各种小细节、易错点

1、各种地址

块地址 = 块号，块地址4B→最多有 2^{32} 个块

磁盘块的链接指针，放的是磁盘块号，有32位就有32个磁盘块！

😱😱😱**字地址**：这个字的首地址（首字节地址），而不是按字编制！

例如字节编址的计算机M，字长32位，字地址 $(1100\quad 1100)_2$

```
我之前理解为，这是第1100 1100个字，对应字节地址是1100 1100 00
事实上，这指的意思是，这个字的地址从1100 1100开始...
```

Cache的字地址：

如果是全相联映射：行内字地址（全相联映射不用行号，直接用比较器比较tag部分定位）
如果是直接映射：Cache行号+行内字地址（直接映射靠index定位）
如果是组相联映射：Cache组号+行内字地址（因为组内是相联存储器，不需要定位到哪一行）

```
总结：Cache总位数不包括Cache地址，Cache地址就是内存地址去掉TAG部分后剩下的部分！
```

Cache地址位数就是表示Cache容量大小所需要的位数，比如64KB按字节编址，就是16位。（跟主存地址一样）

虚拟页（逻辑）和物理页（内存中）一样大，是分配内存资源的单元，通常为4KB
物理**块/行/页框**在Cache和主存，是Cache和主存交换单位，一般在32B-512B之间
磁盘块是主存和磁盘交换单元，一般是512B；
簇是文件存储空间分配单位。

2、计组大题，提到OFFSET使用补码表示的时候，小心三个地方：1️⃣加减运算要使用补码运算方法；2️⃣OFFSET可能有系数，仔细看题目；3️⃣OFFSET可能是个地址，真正的偏移量在寄存器里，小心表示范围这个东西！

3、**连续的取和连续的发送，默认按找流水线方式**来：取一个发一个，取和发并行执行。类似计网那种。

4、题目中提到补码，计算的时候一定老老实实用补码的方法运算

5、时间复杂度（第一道题）1️⃣看内外层是否相关？2️⃣如果不相关，单独算乘起来；3️⃣如果相关，则列式相加

6、窗口大小，如果有单位（比如KB），就要写KB

7、链式存储结构的示意图，要把结点里面的内容都表示出来。结点的内容记得包括题干中的关键信息

8、⚠️题干涉及m,n两个字母的，小心复杂度是 $O(m+n)$ 或者 $O(\max(m,n))$

9、二维数组 `[行][列]`，三维数组 `[层][行][列]` 。只要记：前面的大

10、红黑树的黑高：当前结点到最底层失败/NULL/叶结点的距离，包括叶结点不包括当前结点（根结点）。树高：包括当前结点（根结点）不包括叶结点。

11、虚拟页面已缓存的意思：已经调入主存中。

虚拟页面已分配的意思：虚拟页面已经和物理页面在页表中（可能还在TLB中）建立映射关系

虚拟页面的状态不可能是已缓存未分配（已调入主存但没建立虚拟页和物理页的映射关系）

12、操作系统甘特图：慢慢画，慢就是快

13、二进制指数退避

要区分发送、碰撞以及重传次数：

第i次发送，那么之前发生了i-1次碰撞，这次碰撞即是第i-1次重传，k值选i-1

14、窗口大小以字节为单位，窗口尺寸以帧为单位

15、用双引号缩写只能缩写整数个域0；每个域的前导0可以省略，后导0不可省略

16、以太网最短帧间间隔计算方法：512比特时间，即当前链路传输512比特（即最小帧长64B）需要的时间，10M网络是51.2μs，100M网络是5.12μs。
以太网帧最短64B，则IP中的内容46B，去掉首部数据部分26B！

17、MTU是指数据报总长度！首部+数据部分。

18、NAT换的不只是IP地址，可能连端口号都换（换整个套接字），所以NAT工作在传输层

19、HTTP1.1传文件🚀

注意题干信息是否已经建立连接！建立连接算1个RTT（因为第三次握手已经开始传数据了）如果是小文件，一个RTT传一个（2011大题）；如果是大文件，拆成多个MSS，每个大文件按照TCP拥塞控制里那种方法发送（2022第40题）

20、运算符栈，记得把小括号给消掉！

21、(float)只管1个括号！

```
int a = 5, b = 8;
    cout << (float)(a + b) / 2 << endl; //结果为6.5
    cout << (float)((a+b)/2)    <<endl; //结果为6
```

22、指令判误优先级：优先字长、后看编码

23、loop的值就是第一条指令的地址；exit的地址就是最后一条指令执行完后的地址

24、计网，一行是4B，两个字母是1B！

25、父进程撤销后，子进程可能有两种状态：1️⃣子进程一并被终止；2️⃣子进程成为孤儿进程，被Init进程领养。所以父进程的撤销并不一定引起子进程的撤销

26、网络层及以下是点到点；传输层及以上是端到端。

27、多模块存储器才可以流水线准备数据发数据

28、二进制信息块=以太网帧

29、路由器端口的值一般是这个网络的HOSTID为1的地址。不要用路由聚合！！！路由聚合是用在路由表项的

30、指令系统的偏移量，用补码表示！也用十进制表示。两个都写！

考前一晚

虚拟机、多核

Floyd

5、各种算法的各种复杂度（前一晚）

6、图算法的各种复杂度、查找排序算法的各种复杂度、其他很少见的复杂度（前一晚）

8、顺序表和单链表的常见操作：多指针、动态删除、原地反转、快排快查

9、树的常见操作：前中后、层次遍历、（⚠️递归方法）统计符合要求的结点数量

10、图的常见操作：BFS、DFS、统计度、拓扑排序（矩阵/表的都准备）

12、冲刺150：朴素匹配、并查集优化

```
int BruteMatch(char A[],char B[]){
    int i = 0, j = 0, k = 0;
    while(i < strlen(A) && j < strlen(B)){
        if(A[i] == B[j]){
            i++;
            j++;
        }else{
            k++;
            i = k;
            j = 0;
        }
    }
    if (j >= strlen(B))
        return k;
    else
        return -1;  //表示不成功
}
```

13、浅看一遍25协议

14、

表述题 🍞 🍔 🍗 🍟

散列表平均查找长度的影响因素：装填因子、冲突处理方法、散列函数

影响总线带宽的因素：总线宽度、总线频率、数据传输方式

DMA接口的程序中断部件的作用：向CPU提出传输结束。没有其他功能

基于时间片的调度算法：时间片轮转算法、多级反馈队列调度算法

CPU利用率低，交换空间和磁盘利用率高，应该：增加内存容量、减少多道程序道数

在OSI参考模型中，实现系统间二进制信息块的正确传输，为上一层提供、无错误的数据信息的协议层是【数据链路层】

计网协议

协议名称	位于层级	协议	端口	其他	主要用途	主要特点
SW,GBN,SR	链路层				用于流量控制	GBN:2^n-1,累计确认 SR:2^{n-1},选择确认
ALOHA	链路层				动态介质访问控制的一种	随意 普通18.4，时隙36.8
CSMA 1坚持,非坚持 p坚持	链路层				抢占信道的协议	非坚持是隔一段时间后重听 p坚持是p概率发，1-p下一时间
CSMA/CD,CA	链路层				抢占信道的协议	CD:二进制指数避让(10,<=16) CA:适用于无线局域网
PPP	链路层				使用最广泛的数据链路层协议，电话拨号时就用PPP	面向字节，全双工，CRC，无连接，无序号确认，不可靠 封装方法+LCN链路控制+NCP网络控制
RIP	应用层		520	UDP	距离向量路由算法	UDP，跳数<=15,30s刷新，180s超时，300删除适，小网络，慢收敛，内容是自己路由表
OSPF	网络层			IP	链路状态路由算法	链路状态变化时，洪范法 内容是本路由器的所有出度

协议名称	位于层级	协议	端口	其他	主要用途	主要特点
BGP	应用层		179	TCP	边界网关协议	会话人，数量很少
IPv4	网络层			IP		1总8片首4，校验和只管首部，首部20B
IPv6	网络层			IP		首部40B，没有填充字段，有效载荷长度不包括首部，单位为1
ARP	网络层			IP	IP地址转变为MAC地址	第一次发ff-ff-ff-ff-ff-ff
DHCP	应用层		68	UDP	动态分配主机	客户/服务器方式，即插即用DHCP发现、提供、请求、确认
ICMP	网络层			IP	主机和路由器之间传递控制信息的报文	面向无连接
IGMP	网络层			IP	组播	
TCP	传输层	6				数据偏移以4B为单位实际上是首部长度，校验所有（不含伪首部）
UDP	传输层	17				UDP长度包括首部+数据，校验和检验所有（不含伪首部）
DNS	应用层		53	UDP	域名→IP地址	分为迭代(本地发多次)和递归(本地发1次)
STMP	应用层		25	TCP	(客/服)向邮件服务器发送邮件	

协议名称	位于层级	协议	端口	其他	主要用途	主要特点
POP3	应用层			TCP	从邮件服务器读取邮件	
HTTP	应用层		80	TCP	超文本传输协议	1.0默认非持续，1.1默认持续

数据结构

时间复杂度 😇

树后=林中=二中

先 = 先

树	森林	二叉树
先根遍历	先序遍历	先序遍历
后根遍历	中序遍历	中序遍历

1、拓扑排序只有有向图有，邻接矩阵V2，邻接表V+E
2、Prim(V2)、Kruskal(ElogE)、dijkstra(V2)、Floyd(V3)所有算法都相同
3、BFS、DFS，邻接矩阵V2，邻接表V+E
4、除了Kruskal，其他带E的都是V+E
5、Kruskal适合边稀疏的图，dijkstra适合边密集的图
6、Prim、Kruskal、dijkstra都是贪心算法，而Floyd算法属于动态规划算法

1 排序算法								
算法	时间复杂度		空间复杂度	稳定性	适用范围	每趟确定一个位置	比较次数序列相关	趟数 序列相关
	平均	最坏						

1 排序算法								
直接插入	n2	n2	1	稳定	顺序表、链表	✔	✔	n-1
折半插入排序	n2	n2	1	稳定	顺序表	✔	✔	n-1
希尔排序	n1.3	n2	1	不稳定	顺序表		✔	分组数
冒泡排序	n2	n2	1	稳定	顺序表、链表	✔	✔	✔
快速排序	nlogn	n2	logn(最差时为n)	不稳定	顺序表、链表	✔	✔	✔
优化的快速排序	nlogn	nlogn	logn	不稳定	顺序表	✔	✔	✔
简单选择排序	n2	n2	1	不稳定	顺序表、链表	✔		n-1
堆排序	nlogn	nlogn	1	不稳定	顺序表、链表	✔	✔	n-1
归并排序	nlogn	nlogn	n	稳定	顺序表、链表			log2n
基数排序	d(n+r)	d(n+r)	r	稳定	链表			

层次遍历、BFS、DFS

```
void LayerOrder(BTree T) {
    Queue<BTNode*> Q;
    BTNode* p = T;
    Q.EnQueue(T);
    while (!Q.isEmpty()) {
        Q.DeQueue(p);
        visit(p);
        if (p->lchild)  Q.EnQueue(p->lchild);
        if (p->rchild)  Q.EnQueue(p->rchild);
    }
}
```

```
int visited[MAXV] = { 0 };
void BFS(Graph G,int v) {
    if (visited[v]) return;      //如果访问过，直接跳过这个结点
    int J;                       //结点编号
    ArcNode* p;
    Queue<int> Q;   Q.EnQueue(v);

    while (!Q.isEmpty()) {
        Q.DeQueue(J);
        if (visited[J] == 0) {       //如果没访问过，将其访问，并将所有指向的未访问过的结点都入队
            visited[J] = 1;
            visit(G, J);
            p = G.vexes[J].firstArc;
            while (p) {
                if (visited[p->adjvex] == 0) {
                    Q.EnQueue(p->adjvex);
                }
                p = p->nextArc;
            }
        }
    }
}
```

```
int visited[MAXV] = { 0 };
void DFS(Graph G, int v) {
    ArcNode* p = G.vexes[v].firstArc;
    visited[v] = 1;
    visit(G, v);
    while (p != NULL) {
        if (visited[p->adjvex] == 0)
            DFS(G, p->adjvex);
        p = p->nextArc;
    }
}
```

