第 1 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	200×10^6	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 $10\times5000 > 5\times8000$，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 $(10+0.003)/2\times5000 > (5+0.006)/2\times8000$，所以 M2 的性价比高，应选择 M2。
若用几何平均方式，则：因为 $\text{sqrt}(10\times0.003)\times5000 < \text{sqrt}(5\times0.006)\times8000$，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 8×10^9，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%。

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) $(0101\ 1001\ 0110.0011)_{8421}=(596.3)_{10}=(1001010100.01001100110011\ldots)_{2}=(254.4CCC\ldots)_{16}$

(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$+0.002116H \times 10^{-126}$。

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$-1.002116H \times 10^{-126}$

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. 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。

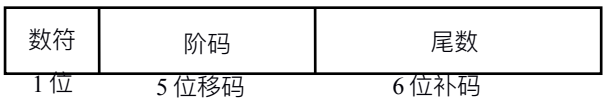
```
1    float sum_elements(float a[], unsigned len)
2    {
3        int    i;
4        float    result = 0;
5
6        for    (i = 0; i <= len–1; i++)
7            result += a[i];
8        return result;
9    }
```

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：



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× 4^1，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× 4^3，故阶码为 3 + 16 = 19 = 10011B，尾数为 0.010011，所以+19 表示为 0 10011 010011。

$-1/8 = -0.125 = -0.001B = -0.100000 \times 4^{-1}$，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：$0.111111B \times 4^{11111}$，即：$0.333 \times 4^{15}$ （$\approx 2^{30} \approx 10^{9}$）

正数最小值：$0.000001B \times 4^{00000}$，即：$0.001 \times 4^{-16}$ （$\approx 2^{-34} \approx 10^{-10}$）

负数最大值：$-0.000001B \times 4^{00000}$，即：$-0.001 \times 4^{-16}$

负数最小值：$-1.000000B \times 4^{11111}$，即：$-1.000 \times 4^{15}$

因此，该格式浮点数的数量级在 $10^{-10} \sim 10^{9}$ 之间。

12 位定点补码整数的表示范围为：$-2^{11} \sim +(2^{11}-1)$，即：$-2048 \sim 2047$

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？

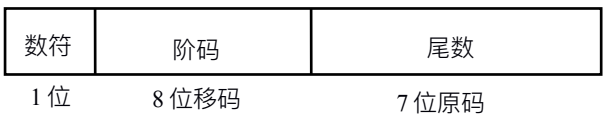
（1） 16 位无符号整数

（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）



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

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：

$+1.75 = +1.11B = 1.11B \times 2^{0}$，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。

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 × 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。

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 × 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

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	00000000	0…00	0	0	0	0
1	01111111	0…00	1	1	1	1
最大规格化数	11111110	1…11	$(2-2^{-23})\times2^{127}$	$a\times10^{38}$	$(2-2^{-52})\times2^{1023}$	$a\times10^{308}$
最小规格化数	00000001	0…00	1.0×2^{-126}	$a\times10^{-38}$	1.0×2^{-1022}	$a\times10^{-308}$
最大非规格化数	00000000	1…11	$(1-2^{-23})\times2^{-126}$	$a\times10^{-38}$	$(1-2^{-52})\times2^{-1022}$	$a\times10^{-308}$
最小非规格化数	00000000	0…01	$2^{-23}\times2^{-126}=2^{-149}$	$a\times10^{-44}$	$2^{-52}\times2^{-1022}$	$a\times10^{-?}$
+∞	11111111	0…00	–	–	–	–
NaN	11111111	非全 0	–	–	–	–

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：

$-0.125 = -0.001B = -1.0 \times 2^{-3}$

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

$7.5 = +111.1B = +1.111 \times 2^{2}$

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：

$M_{16}\ M_{15}\ M_{14}\ M_{13}\ M_{12}\ P_5\ M_{11}\ M_{10}\ M_9\ M_8\ M_7\ M_6\ M_5\ P_4 M_4\ M_3\ M_2\ P_3\ M_1\ P_2\ P_1$

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

$P_1 = M_1 \oplus M_2 \oplus M_3 \oplus M_4 \oplus M_5 \oplus M_7 \oplus M_9 \oplus M_{11} \oplus M_{12} \oplus M_{14} \oplus M_{16} = 1$

$P_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7 \oplus M_{10} \oplus M_{11} \oplus M_{13} \oplus M_{14} = 1$

$P_3 = M_2 \oplus M_3 \oplus M_4 \oplus M_8 \oplus M_9 \oplus M_{10} \oplus M_{11} \oplus M_{15} \oplus M_{16} = 0$

$P_4 = M_5 \oplus M_6 \oplus M_7 \oplus M_8 \oplus M_9 \oplus M_{10} \oplus M_{11} = 0$

$P_5 = M_{12} \oplus M_{13} \oplus M_{14} \oplus M_{15} \oplus M_{16} = 0$

所以此时 $P_5\ P_4\ P_3\ P_2\ P_1 = 00011$，第五位数据出错时，数据字变为：0101 0001 0101 0110，$P_5'P_4'P_3'P_2'P_1' = 01010$，故障字 = $00011 \oplus 01010 = 01001$，说明码字第 9 位出错，即 M_5 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：$G(x) = x^3+1$，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 $M(x) = x^5 + x + 1$，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 $M'(x) = x^3M(x) = x^8 + x^4 + x^3$，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 3 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 A_4~A_1 和 B_4~B_1 分别是四位加法器的两组输入，C_0 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 C_4~C_1 的逻辑表达式。

参考答案：

串行进位：

$$C_1 = X_1C_0+Y_1C_0 + X_1 Y_1$$

$$C_2 = X_2C_1+Y_2C_1 + X_2 Y_2$$

$$C_3 = X_3C_2+Y_3C_2 + X_3 Y_3$$

$$C_4 = X_4C_3+Y_4C_3 + X_4 Y_4$$

并行进位：

$$C_1 = X_1Y_1 + (X_1+Y_1)C_0$$

$$C_2 = X_2Y_2 + (X_2+Y_2)\ X_1Y_1 + (X_2+Y_2)\ (X_1+Y_1)C_0$$

$$C_3 = X_3Y_3 + (X_3+Y_3)\ X_2Y_2 + (X_3+Y_3)\ (X_2+Y_2)\ X_1Y_1 + (X_3+Y_3)\ (X_2+Y_2)(X_1+Y_1)C_0$$

$$C_4=X_4Y_4+(X_4+Y_4)X_3Y_3+(X_4+Y_4)(X_3+Y_3)X_2Y_2+(X_4+Y_4)(X_3+Y_3)(X_2+Y_2)X_1Y_1+(X_4+Y_4)(X_3+Y_3)(X_2+Y_2)(X_1+Y_1)C_0$$

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 B_i 取反，得到 $\overline{B_i}$，和原码 B_i 一起送到一个二路选择器，由进位 C_0 作为选择控制信号。当 C_0 为 1 时做减法，此时，选择将 $\overline{B_i}$ 作为 SN74181 的 B 输入端；否则，当 C_0 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。

$$ZF=\overline{F_{15}+F_{14}+F_{13}+F_{12}+F_{11}+F_{10}+F_9+F_8+F_7+F_6+F_5+F_4+F_3+F_2+F_1+F_0}$$

$$CF=C_{16}$$

$$OF= \overline{C_0}\ (A_{15}B_{15}\overline{F_{15}} + \overline{A_{15}}\overline{B_{15}}F_{15}) + C_0\ (A_{15}\overline{B_{15}}\overline{F_{15}} + \overline{A_{15}}B_{15}F_{15})$$

$$SF = F_{15}$$

8. 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。

（1）写出所需的 SN74181 和 SN74182 芯片数。

（2）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 C_{16} 作为高 16 位 ALU 的低位进位 C_0，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 $[x\times y]_{原}$。

（3）用 MBA（基 4 布斯）乘法计算 $[x\times y]_{补}$。

（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	$P_0=0$
	+ 0 0 0 0		$y_4=0$，+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 P_1
	+ 1 0 1 0		$y_3=1$，+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 P_2
	+ 1 0 1 0		$y_2=1$，+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 P_3
	+ 0 0 0 0		$y_1=0$，+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 P_4

若两个 6 位数相乘的话，则还要右移两次，得 000000 111100

符号位为：$0\oplus 1=1$，因此，$[X\times Y]_{原}=1000\ 0011\ 1100$

即 $X\times Y=-11\ 1100B=-60$

（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 $y_{-1}=0$，$[P_0]_{补}=0$
			$y_0\ y_{-1}=00$，P、Y 直接右移一位
0 0 0 0 0 0	0 1 1 1 0 1	0	得$[P_1]_{补}$
+ 1 1 0 1 1 0			$y_1\ y_0=10$，+$[-X]_{补}$
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$[P_2]_{补}$
+ 0 0 1 0 1 0			$y_2\ y_1=01$，+$[X]_{补}$
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$[P_3]_{补}$
+ 1 1 0 1 1 0	1 0 0 1 1 1	0	$y_3\ y_2=10$，+$[-X]_{补}$
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$[P_4]_{补}$
+ 0 0 0 0 0 0	0 1 0 0 1 1	1	$y_4\ y_3=11$，+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$[P_5]_{补}$
+ 0 0 0 0 0 0	0 0 1 0 0 1	1	$y_5\ y_4=11$，+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$[P_6]_{补}$

因此，$[X\times Y]_{补}=1111\ 1100\ 0100$，即 $X\times Y=-11\ 1100B=-60$

（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：$0\oplus 1=1$，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 $R_0=X$
+1 0 1 0		$R_1=X-Y$
1 0 1 0	1 0 1 0 0	$R_1<0$，则 $q_4=0$，没有溢出
0 1 0 1	0 1 0 0 □	$2R_1$（R 和 Q 同时左移，空出一位商）
+0 1 1 0		$R_2=2R_1+Y$
1 0 1 1	0 1 0 0 0	$R_2<0$，则 $q_3=0$
0 1 1 0	1 0 0 0 □	$2R_2$ （R 和 Q 同时左移，空出一位商）
+0 1 1 0		$R_3=2R_2+Y$
1 1 0 0	1 0 0 0 0	$R_3<0$，则 $q_2=0$
1 0 0 1	0 0 0 0 □	$2R_3$ （R 和 Q 同时左移，空出一位商）
+0 1 1 0		$R_3=2R_2+Y$
1 1 1 1	0 0 0 0 0	$R_4<0$，则 $q_1=0$
1 1 1 0	0 0 0 0 □	$2R_4$ （R 和 Q 同时左移，空出一位商）
+0 1 1 0		$R_5=2R_4+Y$
0 1 0 0	0 0 0 0 1	$R_5>0$，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 $R_0=[X]$
+1 1 0 1 0		$R_1=[X]+[Y]$
1 1 0 1 0	0 1 0 1 0	R_1 与[Y]同号，则 $q_5=1$
1 0 1 0 0	1 0 1 0 1	$2R_1$（R 和 Q 同时左移，空出一位上商 1）
+0 0 1 1 0		$R_2=2R_1+[-Y]$
1 1 0 1 0	1 0 1 0 1	R_2 与[Y]同号，则 $q_4=1$，
1 0 1 0 1	0 1 0 1 1	$2R_2$（R 和 Q 同时左移，空出一位上商 1）
+0 0 1 1 0		$R_3=2R_2+[-Y]$
1 1 0 1 1	0 1 0 1 1	R_3 与[Y]同号，则 $q_3=1$
1 0 1 1 0	1 0 1 1 1	$2R_3$（R 和 Q 同时左移，空出一位上商 1）
+0 0 1 1 0		$R_4=2R_3+[-Y]$
1 1 1 0 0	1 0 1 1 1	R_4 与[Y]同号，则 $q_2=1$
1 1 0 0 1	0 1 1 1 1	$2R_4$ （R 和 Q 同时左移，空出一位上商 0）
+0 0 1 1 0		$R_5=2R_4+[-Y]$
1 1 1 1 1	0 1 1 1 1	R_5 与[Y]同号，则 $q_1=1$，
1 1 1 1 0	1 1 1 1 1	$2R_5$ （R 和 Q 同时左移，空出一位上商 1）
+0 0 1 1 0		$R_6=2R_5+[-Y]$
0 0 1 0 0	1 1 1 1 0	R_6 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：$M_b \leftarrow M_b \times 2^{-1}$，$E_b \leftarrow E_b+1$。右规时注意以下两点：

a) 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。

b) 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：$M_b \leftarrow M_b \times 2^k$，$E_b \leftarrow E_b-k$。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 $E_b \leftarrow E_b-k$ 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）$(15/16) \times 2^7 + (2/16) \times 2^5$　　（2）$(15/16) \times 2^7 - (2/16) \times 2^5$

