

2019 年全国硕士研究生招生考试

考研计算机综合试卷

注意事项：

1. 答卷前，考生务必将姓名、班级等填写清楚，解题时要认真审题，规范作答。
2. 本试卷共 0 道试题，满分 150 分，考试时间 120 分钟。

2019 年全国硕士研究生招生考试

考研计算机综合试卷

目录

0.1	00092-算法	8
0.2	00093-线性表的逻辑特性	8
0.3	00094-线性表的存储结构	8
0.4	00095-线性表的定义	9
0.5	00096-顺序表操作	9
0.6	00097-单链表操作	10
0.7	00098-双链表操作	11
0.8	00099-循环链表操作	11
0.9	00100-一元多项式的表示及相加	12
0.10	00101-栈的定义	12
0.11	00102-队列的定义	12
0.12	00103-顺序栈的定义	12
0.13	00104-顺序队的定义	13
0.14	00105-循环队列	13
0.15	00106-链栈的定义	13
0.16	00107-链队的定义	14
0.17	00108-链队的队空情况	14
0.18	00109-栈的应用	14
0.19	00110-队列的应用	15
0.20	00111-顺序存储	15
0.21	00112-特殊矩阵的压缩存储	16
0.22	00113-树相关的基本概念	16
0.23	00114-二叉树的定义	17
0.24	00115-二叉树的性质	18
0.25	00116-二叉树的存储结构	18
0.26	00117-二叉树的遍历算法	19
0.27	00118-二叉树的构造	20
0.28	00119-线索二叉树	21

0.29 00120-树的存储结构	21
0.30 00121-森林与二叉树的转换	22
0.31 00122-树和森林的遍历	24
0.32 00123-二叉排序树	25
0.33 00124-平衡二叉树	26
0.34 00125-赫夫曼树和赫夫曼编码	27
0.35 00126-图相关的基本概念	28
0.36 00127-图的存储结构	28
0.37 00128-图的遍历	29
0.38 00129-最小（代价）生成树	30
0.39 00130-最短路径	31
0.40 00131-拓扑排序	34
0.41 00132-关键路径	34
0.42 00134-顺序查找法	35
0.43 00135-分块查找	35
0.44 00136-折半查找法	36
0.45 00137-B-树	36
0.46 00138-B+ 树	37
0.47 00139-散列表	37
0.48 00140-平均查找长度 ASL	38
0.49 00141-各种查找算法分析	38
0.50 00142-排序的基本概念	39
0.51 00146-直接插入排序	39
0.52 00147-折半插入排序	39
0.53 00148-起泡排序	40
0.54 00149-简单选择排序	41
0.55 00150-希尔排序	41
0.56 00151-快速排序	42
0.57 00152-堆排序	42
0.58 00153-二路归并排序	43
0.59 00154-基数排序	43
0.60 00155-外部排序	43
0.61 00156-各种排序算法的比较	44
0.62 00275-检错编码和纠错编码	45
0.62.1 1. 检错编码	45
0.63 03686-计算机网络的功能	46
0.64 03687-计算机网络的分类	46
0.65 03689-计算机网络分层结构	47
0.66 03690-协议	47
0.67 03691-接口	47
0.68 03692-服务	48
0.69 03693-ISO-OSI 参考模型和 TCP-IP 参考模型	48
0.70 03694-计算机网络性能指标	49

0.71 03704-信号	50
0.72 03705-信源、信道及信宿	50
0.73 03706-速率、波特及码元	50
0.74 03707-带宽	51
0.75 03708-奈奎斯特定理	51
0.76 03709-香农定理	52
0.77 03710-编码与调制	52
0.78 03711-数据传输方式	53
0.79 03712-数据报与虚电路	55
0.80 03713-传输介质分类	56
0.81 03714-物理层接口特性	57
0.82 03715-中继器	57
0.83 03716-集线器	58
0.84 03717-数据链路层的主要功能	58
0.85 03718-四种组帧方法	58
0.86 03719-检错编码	60
0.87 03720-纠错编码	60
0.88 03721-流量控制	61
0.89 03722-可靠传输机制	61
0.90 03724-停止等待协议	62
0.91 03725-后退 N 帧 (GBN) 协议	62
0.92 03727-发送缓存和接收缓存	63
0.93 03729-信道划分介质访问控制	63
0.94 03730-随机访问介质访问控制	65
0.95 03731-轮询访问介质访问控制	66
0.96 03732-局域网的基本概念与体系结构	66
0.97 03733-以太网的工作原理	66
0.98 03734-以太网的 MAC 帧	67
0.99 03740-PPP 协议	67
0.10003741-HDLC 协议	68
0.10103742-网桥的概念和基本原理	69
0.10203743-网桥的分类	70
0.10303744-局域网交换机及工作原理	70
0.10403750-异构网络互连	70
0.10503751-路由与转发	71
0.10603760-静态路由与动态路由	71
0.10703763-层次路由	71
0.10803764-IPv4 分组	72
0.10903765-IPv4 地址	72
0.11003769-ARP	73
0.11103770-DHCP	74
0.11203771-ICMP	74
0.11303773-IPv6 的格式	74

0.11403774-路由选择协议分类	75
0.11503775-RIP	76
0.11603776-OSPF	76
0.11703777-RIP 和 OSPF 的比较	77
0.11803778-BGP	78
0.11903779-RIP、OSPF、BGP 最终陈述	78
0.12003780-组播的概念	78
0.12103783-移动 IP 的概念	79
0.12203784-移动 IP 的通信过程	79
0.12303786-路由表与路由转发	80
0.12403787-传输层的功能	80
0.12503789-无连接服务与面向连接服务	81
0.12603790-UDP 报文段	81
0.12703791-UDP 校验	82
0.12803792-TCP 报文段	83
0.12903793-TCP 连接管理	84
0.13003794-TCP 可靠传输	84
0.13103796-TCP 拥塞控制的基本概念	85
0.13203797-拥塞控制的 4 种算法	85
0.13303798-C-S 模型与 P2P 模型	86
0.13403799-DNS 系统的概念	87
0.13503801-域名服务器	87
0.13603802-域名解析过程	88
0.13703803-FTP 工作原理	89
0.13803804-控制连接与数据连接	89
0.13903806-电子邮件格式与 MIME	89
0.14003807-SMTP 与 POP3	90
0.14103808-WWW 的概念和组成结构	90
0.14203809-HTTP	91
0.14303934-入门知识	91
0.144入门知识 1：了解门电路	91
0.145入门知识 2：三态门	94
0.146入门知识 3：片选译码器	94
0.147入门知识 5：与存储相关的那些名词	96
0.148入门知识 6：与字、字长相关的那些名词	96
0.1491B=8bit，这个是规定，没有错误。但是很多考生认为一个字等于两个字节，因为他们脑海中有一种概念：一个汉字占用两个字节。计算机中的字和汉字中的字的概念不一样。计算机中的字通常由一个或多个（一般是字节的整数倍）字节构成。现在常用的都是 32 位（4 个字节）字长的机器。	96
0.150以前是存储字长等于机器字长，因为机器字长是机器一次能处理的比特数，这样一次取一个等长的存储字便于机器处理。现在机器字长一般大于存储字长。	96
0.151一般默认字长 = 机器字长 = 存储字长。	96
0.152入门知识 7：与周期相关的那些名词	96

0.15303937-计算机硬件的基本组成	97
0.15403940-计算机系统的层次结构	97
0.15503942-操作系统的概念	98
0.15603943-计算机性能指标	98
0.15703945-操作系统的主要功能和提供的服务	99
0.15803946-进位计数制及其相互转换	100
0.15903947-真值和机器数	101
0.16003949-字符和字符串	101
0.16103950-操作系统的形成与发展	102
0.16203952-校验码	102
0.16303953-内核态与用户态	103
0.16403955-定点数的表示	103
0.16503956-中断与异常	104
0.16603958-模块组合结构	104
0.16703962-进程的定义及描述	104
0.16803963-进程的状态与转换	105
0.16903964-进程的控制	106
0.17003965-线程	107
0.17103966-进程通信	107
0.17203967-处理器的三级调度	108
0.17303968-调度的基本原则	109
0.17403969-进程调度	110
0.17503970-常见调度算法	110
0.17603971-进程同步的基本概念	111
0.17703972-互斥实现方法	112
0.17803973-信号量	112
0.17903974-经典同步问题	113
0.18003975-关于 P、V 问题的解题思路	115
0.18103976-管程	116
0.18203977-死锁的概念	116
0.18303978-死锁产生的原因和必要条件	116
0.18403979-处理死锁的基本方法	117
0.18503980-死锁的预防	117
0.18603981-死锁的避免	117
0.18703982-死锁的检测和解除	118
0.18803983-死锁与饿死	119
0.18903984-定点数的运算	119
0.19003985-浮点数的表示	121
0.19103986-浮点数的加-减运算	122
0.19203987-串行加法器和并行加法器	122
0.19303988-算术逻辑单元的功能和结构	123
0.19403989-存储器的分类	124
0.19503990-存储器的层次化结构	124

0.19603991-半导体随机存取存储器基本概念	124
0.19703992-SRAM	126
0.19803993-DRAM	126
0.19903996-主存储器与 CPU 的连接	127
0.20003997-双口 RAM 和多模块存储器	127
0.20103998-Cache 的基本工作原理	128
0.20203999-内存管理概述	129
0.20304001-连续分配管理方式	130
0.20404002-Cache 和主存之间的映射方式	131
0.20504004-Cache 写操作策略	133
0.20604005-虚拟存储器的基本概念	133
0.20704006-页式虚拟存储器	134
0.20804012-请求分页存储管理方式	134
0.20904013-外存储器	135
0.21004014-页面置换算法	137
0.21104017-指令的基本格式	137
0.21204018-定长操作码指令格式	138
0.21304019-不定长操作码指令格式	138
0.21404021-常见寻址方式	139
0.21504022-CISC 和 RISC 的基本概念	140
0.21604023-CPU 的功能	141
0.21704024-CPU 的基本结构	141
0.21804025-CPU 中的主要寄存器	141
0.21904026-抖动现象与缺页率	142
0.22004027-请求分段存储管理系统	142
0.22104028-内存管理方式之间的比较	143
0.22204029-内存管理计算中地址的处理	143
0.22304031-指令周期	144
0.22404032-请求分页管理方式中有效访问时间的计算	144
0.22504033-指令的执行过程与信息流	145
0.22604034-文件的基本概念	146
0.22704036-文件的逻辑结构	146
0.22804037-目录结构	147
0.22904038-文件共享	148
0.23004039-数据通路的功能和基本结构	149
0.23104042-目录的实现	149
0.23204043-文件的实现	149
0.23304044-控制单元的功能	150
0.23404045-磁盘结构	152
0.23504046-调度算法	152
0.23604047-磁盘管理	153
0.23704048-I-O 设备的分类与 I-O 管理的任务	153
0.23804049-I-O 控制方式	153

0.23904050-I-O 软件层次结构	155
0.24004053-设备分配与回收	155
0.24104054-假脱机技术	156
0.24204055-控制单元的设计	157
0.24304056-指令流水线的基本概念	159
0.24404057-指令流水线的基本实现	159
0.24504058-超标量和动态流水线的基本概念	160
0.24604059-中断系统	160
0.24704060-总线的基本概念	161
0.24804061-总线的分类	161
0.24904062-总线的组成及性能指标	162
0.25004063-总线的结构	162
0.25104064-集中仲裁方式	163
0.25204065-分布仲裁方式	164
0.25304066-总线周期的概念	165
0.25404067-同步定时方式	165
0.25504068-异步定时方式	165
0.25604069-总线标准	166
0.25704070-I-O 系统基本概念	166
0.25804071-I-O 设备分类	166
0.25904072-输入设备	167
0.26004073-输出设备	167
0.26104074-I-O 接口基础知识	168
0.26204075-I-O 接口的功能和基本结构	168
0.26304076-I-O 端口及其编址	169
0.26404077-程序查询方式	169
0.26504078-程序中断方式	170
0.26604079-DMA 方式	171
0.26704081-加密算法	172
0.26804084-FDDI 环	172
0.26904087-浮点数的乘除法运算	173
0.27018659-CIDR	174
0.27118660-子网划分与子网掩码	175
0.27218661-IPv4 地址	175
0.27320845-时间复杂度与空间复杂度分析基础	176
0.27420848-数据物理结构	176
0.27520849-算法	176

0.1 00092-算法

算法可以理解为有基本运算及规定的运算顺序所构成的完整的解题步骤。

1. 算法的特性

- 1) 有穷性；
- 2) 确定性；
- 3) 输入；
- 4) 输出；
- 5) 可行性。

2. 算法的设计目标

- 1) 正确性；
- 2) 可读性；
- 3) 健壮性；
- 4) 高效率与低存储量需求（即更低的时间、空间复杂度）。

0.2 00093-线性表的逻辑特性

只有一个没有前驱的表头元素，只有一个没有后继的表尾元素，除了表头和表尾元素之外，其他元素只有一个直接前驱，也只有一个直接后继。

0.3 00094-线性表的存储结构

线性表的存储结构有顺序存储结构（顺序表）和链式存储结构（链表）两种。

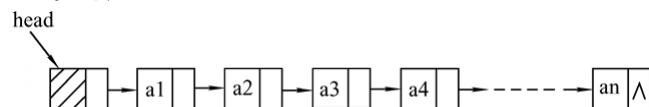
1. 顺序表

顺序表就是把线性表中的所有元素按照其逻辑顺序，依次存储到存储器中从指定位置开始的一块连续的存储空间中。

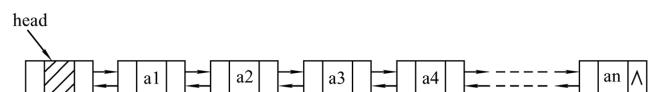
2. 链表

在链表存储中，每个结点不仅包含所存元素本身的信息，还包含元素之间逻辑关系的信息，即前驱结点包含后继结点的地址信息。链表有 5 种形式：

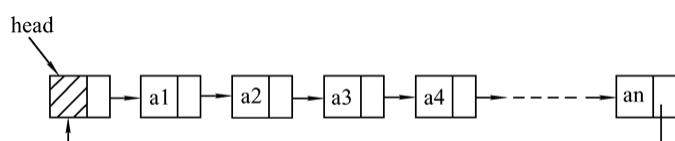
a. 单链表



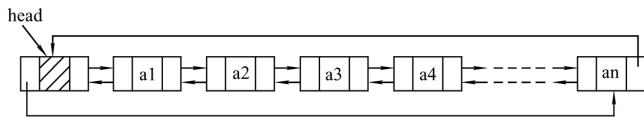
b. 双链表



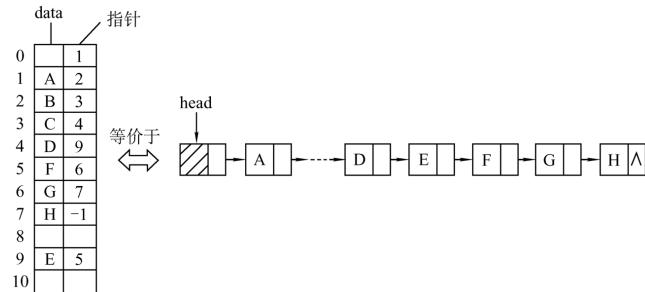
c. 循环单链表



d. 循环双链表



e. 静态链表



0.4 00095-线性表的定义

线性表是具有相同特性数据元素的一个有限序列。该序列中所含元素的个数叫做线性表的长度，用 $n(n \geq 0)$ 表示。

1. 顺序表的结构定义

```
#define max_size 100
typedef struct
{
    int data[max_size];      //存放顺序表元素的数组
    int length;              //存放顺序表长度
}Sqlist;
```

上面是结构体定义，考试中直接定义数组即可。如下：

```
int data[max_size];
int n;
```

2. 单链表结点定义

```
typedef struct LNode
{
    int data;                //数据域，存放int型数值
    struct LNode *link;      //指针域，存放后继结点地址
}LNode;
```

3. 双链表结点定义

跟二叉树结点的定义很像。

```
typedef struct DLNode
{
    int data;                //数据域，存放int型数值
    struct DLNode *prior;    //指向前驱的指针
    struct DLNode *next;     //指向后继的指针
}DLNode;
```

0.5 00096-顺序表操作

1. 按下标访问顺序表元素

```
x=A[i];      //访问下标为i的顺序表元素
```

2. 按元素值的查找算法

```

int LocateElem(Sqlist L,int key)
//在顺序表L中，查找第一个元素值为key并返回下标
{
    int i;
    for(i=0;i<L.length;i++)
        if(key==L.data[i])
            return i; //找到返回下标，小标均大于等于0
    return -1; //没找到返回-1
}

```

3. 插入数据元素的算法

```

int insert(Sqlist &L,int p,int key)
//在顺序表L中下标为p位置插入值key，原p位置元素及以后元素后移一个位置
{
    int i;
    if(p>=0&&p<L.length&&L.length<max_size)
        //p的合法取值范围
    {
        for(i=L.length-1;i>=p;i--)
            L.data[i+1]=L.data[i];
        //原p位置元素及以后元素后移一个位置
        L.data[p]=key; //将key放在插入位置上
        (L.length)++; //表长自增1
        return 1; //插入成功
    }
    else
        return 0; //插入失败
}

```

4. 删除数据元素的算法

```

int delete(Sqlist &L,int p,int &key)
//需要改变值的变量使用引用型声明
//在顺序表L中下标为p位置删除值key，原p位置以后元素前移一个位置
{
    int i;
    if(p>=0&&p<L.length) //p的合法取值范围
    {
        key=L.data[p]; //将被删除元素保存为key变量
        for(i=p;i<L.length;i++)
            L.data[i]=L.data[i+1]; //原p位置以后元素前移一个位置
        (L.length)--; //表长自减1
        return 1; //删除成功
    }
    else
        return 0; //删除失败
}

```

0.6 00097-单链表操作

1. 两个有序链表合并为一个有序链表

```

void merge(LNode*&la,LNode *&lb,LNode *&lc)
{
    //已知单链表la和lb的元素按值非递减排列
    //合并la和lb得到新的单链表lc,lc的元素也按值非递减排列
    LNode *pa=la->next; //pa来跟踪la的最小值结点
    LNode *pb=lb->next; //pb来跟踪lb的最小值结点
    LNode *pc; //pc来指向lc的链尾
    lc=pc=la; //用la的头结点作为lc的头结点
    while(pa&&pb) //若pa和pb都未到达链尾
    {
        if(pa->data<=pb->data) //pa所指结点更小
        {
            pc->next=pa; //将pa所指结点，插入pc所指结点后
            pc=pa; //更新pc指针，指向lc链尾
            pa=pa->next; //更新pa指针，指向la下一个的最小值结点
        }
        else //读者可参考if部分的注释，写出这段注释
        {
            pc->next=pb;
            pc=pb;
            pb=pb->next;
        }
    }
    pc->next=pa?pa:pb; //插入剩余段
    //若pa==NULL，则等价于pc->next=pb，否则等价于pc->next=pa;
    free(lb); //释放lb的头结点
}

```

2. 查找链表 L 中值为 key 的结点，存在就删除并返回 1，否则返回 0

```

int search_and_delete(LNode *&L, int key)
{
    LNode *p,*q;
    //p用于遍历链表，比较的是p所指结点的后一个结点
    //为什么是p所指结点的后一个结点，不是p所指的结点呢？
    //因为后面还有一个删除操作
    //需要知道被删除结点的前驱地址，故采取了这种策略
    //若无删除操作，则二者方法皆可
    //q用于保存待删除结点地址
    p=L;           //p指向L的头结点
    while(p->next !=NULL) //如果p所指结点不是链尾
    {
        if(p->next ->data==key) //比较p所指结点的后一个结点值域是否等于key
            break;
        p=p->next;          //p后移
    }
    if(p->next==NULL) //p到达链尾，没有后继结点
        return 0;         //查找失败
    else //查找成功，删除p所指结点的后一个结点
    {
        q=p->next;        //用q保存待删除结点的地址
        p->next=p->next->next; //将p所指结点的后继指向原后继结点的后继
        free(q);           //释放q所指的待删除结点空间
        return 1;           //查找成功
    }
}

```

0.7 00098-双链表操作

- 在双链表 L 尾部插入值为 key 的结点

```

void append_dlist (DLNode *&L, int key)
{
    DLNode *p,*s;
    while(p->next !=NULL)      //将p移到链表尾
        p=p->next;
    s=(DLNode*) malloc(sizeof(DLNode)); //创建新结点
    s->data=key;                //将新结点的值域赋值为key
    p->next=s;                  //将新结点挂到p所指结点后面
    s->next=NULL;               //将新结点的后驱赋为NULL
    s->prior=p;                 //将新结点的前驱指针指向原链尾
}

```

- 查找双链表 L 结点值为 key 的结点，找到返回其地址，否则返回 NULL

```

DLNode* search(DLNode *L, int key)
{
    DLNode *p=L->next;
    //将p指向L的头结点的后继，即第一个有值域的结点
    while(p!=NULL)
    {
        if(p->data==key) //比较p所指结点的值域是否为key
            break;
        p=p->next;       //p指针后移
    }
    return p;
    //若找到返回结点值为key的结点地址
    //否则p指针到达链尾返回NULL
}

```

- 在双链表中 p 所指结点之后插入结点 s，其中 p 结点不是最后一个结点

```

s->next=p->next;      //将p结点的原后继结点挂在结点s之后
p->next=s;            //将s接单挂在结点p之后
s->prior=p;           //将s前驱指向p结点
s->next->prior=s;    //将s结点后继结点的前驱指向s

```

注意：第一行和第二行一定要记住，先写这两句，就不会错。

4. 在双链表中删除 p 结点的后继结点

```

q=p->next;          //用q保存待删除结点的地址
p->next=q->next;   //将p结点后继指针指向待删除结点的后继结点
q->next->prior=p; //将待删除结点的后继结点的前驱指针指向p结点
free(q);             //释放待删除结点空间

```

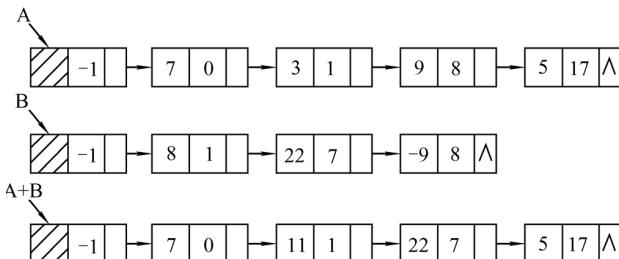
0.8 00099-循环链表操作

循环单链表和循环双链表是由对应的单链表和双链表改造而来，只需在终端结点和头结点间建立关系即可。只需要注意一点不同，如果 p 指针沿着循环链表遍历，判断 p 遍历到表尾结点的条件变为 p->next==head，其余操作都与非循环链表类似。

0.9 00100—一元多项式的表示及相加

一元多项式相加的运算规则：对于两个一元多项式中所有指数相同的项，对应系数相加，若其和不为零，则构成“和多项式”中的一项；对于两个一元多项式中所有指数不相同的项，则分别复抄到“和多项式”中去。

用链式存储结构，实现一元多项式的相加。示意图如下：



方法跟有序链表的合并有类似之处，首先对于结点的第二值域（指数），每个链表都是有序链表。如果两个一元多项式没有指数相同的项，则跟有序链表的合并是一模一样的。

不同的是这里可能会有指数相同的项，多了一个相加第一值域的操作。若和不为零，则构成新的一个结点。若和为零，则两个结点“同归于尽”。

考生根据这个思路和之前给的有序链表合并的代码，不难写出一元多项式的代码，由于不是重点部分，想看代码的考生请看严版教材。

0.10 00101-栈的定义

栈是一种只能在一端（栈顶）进行插入或删除操作的线性表。插入操作称为入栈，删除操作称为出栈。由栈的定义可得出栈的特点就是先进后出（First In Last Out，FILO）或后进先出。

0.11 00102-队列的定义

队列是一种操作受限的线性表，其限制为允许在表的一端（队尾）进行插入（进队），而在表的另一端（队头）进行删除（出队）。

由队列的定义可得出队列的特点就是先进先出（First In First Out, FILO）或后进后出。不同的是这里可能会有指数相同的项。

0.12 00103-顺序栈的定义

1. 顺序栈的定义

```
typedef struct
{
    int data[max_size];
    //存放栈元素, max_size为已定义的常量
    int top;      //栈顶指针
} SqStack;    //顺序栈类型定义
```

考试中要用到栈的情况下，栈的声明以及操作可以写得很简单，下面给出实用的顺序栈的操作的写法：

第一步：声明一个栈并初始化

```
int stack[max_size]; //存放栈元素
int top=-1; //初始化栈顶指针
```

第二步：元素 key 入栈

```
stack[++top]=key;
//top先自增1，然后再后移的位置插入key值
```

第三步：出栈，值保存于变量 key 中

```
key=stack[top--];
//先删除stack[top]，并把值保存于key变量中，然后top自减1
```

0.13 00104-顺序队的定义

1. 顺序队的定义

```
typedef struct
{
    int data[max_size];
    int front;        //队首指针
    int rear;         //队尾指针
} SqQueue;        //顺序队类型定义
```

