的方格中图上黄色的功能。**

struct pt\_color {

int c;

int m;

int y;

int k;

}

struct pt\_color square[8][8];

int i, j;

for (i = 0; i < 8; i++) {

for (j = 0; j < 8; j++) {

square[i][j].c = 0;

square[i][j].m = 0;

square[i][j].y = 1;

square[i][j].k = 0;

}

}

struct pt\_color {

int c;

int m;

int y;

int k;

}

struct pt\_color quare[8][8];

int i, j;

for (i = 0; i < 8; i++) {

for (j = 0; j < 8; j++) {

square [j] [i].c = 0;

square [j] [i].m = 0;

square [j] [i].y = 1;

square [j] [i].k = 0;

}

}

struct pt\_color {

int c;

int m;

int y;

int k;

}

struct pt\_color square[8][8];

int i, j;

for (i = 0; i < 8; i++)

for (j = 0; j < 8; j++) square[i][j].y = 1;

for (i = 0; i < 8; i++)

for (j = 0; j < 8; j++) { square[i][j].c = 0;

square[i][j].m = 0;

square[i][j].k = 0;

}

**程序段A 程序段B 程序段C**

**假设cache的数据区大小为512B，采用直接映射，块大小为32B，存储器按字节编址，sizeof(int)=4。编译时变量i和j分配在寄存器中，数组square按行优先方式存放在000008C0H开始的连续区域中，主存地址为32位。要求：**

* 1. **对三个程序段A、B、C中数组访问的时间局部性和空间局部性进行分析比较。**
  2. **画出主存中的数组元素和cache中行的对应关系图。**
  3. **计算三个程序段A、B、C中的写操作次数、写不命中次数和写缺失率。**

**参考答案：**

**（1） 对于时间局部性来说：**

**程序段A、B和C中，都是每个数组元素只被访问一次，所以都没有时间局部性；**

**对于空间局部性来说：**

**程序段A访问顺序和存放顺序一致，所以，空间局部性好；**

**程序段B访问顺序和存放顺序不一致，所以，空间局部性不好；**

**程序段C虽然访问顺序和存放顺序一致，但同一个主存块有两次访问，所以空间局部性不好；**

**（2）cache的行数为512B/32B=16；数组首地址为0000 0C80H，因为0000 0C80H正好是主存第1100100B（100）块的起始地址。所以数组从主存第100块开始存放，一个数组元素占4×4B=16B，所以每2个数组元素占用一个主存块。8×8的数组共占用32个主存块，正好是cache数据区大小的2倍。**

**主存中的数组元素与cache行的映射关系图如下：**

Cache行号

主存块号

100#

101#

102#

103#

Square[0][0]/ [0][1]

Square[0][2]/ [0][3]

Square[0][4]/ [0][5]

Square[0][6]/ [0][7]

Square[1][0]/ [1][1]

128#

129#

130#

131#

Square[7][0]/ [7][1]

Square[7][2]/ [7][3]

Square[7][4]/ [7][5]

Square[7][6]/ [7][7]

0#

1#

2#

3#

15#

4#

5#

115#

114#

116#

Square[3][4]/ [3][5]

Square[3][6]/ [3][7]

Square[4][0]/ [4][1]

**（3）对于程序段A：**

**每两个数组元素（共涉及8次写操作）装入到一个cache行中，总是第一次访问时未命中，后面7次都命中，所以，总的写操作次数为64 × 4 = 256次，写不命中次数为256×1/8 = 32次，因而写缺失率为12.5%。**

**对于程序段B：**

**每两个数组元素（共涉及8次写操作）装入到一个cache行中，但总是只有一个数组元素（涉及4次写操作）在被淘汰之前被访问，并且总是第一次不命中，后面3次命中。即写不命中次数为256×1/4 = 64次，因而写缺失率为25%。**

**对于程序段C：**

**第一个循环共64次访问，每次装入两个数组元素，第一次不命中，第二次命中；第二个循环，共访问64×3次，每两个数组元素（共涉及6次写操作）装入到一个cache行中，并且总是第一次不命中，后面5次命中。所以总的写不命中次数为32+(3×64)×1/6 = 64次，因而总缺失率为25%。**