（3）$(15/16) \times 2^5 + (2/16) \times 2^7$　　（4）$(15/16) \times 2^5 - (2/16) \times 2^7$

参考答案（假定采用隐藏位）：

$X = (15/16) \times 2^7 = 0.111100B \times 2^7 = (1.111000)_2 \times 2^6$

$Y1 = (2/16) \times 2^5 = 0.001000B \times 2^5 = (1.000000)_2 \times 2^2$

$Y2 = (-2/16) \times 2^5 = -0.001000B \times 2^5 = (-1.000000)_2 \times 2^2$

$K = (15/16) \times 2^5 = 0.111100B \times 2^5 = (1.111000)_2 \times 2^4$

$J1 = (2/16) \times 2^7 = 0.001000B \times 2^7 = (1.000000)_2 \times 2^4$

$J2 = (-2/16) \times 2^7 = -0.001000B \times 2^7 = (-1.000000)_2 \times 2^4$

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，$E_x = 1110$，$M_x = 00\ (1).\ 111000$ ，$E_{y1} = 1010$，$M_y = 00(1).000000$，$E_{y2} = 1010$，$M_y = 11(1).000000$

$E_k = 1100$，$M_K = 00\ (1).\ 111000$ ，$E_{J1} = 1100$，$M_{J1} = 00(1).000000$，$E_{J2} = 1100$，$M_{J2} = 11(1).000000$

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

$[\Delta E]_{补} = [E_x]_{移} + [-[E_{y1}]_{移}]_{补} \pmod{2^n} = 1110 + 0110 = 0100$

$\Delta E = 4$，根据对阶规则可知需要对 y1 进行对阶，结果为：$E_{y1} = E_x = 1110$，$M_{y1} = 000.000100$

尾数相加：$M_b = M_x + M_{y1} = 001.\ 111000 + 000.000100 = 001.111100$，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$(31/32) \times 2^7$

（2） X+Y2

$[\Delta E]_{补} = [E_x]_{移} + [-[E_{y2}]_{移}]_{补} \pmod{2^n} = 1110 + 0110 = 0100$;

$\Delta E = 4$，根据对阶规则可知需要对 y2 进行对阶，结果为：$E_{y2} = E_x = 1110$，$M_{y2} = 111.111100$

尾数相加：$M_b = M_x + M_{y2} = 001.\ 111000 + 111.111100 = 001.110100$，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$(29/32) \times 2^7$

（3） K+J1

$[\Delta E]_{补} = [E_K]_{移} + [-[E_{J1}]_{移}]_{补} \pmod{2^n} = 1100 + 0100 = 0000$;

$\Delta E = 0$，根据对阶规则可知不需要进行对阶。

尾数相加：$M_b = M_K + M_{J1} = 001.\ 111000 + 001.000000 = 010.111000$，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$(23/32) \times 2^6$

（4） K+J2

$[\Delta E]_{补} = [E_K]_{移} + [-[E_{J2}]_{移}]_{补} \pmod{2^n} = 1100 + 0100 = 0000$;

$\Delta E = 0$，根据对阶规则可知不需要进行对阶。

尾数相加：$M_b = M_K + M_{J2} = 00\ 1.\ 111000 + 111.000000 = 000.111000$，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$(7/8) \times 2^4$

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：

$x = 0.75 = 0.110...0B = (1.10...0)_2 \times 2^{-1}$

$y = -65.25 = -1000001.01000...0B = (-1.00000101...0)_2 \times 2^6$

用 IEEE 754 标准单精度格式表示为：

$[x]_{浮} = 0\quad 01111110\quad 10...0$ $[y]_{浮} = 1\quad 10000101\quad 00000101 0...0$

所以，$E_x = 01111110$，$M_x = 0\ (1).\ 1...0$ ，$E_y = 10000101$，$M_y = 1(1).000001010...0$

尾数 M_x 和 M_y 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： $[\Delta E]_{补} = [E_x]_{移} + [-[E_y]_{移}]_{补} \pmod{2^n} = 0111\ 1110 + 0111\ 1011 = 1111\ 1001$

$\Delta E = -7$, 根据对阶规则可知需要对 x 进行对阶, 结果为：$E_x = E_y = 10000101$, $M_x = 00.000000110...000$

x 的尾数 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：$M_b = M_x + M_y = 00.000000110..000 + 11.000001010 ...000$ （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 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：$M_b = 11.000000100\ldots0$

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 $E_b = 10000101$，$M_b = 1(1).00000010\ldots0$，即：$-64.5$。

（2） 0.75–(– 65.25）

① 对阶： $[\Delta E]_{补} = [E_x]_{移} + [-[E_y]_{移}]_{补} \pmod{2^n} = 0111\ 1110 + 0111\ 1011 = 1111\ 1001$

$\Delta E = -7$，根据对阶规则可知需要对 x 进行对阶，结果为：$E_x = E_y = 10000110$，$M_x = 00.000000110...000$

x 的尾数 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：$00.000000110...000 - 11.000001010...000 = 01.00001000\ldots000$

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 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：$M_b = 01.00001000\ldots0$

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 $E_b = 10000101$，$M_b = 0(1).00001000\ldots0$，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：

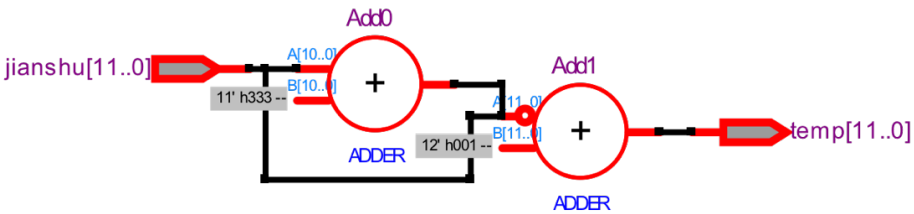
$[-(356)_{10}]_{补} = 0110\ 0100\ 0100$

$$\begin{array}{r} 0110\ 0111\ 0011 \\ +\ 0110\ 0100\ 0100 \\ \hline 1100\ 1011\ 0111 \\ +\ 0110\ 0110\ 0000 \\ \hline 0011\ 0001\ 0111 \end{array}$$

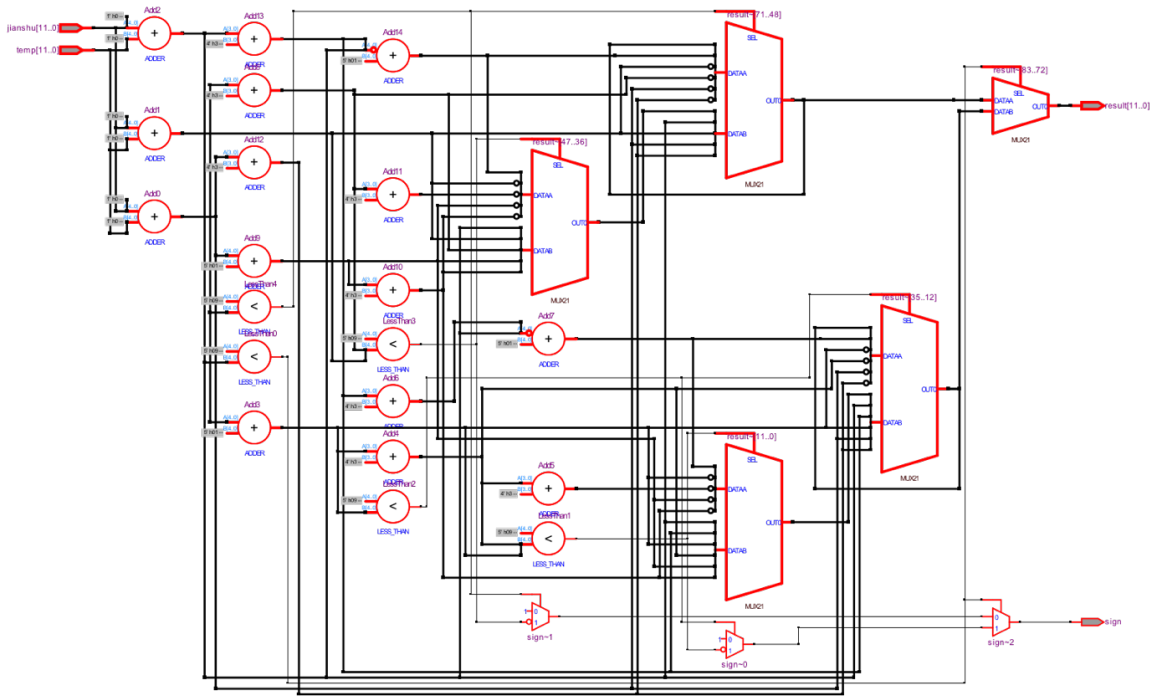
最高位产生进位，因此，结果为正数：0011 0001 0111，故结果为：$(+317)_{10}$