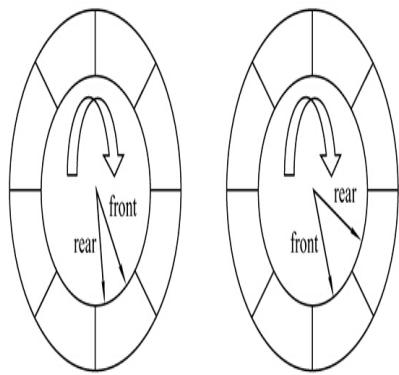
0.14 00105-循环队列

在顺序队列中，通常让队尾指针 rear 指向队尾元素，让队首 front 指针指向队头元素的前一个位置。（但这不是一定的，如 2011 年真题的第 3 道选择，就让 front 指向了队头元素）

为了预防 rear 和 front 都到达数组末端，造成无法让元素进队的情况，就需要把顺序队列改进为循环队列。如何实现指针的循环是一个问题，下面的语句解决了这个问题。

```
front=(front+1)%max_size;
rear=(rear+1)%max_size;
```

对于循环队列两个特殊的状态，表示如下：



考生可以自己先猜一下，哪个表示队空，哪个表示队满。

由定义可知，非空位置范围其实是 front 指针沿顺时针方向到达 rear 位置，扫过的空间为队列部分。但若 front 和 rear 都指向同一位置，就无法区分其为空队，还是满队了。为了区分，循环队列必须损失一个存储空间，如右图的情况，除了 front 位置，其余都为非空，定为队满的情况。就不能让 rear 超过 front 一圈（就好像两个同学在 400m 跑道上进行长跑一样，考虑到跑慢同学的自尊心，要求跑快的同学 rear 不可以超过跑慢同学 front 一圈，即最多超过他 399m，就不能再超了）。

综上，左图为队空 (rear==front)，右图为队满 (rear+1) %max_size==front。

0.15 00106-链栈的定义

1. 链栈的定义

```
typedef struct LNode
{
    int data;           //数据域
    struct LNode *next; //指针域
}LNode;               //链栈类型定义
```

2. 链栈的基本操作

(1) 入栈

```
p->next=l->next;
l->next=p;
```

(2) 出栈

```
p=l->next; //p保存待删除的栈顶地址
key=p->data; //将待删除的栈顶元素值保存于key中
l->next=p->next; //将栈顶指针指向新的栈顶元素
free(p); //释放待删除的栈顶元素空间
```

0.16 00107-链队的定义

1. 队结点类型定义

```
typedef struct QNode
{
    int data;
    struct QNode *next;
}QNode;
```

2. 链队类型定义

```
typedef struct
{
    QNode *front;      //队头指针
    QNode *rear;       //队尾指针
}LiQueue;             //链队类型定义
```

0.17 00108-链队的队空情况

链队有队空的情况：L->rear==NULL 或者 L->front== NULL。

可以这样记住，因为只要链队为非空，二者值都不应该为 NULL，反之为空。

链队在内存无限制的情况下，无队满的情况。

1. 进队（类似在链表尾进行插入操作）

```
lqu->rear->next=p; //将p所指的进队元素，插入到队尾  
lqu->rear=p; //更新rear指针
```

2. 出队（类似在链表头进行删除操作）

```
p=lqu->front; //用p保存待出队元素地址  
lqu->front=p->next; //更新front指针  
key=p->data; //保存出队元素值  
free(p); //释放出队元素所占空间
```

0.18 00109-栈的应用

以下部分只是让考生明白，什么问题可以用栈来解决。

1. 数制转换

如十进制转换为八进制的程序，计算过程是从低位到高位顺序产生八进制的各个位数。**而打印过程是从高位到低位进行，恰好和计算过程相反**。因此，若将计算过程中得到的八进制数的各位依次入栈，最后按出栈序列打印输出即为与输入对应的八进制数。

2. 括号匹配的检验

考虑下列括号序列：

[([])]

计算机接收了第一个左中括号后，期待右中括号的出现。等来的确是左小括号，此时第一个中括号必须等待，左小括号被消掉之后，才有可能被消掉，即这里**满足了栈的“先进后出”的特点**。用栈的处理括号的匹配时，遇左括号就进栈，遇右括号就出栈。

3. 行编辑程序

在编辑程序中，设立了一个缓冲区用于接收用户输入的一行字符，然后逐行存入用户数据区。允许用户输入出差错，并在发现有误时可以及时更正。例如用“#”表示退格符，用“@”表示退行符。其中退格符，删除的是最后输入的字符，**符合栈后入先出的特点**。退行符，删除的是缓冲区全部元素。这里用栈实现很方便。遇到普通字符，入栈。遇到退格符，栈顶元素出栈。遇到退行符，清空栈。

4. 迷宫求解

在“穷举求解”迷宫时，为了保证在任何位置上都能沿着原路退回，显然需要用一个后进先出的结构来保存从入口到当前位置的路径。因此，在求解迷宫通路的算法中应用“栈”也是很自然的事情。

5. 表达式求值

算法基本思想如下：

(1) 首先置操作数栈为空栈，表达式起始符“#”为运算符栈的栈底元素（为了和表达式的结束符，构成整个表达式的一对括号）。

(2) 依次读入表达式每个字符，若是操作数则进 OPND 栈，若是运算符则和 OPTR 栈栈顶运算符比较优先权后作相应操作，直至整个表达式求解完毕（即 OPTR 栈的栈顶元素和当前读入的字符均为“#”）。

6. 递归的实现（知道就行）

0.19 00110-队列的应用

队列应用的例子，其实就是要“排队”的情形，举例如下。

1. CPU 资源的竞争问题

在具有多个终端的计算机系统中，有多个用户需要使用 CPU 各自运行自己的程序，它们分别通过各自终端向操作系统提出使用 CPU 的请求，操作系统按照每个请求在时间上的先后顺序，将其排成一个队列，每次把 CPU 分配给队头用户使用，当相应的程序运行结束，则令其出队，再把 CPU 分配给新的队头用户，直到所有用户任务处理完毕。

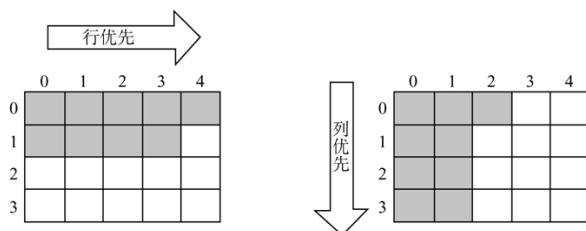
2. 主机与外部设备之间速度不匹配的问题

以主机和打印机为例来说明（09 年真题已经考查），主机输出数据给打印机打印，主机输出数据的速度比打印机打印的速度要快得多，若直接把输出的数据送给打印机打印，由于速度不匹配，显然是不行的。所以解决的方法是设置一个打印数据缓冲区，主机把要打印输出的数据依次写入到这个缓冲区中，写满后就暂停输出，继而去做其他的事情，打印机就从缓冲区中按照先进先出的原则依次取出数据并打印，打印完后再向主机发出请求，主机接到请求后再向缓冲区写入打印数据，这样利用队列既保证了打印数据的正确，又使主机提高了效率。

0.20 00111-顺序存储

1. 顺序存储

举 2 维数组的例子，同样是存储 $A[4][5]$ 中的 9 个元素：



- a. 行优先连续存储的是 $A[0][0 \sim 4]$ 和 $A[1][0 \sim 3]$ 。列优先连续存储的是 $A[0 \sim 3][0]$ 、 $A[0 \sim 3][1]$ 和 $A[0][2]$ 。
- b. 这里主要是涉及元素实际地址的计算。如给出 $A[0][0]$ 的地址求 $A[2][3]$ 的地址，在行优先和列优先就不同。
- c. 在行优先的情况下， $A[2][3]$ 为第 14 个元素，地址为 $A[0][0]$ 的地址加上 13 个元素空间的位移。在列优先的情况下， $A[2][3]$ 为第 15 个元素，地址为 $A[0][0]$ 的地址加上 14 个元素空间的位移。

0.21 00112-特殊矩阵的压缩存储

1. 对称矩阵

将对称矩阵 A 压缩存储到 $SA[n(n+1)/2]$ 中， a_{ij} 的下标 i, j 与在 SA 中的对应元素的下标 k 的关系如下：

当 $0 \leq i, j \leq n-1, 0 \leq k < n(n+1)/2$ 时

$$k = \begin{cases} i(i+1)/2 + j & \text{当 } i \geq j \\ j(j+1)/2 + i & \text{当 } i < j \end{cases}$$

2. 三角矩阵

将上三角矩阵 A 压缩存储到 $SA[n(n+1)/2]$ 中， a_{ij} 的下标 i, j 与在 SA 中的对应元素的下标 k 的关系如下：

当 $0 \leq i, j \leq n-1, 0 \leq k < n(n+1)/2$ 时

$$k = \begin{cases} i(i+1)/2 + j & \text{当 } i \geq j \\ n(n+1)/2 & \text{当 } i < j \end{cases}$$

下三角矩阵 A 压缩存储到 $SA[n(n+1)/2]$ 中，读者可自行给出。

3. 对角矩阵

所有的非零元素集中在以主对角线为中心的带状区域中，即除了主对角线和主对角线相邻两侧的若干条对角线上的元素之外，其余元素皆为零的矩阵称为对角矩阵。

$$\left(\begin{array}{ccccccccc} a_{00} & a_{01} & & & & & & & \\ a_{10} & a_{11} & a_{12} & & & & & & \\ a_{20} & a_{21} & a_{22} & a_{23} & & & & & \\ & & \ddots & \ddots & \ddots & & & & \\ & & a_{n-2,n-3} & a_{n-2,n-2} & a_{n-2,n-1} & & & & \\ & & a_{n-1,n-2} & a_{n-1,n-1} & & & & & \end{array} \right)$$

三对角矩阵

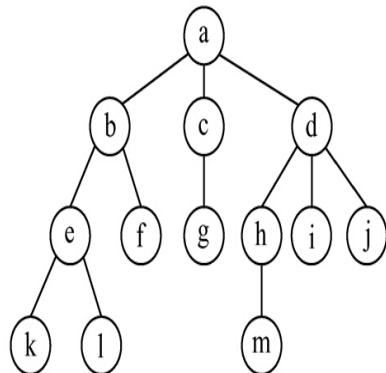
0.22 00113-树相关的基本概念

1. 树的基本概念

树是 $n (n \geq 0)$ 个结点的有限集合 T。当 $n=0$ 时，称为空树；当 $n>0$ 时，该集合满足如下条件：

1. 其中必有一个称为根的特定结点，它没有直接前驱，但有零个或多个直接后继（如结点 a）。
2. 其余 $n-1$ 个结点可以划分为 $m (m \geq 0)$ 个互不相交的有限集合 T_1, T_2, \dots, T_m ，其中 T_i 又是一棵树，称为根 root 的子树。每棵子树的根结点有且仅有一个直接前驱，但有零个或多个直接后继。

2. 树相关术语



1. 结点：a、b、m 都是结点，结点不仅包含数值元素，而且包含指向子树的分支。如结点 a 包含指向 b、c、d 这 3 个子树的指针。
2. 结点的度：结点拥有的子树个数（即分支个数）。如结点 A 的度为 3。
3. 叶子结点（终端结点）：指度为 0 的结点，如 f、g、i、j、k、l、m。
4. 非终端结点：指度不为 0 的结点，如 a、b、c、d、e、h。
5. 孩子：结点的子树的根，如 a 结点的孩子包括 b、c、d 这 3 个结点。
6. 双亲（父结点）：如 b、c、d 的双亲都是 a。
7. 兄弟：如 b、c、d 互为兄弟。
8. 祖先：从根到某结点的路径上的所有结点，都是这个结点的祖先。如 k 的祖先是 a、b、e。
9. 子孙（子树结点）：从某结点为根的子树中的所有结点，都是该结点的子孙。如 d 的子孙为 h、i、j、m。

10. 层次：从根开始，根为第一层，根的孩子们为第二层，以此类推。
11. 树的高度（或深度）：树中结点的最大层次。如例子中高度（或深度）为 4。
12. 结点的深度和高度：结点深度是从根结点算起的，而结点高度是从最底层的叶子结点算其的。如结点 g 的高度为 2，深度为 3。
13. 堂兄弟：双亲在同一层（但不是同一个）的结点互为堂兄弟。如 f、g、h 都互为堂兄弟。
14. 有序树：树中结点的子树从左到右是有次序的，不能交换的，这样的树称为有序树。
15. 无序树：就是子树无次序，可任意交换的树称为无序树。
16. 森林：若干棵互不相交的树的集合。如把根 A 去掉，剩下的 b、c、d 树就组成一个森林。

0.23 00114-二叉树的定义

1. 二叉树的定义

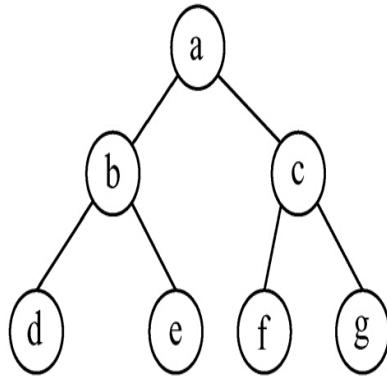
二叉树首先是一棵树。再附加两个限制条件：

条件一：每个结点最多只有两个子树，即二叉树中结点的度只能为 0、1、2。

条件二：子树有左右之分。

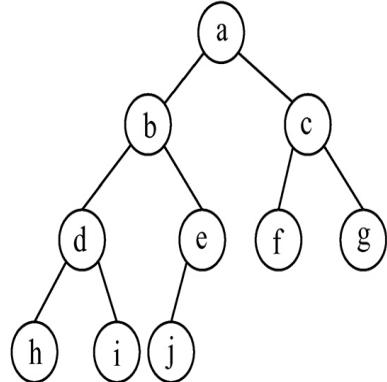
2. 满二叉树

在一棵二叉树中，如果所有的分支结点都有左孩子和右孩子结点，并且叶子结点都集中在二叉树的最后一层，这样的二叉树称为满二叉树，见下图。



3. 完全二叉树

若设二叉树的深度为 h，除第 h 层外，其他各层 ($1 \sim h-1$) 的结点数都达到最大个数，第 h 层所有的结点都连续集中在最左边，这就是完全二叉树，见下图。



0.24 00115-二叉树的性质

性质 1: 非空二叉树上叶子结点数等于双分支结点数加 1。设度为 i 的结点个数为 n_i 。性质 1 等式表示即为: $n_0 = n_2 + 1$ 。

性质 2: 二叉树的第 i 层上最多有 2^{i-1} 个结点 (i 1)

性质 3: 高度为 k 的二叉树最多有 $2^k - 1$ 个结点 (k 1)。

性质 4: 有 n 个结点的完全二叉树, 对各结点从上到下从左到右依次编号 (1 ~ n), 则结点之间有如下关系:

若 i 为结点 a 的编号则:

如果 $i \leq \frac{n}{2}$, 则 a 的双亲结点编号为 $\lfloor \frac{i}{2} \rfloor$ 。

如果 $2 \times i < n$, 则 a 有左孩子, 编号为 $2 \times i$, 反之无左孩子。

如果 $2 \times i + 1 \leq n$, 则 a 有右孩子, 编号为 $2 \times i + 1$, 反之无右孩子。

注: 因为完全二叉树一般直接用数组存储, 故通过双亲关系计算出下标的操作对完全二叉树来说是很重要的。

性质 5: 给定 n 个结点, 能构造出 2^{n-1} 种不同的二叉树。

性质 6: 具有 n 个结点的完全二叉树的深度 k 为 $\lfloor \log_2 n \rfloor + 1$ 或 $\lceil \log_2(n+1) \rceil$ 。

0.25 00116-二叉树的存储结构

1. 顺序存储结构

用一个数组来存储一棵二叉树, 适合于完全二叉树的存储。用于存储一般二叉树则会浪费大量存储空间。将完全二叉树中的结点值按编号一次存入一个一维数组中, 即完成了对一棵完全二叉树的顺序存储。

2. 链式存储结构

```
typedef struct BTNode
{
    int data;           //数值域
    struct BTNode *lchild; //左指针域
    struct BTNode *rchild; //右指针域
}BTNode;
```

0.26 00117-二叉树的遍历算法

二叉树的遍历方式有先序遍历、中序遍历、后序遍历和层次遍历。

识记: 这里的“先”、“中”、“后”指的都是根结点。

1. 先序遍历

```
void pre_order(BTNode *p)
{
    if(p!=NULL)           //二叉树非空
    {
        visit(p); //访问根结点, 假设visit函数已定义过
        pre_order(p->lchild); //先序遍历左子树
        pre_order(p->rchild); //先序遍历右子树
    }
}
```

2. 中序遍历

```

void in_order(BTNode *p)
{
    if(p!=NULL)           //二叉树非空
    {
        in_order(p->lchild); //先序遍历左子树
        visit(p); //访问根结点, 假设visit函数已定义过
        in_order(p->rchild); //先序遍历右子树
    }
}

```

3. 后序遍历

```

void post_order(BTNode *p)
{
    if(p!=NULL)           //二叉树非空
    {
        post_order(p->lchild); //后序遍历左子树
        post_order(p->rchild); //后序遍历右子树
        visit(p); //访问根结点, 假设visit函数已定义过
    }
}

```

4. 层次遍历

设二叉树的根结点所在层数为 1, 层序遍历就是从所在二叉树的根结点出发, 首先访问第一层的树根结点, 然后从左到右访问第 2 层上的结点, 接着是第三层的结点, 以此类推, 自上而下, 自左至右逐层访问树的结点的过程就是层序遍历。

要进行层次遍历需要建立循环队列。先将二叉树头结点入队列, 然后出队列, 访问该结点, 如果它有左子树, 便将左子树根结点入队, 如果它有右子树, 便将右子树根结点入队。然后出队列, 对出队结点访问, 如此反复, 直到队列为空为止。

```

void level(BTNode *p)
{
    int front,rear;
    BTNode *que[max_size];
    //定义一个循环队列, 用来记录将要访问的层次上的结点地址
    front=rear=0;
    //初始化队头和队尾指针
    BTNode *q;
    if(p!=NULL)
    {
        rear=(rear+1)%max_size; //队尾指针循环后移
        que[rear]=p;           //根结点的地址入队
        while(front != rear)   //当队列不为空时
        {
            front=(front+1)%max_size;//队头指针循环后移
            q=que[front];          //队头结点的地址出队
            visit(q);              //访问队头结点
            if(q->lchild!=NULL)    //若左子树非空, 则左子树根结点地址入队
            {
                rear=(rear+1)%max_size; //队尾指针循环后移
                que[rear]=q->lchild; //左子树根结点的地址入队
            }
            if(q->rchild!=NULL) //若右子树非空, 则右子树根结点地址入队
            {
                rear=(rear+1)%max_size; //队尾指针循环后移
                que[rear]=q->rchild; //右子树根结点的地址入队
            }
        }
    }
}

```

扩展:

- 求二叉树的深度, 二叉树以二叉链表为存储方式

```

int get_depth(BTNode *p)
{
    int ld,rd;
    if(p==NULL)           //空树, 返回0
        return 0;
    else
    {
        ld=get_depth(p->lchild); //求左子树深度
        rd=get_depth(p->rchild); //求右子树深度
        return (LD>RD?LD:RD)+1; //左右子树深度的最大值加1, 即为该树深度
    }
}

```

2. 查找 data 域值为 key 的结点

```

void search(BTNode *p,BTNode *&q)
//这里的参数q需要定义为引用型指针, 用于保存找到结点的地址。
//本算法实现的是先序遍历, 采用其他遍历方式也都是正确的
{
    if(p!=NULL) //树为非空
    {
        if(p->data==key) //如果当前结点data域值为key
            q=p;           //保存data域值为key结点的地址
        else
        {
            search(p->lchild,q); //查找左子树
            search(p->rchild,q); //查找右子树
        }
    }
}

```

0.27 00118-二叉树的构造

同一棵二叉树（结点值均不相同）具有唯一的先序、中序、后序和层次序列，但不同的二叉树可能具有相同的先序、中序、后序和层次序列。二叉树的构造就是根据提供的某些遍历序列构造二叉树的结构。

关于二叉树的构造有以下几种方式，构造方法都是用递归的思想：

1. 由先序序列和中序序列构造二叉树

先序：根结点（左子树的先序）（右子树的先序）；中序：（左子树的中序）根结点（右子树的中序）。这样就能确定出根结点，同时又得到了左子树的先序和中序，右子树的先序和中序。继续找左右子树的根结点，直到子树的结点个数为 1。构造完成。

2. 由后序序列和中序序列构造二叉树

后序：（左子树的后序）（右子树的后序）根结点；中序：（左子树的中序）根结点（右子树的中序）。这样就能确定出根结点，同时又得到了左子树的后序和中序，右子树的后序和中序。继续找左右子树的根结点，直到子树的结点个数为 1。构造完成。

3. 由层次序列和中序序列构造二叉树

这个举例会更清晰些，例如中序为 DGBAECF，层次遍历为 ABCDEFG。由层次遍历得知，根结点为 A。再根据中序就等到 DGB 为左子树的中序，ECF 为右子树的中序。又子树的层次遍历的结点顺序跟整棵树的层次遍历的结点顺序一致。得到左子树的层次遍历为 BDG，右子树的层次遍历为 CEF。这样就知道了左子树的层次和中序，右子树的层次和中序，采用同样方法即可，构造出整个二叉树。

0.28 00119-线索二叉树

n 个结点的二叉链表中含有 n+1 个空指针域。利用二叉链表中的空指针域，存放指向结点在某种遍历次序下的前趋和后继结点的指针（这种附加的指针称为“线索”）。这种加上了线索的二叉链表称为线索

链表，相应的二叉树称为线索二叉树。线索二叉树的结点结构定义如下：

lchild	ltag	data	rtag	rchild
--------	------	------	------	--------

其中：

ltag=0 时 lchild 指向左子女；

ltag=1 时 lchild 指向前驱；

rtag=0 时 rchild 指向右子女；

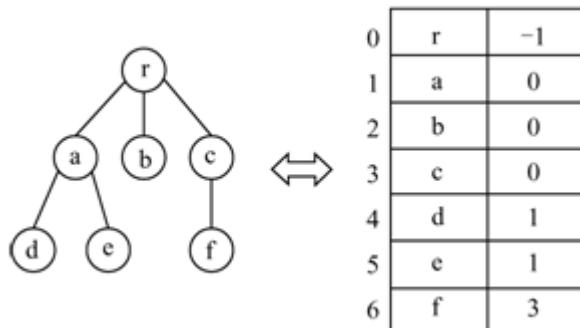
rtag=1 时 rchild 指向后继。

0.29 00120-树的存储结构

1. 双亲表示法

```
typedef struct PTNode           //结点结构
{
    int data;                  //值域
    int parent;                //用于记录双亲位置
}PTNode;
typedef struct                   //树结构
{
    PTNode nodes[max_size];   //结点空间
    int r,n;                 //r用于记录根位置, n用于记录结点数
}PTree;
```

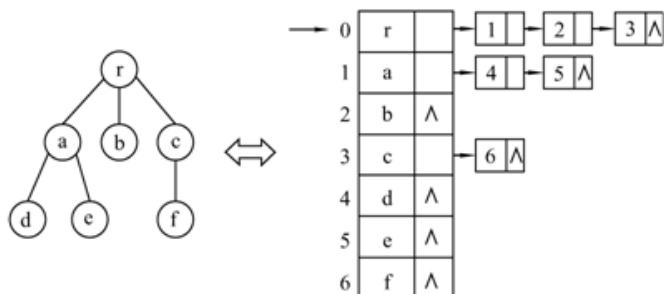
下图为一棵树及双亲表示法的存储结构。



2. 孩子表示法

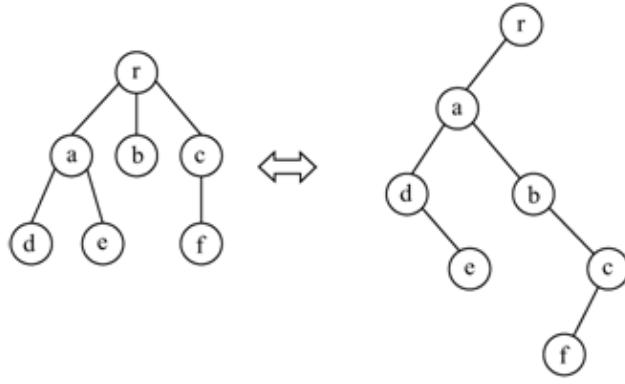
1) 假设 d 为树的度，采用 d 叉链表的存储结构，但由于很多结点的度小于 d，造成很大的浪费。

