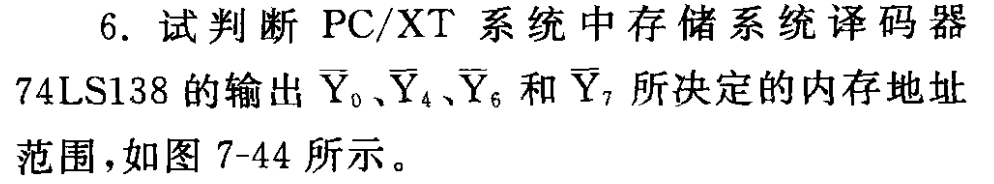
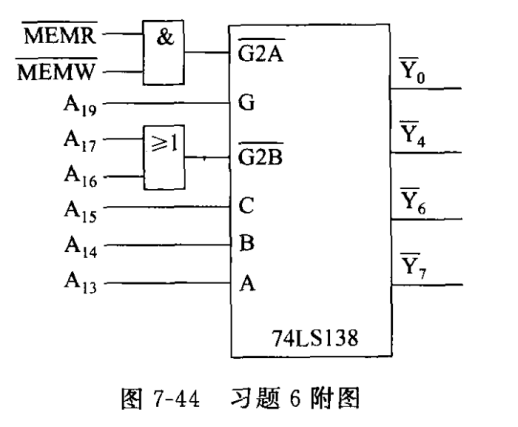
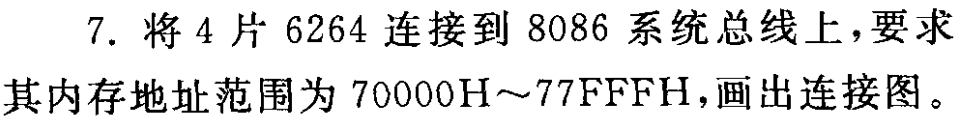
计算机组成作业

