3

(1)

每个内存条需要16KB / (1K x 8bit) = 16个DRAM芯片.

(2)

48KB的主存需要48 / 16 = 3个内存条.

(3)

因为最大寻址空间为64KB, 且按字节编址, 且有64KB = 216 B, 所以主存地址共有16位. 其中低10位用作DRAM芯片的内地址, 内地址的高5位用作行地址, 低5位用作列地址.

5

平均寻道时间为10ms, 磁盘控制器的开销为2ms, 数据传输时间为4KB/40MB 秒, 即4 x 210 / 40 x 106 秒, 即0.1024ms, 平均旋转等待时间为 (1000/(7200/60))/2 ms, 也即25/6ms, 数据处理时间为 (20000 / 500M) 秒, 即0.04ms, 因为数据块在磁道上随机存放, 所以每个数据块进行一次题述操作的时间都相同, 均为 (10 + 2 + 0.1024 + 25/6) \* 2 + 0.04 ms, 大约为32.58ms, 每秒可以完成的的操作次数是 1000/32.58 次, 约为30次.

8

(1)

1GB = 230B, 所以主存地址共有30位. cache总共有64KB/128B = 29行, 块大小为128B = 27B. 所以这30位主存地址划分为高14位的标记, 中间9位的cache行索引, 低7位的块内地址.

(2)

因为采用直接映射加直写法, 所以每个cache行只包括1位的有效位, 14位的标记位和128字节的数据. 由(1)可知共有29个cache行, 数据区总大小为64KB, 所以cache总容量为(29 \* 15/213 + 64)KB = 64.9375 KB.

12

(1)

x和y数组中的元素都只被访问了一次, 所以没有时间局部性. 因为对x和y数组里的元素都是顺序访问的, 且数组中的元素是顺序存放的, 所以空间局部性好.

命中率和块大小, cache行数, x和y的存放地址, cache的实现策略等均有关系, 这里缺少信息无法推断.

(2)

cache总共有32/16 = 2行, 于是内存地址的32位划分为高27位标记, 中间1位cache行号, 低4位块内地址. x的地址的低五位是00000, x数组共有8个4字节的元素, 它们连续存放在内存中, 于是有x[0] – x[3]映射到0号cache行, x[4] – x[7]映射到1号cache行. y的地址的低五位也是00000, 且也由8个4字节元素组成, 所以y[0] – y[3]映射到0号cache行, y[4] – y[7]映射到1号cache行. 该程序每次都以相同下标访问x和y, 这会导致它们在cache里每次都相互替换, 所以该程序cache命中率为0.

(3)

cache总共有32/8 = 4行, 采用2路组相联, 于是内存地址的32位划分为高28位标记, 中间1位cache组号, 低3位块内地址. 第一次访问时, x[0], y[0]均未命中, 但会将x[0], x[1]以及y[0], y[1]搬运到cache第0组的两个不同行中, 于是在下一次访问时均会命中cache. 之后的访问会重复进行类似的处理, 于是最终cache命中率为百分之50.

(4)

其余条件与(2)相同, y的低五位变为10000, 于是x[0] – x[3]和y[0] – y[3]分别映射到第0个cache行和第1个cache行, 而x[4] – x[7]和y[4] – y[7]分别映射到第1个cache行和第0个cache行, 于是只有在访问下标为0和4的元素时不会命中, 其余情况均命中率, 所以cache命中率为6/8 = 百分之75.

23

(1)

addl (%edx, %ecx, 4), %eax

寻址方式为基址 + 比例变址 + 偏移量.

(2)

由表6.2可知, CS和DS对应的段描述符中的DPL都是3, 且题目给出了CPL = 3, 于是这里取指令和取数均不会发生存储保护错.

同时, 由表6.2还可以知道, CS和DS对应的段描述符中的段基址都是0x0. 由线性地址等于对应段基址加上虚拟地址的低32位可知, 要知道MMU得到的线性地址, 现在只需要知道指令I和a[50]的虚拟地址即可. 指令I的虚拟地址由题目给出, 为0x8049c08; a[50]的地址为0x804d000 + 4 \* 50 = 0x804d0c8. 于是, 取指令过程中MMU得到的指令的线性地址为0x8049c08; 取数过程中MMU得到的操作数的线性地址为0x804d0c8.