1. **假设某计算机的主存地址空间大小为64MB，采用字节编址方式。其cache数据区容量为4KB，采用4路组相联映射方式、LRU替换和回写（write back）策略，块大小为64B。请问：**

**（1）主存地址字段如何划分？要求说明每个字段的含义、位数和在主存地址中的位置。**

**（2）该cache的总容量有多少位？**

**（3）若cache初始为空，CPU依次从0号地址单元顺序访问到4344号单元，重复按此序列共访问16次。若cache命中时间为1个时钟周期，缺失损失为10个时钟周期，则CPU访存的平均时间为多少时钟周期？**

**参考答案：**

**（1）cache的划分为：4KB = 212B = 24组×22行/组×26字节/行，所以，cache组号（组索引）占4位。**

**主存地址划分为三个字段：高16位为标志字段、中间4位为组号、最低6位为块内地址。**

**即主存空间划分为：64MB = 226B = 216组群×24块/组群×26字节/块**

**（2）cache共有64行，每行中有16位标志、1位有效位、1位修改(dirty)位、2位LRU位，以及数据64B。故总容量为64×(16+1+1+2+64×8)=34048位。**

**（3）因为每块为64B，CPU访问的单元范围为0～4344，共4345个单元，4345/64=67.89，所以CPU访问的是主存前68块（第0～67块），也即CPU的访问过程是对前68块连续访问16次，总访存次数为16×4345 = 69520。**

0

63

1

4344

128

4288

64

16次

1#

2#

67#

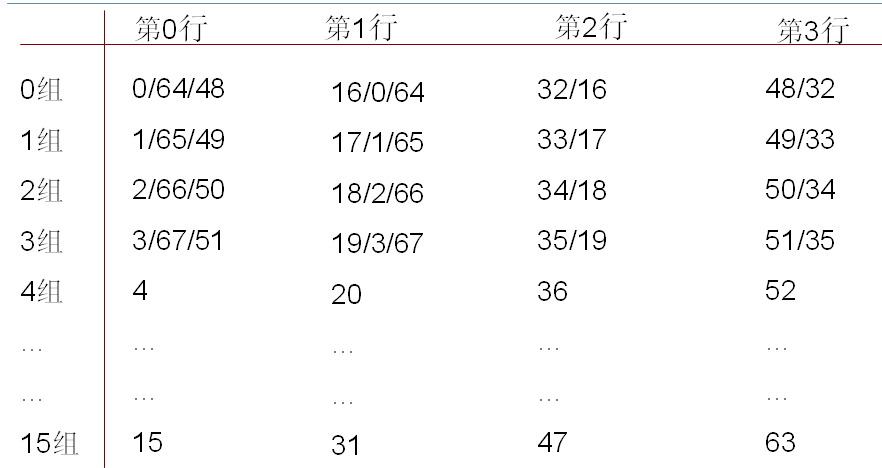
0#

65

68#

4352

**cache共有16组，每组4行，采用LRU算法的替换情况如下图所示：**

****

**根据图中所示可知，第一次循环的每一块只有第一次未命中，其余都命中；以后15次循环中，有20块的第一字未命中，其余都命中。所以命中率p为(69520–68–15×20)/69520 = 99.47%**

**平均访存时间为：Hit Time + (1–p) × Miss Penalty**

**=1+10×(1–p) = 1+0.0053×10 = 1.053个时钟周期**

1. **假定某处理器可通过软件对高速缓存设置不同的写策略，那么，在下列两种情况下，应分别设置成什么写策略？为什么？**

**（1）处理器主要运行包含大量存储器写操作的数据访问密集型应用。**

**（2）处理器运行程序的性质与（1）相同，但安全性要求高，不允许有任何数据不一致的情况发生。**

**参考答案：**

**（1）采用write back策略较好，可减少访存次数。**

**（2）采用write through策略较好，能保证数据的一致性。**

1. **已知cache1采用直接映射方式，共16行，块大小为1个字，缺失损失为8个时钟周期；cache2也采用直接映射方式，共4行，块大小为4个字，缺失损失为11个时钟周期。假定开始时cache为空，采用字编址方式。要求找出一个访问地址序列，使得cache2具有更低的缺失率，但总的缺失损失反而比cache1大。**