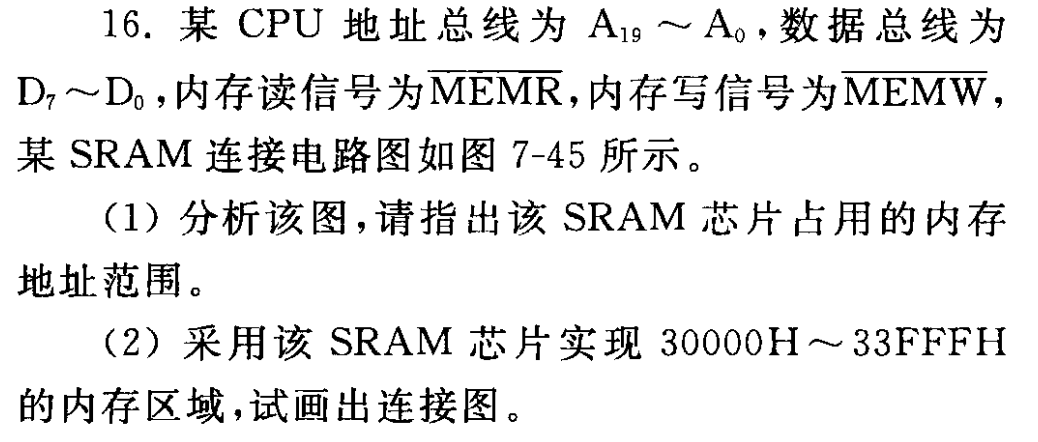
第四章

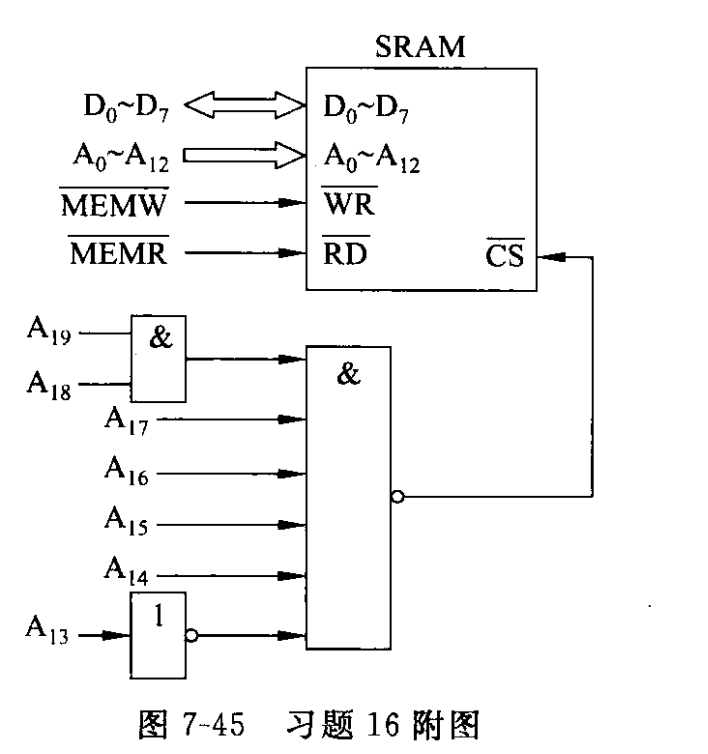


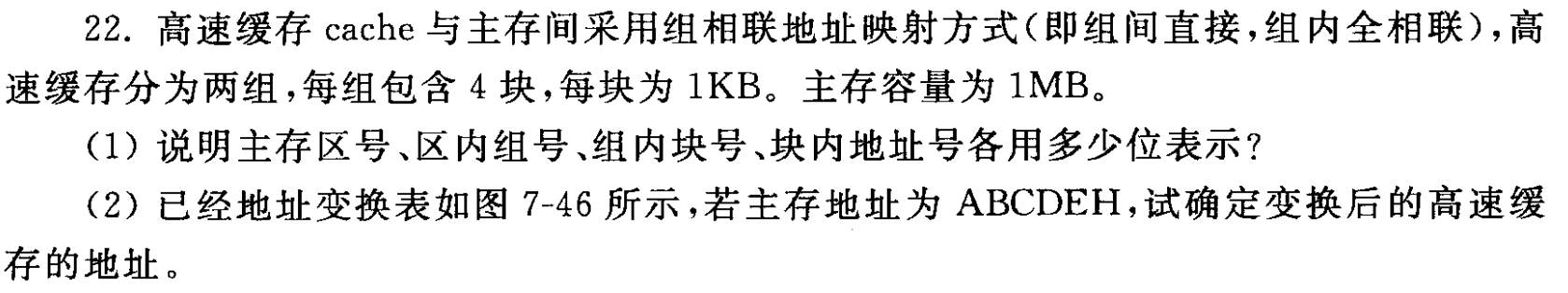


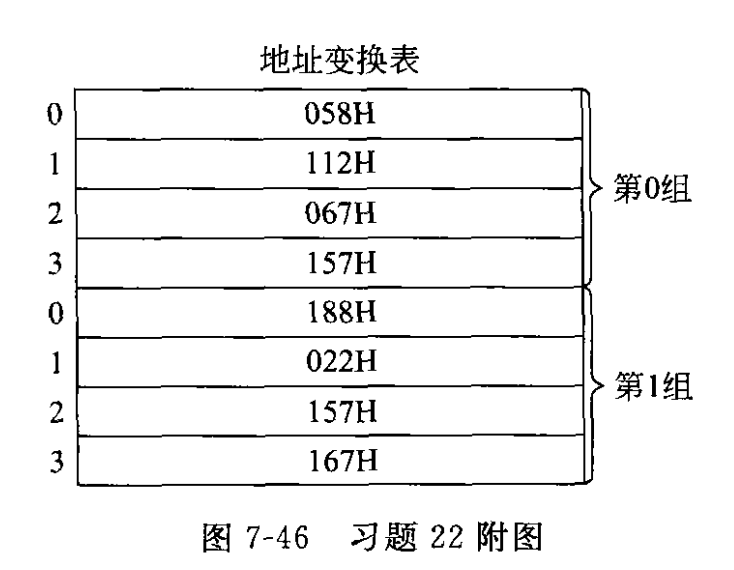
A\_{18}取什么值？











1. Tip：各块总长度与主存容量对应长度一致
2. 1 0101 0111 1 0011011110

TAG为157H 组号是1 块内地址为后十位

变换后地址：关键是把Tag部分去掉，但是保留块内地址，中间的部分要能表征存在Cache的哪里了

这题是组相连，所以要组号+组内块号

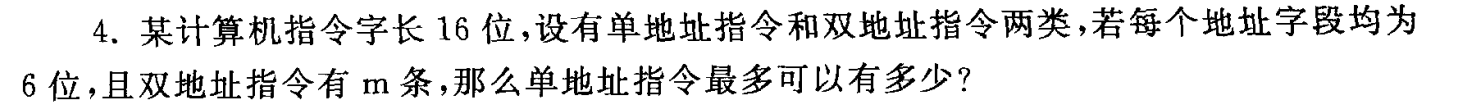
如果是全相联或者直接相联，那就要块号（三位）

将ABCDEH展开为二进制，分割后发现，组号是1，（块号是3？不是！这是全相联映射，可以放在组内的任意一块！所以块号不是在这里用的）将除了块内地址和区内组号的部分拼接在一起，得到101010111，从右往左划分为十六进制，1 0101 0111， 就得到了157H，发现是在第一组第二块。