2) 把每个结点的孩子结点排列起来，看成一个线性表，且以单链表做存储结构，则 n 个结点有 n 个孩子链表。如下图所示的例子。



3. 孩子兄弟表示法

孩子兄弟表示法又称二叉链表表示法，其实就是把树转换为二叉树，进行存储，如下图所示。



0.30 00121-森林与二叉树的转换

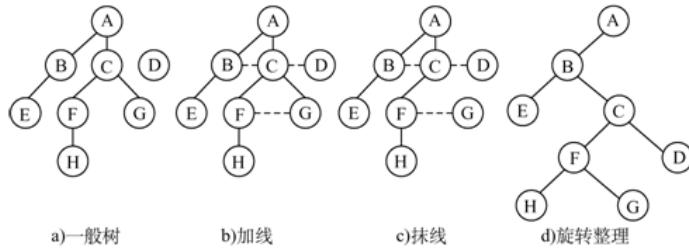
1. 将树转换为二叉树

步骤 1：加线。在各兄弟结点之间用虚线相连。可理解为每个结点的兄弟指针指向它的一个兄弟。

步骤 2：删除线。对每个结点仅保留它与其最左一个孩子的连线，删除该结点与其他孩子之间的连线。可理解为每个结点仅有一个孩子指针，让它指向自己的长子。

步骤 3：旋转。把虚线改为实线从水平方向向下旋转 45 度，成右斜下方向。原树中实线如左斜下方。这样就树的形状成呈现出一棵二叉树。

下图中 a 所示的树，可转换为图中 d 所示的二叉树。

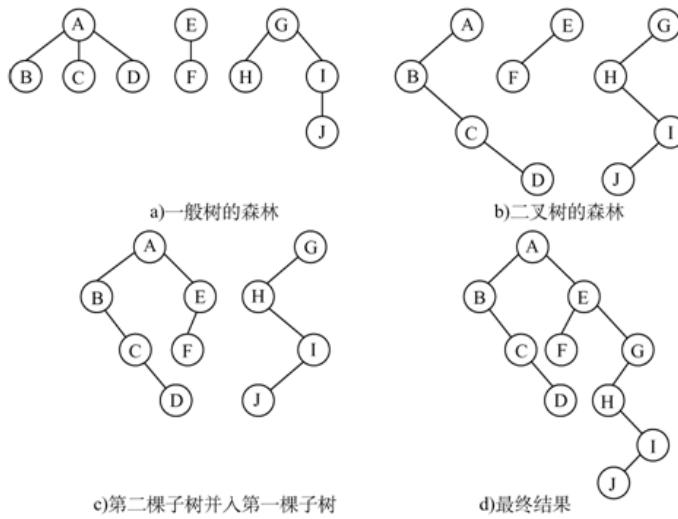


2. 将一个森林转换为二叉树

森林是树的有限集合，如下图 a 所示。由上节可知，一棵树可以转换为二叉树（没有右子树），一个森林就可以转换为二叉树（没有右子树）的森林。将森林转换为二叉树的一般步骤为：

步骤 1：将森林中每棵子树转换成相应的二叉树。形成有若干二叉树的森林，如下图 b 所示。

步骤 2：按森林图形中树的先后次序，依次将后边一棵二叉树作为前边一棵二叉树根结点的右子树，这样整个森林就生成了一棵二叉树，实际上第一棵树的根结点便是生成后的二叉树的根结点。下图是将一个森林转化为一棵二叉树的示例。图中 d 是转化后的一棵二叉树。下图中，图中 a 包含三棵树的森林可转换为图中 d 所示的二叉树。



3. 二叉树转换为一般树

此时的二叉树必须是由某一树（一般树）转换而来的没有右子树的二叉树。并非随便一棵二叉树都能还原成一般树。

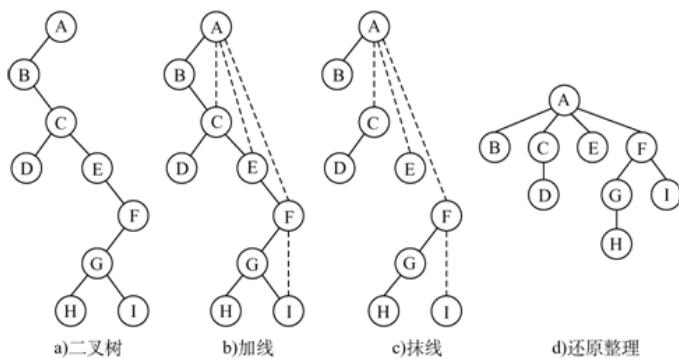
其还原过程也分为三步：

步骤 1：加线。若某结点 i 是双亲结点的左孩子，则将该结点 i 的右孩子以及当且仅当连续地沿着右孩子的右链不断搜索到所有右孩子，都分别与结点 i 的双亲结点用虚线连接。

步骤 2：删除线。把原二叉树中所有双亲结点与其右孩子的连线删除。这里的右孩子实质上是原一般树中结点的兄弟，删除的连线是兄弟间的关系。

步骤 3：进行整理。把虚线改为实线，把结点按层次排列。

下图是把二叉树还原为一般树。



4. 二叉树转换为森林

将一棵二叉树转化成森林，可按如下步骤进行：

删除线：将二叉树根结点与其右孩子之间的连线，以及沿着此右孩子的右链连续不断搜索到的右孩子间的连线删除。这样就得到了若干棵根结点没有右子树的二叉树。将得到的这些二叉树用前述方法分别转化成一般树。

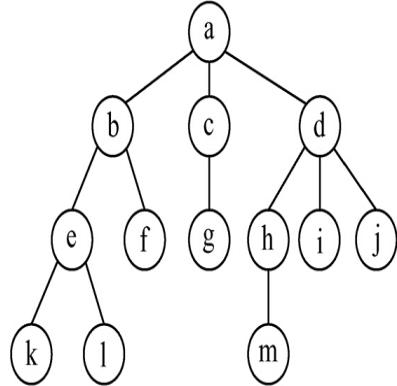
0.31 00122-树和森林的遍历

1. 树的遍历

树的遍历有两种方式：先根遍历和后根遍历。

先根遍历是先访问根结点，再依次访问根结点的每棵子树，访问子树时仍遵循先根再子树的规则。比如下图中的树进行先根遍历为：abeklfcgdmhij。

后根遍历是先依次访问根结点的每棵子树，再访问根结点，访问子树时仍遵循先子树再根的规则。比如下图中的树进行后根遍历为：klefbgcmhijda。



2. 森林的遍历

森林的遍历有两种方式：先序遍历和后序遍历。

先序遍历森林

若森林非空，则按下述规则遍历：

- 1) 访问森林中第一棵树的根结点。
- 2) 先序遍历第一棵树中根结点的子树森林。
- 3) 先序遍历除去第一棵树之后剩余的树构成的森林。

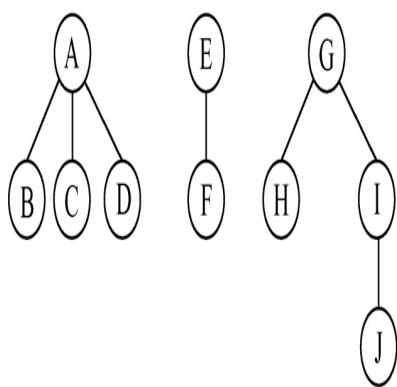
比如下图中的森林，先序遍历为：ABCDEFGHIJ。

中序遍历森林

若森林非空，则按下述规则遍历：

- 1) 中序遍历森林中第一棵树的根结点的子树森林。
- 2) 访问第一棵树的根结点。
- 3) 中序遍历除去第一棵树之后剩余的树构成的森林。

比如下图中的森林，中序遍历为：BCDAFEHJIG。



0.32 00123-二叉排序树

1. 二叉排序树定义

二叉排序树或者是空树，或者是满足如下性质的二叉树：

- (1) 若它的左子树不为空，则左子树上所有关键字的值均小于根关键字的值。
- (2) 若它的右子树不为空，则右子树上所有关键字的值均大于根关键字的值。

(3) 左右子树均为一棵二叉排序树。

如果输出二叉排序树的中序遍历序列，则它这个序列是递增有序的。

2. 查找关键字

```
BTNode* BSTSearch(BTNode* bt,int key)
//在二叉排序树，查找关键字为key的结点，并返回指向该结点指针
{
    if(bt==NULL)
        return NULL; //查找不成功，返回NULL
    else
    {
        if(bt->key==key) //查找成功，返回指向关键字所在结点的指针
            return bt;
        else if(key<bt->key) //key比根结点中关键字小，则查找左子树
            return BSTSearch(bt->lchild,key); //逐层
        else //key比根结点中关键字大，则查找右子树
            return BSTSearch(bt->rchild,key); //逐层
    }
}
```

3. 插入关键字

对查找关键字的算法进行修改即能完成插入关键字的算法。

```
BTNode* BSTInsert(BTNode * &bt,int key)
//在二叉排序树，查找关键字为key的结点，并返回指向该结点指针
//这里必须用引用声明，否则肯定出错误，因为插入新结点时，需要更新上一层bt->lchild
//或bt->rchild的值，故这里的bt必须是引用指针型
{
    if(bt==NULL) //当前为空指针，说明找到插入的位置
    {
        bt=(BTNode*)malloc(sizeof(BTNode)); //创建新结点
        bt->lchild=bt->rchild=NULL;
        bt->key=key; //将待插入关键字存入新结点中
        return 1; //插入成功，返回1
    }
    else
    {
        if(bt->key==key) //关键字存在
            return 0; //插入失败，返回0
        else if(key<bt->key) //key比根结点中关键字小，则查找左子树
            return BSTInsert(bt->lchild,key);
        else //key比根结点中关键字大，则查找右子树
            return BSTInsert(bt->rchild,key);
    }
}
```

4. 删 除关键字

假设在二叉排序树上被删除结点为 p, f 为其双亲结点，则删除结点 p 的过程分为 3 种情况：

(1) 若 p 结点为叶子结点，直接删除即可。

(2) 若 p 结点只有右子树而无左子树，或者只有左子树而无右子树。只需把 p 删掉，将 p 的子树直接连接在原来 p 与其双亲结点 f 相连的指针上即可。

(3) 若 p 结点既有左子树又有右子树。

沿 p 的左子树根结点的右指针一直往右，直到右子树的最右边的结点 r。将 p 的关键字用 r 中的关键字代替。最后判断，如果 r 是叶子结点按 1) 方法删除，若 r 是非叶子结点，则按 2) 方法删除 r。

0.33 00124-平衡二叉树

1. 平衡二叉树的概念

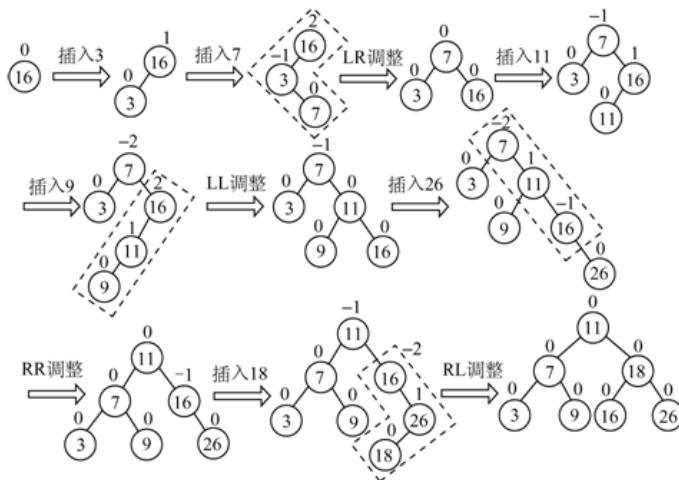
平衡二叉树（AVL 树）是一种特殊的二叉排序树。其左右子树都是平衡二叉树，且左右子树高度之差的绝对值不超过 1。也可以表述为：以树中所有结点为根的树的左右子树高度之差的绝对值（平衡因子）不超过 1。

2. 平衡二叉树的插入

前面跟二叉排序树的插入一样，就是在插入新的关键字后要进行检查，看新关键字的插入是否会使原平衡二叉树失去平衡，若失去平衡，就要进行平衡调整。

当失去平衡的最小子树被调整为平衡子树的时候，原有其他所有不平衡子树无需调整，整个二叉排序树就会成为一棵平衡二叉树。

失去平衡的最小子树就是以距离插入结点最近，且平衡因子等于 2 或 -2 的结点作为根的子树。平衡调整有 4 种情况，分别为 LL 型、RR 型、LR 型和 RL 型。举例如下，以关键字序列 {16, 3, 7, 11, 9, 26, 18} 构造一棵平衡二叉树，如下图所示。



3. 平衡二叉树的删除

前面与二叉排序树的删除一样，就是在删除关键字后要进行检查，看新关键字的插入是否会使原平衡二叉树失去平衡，若失去平衡，就要进行平衡调整。

0.34 00125-赫夫曼树和赫夫曼编码

1. 赫夫曼树

赫夫曼树又称最优二叉树，是一种带权路径长度最短的二叉树。所谓树的带权路径长度，就是树中所有的叶结点的权值乘上其到根结点的路径长度（路径长度即路径上的分支数）。树的带权路径长度记为：

$$WPL = (W_1 * L_1 + W_2 * L_2 + W_3 * L_3 + \dots + W_n * L_n)$$

N 个权值 W_i ($i=1, 2, \dots, n$) 构成一棵有 N 个叶结点的二叉树，相应的叶结点的路径长度为 L_i ($i=1, 2, \dots, n$)。可以证明赫夫曼树的 WPL 是最小的。

2. 构造赫夫曼树的方法

第一步：将给定的 n 个权值分别看成只有根的 n 棵二叉树，这些二叉树构成的集合记为 F。

第二步：从 F 中选出两棵根结点的权值最小的树（假设 a 和 b）作为左右子树构造一棵新的二叉树（假设为 c），新的二叉树的根结点的权值为左右子树根结点权值之和。

第三步：从 F 中删除 a 和 b，加入 c。

第四步：重复第二步和第三步，直到 F 中只剩下棵树为止，这棵树便是要遍历的赫夫曼树。

3. 对赫夫曼树的性质总结

性质 1：用 n 个权值的赫夫曼树，需要 $n-1$ 次合并。

性质 2：赫夫曼树中只有度为 0 和 2 的结点。

性质 3: n 个权值对应的赫夫曼树不唯一。

性质 4: 深度为 h 的赫夫曼树, 最大编码长度为 h-1。

性质 5: 赫夫曼编码是能使电文代码总长最短的编码方式。

性质 6: 赫夫曼是一种前缀编码 (就是任一字符的编码, 都不是另一个字符编码的前缀)。

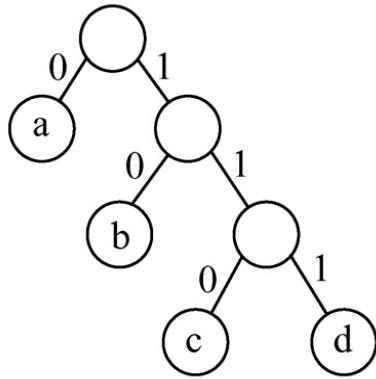
4. 赫夫曼编码

根据给定字符值求赫夫曼编码的过程, 实质就是根据给定权值构造赫夫曼树的过程。以 a、b、c、d 的出现频率为 0.4、0.3、0.1、0.2 为例, 求赫夫曼编码过程如下:

第一步: 根据权值构造赫夫曼树。

第二步: 对构造好的赫夫曼树, 约定左分支为 0, 右分支为 1。

第三步: 从根结点到叶子结点的分支上的编码组成该叶子结点的赫夫曼编码, A 为 0, B 为 10, C 为 110, D 为 111, 如下图所示。



0.35 00126-图相关的基本概念

(1) 图: 图是有结点的有穷集合 V 和边的集合 E 组成。

(2) 有向图和无向图: 有向图每条边都有方向, 无向图每条边都没有方向。

(3) 弧: 有向图中, 边称为弧, 记为 $\langle v_i, v_j \rangle$, 表示一条从顶点 v_i 到顶点 v_j 的边。

(4) 顶点的度、入度和出度: 与顶点 v 相关的边数称为顶点 v 的度。入度和出度都是有向图的概念。在有向图中指向顶点 v 的边数称为顶点 v 的入度, 由顶点 v 发出的边数称为顶点 v 的出度。

(5) 有向完全图和无向完全图: 有向图最多有 $n(n-1)$ 条边, 称这样具有 $n(n-1)$ 条边的有向图为有向完全图。无向图最多有 $n(n-1)/2$ 条边, 称这样具有 $n(n-1)/2$ 条边的无向图为无向完全图。

(6) 路径和路径长度: 路径即为相邻顶点序偶构成的序列, 如 $\langle a, b \rangle, \langle b, c \rangle$ 构成一条路径。路径长度指路径上边的数目。

(7) 简单路径: 路径序列中顶点和边都不重复出现的路径为简单路径。

(8) 回路: 一条路径第一个顶点和最后一个顶点相同, 称为回路。

(9) 无向图的连通、连通图和连通分量: 这 3 个概念都是针对无向图的。从顶点 v_i 到顶点 v_j 有路径, 称 v_i 和 v_j 连通。任意两个顶点之间都连通, 成为连通图。否则, 将其中最大的极大连通子图称为连通分量。

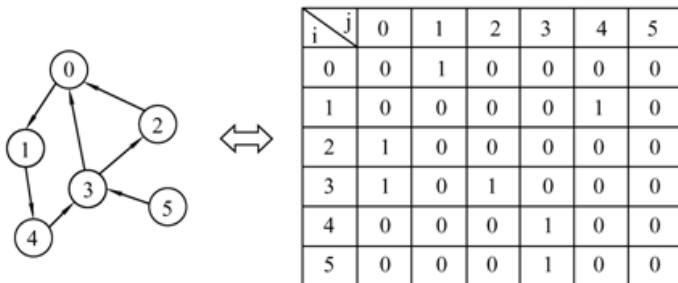
(10) 有向图的连通、强连通图和强连通分量: 这 3 个概念都是针对有向图的。从顶点 v_i 到 v_j 有路径, 称 v_i 和 v_j 连通。对于每一对顶点 v_i 和 v_j , 从 v_i 到 v_j 和 v_j 到 v_i 都有路径, 称该图为强连通图。否则, 将其中极大强连通子图称为强连通分量。

(11) 权和网: 图中每条边都可以有值, 用于表示从一个顶点到另一个顶点的距离 (代价), 该数值称为权。边上带权的图称为带权图, 也称为网。

0.36 00127-图的存储结构

1. 邻接矩阵

邻接矩阵是图的顺序存储结构。 $A[i][j]=1$ 表示顶点 i 到顶点 j 存在边或弧。 $A[i][j]=0$ 则没有，如下图所示。

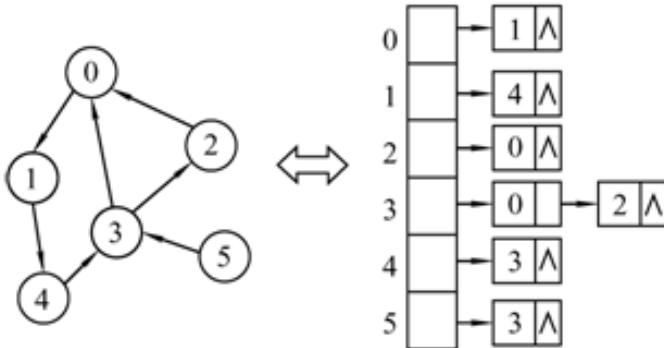


邻接矩阵的结构型定义如下：

```
typedef struct
{
    int no;                                //顶点编号
    char info;                             //顶点其他信息
}Vertex;
typedef struct
{
    int edges[max_size][max_size]; //邻接矩阵
    int n,e;                            //顶点数和边数
    Vertex vex[max_size];             //存放结点信息
}MGraph;                                //图的邻接矩阵类型
```

2. 邻接表

邻接表是图的链式存储结构。对每个顶点建立一个单链表，每个单链表的第一个结点存放顶点信息，其余结点存放边信息。如下图所示。



邻接表存储表示的定义如下：

```

typedef struct ArcNode           //定义边结点
{
    int adjvex;                //该边所指向的结点的位置
    struct ArcNode *nextarc;   //指向下一条边的指针
    int info;                  //边的其他信息，如权值
}ArcNode;

typedef struct VNode             //定义顶点结点
{
    char data;                 //顶点信息
    ArcNode *firstarc;         //指向第一条边的指针
}VNode;

typedef struct                   //定义图的邻接表
{
    VNode adjlist[max_size];   //邻接表
    int n,e;                  //顶点数和边数
}AGraph;

```

0.37 00128-图的遍历

1. 深度优先搜索遍历 (DFS)

从图中的某结点 v 开始访问，访问他的任意一个邻结点 w1；再从 w1 出发，访问与 w1 邻接但是没有被访问过的结点 w2；然后再从 w2 出发，进行类似的访问……，如此进行下去，当一个顶点的所有邻接结点都被访问过时，则依次退回到前一次刚访问过的结点，看是否还有其他没有被访问的邻接结点。如果有，则访问此结点，然后再从此结点出发，进行与前述类似的访问。重复上述过程，直到连通图中所有结点都被访问过为止。

图的深度优先搜索遍历类似于二叉树的先序遍历，唯一的区别在于：二叉树的先序遍历对于每个结点要递归地访问两个分支，图的深度优先搜索遍历则是要递归地访问多个分支。

以邻接表为存储结构的深度优先搜索遍历算法：

```

int visit[max_size];           //全局数组，作为顶点的访问标记，全部初始化为0
void DFS(AGraph *G,int v)
{
    ArcNode *p;
    visit[v]=1;               //置已访问标记
    visit(v);                 //访问顶点v
    p=G->adjlist[v].firstarc; //p指向顶点v的第一条边
    while(p!=NULL)
    {
        if(visit[p->adjvex]==0) //若顶点未访问，则递归
            visit(p->adjvex); //访问它
        p=p->nextarc;          //p指向顶点v的下一条边的结点
    }
}

```

2. 广度优先搜索遍历 (BFS)

从图中某个顶点 v 出发，并访问该顶点，然后访问 v 的各个未曾被访问的邻接点 w1, w2, … , wk，然后依次从 w1, w2, … , wk 出发访问各自未被访问的邻接点。重复步骤 2，直到全部顶点都被访问为止。

以邻接表为存储结构的广度优先搜索遍历算法：

```

int visit[max_size];           //全局数组，作为顶点的访问标记，全部初始化为0
void BFS(AGraph *p,int v)
{
    ArcNode *p;
    int que[max_size],front=0,rear=0; //定义队列
    int j;
    Visit(v);                      //Visit为访问顶点的函数
    visit[v]=1;                     //标记顶点v为已访问
    rear=(rear+1)%max_size; //队列的队尾指针后移
    que[rear]=v;                   //顶点v入队
    while(front!=rear)            //若队非空
    {
        front=(front+1)%max_size;//队头指针后移
        j=que[front];           //顶点出队，用j保存顶点号
        p->adjlist[j].firstarc; //p指向出队顶点的第一条边
        while(p!=NULL)          //遍历顶点的所有边结点
        {
            if(visit[p->adjvex]==0) //若当前邻接顶点未被访问则进队
            {
                Visit(p->adjvex); //访问邻接顶点
                visit[p->adjvex]=1; //标记为已访问
                rear=(rear+1)%max_size; //队尾指针后移
                que[rear]=p->adjvex; //将该项点入队
            }
            p=p->nextarc;        //p指针后移，指向下一条边
        }
    }
}

```

0.38 00129-最小（代价）生成树

最小生成树是一个有 n 个结点的连通图的生成树是原图的极小连通子图，且包含原图中的所有 n 个结点，并且有保持图连通的最少的边。

最小生成树的总结：

性质 1：使用不同的遍历图的方法，可以得到不同的生成树；从不同的顶点出发，也可能得到不同的生成树。

性质 2：按照生成树的定义， n 个顶点的连通网络的生成树有 n 个顶点、 $n-1$ 条边。

性质 3：构造最小生成树的准则：

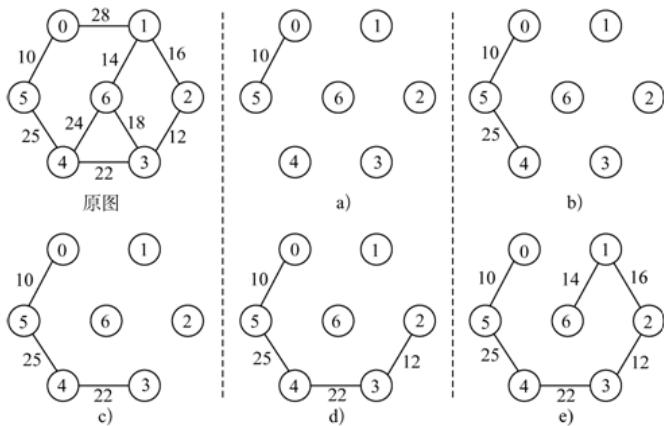
必须使用且仅使用该网络中的 $n-1$ 条边来连接网络中的 n 个顶点；

不能使用产生回路的边；

各边上的权值的总和达到最小。

2. 普里姆算法

从连通网络 $N=\{V, E\}$ 中的某一顶点 u 出发，选择与它关联的具有最小权值的边 (u, v) ，将其顶点加入到生成树顶点集合 U 中。以后每一步从一个顶点在 U 中，而另一个顶点不在 U 中的各条边中选择权值最小的边 (u, v) ，把它的顶点加入到集合 U 中。如此继续下去，直到网络中的所有顶点都加入到生成树顶点集合 U 中为止（读者可以用下图所示的例子实践一下，明白过程即可，一般考研试卷不直接考查代码实现）。



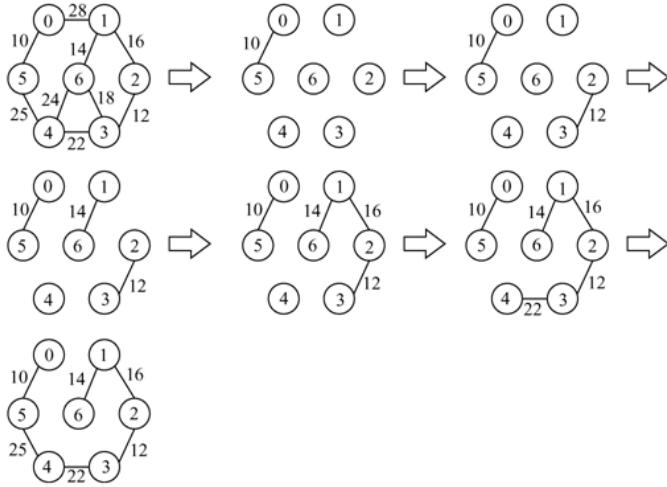
总结：

(1) 普里姆算法的复杂度为 $O(n^2)$ 。

(2) 普里姆算法适用于稠密图。

2. 克鲁斯卡尔算法

将下图中边按权值从小到大排序，然后从最小边开始扫描，并检测当前边是否为候选边（即是否该边的并入会构成回路），如不构成回路，则将该边并入当前生成树中，直到所有边都被检测完为止。（读者可以用如下图所示的例子实践一下，明白过程即可，一般考研试卷不直接考查代码实现）