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 4 章　习　题 答 案

3. 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？

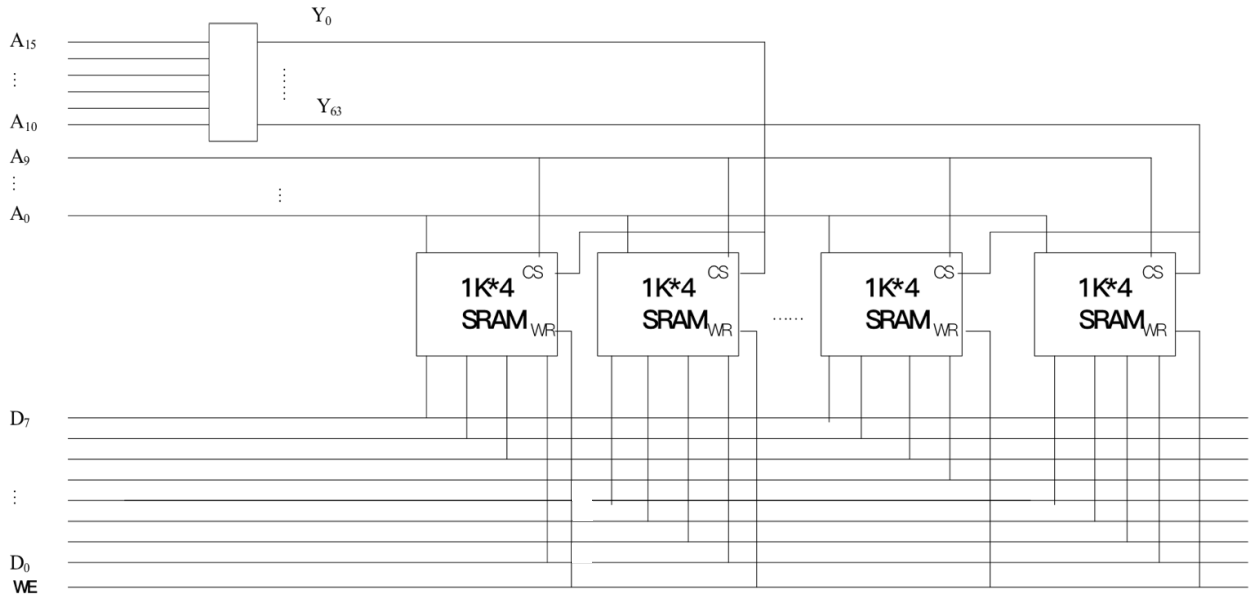
（3）画出该存储器的逻辑框图。

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。



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

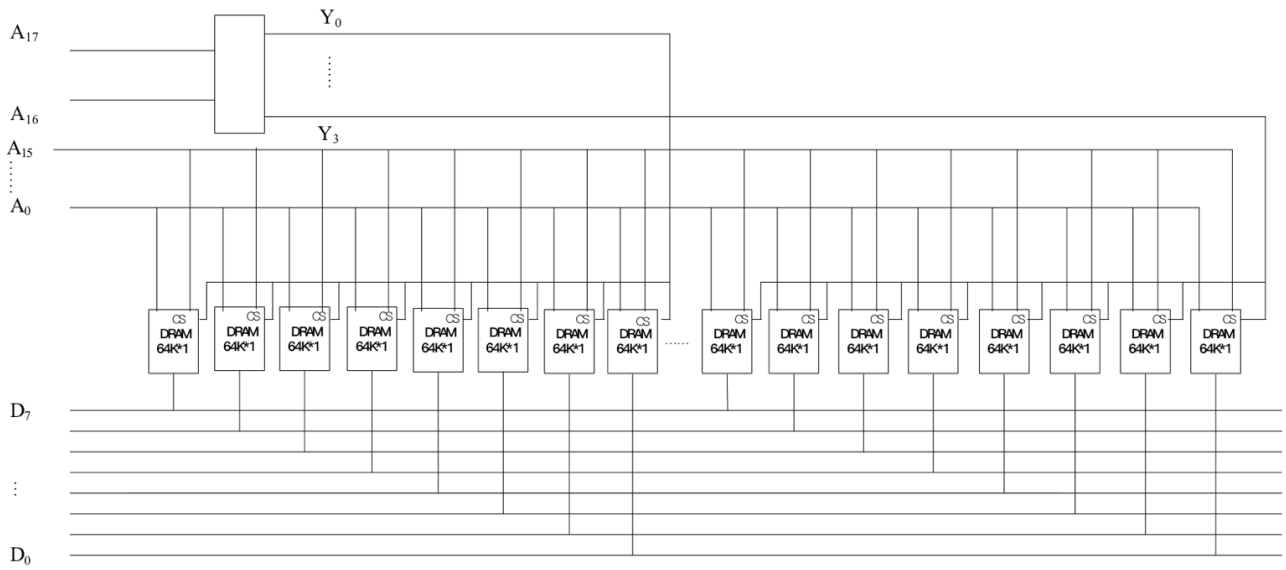
（1） 计算所需芯片数，并画出该存储器的逻辑框图。

（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。

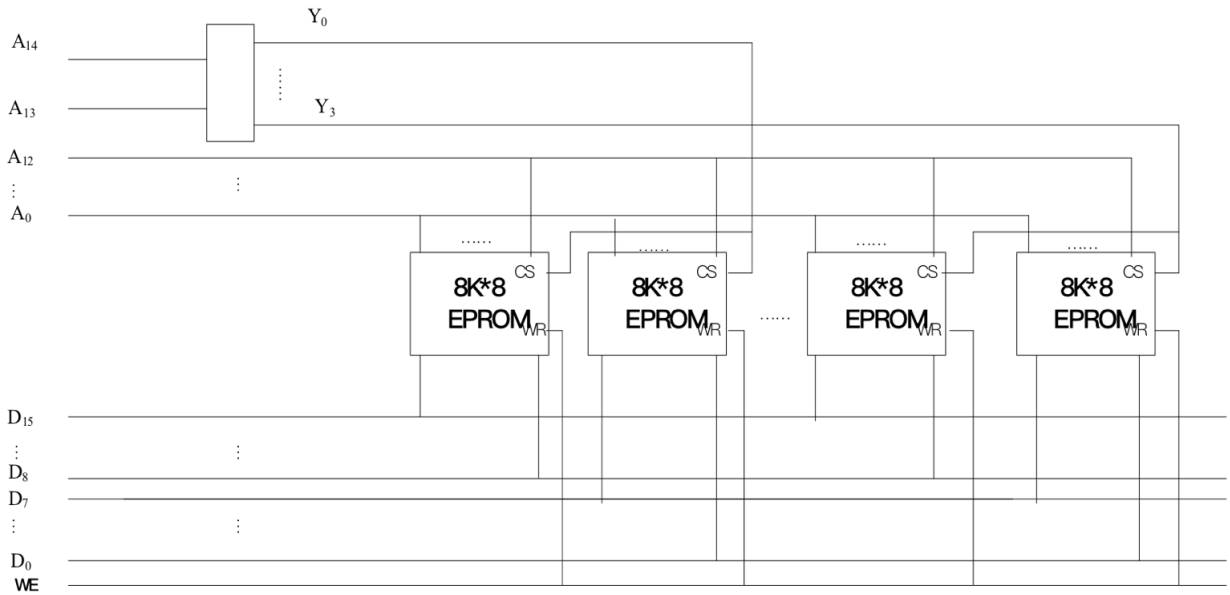
参考答案：

（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）。



6. 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 2^{15}=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 A_{15} 来区分，当 A_{15} 为 0 时选中 ROM 芯片；为 1 时选中 RAM 芯片，此时，根据 A_{14} 和 A_{13} 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=2^{14}KB=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。

7. 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。

8. 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。

9. 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。

10. 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。

11. 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!

12. 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。

13. 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 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;
            }
      }
}
```

```
# 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;
      }
}
```

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。

14. 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%。

15. 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。

16. 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;
    }
}
```

程序段A

```
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;
    }
}
```

程序段B

```
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：

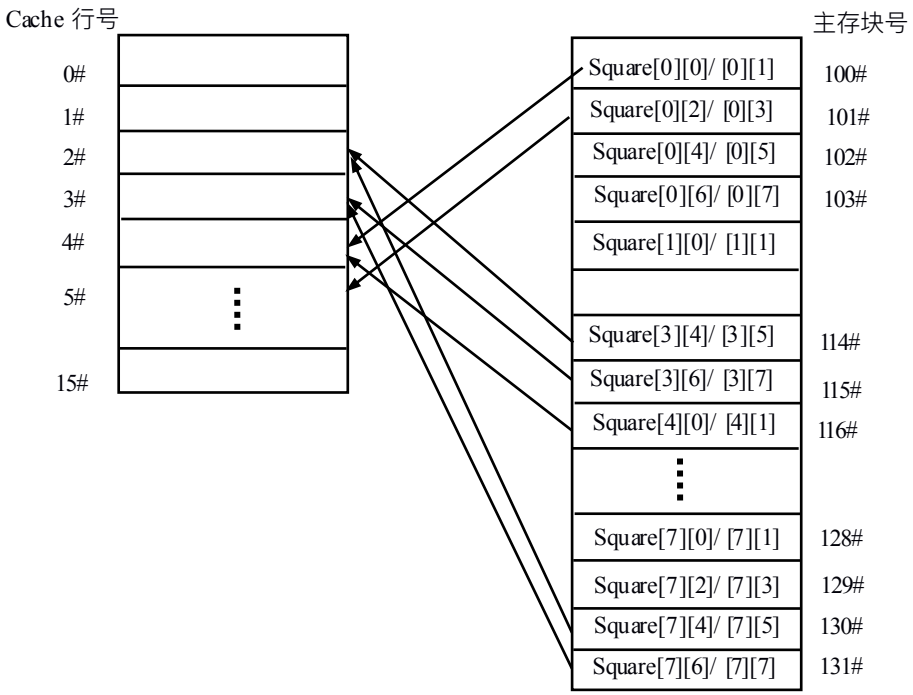
程序段 A 访问顺序和存放顺序一致，所以，空间局部性好；

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：



（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%。

17. 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？

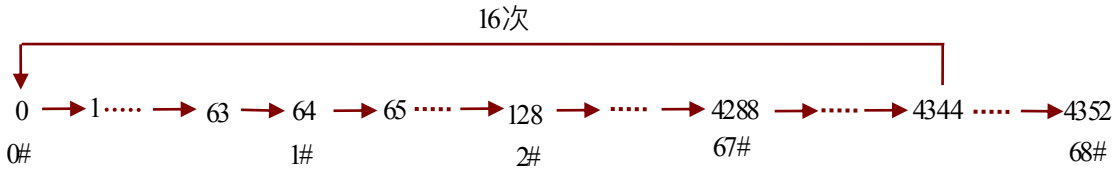
参考答案：

（1）cache 的划分为：$4KB = 2^{12}B = 2^4$ 组×2^2 行/组×2^6 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：$64MB = 2^{26}B = 2^{16}$ 组群×2^4 块/组群×2^6 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。



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

	第0行	第1行	第2行	第3行
0组	0/64/48	16/0/64	32/16	48/32
1组	1/65/49	17/1/65	33/17	49/33
2组	2/66/50	18/2/66	34/18	50/34
3组	3/67/51	19/3/67	35/19	51/35
4组	4	20	36	52
…	…	…	…	…
…	…	…	…	…
15组	15	31	47	63

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

18. 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。

19. 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。

20. 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%！

21. 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=1008 N 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。

22. 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。

23. 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 2^{40}B/16KB=$2^{(40-14)}$=2^{26} 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：2^{26}×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。

24. 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

(a) TLB（四路组相联）：四组、16 个页表项

虚页号	页框号	有效位
00	08	1
01	03	1
02	14	1
03	02	1
04	–	0

行索引	标记	有效位	字节 3	字节 2	字节 1	字节 0
0	19	1	12	56	C9	AC
1	15	0	–	–	–	–
2	1B	1	03	45	12	CD
3	36	0	–	–	–	–
4	32	1	23	34	C2	2A

05	16	1
06	–	0
07	07	1
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	19	1
0D	–	0
0E	11	1
0F	0D	1

(b) 部分页表：（开始 16 项）

5	0D	1	46	67	23	3D
6	–	0	–	–	–	–
7	16	1	12	54	65	DC
8	24	1	23	62	12	3A
9	2D	0	–	–	–	–
A	2D	1	43	62	23	C3
B	–	0	–	–	–	–
C	12	1	76	83	21	35
D	16	1	A3	F4	23	11
E	33	1	2D	4A	45	55
F	14	0	–	–	–	–

(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。

25. 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。

第 5 章　习　题 答 案

3. 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。）

4. 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。

5. 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:

指令名称	汇编格式	指令格式						功能
add	add rd,rs,rt	000000	rs	rt	rd	00000	100000	GPR[rd] ← GPR[rs] + GPR[rt]
addu	addu rd,rs,rt	000000	rs	rt	rd	00000	100001	GPR[rd] ← GPR[rs] + GPR[rt]
addi	addi rt,rs,immediate	001000	rs	rt	immediate			GPR[rt] ← GPR[rs] + immediate
addiu	addiu rt,rs,immediate	001001	rs	rt	immediate			GPR[rt] ← GPR[rs] + immediate

Intel 80x86:

指令名称	Opcode	Instruction	Description
ADC	14ib	ADC AL, imm8	Add with carry imm8 to AL
	15iw	ADC AX, imm16	Add with carry imm16 to AX
	15id	ADC EAX, imm32	Add with carry imm32 to EAX
	10 /r	ADC r/m8, r8	Add with carry byte register to r/m8
	11 /r	ADC r/m16, r16	Add with carry r16 to r/m16
	11 /r	ADC r/m32, r32	Add with CF r32 to r/m32
	12 /r	ADC r8, r/m8	Add with carry r/m8 to byte register
	13 /r	ADC r16, r/m16	Add with carry r/m16 to r16
	13 /r	ADC r32, r/m32	Add with CF r/m32 to r32
ADD	05id	ADD EAX, imm32	Add imm32 to EAX.
	01 /r	ADD r/m32, r32	Add r32 to r/m32
	03 /r	ADD r32, r/m32	Add r/m32 to r32

6. 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 $((16-k2)\times 2^6-k1)\times 2^6=k0$，所以 $k1=(16-k2)\times 2^6-k0/2^6$

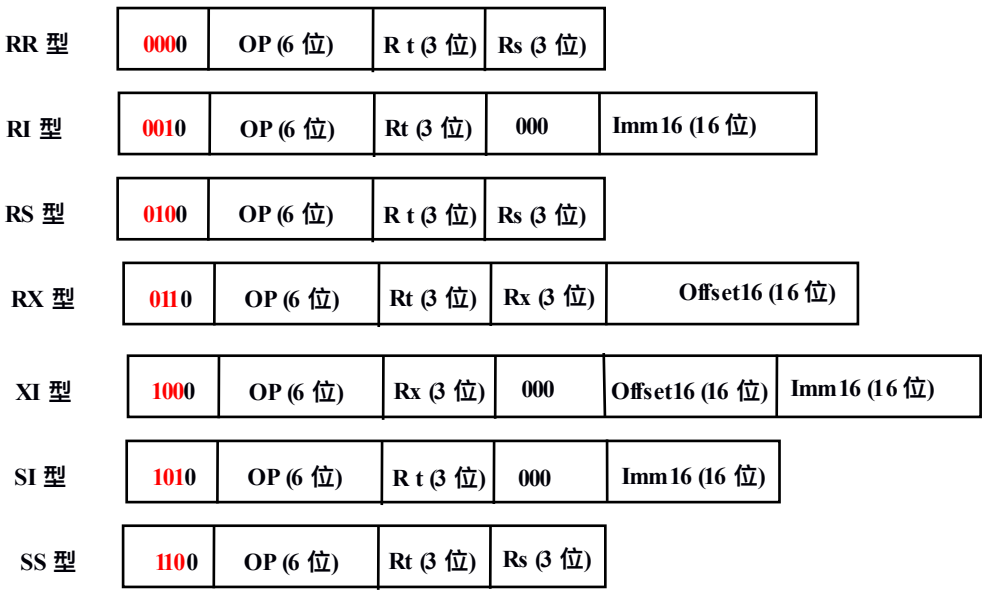
7. 某计算机字长 16 位，每次存储器访问宽度 16 位，CPU 中有 8 个 16 位通用寄存器。现为该机设计指令系统，要求指令长度为字长的整数倍，至多支持 64 种不同操作，每个操作数都支持 4 种寻址方式：立即（I）、寄存器直接（R）、寄存器间接（S）和变址（X），存储器地址位数和立即数均为 16 位，任何一个通用寄存器都可作变址寄存器，支持以下 7 种二地址指令格式（R、I、S、X 代表上述四种寻址方式）：RR 型、RI 型、RS 型、RX 型、XI 型、SI 型、SS 型。请设计该指令系统的 7 种指令格式，给出每种格式的指令长度、各字段所占位数和含义，并说明每种格式指令需要几次存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

参考答案：

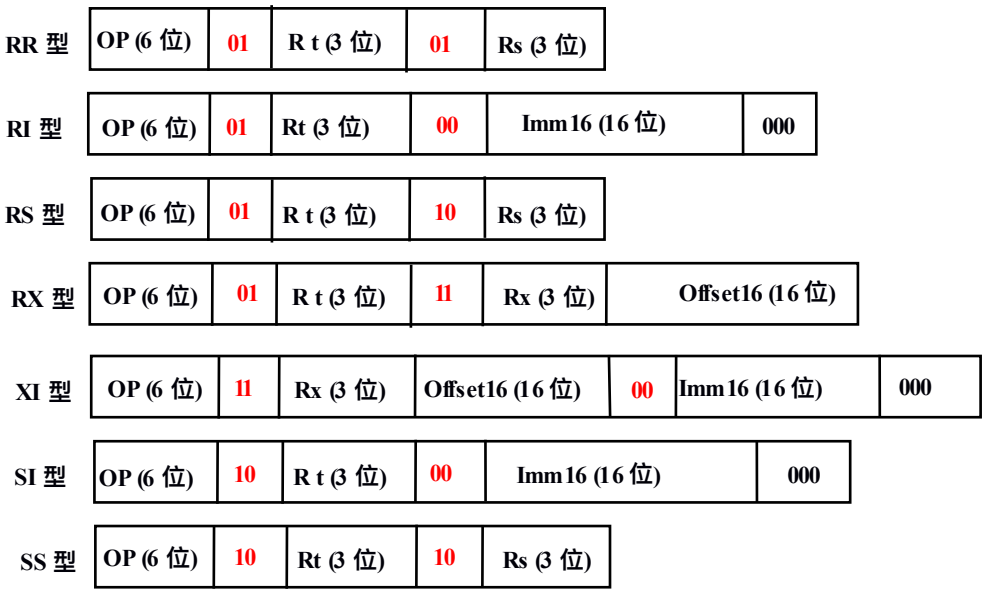
指令格式可以有很多种，只要满足以下的要求即可。

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:

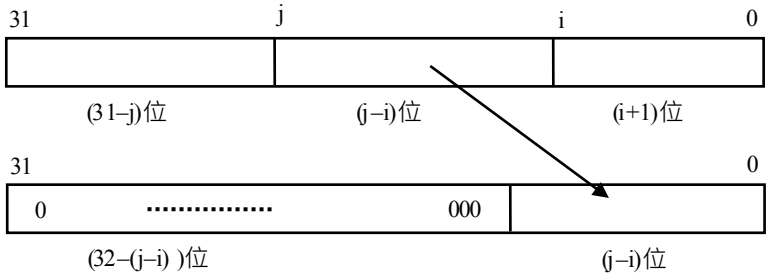


指令格式例 2:



寻址方式字段(2 位)---00：立即；01：寄直；10：寄间；11-变址

8. 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。



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!

9. 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”

10. 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

11. 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
```