题目要求的东西，高速缓存地址，就是主存块在Cache中存放的位置，加上快内地址，所以就是 **1 +2（组内块号）+10 块内地址（10位）**

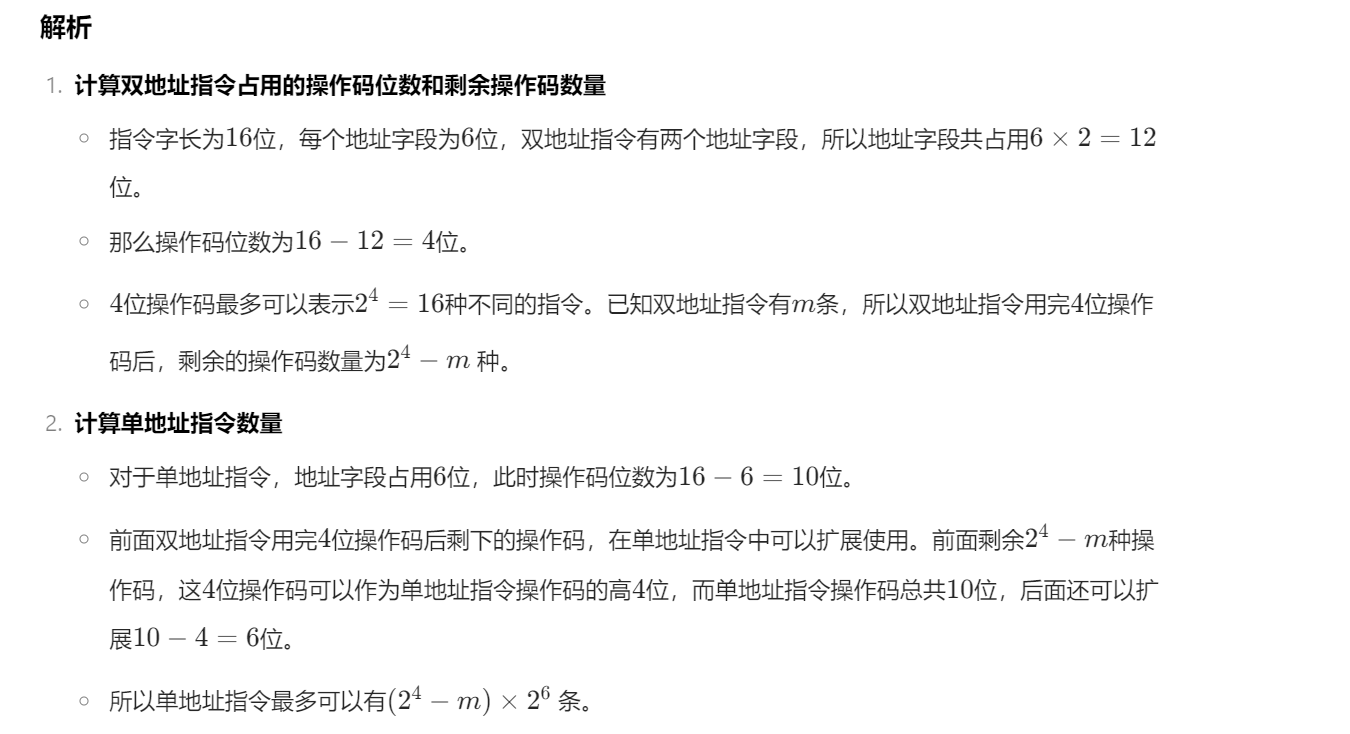
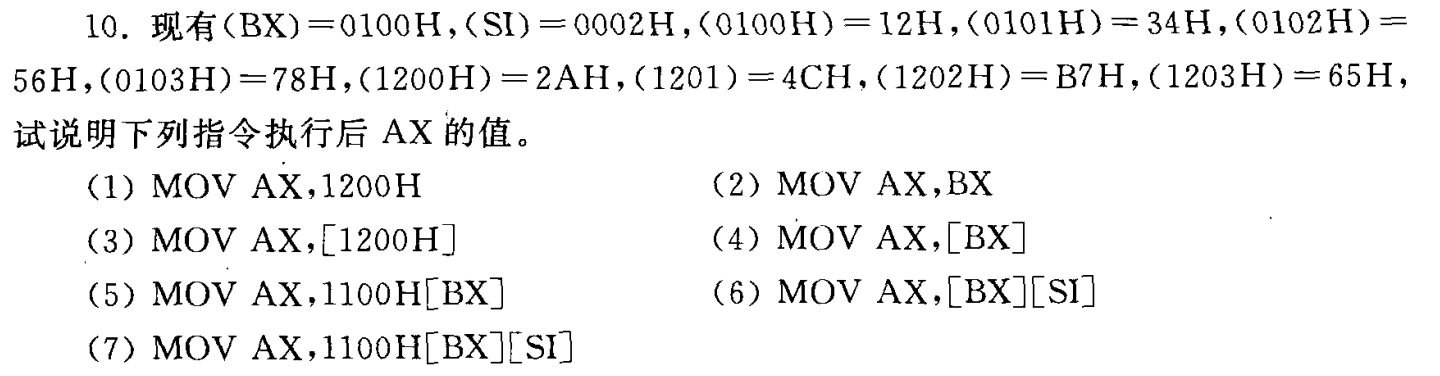
第五章