总结：

(1) 复杂度取决于排序算法。

(2) 克鲁斯卡尔算法适用于稀疏图。

0.39 00130-最短路径

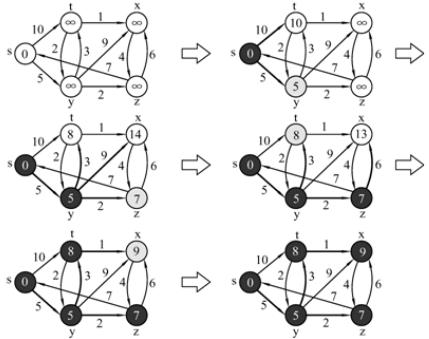
最短路径问题是图论研究中的一个经典算法问题，旨在寻找图（由结点和路径组成的）中两结点之间的最短路径。算法具体的形式包括：

第一种：迪杰斯特拉算法，此算法是一种确定起点求最短路径的问题。

第二种：弗洛伊德算法，此算法是求图中任意一对顶点间的最短路径。

1. 迪杰斯特拉算法

设有两个顶点集合 S 和 T ，集合 S 中存放图中已找到最短路径的顶点，集合 T 存放图中剩余顶点。初始状态时，集合 S 中只包含源点 v_0 ，然后不断从集合 T 中选取顶点 v_0 路径长度最短的顶点 v_u 并入到集合 S 中。集合 S 每并入一个新顶点 v_u ，都要修改顶点 v_0 到集合 T 中顶点的最短路径长度值。不断重复此过程，直到集合 T 的顶点全部并入到 S 中为止。（举例如下图所示）



源点 s 为最左端顶点。顶点内的值为最短路径估计，粗线的边指出该最短路径估计的前驱顶点。黑色顶点在集合 S 中，灰色顶点和白色顶点都在 T 中，灰色顶点为当前 T 中最短路径估计最短的顶点, v_u 。

总结：

- 1) 迪杰斯特拉算法复杂度为 $O(n^2)$ 。
- 2) 用迪杰斯特拉求解每一对顶点之间的最短路径，解决办法就是每次以一个顶点为源点，重复执行迪杰斯特拉算法 n 次。就能求得每一对顶点之间的最短路径。总时间复杂度为 $O(n^3)$ 。
- 3) 迪杰斯特拉算法实现比较复杂，一般不会考查代码实现，但求解过程一定要掌握，即读懂上图的全部过程，并且能独立正确求解。

2. 弗洛伊德算法（可选看，掌握迪杰斯特拉算法即可）

设 D_{ijk} 为从 i 到 j 的只以 $(1 \dots k)$ 集合中的结点为中间结点的最短路径的长度。

- 1) 若最短路径经过点 k ，则 $D_{ijk}=D_{ik,k-1}+D_{kj,k-1}$ ；
- 2) 若最短路径不经过点 k ，则 $D_{ijk}=D_{ijk-1}$ 。

因此， $D_{ijk}=\min(D_{ik,k-1}+D_{kj,k-1}, D_{ijk-1})$ 。

代码描述如下：

```
for k←1 to n do
    for i←1 to n do
        for j←1 to n do
            if (Di,k+Dk,j<Dij) then
                Di,j←Di,k+Dk,j;
```

其中， D_{ij} 表示由点 i 到点 j 的代价，当 D_{ij} 为 ∞ 表示两点之间没有任何连接。（参考下面的例子）



$$A_{-1} = \begin{bmatrix} 0 & 10 & 5 & \infty & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & 3 & 0 & 9 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$A_{-1} = \boxed{\begin{bmatrix} 0 & 10 & 5 & \infty & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & 3 & 0 & 9 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & \infty & \infty & 6 & 0 \end{bmatrix}}$$

$$\text{Path}_{-1} = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

$$A_0 = \boxed{\begin{bmatrix} 0 & 10 & 5 & \infty & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & 3 & 0 & 9 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & 17 & 12 & 6 & 0 \end{bmatrix}}$$

$$\text{Path}_0 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & 0 & 0 & -1 & -1 \end{bmatrix}$$

$$A_1 = \boxed{\begin{bmatrix} 0 & 10 & 5 & 11 & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & 3 & 0 & 4 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & 17 & 12 & 6 & 0 \end{bmatrix}}$$

$$\text{Path}_1 = \begin{bmatrix} -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & 0 & 0 & -1 & -1 \end{bmatrix}$$

$$A_2 = \boxed{\begin{bmatrix} 0 & 8 & 5 & 9 & 7 \\ \infty & 0 & 2 & 1 & 4 \\ \infty & 3 & 0 & 4 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & 15 & 12 & 6 & 0 \end{bmatrix}}$$

$$\text{Path}_2 = \begin{bmatrix} -1 & 2 & -1 & 2 & -1 \\ -1 & -1 & -1 & -1 & 2 \\ -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & 2 & 0 & -1 & -1 \end{bmatrix}$$

$$A_3 = \boxed{\begin{bmatrix} 0 & 8 & 5 & 9 & 7 \\ \infty & 0 & 2 & 1 & 4 \\ \infty & 3 & 0 & 4 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & 15 & 12 & 6 & 0 \end{bmatrix}}$$

$$\text{Path}_3 = \begin{bmatrix} -1 & 2 & -1 & 2 & -1 \\ -1 & -1 & -1 & -1 & 2 \\ -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & 2 & 0 & -1 & -1 \end{bmatrix}$$

$$A_4 = \boxed{\begin{bmatrix} 0 & 8 & 5 & 9 & 7 \\ 11 & 0 & 2 & 1 & 4 \\ 9 & 3 & 0 & 4 & 2 \\ 11 & 19 & 16 & 0 & 4 \\ 7 & 15 & 12 & 6 & 0 \end{bmatrix}}$$

$$\text{Path}_4 = \begin{bmatrix} -1 & 2 & -1 & 2 & -1 \\ 3 & -1 & -1 & -1 & 2 \\ 3 & -1 & -1 & 1 & -1 \\ 3 & 3 & 3 & -1 & -1 \\ -1 & 2 & 0 & -1 & -1 \end{bmatrix}$$

...

A_k 代表以 k 为中间结点，检测更新 D_{ij} 的最短路径。 Path_k 代表当前两顶点间最短路径上要经过的中间顶点。在图 5-6 中， A_{k-1} 的虚线方框中就是生成 A_k 时，需要进行加和比较的数值。当 $k=1$ 时， $k=2$ 时， $A[4][1]=17$ ，就需要和 $A[4][2]+A[2][1]=12+3=15$ 进行比较，取最小值，修改 $A[4][1]=15$ ，并修改 $\text{path}[2][1]=2$ 。

由于该算法实现代码还是挺简单的，只要理解了上面的过程就能很轻松写出。

```

void Floyd(MGraph g,int A[][max_size],int Path[]
[max_size])
{
    int i,j,k;
    for(i=0;i<g.n;++i) //两个循环对A和Path进行初始化
        for(j=0;j<g.n;++j)
    {
        A[i][j]=g.edges[i][j];
        Path[i][j]=-1;
    }
    for(k=0;k<g.n;++k) //算法的主要操作部分，以k为中间
                        //结点，对所有顶点
        for(i=0;i<g.n;++i) //对进行检测和修改
            for(j=0;j<g.n;++j)
                if(A[i][j]>A[i][k]+A[k][j])
                {
                    A[i][j]=A[i][k]+A[k][j];
                    Path[i][j]=k;
                }
}

```

0.40 00131-拓扑排序

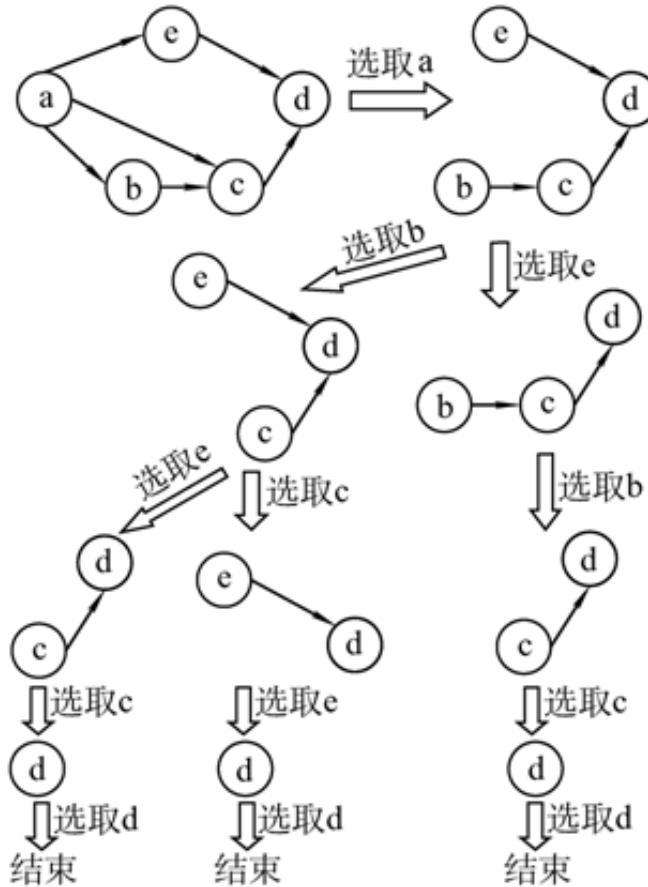
AOV (Activity On Vertex network) 网是一种顶点表示活动，边表示活动的先后次序，没有回路的有向图。

对一个有向无环图 G 进行拓扑排序，是将 G 中所有顶点排成一个线性序列，使得图中任意一对顶点 u 和 v，若存在从 u 到 v 的路径，则 u 在线性序列中出现在 v 之前。

拓扑排序的步骤为：

1. 在有向图中选一个没有前驱的顶点并且输出之。
2. 从图中删除该顶点和所有以它为尾的弧。
3. 重复上述两步，直至全部顶点均已输出或图中剩余的顶点中都有前驱。后者说明有向图中有环。

由于没有前驱的顶点可能不唯一，所以拓扑排序的结果也不唯一，如下图所示。



故本例有 3 个不同的拓扑排序序列，分别为：abced、abecd、aebcd。

0.41 00132-关键路径

AOE (Activity On Edge network) 网的边表示活动，边有权值代表活动持续时间，顶点表示事件，事件是图中新活动开始或者旧活动结束的标志。

在 AOE 网络中，**从源点到汇点的所有路径中，具有最大路径长度的路径称为关键路径**。关键路径代表了整个工程所完成的**最短时间**。这么理解吧，工程的各个段是可以并行进行的，即使这个工程并行程度足够高，最长路径代表了工程至少需要的完成时间。

求关键路径的一般方法：

1) 根据图求出拓扑有序序列 a 和逆拓扑有序序列 b。(若有多条，任取一条即可)

2) 根据序列 a 和 b 分别求出每个事件的最早发生时间和最迟发生时间，求解方法如下：

一个事件的最早发生时间为指向它的边（假设为 a）的取值加上发出 a 这条边的事件的最早发生时间。如果有多条边则逐一求出对应的时间并选其中最大的结果作为当前时间的最早发生时间。

一个时间的最迟发生时间为由它所发出的边（假设为 b）所指向的事件的最迟发生时间减去 b 这条边的权值，如果有多条边则逐一求出对应的时间并选其中最小的结果作为当前事件的最迟发生时间。

3) 根据 2) 中结果求出每个活动的最早发生时间和最迟发生时间。

4) 根据 3) 中结果找出最早发生时间和最迟发生时间相同的活动，即为关键活动：由关键活动所连成的路径即为关键路径。

0.42 00134-顺序查找法

算法要求：必须采用顺序存储结构。

基本思路：从表的一端开始，顺序扫描线性表，依次将扫描到的关键字和给定值 k 比较，若当前扫描的关键字与 k 相等，则查找成功；若扫描结束后，仍未有关键字与 k 相等，则查找失败。代码实现非常简单，如下：

```
int Search(int a[], int n, int k)
{
    int i;
    for(i=0; i<n; ++i)
        //顺序扫描线性表，查找成功则返回下标
        if(a[i]==k)
            return i;
    return 0;    //查找失败返回0
}
```

0.43 00135-分块查找

1. 概念描述

分块查找又称为索引顺序查找，其数据结构可以简单描述为：分块查找把线性表分成若干块，每一块中元素的存储顺序是任意的，但是块与块之间必须按照关键字大小有序排列，即前一块中的最大关键字要小于后一块中的最小关键字。对顺序表进行分块查找需要额外建立一个索引表，表中的每一项对应线性表中的一块。索引表的定义如下：

```
typedef struct indexElem{
    int key; //假设表内的元素为int型
    int low,high; //记录某块中第一个和最后一个元素的位置
} indexElem;
indexElem index[maxSize];
//定义索引表，maxSize是已经定义的常量
```

2. 算法描述

分块查找算法非常简单，可以分为两步进行，首先确定待查找的元素属于哪一块，然后在块内精确查找该元素。由于索引表是递增有序的，因此第一步采用二分查找。块内元素一般个数较少，因此第二步采用顺序查找即可。

分块查找实际上进行**两次查找**，整个算法的平均查找长度是两次查找的平均查找长度之和，即二分查找平均查找长度 + 顺序查找平均查找长度。

0.44 00136-折半查找法

算法要求：

必须采用顺序存储结构；

必须按关键字大小有序排列。

基本思路：首先，假设表中元素是按升序排列，将表中间位置记录的关键字与查找关键字比较，如果两者相等，则查找成功；否则利用中间位置记录将表分成前、后两个子表，如果中间位置记录的关键字大于查找关键字，则进一步查找前一子表，否则进一步查找后一子表。重复以上过程，直到找到满足条件的记录，使查找成功，或直到子表不存在为止，此时查找不成功。对应代码也并不难，如下：

```
int Bsearch(int R[], int Min, int Max, int k)
{
    int mid;
    while(Min<=Max)      // [Min, Max] 为查找范围
    {
        mid=(Min+Max)/2; // 求中间元素下标
        if(R[mid]==k)    // 中间元素等于要查找的值，查找成功，返回下标
            return mid;
        else if(R[mid]<k) // 查找的值大于中间元素，则进一步查找后一子表
            Min=mid+1;    // 改变查找范围 [mid+1, Max]
        else               // 查找的值小于中间元素，则进一步查找前一子表
            Max=mid-1;    // 改变查找范围 [Min, mid-1]
    }
    return 0;             // 查找失败，返回0
}
```

0.45 00137-B-树

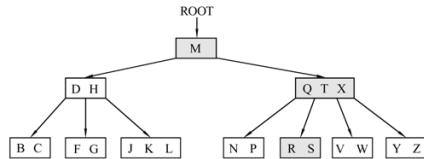
一棵 m 阶 B 树是一棵平衡的 m 叉查找树，深度为 m。它或者是空树，或者是满足下列性质的树：

- 1) 根结点至少有两个子女。
- 2) 每个非根结点所包含的关键字个数 j 满足： $\lceil m/2 \rceil - 1 \leq j \leq m - 1$ 。
- 3) 除根结点以外的所有结点（不包括失败结点）的度数正好是关键字总数加 1，故内部结点个数 k 满足： $\lceil m/2 \rceil \leq k \leq m$ 。
- 4) 所有的失败结点都位于同一层。

注意，B-树或者 B_树，不要误解成“B 减树”，“-”“_”只是连字符。

1. B-树的查找

首先把根结点取来，在根结点所包含的关键字 K_1, \dots, K_j 查找给定的关键字（因为关键字是有序的，故可用顺序查找或二分查找法），若找到等于给定值的关键字，则查找成功；否则，一定可以确定要查的关键字在某个 K_i 或 K_{i+1} 之间，于是取 P_i 所指的结点继续查找，直到找到，或指针 P_i 为空时查找失败。如下图所示（搜索关键字 R 的过程，黑色结点为被检查的结点）。



2. B-树的插入

对于关键字的插入，需要找到插入位置。在 B-树的查找过程中，当遇到空指针则证明查找不成功，同时也找到了插入位置，即根据空指针可以确定在叶结点中的插入位置。B-树的插入总是发生在叶子结点上。在插入过程中有可能破坏 B-树的特性，比如新关键字插入使得结点中关键字的个数超出规定个数，这时要进行结点的拆分。

3. B-树的删除

1) 删掉叶结点上的关键字, 有 3 种情况:
 ① 直接删除关键字。
 ② 兄弟够借: 把该兄弟结点中的关键字移到双亲结点中, 同时把双亲结点中关键字下移到关键字的结点中。
 ③ 兄弟不够借: 需要把兄弟结点以及双亲结点中分割二者的关键字合并成一个结点。如果双亲结点中关键字也少于 $\lceil m/2 \rceil$, 对双亲结点也做同样处理。如此反复, 可能直到对根结点做这样的处理, 从而使 B 树减少一层。
 2) 删掉的关键字不在叶子结点上, 需要将其转化成在叶子结点上的情况。
 若要删除非叶子结点上的关键字 a , 则需要找出其相邻关键字 (即 a 左指针树中最大的值, 或者 a 右指针树中最小的值), 下面以左指针树为例。
 沿着 a 的左指针来到其子树根结点, 然后沿着最优关键字的右指针, 找到叶子结点上的关键字 d 。用 d 来取代 a , 然后按照 1) 中的方法删除 d 即可。

0.46 00138-B+ 树

B+ 树是应文件系统所需而出的一种 B-树的变型树。一棵 m 阶的 B+ 树和 m 阶的 B-树的差异在于:

1. 有 n 棵子树的结点中含有 n 个关键字。
2. 所有的叶子结点中包含了全部关键字的信息, 及指向含这些关键字记录的指针, 且叶子结点本身依关键字的大小自小而大顺序链接。
3. 所有的非终端结点可以看成是索引部分, 结点中仅含其子树 (根结点) 中的最大 (或最小) 关键字。

0.47 00139-散列表

1. 概念

散列表就是根据给定的关键字来计算出关键字在表中的地址。

2. 构造

常用的 Hash 函数的构造方法

(1) 直接定址法

即 $H(key)=key$ 或 $H(key)=a*key+b$

(2) 数字分析法

假设关键字是 r 进制, 并且 Hash 表中可能出现的关键字都事先知道, 则去关键字的若干位数组成散列地址。

(3) 平方取中法

取关键字平方后的中间几位为散列地址。

(4) 除留余数法

即 $H(key)=key \bmod p$, 一般 p 选择小于或者等于表长的最大素数, 这样可以减少冲突。

3. 冲突处理

冲突: 因为 Hash 函数的关系, 常常会发生多个关键字共用一个地址的情况, 这种情况就称为冲突。冲突处理就是为共用一个地址的关键字找到另一个“空”的散列地址。

(1) 开放定址法

$H_i = (H(key)+d_i) \bmod m$ $i=1, 2, \dots, k$ ($k \leq m-1$)

其中: $H(key)$ 为散列函数; m 为散列表表长; d_i 为增量序列, 可能有以下 3 种取法。

1) $d_i=1, 2, 3, \dots, m-1$, 称线性探测再散列。

2) $d_i=1^2, -1^2, 2^2, -2^2, 3^2, \dots, \pm k^2$, ($k \leq m/2$) 称为二次探测再散列。

3) $d_i=伪随机数序列$, 称为伪随机探测再散列。

(2) 链地址法

在链地址法中, 把散列到同一槽中的所有元素都放在一个链表中 (类似图的邻接表的结构)。

4. 性能分析

1) 虽然散列表在关键字和记录的存储位置之间建立了直接映像，但由于“冲突”的产生，使得散列表的查找过程仍然是一个给定值和关键字进行比较的过程。因此，仍需以平静查找长度为衡量散列表的查找效率的度量。

2) 查找过程中需和给定值进行比较的次数取决于下列 3 个因素：①散列函数；②散列处理方法；③散列表的装填因子（关键字个数与表长度的比例）。

3) 线性探测再散列在处理冲突的过程中易产生记录的二次聚集，如使散列地址不相同的记录又产生新的冲突；而链地址法处理冲突不会发生类似的情况，因为散列地址不同的记录在不同的链表中。

4) 最坏情况下，在散列表中，查找一个元素的时间与在链表中查找一个元素的时间相同。但在实践中，散列技术的效率是很高的，在一些合理的假设下，在散列表中查找一个元素的期望时间为 $O(1)$ 。

0.48 00140-平均查找长度 ASL

查找运算的主要操作是关键字的比较，所以通常把查找过程中对关键字需要执行的平均比较次数（也称为平均查找长度）作为衡量一个查找算法效率优劣的标准。

平均查找长度 ASL (Average Search Length) 定义为： $ASL = P_1C_1 + P_2C_2 + \dots + P_nC_n$

其中：

(1) n 是节点的个数；

(2) P_i 是查找第 i 个节点的概率。若不特别声明，认为每个节点的查找概率相等，即 $P_1=P_2=\dots=P_n=1/n$ ；

(3) C_i 是找到第 i 个节点所需进行的比较次数。

0.49 00141-各种查找算法分析

顺序查找

(1) 顺序查找的优点：算法简单，且对表的结构无任何要求，无论是用向量还是用链表来存放节点，也无论节点之间是否按关键字有序，它都同样适用。

(2) 顺序查找的缺点：查找效率低，因此，当 n 较大时不宜采用顺序查找。

(3) 等概率情况下平均查找长度 $ASL = (n+1)/2$ ，查找不成功的平均查找长度为 n 。

分块查找

分块查找首先查找索引表。因为索引表是有序表，所以我们可以采用二分查找或顺序查找，以确定待查找的节点在哪一块。一般来说，如果题目中没有给定是什么查找方式，默认采用顺序查找。然后，在已确定的块中进行顺序查找。这是因为在快中的数据是无序的。分块查找是两次查找过程。整个查找过程的平均查找长度是两次平均查找长度之和。

折半查找

折半查找的平均查找长度如下：

(1) 查找成功时的平均查找长度：设内部节点的总数为 $n=2^h-1$ ，则判定树是深度为 $h=\log_2(n+1)$ 的满二叉树（深度 h 不计外部节点）。树中第 k 层上的节点个数为 2^{k-1} ，查找它们所需的比较次数为 k ，因此在等概率假设下，二分查找成功时的平均查找长度为： $ASL_{bn}=\log_2(n+1)-1$ 。

(2) 查找失败时的平均查找长度：二分查找在查找失败时所需比较的关键字个数不超过判定树的深度，在最坏的情况下查找成功的比较次数也不超过判定树的深度，即为： $[\log_2(n+1)]$ 。

二分查找的最坏性能和平均性能相当接近。

B-树

(1) B-树中节点的关键字的个数要比节点的儿子数少 1，一定不能将两者混淆。比如，除根之外的所有非终端节点至少有 $[m/2]$ 棵子树，则说明除根之外的所有非终端节点至少有 $[m/2]-1$ 个关键字。

(2) B-树最重要的特征就是平衡性，通常被用来设计磁盘的访问策略。

哈希表

(1) 设 m 和 n 分别表示表长和表中填入的节点数，则将 $a=n/m$ 定义为散列表的装填因子 (Load Factor)。 a 越大，表越满冲突的机会也越大。通常取 $a \leq 1$ 。

(2) 散列表解决冲突的方法通常有开放定址法和链地址法。链地址法会消耗额外的存储空间，如果把这些额外的空间用来加长散列表，会减小装填因子，从而加快查找速度。但是开放定址法对哈希表的删除操作支持不好。

(3) 散列表的平均查找长度不是结点个数 n 的函数，而是装填因子 a 的函数。因此在设计散列表时可选择 a 以控制散列表的平均查找长度。

0.50 00142-排序的基本概念

定义：所谓排序，即将原来一个无序的序列重新排列成有序的序列。注意这个序列中的每一项可能是单独的数据元素，也可能是一条记录（记录是由多个数据元素组成的，例如一个学生记录就是由学号、姓名、年龄、专业等数据元素组成的）。

0.51 00146-直接插入排序

插入排序的工作机理与很多人打牌时，整理手中牌时的做法差不多。每次摸牌后将它插入到左手一把牌中的正确位置。为了找到这张牌的正确位置，要将它和手中的每一张牌从右到左进行比较。

算法描述：每趟将一个待排序的关键字，按照其关键字值的大小顺序查找到适当位置，完成插入，直到待排序的关键字为空。代码如下：

```
void InsertSort(int R[], int n)
{
    int i, j, temp;
    for(i=2; i<=n; ++i)      //挨个取代排序的关键字
    {
        temp=R[i];           //将待排序的元素暂存于temp中
        j=i-1;                //让前面有序序列有后移一位的空间
        while(j>=1 && temp<R[j]) //扫描前面有序序列，若比temp大，则后移一位
        {
            R[j+1]=R[j];
            --j;
        }
        R[j+1]=temp;          //找到插入位置，插入temp值
    }
}
```

算法分析：直接插入排序的比较次数取决于原记录序列的有序程度。如果原始记录的关键字正好为递增顺序时，比较次数最少为 $n-1$ 次；如果为递减顺序时，比较次数最多，为 $(n-1)(n+2)/2$ 次，因此，直接插入排序的时间复杂度为 $O(n^2)$ 。