树的性质 ⭐

【所有树】结点数=度数+1

【二叉树】给定n，最少 $\lceil \log_2(n+1) \rceil$ 或 $\lfloor \log_2 n \rfloor + 1$ 层

【K叉树】给定层高h，最多有 $\frac{k^h - 1}{k-1}$ 个结点；给定结点n， $h \geq \lceil \log_k [n(k-1)+1] \rceil$

给定结点i（从1开始编号），其第一个孩子结点是 $k(i-1)+1+1$，其中前i个结点每个都有k个孩子，+1是根结点，再+1是这个结点的第一个孩子

【二叉哈夫曼树】N个结点组成的哈夫曼树有2N-1个结点

【K叉哈夫曼树】K叉N个结点，如果N-1不能被K-1整除，补：(K-1) - (N-1)%(K-1) 个空结点；

N个结点组成的K叉哈夫曼树有N+(N-1)/(K-1)个结点（不算补的空结点）

【完全/满二叉树】N0一定比N2多一个，偶数有1个N1，奇数没有N1

【二叉平衡树】层高h求最少结点数（1，2，4，7，12，20，33，...），最多节点数同满二叉树；结点数求最少层高，同满二叉树。

【二叉排序树】根节点比左子树上所有结点都要大；比右子树上所有结点都要小【✔】

比左节点大，比右节点小【✖】

【二叉线索树】

带头结点的线索树，原来的NULL指向头结点

非空线索树至少有1个NULL指针域，很多情况下有2个NULL指针域，注意区分

TAG=0代表孩子，1代表线索（0比较像褶裥，1是长条的）

后序二线索树需要用到栈（用于保存前驱结点）

二叉树的前序、中序、后序序列中，所有叶子结点的先后顺序完全相同

中序遍历得到的是升序序列

【其他表述】

分支结点：度大于0的结点

叶子结点：度等于0的结点

所以一棵树只有一个结点的时候，根结点就是叶子结点。一棵树有多个结点的时候，根结点就是分支结点！

m叉树可以啥也不是，度为m则至少有一个分支结点有m个孩子！

一棵二叉树的先序遍历序列和其后续遍历序列正好相反，则该二叉树一定是：高度等于其结点树！

链表的性质

线性表的顺序存储结构是一种顺序存储的存储结构，具有**随机存取、顺序存储**的特性；

线性表的链式存储结构是一种链式存储的存储结构，具有**顺序存取、随机存储**的特性；

最适合的链表

这种题最核心的就是**尾删除**，因为需要借用到最后一个元素前面的那个指针，尾指针就不好用了！如果不是双循环统统都是O(N)，一定要用双循环链表。

首查、首删，直接排除不带尾指针的单循环（O(N)保持循环性质）

尾插，直接排除，不带尾指针的单链表、不带尾指针的双链表、单循环链表。

循环链表的操作**注意保持循环的性质。**

能用单循环实现的功能，单循环优先于双循环。

其他表述：用单链表（含有头结点）表示的队列，队头可能在链中（元素超过1个）、链尾（元素仅1个）

图的性质

十字链表存有向图，邻接多重表存无向图

栈和队列的性质及应用

栈的应用：①符号匹配；②系统调用、过程调用；③保护断点、保护现场；④中缀转前后缀

队列的应用：①二叉树的层次遍历；②解决主机和外部设备之间速度不匹配的问题，如缓冲区；③解决由多个用户引起的资源竞争问题，如进程的就绪队列

栈和二叉树的一一对应：

入栈序列当作前序遍历序列，出栈序列当作中序遍历序列。两个序列可以唯一确定一棵二叉树。

一些表述：

越靠后被调用的函数信息越靠近栈顶，某个函数return后该函数在栈中的信息就会消失

消除单向递归和尾递归只需要用到循环就可以了

同一组不重复输入序列执行不同的入、出栈组合操作，所得的结果也可能相同

循环队列判断队空的条件是`rear==front`，判断队满的条件是`(rear+1)%m == front`

中缀转前后缀表达式😈

难点是①后缀的数栈和符号栈和②前缀的求法！王道P98,99两道练习题，吃透即可！

先弹出的是右操作数，后弹出的是左操作数！

后缀表达式从左往右写，机器也是从左往右遍历！前缀表达式从右往左写，机器也是从右往左遍历！

转前缀前缀和执行的过程，数栈和符栈

案例：

```
(6+(3*(7-4)))-8/2        后缀:6374-*+82/-       前缀:-+6*3-74/82
a+b-a*((c+d)/e-f)+g      后缀:ab+acd+e/f-*-g+   前缀:+a-b+*a-/+cdefg
```

数组

稀疏矩阵的存储：三元组或十字链表；

稀疏矩阵压缩存储后就失去了随机存储特性。

三元组的**转置**：三元组数组是按照行优先或者列优先存取的，因此将行标和列标互换后，还需要对整个三元组数组进行重排序

因为需要维护三元组数组的有序性，每一次变化都会导致数组的调整，**三元组每一次调整的时间复杂度都为O(N)**，所以需要非零元个数和位置保持相对的稳定

⚠️易错，在下三角矩阵中，问 `A[i][j](i<j)` 在数组中的坐标，实际上问的是`A[j][i]`

数据结构定义

重点：并查集（后面单独）、二叉排序树、邻接表表示的图、m阶B树定义。其他的参考“数据结构各种定义”

二叉排序树：

```c
typedef struct ThreadNode{
    int data;
    struct ThreadNode* lchild,*rchild;
    int ltag,rtag;      //0是孩子！1是线索！
}ThreadNode,*ThreadTree;
```

邻接表：

```c
#define MaxVertexNum 100;

typedef struct ArcNode{
```

```
    int adjvex;              //边/弧指向哪个节点
    struct ArcNode* next;    //指向下一条弧的指针
    /*int info;              //权值 */
}ArcNode;

typedef struct VNode{
    int data;
    ArcNode* first;
}VNode,AdjList;          //顶点

typedef struct{
    AdjList[MaxVertexNum] vertices;
    int vexnum,arcnum;
}ALGraph;                //邻接表法图
```

m阶B树的结点定义

```
struct Node{
    ElemType keys[m-4];     //最多m-1个关键字
    strcut Node* child[m];  //最多m个孩子
    int num;                //结点中有几个关键字
}
```

并查集及应用 😈

并查集定义及基本操作

```

#define SIZE 13
int UFSets[SIZE];

int Initial(int S[]) {
    for (int i = 0; i < SIZE; i++)
        S[i] = -1;
}

//Find 查操作
int Find(int S[], int x) {
    while (S[x] >= 0)
        x = S[x];
    return x;   //根的S[]小于0
}

//Union 并操作
```

```
void Union(int S[], int root1, int root2) {
    //要求root1与root2不是同一个集合
    if (root1 == root2) return;
    S[root2] = root1;
}

//Union2 优化并操作
//让小树并入大树，根节点的绝对值代表树下有几个结点!
void Union2(int S[], int root1, int root2) {
    //要求root1与root2不是同一个集合
    if (root1 == root2) return;
    if (S[root2] > S[root1]) {  //root2结点数更少
        S[root1] += S[root2];
        S[root2] = root1;
    }
    else
    {
        S[root2] += S[root1];
        S[root1] = root2;
    }
}

//Find2 优化查找操作（压缩路径）
int Find(int S[], int x) {
    //第一次循环找根，第二次循环将路径上各个结点都指向根
    int root = x;
    while (S[root] >= 0)    root = S[root];
    while (x != root) {//压缩路径
        int t = S[x];
        S[x] = root;
        x = t;
    }
    return root;
}
```

并查集的时间复杂度：

初始化：所有数组元素设为-1

Find查（无优化）最坏树高为N，最坏的时间复杂度O(N)

并（无优化）的时间复杂度O(1)

无优化情况下，将n个独立元素通过多次Union合并为一个集合$O(n^2)$

总体复杂度

1️⃣优化合并：小树并入大树，根节点的绝对值（本身是负的）表示这棵树的结点数

优化合并后树的高度不超过O(logN)，故查询的时间复杂度O(logN)

优化合并本身合并时间复杂度O(1)

优化合并后，将n个独立元素通过多次Union合并为一个集合 $O(n\log_2 n)$

2️⃣压缩路径：Find找到根后，把查找路径上的所有结点都指向根结点

压缩路径后能够使得树的高度不超过O(α(n))，在一个很大的范围内，α(n)≤4。

压缩路径后，将n个独立元素通过多次Union合并为一个集合 $O(n\alpha(n))$

总结：n个合并为一个，前面要加n。

并查集的应用：

Kruskal、判环

红黑树的插入、删除😈😈

插入看叔叔，删除看兄弟。考前1天画20分钟过一下就好了

红黑树插入：

插入新结点最初为红色，**如果插入的是根则染黑。**

父亲为红时不满足红黑树性质，需要调整，两种情况：

1️⃣同辈（父和叔）有强有弱则同辈解决（按平衡二叉树规则旋转）

这种情况是父红叔黑：**旋转+染色**。

LL型：右单旋，父换爷+染色

RR型：左单旋，父换爷+染色

LR型：左、右双旋，儿换爷+染色

RL型：右、左双旋，儿换爷+染色

2️⃣同辈势均力敌则交给长辈处理（都是红则把爷爷看作新节点上级去处理）

这种情况，父红叔红：**染色+变新**。

爷爷一定是黑的，叔父变黑，爷变红，向上传递！（如果向上传递，根红了怎么办？再把根染黑！）

总结：不红红，根叶黑，有强弱内部解决，势均力敌给爷爷解决。爷爷需要向上处理时为红，不需要向上处理时为黑。

```
【练习】动手画一画
将1、2、3、4、5、6、7插入到空的红黑树T1中，最终应该有3个红结点
将5、4、3、2、1插入空的红黑树T2中
```

红黑树的删除：

红黑树删除详细图解

如果是末端节点（不算NULL黑叶）：

红色，直接删

黑色，1兄黑，右红侄。红侄变黑，兄父换色，右旋。

2兄黑，左红侄。左红侄右旋，兄侄换色，变成1

3兄红，兄父换色，左旋，变成1或2

如果不是末端节点

只有一个孩子的：替父从军

有两个孩子的：找前驱或后继结点进行交换，从而变成删除末端节点的情况

```
【练习】动手画一画
在上面的T1中，分别删除T6,T3,T4,T2,T1，结果如何？
```

补充：平衡二叉树插入调整

核心是**找最高**，下面为例，应该调整245，而不是243。

1插入相应位置；2从位置往上找最小不平衡子树；3调整这个最小不平衡子树，调整的三个结点是查找路径上的

```
        2
    1       4
          3   5
              6
```

B-树的插入、删除😈

B树的插入😈

1新元素一定是插入到最底层“终端结点”，用“查找”来确定插入位置

2在插入key后，若导致原结点关键字数超过上限，则从中间位置 $\lceil m/2 \rceil$ 将其中的关键字分为两部分，左部分包含的关键字放在原结点中，右部分包含的关键字放到新结点中，中间位置 $\lceil m/2 \rceil$ 的结点插入原结点的父结点。若此时导致其父结点的关键字个数也超过了上限，则继续进行这种分裂操作，直至这个过程传到根结点为止，进而导致B树高度增1。

B树的删除 😈

1️⃣ 若删除结点是终端节点，直接删除（删除前关键字数量 $\geq \lceil m/2 \rceil$）

2️⃣ 若被删除关键字在非终端结点，则用直接前驱（左侧指针最右下元素）或直接后继（右侧指针最左下元素）来代替被删除的关键字

3️⃣ 若删除结点是终端节点，且删除后关键字数量 $< \lceil m/2 \rceil - 1$）

- 兄弟宽裕时，向兄弟借，父亲下来，兄弟上去（保持有序性）
- 兄弟也不宽裕时，父亲关键字+两兄弟合并成一个满的结点

采用不同的合并方法将产生不同的B树。

B树和B+树性质 ⭐ ⭐

m阶B树性质

B树=多路平衡查找树。B树中所有结点的孩子个数的最大值称为B树的阶（最多有几个分叉），通常用m表示。一棵m阶B树或为空树，或为满足如下特性的m叉树：

1️⃣ 每个结点至多有m棵子树，即至多含有m-1个关键字

2️⃣ 若根节点不是终端结点，则至少有两棵子树。

3️⃣ 除了根节点外，任何结点至少有 $\lceil m/2 \rceil$ 个分叉，即至少含有 $\lceil m/2 \rceil - 1$ 个关键字（保证每层关键字不会过少，树不会过于细长）