(3)

movl $0x0, %ecx

.LOOP

cmpl %ecx, %ebx

jle .EXIT

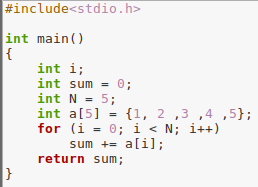
addl (%edx, %ecx, 4), %eax

addl $0x1, %ecx

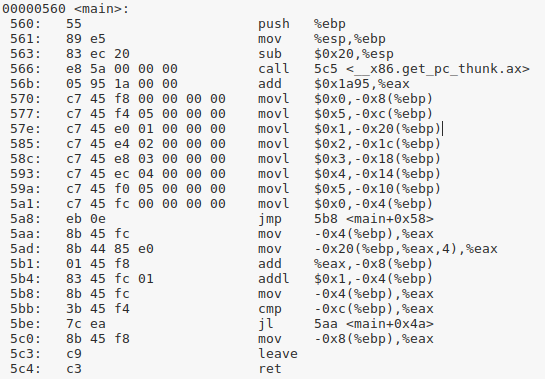
jmp .LOOP

.EXIT

c代码如下:



反汇编结果如下:



(4)

因为已启用分段加分页管理, 所以PE和PG都是1.

(5)

因为指令I的地址不在一页的起始处, 而a的地址在一页的起始处, 所以取指令时不会发生缺页异常, 而取操作数时有可能发现缺页异常. 若取操作数时发生缺页异常, 则页故障线性地址为0x804d000(a[0]的线性地址), 存放在控制寄存器CR2中.

(6)

每页大小4KB的情况下, 将指令I的线性地址划分为高20位和低12位. 高20位是指令I所在页的虚页号, 为0x08049, 低12位为页内偏移量为0xc08.

其中高20位的前10位为页目录索引, 为0000 1000 00; 后10位为页表索引, 为 00 0100 1001.

第一次执行指令I时, 字段P为1, 表示该页已在内存中; 字段R/W为0, 表示该页只能读和执行, 不能写; 字段U/S为1, 表示该页可被用户进程访问; 字段A为1, 表示该页已被访问过; 字段D为0, 表示该页未被修改过.

(7)

指令I第一次执行过程中, 取指令时不会发生TLB缺失, 因为指令I不在页面起始处, 在执行I前面的指令发生TLB缺失时, 便会将指令I所在页的页表项装载进TLB中. 但取操作数a[0]时可能会发生TLB缺失, 因为a[0]在一个页面的起始处.

TLB共有16/4 = 4组, 于是虚页号的高18位为TLB标记, 低2位为TLB组索引.

第一次执行到指令I时, 指令I对应的虚页号为0x08049, 对应的TLB标记为 0000 1000 0000 0100 10, 转换成16进制是0x02012; 对应的TLB组索引是01, 即在第1组中.

在TLB第1组中有标记为0x02012且标记位为1的一项, 于是TLB命中, 对应的页框号为0x028b0, 加上指令I的线性地址的低12位表示的页内偏移量, 得到指令I所存放的主存地址为0x028b0c08.

(8)

cache总共有8kb / 32b = 28行, 采用2路组相联映射, 则共分为27组.

于是主存地址的32位被分为高20位标记, 中间7位cache组号, 低5位块内偏移量. 指令I的主存地址的低12位0xc08, 低5位为0 1000, 这表明指令I不在一个主存块的起始位置, 因此在第一次执行指令I的取指令过程中, 不会发生cache缺失, 因为在取它前面的指令的时候就已经把指令I装入了cache中.

指令I的主存地址的中间7位为 1100 000, 转换成十进制是96, 于是指令I所在主存块应映射到指令cache的第96组中.

(9)

当N为2000时, 数组a大小为8000B. 因为a的起始地址为一个页面的开始处, 所以数组a占用8000B/4KB = 1.953125个页面. 第一个页的虚页号为0x0804d, 第二个页的虚页号为0x0804e. 因为1200 \* 4 = 4800 > 4096, 所以数组元素a[1200]在第二个页中.