0.52 00147-折半插入排序

折半插入排序，跟直接插入排序差不多，举整理牌的例子，它只不过是找牌的插入位置方法不一样（其实也就是用了折半查找）。他直接看左手中间的牌，若比中间牌大，则再看中间牌到右端中间的一张牌，如此反复，直到找到插入位置。

算法描述：每趟将一个待排序的关键字，按照其关键字值的大小折半查找到适当位置，完成插入，直到待排序的关键字为空。

```

void BInsertSort(int R[],int n)
{
    int i,j,low,high,mid,temp;
    for(i=2;i<=n;i++)
    {
        temp=R[i];
        low=0;
        high=i-1;
        while(low<=high)          //折半查找
        {
            mid=(low+high)/2;
            if(r[0]<r[mid])
                high=mid-1;        //插入点在前半区
            else
                low=mid+1;        //插入点在后半区
        }
        for(j=i-1;j>=low;--j)
            r[j+1]=r[j];           //记录后移
        r[low]=temp;               //插入
    }
}

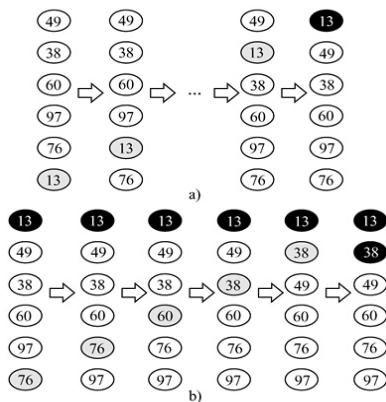
```

算法分析：折半插入排序的比较次数比直接插入排序的少，而移动次数相同，因此，总的时间复杂度仍为 $O(n^2)$ ，另外，折半插入排序也是一种稳定的排序方法。

0.53 00148-起泡排序

将被排序的记录数组 $R[1..n]$ 垂直排列，每个记录 $R[i]$ 看作是重量为 $R[i].key$ 的气泡。根据轻气泡不能在重气泡之下的原则，从下往上扫描数组 R ：凡扫描到违反本原则的轻气泡，就使其向上“飘浮”。如此反复进行，直到最后任何两个气泡都是轻者在上，重者在下为止。

如下图所示。



解释如下：上图中的 a 和上图中的 b 分别是第一趟起泡和第二趟起泡的过程。白色和灰色的都是无序的起泡，灰色是当前向上浮动的气泡。黑色为已经有序的气泡。当向上浮动的起泡遇到比它重的气泡时，就跟大气泡交换位置。若遇到比它轻的气泡时，轻气泡就变成灰色气泡（向上浮动的气泡）。当某一轮起泡，都没有发生气泡交换位置的情况时，说明起泡排序结束。

代码如下：

```

Void BubbleSort(int R[],int n)
int i,j,flag,temp;
for(i=0;i<n;++i)//第i趟起泡的终点位置的下标为i
{
    flag=0; //一趟排序是否发生交换的标记位，没有为0，有为1
    for(j=n;j>=i+1;--j) //每趟起泡都从最底下的元素开始起泡
    //直到j-1到达位置i，即j==i+1，故结束条件为j>=i+1
        if(R[j-1]>R[j])
            //遇到比自己重的气泡，交换之，修改标记位为1
        {
            temp=R[j-1];
            R[j-1]=R[j];
            R[j]=temp;
            flag=1;
        }
    if(flag==0)
        //某一轮起泡，没有发生交换，说明序列有序，起泡结束
        break;
}
}

```

算法分析：最好情况下的时间复杂度为 $O(n)$ ，最坏情况下的时间复杂度为 $O(n^2)$ 。空间复杂度为 $O(1)$ 。

0.54 00149-简单选择排序

算法描述：通过 $n-1$ 次关键字间的比较，从 $n-i+1$ 个记录中选出关键字最小的记录，并和第 i ($1 \leq i \leq n$) 个记录进行交换。

```

void SelectSort(int R[],int n)
{
    int i,j,k;
    int temp;
    for(i=0;i<=n;++i)
    {
        k=i;
        for(j=i+1;j<=n;++j)
            //寻找关键字最小的记录，并用k记录其下标
            if(R[k]>R[j])
                k=j;
        temp=R[i];           //交换R[i]和R[k]的值
        R[i]=R[k];
        R[k]=temp;
    }
}

```

算法分析：时间复杂度为 $O(n^2)$ ，空间复杂度为 $O(1)$ 。

0.55 00150-希尔排序

希尔排序是对直接插入排序进行改进后提出来的，又称缩小增量排序。其基本思想是：不断地把待排序的一组记录按间隔值分成若干小组，分别进行组内直接插入排序，这个规则就是增量。举例如下。

初始关键字: 47 55 10 40 15 94 5 70

第一趟排序, 增量为 4, 分四组, 组内采用直接插入排序法

15	47
55	94
5	10
40	70

第二趟排序, 增量为 2, 分两组, 组内采用直接插入排序法

5	10	15	47
40	55	70	94

第三趟排序, 增量为 1, 分一组, 组内采用直接插入排序法

5	10	15	40	47	55	70	94
---	----	----	----	----	----	----	----

算法分析: 希尔排序速度一般要比直接插入排序快。希尔排序的平均时间复杂度为 $O(n \log_2 n)$ 。希尔排序是不稳定的排序方法（即相同值的元素，排序前后的顺序发生变化了）。

0.56 00151-快速排序

基本思想:

- 先从数列中取出一个数作为基准数。
- 分区过程, 将比这个数大的数全放到它的右边, 小于或等于它的数全放到它的左边。
- 再对左右区间重复第 2 步, 直到各区间只有一个数。

代码如下:

```
void QuickSort(int R[], int l, int r)
{
    if(l < r)
    {
        int temp;
        int i=l, j=r;
        temp=R[i];
        //通过temp挖出了一个空缺位置R[i], temp暂时保存该值
        while(i < j)
        {
            while(i < j && R[j] >= temp)
                //找到右边第一个比temp值小的数, j保存其下标
                j--;
            if(i < j)
                R[i++]=R[j];      //将其值填入空缺位置R[i], i后移
                //同时产生新的空缺位置R[j]
            while(i < j && R[i] < temp)//找到左边第一个比temp值大的数, i保存其下标
                i++;
            if(i < j)
                R[j--]=R[i];      //将其值填入空缺位置R[k],
                //j后移
                //又产生新的空缺位置R[i]
        }
        R[i]=temp;
        QuickSort(R, l, i-1); //递归对左段序列进行排序
        QuickSort(R, i+1, r); //递归对右段序列进行排序
    }
}
```

算法分析: 快速排序的时间复杂度为 $O(n \log_2 n)$, 快排在同为 $O(n \log_2 n)$ 的几种排序方法中效率较高, 快排的空间复杂度为 $O(\log_2 n)$, 因为是递归函数, 需要栈的辅助, 分析不会考。

0.57 00152-堆排序

堆是一种数据结构, 可以把堆看成是一个完全二叉树, 这个完全二叉树满足: 任何一个非叶子结点的值, 都不大于(或小于)其左右孩子结点的值, 称之为小顶堆(或大顶堆)。

将无序序列调整成一个堆后, 就可以找出这个序列的最小(或最大)值, 然后将其交换到序列的最前(或最后), 这样这个被交换的最值就排好序了。对新的无序序列重复这样的操作, 就实现了排序。因此堆

排序中最关键的操作是建堆和调整堆。之前认真看二叉树存储的考生，应该还记得完全二叉树适合用一个数组来存储，而且左右孩子的下标都是可以直接根据父结点下标算出来的。

算法分析：时间复杂度为 $O(n \log_2 n)$ ，空间复杂度为 $O(1)$ 。

0.58 00153-二路归并排序

归并（Merge）排序法是将两个（或两个以上）有序表合并成一个新的有序表，即把待排序序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列。

举例如下：

初始状态：6, 202, 100, 301, 38, 8, 1

第一趟两两归并：{6, 202}, {100, 301}, {8, 38}, {1}

第二趟两两归并：{6, 100, 202, 301}, {1, 8, 38}

第三趟两两归并：{1, 6, 8, 38, 100, 202, 301}

该算法代码较复杂，不需要掌握。

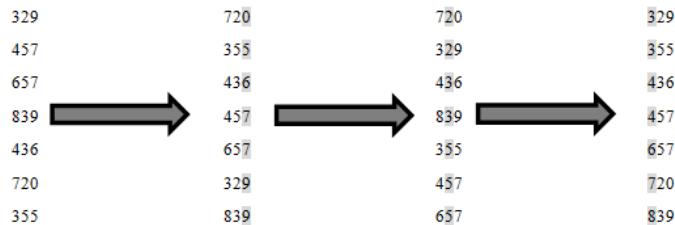
算法分析：归并排序时间复杂度和初始序列无关，故平均/最好/最坏时间复杂度都为 $O(n \log_2 n)$ 。

0.59 00154-基数排序

基数排序（Radix sort）是一种非比较型整数排序算法，其原理是将整数按位数切割成不同的数字，然后按每个位数分别比较。由于整数也可以表达字符串（比如名字或日期）和特定格式的浮点数，所以基数排序也不是只能使用于整数。

将所有待比较数值（正整数）统一为同样的数位长度，数位较短的数前面补零。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后，数列就变成一个有序序列。

基数排序的方式可以采用 LSD（Least significant digit）或 MSD（Most significant digit），LSD 的排序方式由键值的最右边开始，而 MSD 则相反，由键值的最左边开始。如图 7-4 所示。



关于这个算法，很重要的一点就是按位排序更稳定。否则排序会出错。

算法分析：给定 n 个 d 位数，每一个数位可以取 k 种可能的值，如果所用的稳定排序需要 $O(n+k)$ 的时间。则平均和最坏时间复杂度都为 $O(d(n+k))$ 。空间复杂度为 $O(k)$ 。

0.60 00155-外部排序

1. 外部排序的基本概念

内排序就是在内存中完成的排序（如快速排序、堆排序、归并排序等）。区别于内排序，外部排序指的是大文件的排序，即待排序的记录存储在外存储器上，待排序的文件无法一次装入内存。外部排序最常用的算法是多路归并排序，分为两步：

将文件中的数据分段输入内存，用内排的方法进行排序，然后写回到外存；

对这些有序段，采用归并方法，在外存上形成整个文件的单一归并段。

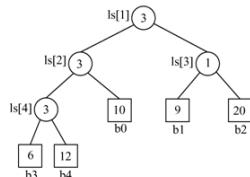
2. 最佳归并树

文件经过内排序后，得到的是长度不等的归并段，归并方案不同，所得归并树也就不同。在归并过程中，让记录少的初始归并段最先归并，记录数多的最晚归并，就可以建立总的 I/O 次数达到最少的最佳归并树（二路归并的话，其实就是用赫夫曼树的构造方法构造最佳归并树）。二路归并的缺点是扫描次数太多，因为如果有序段的个数为 m 个，那么二路归并的扫描次数就是 $\log_2 m$ ，采用多路归并可以减少扫描次数，一般情况下使用败者树来进行多路归并。之所以引入败者树，是因为在 k 路归并中，若不使用败者树，则每次对 k 路需要比较 $k-1$ 次得到最值，引入败者树之后，只需要 $\log_2(k)$ 次即可。

胜者树和败者树都是完全二叉树，是树形选择排序的一种变形。每个叶子结点相当于一个选手，每个中间结点相当于一场比赛，每一层相当于一轮比赛。不同的是，胜者树的中间结点记录的是胜者的标号；而败者树的中间结点记录着败者的标号。胜者树与败者树可以在 $\log_2(n)$ 的时间内找到最值。任何一个叶子结点的值改变后，利用中间结点的信息，还是能够快速地找到最值。

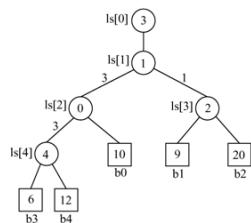
3. 胜者树

胜者树的一个优点是，如果一个选手的值改变了，可以很容易地修改这棵胜者树。只需要沿着从该结点到根结点的路径修改这棵二叉树，而不必改变其他比赛的结果。如下图所示。



4. 败者树

败者树是胜者树的一种变体。在败者树中，用父结点记录其左右子结点进行比赛的败者，而让胜者参加下一轮的比赛。败者树的根结点记录的是败者，需要加一个结点来记录整个比赛的胜利者。败者树的重构跟胜者树是不一样的，败者树的重构只需要与其父结点比较。因此，采用败者树可以简化重构的过程（重构过程不需要掌握）。如下图所示。



0.61 00156-各种排序算法的比较

1. 复杂度总结

排序方法	平均时间	最坏情况	空间复杂度
起泡排序			
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$
简单选择排序			
快速排序	$O(n \log_2 n)$	$O(n^2)$	$O(\log_2 n)$
堆排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(1)$
归并排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n)$
基数排序	$O(d(n+k))$	$O(d(n+k))$	$O(k)$

2. 稳定性

不稳定的排序方法有：快速排序、希尔排序、简单选择排序、堆排序。其余排序方法都是稳定的。

3. 一趟排序确定一个元素位置的排序方法

凡是每趟产生的有序区为全局有序区的排序方法，则每一趟排序结束时都至少能够确定一个元素的最终位置，这样的排序方法有简单选择排序、堆排序和快速排序。快速排序尽管不是每趟都产生有序区，但它将基准元素放在最终位置上。

4. 元素比较次数与原始序列无关的排序方法

简单选择排序和折半插入排序。

0.62 00275-检错编码和纠错编码

0.62.1 1. 检错编码

通过一定的编码和解码，能够在接收端解码时检查出传输的错误，但不能纠正错误。常见的检错编码有奇偶校验码和循环冗余码（CRC）。

1) **奇(偶)校验码**。添加一位校验码后，使得整个码字里面 1 的个数是奇(偶)数。接收端收到数据后就校验一下数据里 1 的个数，如果正好为奇(偶)数，则认为传输没有出错；如果检测到偶(奇)数个 1，则说明传输过程中，数据发生了改变，要求重发。

注意：当数据中有一位数据发生改变，通过奇偶校验能够检测出来，但并不知道是哪个位出错了；如果数据中同时有两位数发生了改变，此时奇偶校验是检测不到数据出错的，所以它的查错能力有限。

2) **循环冗余码**。循环冗余校验码是通过除法运算来建立有效信息位和校验位之间的约定关系。假定，待编码的有效信息以多项式 $M(x)$ 表示，将它左移若干位后，用另一个约定的多项式 $G(x)$ 去除，所产生的余数就是校验位。有效信息位与校验位相拼接就构成了 CRC 码，如图 3-3 所示。当接收方收到发来的 CRC 码后，仍用约定的多项式 $G(x)$ 去除，若余数为 0，表明该代码接收无误；若余数不为 0，表明某一位出错，再进一步由余数值确定出错的位置，并予以纠正。



图 3-3 循环冗余校验码的格式

2. 纠错编码

就是在接收端不但能检查错误，而且能纠正检查出来的错误。常见的纠错编码是海明码。海明码又称为汉明码，它是在信息字段中插入若干位数据，用于监督码字里的哪一位数据发生了变化，具有一位纠错能力。

(1) 海明码的编码过程

接起来即可，最后可得如下公式（按照表 3-1 的数据）：

$$c_1 = M_1 \oplus M_3 \oplus M_5 \oplus M_7 \oplus M_9$$

$$c_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_5 \oplus M_{10}$$

$$c_3 = M_1 \oplus M_2 \oplus M_3 \oplus M_7$$

$$c_4 = M_4 \oplus M_5 \oplus M_{10}$$

然后将表 3-1 中求出的数据对应过来，即

$$c_1 = P_1 \oplus D_1 \oplus D_2 \oplus D_3$$

$$c_2 = P_2 \oplus D_1 \oplus D_2 \oplus D_4$$

$$c_3 = P_3 \oplus D_2 \oplus D_3 \oplus D_4$$

$$c_4 = P_4 \oplus D_3 \oplus D_6$$

如果海明码没有错误信息， c_1 、 c_2 、 c_3 、 c_4 都为 0，等式右边的值也得为 0。

由于是异或，所以 P_i (i 取 1, 2, 3, …) 的值跟后边的式子必须一样才

能使整个式子的值为 0，即

$$P_1 = D_1 \oplus D_2 \oplus D_3 \oplus D_5$$

$$P_2 = D_1 \oplus D_2 \oplus D_4 \oplus D_6$$

$$P_3 = D_2 \oplus D_3 \oplus D_4$$

$$P_4 = D_3 \oplus D_6$$

下面只需要将值代入计算即可，

$$P_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

$$P_3 = D_2 \oplus D_3 \oplus D_4 = 0 \oplus 1 \oplus 1 = 0$$

$$P_4 = D_3 \oplus D_6 = 0 \oplus 1 = 1$$

4) 最后将 P_i 填入表 3-1 中，海明码就形成了。

(2) 校验海明码的过程

1) 直接写出纠错位 e_1, \dots, e_m 与 M_1, \dots, M_n 的对应关系，计算出

e_1, \dots, e_m 的值。

2) 求出二进制序列 e_m, \dots, e_1 对应十进制的值，则此十进制数就是出

错的位数，取反即可得到正确的编码。

0.63 03686-计算机网络的功能

1. **数据通信：**是计算机网络最基本和最重要的功能，包括连接控制、传输控制、差错控制、流量控制、路由选择、多路复用等子功能。
2. **资源共享：**包括数据资源、软件资源以及硬件资源。
3. **分布式处理：**当计算机网络中的某个计算机系统负荷过重时，可以将其处理的任务传送给网络中的其他计算机系统进行处理，利用空闲计算机资源提高整个系统的利用率。
4. **信息综合处理：**将分散在各地计算机中的数据资料进行集中处理或分级处理，如自动订票系统、银行金融系统、数据采集与处理系统等。
5. **负载均衡：**将工作任务均衡地分配给计算机网络中的各台计算机。
6. **提高可靠性：**计算机网络中的各台计算机可以通过网络互为替代机。

当然，为了满足人们的学习、工作和生活需要，计算机网络还有其他一些功能，如远程教育、电子化办公与服务、娱乐等。

0.64 03687-计算机网络的分类

1. 按**分布范围**分类：广域网、城域网、局域网、个人区域网。
2. 按**拓扑结构**分类：星形网络、总线型网络、环形网络、树状形网络、网状形网络。
3. 按**传输技术**分类：广播式网络、点对点网络。
4. 按**使用者**分类：公用网、专用网。
5. 按**数据交换技术**分类：电路交换网络、报文交换网络、分组交换网络。

注意：接入网（AN）了解即可！

0.65 03689-计算机网络分层结构

计算机网络为什么要采用分层结构？

这里用一个小的生活实例来解释。任何一个公司都是从小企业创办而来的，当公司规模很小（比如只有一个老总和3个员工）时，老总和员工可以同处于一个平面，不需要分层，员工可以直接向老总汇报问题。但是，如果该公司是诸如微软这样的公司（也就是计算机网络具有相当大的规模时），比尔·盖茨当然处于最高层，他的作用就是实现公司的长远发展，而不可能每天与公司的员工讨论某功能模块应该使用哪种算法。同理，当网络结构大时，就必须要有分层，并且每一层都需实现所对应的功能，这样才会有更好的发展。但是，分层又不能太多，如果分层太多，资源浪费就很多。所以，TCP/IP 折中地采用了4层结构模型（在教材中为了更好地描述各层的工作原理经常被看作5层）。

3个专业术语

1. **实体：**任何可发送或接收信息的硬件或软件进程，通常是一个特定的软件模块。
2. **对等层：**不同机器上的同一层。
3. **对等实体：**同一层上的实体。

理解方式：A省和B省分别表示不同的机器，可将A省和B省的各层干部看成实体，将A省省长职位和B省省长职位看成对等层，而将此对等层上的实体，即A省省长和B省省长，可看成对等实体。

0.66 03690-协议

协议是一种规则，并且是控制两个对等实体进行通信的规则，也就是水平的。

协议由以下3个部分组成：

1. **语义：**对构成协议元素的含义的解释，即“讲什么”。
2. **语法：**数据与控制信息的结构或格式，即“怎么讲”。
3. **同步：**规定了事件的执行顺序。

0.67 03691-接口

接口又被称为服务访问点，从物理层开始，每一层都向上层提供服务访问点，即没有接口就不能提供服务。

五个不得不知道的专业术语：

1. **服务数据单元 (SDU)** 第n层的服务数据单元，记作n-SDU。
2. **协议控制信息 (PCI)** 第n层的协议控制信息，记作n-PCI。
3. **接口控制信息 (ICI)** 第n层的接口控制信息，记作n-ICI。
4. **协议数据单元 (PDU)** 第n层的服务数据单元 (SDU) + 第n层的协议控制信息 (PCI) = 第n层的协议数据单元，即 $n\text{-SDU}+n\text{-PCI}=n\text{-PDU}$ ，表示的是同等层对等实体间传送的数据单元。另外， $n\text{-PDU}=(n-1)\text{-SDU}$ 。这个公式看完，后面的内容就会很清楚。例如，网络层的整个IP分组交到数据链路层，整个IP分组成为数据链路层的数据部分。
5. **接口数据单元 (IDU)** 第n层的服务数据单元 (SDU) + 第n层的接口控制信息 (ICI) = 第n层的接口数据单元，即 $n\text{-SDU}+n\text{-ICI}=n\text{-IDU}$ ，表示的是在相邻层接口间传送的数据单元。

0.68 03692-服务

服务指下层为相邻上层提供的功能调用。协议是水平的，而服务则是垂直的，即下层向上层通过接口提供服务。服务分为以下 3 类：

1. 面向连接的服务和面向无连接的服务

面向连接的服务：当通信双方通信时，要事先建立一条通信线路，该线路包括建立连接、使用连接和释放连接 3 个过程。TCP 协议就是一种面向连接服务的协议，电话系统就是一个面向连接的模式。

面向无连接的服务：通信双方不需要事先建立一条通信线路，而是把每个带有目的地址的包（报文分组）传送到线路上，由系统选定路线进行传输。IP 协议和 UDP 协议就是两种无连接服务的协议，邮政系统就是一个无连接的模式。

面向连接与面向无连接的对比见表 1-1。

服 务	优 点	缺 点
面向连接	可靠信息流、信息回复确认	占用通信信道
面向无连接	不占用通信信道	信息流可能丢失、信息无回复确认

2. 有应答服务与无应答服务

有应答服务：指接收方在收到数据后向发送方给出相应的应答。

无应答服务：指接收方收到数据后不自动给出应答。

3. 可靠服务与不可靠服务

可靠服务：指网络具有检错、纠错、应答机制，能保证数据正确、可靠地传送到目的地。

不可靠服务：指网络不能保证数据正确、可靠地传送到目的地，网络只能是尽量正确、可靠，是一种“尽力而为”的服务。

注意：并非在一个层内完成的全部功能都称为服务，只有那些能够被高一层实体“看得见”的功能才称为服务。

0.69 03693-ISO-OSI 参考模型和 TCP-IP 参考模型

1. 5 层结构的总结

OSI 参考模型具有 7 层结构，而 TCP/IP 模型仅有 4 层结构（一般看作 5 层）。在 OSI 参考模型中表示层和会话层不是重点，大致浏览一遍即可，无须深究，所以只需掌握 5 层结构即可。读者应该能快速地默写出 5 层结构以及每层所完成的任务、功能、协议（遇到选择题能选对即可），5 层参考模型各层的总结见表 1-2。

表 1-2 5 层参考模型各层的总结

应用层 (用户对用户)	任务: 提供系统与用户的接口 功能: ①文件传输; ②访问和管理; ③电子邮件服务 协议: FTP、SMTP、POP3、HTTP
传输层 (运输层) (应用对应用 进程对进程)	传输单位: 报文段 (TCP) 或用户数据报 (UDP) 任务: 负责主机中两个进程之间的通信 功能: ①为端到端连接提供可靠的传输服务; ②为端到端连接提供流量控制、差错控制、服务质量等管理服务 协议: TCP、UDP
网络层 (网际层、IP 层) (主机对主机)	传输单位: 数据报 所实现的硬件: 路由器 任务: ①将传输层传下来的报文段封装成组; ②选择适当的路由, 使传输层传下来的分组能够交付到目的主机 功能: ①为传输层提供服务; ②组包和拆包; ③路由选择; ④拥塞控制 协议: ICMP、ARP、RARP、IP、IGMP
数据链路层 (链路层)	传输单位: 帧 所实现的硬件: 交换机、网桥 任务: 将网络层传下来的 IP 数据报组装成帧 功能: ①链路连接的建立、拆除、分离; ②帧定界和帧同步; ③差错检测 协议: PPP、HDLC、ARQ
物理层	传输单位: 比特 所实现的硬件: 集线器、中继器 任务: 透明地传输比特流 功能: 为数据端设备提供传送数据通路

2. OSI 参考模型和 TCP/IP 模型的区别

OSI 参考模型和 TCP/IP 模型的特性对比见表 1-3。

表 1-3 OSI 参考模型和 TCP/IP 模型的特性对比

OSI 参考模型	TCP/IP 模型
① 3 个主要概念: 服务、接口、协议 ② 协议有很好的隐蔽性 ③ 产生在协议发明之前 ④ 共有 7 层 网络层: 连接和无连接 传输层: 仅有面向连接	① 没有明确区分服务、接口、协议 ② 产生在协议发明之后 ③ 共有 4 层 (不是 5 层) 网络层: 仅有无连接 传输层: 面向连接和无连接