12. 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:
```

13. 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（2^{17}=4+(2^{15}–1)×4）

负跳范围为：1000 0000 0000 0000（4–2^{17}=4+(–2^{15})×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
```

14. 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
```

15. 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
```

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。

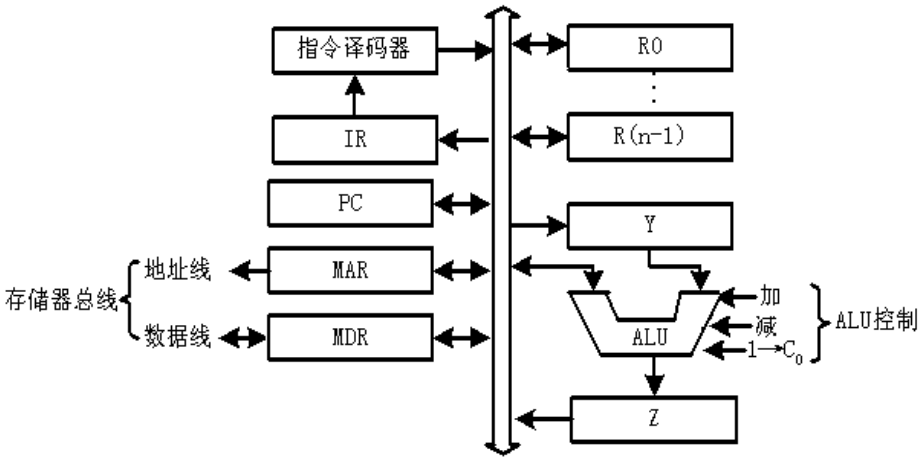
(1) CPU的基本组成和基本功能各是什么?
(2) 取指令部件的功能是什么?
(3) 控制器的功能是什么?
(4) 为什么对存储器按异步方式进行读写时需要WMFC信号？按同步方式访问存储器时，CPU如何实现存储器读写?
(5) 单周期处理器的CPI是多少？时钟周期如何确定？为什么单周期处理器的性能差？元件在一个指令周期内能否被重复使用？为什么?
(6) 多周期处理器的设计思想是什么？每条指令的CPI是否相同？为什么在一个指令周期内某个元件可被重复使用?

（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？

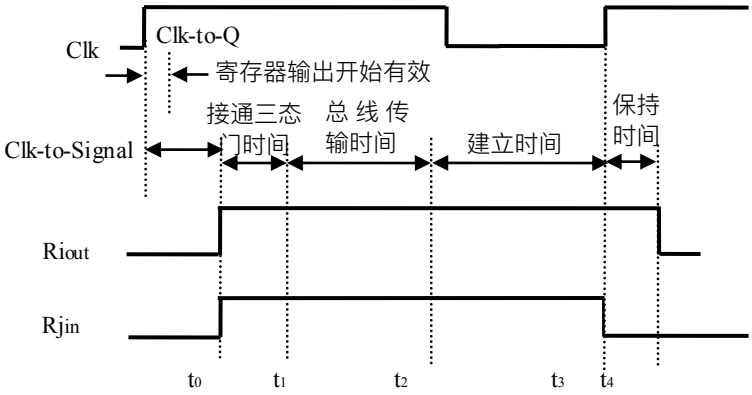
3. 在书中图6.9中，假定总线传输延迟和ALU运算时间分别是20ps和200ps，寄存器建立时间为10ps，寄存器保持时间为5ps，寄存器的锁存延迟（Clk-to-Q time）为4ps，控制信号的生成延迟（Clk-to-signal time）为7ps，三态门接通时间为3ps，则从当前时钟到达开始算起，完成以下操作的最短时间是多少？各需要几个时钟周期？

（1）将数据从一个寄存器传送到另一个寄存器

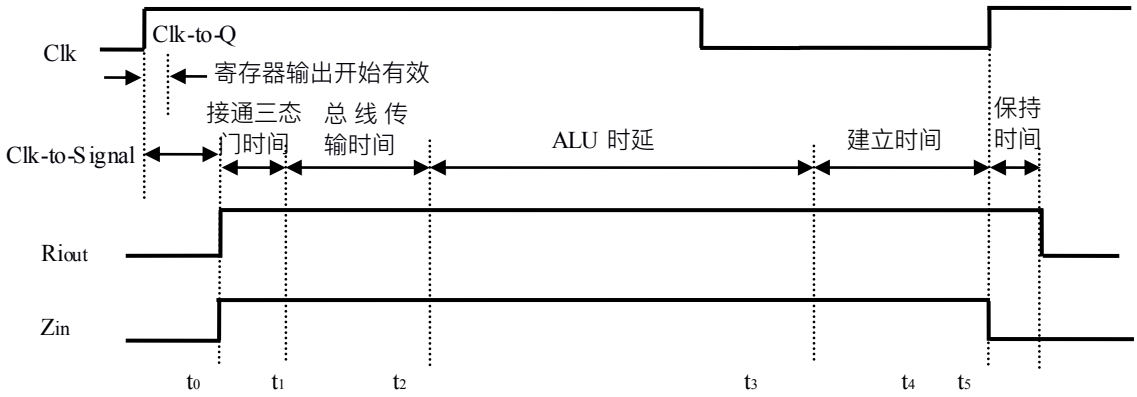
（2）将程序计数器PC加1



所示。



(a) 当前周期内不执行 ALU 运算



(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）寄存器的锁存延迟与控制信号的生成延迟的时间重叠，

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6.30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）。

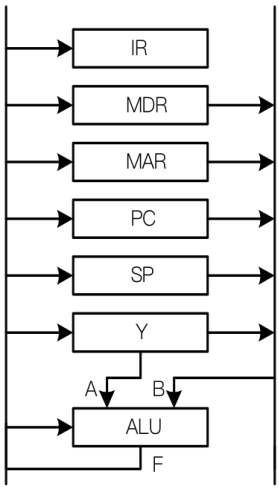


图30

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
   MDRout, MOVb, IRin
2. 取子程序首址：PCout, MOVb, MARin
   Read, b+1, Yin (返回地址在Y中)
   MDRout, MOVb, **PCin**(子程序首址在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6.9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]

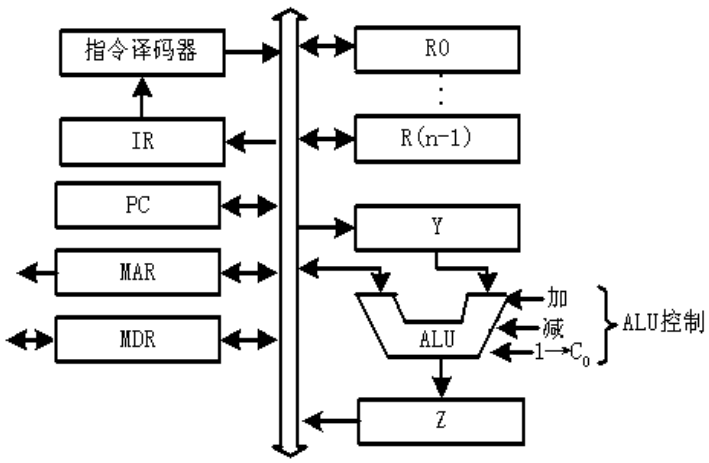
（3）将存储单元Imm16的内容作为地址所指的存储单元的内容加到寄存器R1中。此时，Imm16为间接地址。即：R[R1]←R[R1]+ M[M[Imm16]]