**参考答案：**

**假设cache1和cache2的缺失次数分别为x和y，根据题意，x和y必须满足以下条件：**

**11×y > 8×x 且 x > y，显然，满足该条件的x和y有许多，例如，x=4，y=3、x=5，y=4等等。**

**对于以下的访问地址序列：0，1，4，8，cache1缺失4次，而cache2缺失3次；**

**对于以下的访问地址序列：0，2，4，8，12，cache1缺失5次，而cache2缺失4次；**

**对于以下的访问地址序列：0，3，4，8，12，16，20，cache1缺失7次，而cache2缺失6次；**

**如此等等，可以找出很多。**

1. **提高关联度通常会降低缺失率，但并不总是这样。请给出一个地址访问序列，使得采用LRU替换算法的2-路组相联映射cache比具有同样大小的直接映射cache的缺失率更高。**

**参考答案：**

**2-路组相联cache的组数是直接映射cache的行数的一半，所以，可以找到一个地址序列A、B、C，使得：A映射到某一个cache行，B和C同时映射到另一个cache行，并且A、B、C映射到同一个cache组。这样，如果访存的地址序列为A、B、C、A、B、C、A、B、C …，则对于直接映射cache，其命中情况为：miss/miss/miss /hit/miss/miss /hit/miss/miss/… 命中率可达33.3%。**

**对于组相联cache，因为A、B、C映射到同一个组，每组只有2行，采用LRU替换算法，所以，每个地址处的数据刚调出cache就又被访问到，每次都是miss，命中率为0。**

**例如：假定直接映射cache为4行×1字/行，同样大小的2-路组相联cache为2组×2行/组×1字/行**

**当访问序列为：0、2、4、0、2、4、0、2、4、 …（局部块大小为3）时，则出现上述情况。**

**当访问的局部块大于组的大小时，可能会发生“颠簸”现象：刚被替换出去的数据又被访问，导致缺失率为100%！**

1. **假定有三个处理器，分别带有以下不同的cache：**

**cache1：采用直接映射方式，块大小为1个字，指令和数据的缺失率分别为4%和6%；**

**cache2：采用直接映射方式，块大小为4个字，指令和数据的缺失率分别为2%和4%；**

**cache3：采用2-路组相联映射方式，块大小为4个字，指令和数据的缺失率分别为2%和3%。**

**在这些处理器上运行相同的程序，该程序的CPI为2.0，其中有一半是访存指令。若缺失损失为（块大小+6）个时钟周期，处理器1和处理器2的时钟周期都为420ps，带有cache3的处理器3的时钟周期为450ps。请问：哪个处理器因cache缺失而引起的额外开销最大？哪个处理器执行速度最快？**