**不熟！**



**不变的是地址码的位数，而操作码是根据地址码的数量变化，用指令字长-地址码数量\*每个地址码长度，这样算出来的**

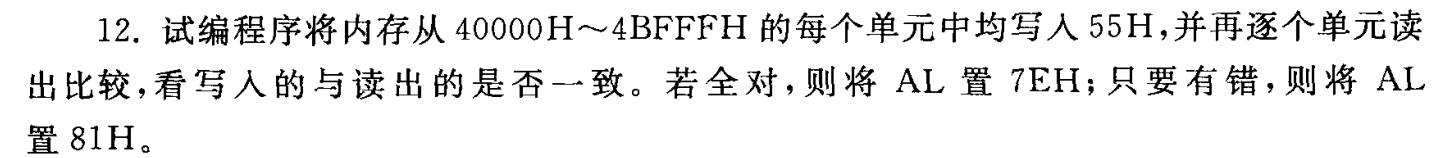
**单地址码和双地址码之间的制约关系：保证地址码不可重复性【前缀码思想】**

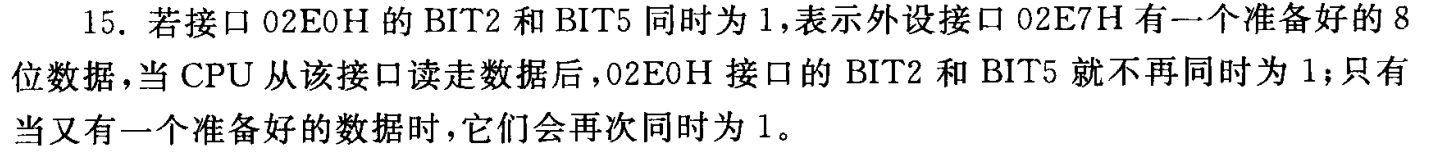
注意：

1. 从1200H开始的两个字节，查找结果**从高到低**拼接在一起，得到4C2AH
2. 0100H开始的两个字节，得到3412H
3. 0100H + 1100H = 1200H【逢十六进一！】，从1200H开始的两个字节，4C2AH
4. 5.基址相对寻址（以BX为基址寄存器） 6.基址变址寻址（有两个寄存器，si是变址寄存器） 7.相对基址变址寻址（有偏移量就是了两

| **指令序号** | **指令** | **寻址方式** | **有效地址计算（若有）** |
| --- | --- | --- | --- |
| 1 | MOV AX,1200H | 立即寻址 | - |
| 2 | MOV AX,BX | 寄存器寻址 | - |
| 3 | MOV AX,[1200H] | 直接寻址 | 有效地址 = 1200H |
| 4 | MOV AX,[BX] | 寄存器间接寻址 | 有效地址 = BX（0100H ） |
| 5 | MOV AX,1100H[BX] | 寄存器相对寻址  【基址寻址】 | 有效地址 = BX + 1100H = 1200H |
| 6 | MOV AX,[BX][SI] | 基址变址寻址  【变址寻址】 | 有效地址 = BX + SI = 0102H |
| 7 |  | 相对基址变址寻址 |  |
| 补充 | JMP 指令如JMP NEAR PTR 100H | 【相对寻址，用于跳转】 |  |



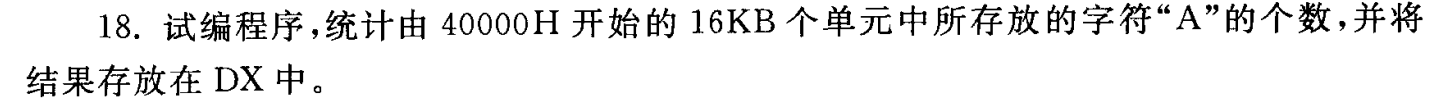
【代码见csdn笔记】



试编写程序，从上述接口读入32个数据，顺序存放在A0100H单元开始的各单元中

**不熟！轮询思想**，反复查询是否满足执行条件，不满足就返回，继续询问；满足的话就执行接受操作，然后再跳转到轮询中，如此反复

【代码见CSDN】



关于字符‘A’：字符在计算机中以 ASCII 码形式存储 。字符‘A’ 的 ASCII 码值用十六进制表示为 41H ，用十进制表示是 65 。所以判断寄存器中的值是否为字符‘A’ ，实际是判断寄存器中的值是否等于 41H（或 65 ）

【代码见csdn笔记】

T21看b站视频介绍



**复杂指令集CICS【**Complex Instruction Set Computer**】**：一条指令完成一个复杂的基本功能，对应的硬件实现也非常复杂，条目更多【类似C++的那么多库】

**精简指令集RISC【**Reduced Instruction Set Computer**】**：一条指令只完成一个基本动作，复杂指令可以通过简单指令的组合来实现（也就是微程序），硬件实现更加简单，功耗更低【类似C语言中的最基本的基础语法】，

只有Load和Store两个指令访问主存，需要更多寄存器，为了在这种简单指令的基础上实现复杂的运算，需要频繁地进行数据读取和写入操作。更多的寄存器可以提供快速的数据存储和访问，减少对内存的访问次数