4️⃣ **所有叶结点都出现在同一层次上，并且不带信息**（可以视为外部结点或类似于折半查找判定树的查找失败结点，实际上这些结点不存在，指向这些结点的指针为空）

失败结点：实际上就是NULL指针

5️⃣ 根结点子树∈[2,m]，关键字数∈[1,m-1]；其他结点子树∈[m/2,m]，关键字数∈[m/2-1,m-1]

6️⃣ 对任一结点，其所有子树高度相同

7️⃣ 关键字的值：子树0<关键字1<子树1<关键字2<子树2....<关键字N<子树N （类似于左根右）

有n个结点的**最小高度**：每个结点m-1个，则有

$n \leq (m-1)(1+m+m^2+\ldots+m^{h-1}) = m^h - 1$，因此有 $h \geq \log_m(n+1)$

有n个结点的**最大高度**：让各层的分叉尽可能的少，根结点只有2个分叉，其他结点只有 $\lceil m/2 \rceil$ 个分叉，各层结点至少有：第一层1 、第二层2 、第三层 2 $\lceil m/2 \rceil$ 、...、第h层 $2(\lceil m/2 \rceil)^{h-2}$

因此第h+1层共有叶子结点（失败结点） $2(\lceil m/2 \rceil)^{k-1}$ 个

n个关键字的B树必有n+1个叶子结点（n个关键字将数域分割成n+1个区间），则 $n+1 \geq 2(\lceil m/2 \rceil)^{h-1}$，即 $h \leq \log_{\lceil m/2 \rceil} \frac{n+1}{2} + 1$

令k=[m/2],h层的m阶B树**至少**包含关键字总数：
$1 + 2(k-1)(k^0 + k^1 + k^2 + \ldots k^{h-2}) = 1 + 2(k^{h-1} - 1)$

综上，含n个关键字的m叉B树，$\lceil \log_m(n+1) \rceil \leq h \leq \lceil \log_{\lceil m/2 \rceil} \frac{n+1}{2} \rceil + 1$

非叶结点的结构

`n|P0|K1|P1|K2|P2|K3|P3|...|Kn|Pn`

其中，K_i 为结点的关键字，且满足K1<K2<K3...。P_i 为指向树根节点的指针，且指针 P_{i-1} 所指子树中所有结点的关键字均小于 K_i，P_i 所指子树中所有结点的关键字均大于 K_i。n为节点中关键字个数，满足

$$\lceil m/2 \rceil - 1 \leq n \leq m - 1$$

m阶B+树性质

需满足以下条件

1. 每个分支结点最多有m棵子树（孩子结点）
2. 非叶根节点至少有两棵子树，其他每个分支结点至少有 $\lceil m/2 \rceil$ 棵子树
3. 结点的子树个数与关键字个数相等！
4. **所有叶结点包含全部关键字及指向相应记录的指针**，叶结点中将关键字按大小顺序排列，并且相邻叶结点按大小顺序相互链接起来

B+树中，无论查找成功与否，最终一定都要走到最下面一层结点

B+树和B树的主要区别：

1. B树：n个关键字，n+1个分叉；B+树：n个关键字，n个分叉
2. B树：根结点关键字数∈[1,m-1]，其他结点∈[[m/2]-1,m-1]

   B+树：根结点关键字数∈[1,m]，其他结点∈[[m/2],m]
3. B+树中，叶结点包含全部关键字；B树中各结点中的关键字是不重复的
4. B+树中，叶结点包含信息，**所有非叶结点仅起到索引作用**，非叶结点中的每个索引项只含有对应子树的最大关键字和指向该子树的指针，不含有该关键字对应记录的存储地址
5. （**超纲**）B+树中，非叶结点不含有该关键字对应记录的存储地址。可以使一个磁盘块可以包含更多个关键字，使得B+树的阶更大，树高更矮，读磁盘次数更少，速度更快。（总之：磁盘方面考虑，B+树比B树快）

在B+树中，有一个指针指向关键字最小的叶子结点，所有叶子结点链接成一个线性链表

多路平衡归并、败者树、选择置换、最佳归并树（类似K叉哈夫曼）😨😨😨

多路平衡归并——用于减少归并趟数

1、N个元素，做K路归并，归并趟数为 $S = \lceil \log_k N \rceil$ 。使用多路归并的目的是减少归并趟数进而减少I/O次数

2、如果不使用败者树，总的关键字比较次数上限为 $S(n-1)(k-1) = \lceil \log_2 N \rceil (n-1)(k-1)/\lceil \log_2 k \rceil$ ，k-1增长得比log2k要快，所以增加的比较次数会抵消减少外存访问次数带来的增益。

3、使用败者树，(k-1)变成log2k，总的关键字比较次数上限为 $\lceil \log_2 N \rceil (n-1)$ ，关键字比较次数上限不变！

4、需要注意，k也不是越大越好，过大会引起读写外存次数增加。

置换选择排序——用于产生更长的（但不等长）的初始归并段

设初始待排文件为FI，初始归并段输出文件为FO，内存工作区为WA，FO和WA的初始状态为空，WA可容纳w个记录。步骤如下：

1️⃣ 从F1输入w个记录到工作区WA

2️⃣ 从WA中选出其中关键字最小的记录，即为MINIMAX

3️⃣ 将MINIMAX记录输出到FO中去

4️⃣ 若FI不空，则从FI输入下一个记录到WA中

5️⃣ 从WA中所有关键字比MINIMAX记录的关键字大的记录中选出最小关键字记录，作为新的MINMAX记录

6️⃣ 重复3️⃣-5️⃣，直到WA中选不出新的MINIMAX记录为止，由此得到一个初始归并段，输出一个归并段的结束标志到FO中去。

7️⃣ 重复2️⃣-6️⃣，直至WA为空。由此得到全部初始归并段。

```
练习FI={17,21,05,44,10,12,56,32,29},WA容量为3
```

最佳归并树——解决不等长归并段归并顺序的问题

类似K叉哈夫曼树，优先归并短的，最后归并长的。

如果 $(n-1)\%(k-1) = 0$，说明经过整数次归并刚好能够将n个归并段归并成1个，不用加虚结点

如果 $(n-1)\%(k-1) \neq 0$，记起为u，要将其凑到刚好够一次归并，所以要补 $(k-1)-u$ 个

进阶代码/语言描述过程（DFS/BFS/Floyd/TopSort）

构造哈夫曼树：

创建一个新结点，在其他所有结点中选择两个根结点权值最小的树或结点作为这个新结点的左右子树，新结点的权值等于两子树权值之和，重复以上步骤直到只剩一棵树

大顶堆、小顶堆的调整过程：

建堆：从第 $\lfloor n/2 \rfloor$ 个孩子开始逐个调整（注意，输出完堆顶元素后是默认要调整的）

输出：将当前元素输出，与未输出的最后一个元素交换，进入【调整】

调整：（大顶堆为例）将当前元素X与左右元素比较，若比其中任意一个小，选择较大的元素进行交换，从而X进入了下一层，继续与新的左右元素进行比较，直到X比它们都大或者X没有孩子。

大根堆：L[i] > L[2i] 且 L[i] ≥ L[2i+1]，即父结点比子结点大

小根堆：即父结点比两个子结点都小

一趟排序：调整成大根堆的样子，并将堆顶和最后一个元素互换，并再调整成（除了原来堆顶的那个元素）大根堆

大根堆最终会形成一个递增的序列，小根堆最终会形成一个递减的序列。

注意，堆从下标1开始，0是空着的（方便实现孩子的寻找）

```
void HeadAdjust(int A[],int k,int len){
    A[0] = A[k];                //A[0]存放被选中的值
    for (int i = 2*k;i<=len;i*=2){
        if (i<len && A[i] < A[i+1]) //选左右子树中较大的一个
            i++;
        if (A[0]>=A[i]) break;      //如果比较大的子树都大，停止循环
        else{
            A[k] = A[i];
            k = i;
        }
    }
    A[k] = A[0];            //被筛选结点的值放入最终位置
}

void BuildMaxHeap(int A[],int len){
    for (int i = len/2;i>0;i--)
        HeadAdjust(A,i,len);
}

void HeapSort(int A[],int len){
    BuildMaxHeap(A,len);        //初始建堆
    for (int i = len;i > 1;i--){
        swap(A[i],A[1]);        //输出堆顶元素
        HeadAdjust(A,1,i-1);    //调整，把剩余的i-1个元素整理成堆
    }
}
```

建堆过程，关键字对比次数不超过4n，时间复杂度O(n) ⚠️因为是自下而上建堆！**可以在线性时间内将一个无序数组建成一个堆**。有可能出选择题哦！

排序过程，时间复杂度为O(nlog2n)，不稳定

插入过程

对于小根堆，新元素放到表尾，然后与其父结点进行比较，如果更小则与父结点进行交换，继而与新的父结点进行交换。否则退出。

删除过程

被删除的元素用堆底元素替代，然后让被换上来的元素不断向下调整，直到无法调整。

注意：下坠过程中如果有一个孩子，要比较关键字一次，如果有两个孩子，要比较关键字两次

深度优先遍历DFS

```
int visit[NumOfVex];
void 题目要求操作();
void DFS(Graph G,int v){
    ArcNode* p = G.vexes[v].firstarc;    //p指向v结点的第一条边
    visit[v] = 1;
    int i;
    题目要求操作();
    while(p != NULL){
        i = p->adjvex;           //i是p这条边指向的结点
        if (visit[i] == 0){      //如果没有访问过i，则对其访问
            DFS(G,i);
        }
        p = p->nextarc;
    }
}
```

广度优先遍历BFS

```
#define GSize G.vexnum
void BFS(Graph G,int v){
    int i,j;
    int visit[GSize] = {0};      //【如果不想重复遍历结点visit写里面，否则visit写在外面】
    Queue Q;    EnQueue(Q,v);    //队列Q用于存放结点
    ArcNode* p;
    while(!isEmpty(Q)){          //如果队列不空就继续循环
        i = DeQueue(Q);
        visit[i] = 1;
        p = G.vexes[i].firstarc;
        while(p != NULL){
            j = p->adjvex;       //j为p这条弧指向的结点
```

```
            if (visit[j] == 0)  //没访问过的结点入队
                EnQueue(Q,j);
            p = p->nextarc;
        }
    }
}
void bfs(Graph G){
    for (int i = 0;i < G.vexnum; i++){
        其他操作...
        BFS(G,i);
        其他操作...
    }

}
```

拓扑排序TopSort

用来找到一个工程中活动完成的顺序，保证每个活动需要进行时前序活动已经完成

1️⃣从AOV网中选择一个没有前驱的顶点并输出

2️⃣从网中删除该顶点和所有以它为起点的有向边

3️⃣重复1️⃣和3️⃣，直到当前的AOV网为空或当前网中不存在无前驱的顶点为止。后一种情况说明有向图中必然存在环

```
bool TopSort(Graph G){
    int i,j,n = 0;  //n对已经输出的结点计数
    ArcNode* p;
    Stack S;        //栈用于保存出度为0的顶点编号(int)
    for (i = 0;i < G.vexnum;i++){
        if (G.vexes[i].count == 0)
            Push(S,i);
    }
    while(!isEmpty(S)){
        i = Pop(S);
        n++;
        cout << i << " ";
        p = G.vexes[i].firstarc;
        while(p != NULL){
            j = p->adjvex;
            G.vexes[j].count--;
            if (G.vexes[j].count == 0)
                Push(S,j);
            p = p->nextarc;
        }
    }
    if (n == G.vexnum)
        return true;
    else
        return false;
```

```
}
```

Prim

```
void Prim(MGraph G, int v0, int &sum){
    int lowcost[maxSize],vset[maxSize],v;
    int i,j,k,min;
    v = v0;
    for(int i = 0;i < G.n; i++){
        lowcost[i] = G.egdes[v0][i];
        vset[i] = 0;
    }
    vset[0] = 1;
    sum = 0;
    for( i = 0; i < G.n; i++){
        min = 0x7FFFFFF;
        for (j = 0; j < G.n; j++){
            if (vset[j] == 0 && lowcost[j] < min){
                min = lowcost[j];
                k = j;
            }
        }//k现在是指向了未连接边最短的未连接结点
        vset[k] = 1;    //将k加入树中
        v = k;
        sum+= min;
        for (j = 0; j < G.n; j++){
            if (vset[j] == 0 && G.edges[v][j] < lowcost[j])
                lowcost[j] = G.edges[v][j];     //更新到未连接结点的最短边
        }
    }
}
```