3. 会话层与表示层的基本功能

1. 会话层。会话层的主要功能是在两个结点间建立、维护和释放面向用户的连接，并对会话进行管理和控制，保证会话数据可靠传送。

2. 表示层。负责处理在两个内部数据表示结构不同的通信系统间交换信息的表示格式（数据格式转换，2013 年统考真题考查了此功能），为数据加密和解密以及为提高传输效率提供必需的数据压缩及解压等功能。

0.70 03694-计算机网络性能指标

1. 时延

时延是指数据从网络或链路的一端传送到另一端所需要的时间，有时也被称为延迟或迟延。网络时延由以下几部分组成。

发送时延 (或者称为传输时延): 主机或路由器发送数据帧所需要的时间，即从发送数据帧的第一位算起到该帧的最后一一位发送完毕所需要的时间。因此，发送时延也被称为传输时延。

$$\text{发送时延} = \text{数据帧长度 (bit)} / \text{发送速率 (bit/s)}$$

传播时延。 是指电磁波在信道中传播一定的距离所需要的时间。传播时延的计算公式为

$$\text{传播时延} = \text{信道长度 (m)} / \text{电磁波在信道上的传播速度 (m/s)}$$

处理时延。 是指主机或路由器在接收到分组时进行处理所需要的时间。

排队时延。 分组在进入网络传输时，要经过许多路由器，但分组在进入路由器后要先在输入队列中排队等待处理，在路由器确定了转发接口后，还需要在输出队列中排队等待转发，这就产生了排队时延。

$$\text{总时延} = \text{发送时延} + \text{传播时延} + \text{处理时延} + \text{排队时延}$$

2. 时延带宽积

时延带宽积又称为以比特为单位的链路长度。时延带宽积 = 传播时延 \times 带宽

3. 往返时间

从发送方发送数据开始，到发送方收到来自接收方的确认消息（接收方收到数据后便立即发送确认）总共经历的时间。

4. 利用率

包括信道利用率和网络利用率两种。

信道利用率指某信道有百分之几的时间是被利用的（有数据通过），完全空闲的信道的利用率为零。

网络利用率是全网络的信道利用率的加权平均值。但是需要注意一点，不是信道利用率与网络利用率越高越好，因为利用率越高，会导致数据在路由器中转发时延过长。

0.71 03704-信号

信号：数据的电气或电磁的表现（就是将数据用另外一种形态表现出来，就好像水转换成冰，其实质还是水，仅仅是形态变了）。而数据是传送信息（如图片和文字等）的实体。

注意 1：无论数据或信号，都既可以是模拟的，也可以是数字的。“模拟的”就是连续变化的，如图 2-1 所示；而“数字的”表示取值仅允许是有限的离散值，如图 2-2 所示。

注意 2：信道上传送的信号分为基带信号和宽带信号。基带信号是将数字信号 0 和 1 直接用两种不同的电压表示，然后传送到数字信道上去传输，称为基带传输；宽带信号是将基带信号进行调制后形成模拟信号，然后再传送到模拟信道上去传输，称为宽带传输。总之，记住一句话：基带对应数字信号，宽带对应模拟信号。

注意 3：宽带传输在考研中可以等同于频带传输（都是传输模拟信号），只是宽带传输比频带传输有更多的子信道，并且这些子信道都可以同时发送信号。

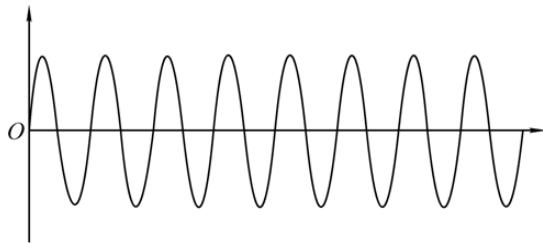


图 2-1 模拟信号

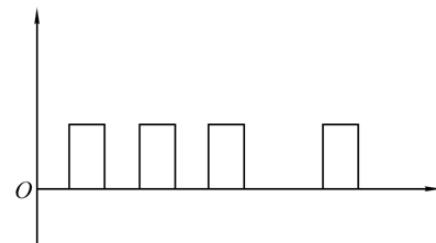


图 2-2 数字信号

0.72 03705-信源、信道及信宿

信源：字面理解就是信息的源泉，也就是通信过程中产生和发送信息的设备或计算机。

信道：字面理解就是信息传送的道路，也就是信号的传输媒质，分为有线信道和无线信道，人们常说的双绞线和人造卫星传播信号分别是有线信道和无线信道的典型代表。

信宿：字面理解就是信息的归宿地，也就是通信过程中接收和处理信息的设备或计算机。

故事助记：某公司要将货物从 A 地运送到 B 地（通过铁路），B 地把货物加工为成品销售给用户。这里的 A 地就是信源，铁路就是信道，B 地就是信宿，货物就是数据，货物加工成的成品就是信息。

信号、数据、信息三者的关系则是：比如在使用万用表时，输入（电）信号得到（电压/电流）数据，数据通过整理就是信息。

0.73 03706-速率、波特及码元

在计算机网络中，速率顾名思义是指数据的传输速率，即单位时间内传输的数据量。一般速率有两种描述形式：**波特率**和**比特率**。

1. 码元

在数字通信中常常用时间间隔相同的符号来表示一个二进制数字，这样的时间间隔内的信号称为二进制码元。

2. 波特率

又称为码元传输速率，它表示单位时间内数字通信系统所传输的码元个数（也可以称为脉冲个数或者信号变化的次数，对理解某些题有好处，一定记住！），单位是波特（Baud）。1 波特表示数字通信系统每秒传输 1 个码元。码元可以是二进制表示，也可以是多进制表示。

3. 比特率

又称为信息传输速率，它表示单位时间内数字通信系统所传输的二进制码元个数，即比特数，单位为 bit/s。

4. 比特率与波特率的关系

正常情况下，每比特只能表示两种信号变化（0 或 1），可看成二进制。此时每个码元只能携带 1bit 的信息，所以在数量上，波特率就和比特率相等了。因此，在二进制码元的情况下，比特率在数量上和波特率是相等的。但是，一个码元仅携带一个比特，数据率很低，所以编码专家想办法让一个码元携带更多的比特，以此来提高传输速率，即通过一些手段将信号的变化次数增加，从而让一个码元携带更多的比特，例如，增加到 16 种信号变化（可以看成十六进制），那么自然就需要 4bit ($\log_2 16 = 4$ ，记住这个公式！) 来表示，此时一个码元携带了 4bit，此时的波特率在数量上就是比特率的 4 倍。

0.74 03707-带宽

带宽分为模拟信号的带宽和数字信号的带宽。

在过去很长一段时间里，通信的主干线路传送的是模拟信号，此时带宽的定义为：通信线路允许通过的信号频带范围，就是允许通过的最高频率减去最低频率，例如某通信线路允许通过的最低频率为 300Hz，最高频率为 3400Hz，则该通信线路的带宽就为 3100Hz。

但是，在计算机网络中，带宽不是以上的定义。此时的带宽是用来表示网络的通信线路所能传送数据的能力。因此，带宽表示在单位时间内从网络中的某一点到另一点所能通过的“最高数据率”。显然，此时带宽的单位不再是 Hz，而是 bit/s，读作“比特每秒”。

0.75 03708-奈奎斯特定理

1. 采样定理

讲解带宽的时候提到，在通信领域带宽是指信号最高频率与最低频率之差，单位为 Hz。因此将模拟信号转换成数字信号时，假设原始信号中的最大频率为 f ，那么采样频率 f 采样必须大于或等于最大频率 f 的两倍，才能保证采样后的数字信号完整保留原始模拟信号的信息（只需记住结论，不要试图证明，切记！）。另外，采样定理又称为奈奎斯特定理。

2. 奈奎斯特定理

具体的信道所能通过的频率范围总是有限的（因为具体的信道带宽是确定的），所以信号中的大部分高频分量就过不去了，这样在传输的过程中会衰减，导致在接收端收到的信号的波形就失去了码元之间的清晰界限，这种现象叫做码间串扰。所以是不是应该去寻找在保证不出现码间串扰的条件下的码元传输速率的最大值呢？没错，这就是奈奎斯特定理的由来。奈奎斯特在采样定理和无噪声的基础上，提出了奈奎

斯特定理。奈奎斯特定理的公式为

$$C_{\max} = f_{\text{采样}} \times \log_2 N = 2f \times \log_2 N \text{ (bit/s)}$$

式中， f 表示理想低通信道的带宽； N 表示每个码元的离散电平的数目。

0.76 03709-香农定理

1. 信噪比

介绍香农定理之前需要引入一个概念，即**信噪比**。要清楚噪声的影响是相对的，也就是说，信号较强，噪声的影响就相对较小（两者是同时变化的，仅考虑两者之一是没有任何意义的），所以求信号的平均功率和噪声的平均功率之比（记为 S/N ，读作“信噪比”）才有意义，即

$$\text{信噪比 (dB)} = 10 \log_{10}(S/N) \text{ (dB)}$$

2. 香农公式

引入信噪比之后可得出香农公式，如下：

$$C_{\max} = W \times \log_2(1+S/N) \text{ (bit/s)}$$

其中， W 为信道的带宽，所以要想提高最大数据传输速率，就应设法提高传输线路的带宽或者设法提高所传信号的信噪比。

0.77 03710-编码与调制

模拟数据和数字数据都可以转换为模拟信号或数字信号。将模拟数据或数字数据（可统称为数据）转换为模拟信号的过程称为调制；将模拟数据或数字数据转换为数字信号的过程称为编码，如图 2-3 所示。

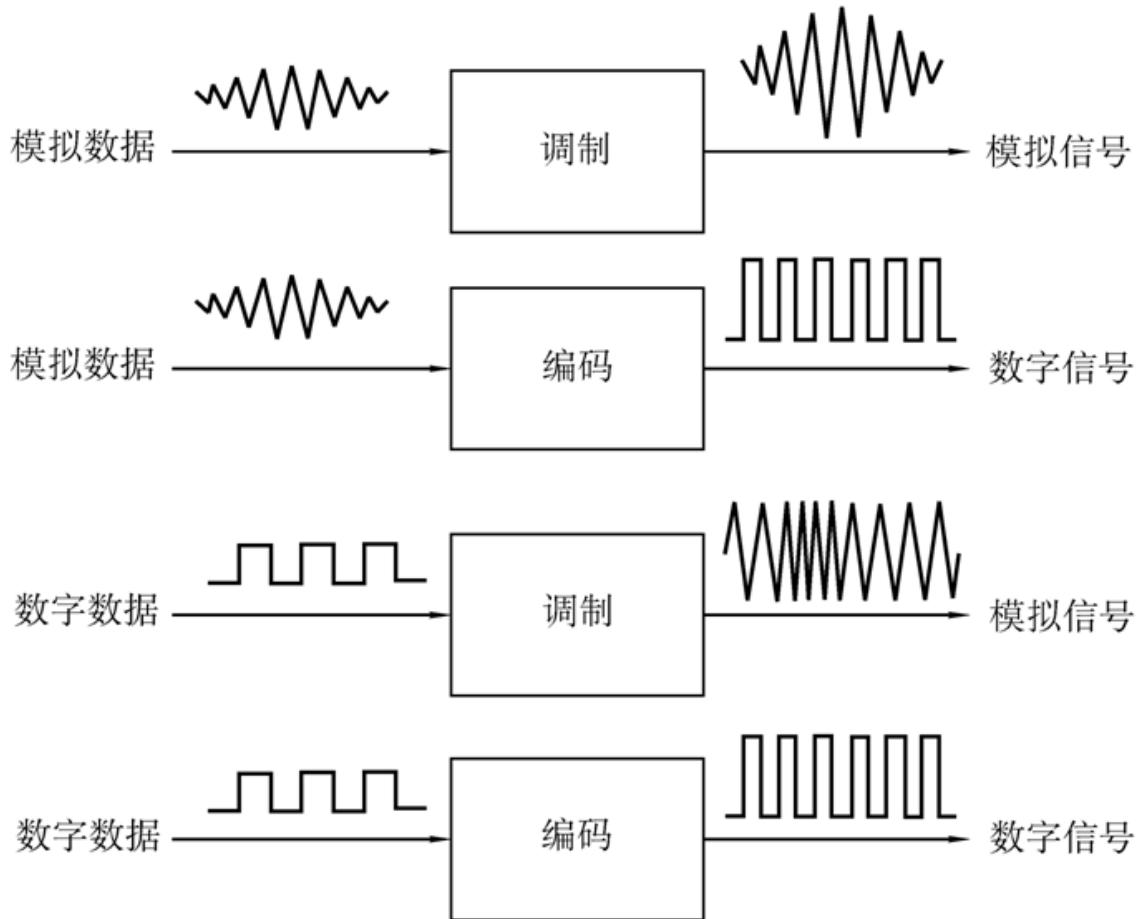


图 2-3 调制与编码

调制

一、数字数据调制为模拟信号

数字数据调制技术在发送端将数字信号转换为模拟信号，而在接收端将模拟信号还原为数字信号，分别对应于调制解调器的调制和解调过程。考研中理解这两种转换即可，其他的了解即可。

故事助记：调制解调器的调制是为了将数字数据转换成模拟信号，因为数字数据含有太多的低频成分（可以看成矮个子），而该信道不让他过去的原因有两种：

原因一：太矮了（都是低频成分），不让他过去。

原因二：他穿的衣服不适合该场合（低频成分不能与信道的特性相适应）。

针对以上两种原因，可以想出两种办法。

针对第一种原因：让他变高。

针对第二种原因：换件正式的西装。

这样就引入了两种调制，如下：

1. 带通调制（把矮个子变高）：类似于增高垫，让矮个子变高了，这样就可以过去了，即教材所讲的将基带信号的频率范围搬移到较高的频段以便在信道中传输由此引出了 3 种方式：调幅、调频和调相。
2. 基带调制（换件西装）：给基带信号的低频成分改变波形，使之适应信道的特性（也就是说给矮个子穿上西装，改变一下外表，使之适应这个场合）；但是穿上西装仍然是矮子，也就是说基带信号的低频成分改变波形仍然是基带信号，没有变成其他信号。

二、模拟数据调制为模拟信号

模拟数据调制为模拟信号主要有以下原因：

- 为了实现传输的有效性，可能需要较高的频率。
- 充分利用带宽。

编码

一、数字数据编码为数字信号

数字数据编码用于基带信号传输中，可以在基本不改变数字数据信号频率的情况下，直接传输数字信号，即直接让矮子过去，不用穿增高垫了。既然不用穿增高垫，那就必须穿西装过去，而现在西装又分很多种牌子（非归零码、曼彻斯特编码、差分曼彻斯特编码）。

1. 非归零码：用低电平表示 0，高电平表示 1；或者反过来。缺点是无法判断一个码元的开始和结束，收发双方难以保持同步。

2. 曼彻斯特编码：将每个码元分成两个相等的间隔。前一个间隔为高电平而后一个间隔为低电平表示码元 1；码元 0 正好相反。曼彻斯特编码的特点是将每个码元的中间跳变作为收发双方的同步信号，无需额外的同步信号；但它所占的频带宽度是原始的基带宽度的两倍。

3. 差分曼彻斯特编码：若码元为 1，则其前半个码元的电平与上一个码元的后半个码元的电平一样；若码元为 0，则其前半个码元的电平与上一个码元的后半个码元的电平相反。在每个码元的中间，都有一次电平的跳转。该编码技术较复杂，但抗干扰性较好。

二、模拟数据编码为数字信号

此编码最典型的例子就是脉冲编码调制。

脉冲编码调制：只需记住 3 个步骤，采样、量化和编码以及它是将模拟数据进行数字信号编码即可。

0.78 03711-数据传输方式

数据传输方式包括电路交换、报文交换和分组交换。

电路交换、报文交换和分组交换的数据传输方式如图 2-4 所示。

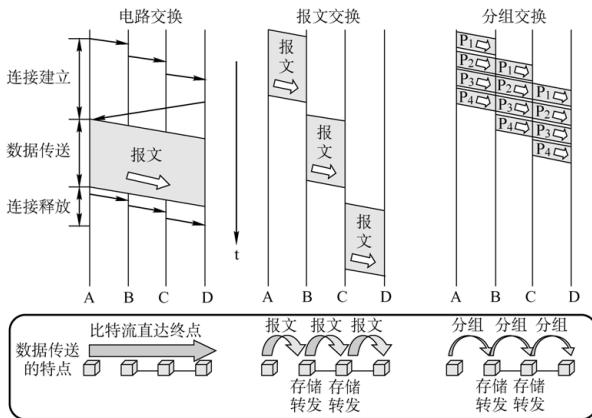


图 2-4 电路交换、报文交换和分组交换的数据传输方式

1. 电路交换

由于电路交换在通信之前要在通信双方之间建立一条被双方独占的物理通路，因此有以下优缺点。

优点：通信时延小、实时性强、有序传输，并且不同的通信双方拥有不同的信道，不会出现争用物理信道的问题。

缺点：建立连接时间长、信道利用率低、缺乏统一标准、灵活性差。

2. 报文交换

数据交换的单位是报文，报文携带有目标地址、源地址等信息。报文交换在交换结点采用存储转发的传输方式，因而有以下优缺点。

优点：无需建立连接、动态分配线路、可靠性高、线路利用率高、提供多目标服务。

缺点：由于数据进入交换结点后要经历存储、转发这一过程，从而引起转发时延（包括接收报文、检验正确性、排队、发送时间等）。报文交换对报文的大小没有限制，这就要求网络结点需要有较大的存储缓存空间。

3. 分组交换

分组交换仍采用存储转发传输方式，但将一个长报文先分割为若干个较短的分组，然后把这些分组（携带源、目的地址和编号信息）逐个地发送出去，因此分组交换除了具有报文的优点外，与报文交换相比有以下优缺点。

优点：加速传输、简化了存储管理、减少了出错几率和重发数据量。

缺点：存在传输时延、可能出现失序、丢失或重复分组。

综上，若要传送的数据量很大，且其传送时间远大于呼叫时间，则采用电路交换较为合适；当端到端的通路由很多段的链路组成时，采用分组交换传送数据较为合适。从提高整个网络的信道利用率上看，报文交换和分组交换优于电路交换，其中分组交换比报文交换的时延小，尤其适合于计算机之间的突发式的数据通信。

4. 电路交换与分组交换的特性比较

见表 2-1。

表 2-1 电路交换与分组交换的特性比较

比较标准	电路交换	分组交换
建立连接	要求	不要求
专用物理路径	是	否
每个分组沿着规定的路径	是	否
分组按序到达	是	否
路由器的瘫痪对整体产生影响	是	否
可用带宽	固定	动态
可能拥塞的时间点	建立呼叫连接的时候	每个分组传送的时候
可能有浪费的带宽	是	否
使用存储转发	否	是
透明性	是（信息以数字信号形式在数据通路中“透明”传输，交换机对用户的报文信息不存储、分析和处理）	否（每到一个路由器都要对分组首部进行分析，然后转发到下一个路由器）
收费	每分钟（打电话是按照分钟计算的，肯定不是说一句话付一句话的钱，从这个角度也可以推出打电话是电路交换）	每个分组（手机上网是按照流量算的，不是按分钟算的，从这个角度也可以推出因特网使用的也是分组交换）

0.79 03712-数据报与虚电路

分组交换可进一步分为面向连接的虚电路方式和无连接的数据报方式。

1. 数据报

如图 2-5 所示，假设主机 A 给主机 B 发送一个报文，高层协议会将报文拆分成若干个带有序号和完整目的地址的分组，交换机根据转发表转发分组。其原理如下：

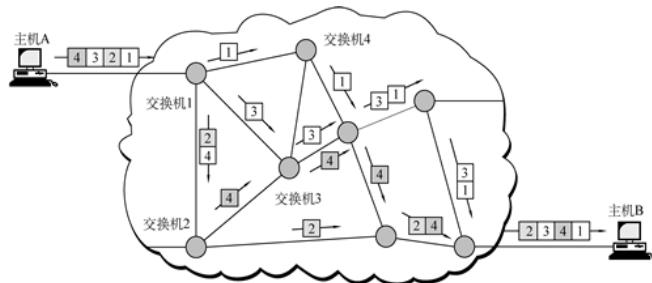


图 2-5 数据报方式转发分组

- 1) 首先主机 A 先将分组逐个地发往与它直接相连的交换机 1，交换机 1 将主机 A 发来的分组缓存。
- 2) 然后查找自己的转发表，不同时刻转发表的内容可能不相同，因此有的分组转发给交换机 2，有的分组转发给交换机 3 和交换机 4。

3) 依次类推, 直到所有分组到达主机 B。

由以上分析可知数据报方式具有以下特点:

- 1) 无需建立连接;
- 2) 网络尽最大努力交付, 传输不保证可靠性;
- 3) 减小了延迟, 大大提高吞吐量;
- 4) 对故障适应能力强;
- 5) 不独占某一链路, 资源利用率高。

2. 虚电路

虚电路方式要求在发送数据之前, 在源主机和目的主机之间建立一条虚连接。一旦虚连接建立以后, 用户发送的数据 (以分组为单位) 将通过该路径按顺序传送到目的主机。当通信完成之后用户发出释放虚电路请求, 由网络清除该虚连接。

以上描述是不是有一种似曾相识的感觉? 没错, 虚电路方式与电路交换方式极其相似。其实虚电路方式就是将数据报方式与电路交换方式结合起来, 充分发挥二者优点。由以上分析可知, 虚电路方式的通信过程分为 3 个阶段: 虚电路建立、数据传输与虚电路释放阶段。

如图 2-6 和图 2-7 所示, 假设主机 A 给主机 B 发送一个报文, 原理如下:

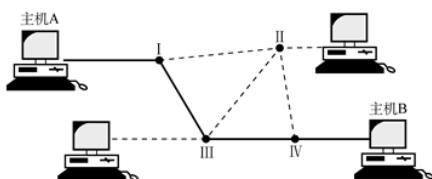


图 2-6 虚电路方式转发分组 (一)

1) 主机 A 先发出一个特殊的“呼叫请求”分组, 该分组通过中间交换机 (图 2-6 中的小圆点) 送往主机 B。如果同意连接, 主机 B 就发送“呼叫应答”分组进行确认, 虚电路就建立好了。

2) 虚电路建立之后, 主机 A 就可以向主机 B 发送分组了。由于所有分组都是走同样的路径, 因此分组一定按序到达目的主机。

3) 分组传输结束后, 主机通过发送“释放请求”分组以拆除虚电路, 整个连接就断开了。

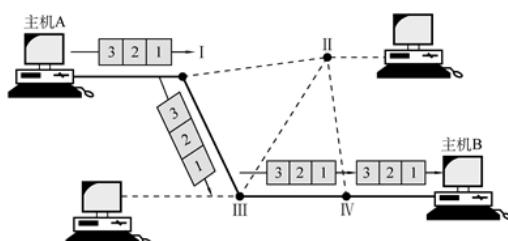


图 2-7 虚电路方式转发分组 (二)

由以上分析可知虚电路方式具有以下特点:

- 1) 通信必须建立连接;
- 2) 分组按序到达目的主机;
- 3) 开销小;
- 4) 当某个交换机或链路出现故障, 所有虚电路将遭到破坏。

3. 数据报服务与虚电路服务比较

见表 2-2。

表 2-2 数据报服务与虚电路服务的特性比较

比较标准	数据报服务	虚电路服务
连接的建立	不需要	需要
地址信息	每个分组包含完整的源地址和目的地址	每个分组包含一个虚电路号
状态信息	路由器不保留任何有关连接的状态信息	每个虚电路都要求路由器为每个连接建立表项
分组的转发	每个分组有独立的路径	当虚电路建立的时候选择路径，所有分组都沿着这条路径
路由器失效的影响	没有	所有经过此失效的路由器的虚电路都将终止
端到端的差错处理	由主机负责	由通信子网负责
端到端的流量控制	由主机负责	由通信子网负责
分组的顺序	到达目的站不一定按序	总是按发送顺序到达目的地
思路	可靠通信应当由用户主机来保证	可靠通信应当由网络层来保证

注：数据报服务和虚电路服务都由网络层提供。

0.80 03713-传输介质分类

1. 传输介质的分类

传输介质分为两大类：**导向性传输介质**（如双绞线和光纤）和**非导向性传输介质**（如红外线、微波）。

2. 双绞线

把两根互相绝缘的铜导线绞合起来。其特点是既可以传输模拟信号，又可以传输数字信号。双绞线又可分为**无屏蔽双绞线**和**屏蔽双绞线**。屏蔽双绞线就是在普通的双绞线外加上金属丝编织的屏蔽层，以起到提高抗电磁干扰的能力，如图 2-8 所示。

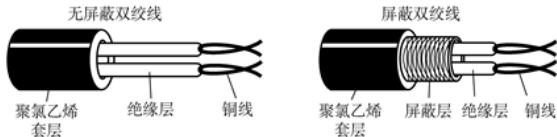


图 2-8 双绞线的结构

3. 同轴电缆

由内导体铜质芯线、绝缘层、网状编织的外导体屏蔽层以及保护塑料外层组成。它比双绞线的抗干扰能力强，因此传输距离更远。如图 2-9 所示。

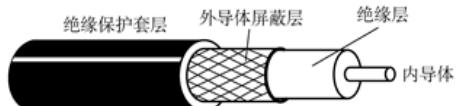


图 2-9 同轴电缆的结构

4. 光纤

即光导纤维，根据光线传输方式不同，光纤可分为**单模光纤**和**多模光纤**。其主要优点是频带宽、衰减小、速率高、体积小、抗雷电和电磁干扰性好、误码率低、质量轻、保密性好等。