**参考答案：**

**假设所运行的程序共执行N条指令，每条访存指令仅读写一次内存数据，则在该程序执行过程中各处理器因cache缺失而引起的额外开销和执行时间计算如下。**

**对于处理器1：**

**额外开销为：N×(4% + 6%×50%)×(1+6) = 0.49 N个时钟周期**

**执行程序所需时间为：(N×2.0 +0.49N)×420ps = 1045.8N ps**

**对于处理器2：**

**额外开销为：N×(2%+4%×50%)×(4+6) = 0.40N个时钟周期**

**执行程序所需时间为：(N×2.0+0.40N)×420ps=1008N ps**

**对于处理器3：**

**额外开销为：N×(2%+3%×50%)×(4+6) = 0.35N个时钟周期**

**执行程序所需时间为：(N×2.0+0.35N)×450ps=1057.5N ps**

**由此可见，处理器1的cache缺失引起的额外开销最大，处理器2的执行速度最快。**

1. **假定某处理器带有一个数据区容量为256B的cache，其块大小为32B。以下C语言程序段运行在该处理器上，sizeof(int) = 4，编译器将变量i, j, c, s都分配在通用寄存器中，因此，只要考虑数组元素的访存情况。****若cache采用直接映射方式，则当s=64和s=63时，缺失率分别为多少？若cache采用2-路组相联映射方式，则当s=64和s=63时，缺失率又分别为多少？**

**int i, j, c, s, a[128];**

**……**

**for ( i = 0; i < 10000; i++ )**

**for ( j = 0; j < 128; j=j+s )**

**c = a[j];**

**参考答案：**

**已知块大小为32B，cache容量为256B = 8行×8字/行× 4B/字，仅考虑数组访问情况。**

**1) 直接映射，s=64: 访存顺序为a[0]、a[64] , a[0]、a[64], … … , 共循环10000次。这两个元素被映射到同一个cache行中，每次都会发生冲突，因此缺失率为100%。**

**2) 直接映射，s=63: 访存顺序为a[0]、a[63]、a[126], a[0]、a[63]、a[126], … …共循环10000次。这三个元素中后面两个元素因为映射到同一个cache行中，因此每次都会发生冲突，而a[0]不会发生冲突，故缺失率为67%。**

**3) 2-路组相联，s=64: 访存顺序为a[0]、a[64] , a[0]、a[64], … …, 共循环10000次。这两个元素虽然映射到同一个cache组中，但可以放在该组不同cache行中，所以不会发生冲突，缺失率为0。**

**4) 2-路组相联，s=63: 访存顺序为a[0]、a[63]、a[126], a[0]、a[63]、a[126], … …共循环10000次。这三个元素中后面两个元素虽映射到同一个cache组中，但可放在不同cache行中, 而a[0]不会发生冲突，故缺失率为0。**

1. **假定一个虚拟存储系统的虚拟地址为40位，物理地址为36位，页大小为16KB，按字节编址。若页表中有有效位、存储保护位、修改位、使用位，共占4位，磁盘地址不在页表中，则该存储系统中每个进程的页表大小为多少？如果按计算出来的实际大小构建页表，则会出现什么问题？**

**参考答案：**

**因为每页大小有16KB，所以虚拟页数为240B/16KB=2(40-14)=226页。**

**物理页面和虚拟页面大小相等，所以物理页号的位数为36–14=22位。**

**页表项位数为：有效位+保护位+修改位+使用位+物理页号位数=4+22=26位。**

**为简化页表访问，每项大小取32位。因此，每个进程的页表大小为：226×32b=256MB。**

**如果按实际计算出的页表大小构建页表，则页表过大而导致页表无法一次装入内存。**

1. **假定一个计算机系统中有一个TLB和一个L1 data cache。该系统按字节编址，虚拟地址16位，物理地址12位；页大小为128B，TLB为四路组相联，共有16个页表项；L1 data cache采用直接映射方式，块大小为4B，共16行。在系统运行到某一时刻时，TLB、页表和L1 data cache中的部分内容（用十六进制表示）如下：**

**组号 标记 页框号 有效位 标记 页框号 有效位 标记 页框号 有效位 标记 页框号 有效位**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **03** | **–** | **0** | **09** | **0D** | **1** | **00** | **–** | **0** | **07** | **02** | **1** |
| **1** | **03** | **2D** | **1** | **02** | **–** | **0** | **04** | **–** | **0** | **0A** | **–** | **0** |
| **2** | **02** | **–** | **0** | **08** | **–** | **0** | **06** | **–** | **0** | **03** | **–** | **0** |
| **3** | **07** | **–** | **0** | **63** | **0D** | **1** | **0A** | **34** | **1** | **72** | **–** | **0** |