Kruskal

```
typedef struct{
    int a,b;
    int w;
}Road;
Road road[maxSize];
int v[maxSize];
int getRoot(int a){
    while(a != v[a])    a = v[a];
    return a;
}

void Kruskal(MGraph G, int &sum, Road road[]){
    int i,a,b;
    int N = G.n,    E = G.e;
```

```
    sum = 0;
    for(i = 0;i < N;i++)
        V[i] = i;   //集合初始化
    sort(road,E);   //对road数组中的E条边按其权值大小进行排序
    for (i = 0; i<E; i++){
        ra = getRoot(road[i].a);
        rb = getRoot(road[i].b);
        if (ra != rb){
            v[a] = b;
            sum += road[i].w;
        }
    }
}
```

Floyd（概率不小）

```
void Floyd(Graph G, int A[][maxsize], int path[][maxsize]){
    int i,j,k;
    for (i = 0;i < G.vexnum;i++){
        for (j = 0; j < G.vexnum;j++){
            A[i][j] = G.edges[i][j];
            path[i][j] = -1;
        }
    }
    for (k = 0;k < G.vexnum;k++){
        for (i = 0;i < G.vexnum;i++){
            for (j = 0;j < G.vexnum;j++){
                if (A[i][j] > A[i][k] + A[k][j]){
                    A[i][j] = A[i][k] + A[k][j];
                    path[i][j] = k;
                }
            }
        }
    }
}
```

证明题

证明最小生成树唯一

设所有边的集合为A，将边从大到小排序，由于权值各不相同，排序结果是唯一的。对边从大至小遍历，若在环中，将其去掉，一趟遍历后剩下的结点和边即为最小生成树。由于排序结果唯一，去除的边的集合B也是唯一，剩下边的集合A-B也是唯一的，所以最小生成树唯一。

堆排序

堆排序：先看最终序列，判断用大顶堆还是小顶堆

序列到底是输出后立刻的还是输出后调整的？辩证地看待一下，最好两个都写出来

【2018真题】，堆排序的建堆过程，每次变动一次都写出来

【堆排序插入】插到末尾，然后向上调整

算法性质

拓扑排序只有有向图有，矩阵V2，邻接表V+E；

DJ V2，Prim V2，Floyd V3，Kruskal ElogE

其他都是矩阵V2，邻接表V+E

比较次数和初始序列**无关**：简单选择排序

趟数和初始序列**相关**：冒泡，快速排序

计算机组成原理

冯诺依曼思想

程序和数据以二进制表示

计算机按照程序顺序执行

指令和数据以**同等地位**存储在存储器中，**都通过地址访问**，形式上没有区别

运算器、存储器、控制器、输入设备、输出设备五大部件组成

程序的功能都通过中央处理器实现，数据和指令通过地址访问

冯诺依曼机**工作方式的特点**是：**按地址访问并顺序执行指令**

向后兼容：指的是在老机器上编写的程序可以不加修改的运行在更新的机器上。

冯诺依曼系统以运算器为核心。现代计算机以存储系统为核心。

冯诺依曼机可以区分

概述、性能指标、十进制单位

系统软件：操作系统、数据库**管理系统**、语言处理**程序**、分布式软件系统、网络软件熊、**标准库程序、服务性程序**。

应用程序：**数据库系统**、其他。

```
易错点：数据库系统是应用程序，数据库管理系统是系统程序
语言处理程序、标准库程序、服务性程序都是系统软件
```

标志系统性能最有用的参数是MFLOPS

CPI与时钟周期无关

用户观点看，评价计算机系统性能的综合参数是吞吐率

兼容指计算机软件或硬件的通用性，通常在同一系列不同型号计算机间通用

CPU时钟频率：通常为节拍脉冲或者周期，是主频的倒数。CPU中最小的时间单位

主频：时钟频率的倒数。KMGTPEZ

CPI：执行一条指令所需的时间周期数。

```
某工作站采用时钟频率f为15MHz、处理速率为10MIPS的处理机来执行一个已知混合程序。假定该混合型程序平均每条指令需要1次访存，且每次存储器存取为1周期延迟。
此计算机的有效CPI是15/10=1.5。后面那句为干扰信息。
```

MIPS：每秒执行多少百万条指令

MFLOPS：每秒执行多少百万次浮点运算

IEEE754的一切

1️⃣【**阶数计算**】如果是大于1的阶数，1XXXXX，后面的XXX+1就是阶数；如果是小于1的阶数，0XXXXXX，后面少了几个1，加权相加就是负的多少；特别地，10000000代表1，01111111代表0

2️⃣IEEE754单精度浮点类型能表示的最大正整数?

IEEE阶码偏置值为-127，而表示范围为1到254，因此阶码范围为-126到127。

阶码范围：(0000 0001 ~ 11111110)，负126到正127，可表示的最大整数，阶码取127，后面取(2-2^-23)，即为 $2^{128}-2^{104}$，可表示的最小整数取符号。可表示的绝对值最小的数字， 2^{-126}

阶码为全0时表示正/负0；阶码为全1时表示正/负无穷

3️⃣【**精度损失**】float结构1+8+23，尾数是隐含1位+小数点后23位**实际是24位**。int转float，int的有效位数**超过24位**就会发生精度损失，刚刚好24位不会发生精度损失。换句话说，不损失精度的最大int是 $2^{24}-1$，16777215

4️⃣⚠️n位数值最大可表示 2^n-1，n位的“1000...000”表示的是 2^{n-1}

4、y = x，是否进行符号扩展看的是x的类型而不是y的类型，如果x是int就要符号扩展，如果x是unsigned int就不进行符号扩展。扩展完后怎么解释看的是y的类型，如果y是int最高位就解释为符号，如果y是unsigned最高位就解释为数位。short 变 int要符号扩展，int变short不管有没有符号直接高位截断。

IEEE754标准浮点数进行乘法运算不需要左规处理，但可能要右规处理

IEEE754的舍入方法有：直接截断（最简单）、四舍五入、恒置1

计算标志、计算相关概念😱

带标志加法器是在无符号加法器的基础上增加了一些逻辑门电路而产生额外的功能：

1️⃣ OF（Overflow Flag），表示是否溢出，逻辑表达式为 $OF = C_n \oplus C_{n-1}$ 。**实际含义为最高位的进位和次高位的进位的异或。**在带标志加法器中数值位的FA和符号位的FA是相同的，唯一的区别是「符号位FA」和「数值最高位的FA」输出的C会额外被接入一个异或逻辑门内进行运算。

很好理解，当两个正数相加，0xxx + 0xxx，Cn必定为0，此时若有Cn-1≠0，则OF=1，即溢出；

当两个负数相加，1xxx + 1xxx，Cn必定为1，此时若有Cn-1≠1，则OF=0，即溢出。

可见**带标志加法器默认有符号数是用补码运算的**。因为若是用原码，1001 + 1001 = 0010，不合理

OF用于判断「有符号数」运算的溢出情况，1为溢出，0为无溢出

2️⃣ SF（Sign Flag），即符号位标志，等于FAn输出的S；

3️⃣ ZF（Zero Flag），即是否为0，由于是使用补码运算，因此结果0只有一个表达式 0000，故有 $ZF = \neg(S_1 + S_2 + \ldots + S_n)$

或表示为 $ZF = \neg(F_1 + F_2 + \ldots + F_n)$，注意结果是所有位的结果「先取或」，然后「再取非」。

当所有位全为0时，F1或F2...或Fn=0，ZF=¬0=1；当有至少1个1时，F1或F2或...或Fn=1，ZF=¬1=0；

4️⃣ CF（Carry Flag），进位/借位标志，**逻辑表达式为 $CF = C_{out} \oplus C_{in}$，或者 $CF = C_{out} \oplus Sub$。**
其中 $C_{in} = Sub$，当为加法时sub=0，当为减法时sub=1（取反加1的那个1就是这个Cin）；

CF用于判断「无符号数」运算的溢出情况，1为溢出，0为无溢出。具体意义为：加法时，sub=0，CF=1表示结果超过最大范围，有溢出；减法时，Sub=1，

举例：

- A = 1000，B = 0111，Cin = 1。即表示 1000-0111，将B取反得「B反」1000，$A + B_{反} + C_{in} = 0001$，这种情况下 Cout = 1，Cin = 1，CF = 0，表示无溢出；
- A = 1000，B = 0111，Cin = 0。即表示 1000+0111，$A + B + C_{in} = 1111$，这种情况下Cout = 0，Cin = 0，CF = 0，表示无溢出；
- A = 1000，B = 1111，Cin = 0。即表示 1000+1111，$A + B + C_{in} = 0111$，这种情况下Cout = 1，Cin = 0，CF = 1，表示有溢出（超出最大范围）；
- A = 1000，B = 1111，Cin = 1。即表示 1000-1111，将B取反得「B反」0000，$A + B_{反} + C_{in} = 1001$，这种情况下，Cout = 0，Cin = 1，CF = 1，表示有溢出（有借位，不够减）。

总结：CF用于判断无符号数是否溢出，计算有符号数时CF无用；OF用于判断有符号数是否溢出，计算无符号数时OF无用。ZF对无符号数、有符号数均有用，SF只对有符号数有用。

	无符号数	有符号数	逻辑门

	无符号数	有符号数	逻辑门
OF	/	0代表无溢出，1代表溢出	异或
SF	/	0代表正，1代表负	/
ZF	0代表结果非0，1代表结果为0	0代表结果非0，1代表结果为0	或非
CF	0代表无溢出，1代表溢出	/	异或

模4补码存储时只需要一个符号位，用于检查加减运算中的溢出问题。

算术移位的情况下，双符号位的位移只有较低的那个符号位参与。下面是对双符号位多次左移的例子

```
[11]10000 → [11]00000 → [10]00000（溢出了）→ [10]00000（最高位不参与）
```