参考答案：

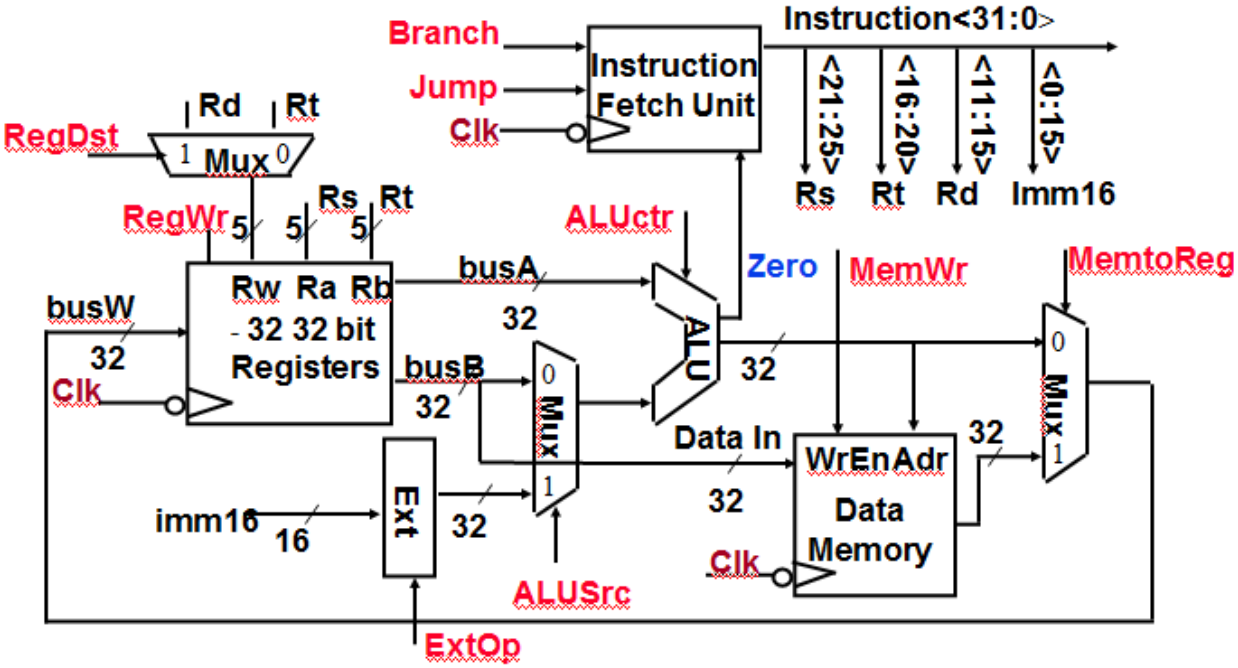
（1） MDRout, Yin
Rlout, add, Zin
Zout, Rlin
需3个时钟周期

（2） MDRout, MARin
Read1,（Rlout, Yin也可以放在该控制信号所在的时钟周期中）
Read2, Rlout, Yin
MDRout, add, Zin
Zout, Rlin
需5个时钟周期

（3） MDRout, MARin
Read1
Read2
MDRout, MARin
Read1,（Rlout, Yin）
Read2, Rlout, Yin
MDRout, add, Zin
Zout, Rlin
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：

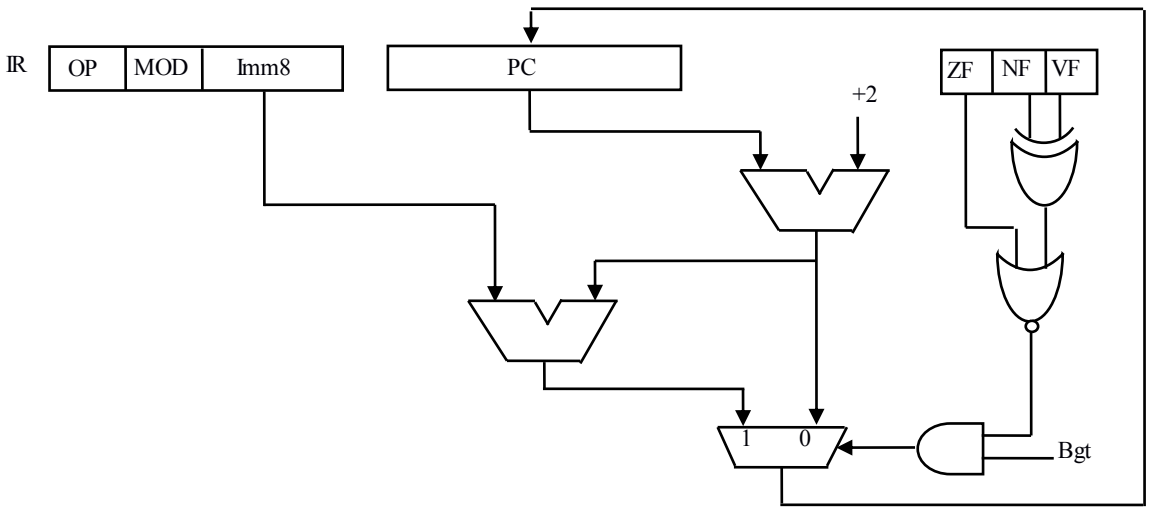
（1）该计算机的编址单位是字节。

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。



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。

习题

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 10^6=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 10^6=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。

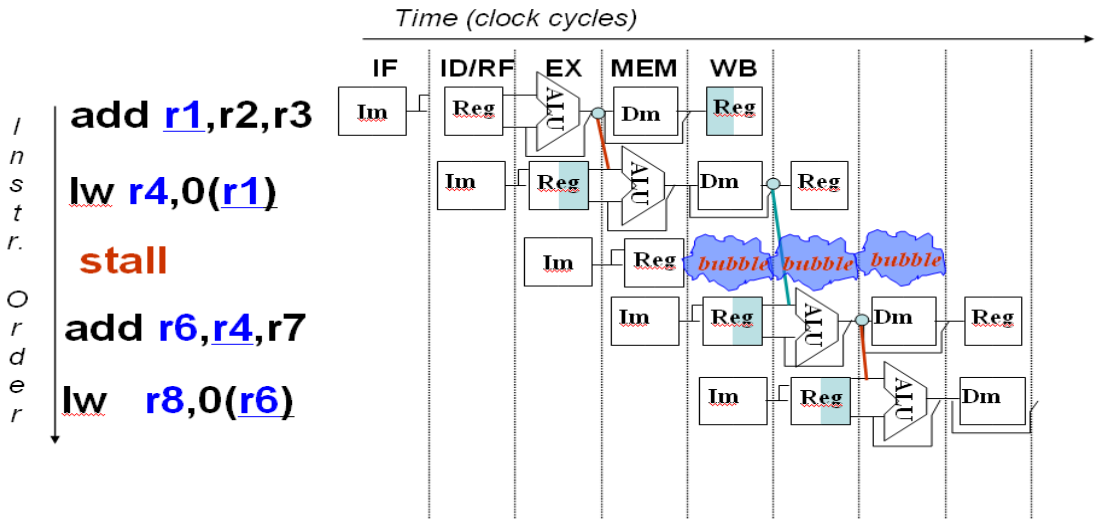
指令 5 在“IF”阶段，指令正被读出。结束时已送到流水段寄存器 IF/ID 的输入端。因为之前发生了 load-use 数据冒险，所以该指令将在随后的第 6 个时钟周期内重新被读出。

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

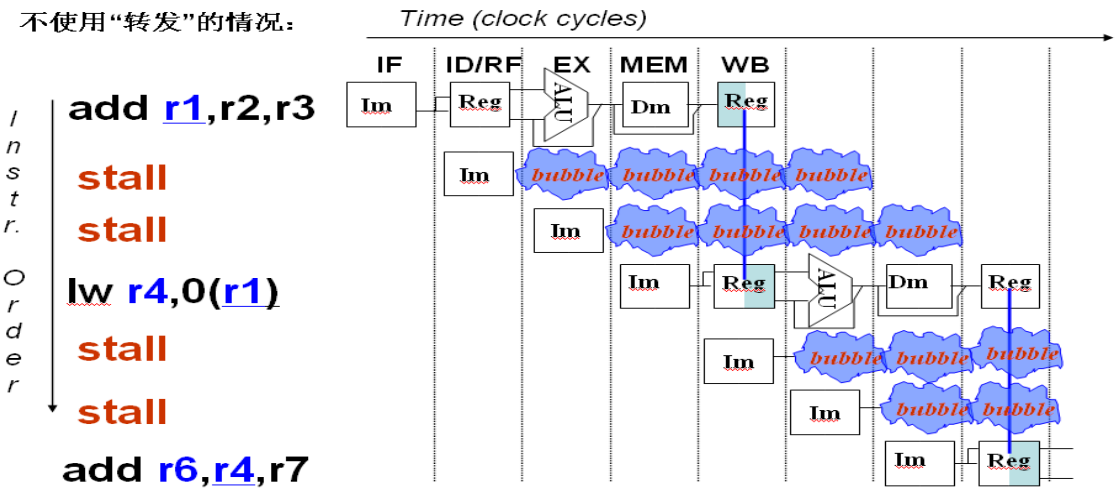
使用“转发”的情况：



使用“转发”时，只有lw指令后需要一次阻塞！

（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

不使用“转发”的情况：



通过寄存器写口/读口分别安排在前半/后半周期，在不使用“转发”时使得每条指令之间只要阻塞两次就可解决！

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

1. 给出以下概念的解释说明。

总线（Bus）
系统总线（System Bus）
并行总线（Parallel Bus）
片内总线（Internal Bus）
通信总线（Communication Bus）
串行总线（Serial Bus）

底板总线（Backplane Bus）　　I/O 总线（I/O Bus）
处理器—存储器总线（Processor-memory Bus）
处理器总线（Processor Bus）　　存储器总线（memory Bus）
信号线复用（Signal Lines Multiplexing）
最大数据传输率（Maximum Data Transfer Rate）
总线传输周期（Bus Transmission Cycle）
总线宽度（Bus Width）　　总线时钟频率（Bus Clock Frequency）
总线带宽（Bus Bandwidth）　　总线事务（Bus Transaction）
主设备（Master，Initiator ）　　从设备（Slave，Target）
突发传送（Burst Transmission）　　背对背（Back-to-Back）传送
总线裁决（Bus Arbitration）　　总线控制器（Bus Arbiter）
总线请求信号（Bus Request）　　总线允许信号（Bus Grant）
集中裁决（Centralized Arbitration）　分布裁决（Distributed Arbitration）
同步总线（Synchronous Bus）　　异步总线（Asynchronous Bus）
握手信号（Handshaking Signal）　　半同步总线（Semi-Synchronous Bus）
分离事务协议（Split transaction protocol）
ISA 总线（Industrial Standard Architecture Bus）
EISA 总线（Extended Industrial Standard Architecture Bus）
PCI 总线（Peripheral Component Interconnect Bus）
SCSI 总线（Small Computer Standard Interface Bus）

2. 简单回答下列问题。

（1）什么情况下需要总线仲裁？总线仲裁的目的是什么？有哪几种常用的仲裁方式？各有什么特点？

（2）总线通信采用的定时方式有哪几种？各有什么优缺点？

（3）在异步通信中，握手信号的作用是什么？常见的握手协议有哪几种？各有何特点？

（4）什么叫非突发传送和突发传送？

（5）提高同步总线的带宽有哪几种措施？

（6）制定总线标准的好处是什么？总线标准是如何制定出来的？

3. 假设一个同步总线的时钟频率为 50MHz，总线宽度为 32 位，该总线的最大数据传输率为多少？若要将该总线的带宽提高一倍，可以有哪几种方案？

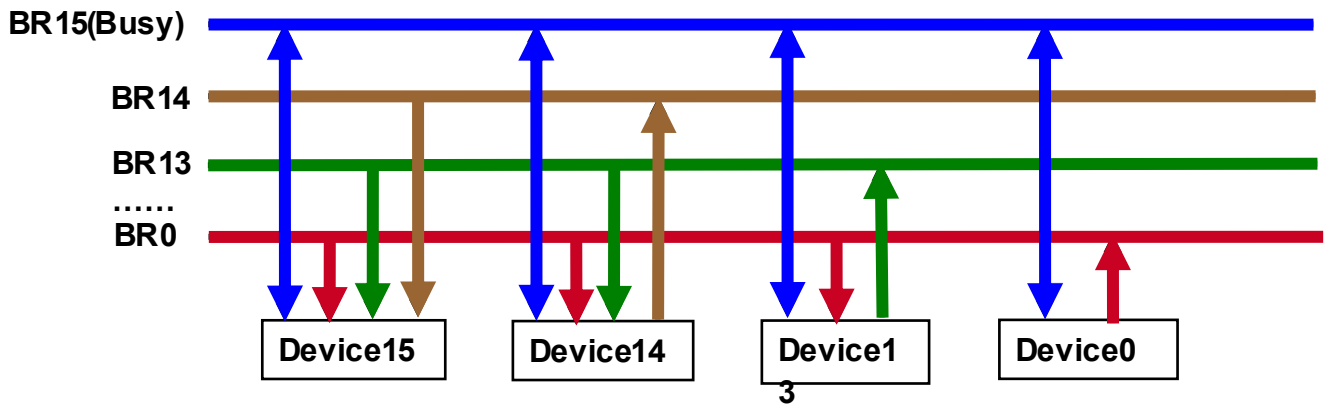
参考答案：

最大数据传输率为：4B×50M=200MB/s

方案一：将时钟频率提高一倍；方案二：将总线宽度提高一倍。

4. VAX SBI 总线采用分布式的自举裁决方案，总线上每个设备有惟一的优先级，而且被分配一根独立的总线请求线 REQ，SBI 有 16 根这样的请求线（REQ0,…REQ15），其中 REQ0 优先级最高，请问：最

多可有多少个设备连到这样的总线上？为什么？



参考答案：

最多可连接 16 个设备，设有 16 个设备（DEV0,…DEV15）,其优先级依次降低，将 REQ15 作为总线忙信号线。DEV0 在总线空闲（REQ15 没有请求信号）时可直接使用总线；DEV1 在总线空闲时且 REQ0 没有请求信号时使用总线；依次类推，DEV15 在总线空闲时且 REQ0 至 REQ14 都没有请求信号时使用总线。这样最多可以有 16 个设备无冲突的使用总线。

5. 假定一个 32 位微处理器的外部处理器总线的宽度为 16 位，总线时钟频率为 40MHz，假定一个总线事务的最短周期是 4 个总线时钟周期，该处理器的最大数据传输率是多少？如果将外部总线的数据线宽度扩展为 32 位，那么该处理器的最大数据传输率提高到多少？这种措施与加倍外部总线时钟频率的措施相比，哪种更好？

参考答案：

一次总线事务至少为 4×1/40M(秒)，只能传送 16 位数据，故处理器最大数据传输率为：

2B/(4×1/40M) = 20MB/秒。若采用 32 位总线宽度，则可提高到 4B/(4×1/40M) = 40MB/s.

