1. 判断以下描述更符合SRAM还是DRAM，还是都符合.

|  |  |  |
| --- | --- | --- |
|  | SRAM | DRAM |
| (1)访问速度更快 |  |  |
| (2)每比特需要的晶体管数目少 |  |  |
| (3)单位容量造价更便宜 |  |  |
| (4)常用作主存 |  |  |
| (5)需要定期刷新 |  |  |
| (6)断电后失去存储的信息 |  |  |
| (7)支持随机访问 |  |  |

1. 已知一个双面磁盘有2个盘片、10000个柱面，每条磁道有400个扇区，每个扇区容量为512字节，则它的存储容量是\_\_\_\_\_\_\_GB
2. 已知一个磁盘的平均寻道时间为6ms，旋转速度为7500RPM，那么它的平均访问时间大约为\_\_\_\_\_\_\_ms
3. 已知一个磁盘每条磁道平均有400 个扇区，旋转速度为6000RPM，那么它的平均传送时间大约为\_\_\_\_\_\_\_\_ms
4. 考虑如下程序

|  |
| --- |
| **for** (**int** i = 0; i < n; i++) {  B[i] = 0;  **for** (**int** j = 0; j < m; j++)  B[i] += A[i][j];  } |

判断以下说法的正确性

|  |  |
| --- | --- |
| ( ) | 对于数组A的访问体现了时间局部性。 |
| ( ) | 对于数组A的访问体现了空间局部性。 |
| ( ) | 对于数组B的访问体现了时间局部性。 |
| ( ) | 对于数组B的访问体现了空间局部性。 |

1. 高速缓存
   1. 一个容量为8K的直接映射高速缓存，每一行的容量为32B，那么它有\_\_\_\_\_\_组，每组有\_\_\_\_\_行。
   2. 一个容量为8K的全相联映射高速缓存，每一行的容量为32B，那么它有\_\_\_\_\_组，每组有\_\_\_\_\_行。
   3. 一个容量为8K的4路组相联映射高速缓存，每一行的容量为32B，那么它有\_\_\_\_\_组，每组有\_\_\_\_\_行。
   4. 一个容量为16K的4路组相联高速缓存，每一行的容量为64B，那么一个16位地址0xCAFE应映射在第\_\_\_\_\_组内。
2. 判断以下说法的正确性

|  |  |
| --- | --- |
| ( ) | 保持块大小与路数不变，增大组数，命中率一定不会降低。 |
| ( ) | 保持总容量与块大小不变，增大路数，命中率一定不会降低。 |
| ( ) | 保持总容量与路数不变，增大块大小，命中率一定不会降低。 |
| ( ) | 使用随机替换代替LRU，期望命中率可能会提高。 |

1. 有以下定义：

|  |
| --- |
| // 以下都是局部变量  **int** i, j, temp, ians;  **int** \*p, \*q, \*r;  **double** dans;  // 以下都是全局变量  **int** iMat[100][100];  **double** dMat[100][100];  // 以下都是函数  **int** foo(int x); |

如果将下列左侧代码优化为右侧代码，有没有可能有副作用？如果有请说明.

|  |  |  |
| --- | --- | --- |
| (1) | ians = 0;  **for** (j = 0; j < 100; j++)  **for** (i = 0; i < 100; i++)  ians += iMat[i][j]; | ians = 0;  **for** (i = 0; i < 100; i++)  **for** (j = 0; j < 100; j++)  ians += iMat[i][j]; |
| (2) | dans = 0;  **for** (j = 0; j < 100; j++)  **for** (i = 0; i < 100; i++)  dans += dMat[i][j]; | dans = 0;  **for** (i = 0; i < 100; i++)  **for** (j = 0; j < 100; j++)  dans += dMat[i][j]; |
| (3) | **for** (i = 0;i < foo(100);i++)  ians += iMat[0][i]; | temp = foo(100);  **for** (i = 0; i < temp; i++)  ians += iMat[0][i]; |
| (4) | \*p += \*q;  \*p += \*r; | temp = \*q + \*r;  \*p += temp; |

1. 假设已有声明int i, int sum, int \*p, int \*q, int \*r, const int n = 100, float a[n], float b[n], float c[n], int foo(int), void bar(),以下哪种优化编译器总是可以进行?

|  |  |  |
| --- | --- | --- |
| A | **for**(i = 0; i < n; ++i){  a[i] += b[i];  a[i] += c[i];  } | **float** temp;  **for**(i = 0; i < n; ++i){  temp = b[i] + c[i];  a[i] += temp;  } |
| B | \*p += \*q;  \*p += \*r; | **int** temp;  temp = \*q + \*r;  \*p += temp; |
| C | **for**(i = 0; i < n; ++i)  sum += i\*4; | **int** N = n \* 4;  **for**(i = 0; i < N; i += 4)  sum += i; |
| D | **for**(i = 0; i < foo(n); ++i)  bar(); | **int** temp = foo(n);  **for**(i = 0; i < temp; ++i)  bar(); |

1. 阅读下列C 代码以及它编译生成的汇编语言

|  |
| --- |
| **long** func() {  **long** ans = 1;  **long** i;  **for** (i = 0; i < 1000; i += 2)  ans = ans ?? (A[i] ?? A[i+1]);  **return** ans;  } |
| func:  **movl** $0, %**edx**  **movl** $1, %**eax**  leaq A(%rip), %**rsi**  **jmp** .L2  .L3:  **movq** 8(%**rsi**,%**rdx**,8), %**rcx** // 2 cycles  ?? (%**rsi**,%**rdx**,8), %**rcx** // k + 1 cycles  ?? %**rcx**, %**rax** // k cycles  **addq** $2, %**rdx** // 1 cycles  .L2:  **cmpq** $999, %**rdx** // 1 cycles  **jle** .L3  **rep ret** |

该程序每轮循环处理两个元素。在理想的机器上（执行单元足够多），每条指令消耗的时间周期如右边所示。

(1) 当问号处为乘法时，k = 8。此时这段程序的CPE为\_\_\_\_\_

(2) 当问号处为加法时，k = 1。此时这段程序的CPE为\_\_\_\_\_

1. k \* k循环展开在k很大时反而可能获得较差的效果，这是因为k很大时会导致\_\_\_\_\_\_\_\_