规格化浮点数，1.1xx和0.0xx需要左规

对阶不会溢出，小的阶码往大的对（阶码只会变大）

加法器、电路

进位产生函数：$G_i = A_iB_i$

进位传递函数：$P_i = A_i \oplus B_i$

$C_i = G_i + P_iC_{i-1}$，Ci表示第i位数字的进位，C0就是Cin代表输入，1为减法，0为加法

箭头上一个斜杠，写了数字n，代表这根箭头包含了n根线

与门屁股是平的，或门屁股往里凹的，异或门的屁股后面还有额外一条弯线

小圆圈代表取反

列阵乘法器的乘法指令可以在一个时钟周期内完成

存储器🥵🥵

CD-ROM不支持随机存取，只支持串行存取。

CD-ROM是光盘存储器，是一种机械式存储器，**CD-ROM不属于ROM**。

ROM全部支持随机存取方式【✔】，都可以随机访问。但不能说ROM是随机访问存储器，随机访问存储器特指RAM，必须是非易失性的。

DRAM集成度比SRAM高，速度慢，行列共用（地址线少一半，行选、列选多一位），按行刷新（重新写一遍）

SRAM使用双稳态触发器存储数位；DRAM使用电容存储，**DARM内容不会变化，只会漏电消失**。

相联存储器：可以按内容，也可以按地址。

双端口存储器：可以同时访问同一单元。可以同时读，不可以同时写

Cache由硬件实现，Cache缺失不会中断；虚拟存储器由硬件和OS共同完成，缺页会导致中断

虚拟存储器的容量：1️⃣**≤主存和辅存之和**，2️⃣**≤计算机地址的位数能容纳的最大容量。**

替换位数-比较器个数位数

若采用直接映射方式，替换信息位=0bit

若采用全相联方式：若随机替换，0bit；若LRU/FIFO，位数为 $\log_2 总行数$

若采用组相联方式：若随机替换，0bit；若LRU/FIFO，位数为 $\log_2 组内行数$ ，或者说路数

tag位数（标记位数）就是比较器的位数

直接映射，1个比较器，比较器位数 = tag位数 = 块号位数-cache行号位数

全相联映射，个数为cache行数，比较器位数 = tag位数 = 块号位数

N路组相联，个数为N，比较器位数 = tag位数 = 块号位数-cache组号位数

（组相联相当于外层是直接映射，内层是全相联映射）

交换单位😈

Cache和CPU数据交换单位是字

主存和CPU数据交换单位是字

Cache和主存数据交换单位是块

页式和段页式下，主存和虚拟存储器交换单位是页

段氏下，主存和虚拟存储器交换单位是段

大小：块=页框<(虚)页，题目中块一般32B-512B不等，页一般是4KB

地址转换、取数据全过程及缺页和指令Cache缺失处理🏆🏆🏆🏆🏆🏆🏆🏆

要注意区分**转换地址**和**找数据**两个过程

转换地址：虚拟地址VA，先找快表TLB，TLB缺失找主存中的页表，如果页表中对应项为0，则发生缺页异常，需要缺页中断处理，重新执行指令。形成物理地址PA

页表一定包含所有的页信息，只不过有是否在主存中的区别，通过有效位来区分，**地址转换的过程只会发生TLB不命中和缺页，不会发生Cache未命中**

TLB和页表中只保存**「虚拟页号」 到 「物理页框号」的映射**，不保存具体的数据内容

块号只在找数据的过程中使用，地址转换过程没有块号的概念。

缺页中断后需要重新写页表（将有效位改为0）。

```
1 至于缺页中断改不改快表这个问题，2009年真题中的答案是改的，所以默认缺页中断也改TLB
```

TLB中TAG匹配但是有效位为0，表示缺页；TLB中TAG未命中，则可能在页表中

找数据：物理地址PA，先找Cache，若Cache不命中找主存，若主存中一定有对应的页，因为缺页的情况在地址转换的过程中已经解决了。

转换地址TLB缺失页表命中的全过程：

1️⃣访问TLB，未命中

2️⃣访问页表，命中，改TLB

转换地址TLB缺失页表缺页的全过程

1️⃣访问TLB，未命中

2️⃣访问页表，缺页，则启用缺页处理重新执行指令（意味着再访问一次TLB）

指令操作数Cache未命中全过程：

1️⃣访问Cache，未命中

2️⃣访问主存，**一定命中，直接送到CPU，同时送入Cache**（要访问的物理页框所在的页，在地址转换过程已经调入内存）

Cache机制完全由硬件完成，**Cache未命中不会引起中断**。

中断机制由软硬件完成，硬件完成中断隐指令（关中断、保护断点、中断服务程序入口送入PC）

😈**【没见过】取指令Cache缺失的处理过程**：

1️⃣程序计数器恢复当前指令的值（PC-"1"）

2️⃣对主存进行读的操作

3️⃣将读入的指令写入Cache中，更改有效位和标记位

4️⃣重新执行当前指令

【缺页中断处理过程】王道P203

1️⃣中断隐指令（这个不算中断服务程序的内容，由硬件实现）

2️⃣保留CPU现场

3️⃣从外存中找到**所缺页框所在的页**，判断内存是否满，若满则4️⃣，若不满则5️⃣

4️⃣选择一页换出，如果这页没被改过直接覆盖，如果被改过则写回外存。

5️⃣OS命令CPU从外存读取缺页

6️⃣启动I/O硬件

7️⃣将一页从外存换入内存

8️⃣修改页表

寄存器😡

可见的寄存器：PC、PSW、GRs、BR（基址寄存器）

不可见的寄存器：MAR、MDR、IR（三兄弟）、其他判断不出来的多半是不可见

1、CPU中：程序员可见：PC（程序计数器）、PSW（状态字寄存器、中断字寄存器）、GR（通用寄存器组）、BR（基址寄存器）；程序员不可见（透明）：IR、MAR、MDR。PC、PSW、GR是必须的

2、汇编程序员不可的其他：移位器、指令缓冲器、时标发生器、条件寄存器、乘法器、主存地址寄存器

CPU内寄存器的各种功能

程序计数器PC，保存要执行的指令的直接**地址**，位宽与**主存地址总线位宽**相同。

指令寄存器IR（可选），保存当前正在执行的**指令**，位宽与**指令字**相同。

指令译码器，仅对操作码字段进行译码，借以确定指令的操作功能。

存储器地址寄存器MAR（可选），保存CPU访问主存的单元**地址**，位宽与**主存地址总线位宽**相同。

存储器数据寄存器MDR（可选），保存主存中读出的数据或准备写入主存的**数据**，位宽与**存储字长**相同。

时序系统（时序产生器）：用于产生各种时序信号，它们都由统一时钟（CLOCK）分频得到。

操作控制器（微操作信号发生器）：控制的决策机构，其产生的的微操作控制信号序列就是控制流。根据时序调制方法不同，可以分为**硬布线控制器（组合逻辑型）**和**微程序控制器（存储逻辑型）**。

4、程序计数器PC具有寄存信息和计数两种功能

5、寄存器之间的数据传送一般通过CPU内部总线完成

7、CU输入信息三个来源：指令译码器译码产生的指令信息；机器周期信号、节拍信号；执行单元的反馈信息即标志PSW。

8、如果有些微操作所占的时间不长，则应该将它们安排在一个节拍内完成，并且允许这些微操作有先后次序

9、微指令全部存放在控制存储器中（CM），包含两大部分信息：操作控制字段（微操作码）、顺序控制字段（控制产生下一条要执行的微指令的地址）

10、微指令寄存器/微地址寄存器（μMAR/CMAR）：存放由控制存储器读出的一条微指令信息

控制存储器（CM）用来存放全部微程序，其输出的控制信号为Ci指向的门电路。其容量是指令字长×2^N，N必须是整数！如果题目中能得出下地址字段位数，N就按位数。如果没给位数，就按能装下所有指令的最小的N取！

微命令寄存器/控制数据寄存器（CMDR）：保存一条微命令的操作控制字段和判别测试字段信息，即当前要执行的已读出的微指令

地址寄存器MAR：在CPU中的，用于存放要访问的主存单元的地址。其字长和主存地址位数有关（不一定和实际内存大小有关）

地址译码器在内存中。

I/O输入输出、中断

1、通道指令存放在主存里面

2、I/O总线包括：数据线（双向）、设备选择线（单向）、命令线（单向）、状态线（单向）

3、设备选择，设备选择线上的设备码发送到所有**设备接口**的设备选择电路上。当设备选择线上的设备码与本设备码相符合时，应当发出设备选择信号SEL给**接口内的**命令寄存器和命令译码器。（天勤P280）

设备选择电路：用于判断地址总线上呼叫的是不是本设备

4、每个I/O端口有一个地址，而一个I/O接口可能有多个端口，所以一个I/O接口可能有多个地址。

当I/O设备通过接口与主机相连时，CPU可以通过接口地址来访问I/O设备。因为一个接口对应一个I/O设备

5、中断判优使用硬件和软件都可以实现。

6、周期挪用：CPU正在使用内存的时候，等待存储周期结束后总线占有权交给DMA；CPU和DMA同时争夺，则优先给DMA

7、I/O设备每挪用一个主存周期都要申请总线控制权、建立总线控制权和归还总线控制权。

8、DMA传送预处理：

主存起始地址→DMA

设备地址→DMA

传送数据个数→DMA

启动设备

9、中断方式的中断包含数据的传输时间，而DMA的中断仅仅是后处理的时间，并不包含数据传输的时间

10、CPU对DMA请求和中断请求的响应时间不一样。DMA方式下，向CPU请求的是总线控制权，要求CPU让出总线控制权给DMA控制器，所以响应时间小于一个总线周期；中断请求申请的是CPU的时间，所以要等CPU执行指令完毕，所以响应时间小于一个指令周期

11、周期窃取方式下，DMA控制器窃取的是**主存的存取周期。**

12、外部设备的定义：除了主机以外的部分都算外部设备

主机：CPU（控制器+运算器）+内存储器

13、禁止中断的功能可以由中断允许触发器实现。设置中断允许触发器的目的就是来控制是否允许某设备发出中断请求。要区分中断禁止和中断屏蔽的区别，中断禁止是禁止所有，中断屏蔽针对某个中断源

14、在DMA方式中，由**外部设备**向DMA控制器发出**DMA请求信号**，然后由**DMA控制器**向CPU发出**总线请求信号**

15、字符显示器，主机送给显示器的应是**显示字符的ASCII码**，然后由显示器中的字库将其转换成相应字符的字形（模）点阵码

16、外部设备均通过「接口」电路，才能连接到系统总线上

17、中断隐指令：1️⃣关中断；2️⃣保护断点；3️⃣将中断服务程序首地址送PC

单重中断：保护现场；设备服务；**中断事件处理**；开中断；中断返回

多重中断：保护现场；开中断；**中断事件处理**；恢复现场；中断返回

设置屏蔽字的多重中断：保护现场；设置屏蔽字；开中断；**中断事件处理**；关中断；恢复现场；恢复屏蔽字；开中断；中断返回

18、中断方式下数据在程序控制下（软件）进行，DMA方式在硬件控制下进行

19、在程序中断I/O方式中，由CPU和设备的I/O接口直接进行交换，数据直接送到I/O端口，不需要用到主存地址

20、中断：来自CPU指令执行以外的事情发生

异常：CPU执行导致的，如非法操作码，除0，越界，缺页、溢出、陷入

只要记住，异常是指令造成的，不是指令造成的都是中断!

21、I/O接口中数据可以串行也可以并行

22、CPU想要**从用户态切换至内核态**，需要由用户程序触发中断或异常后，由**硬件实现**将PSW标志位设置为1后，就会切换到内核态。

CPU想要**从内核态切换至用户态**，由**内核程序调用一条修改PSW的特权指令**即可。

23、保护断点中的断点是响应中断时程序计数器PC的值和程序状态字PSW的值，以便中断处理完毕返回时还能按照原来的顺序执行程序；保护现场是保存当前用户态的上下文信息（包括通用寄存器，PC，PSW，栈指针等），这个现场是被中断的原程序在断点处各个寄存器的值（当然也包括PC）。

多处理器和硬件多线程

```
硬件多线程：为每个线程准备了单独的寄存器、PC。只是将传统多线程的实现过程硬件化了。
SIMD是1CPU+N存储器
MIMD是NCPU+1存储器，等于多核
向量处理器：含有直接处理多条数据指令集的CPU，软件实现同时处理多数据，属于SIMD
超标量处理器：同指令，处理多条数据，硬件实现同时处理多数据，属于SIMD

传统多线程是1CPU+1组(通用处理器+PC)+?存储器，不等于多核
硬件多线程是1CPU+N组(通用处理器+PC)+?存储器，不等于多核

细粒度多线程：每个时钟周期切换
粗粒度多线程：较大阻塞时切换
同时多线程SMT：Simultaneous Multi-Threads。多个指令流+多线程，等于多核
共享存储多处理器SMP: Shared Memory Processors。共享变量通信、共享存储地址，多核

UMA，Unified Memory Access。统一存储访问。访问内存中各个地址的时间差不多
NUMA，Non-unified Memory Access。非统一存储访问。内存被分割成不同部分给多核，产生了本地和远程的区别。处理器访问自己的(本地)内存快一些，别人的(远程)慢一些
NC-NUMA，不带高速缓存的NUMA；CC-NUMA，带高速缓存的NUMA
多处理器的冲突无法通过关中断实现，因为关中断只对一个核心有用。得通过对共享变量加锁的方式处理。
```

具体内容如下

常规的多处理器属于MIMD，常规的单处理器属于SISD，不存在MISD

1、SISD：单指令流单数据流：**串行**计算机结构，这种计算机通常仅包含一个处理器和一个存储器，处理器在一段时间内仅执行一条指令，按指令流规定的顺序串行执行指令流中的若干条指令。为了提高速度，**有些SISD计算机采用流水线的方式**（SISD可以使用流水线），因此SISD处理器有时会设置多个功能部件，并且采用多模块交叉方式组织存储器。

```
总结：SISD也可以使用流水线
```

⚠️冯诺依曼机器属于典型的SISD机器。

2、SIMD：单指令流多数据流：一个指令流同时对多个数据流进行处理，一般称为**数据级并行技术**。这种结构的计算机通常由**一个指令控制部件、多个处理单元和多个存储器**组成。**每个处理单元执行的都是同一条指令，但是每个单元都有自己的地址寄存器，这样每个单元就都有不同的数据地址**，因此，不同处理单元执行的同一条指令所处理的数据是不同的。