若倍频，也可提高到 2B/(4×1/80M)=40MB/s. 两者效果相同。

6. 试设计一个采用固定优先级的具有 4 个输入的集中式独立请求裁决器。

参考答案：

设计一个并行判优电路即可。

如 P0-P3 为四条总线请求线，优先由高到低。

G0-G3 为四条总线允许线，则：

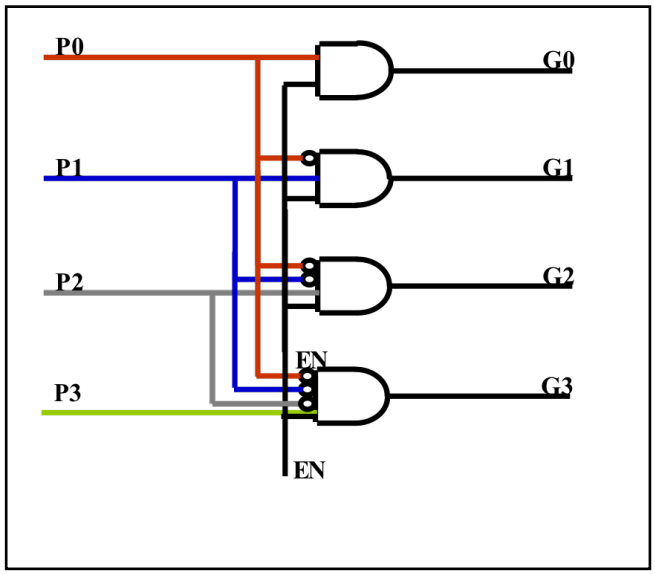
G0=P0；

G1=（P 1）&（~ P 0）；

G2=（P 2）&（~ P 1）&（~ P 0）；

G3=（P 3）&（~ P 2）&（~ P 1）&（~ P 0）

EN 信号可有可无



7. 假设某存储器总线采用同步通信方式，时钟频率为 50MHz 时钟，每个总线事务以突发方式传输 8 个字，以支持块长为 8 个字的 Cache 行读和 Cache 行写，每字 4 字节。对于读操作，访问顺序是 1 个时钟周期接受地址，3 个时钟周期等待存储器读数，8 个时钟周期用于传输 8 个字。对于写操作，访问顺序是 1 个时钟周期接受地址，2 个时钟周期延迟，8 个时钟周期用于传输 8 个字，3 个时钟周期恢复和写入纠错码。对于以下访问模式，求出该存储器读/写时在存储器总线上的带宽。

① 全部访问为连续的读操作；

② 全部访问为连续的写操作；

③ 65%的访问为读操作，35%的访问为写操作。

参考答案：

① 8 个字用 1+3+8=12 个周期，故 8×4B/(12×1/50M) = 133 MB/s.

② 8 个字用 1+2+8+3=14 个周期，故 8×4B/(14×1/50M) = 114 MB/s.

③ 故. 133×65% + 114×35% = 126.0MB/s.

用另外一种计算方式结果差不多：8x4B/((12x65%+14x35%)x 1/50M) = 126 MB/s

8. 考虑以下两种总线：

总线 1 是 64 位数据和地址复用的总线。能在一个时钟周期中传输一个 64 位的数据或地址。任何一个读写操作总是先用一个时钟周期传送地址，然后有 2 个时钟周期的延迟，从第四时钟周期开始，存储器系统以每个时钟 2 个字的速度传送，最多传送 8 个字。

总线 2 是分离的 32 位地址和 32 位数据的总线。读操作包括：一个时钟周期传送地址，2 个时钟周期延迟，从第四周期开始，存储器系统以每时钟 1 个字的速度传输最多 8 个字。对于写操作，在第一个时钟周期内第一个数据字与地址一起传输，经过 2 个时钟周期的延迟后，以每个时钟 1 个字的速度最多传输 7 个余下的数据字。假定进行 60%的读操作和 40%的写操作。

在以下两种情况下，求这两种总线和存储器能提供的带宽。

① 只进行单数据字的传输。

② 所有的传输都是 8 个字的数据块。

参考答案：

设时钟周期为 T，一个字为 32 位，64 位则为 2 个字。

总线 1：地址/数据复用。所以，读和写操作所花时间都一样。

总线 2：地址和数据分离。所以，读和写操作所花时间不一样。

① 单数据字传送的情况

总线 1：虽然每个时钟周期可传 2 个字，但只需传一个字，所花时间为 4T。每个时钟周期只传送一个字。因此带宽为 4B/4T=1 B/T。

总线 2：读一字时间为：3+1=4T；写一字时间为：3T。因此带宽为：4B/4T×60%+4B/3T×40%=1.1 B/T。 （比总线 1 快）

② 8 个字的数据块传送情况

总线 1：对于传送 8 个字的数据块，所花时间为 4T+3T。也即读或写 8 个字所花时间都为 7T。因此带宽为：8×4B/7T=(32/7) B/T。

总线 2：读 8 个字时间为：3+8=11T；写 8 个字时间为：3+7=10T。因此带宽为：

8×4B/11T×60%+8×4B/10T×40%=(32/10.6) B/T。

（比总线 1 慢）

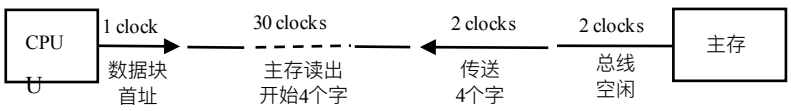
另外：地址线宽度不等，寻址空间大小也不同

9. 假定主存和 CPU 之间连接的同步总线具有以下特性：支持 4 字块和 16 字块（字长 32 位）两种长度的突发传送，总线时钟频率为 200MHz，总线宽度为 64 位，每个 64 位数据的传送需 1 个时钟周期，向主存发送一个地址需要 1 个时钟周期，每个总线事务之间有 2 个空闲时钟周期。若访问主存时最初四个字的存取时间为 200ns，随后每存取一个四字的时间是 20ns，则在 4 字块和 16 字块两种传输方式下，该总线上传输 256 个字时的数据传输率分别是多少？你能从计算结果中得到什么结论？

参考答案：ppt

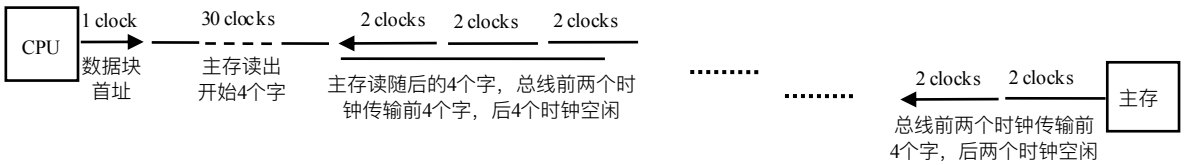
10. 上题所述的系统中，假定访问主存时最初四个字的读取时间为 148ns，随后每读一个四字的时间为 26ns，则在 4 字块和 16 字块两种传输方式下，CPU 从主存读出 256 个字时，该总线上的数据传输率分别是多少？和上题计算结果进行比较分析，并给出相应的结论。

参考答案：因为最初 4 个字的读取时间从 200ns 变为 148ns，所以主存读开始的 4 个字只用了 148ns/5ns=29.6 个时钟周期，当主存存取时间不是总线时钟周期的整数倍时，主存会先准备好数据，等待下个总线时钟周期到来后，开始在总线上传送数据。因此，开始 4 字的读取时间实际上相当于 30 个时钟周期的时间。



从图中可以看出，一次总线事务总共需要 1+30+2+2=35 个时钟周期，256 个字需 256/4=64 个事务，因而整个传送需 35×64=2240 个时钟周期，总线的数据传输率为(256×4B)/(2240×5ns) =91.43MB/s。

因为从第二个 4 字开始，每读一个 4 字的时间为 26ns，相当于 26ns/5ns=5.2 个总线时钟周期的时间。在 16 字传输方式下，每个数据块的传送过程如图所示。



从图中可以看出，一次总线事务总共需要 1+30+3×6+2+2=53 个时钟周期，256 字需 256/16=16 个事务，因而整个传送需 53×16=848 个时钟周期，总线的数据传输率为(256×4B)/(848×5ns) =241.51MB/s。

与上一题给出的系统相比，4 字传输方式下，速度提高了(91.43－71.11)/71.11=28.6%；16 字传输方式下，速度只提高了(241.51－224.56)/224.56=7.5%。由此可知，对于小数据块的传输，主存首次读取的速度更重要；而对于大数据块的传输，则首次读取速度和随后的读取速度都很重要。

3. 假定一个政府机构同时监控 100 路移动电话的通话消息，通话消息被分时复用到一个带宽为 4MBps 的网络上，复用使得每传送 1KB 的通话消息需额外开销 150μs，若通话消息的采样频率为 4KHz，每个样本的量化值占 16 位，要求计算每个通话消息的传输时间，并判断该网络带宽能否支持同时监控 100 路

通话消息?

参考答案：

每路移动电话 1 秒钟所要传输的数据量：4000HZ x(16/8)B=8000B=7.8125KB

该网络传输 1KB 数据所需要的时间为：150μs+(1KB / 4MB)=394μs

所以实际传输 100 路移动电话所需时间为：394μs/KB x7.8125KB x 100=0.31s

因为 0.31s 小于 1 秒钟，故该网络带宽支持同时监控 100 路通话消息。

4. 假定一个程序重复完成将磁盘上一个 4KB 的数据块读出，进行相应处理后，写回到磁盘的另外一个数据区。各数据块内信息在磁盘上连续存放，并随机地位于磁盘的一个磁道上。磁盘转速为 7200RPM，平均寻道时间为 10ms，磁盘最大数据传输率为 40MBps，磁盘控制器的开销为 2ms，没有其他程序使用磁盘和处理器，并且磁盘读写操作和磁盘数据的处理时间不重叠。若程序对磁盘数据的处理需要 20000 个时钟周期，处理器时钟频率为 500MHz，则该程序完成一次数据块“读出-处理-写回”操作所需的时间为多少？每秒钟可以完成多少次这样的数据块操作?

参考答案：

平均旋转等待时间：(1s / (7200/60)) / 2 ≈ 8.33/2 ≈ 4.17ms

因为块内信息连续存放，所以数据传输时间：4KB / 40MBps ≈ 0.1ms

平均存取时间 T ：寻道时间 + 旋转等待时间 + 数据传输时间

= 10ms + 4.17ms + 0.1ms = 14.27ms

读出时间(写回时间)：14.27ms+2ms = 16.27ms

数据块的处理时间：20000 / 500MHz ≈ 0.04ms

因为数据块随机存放在某个磁道上，所以，每个数据块的 “读出–处理–写回”操作时间都是相同的，**所以完成一次操作时间：16.27ms x 2+0.04ms = 32.58ms**

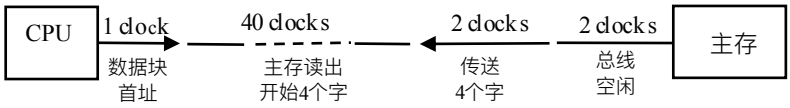
每秒中可以完成这样的数据块操作次数：1s / 32.58ms ≈ 30 次

5. 假定主存和磁盘存储器之间连接的同步总线具有以下特性：支持 4 字块和 16 字块两种长度（字长 32 位）的突发传送，总线时钟频率为 200MHz，总线宽度为 64 位，每个 64 位数据的传送需 1 个时钟周期，向主存发送一个地址需要 1 个时钟周期，每个总线事务之间有 2 个空闲时钟周期。若访问主存时最初四个字的存取时间为 200ns，随后每存取一个四字的时间是 20ns，磁盘的数据传输率为 5MBps，则在 4 字块和 16 字块两种传输方式下，该总线上分别最多可有多少个磁盘同时进行传输?