单模光纤：直径只有一个光波的波长，光线在其中一直向前传播，不会发生多次反射，如图 2-10 所示。**单模光纤的光源使用的是昂贵的半导体激光器**，而不使用较便宜的发光二极管，因此**单模光纤的衰减较小，适合远距离传输**。



图 2-10 单模光纤

多模光纤：利用光的全反射特性，如图 2-11 所示。**多模光纤的光源为发光二极管**。由于光脉冲在多模光纤中传输会逐渐展宽，造成失真，因此**多模光纤只适合近距离传输**。

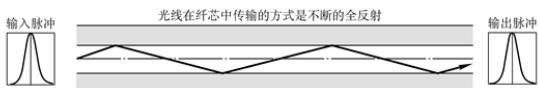


图 2-11 多模光纤

5. 非导向性传输介质

非导向性传输介质有短波、微波、红外线与可见光等。常见的通信方式有短波通信、微波通信、卫星通信、激光通信等。

0.81 03714-物理层接口特性

1. 物理层接口特性概念

物理层应尽可能地屏蔽各种物理设备的差异，使得数据链路层只需考虑本层的协议和服务。换句话说，物理层主要的功能其实就是确定与传输介质的接口有关的一些特性，即物理层接口的特性。

2. 物理层的四个特性

机械特性：指明接口的形状、尺寸、引线数目和排列等；

电气特性：电压的范围，即何种信号表示电压 0 和 1；

功能特性：接口部件的信号线（数据线、控制线、定时线等）的用途。

规程特性（2012 年真题已考）：或称为过程特性，物理线路上对不同功能的各种可能事件的出现顺序，即时序关系。

0.82 03715-中继器

物理层设备主要包含**中继器**和**集线器**，当然还有其他设备，但考研只需掌握此两种即可。

在计算机网络中，最简单的就是两台计算机通过两块网卡构成双机互连，这两台计算机的网卡之间一般是由非屏蔽双绞线来充当信号线的。由于双绞线在传输信号时信号功率会逐渐衰减，当信号衰减到一定程度时会造成信号失真，因此在保证信号质量的前提下，双绞线的最大传输距离为 100m。当两台计算机之间的距离超过 100m 时，为了实现双机互连，人们便在这两台计算机之间安装一个中继器，它的作用就是将已经衰减得不完整的信号经过整理，重新产生出完整的信号再继续传送。

注意：放大器和中继器都是起放大信号的作用，只不过放大器放大的是模拟信号，中继器放大的是数字信号。

0.83 03716-集线器

1. 集线器的基本概念

中继器是普通集线器的前身，**集线器实际就是一种多端口的中继器**。集线器一般有 4、8、16、24、32 等数量的接口，通过这些接口，集线器便能为相应数量的计算机完成“中继”功能。由于它在网络中处于一种“中心”位置，因此集线器也叫做 Hub。

2. 集线器的工作原理

假设有一个 8 个接口的集线器，共连接了 8 台计算机。集线器处于网络的“中心”，通过集线器对信号进行转发，可以实现 8 台计算机之间的互连互通。

集线器通信过程模拟：

假如计算机 1 要将一条信息发送给计算机 8，当计算机 1 的网卡将信息通过双绞线送到集线器上时，集线器并不会直接将信息送给计算机 8，它会将信息进行“广播”，即将信息同时发送给其他 7 个端口。当其他 7 个端口上的计算机接收到这条广播信息时，会对信息进行检查，如果发现该信息是发给自己的，则

接收，否则不予理睬。由于该信息是计算机 1 发给计算机 8 的，因此最终计算机 8 会接收该信息，而其他 6 台计算机检查信息后，会因为信息不是发给自己的而不接收该信息。

0.84 03717-数据链路层的主要功能

1. 数据链路层的概念

数据链路层在物理层所提供的服务的基础上向网络层提供服务，即将原始的、有差错的物理线路改进成逻辑上无差错的数据链路，从而向网络层提供高质量的服务。它一般包括 3 种基本服务：无确认的无连接服务、有确认的无连接服务和有确认的有连接服务。

记忆方式：有连接就一定要有确认，因为对方主机必须确认才可建立连接，即不存在无确认有连接服务）。

2. 数据链路层的主要功能

1) 链路管理：负责数据链路的建立、维持和释放，主要用于面向连接的服务。

2) 帧同步：接收方应当能从接收到的二进制比特流中区分出帧的起始与终止。

3) 差错控制：用于使接收方确定接收到的数据就是由发送方发送的数据。

4) 透明传输：不管数据是什么样的比特组合，都应当能在链路上传送。

0.85 03718-四种组帧方法

组帧只需要掌握四种方法，分别是：字符计数法、字节填充的首尾界符法、比特填充的首尾标志法、违规编码违例法。

1. 字符计数法

字符计数法是用一个特殊的字符来表示一帧的开始，然后用一个计数字段来表明该帧包含的字节数。当目的主机接收到该帧时，根据此字段提供的字节数，便可知道该帧的结束位和下一帧的开始位，如图 3-2 所示。

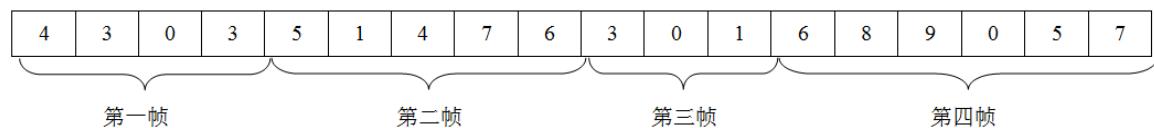


图 3-2 字符计数法

2. 字节填充的首尾界符法

由 C 语言的知识可以知道 ASCII 码是 7 位编码，可以组成 128 个不同的 ASCII 码，但是可以打印（就是可以从键盘输入的字符）的只有 95 个字符，那么当传送的帧是文本文件（都是从键盘输入的）时，就可以在剩下的 33 个控制字符中选定 2 个字符（教材中选用了 SOH 与 EOT 分别作为帧开始符和帧结束符）作为每一帧的开始和结束，这样接收端只需要判断这两个控制字符出现的位置就能准确地分割成帧，如图 3-3 所示。



图 3-3 SOH 与 EOT 分别作为帧开始符和帧结束符

这种方式对于帧数据为文本文件是绝对没有问题的。但是还有一种情况就比较麻烦了，假设要传送的帧不是文本文件，即帧数据部分可能包含控制字符，就不能仅仅使用控制字符去进行帧定界了，否则将会

导致错误地“找到帧的边界”，把部分帧收下（误认为是一个完整的帧），如图 3-4 所示。

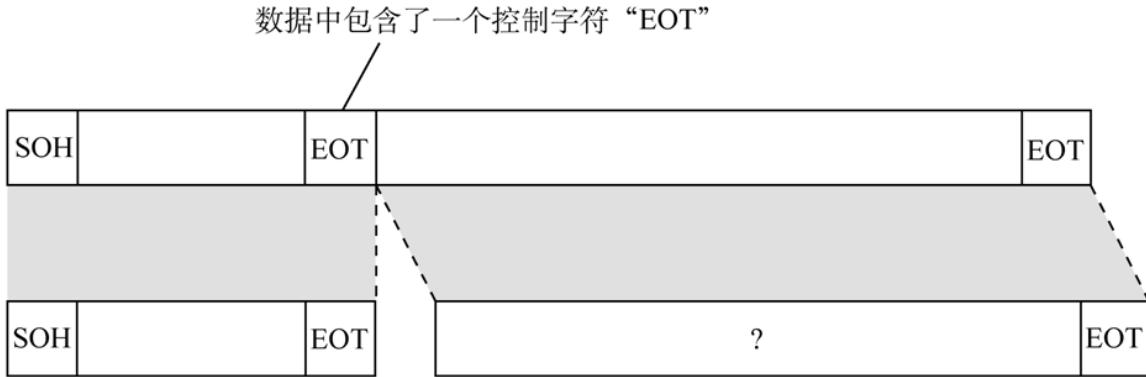


图 3-4 帧数据部分包含控制字符

从图 3-3 中可以看出确实解决了帧定界问题，但是从图 3-4 中看得出并不是所有比特流都可以被正确地传输，所以说此时透明传输问题仍未得到解决，首尾界符法是不严谨的，于是出现了字节填充的首尾界符法。

字节填充的首尾界符法设法将数据中可能出现的控制字符“SOH”和“EOT”在接收端不解释为控制字符。

其方法如下：

在数据中出现字符“SOH”或“EOT”时就将其转换为另一个字符，而这个字符是不会被错误解释的。但所有字符都有可能在数据中出现，于是就将数据中出现的字符“SOH”转换为“ESC”和“x”两个字符，将数据中出现的字符“EOT”转换为“ESC”和“y”两个字符。而当数据中出现了控制字符“ESC”时，就将其转换为“ESC”和“z”两个字符。这种转换方法能够在接收端将收到的数据正确地还原为原来的数据。“ESC”是转义符，它的十六进制编码是 1B。

如图 3-5 所示，在上方的数据中出现了 4 个控制字符“ESC”、“EOT”、“ESC”和“SOH”。按以上规则转换后的数据即为图 3-5 下方的数据。

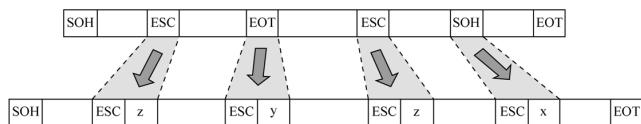


图 3-5 控制字符的转换

读者可以很容易地看出，在接收端只要按照以上转换规则进行相反的转换，就能够还原出原来的数据（如遇到“ESC”和“z”，就还原为“ESC”）。

3. 比特填充的首尾标志法

比特填充的首尾标志法是使用 01111110 作为帧的开始和结束标志，似乎帧定界又解决了，但是如果帧数据部分出现了 01111110 怎么办？透明传输仍然是个问题。其解决方法如下：

不难发现 01111110 中有 6 个连续的“1”，只要数据帧检测到有 5 个连续的“1”，马上在其后插入“0”，而接收方做该过程的逆操作，即每收到 5 个连续的“1”，自动删除后面紧跟的“0”，以恢复原始数据。因此，此方法又称为零比特填充法，具体可见下面的模拟过程。

模拟过程：

- 1) 原始数据。011010111110010111111011（数据中出现两次 01111110）。
- 2) 零比特填充后的数据。0110101111101001011111010111（加下画线的 0 表示填充的 0）。
- 3) 接收方收到数据，一旦遇到 5 个连续的“1”就将后面的“0”去掉，即可得到原始数据。

4. 物理编码违例法

物理编码违例法利用物理介质上编码的违法标志来区分帧的开始与结束，例如，在曼彻斯特编码中，码元 1 编码成高-低电平，码元 0 编码成低-高电平，而高-高和低-低电平的编码方式是无效的，可以分别

用来作为帧的起始标志和结束标志。

0.86 03719-检错编码

检错编码：通过一定的编码和解码，能够在接收端解码时检查出传输的错误，但不能纠正错误。常见的检错编码有奇偶校验码和循环冗余码（CRC）。

1. 奇偶校验码

奇偶校验码就是在信息码后面加一位校验码，分奇校验和偶校验。

奇校验：添加一位校验码后，使得整个码字里面 1 的个数是奇数。接收端收到数据后就校验数据里 1 的个数，若检测到奇数个 1，则认为传输没有出错；若检测到偶数个 1，则说明传输过程中，数据发生了改变，要求重发。

偶校验：添加一位校验码后，使得整个码字里面 1 的个数是偶数。接收端收到数据后就校验数据里 1 的个数，若检测到偶数个 1，则认为传输没有出错；若检测到奇数个 1，则说明传输过程中，数据发生了改变，要求重发。

2. 循环冗余码

奇偶校验码的检错率极低，不实用。目前，在计算机网络和数据通信中，用得最广泛的是检错率极高、开销小、易实现的循环冗余码（CRC）。循环冗余码的原理比较简单，这里就不再赘述了，教材讲解得很细致。

0.87 03720-纠错编码

纠错编码：就是在接收端不但能检查错误，而且能纠正检查出来的错误。

常见的纠错编码是海明码。

海明码：又称为汉明码，它是在信息字段中插入若干位数据，用于监督码字里的哪一位数据发生了变化，具有一位纠错能力。

假设信息位有 k 位，整个码字的长度就是 $k+r$ 位；每一位的数据只有两种状态，不是 1 就是 0，有 r 位数据就能表示出 2^r 种状态。如果每一种状态代表一个码元发生了错误，有 $k+r$ 位码元，就要有 $k+r$ 种状态来表示，另外还要有一种状态来表示数据正确的情况，所以 2^r-1 $k+r$ 才能检查一位错误，即 $2^r-1 \geq k+r+1$ 。例如，信息数据有 4 位，由 $2^r-1 \geq k+r+1$ 得 $r \geq 3$ ，也就是至少需要 3 位监督数据才能发现并改正 1 位错误。

海明码求解的具体步骤如下：

1) 确定校验码的位数 r 。

2) 确定校验码的位置。

3) 确定数据的位置。

4) 求出校验位的值。

实战例题请见《计算机网络高分笔记》。

考生在教材中肯定见到过以下公式：

$$L-1=D+C \quad \text{且 } D \leq C$$

如果要纠正 d 位错误，说明至少要检测出 d 位错误（当然可以检测得更多），代入即可得到 $L-1=d+d$ ，即 $L=2d+1$ 。

0.88 03721-流量控制

1. 流量控制

a. 基本概念：流量控制就是要控制发送方发送数据的速率，使接收方来得及接收。一个基本的方法是由接收方控制发送方的数据流。常见的有两种方式：**停止-等待流量控制**和**滑动窗口流量控制**。

2. 停止-等待流量控制

它是流量控制中最简单的形式。停止-等待流量控制的工作原理就是发送方发出一帧，然后等待应答信号到达再发送下一帧；接收方每收到一帧后，返回一个应答信号，表示可以接收下一帧，如果接收方不回应答，则发送方必须一直等待。

3. 滑动窗口流量控制

a. 基本概念：停止-等待流量控制中每次只允许发送一帧，然后就陷入等待接收方确认信息的过程中，传输效率很低。而**滑动窗口流量控制**允许一次发送多个帧。

b. 工作原理：在任意时刻，发送方都维持了一组连续的允许发送的帧的序号，称为**发送窗口**。同时，接收方也维持了一组连续的允许接收的帧的序号，称为**接收窗口**。发送窗口和接收窗口的序号的上下界不一定要一样，甚至大小也可以不同。**发送方窗口内的序列号代表了那些已经被发送但是还没有被确认的帧，或者是那些可以被发送的帧。发送端每收到一个帧的确认，发送窗口就向前滑动一个帧的位置。**当发送窗口尺寸达到最大尺寸时，发送方会强行关闭网络层，直到有一个空闲缓冲区出来。在接收端只有当收到的数据帧的发送序号落入接收窗口内才允许将该数据帧收下，并将窗口前移一个位置。**若接收到的数据帧落在接收窗口之外**（就是说收到的帧号在接收窗口中找不到相应的该帧号），则一律将其丢弃。

0.89 03722-可靠传输机制

可靠传输与无差错接收的区别总结。

解析：在数据链路层若仅仅使用循环冗余码检验差错检测技术，只能做到对帧的无差错接收，即“凡是接收端数据链路层接收的帧，都能以非常接近于 1 的概率认为这些帧在传输过程中没有产生差错”。接收端的帧虽然收到了，但最终还是因为有差错被丢弃，即没有被接收。

以上所述可以近似地表述为“凡是接收端数据链路层接收的帧均无差错”。

注意：现在并没有要求数据链路层向网络层提供“可靠传输”的服务。所谓“可靠传输”，就是数据链路层的发送端发送什么，接收端就接收什么。传输差错可分为两大类，一类就是比特差错（可以通过 CRC 来检测），而另一类传输差错更复杂，这就是收到的帧并没有出现比特差错，但却出现了帧丢失（如发送 1、2、3，收到 1、3）、**帧重复**（如发送 1、2、3，收到 1、2、2、3）、**帧失序**（如发送 1、2、3，收到 1、3、2）。这 3 种情况都属于“出现传输差错”，但都不是这些帧里有“比特差错”。

帧丢失很容易理解，但是帧重复、帧失序的情况较为复杂，在这里暂不讨论，学完可靠传输的工作原理后，就会知道在什么情况下接收端可能会出现帧重复或帧失序。

总之，“**无比特差错**”和“**无传输差错**”并不是同样的概念，在数据链路层使用 CRC 检验只能实现无比特差错的传输，但这还不是可靠传输。

0.90 03724-停止等待协议

从名称上来看，也可以看出停止-等待协议是基于停止-等待流量控制技术的。从滑动窗口的角度来理解就是其发送窗口大小为 1，接收窗口大小也为 1。

停止-等待协议的基本思想：发送方传输一个帧后，必须等待对方的确认才能发送下一帧。若在规定时间内没有收到确认，则发送方超时，并重传原始帧。看到这里也许有人会问，停止-等待流量控制技术（这里是停止-等待流量控制技术而不是停止-等待协议）为什么要一直等待？为什么不设置一个规定时间？这里就要回到第 1 章协议的制定。首先协议需要建立在一定技术（停止-等待流量控制技术）之上，然后在此技术之上需要考虑一切可能突发的不利状况（**可以这么理解：协议 = 技术 + 考虑不利因素**，即停止-等

待协议 = 停止-等待流量控制技术 + 不利因素)，设置规定时间重传就是为了解决这些不利因素。如果不设置时间就会造成死锁，这样就无法推进，在这里可以联系到操作系统的死锁，如果没有外力参与去打破死锁，就会一直等待下去，而这里的外力就是重传计时器。

停止-等待协议中会出的差错主要有以下两类（虽然简单，请仔细看，这里有很多考生的疑问点，其他辅导书都没有涉及）。

1) **帧一般被分为数据帧和确认帧。**第一类错误就是数据帧被损坏或者丢失，那么接收方在进行差错检验时，会检测出来。处理数据帧被损坏的情况时，使用计时器即可解决。这样发送方在发送一个帧后，若数据能够正确地接收到，接收方就发送一个确认帧，没有问题；若接收方收到的是一个被损坏的数据帧，则直接丢弃，此时发送方还在那里苦等，不过没关系，只要计时器超时了，发送方就会重新发送该数据帧，如此重复，直到这一数据帧无错误地到达接收方为止。

2) **第二类错误是确认帧被破坏或者丢失。**一旦确认帧被破坏或者丢失，造成的后果就是发送方会不断地重新发送该帧，从而导致接收方不断地重新接收该帧。怎么解决？显然，对于接收方而言，需要有能够区分某一帧是新帧还是重复帧的能力。解决方法很简单，就是让发送方在每个待发的帧的头部加一个编号，而接收方对每个到达的帧的编号进行识别，判断是新帧还是要抛弃的重复帧。

0.91 03725-后退 N 帧 (GBN) 协议

后退 N 帧协议基于滑动窗口流量控制技术。若采用 n 个比特对帧进行编号，其发送窗口尺寸 WT 必须满足 $1 < WT \leq 2^n - 1$ （请参考下面的补充知识点），接收窗口尺寸为 1。若发送窗口尺寸大于 $2^n - 1$ ，会造成接收方无法分辨新、旧数据帧的问题。由于接收窗口尺寸为 1，因此接收方只能按序来接收数据帧。

后退 N 帧协议的基本原理：发送方发送完一个数据帧后，不是停下来等待确认帧，而是可以连续再发送若干个数据帧。如果这时收到了接收方的确认帧，那么还可以接着发送数据帧。**如果某个帧出错了，接收方只能简单地丢弃该帧及其所有的后续帧。**发送方超时后需重发该出错帧及其后续的所有帧。由于减少了等待时间，后退 N 帧协议使得整个通信的吞吐量得到提高。但接收方一发现错误帧，就不再接收后续的帧，造成了一定的浪费。据此改进，得到了选择重传协议。

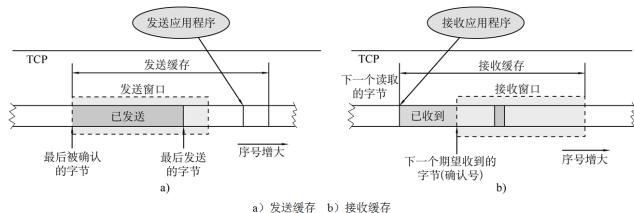
补充知识点：为什么后退 N 帧协议的发送窗口尺寸 WT 必须满足 $1 < WT \leq 2^n - 1$ ？

解析：假设发送窗口的大小为 2^n ，发送方发送了 0 号帧，接收窗口发送 ACK1（0 号帧已收到，希望接收 1 号帧，但是 ACK1 丢失），接着发送方发送了 1 号帧，接收窗口发送 ACK2（1 号帧已收到，希望接收 2 号帧，但是 ACK2 丢失），以此类推，直到发送方发了第 $2^n - 1$ 号帧，接收方发送 ACK 2^n （丢失），此时不能再发送数据了，因为已经发送了 2^n 个帧，但一个确认都没有收到，所以过一段时间 0 号帧的计时器会到达预定时间进行重发，此时发过去接收方认为是新一轮的 0 号帧还是旧一轮重传的呢？接收方并不知道，很有可能接收方就把该 0 号帧当作新一轮的帧接收了，但实际上这个 0 号帧是重传的，所以出现了错误，即发送窗口的大小不可能为 2^n 。现在假设发送窗口的大小为 $2^n - 1$ ，情况和上面一样，发送方发送了 $0 \sim 2^n - 2$ 号帧，接收方发送的确认帧都丢失了，如果没有丢失就应该接着传 $2^n - 1$ 号帧，但是丢失了，发送方应该发送 0 号帧。由于这种情况接收方可以判断出来（即下一帧只要不是第 $2^n - 1$ 号帧就是重传），因此不会发生错误。如果发送窗口尺寸小于 $2^n - 1$ ，那就更不会发生错误了。

综上所述，后退 N 帧协议的最大发送窗口是 $2^n - 1$ 。

0.92 03727-发送缓存和接收缓存

如下图所示，发送窗口与发送缓存以及接收窗口与接收缓存有什么区别？为什么计算机进行通信时发送缓存和接收缓存总是需要的？



从上图 b) 中可以看到，按序到达的且没有被交付给主机的帧被放在**接收缓存**（接收窗口外的那一部分接收缓存，以下讲的接收缓存都是指这部分）里面（因为已经发送过确认帧了，仅仅是等主机的应用程序来取），而不是接收窗口里面。那些不是按序到达的数据且没有错误的帧一定是要放在接收窗口里面，因为这些帧不能直接给主机，而放在**接收缓存**的帧是要给主机的，等到缺少的帧收到后，再一起放到接收缓存，这一点要注意区分，其他都比较好理解，不再赘述。

发送窗口的大小不一定等于接收窗口的大小（但是通常情况下都是等于），这里先记住这个结论，第5章讲到拥塞控制的时候就会很清楚了。

当计算机的两个进程（在同一台机器中或在两个不同的机器中）进行通信时，如果发送进程将数据直接发送给接收进程，那么这两个动作（一个是发送，另一个是接收）是非常难协调的。这是因为计算机的动作很快，如果在某一时刻接收进程开始执行接收的动作，但发送进程的发送动作稍微早了一点或稍微晚了一点（在收发双方事先未进行同步的情况下，发送时刻不可能恰好和接收时刻精确地重合），这都会使接收失败。

综上所述，在计算机进程之间的通信过程中，广泛使用缓存。缓存就是在计算机的存储器中设置的一个临时存放数据的空间。

发送进程将欲发送的数据先写入缓存，然后接收进程在合适的时机读出这些数据。缓存类似于邮局在街上设立的邮筒。人们可以将欲发送的信件投到邮筒中。邮局的邮递员按照他的计划在适当的时候打开邮筒，将信件取走，交到邮局，进行下一步处理。缓存可以很好地解决发送速率和接收速率不一致的矛盾，还可以很方便地进行串并转换，即比特流串行写入并行读出，或并行写入串行读出。**缓存也可称为缓冲或缓冲区**（有关缓存更详细的讲解可参考《操作系统高分笔记》）。

0.93 03729-信道划分介质访问控制

信道划分介质访问控制分为以下 4 种。

1. 频分多路复用

将一条信道分割成多条不同频率的信道，就类似于将一条马路分割成多个车道，尽管同一时间车辆都在这条马路上行驶，但是分别行驶在不同的车道上，所以不会发生冲突。现在假设每个车道的宽度不能改变了，但是需要加车道，所以马路就必须变宽，类似于使用频分复用时，如果复用数增加，那么信道的带宽（此时的带宽是频率带宽，不是数据的发送速率）必须得增加。

注意：每个子信道分配的带宽可以不相同，但它们的总和一定不能超过信道的总带宽（可联想人行道和机动车道是不一样宽的）。在实际应用中，为防止子信道之间的干扰，相邻信道之间要加入“保护频带”（可联想人行道与机动车道、机动车道与机动车道之间的栏杆的作用）。

2. 时分多路复用

假设现在只有一个玩具，却有 10 个小孩要玩，这时候只能将一个固定的时间分割成 10 份，10 个小孩轮流玩这个玩具，即时分多路复用。

所以当使用时分多路复用时，复用数增加并不需要加大信道带宽，只需将每个信道分得的时间缩小即可。也许很多人在这里会有疑问，如果恰好某个时间轮到一个小孩玩了，但是这个小孩现在睡着了，岂不是这段时间就浪费了吗？没错，是浪费了，这时候就需要把时分复用改进，于是引入**统计时分复用**。继续