⚠️一个顺序应用程序被编译后，可能按SISD组织并运行于串行硬件上，也可能按SIMD组织并运行于并行硬件上。

⚠️SIMD在**使用for循环处理数组时最有效**，比如，一条分别对16对数据进行运算的SIMD指令如果在16个ALU中同时运算，则只需要一次运算时间就能完成运算。

⚠️SIMD在**使用case或switch语句时效率最低**，此时每个执行单元必须根据不同的数据执行不同的操作。

```
总结：SIMD，同一指令，多个数据。一个指令控制部件+多个处理单元、存储器
```

3、MISD：多指令流单数据流：同时执行多条指令，共同处理一个数据，**实际上不存在**这样的计算机。

4、MIMD：多指令流多数据流：同时执行多条指令分别处理多个不同的数据。**是一种并行程度更高的线程级并行或线程级以上并行计算模式**。分为以下两种！

多计算机系统：**由多台计算机组成**，每个计算机节点都具有各自的私有存储器，并且具有独立的主存地址空间，不能通过存取指令来访问不同节点的私有存储器，而要通过消息传递进行数据传递，**也称消息传递MIMD。**

多处理器系统：也称共享存储多处理器（SMP）系统，具有**共享的单一地址空间**，每个处理器都可通过**存取指令（LOAD,STORE）访问系统中的所有存储器，也称共享存储MIMD。**

```
SMP:Shared Memory Processor 共享存储处理器
共享单一存储空间，每个存储器可以访问系统中所有存储器
```

5、向量处理器是SIMD的变体，是一种实现了直接操作一维数组（向量）指令集的CPU，而串行处理器只能处理单一数据集。其基本理念是将从存储器中收集的一组数据按顺序放到一组向量寄存器中，然后以流水化的方式对它们依次操作，最后将结果写回寄存器。**向量处理器在特定工作环境比如数值模拟或者相似的领域中极大提升了性能。**

```
SIMD数据级并行。MIMD是一种并行程度更高的线程级并行或线程级以上并行计算模式。
```

6、为了避免线程频繁切换引起的开销，有了硬件多线程。**硬件多线程可以减少线程切换过程中的开销。**

7、支持硬件多线程的CPU中，必须为每个线程提供单独的通用寄存器组、**单独的程序计数器**等，**线程的切换只需激活选中的寄存器**，从而省略了与存储器数据交换的环节，大大减少了线程切换的开销。**硬件多线程允许多个线程以重叠的方式共享一个处理器的功能单元，硬件多线程≠多核**，这样可以有效地利用硬件资源；硬件多线程中硬件必须具有以相对较快的速度切换进程的能力，在线程阻塞时处理器可以切换到另一个线程进行运行。

8、细粒度多线程（轮流交叉）：多个线程之间轮流交叉执行指令，多个线程之间的指令是不相关的，**可以乱序并行执行。**在这种方式下，**处理器能在每个时钟周期切换线程**。细粒度多线程的**主要缺点是会减慢单个线程的执行速度**,因为准备好执行而不会停顿的线程将被其他线程的指令所延迟。

9、粗粒度多线程：仅在一个线程出现了较大开销的阻塞时，才切换线程，如Cache缺失。在这种方式下，**当发生流水线阻塞时，必须清除被阻塞的流水线，新线程的指令开始执行前需要重载流水线，因此线程切换的开销比细粒度多线程更大。**

10、同时多线程（SMT）：是细粒度多线程和粗粒度多线程的变体。它在实现指令级并行的同时，实现线程级并行，也就是说，它**在同一个时钟周期中，发射多个不同线程中的不同指令。**同时多线程**利用多发射和动态调度微体系结构中的资源实现，降低了多线程的开销。**　　王道P267

SMT技术是超标量技术与多线程技术的充分协同与结合（百度百科）

11、多核处理器：多个处理单元集成到单个CPU中，每个处理单元称为一个核。每个核**可以有自己的cache，也可以共享一个cache。**所有核一般是**对称**的，并且**共享主存储器**，因此多核属于共享存储的对称多处理器。

12、共享内存多处理器（SMP，Shared Memory Processor）：具有共享单一物理地址空间的多处理器。

⚠️**通过存储器中的共享变量**相互通信

⚠️**所有**处理器都能通过**存取指令**访问**任何**存储器的位置。

⚠️即使这些系统共享同一个物理地址空间，它们**仍然可以在自己的虚拟地址空间中单独地运行程序。**

UMA：统一存储访问多处理器：**所有处理器访问时间大致相同**，根据处理器与共享存储器之间的连接方式，可分为**基于总线、基于交叉开关网络和基于多级交换网络连接**等

NUMA：非统一存储访问多处理器：某些访存请求要比其他的快，**主存被分割并分配给了同一机器上的不同处理器或内存控制器**。处理器中**不带**高速缓存时，被称为**NC**-NUMA；有一致性高速缓存时称为CC-NUMA。

⚠️**NUMA架构下**，内存的访问出现了本地和远程的区别，**访问本地内存明显快于访问远程内存。**

13、多处理器冲突：同时访问统一共享变量，需要通过对共享变量加锁的方式来控制对共享变量的互斥访问。

C语言内存分配

1、每个进程的虚拟空间都分为系统区和用户区，用户区是各自的可以随意访问，系统区是共享的不可以随意访问

操作系统内核区

用户栈（运行时创建）（向下👇）：局部变量

共享库的存储映射区

动态生成的堆（向上👆）：　　malloc或new申请

读/写数据段：　　存放全局变量和静态变量

只读代码段：　　存放常量

2、非静态局部变量可以和全局变量同名！因为它们被分配在不同存储区中

```
int a = 1;
int main(){
    int a = 2;
}//这种情况是允许的
```

总线事务

从请求总线到完成总线使用的操作序列称为总线事务，它是在一个总线周期中发生的一系列活动。典型的总线事务包括请求操作、仲裁操作、地址传输、数据传输和总线释放。

请求阶段：主设备（CPU或DMA）发出总线传输请求，并且获得总线控制权。

仲裁阶段：总线仲裁机构决定将**下一个传输周期的总线使用权**授予某个申请者。

寻址阶段：主设备通过总线给出要访问的从设备地址及有关命令，启动从模块(从设备)。

传输阶段：**主模块(即主设备)和从模块(即从设备)**进行数据交换，可**单向或双向**进行数据传送。

释放阶段：主模块的有关信息均从系统总线上撤除，让出总线使用权。

在总线事务的传输阶段，主、从设备之间**一般只能传输一个字长的数据**。

突发（猝发）传送方式：寻址阶段发送的是**连续数据单元的首地址**。在传输阶段传送多个连续单元的数据，**每个时钟周期可以传送一个字长的信息**，但**不释放总线**，直到一组数据全部传送完毕后，再释放总线。

```
总线仲裁：是下一个周期的总线使用权
传输阶段：主和从设备之间，单向或双向
一个时间周期传送一个总线字长
突发传送指传完之前不释放；非突发指每传一个字就释放一次
取数据一般要时间。连续的取和连续的发送，默认按找流水线方式来：取一个发一个，取和发并行执行。类似计网那种。
```

通道工作方式

1️⃣编制通道程序

2️⃣启动I/O通道

3️⃣组织I/O操作

4️⃣向CPU发出中断请求

单CPU单通道下，结构是CPU——通道——设备。连在一起的不能完全并行。程序和设备可以完全并行

操作系统

操作系统功能 💩

并发和共享是两个最基本特征

1、内存管理功能：内存分配与回收、地址转换、内存空间的（逻辑）扩充、内存共享，存储保护

2、**文件系统功能**：实现“按名存取”，对文件存储空间的管理、对文件目录的管理、用于**将文件的逻辑地址转换为物理地址**的机制、对文件读和写的管理以及对文件的共享与保护等功能。

3、设备管理功能：设备分配、设备处理、缓冲管理、设备独立性

微内核结构 💩

微内核结构通常利用“机制与策略分离”的原理来构造OS结构，将机制部分以及硬件紧密相关的部分放入微内核。

进程线程管理、低级存储器管理、中断和陷入处理、客户和服务器之间的通信

主要问题是性能问题，切换状态开销大。

扩展性和灵活性、可靠性和安全性、可移植性、分布式计算

虚拟机 💩

逻辑上的计算机，**虚拟机作为用户的一个进程运行**，虚拟机上的操作系统认为自己运行在内核态，实际上不是，是“虚拟内核态”。

运行在两类虚拟机管理程序上的操作系统都称为**客户操作系统。**对于第二类虚拟机管理程序，运行在底层硬件上的操作系统称为**宿主操作系统。**

操作系统引导 😈 😈 😈

1️⃣ 激活CPU读取BIOS ROM中的boot程序即【**自举程序**】，将指令寄存器设置为BIOS的第一条指令

```
1 自举是指计算机系统自动完成启动，它包括硬件和软件准备，即“加电自检”和“磁盘引导”。自检是自举程序的一部分，所以自检程序也视为自举程序
```

2️⃣ 执行【**自检**】（自检是自举程序的一部分）：启动BIOS后，进行硬件自检，检查硬件是否出现故障

3️⃣ 加载硬盘

4️⃣ 加载**硬盘的【第一个扇区】主引导记录MBR**：主引导记录MBR的作用是告诉CPU去硬盘的哪个主分区找操作系统。**MBR包含：【引导装入程序（引导代码）】、一个磁盘分区表、一个标志**（指示从哪个分区引导系统）

5️⃣ 扫描硬盘分区表，找到**装有操作系统的【活动分区】**：主引导记录扫描硬盘分区表，识别含有操作系统的硬盘分区（活动分区）。硬盘分区以特定的标识符区分活动分区和非活动分区。

6️⃣ 加载**活动分区的【第一个扇区】分区引导记录PBR。**分区引导记录PBR中包含用来寻找并激活引导操作系统的程序【**引导程序**】，即启动管理器

7️⃣ 加载【**引导程序**】，即启动管理器

8️⃣ 加载【**操作系统**】。

BIOS ROM[自举程序] → MBR[引导装入程序] → PBR[引导程序] → 操作系统

【位于整个磁盘第一个扇区】【活动分区的第一个扇区】

磁盘格式化、地址、RAID 😈

格式化过程：

1️⃣ 低级格式化/物理格式化：划分扇区；每个扇区的数据结构、头部尾部填充。
2️⃣ 磁盘分区：分为C\D\E\F盘等，**每个分区的起始扇区和大小都记录在主引导记录MBR的分区表**中
3️⃣ 逻辑格式化（**创建文件系统**）：操作系统将初始的文件系统数据结构存储到磁盘上，这些数据结构包括空闲空间和已分配的空间以及一个初始为空的目录（即**空闲分区表、目录文件**）。

```
重点：物理--分区--逻辑
分区的依据在主引导记录MBR中
空闲分区表、目录、文件系统这种都是在逻辑格式化完成的
```

文件的**最小存储分配单位是簇**。

磁盘地址格式：柱面号、盘面号、扇区号

RAID0：交叉编址，可以让多个磁盘并行工作、加快数据的输入输出

RAID1：镜像冗余

RAID2~3：海明、奇偶

RAID4：独立校验盘，磨损最快

RAID5：校验数据循环存放在每个磁盘中

临界区性质判定😈

一般这类题会问互斥性和饥饿性，做题步骤：

0️⃣如果题干中出现了「i，i=0,1」这个恶心的玩意，一定把它写在草稿纸上。（**慢就是快**）

1️⃣先找会导致无法互斥的情况，这个比较好找

2️⃣一般指令数较少，穷举尽可能多的情况，如果都无法造成饥饿或死锁。那就认为不会。

产生饥饿的主要原因是：在一个动态系统中，资源分配策略是倾斜的，即使没有发生死锁也无法判断等待时间的上界。如果策略没有倾斜，两个进程只有苛刻地满足某个特定序列才进不去临界区时，不认为是饥饿。