参考答案：

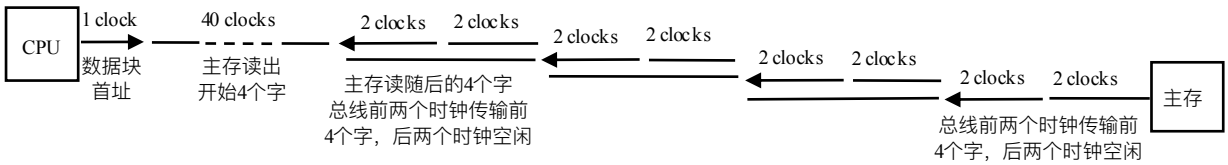
总线时钟频率为 200MHz，因而总线时钟周期为 1/200M=5ns。

对于 4 字传送方式，每个总线事务由一个地址传送后跟一个 4 字的数据块传送组成。



一次总线事务总共需要 1+40+2+2=45 个时钟周期，256 个字需 256/4=64 个事务，因而整个传送需 45×64=2880 个时钟周期，得到总延时为 2880×5ns=14400ns。每秒钟进行的总线事务数为 64/14400ns = 4.44M。总线的数据传输率为(256×4B)/14400ns =71.11MB/s。

对于 16 字传送方式，每个总线事务由一个地址传送后跟一个 16 字的数据块传送组成。



一次总线事务（传输 16 字）的时钟周期数为 1+40+4×(2+2)=57，256 个字需 256/16=16 个事务，因此整个传送需 57×16=912 个时钟周期。因而总延时为 912×5ns=4560ns。每秒钟的总线事务数为 16/4560ns= 3.51M。总线的数据传输率为(256×4B)/4560ns=224.56MB/s。

由以上第 8 章 ppt 中的例题可知，在 4 字传输方式下，总线的数据传输率为 71.11MB/s，因为 **71.11/5=14.2**，所以，该总线上最多可以有 14 个磁盘同时进行传输。在 16 字传输方式下，总线的数据传输率为 224.56MB/s，因为 **224.56/5=44.9**，因此，此时该总线上最多可以有 44 个磁盘同时进行传输。

6. 假定有两个用来存储 10TB 数据的 RAID 系统。系统 A 使用 RAID1 技术，系统 B 使用 RAID5 技术。

(1) 系统 A 需要比系统 B 多用多少存储量?

(2) 假定一个应用需要向磁盘写入一块数据，若磁盘读或写一块数据的时间为 30ms，则最坏情况下，在系统 A 和系统 B 上写入一块数据分别需要多长时间?

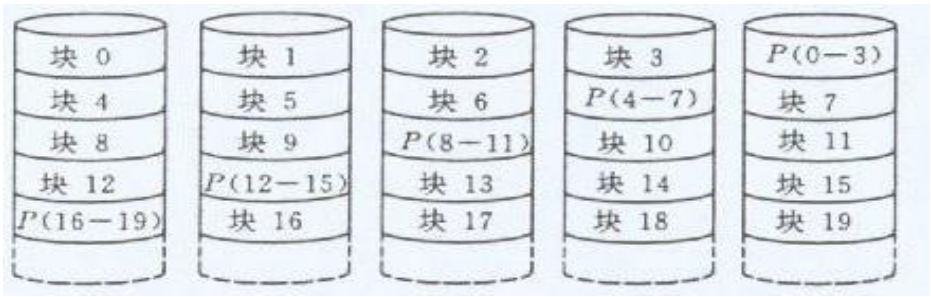
(3) 那个系统更可靠？为什么?

参考答案：

(1) 系统 A 使用 RAID1 技术，所以存储 10TB 数据的情形下要使用 20TB 的磁盘。

系统 B 使用 RAID5 技术，假设是使用 5 个磁盘阵列，那么 10TB 的数据需要 2.5TB 的磁盘来存放冗余的奇偶校验数据，所以系统 A 要比系统 B 多用 7.5TB 存储量。

（2）系统 A 的写入速度取决于原磁盘和备份磁盘中速度慢的一块，但两个磁盘并行写。因为写一块数据的时间都是 30ms，故系统 A 写入一块数据的时间是 30ms。系统 B 在写入一块数据后可能要更改相关的校验数据，冗余数据分布在不同磁盘上，所以最坏的情况下，写一块数据的时间为 2 次读和 2 次写，即所用时间为 4×30=120ms 。。



假定考虑一个有5个磁盘的阵列，且假设要写入的数据在磁盘X_0上的块0，则与之相关的数据为X_1到X_3上的块1-3，以及X_4中的奇偶校验数据P(0-3)，

可知： $p(i)= X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X_0(i)$

写操作后，可能改变的情况如下：

$$P'(i) = X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X'_0(i)$$

$$= X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X'_0(i) \oplus X_0(i) \oplus X_0(i)$$

化简后得到： $p'(i) = p(i) \oplus X_0(i) \oplus X'_0(i)$

因此，要更新一个$X_0(i)$，必须先读$p(i)$和$X_0(i)$，然后写$X'_0(i)$和$p'(i)$

(3) 相对来说系统 A 更可靠一些，因为系统对整个磁盘进行了完整备份，所以只有互为镜像的两个盘上的对应数据都损坏时才不能恢复；而系统 B 是分散记录了原数据的部分冗余信息，如果其中两个磁盘的相同位都损坏了就恢复不出来了。

7. 假定在一个使用 RAID5 的系统中，采用先更新数据块、再更新校验块的信息更新方式。如果在更新数据块和更新校验块的操作之间发生了掉电现象，那么会出现什么问题？采用什么样的信息更新方式可避免这个问题？

参考答案：

答：对于 RAID 5 来说，如果在写完数据块但未写入校验块时发生断电，则写入的数据和对应的校验信息不匹配，无法正确恢复数据。这种情况可以避免，因为 RAID 5 是大数据块交叉方式，每个盘独立进行操作，所以，只要同时写数据块所在盘和校验块所在盘即可。

8. 某终端通过 RS-232 串行通信接口与主机相连，采用起止式异步通信方式，若传输速率为 1200 波特，采用两相调制技术。通信协议为 8 位数据、无校验位、停止位为 1 位。则传送一个字节所需时间约为多少？若传输速度为 2400 波特，停止位为 2 位，其他不变，则传输一个字节的时间为多少？

参考答案：

采用两相调制技术，所以，波特率=比特率，且每个字符都有一个起始位，

(a) 1200 波特时，一个字符共占：1+8+1=10 位

所以一个字符所需时间约为：10x(1÷1200)=8.3 毫秒

(b) 2400 波特时，一个字符共占：1+8+2=11 位

所以一个字符所需时间约为：11x(1/2400)=4.6 毫秒

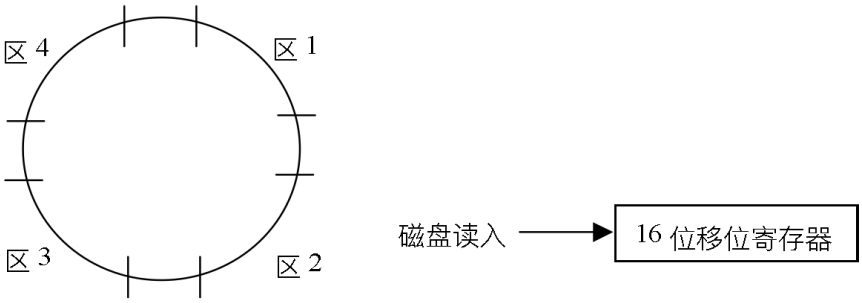
9. 假定采用独立编址方式对 I/O 端口进行编号，那么，必须为处理器设计哪些指令来专门用于进行 I/O 端口的访问？连接处理器的总线必须提供哪些控制信号来表明访问的是 I/O 空间？

参考答案：

IO 读指令和 IO 写指令

IO 读控制信号，IO 写控制信号。

10. 假设有一个磁盘，每面有 200 个磁道，盘面总存储容量为 1.6 兆字节，磁盘旋转时间为 25ms/圈，每道有 4 个区，每两个区之间有一个间隙，磁头通过每个间隙需 1.25ms。(1) 问：从该磁盘上读取数据时的最大数据传输率是多少（单位为字节/秒）？ (2) 假如有人为该磁盘设计了一个与计算机之间的接口，如下图所示，磁盘每读出一位，串行送入一个移位寄存器，每当移满 16 位后向处理器发出一个请求交换数据的信号。在处理器响应该请求信号并读取移位寄存器内容的同时，磁盘继续读出一位一位数据并串行送入移位寄存器，如此继续工作。已知处理器在接到请求交换的信号以后，最长响应时间是 3 微秒，这样设计的接口能否正确工作？若不能则应如何改进？



参考答案： 每个磁道的存储容量：$1.6\times10^6 / 200=8000B$

→ 每个区容量为：$8000 / 4 = 2000B$

而当仅读取一个区内数据的时候，转过一个区只需要：$(25-1.25\times4) / 4 = 5ms$

所以最大数据传输率：$2000B / 5ms = 4\times10^5$ 字节/秒

————————————————

因此，传送 1 位的最短时间为：$1 / (8\times4\times10^5) = 0.31\mu s << 3\mu s$

因此，当处理器经过 3μs 来读取移位寄存器中的数据时，磁盘已经读出了新的数据位，并将原先请求被读的移位寄存器中的数据冲刷掉了。所以这样的设计接口不能正确工作。

改进方法：传送 16 位数据需 $0.31 \times 16 = 5\mu s > 3\mu s$

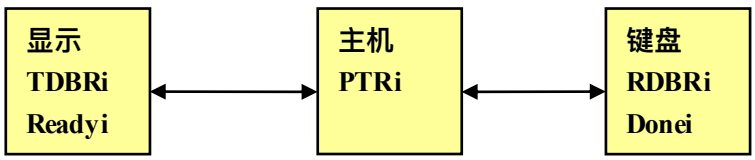
所以可以增加一个 16 位数据缓冲器。当 16 位移位寄存器装满后，先送入数据缓冲寄存器，在读出下一个 16 位数据期间（5μs），上次读出的 16 位数据从数缓器中被取走（3μs）。

11. 假设某计算机带有 20 个终端同时工作，在运行用户程序的同时，能接受来自任意一个终端输入的字符信息，并将字符回送显示（或打印）。每一个终端的键盘输入部分有一个数码缓冲寄存器 RDBRi(i=1~20)，当在键盘上按下某一个键时，相应的字符代码即进入 RDBRi，并使它的“完成”状态标志 Donei (i=1~20)置 1，要等处理器把该字符代码取走后， Donei 标志才置 0。每个终端显示（或打印）输出部分也有一个数码缓冲寄存器 TDBRi(i=1~20)，并有一个 Readyi (i=1~20)状态标志，该状态标志为 1 时，表示相应的 TDBRi 是空着的，准备接收新的输出字符代码，当 TDBRi 接收了一个字符代码后， Readyi 标志才置 0，并送到终端显示（或打印），为了接收终端的输入信息，处理器为每个终端设计了一个指针 PTRi (i=1~20)指向为该终端保留的主存输入缓冲区。处理器采用下列两种方案输入键盘代码，同时回送显示（或打印）。