* 1. **TLB（四路组相联）：四组、16个页表项**

**虚页号 页框号 有效位 行索引 标记 有效位 字节3 字节2 字节1 字节0**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **00** | **08** | **1** | **0** | **19** | **1** | **12** | **56** | **C9** | **AC** |
| **01** | **03** | **1** | **1** | **15** | **0** | **–** | **–** | **–** | **–** |
| **02** | **14** | **1** | **2** | **1B** | **1** | **03** | **45** | **12** | **CD** |
| **03** | **02** | **1** | **3** | **36** | **0** | **–** | **–** | **–** | **–** |
| **04** | **–** | **0** | **4** | **32** | **1** | **23** | **34** | **C2** | **2A** |
| **05** | **16** | **1** | **5** | **0D** | **1** | **46** | **67** | **23** | **3D** |
| **06** | **–** | **0** | **6** | **–** | **0** | **–** | **–** | **–** | **–** |
| **07** | **07** | **1** | **7** | **16** | **1** | **12** | **54** | **65** | **DC** |
| **08** | **13** | **1** | **8** | **24** | **1** | **23** | **62** | **12** | **3A** |
| **09** | **17** | **1** | **9** | **2D** | **0** | **–** | **–** | **–** | **–** |
| **0A** | **09** | **1** | **A** | **2D** | **1** | **43** | **62** | **23** | **C3** |
| **0B** | **–** | **0** | **B** | **–** | **0** | **–** | **–** | **–** | **–** |
| **0C** | **19** | **1** | **C** | **12** | **1** | **76** | **83** | **21** | **35** |
| **0D** | **–** | **0** | **D** | **16** | **1** | **A3** | **F4** | **23** | **11** |
| **0E** | **11** | **1** | **E** | **33** | **1** | **2D** | **4A** | **45** | **55** |
| **0F** | **0D** | **1** | **F** | **14** | **0** | **–** | **–** | **–** | **–** |

**(b) 部分页表：（开始16项） (c) L1 data cache：直接映射，共16行，块大小为4B**

**请回答下列问题：**

**（1）虚拟地址中哪几位表示虚拟页号？哪几位表示页内偏移量？虚拟页号中哪几位表示TLB标记？哪几位表示TLB索引？**

**（2）物理地址中哪几位表示物理页号？哪几位表示页内偏移量？**

**（3）主存（物理）地址如何划分成标记字段、行索引字段和块内地址字段？**

**（4）CPU从地址067AH中取出的值为多少？说明CPU读取地址067AH中内容的过程。**

**参考答案：**

**（1）16位虚拟地址中低7位为页内偏移量，高9位为虚页号；虚页号中高7位为TLB标记，低2位为TLB组索引。**

**（2）12位物理地址中低7位为页内偏移量，高5位为物理页号。**

**（3）12位物理（主存）地址中，低2位为块内地址，中间4位为cache行索引，高6位为标记。**

**（4）地址067AH=0000 0110 0111 1010B，所以，虚页号为0000011 00B，映射到TLB的第00组，将0000011B=03H与TLB第0组的四个标记比较，虽然和其中一个相等，但对应的有效位为0，其余都不等，所以TLB缺失，需要访问主存中的慢表。直接查看0000011 00B =00CH处的页表项，有效位为1，取出物理页号19H=11001B，和页内偏移111 1010B拼接成物理地址：11001 111 1010B。根据中间4位1110直接找到cache第14行(即：第E行)，有效位为1，且标记为33H=110011B，正好等于物理地址高6位，故命中。根据物理地址最低两位10，取出字节2中的内容4AH=01001010B。**

1. **缓冲区溢出是指分配的某个内存区域（缓冲区）的大小比存放内容所需空间小。例如，在栈(stack)中分配了一块空间用于存放某个过程中的一个字符串，结果字符串长度超过了分配空间的大小。黑客往往会利用缓冲区溢出来植入入侵代码。请说明可以采用什么措施来防止缓冲区溢出漏洞。**