文件逻辑结构和文件物理结构😈😈😈

顺序存取：磁带

直接存取：磁盘、光盘。存放在直接存取存储器上面的文件也能顺序访问，但一般效率较差

随机存取：ROM、RAM、Flash、固态硬盘

顺序文件：串结构（时间顺序）；顺序结构（关键字顺序）

磁带上的顺序文件添加新纪录时，直接加在最后就行了，不用复制整个文件

索引文件：索引表直接定位，不会放在磁带上

索引顺序文件：有点像分组查找，组间有序，组内无序。既能顺序访问，也能随机访问。不会放在磁带上。

散列文件：通过键值直接决定物理地址

连续分配：占有连续的块

隐式链接：单链表，因为没有一张表指出哪里是哪里，所以是隐式。**注意计算最大文件的时候要考虑指针的大小。**

显式链接：FAT在系统启动时就会被读入内存

索引、混合索引分配：iNode，UNIX默认采用混合索引分配

Windows、Mac、Linux都用的是树状结构目录

管程

管程是进程**同步工具**，其特性**保证了进程互斥（每次仅允许一个进程进入管程）**，不需要程序员自己实现。

管程定义了一个数据结构和能为并发进程所执行（在该数据结构上）的一组操作，这组操作能同步进程和改变管程中的数据。（像类）

在管程中设置了多个条件变量，每个条件变量只能进行两种操作，即wait和signal

x.wait：当x对应的条件不满足时，正在调用管程的进程调用x.wait将自己插入x条件的等待队列，并释放管程。此时其他进程可以使用该管程。

x.signal：x对应的条件发生了变化，则调x.signal，唤醒一个因x条件而阻塞的进程。

```
1 这两个命令一定是阻塞和唤醒，没有值的判断过程。值的判断过程是写在前面的if
```

内存映射文件

内存映射文件所进行的**任何实际交互都是在内存中进行的**，以标准内存地址形式来访问。磁盘的周期性分页是由操作系统在后台隐蔽实现的，对应用程序而言是完全透明的。虚拟存储器以统一的方式处理所有磁盘I/O。当进程退出或显式地解除文件映射时，所有被改动页面会被写回磁盘文件。

用户态(不能干的)、核心态😈

用户态又称目态。 核心态又称管态、系统态、内核态

CPU想要**从用户态切换至内核态**，需要由用户程序触发中断或异常后，由**硬件实现**将PSW标志位设置为1后，就会切换到内核态。CPU想要**从内核态切换至用户态**，由**内核程序调用一条修改PSW的特权指令**即可。

```
1 即从核心态到用户态的转换是由操作系统程序执行完成的，而用户态到核心态的转换则是由硬件完成的。
```

由用户态转向核心态的例子：①系统调用；②中断（处理）；③用户程序产生了错误状态；④用户程序企图执行特权指令；⑤访管/自陷trap

由用户态进入核心态，不仅状态需要切换，而且**所用的堆栈也【可能】需要由用户堆栈切换为系统堆栈，但这个系统堆栈也是属于该进程的**。

核心态下，可以使用除了访管指令以外的所有指令。

用户态下可执行

通用寄存器清零

核心态专属

设置定时器的初值、关中断、内存单元复位、进程切换、输入输出、系统调用过程、执行I/O指令。看上去和底层硬件很紧密的或者很像操作系统干的事，用户都不可以干。

```
区分用户态和核心态：有两种方法：第一种是看指令的频率，因为从用户态切换到核心态需要大量的时间，所以只能在核心态运行指令很少，算术运算、从内存取数、把结果送入内存在每条指令中都可能出现多次，所以是用户态指令，而输入/输出指令频率较低，更有可能是只能在核心态运行；
第二种是看指令对系统的影响，算术运算、从内存取数、把结果送入内存一般只会影响到计算的结果，而输入/输出指令需要使用I/O设备，涉及资源使用，有可能影响到其他进程及危害计算机，所以不能在用户态。
```

透明

是否涉及操作系统：

CPU核心态切换为用户态：【✔】操作系统执行程序（一条特权指令）完成

CPU用户态切换为核心态：【⚠❌】硬件自动完成

页面置换：【✔】操作系统可以置换页面，**页面对操作系统不透明**，对用户透明！

周期性分页：【✔】操作系统完成，对应用程序和用户透明

分配页框：【✔】在页式系统中进程建立时,操作系统为进程中所有的页分配页框。当进程撤销时收回所有分配给它的页框。

内存保护：【✔】内存保护由操作系统和硬件机构合作完成（重定位寄存器），**需要硬件支持。**

进程/线程调度：【✔】进程调度和线程调度是操作系统的主要功能之一，**纯软件，不需要硬件支持**。

地址重定位（地址映射）：【✔】**需要硬件支持**（重定位寄存器）。

时钟管理：【✔】**需要硬件支持。**

中断系统：【✔】**需要硬件支持。**包括下面的内容

缺页处理：【✔】**需要硬件支持**，缺页属于中断的一种。

保护**现场**：【✔】**需要硬件支持**，操作系统和硬件共同完成。

SPOOLing技术，输出井/输入井和设备之间数据传送：【✔】由操作系统完成

中断响应、保护断点：【✖】硬件自动完成

越界错误：【⚠✖】如果调页时逻辑地址中的页号超过页表的范围会产生越界中断，越界错误处理过程由硬件完成。并不是操作*系统*的任务。

对用户的（不）透明：

段对用户不透明，页对用户透明

各种单位

在消息传递系统中，进程间**以消息为单位**交换数据

对主存的访问以字节或字为单位；**内存和外存**间的传送以磁盘块为单位；对磁盘的访问以磁盘块为单位

CPU和Cache之间以字为传输单位；Cache与主存之间以**内存块（物理块/页框）**为单位。

对主存储器的访问，以字节或字为单位；对主存储器的分配，以块为单位

逻辑记录是对文件进行存取操作的基本单位

文件存储空间的分配常以块或簇为分配单位

页是存放信息的物理单位，段是信息的逻辑单位。

各种表和寄存器

页式：每个进程一张页表，驻留在内存中

段式：每个进程一张段表；

段页式：每个进程一张段表，每个分段一张页表！

中断向量表：在操作系统进行初始化过程中创建。

重定位寄存器：整个系统有一个，加载时要使用特权指令。

界地址寄存器：含逻辑地址的最大值，加载时要使用特权指令。

页表基址寄存器：存放的是顶级页表的起始物理地址。当一个进程不运行时，其页表始址会被操作系统保存到PCB中。当再被调用的时候，从PCB中找出页表始址写入寄存器中。

控制块 描述符 上下文 现场⭐

进程控制块PCB

包含进程描述信息、进程控制信息和管理信息、资源分配清单、处理及相关信息。（进程存在的唯一标志！）

```
进程控制块的具体内容：
1️⃣进程描述信息：进程标识符PID、用户标识符UID
2️⃣进程控制信息和管理信息：进程当前状态、进程优先级、代码运行入口地址、程序的外存地址、进入内存时间、处理机占用时间、信号量使用、进程队列指针
3️⃣资源分配清单：代码段指针、数据段指针、堆栈段指针、文件描述符、键盘、鼠标
4️⃣处理机相关信息：通用寄存器值、地址寄存器值、控制寄存器值、标志寄存器值
```

操作系统对进程进行管理工作所需要的信息都保存在PCB中。

一个进程实体（进程映像）由PCB、程序段、数据段组成，进程是动态的，进程实体（映像）是静态的。

线程控制块TCB

TCB包括以下内容：1️⃣线程标识符 2️⃣一组寄存器 3️⃣线程运行状态 4️⃣优先级 5️⃣线程专有存储区 6️⃣堆栈指针

同一进程中的所有线程都完全共享进程的地址空间和全局变量。各个线程都可以访问进程地址空间的每个单元，所以一个线程可以读、写或甚至清除另一个线程的堆栈。但是一个另一个线程的栈指针对该线程是透明的。

文件控制块FCB

实际上就是目录项，用来存放控制文件需要的各种信息的数据结构。

主要包含：1️⃣基本信息（文件名、物理位置、逻辑结构、物理结构）、2️⃣存取控制信息（文件主/核准用户/一般用户的存取权限）、3️⃣使用信息（建立时间、上次修改时间）

索引结点inode

分为磁盘索引结点和内存索引结点，但不分那么细了。不包括文件名

主要包括：1️⃣文件主标识符 2️⃣文件类型 3️⃣存取权限 4️⃣物理地址 5️⃣文件长度 6️⃣存取时间 7️⃣链接计数

（如果被调入内存，还有）索引结点编号、状态、访问计数、逻辑设备号、链接指针

打开文件表

注意和 文件描述符/文件句柄 区分开。

文件描述符指向进程的打开文件表，进程的打开文件表指向系统打开文件表，系统打开文件表指向文件。

包括：1️⃣文件指针 2️⃣文件打开计数 3️⃣文件磁盘位置 4️⃣访问权限

进程上下文

进程上下文是指进程切换时需要保持的进程状态包括 1 寄存器值、 2 **用户和系统内核栈状态**。

补充：进程上下文包括三个，用户级上下文，寄存器上下文和系统级上下文
用户级上下文：指令，数据，共享内存、用户栈
寄存器上下文：PC，通用寄存器，控制寄存器，状态字寄存器，栈指针（用来指向用户栈或者内存栈）
系统级上下文：PCB，主存管理信息（页表&段表）、核心栈

```
1 如果是单选，优先选寄存器值和内核栈状态。如果是多选，PCB等也可以算
```

虚拟文件系统VFS对象

超级块对象：表示一个已安装（或者叫挂载）的特定文件系统

索引结点对象：表示一个特定的文件

目录项对象：表示一个特定的目录项

文件对象：表示一个与进程相关的已打开文件

现场和断点

保护现场是指将现场信息（PC+PSW+GRs+栈指针）**保存至进程的PCB**中。

保护断点：将CS（代码段寄存器），IP指针（指令指针寄存器）压入堆栈中，其中IP也叫PC。

xxx的影响因素😈😈😈

缺页率的影响因素：

【✔】页面置换算法（显然）

【✔】驻留集大小/分配物理块数量（越大缺页率越低）

【✔】工作集大小（工作集越大，访问的不同页面越多，越可能缺页）（驻留集是分配给进程的物理块数 工作集是某段时间内进程要访问的页面的集合）

```
1 工作集：工作集窗口大小w，工作集大小≤w
2 驻留集：分配给进程的物理块数，一般来说要大于工作集大小
```

【✔】进程的数量（进程越多，每个进程可分配的越少）

【✔】程序的编制方法

【✖】页缓冲队列的长度（能改变页面置换的速度，减小页面置换开销，但是不会影响缺页率）

Cache命中率的影响因素：

计算机网络

一些概念

复用：信道的复用。

分用（Demultiplexing）是指将经复用所形成的合成信号**恢复为**多个原独立信号，或恢复为由这些独立信号所组成的信号群的处理过程。**分用是在接收端进行的**，接收方的传输层剥去报文首部后，能把这些数据正确交付到目的进程，需要用到源端口号。

WWW高速缓存：WWW高速缓存将最近的一些请求和响应暂存在磁盘中，当与暂存的请求相同的新请求到达时，WWW高速缓存就将暂存的响应发送出去从而降低了广域网的带宽。

传播时延：信道上从一端到另一端的时延

传输时延：在发送端把数据完全地发送到信道上的时延

可靠服务：当网络是可靠的时候，因为检错、纠错、应答机制的存在，所以一定能保证数据最后准确传输到目的地

不可靠服务：不可靠服务是出于速度、成本等原因的考虑，而忽略了网络本身的数据传输的保证机制，但可以通过应用或用户判断数据的准确性，再通知发送方采取措施，把不可靠的服务变为可靠的服务

服务访问点SAP：上层协议实体和下层协议实体之间的接口
链路层的SAP是MAC地址
网络层的SAP是IP地址
传输层的SAP是端口

在客户/服务器模型中，默认端口号通常是指服务器端，而客户端的端口号通常都是动态分配。

SDN

控制平面从路由器物理上分离。路由器仅实现转发，远程控制器计算和分发换发表以供每台路由器所使用。路由器通过交换「包含转发表和其他路由选择信息」的报文与远程控制器通信。因为计算转发并与路由器交互的控制器是用软件实现的。