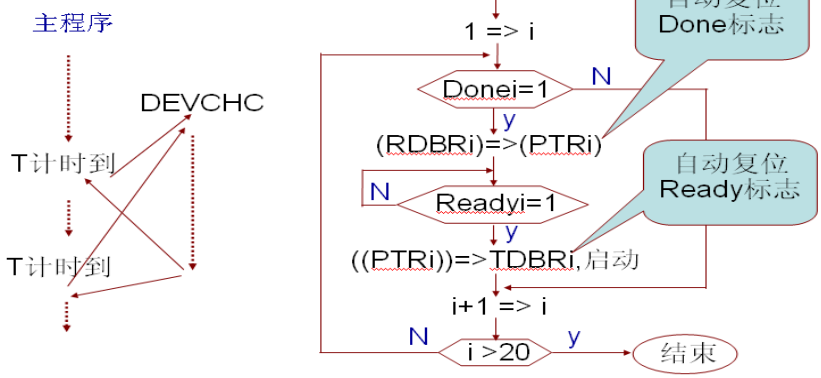
（1）每隔一固定时间 T 转入一个状态检查程序 DEVCHC，顺序地检查全部终端是否有任何键盘信息要输入，如果有，则顺序完成之。

（2）允许任何有键盘信息输入的终端向处理器发出中断请求。全部终端采用共同的向量地址，利用它使处理器在响应中断后，转入一个中断服务程序 DEVINT，由后者询问各终端状态标志，并为最先遇到的请求中断的终端服务，然后转向用户程序。

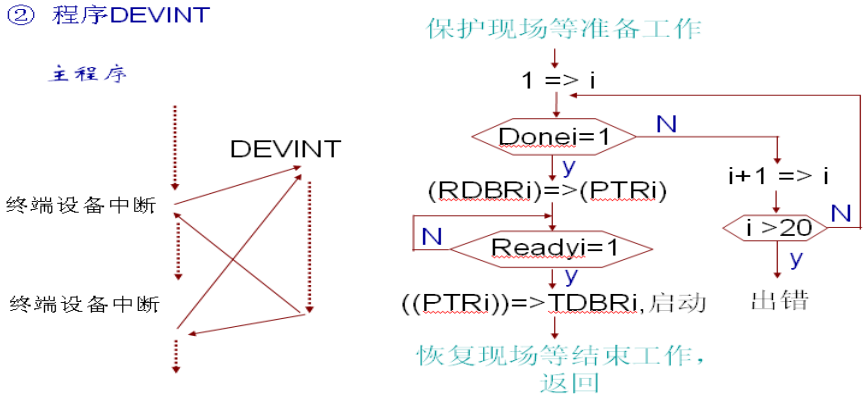
要求画出 DEVCHC 和 DEVINT 两个程序的流程图。



① 程序DEVCHC



② 程序DEVINT



12. 假定某计算机的 CPU 主频为 500MHz，所连接的某个外设的最大数据传输率为 20KBps，该外设接口中有一个 16 位的数据缓存器，相应的中断服务程序的执行时间为 500 个时钟周期，则是否可以用中断方式进行该外设的输入输出？假定该外设的最大数据传输率改为 2MBps，则是否可以用中断方式进行该外设的输入输出？

参考答案：

（1）外设最大传输率为 20KBps

每传输完 16 位进行一次中断处理，因此一秒钟内的中断次数为：20K/2=10K 次

中断响应过程就是执行一条隐指令的过程，所用时间相对于中断处理时间（即执行中断服务程序的时间）而言，几乎可以忽略不计，因而整个中断响应并处理的时间大约：**(1s / 500MHz)x500=1μs**

一秒钟内的中断服务程序执行要用去 10K x 1μs = 10ms ≪ 1s

所以可以用中断方式进行该外设的输入输出。

（2）外设最大传输率为 2MBps

每传输完 16 位进行一次中断处理，因此一秒钟内的中断次数为：2M/2=1M 次

一秒钟内的中断服务程序执行要用去 1M x 1μs = 1s！

所以不可以用中断方式进行该外设的输入输出。

（另外一种算法：若外设的最大传输率为 2MBps，则中断请求的时间间隔为 2B/2MBps=1μs。而整个中断响应并处理的时间也是 1μs，中断请求的间隔时间小于中断响应和处理时间，也即中断处理还未结束就会有该外设新的中断到来，所以不可以。）

13. 若某计算机有 5 级中断，中断响应优先级为 1>2>3>4>5，而中断处理优先级为 1>4>5>2>3。要求：

（1）设计各级中断处理程序的中断屏蔽位(假设 1 为屏蔽，0 为开放)；

（2）若在运行主程序时，同时出现第 2、4 级中断请求，而在处理第 2 级中断过程中，又同时出现 1、3、5 级中断请求，试画出此程序运行过程示意图。

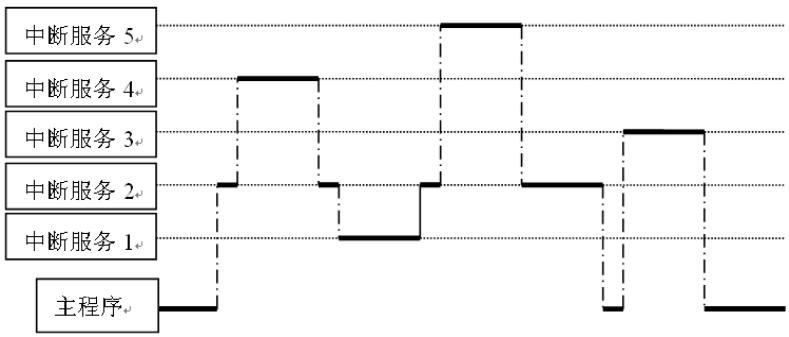
参考答案：

（1）1 级中断的**处理优先级**最高，说明 1 级中断对其他所有中断都屏蔽，其屏蔽字为全 1；3 级中断的处理优先级最低，所以除了 3 级中断本身之外，对其他中断全都开放，其屏蔽字为 00100。以此类推，得到所有中断屏蔽字：

中断程序级别	中断屏蔽字				
	1 级	2 级	3 级	4 级	5 级
第 1 级	1	1	1	1	1
第 2 级	0	1	1	0	0
第 3 级	0	0	1	0	0
第 4 级	0	1	1	1	1
第 5 级	0	1	1	0	1

（2）程序运行过程示意图

在运行用户程序时，同时出现 2、4 级，因为用户程序对所有中断都开放，所以，（1）关中断，在中断响应优先级排队电路中，有 2、4 两级中断进行排队判优，根据**中断响应优先级 2>4**，因此先响应 2 级中断。在 CPU 执行 2 级中断服务程序过程中，首先保护现场、保护旧屏蔽字、设置新的屏蔽字 01100，（2）在中断处理前先开中断。一旦开中断，则马上响应 4 级中断，因为 **2 级中断处理优先级低于 4 级中断**。则响应并处理 4 级中断，结束后才回到 2 级中断服务程序执行；（3）在处理 2 级中断过程中（此时开中断状态），同时发生了 1、3、5 级中断，因为 2 级中断对 1、5 级中断开放，对 3 级中断屏蔽，所以只有 1 和 5 两级中断进行排队判优，根据中断响应优先级 1>5，所以先响应 1 级中断。因为 1 级中断处理优先级最高，所以在其处理过程中不会响应任何新的中断请求，直到 1 级中断处理结束，然后返回 2 级中断；（4）因为 2 级中断对 5 级中断开放，所以在 2 级中服务程序中执行一条指令后，又转去执行 5 级中断服务程序，执行完后回到 2 级中断，（5）在 2 级中断服务程序执行过程中，虽然 3 级中断有请求，但是，因为 2 级中断对 3 级中断不开放，所以，3 级中断一直得不到响应。直到 2 级中断处理完回到用户程序，才能响应并处理 3 级中断。



14. 假定某计算机字长 16 位，没有 cache，运算器一次定点加法时间等于 100 毫微秒，配置的磁盘旋转速度为每分钟 3000 转，每个磁道上记录两个数据块，每一块有 8000 个字节，两个数据块之间间隙的越过时间为 2 毫秒，主存周期为 500 毫微秒，存储器总线宽度为 16 位，总线带宽为 4MBps。

(1) 磁盘读写数据时的最大数据传输率是多少?

(2) 当磁盘按最大数据传输率与主机交换数据时，主存频带空闲百分比是多少?

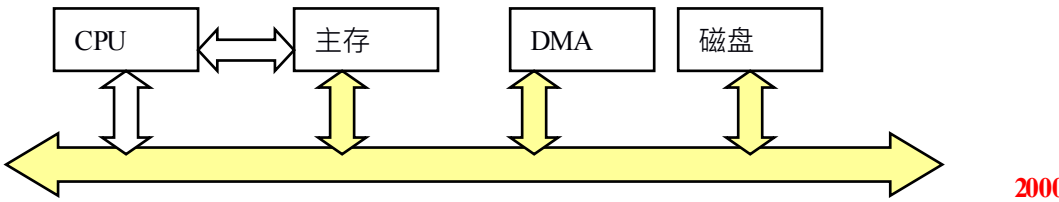
(3) 直接寻址的“存储器-存储器”SS 型加法指令在无磁盘 I/O 操作打扰时的执行时间为多少？当磁盘 I/O 操作与一连串这种 SS 型加法指令执行同时进行时，则这种 SS 型加法指令的最快和最慢执行时间各是多少？（假定采用多周期处理器方式，CPU 时钟周期等于主存周期）**参考答案：**

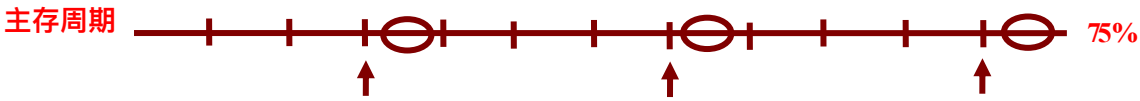
(1) **磁头旋转一周所需要的时间为：60s / 3000 = 20ms**

所以读完单个数据块所需时间为：（20 – 2x2）/ 2 = 8ms

所以最大数据传输率为：8000B / 8ms =1×10⁶Bps

(2) **因为总线宽度为 16 位，而最大数据传输率为 1×10⁶Bps，故每隔 2B/1MBps = 2000ns 需要产生一个** DMA 请求（准备好 16 位数据），而主存周期为 500ns（在总线上与主存交换 16 位数据），**即每 4 个存储周期中有一次被 DMA 挪用，故主存频带空闲比是 75%**

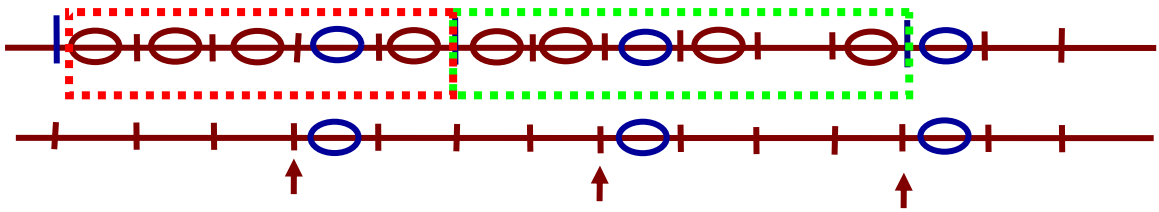




(3) **无 I/O 打扰时，执行一条直接寻址的 SS 型加法指令的时间为：**

取指 500ns+取源 500ns+取目 500ns+执行 500ns+存结果 500ns = 2.5 μs

当磁盘 I/O 操作与一连串这种 SS 型加法指令同时进行时，则 CPU 和 DMA 可能同时要求访问主存，此时 DMA 优先级高，CPU 的访存请求被延迟，从而导致指令执行时间延长。



由此可见，最好的情况是在 SS 型加法指令执行过程中没有访存冲突，此时最快，需 5 个存储周期的时间：500nsx5=2.5μs;最坏的情况是有一次访存冲突，此时最慢，需 6 个存储周期的时间：500nsx6=3μs。

15. 假定某计算机所有指令都可用两个总线周期完成，一个总线周期用来取指令，另一个总线周期用来存取数据。总线周期为 250ns，因而，每条指令的执行时间为 500ns。若该计算机中配置的磁盘上每个磁道有 16 个 512 字节的扇区，磁盘旋转一圈的时间是 8.192ms，则采用周期挪用法进行 DMA 传送时，总线宽度为 8 位和 16 位的情况下该计算机指令执行速度分别降低了百分之几?

参考答案：

磁盘存取 1 个字节数据需要的时间：8.192ms / 16x512B = 1000ns / B = 4 个总线周期

总线为 8 位： 每 4 个总线周期中要挪用 1 个进行 8 位传送，每次挪用都导致指令执行延后 1 个周期，即原来要 2N 个周期完成的指令现在需要 2.5N 个周期完成，因此降低了 0.5/2=25%

磁盘存取 2 个字节数据需要的时间：8 个总线周期

总线为 16 位：每 8 个总线周期挪用 1 次。降低了 1/8 = 12.5%。