SDN分为三个层次：

👆SDN控制器通过**"北向接口"**与网络控制应用程序交互。该API允许网络控制应用程序在状态管理层之间读写网络状态。

👉由SDN控制平面做出的最终控制决定，将要求控制器具有有关网络的主机、链路等最新状态信息。控制器之间的是东向接口、西向接口

👇SDN控制器通过**"南向接口"**与受控网络设备（路由器）之间的通信（OpenFlow协议）

协议大全😇😇😇

PPP：透明传输（0比特填充）、差错检测但不纠错CRC（错就丢弃，硬件实现）、简单无序号和确认机制不可靠。

路由表写法

注意了！！！如果给的是**分类地址**，要有（目的网络地址，子网掩码，下一跳地址，端口）。而且不能用后缀来表示。

如果给的是CIDR形式，要有（目的网络地址，下一跳地址，端口）

特别地：Internet 0.0.0.0　后缀是/0，或者子网掩码为0.0.0.0

要记的数字

1、熟知端口号：0-1023；登记(IANA)端口号1024-49151；短暂端口号49152-65535

2、OSPF：10s发一次HELLO，30分钟更新一次链路状态

3、RIP：30s向邻居发送一次；180s没收到邻居视为不可达；≤15可达，=16不可达

4、端口号和协议：

FTP数	FTP控	TELNET	STMP	DNS	FTFP	HTTP	POP3	SNMP
TCP	TCP	TCP	TCP	UDP	UDP	TCP	TCP	UDP
20	21	23	25	53	69	80	110	161

帧格式、首部😈

1、MAC：MAC地址6B，首部14B=6+6+2服务，尾部FCS 4B
VLAN-MAC：目的6+源6+VLAN标签4+服务2，尾部FCS 4B。
VLAN标签包括2B标签+4b无效+12bVID，VID有效的取值范围为0-4095
802.11帧：四个地址，这样记："收"+"发"，收在前，地址三是另一个地址；WDS类型是收发收发
PPP帧：F(0111 1110) A(FF) C(03) 协议(2字节) 信息字段（不超过1500B）FCS 2B F(0111 1110)

组播用的是UDP，因为是一对多。

```
MAC目的、源MAC地址每过一个路由器就要变化
IP地址在经过网关的时候由NAT改变
```

2、IPv4：IP地址32位，1总8片首4，注意分割的数据部分需要是8的整数倍（除了最后一片）

3、IPv6：IP地址128位，首部固定40B，没有补充，**没有校验和。**

4、UDP：端口号2B，首部8B，首部长度包括首部+数据部分，伪首部校验和（校验所有）

5、TCP：端口号2B，首部20B，数据偏移4B为单位，实际上就是首部长度，伪首部校验和（校验所有）

特殊地址

MAC地址目的MAC在前面，源MAC地质在后面。MAC地址每经过一个路由器，其目的和源都会改变。直通式交换机只看目的MAC；补充：交换机流量：双端口：2N×V；半双工：N×V

ARP请求MAC帧，目的地址为ff-ff-ff-ff-ff-ff

DHCP发现：客户源地址0.0.0.0，目的地址255.255.255.255
DHCP提供：服务器地址xxxx，目的地址255.255.255.255
DHCP请求：客户源地址0.0.0.0，目的地址255.255.255.255
DHCP确认：服务器地址xxxx，目的地址255.255.255.255

A类可用地址：1-126；10是私有，127环回
B类可用地址：128-191；172.16-31是私有
C类可用地址：192-223；192.168是私有，注意224是D类了
D类（组播）：224-239；不可作为源地址，注意边界是224-239，240是E类了
E类（不可用）：240-255；
其他陷阱：数字超过255，如256。这也是不可能的网络号

网络号	主机号	可作源地址	可作目的地址	用途
全0	全0	✓	✗	表示本网范围本主机，在路由表中是Internet
全1	全1	✗	✓	本网广播
全0	特定值	✗	✓	表示本网某个主机
特定值	全0	✗	✗	表示某个网络，只出现在路由表中
特定值	全1	✗	✓	向某个网络广播
127	非全0/全1	✓	✓	本地环回测试
D类		✗	✓	组播地址

三类本地互联网地址：10、172.16-31、192.168

本网广播：最好写255.255.255.255，也可是本网网络号+全1

TCP握手

主要是记各阶段名字：

CLOSED、SYN-SENT、ESTABLISHED

CLOSED、LISTEN、SYN-RCVD、ESTABLISHED

ESTABLISHED、FIN-WAIT1、FIN-WAIT2、TIME-WAIT（等待2MSL）、CLOSED

ESTABLISHED、CLOSE-WAIT、LASK-CHECK、CLOSED

HTTP1.1传文件🚀

注意题干信息是否已经建立连接!

建立连接算1个RTT（因为第三次握手已经开始传数据了）

如果是小文件，一个RTT传一个（2011大题）；

如果是大文件，拆成多个MSS，每个大文件按照TCP拥塞控制里那种方法发送（2022第40题）

考前抱佛脚🙏🙏🙏

定点数的乘法⭐⭐

乘法运算由「**累加**」和「**右移**」操作实现，可分为原码一位乘法和补码一位乘法。

无符号数乘法

原理和「原码一位乘法」是一模一样的，因为原码一位乘法的底层就是进行无符号数乘法。

原码一位乘法

原码一位乘法的特点是「**符号位与数值位是分开求的**」，乘积符号取两个操作数符号**异或**得到

运算规则：

1️⃣ 被乘数和乘数均取绝对值参加运算，**看作无符号数。**符号位单独处理 $S_{x\times y}=x_s\oplus y_s$

2️⃣ 部分积是乘法过程中的中间结果。乘数的每一位 y_i 乘以被乘数得 $X\times y_i$ 后与之前的部分积累加，部分积初值为0

3️⃣ 从乘数的最低位开始判断，若 $y_n=1$，则部分积加上被乘数|x|，然后右移一位；若若 $y_n=0$，则部分积加上0，右移一位。

4️⃣重复步骤3️⃣，重复进行n次。

每加一次，右移一次。

若原先两个数各是n+1位（包括一位符号位），其乘法得到的结果是2n+1位（包括一位符号位）

```
详见王道《计组》p43
```

原码一位乘法（无符号数乘法）需要的逻辑电路

包含被乘数寄存器X、乘数寄存器Y、乘积寄存器P、ALU、逻辑控制计数器Cn、进位位C

对于 $A_sA_1A_2...A_n \times B_sB_1B_2...B_n$

被乘数寄存器X：存放 $A_1A_2...A_n$，且**不再变化**；

乘数寄存器Y：存放 $B_1B_2...B_n$，随着每次加和以及右移，都有一位Bi被丢弃，最高位由计算得到的尾部补充，直到最后一次位移B1被丢失，乘法完成；

乘积寄存器P：存放过程中的积的和，每进行一次「加和」就右移一次；

ALU：进行加法核心操作

逻辑控制计数器Cn：主要三个功能，1）向C、P和Y发出右移信号；2）向ALU发出做加法信号；3）控制「加和+右移」的总次数，Cn初值为n，每进行一次右移Cn减1，当Cn=0，操作不再进行。

进位位C：乘积寄存器中可能会发生最高位的进位，用于保存最高位的进位。一般初值为00.

```
机器字长：计算机进行一次整数运算（即定点整数运算）所能处理的二进制数位数。
一般来说「机器字长 = 通用寄存器位数 = 算术逻辑单元ALU位数」
```

补码一位乘法（Booth算法）

这是一种「**有符号数**」的乘法，采用「相加」「相减」「右移」操作计算补码数据

运算规则：

设 $[X]_{原} = x_s.x_1x_2 \cdots x_n$，$[Y]_{原} = y_s.y_1y_2 \cdots y_n$

1️⃣**符号位参与运算**，运算的数均以补码表示；

2️⃣被乘数取「双符号位」参与运算（避免溢出错误）；部分积取「双符号位」，初值为00；乘数取「单符号位」，由于乘数寄存器Y和乘积寄存器P位数相同，多出的一位用来保存 y_{n+1} 来确定下一步操作，y_{n+1} 初值为0.

3️⃣根据 (y_n, y_{n+1}) 的取值来确定操作，移位按照补码右移规则进行

yn	yn+1	操作
0	0	部分积右移一位

yn	yn+1	操作
0	1	部分积加[X]补，右移一位
1	0	部分积加[-X]补，右移一位
1	1	部分积右移一位

4️⃣按照上述算法进行n+1步操作，但n+1步不再位移，仅根据yn和y{n+1}的比较结果和上表做相应运算。

共进行n+1次累加和n次右移

若原先两个数各是n+1位（包括一位符号位），其乘法得到的结果是2n+1位（包括一位符号位）

```
```
x=-0.1011 y=0.1101
x+=11.0101 x-=00.1011 y补=00.1101
            00.0000   0.1101|0   x-
            00.1011   0.1101|0

            00.1011   0.1101|0
            00.0101   10.110|10   x+
            11.0101   10.110|10

            11.1010   10.110|10
            11.1101   010.11|010   x-
            00.1011   010.11|010

            00.1000   010.11|010
            00.0100   0010.1|1010   +0
            00.0000   0010.1|1010

            00.0100   0010.1|1010
            00.0010   00010.|11010   x+
            11.0101   00010.|11010

            11.0111   00010.|11010
```
```

要点：双符号位、部分积一开始用00.0000，如果高位比低位低，就加[-x]补，高位比低位高，就加[x]补，如果相等则加0.每次加完之后都右移，最后一次加完不右移。**得到的结果是[x·y]补，如果答案要求原码或者真值需要进行转换！**

补码一位乘法运算电路

需要结构：32位被乘数寄存器X、32位ALU、32位乘积寄存器P、32位乘数寄存器Y、控制逻辑计数器Cn

被乘数寄存器X：不变；

ALU：按控制逻辑计数器的信号进行加或减运算

乘积寄存器P：保存部分积

乘数寄存器Y：初值为单符号的乘数+1位0(yn+1)

控制逻辑计数器Cn：主要三个功能，1）计时；2）将乘积寄存器P和乘数寄存器Y右移；3）根据yn和yn+1对ALU发出加法或减法指令。

定点数的除法 ⭐ ⭐

和乘法运算相对应，可以分为「**累加**」和「**左移**」，分为**原码除法**和**补码除法。**

符号拓展

在算数运算中，如果要一个8位的整数和32位的整数相加，则要将8位的整数先转换为32位整数形式，这称「**符号拓展**」

正数拓展：所有拓展位都为0；

负数拓展：原码补0；补码整数向前补1，小数向后补0。

原码一位除法（不恢复余数法）

不恢复余数法也称原码加减交替除法。**「商符」和「商值」的计算是分开的**，减法操作用补码加法实现，**商符**由两个操作数的符号位**异或**得到。

商的符号：$Q_s = x_s \oplus y_s$

商的数值：$|Q| = |X|/|Y|$

运算规则：

1️⃣先用被除数减去除数（|X|-|Y|=|X|+(-|Y|)=|X|+[-|Y|]补），当余数为正时，商上1，余数和商左移一位，再减去除数；当余数为负时，商上0，余数和商左移一位，再加上除数。

2️⃣若余数为正，商上1，左移，+|y|；若余数为负，商上0，左移，+[-|y|]补；

3️⃣第n+1步若余数为负，则+|y|得到正确的余数

总共进行了n+1次加法运算（其中一次在最开始），进行了n次左移。

补码一位除法（加减交替法）

补码一位除法的特点是，**「符号位和数值位」一起参加运算**，商符自然形成。

运算规则：

1️⃣ 符号位参与运算，除数和被除数均用补码表示，商和余数也用补码表示。

2️⃣ 第一步：若被除数与除数同号，则被除数减去除数；若被除数与除数异号，则被除数加上除数；

3️⃣ 若余数与除数同号，则商上1，余数左移一位减去除数；若余数与除数异号，则商上0，余数左移一位加上除数；

4️⃣ 重复执行第 3️⃣ 步操作n次。

5️⃣ 最后一步计算的是最终的余数，不再上商，**商恒置0。**

同号相除，够除商1，不够除商0；异号相除，够除商0，不够除商1。

总共进行了n+1次加法运算（其中一次在最开始），进行了n次左移。

除法得到的商有n+1位，余数也有n+1位尾数。

n位定点数的除法运算，实际上是用一个2n位的数去除以一个n位的数得到一个n位的商。因此需要对被除数进行拓展。对于n位定点正小数，需要在低位添n个0；对于n位无符号数或定点正整数，只需在被除数高位添n个0。