上面的例子，现在如果该玩具轮到某个小孩玩，但是他睡着了，立刻跳过他，给下一个小孩玩，这样就基本可以保证玩具没有空闲时刻。可见每个孩子下次轮到自己玩的时间都是不确定的，如果睡觉的人多了，很快就轮到了；如果睡觉的人少，就很慢。因此，**统计时分复用是一种动态的时间分配**（会考选择题，请记住），同时又是异步的（每个孩子玩玩具的时间周期是不固定的），所以统计时分复用又称为异步时分复用。而普通的时分复用就是同步时分复用（因为每个孩子都在一个固定的周期才能得到玩具，即使中间有孩子睡觉也要等）。

3. 波分多路复用

波分多路复用就是光的频分多路复用，在一根光纤中传输多种不同频率（波长）的光信号，由于各路光的频率（波长）不同，因此各路光信号不互相干扰。最后，再用分波器将各路波长不一样的光分解出来。

4. 码分多路复用

码分多路复用又称为码分多址（CDMA），它既共享信道的频率，又共享时间，是一种真正的动态复用技术。本书主要讲解 CDMA 原理中的一个考点，其他内容考研不会涉及，有兴趣的考生可参考教材上的详细讲解。考点分析如下。

概念：每个站点都维持一个属于该站点的芯片序列，并且是固定的。假如站点 A 的芯片序列为 00011011，则 A 站点发送 00011011 表示发送比特 1；而将 00011011 每位取反，即发送 11100100 表示发送比特 0。习惯将芯片序列中的 0 写为 -1，1 写为 +1，所以 A 站的芯片序列就是 $(-1 \ -1 \ -1 \ +1 \ +1 \ -1 \ +1 \ +1)$ ，一般将该向量称为该站的码片向量。以下两个定理记住即可。

- 1) 任意两个不同站的码片向量正交，即任意两个站点的码片向量的规格化内积一定为 0。
- 2) 任意站点的码片向量与该码片向量自身的规格化内积一定为 1；任何站点的码片向量和该码片的反码片向量的规格化内积一定为 -1。

考点：某个 CDMA 站接收到一个碎片序列，怎么去判断是哪站发来的数据，并怎么识别发送了什么信息？见例 3-7。

【例 3-7】 某个 CDMA 站接收方收到一条如下所示的碎片系列：

$$(-1 \ +1 \ -3 \ +1 \ -1 \ -3 \ +1 \ +1)$$

假设各个站点的码片向量如下所示。

$$\text{站点 A: } (-1 \ -1 \ -1 \ +1 \ +1 \ -1 \ +1 \ +1)$$

$$\text{站点 B: } (-1 \ -1 \ +1 \ -1 \ +1 \ +1 \ +1 \ -1)$$

$$\text{站点 C: } (-1 \ +1 \ -1 \ +1 \ +1 \ +1 \ -1 \ -1)$$

$$\text{站点 D: } (-1 \ +1 \ -1 \ -1 \ -1 \ +1 \ -1)$$

试问：哪些站点发送了数据？分别发送了什么数据？

解析：此题的解答步骤较为固定，只需将接收到的碎片序列分别与站点 A、B、C、D 的码片向量进行规格化内积即可。内积为 1 表示发送了比特 1，内积为 -1 表示发送了比特 0，内积为 0 表示没有发送数据，计算如下。

$$\text{站点 A: } (-1 \ +1 \ -3 \ +1 \ -1 \ -3 \ +1 \ +1) \cdot (-1 \ -1 \ -1 \ +1 \ +1 \ -1 \ +1 \ +1) / 8 = 1.$$

$$\text{站点 B: } (-1 \ +1 \ -3 \ +1 \ -1 \ -3 \ +1 \ +1) \cdot (-1 \ -1 \ +1 \ -1 \ +1 \ +1 \ +1 \ -1) / 8 = -1.$$

$$\text{站点 C: } (-1 \ +1 \ -3 \ +1 \ -1 \ -3 \ +1 \ +1) \cdot (-1 \ +1 \ -1 \ +1 \ +1 \ +1 \ -1 \ -1) / 8 = 0.$$

$$\text{站点 D: } (-1 \ +1 \ -3 \ +1 \ -1 \ -3 \ +1 \ +1) \cdot (-1 \ +1 \ -1 \ -1 \ -1 \ +1 \ -1) / 8 = 1.$$

由以上计算结果可知，站点 A 和站点 D 发送了比特 1，站点 B 发送了比特 0，站点 C 没有发送数据。

0.94 03730-随机访问介质访问控制

随机接人在考研中需要掌握 4 种，即 ALOHA 协议、CSMA 协议、CSMA/CD 协议和 CSMA/CA 协议。

以上 4 种协议的核心思想是通过争用，胜利者才可以获得信道，从而获得信息的发送权。正因为这种思想，随机访问介质访问控制又多了一个绰号：**争用型协议**。

1. ALOHA 协议

最初的 ALOHA 称为纯 ALOHA 协议，其基本思想比较简单：当网络中的任何一个结点需要发送数据时，可以不进行任何检测就发送数据。如果在一段时间内没有收到确认，该结点就认为传输过程中发生了冲突。发生冲突的结点需要等待一段随机时间后再发送数据，直至发送成功为止。

纯 ALOHA 协议虽然简单，但其性能特别是信道利用率并不理想。于是，后来又有了时分 ALOHA (Slotted ALOHA)。在时分 ALOHA 中，所有结点的时间被划分为间隔相同的时隙 (Slot)，并规定每个结点只有等到下一个时隙到来时才可发送数据。

2. CSMA 协议

载波监听多路访问 (CSMA) 协议是在 ALOHA 协议的基础上改进而来的一种多路访问控制协议。在 CSMA 中，每个结点发送数据之前都使用载波监听技术来判定通信信道是否空闲。**常用的 CSMA 有以下 3 种策略**。

- 1) 1-坚持 CSMA。当发送结点监听到信道空闲时，**立即发送数据**，否则继续监听。
- 2) p-坚持 CSMA。当发送结点监听到信道空闲时，**以概率 p 发送数据**，以概率 (1-p) 延迟一段时间并重新监听。
- 3) 非坚持 CSMA。当发送结点一旦监听到信道空闲时，**立即发送数据**，否则**延迟一段随机的时间再重新监听**。

3. CSMA/CD 协议

CSMA/CD 工作流程：每个站在发送数据之前要先检测一下总线上是否有其他计算机在发送数据，若有，则暂时不发送数据，以免发生冲突；若没有，则发送数据。计算机在发送数据的同时检测信道上是否有冲突发生，若有，则采用截断二进制指数类型退避算法来等待一段随机时间后再次重发。总体来说，可概括为“先听后发，边听边发，冲突停发，随机重发”。

争用期：指以太网端到端的往返时延（用 $2t$ 表示），又称为冲突窗口或者碰撞窗口。只有经过争用期这段时间还没有检测到冲突，才能肯定这次发送不会发生冲突。

截断二进制指数类型退避算法：发生碰撞的站在停止发送数据后，要推迟一个随机时间才能再发送数据。退避的时间按照以下算法计算。

- 1) 确定基本退避时间，一般取争用期 $2t$ 。
- 2) 定义重传参数 k ， $k = \text{Min}[\text{重传次数}, 10]$ 。可见，当重传次数不超过 10 时，参数 k 等于重传次数；当重传次数超过 10 时， k 就不再增大而一直等于 10。
- 3) 从整数集合 $[0, 1, \dots, 2^k - 1]$ 中随机选择一个数记为 r ，重传所需时延就是 r 倍的基本退避时间，即 $2rt$ 。
- 4) 当重传次数达到 16 次仍不能成功时，说明网络太拥挤，直接丢弃该帧，并向高层报告。

4. CSMA/CA 协议

CSMA/CA 主要用在无线局域网中，由 IEEE 802.11 标准定义，它在 CSMA 的基础上增加了冲突避免的功能。冲突避免要求每个结点在发送数据之前监听信道。如果信道空闲，则发送数据。发送结点在发送完一个帧后，必须等待一段时间（称为帧间间隔），检查接收方**是否发回帧的确认**（说明 CSMA/CA 协议对正确接收到的数据帧进行确认，2011 年真题中的一道选择题考查了此知识点）。若收到确认，则表明无冲突发生；若在规定时间内没有收到确认，表明出现冲突，重发该帧。

0.95 03731-轮询访问介质访问控制

轮询访问介质访问控制主要用在令牌环局域网中，目前使用得很少。在轮询访问介质访问控制中，用户不能随机地发送信息，而是通过一个集中控制的监控站经过轮询过程后再决定信道的分配。典型的轮询访问介质访问控制协议就是**令牌传递协议**。

令牌环局域网把多个设备安排成一个物理或逻辑连接环。为了确定哪个设备可以发送数据，让一个令牌（特殊格式的帧）沿着环形总线在计算机之间依次传递。当计算机都不需要发送数据时，令牌就在环形网上“游荡”，而需要发送数据的计算机只有拿到该令牌才能发送数据帧，所以不会发生冲突（因为令牌只有一个），这就是所谓的受控接入。

0.96 03732-局域网的基本概念与体系结构

局域网（Local Area Network, LAN）是指一个较小范围（如一个公司）内的多台计算机或者其他通信设备，通过双绞线、同轴电缆等连接介质互连起来，以达到资源和信息共享目的的互联网络。

1. 局域网最主要的特点

- a. 局域网为一个单位所拥有（如学校的一个系使用一个局域网）。
- b. 地理范围和站点数目有限（双绞线的最大传输距离为 100m，如果要加大传输距离，则在两段双绞线之间安装中继器，最多可安装 4 个中继器，例如，安装 4 个中继器连接 5 个网段，则最大传输距离可达 500m，所以地理范围有限。局域网一般可以容纳几台至几千台计算机，所以站点数目有限）。
- c. 与以前非光纤的广域网相比，局域网具有较高的数据率、较低的时延和较小的误码率（现在局域网的数据率可以达到万兆了；传输距离较短所以时延小；距离短了失真就小，误码率自然就低）。

2. 局域网的主要优点

- a. 具有广播功能，从一个站点可很方便地访问全网。局域网上的主机可共享连接在局域网上的各种硬件和软件资源。
- b. 便于系统的扩展和演变，各设备的位置可灵活地调整和改变。
- c. 提高了系统的可靠性、可用性。

3. 局域网的主要技术要素

局域网的主要技术要素包括**网络拓扑结构、传输介质与介质访问控制方法**。其中，介质访问控制方法是最为重要的技术特性，决定着局域网的技术特性。

4. 局域网的主要拓扑结构

局域网的主要拓扑结构包括星形网、环形网、总线型网和树形网。

5. 局域网的主要传输介质

局域网的主要传输介质包括双绞线、铜缆和光纤等，其中**双绞线为主流传输介质**。

6. 局域网的主要介质访问控制方法

局域网的主要介质访问控制方法包括 CSMA/CD、令牌总线和令牌环。

0.97 03733-以太网的工作原理

1. 以太网

a. 知识背景：IEEE 802.3 标准是一种基带总线型的局域网标准。在不太严格区分的时候，IEEE 802.3 可以等同于以太网标准，因为它是基于原来的以太网标准诞生的一个总线型局域网标准。随着快速以太网、千兆以太网和万兆以太网相继进入市场，**以太网现在几乎成了局域网的同义词**。

b. 工作原理：**以太网采用总线拓扑结构**，所有计算机都共享一条总线，**信息以广播方式发送**。为了保证数据通信的方便性和可靠性，以太网使用了 CSMA/CD 技术对总线进行访问控制。考虑到局域网信道质量好，以太网采取了以下两项重要的措施以使通信更加简便。

1) 采用无连接的工作方式。

2) 不对发送的数据帧进行编号，也不要求对发送方发送确认。

因此以太网提供的服务是**不可靠的服务**，即尽最大努力交付，差错的纠正由传输层的 TCP 完成。

0.98 03734-以太网的 MAC 帧

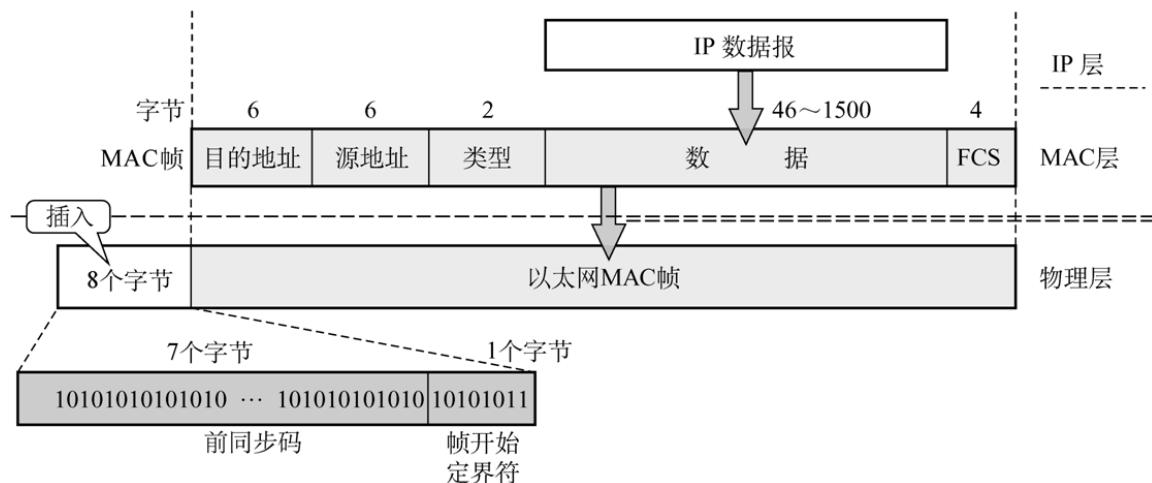
1. MAC 地址

局域网中的每台计算机都有一个唯一的号码，称为 **MAC 地址或物理地址、硬件地址**。

每块网卡出厂即被赋予一个全球唯一的 MAC 地址，它被固化在网卡的 ROM 中，共 48bit (6B)，例如，01-3e-01-23-4e-3c 十六进制表示，01 其实是 0000 0001。高 24bit 为厂商代码，低 24bit 为厂商自行分配的网卡序列号。

由于总线上使用的是广播通信，因此网卡从网络上每收到一个 MAC 帧，先要用硬件检查 MAC 帧中的 MAC 地址。**如果是发往本站的帧就收下，否则丢弃。**

2. MAC 帧的格式



1. **前导码**。在帧的前面插入 8B，使接收端与发送端进行时钟同步。这 8B 又可分为前同步码 (7B) 和帧开始定界符 (1B) 两部分。

2. **目的地址、源地址**。均使用 48bit (6B) 的 MAC 地址。

3. **类型**。占 2B。指出数据域中携带的数据应交给哪个协议实体处理。

4. **数据**。占 46 ~ 1500B。46 和 1500 是怎么来的？首先，由 CSMA/CD 算法可知，以太网帧的最短帧长为 64B，而 MAC 帧的首部和尾部的长度为 18B，所以数据最短为 64B-18B= 46B。其次，最大的 1500B 是规定的，没有为什么。

5. **填充**。前面讲过，由于 CSMA/CD 算法的限制，最短帧长为 64B，因此除去首部 18B，如果数据长度小于 46B，那么就得填充，使得帧长不小于 64B。当数据字段长度小于 46B 时，需要填充至 46B；当数据字段长度大于或等于 46B 时，则无需填充。因此，填充数据长度的范围为 0 ~ 46B。

6. **校验码 (FCS)**。占 4B。采用循环冗余码，不但需要校验 MAC 帧的数据部分，还要校验目的地址、源地址和类型字段，但是**不校验前导码**。

0.99 03740-PPP 协议

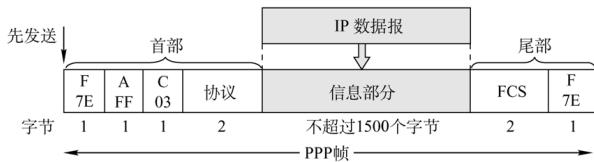
1. PPP 协议的组成

- 一个将 IP 数据报封装到串行链路的方法；
- 一个链路控制协议 (LCP)，用于建立、配置和测试数据链路连接，并在不需要时将它们释放；

c. 一套网络控制协议 (NCP)，其中每个协议支持不同的网络层协议，用来建立和配置不同的网络层协议。

2. PPP 协议的帧格式

PPP 协议的帧格式如下图所示。



1) **标志字段 (F)**。首部和尾部各占 1 个字节，规定为 Ox7E。

2) **地址字段 (A)**。占 1 个字节。规定为 OxFF，没有为什么。

3) **控制字段 (C)**。占 1 个字节。规定为 Ox03，没有为什么。

4) **协议字段**。占 2 个字节。例如，当协议字段为 0x0021 时，PPP 帧的信息字段就是 IP 数据报；若为 0xC021，则信息字段是 PPP 链路控制数据；若为 0x8021，则表示这是网络控制数据。

5) **信息部分**。占 0 ~ 1500 个字节。为什么不是 46 ~ 1500 个字节？因为 PPP 是点对点的，并不是总线型，所以无需采用 CSMA/CD 协议，自然就没有最短帧。另外，当数据部分出现和标志位一样的比特组合时，就需要采用一些措施来实现透明传输（上面的补充知识点已讲）。

6) **帧检验序列 (FCS)**。占 2 个字节，即循环冗余码检验中的冗余码。检验区间包括地址字段、控制字段、协议字段和信息字段。

3. PPP 的工作状态

当用户拨号接入 ISP 时，路由器的调制解调器对拨号做出确认，并建立一条物理连接。这时，个人计算机向路由器发送一系列的 LCP 分组（封装成多个 PPP 帧）。这些分组及其响应选择了将要使用的一些 PPP 参数。接着就进行网络层配置，网络控制协议 (NCP) 给新接入的个人计算机分配一个临时的 IP 地址。这样，个人计算机就成为因特网上的一个主机了。

当用户通信完毕时，NCP 释放网络层连接，收回原来分配出去的 IP 地址。接着，LCP 释放数据链路层连接，最后释放物理层连接。

4. 总结

a. PPP 是一个面向字节的协议。

b. PPP 不需要的功能：纠错（PPP 只负责检错）、流量控制（由 TCP 负责）、序号（PPP 是不可靠传输协议，所以不需要对帧进行编号）、多点线路（PPP 是点对点的通信方式）、半双工或单工（PPP 只支持全双工链路）。

0.100 03741-HDLC 协议

1. HDLC 协议的基本特点

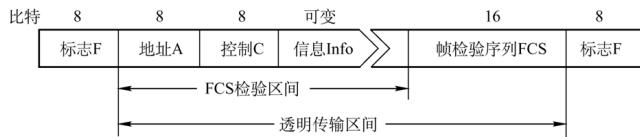
高级数据链路控制 (HDLC) 协议是 ISO 制定的面向比特（PPP 是面向字节的，这个要记住）的数据链路控制协议。它可适用于链路的两种基本配置：非平衡配置和平衡配置。

a. 非平衡配置的特点是由一个主站控制整个链路的工作；

b. 平衡配置的特点是链路两端的两个站都是复合站，每个复合站都可以平等地发起数据传输，而不需要得到对方复合站的允许。

2. HDLC 协议的帧格式

当采用 HDLC 协议时，从网络层交下来的分组，变成了 HDLC 协议帧的数据部分，数据链路层在信息字段的头尾各加上 24 位控制信息，这样就构成了一个完整的 HDLC 协议帧，如下图所示。



1) **标志字段 (F)**。占 8 位，为“01111110”，首尾各有一个“0”作为帧的边界。为防止在两个标志字段 F 之间出现“01111110”，HDLC 使用比特填充的首尾标志法。当一串比特流未加上控制信息时，扫描整个帧，只要发现有 5 个连续“1”，就立即填入一个“0”。

2) **地址字段 (A)**。占 8 位。若使用非平衡方式传送数据，为次站的地址；若使用平衡方式传送数据，为确认站的地址。全“1”为广播方式，全“0”为无效地址。

3) **控制字段 (C)**。占 8 位。最复杂的字段，HDLC 的许多重要功能都靠控制字段实现。根据其最前面两位的取值，可将 HDLC 帧划分为三类：信息帧 (I 帧)、监督帧 (S 帧) 和无编号帧 (U 帧)。

提醒：三类帧的记忆方式，每当看到 HDLC 帧的分类就想到“无监息”=“无奸细”。

信息帧用来传输数据信息，或使用捎带技术对数据进行确认和应答；监督帧用于流量控制和差错控制，执行对信息帧的确认、请求重发和请求暂停发送等功能；无编号帧用于提供对链路的建立、拆除以及多种控制功能。

4) **信息字段**。长度任意，存放来自网络层的协议数据单元。

5) **帧检验序列 (FCS)**。占 16 位，即循环冗余码检验中的冗余码。检验区间包括地址字段、控制字段和信息字段。

0.101 03742-网桥的概念和基本原理

1. 网桥的基本概念

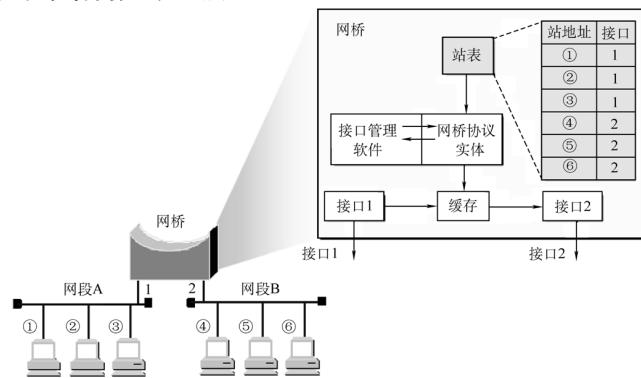
a. **基本介绍：**在数据链路层扩展局域网是使用网桥。网桥工作在数据链路层，其特点是具有过滤帧的功能。网桥至少有两个端口，每个端口与一个网段相连。

b. **网桥的优点：**过滤通信量；扩大了物理范围；提高了可靠性；可互连不同物理层、不同 MAC 子层和不同速率（如 10Mbit/s 和 100Mbit/s）的以太网。

c. **网桥的缺点：**存储转发增加了时延；在 MAC 子层并没有流量控制功能；具有不同 MAC 子层的网段桥接在一起时延更大；网桥只适合于用户数不太多（不超过几百个）和通信量不太大的局域网，否则有时还会因传播过多的广播信息而产生网络拥塞，即广播风暴。

2. 网桥的基本原理

网桥每从一个端口接收到一个帧，就先暂存到缓存中。若该帧未出现差错，且欲发往的目的站 MAC 地址属于另一个网段（同一个网段无需转发，应该丢弃），则通过查找转发表，将该帧从对应的端口发出。因此，仅在同一个网段中通信的帧，不会被网桥转发到另一个网段中，因而不会加重整个网络的负担。网桥的内部结构如下图所示。



0.102 03743-网桥的分类

1. 透明网桥（选择的不是最佳路由）

a. 基本概念：透明网桥是目前使用最多的网桥。“透明”是指局域网上的站点并不知道所发送的帧将经过哪几个网桥，因为网桥对各站来说是看不见的。透明网桥是一种即插即用设备，意思是只要把网桥接入局域网，不用人工配置转发表，网桥就可以开始工作。既然上面提到了网桥可以不用人工配置转发表，网桥如何自学习？

b. 自学习步骤：

1) 网桥收到一帧后先进行自学习。查找转发表中与收到帧的源地址有无相匹配的项目。若没有，就在转发表中增加一个项目（源地址、进入的接口和时间）。若有，则把原有的项目进行更新。

2) 转发帧。查找转发表中与收到帧的目的地址有无相匹配的项目。若没有，则通过所有其他接口（但进入网桥的接口除外）进行转发。若有，则按转发表中给出的接口进行转发。若转发表中给出的接口是该帧进入网桥的接口，则应丢弃这个帧（因为这时不需要经过网桥进行转发）。

2. 源选径网桥（选择最佳路由）

在源选径网桥中，路由选择由发送数据帧的源站负责，网桥只根据数据帧中的路由信息对帧进行接收和转发。

为了发现合适的路由，源站先以广播方式向欲通信目的站发送一个发送帧。发送帧将在整个局域网中沿着所有可能的路由传送，并记录所经过的路由。当发送帧到达目的站时，就沿原来的路径返回源站。源站在得知这些路由后，再从所有可能的路由中选择一个最佳路由。

0.103 03744-局域网交换机及工作原理

1. 局域网交换机的基本概念

局域网交换机实质上是多端口网桥，它工作在数据链路层。局域网交换机的每个端口都直接与主机或集线器相连，并且一般都工作在全双工方式。当主机需要通信时，交换机能同时连通许多对的端口，使每一对相互通信的主机都能像独占通信媒体那样，进行无冲突的传输数据，通信完成后断开连接。

2. 交换机总容量计算方式

- 如果是半双工：总容量 = 端口数 × 每个端口带宽；
- 如果是全双工：总容量 = 端口数 × 每个端口带宽 × 2。

3. 交换机的两种交换模式

a. 直通式交换：只检查帧的目的地址，这使得帧在接收后能马上被转发出去。这种方式速度很快，但缺乏安全性，也无法支持具有不同速率的端口的交换。

b. 存储转发式交换：先将接收到的帧存储在高速缓存中，并检查数据是否正确，确认无误后，查找转发表，并将该帧从查询到的端口转发出去。如果发现该帧有错误，就将其丢弃。存储转发式交换的优点是可靠性高，并能支持不同速率端口间的转换，其缺点是延迟较大。

4. 局域网交换机的工作原理

与网桥类似，检测从某端口进入交换机的帧的源 MAC 地址和目的 MAC 地址，然后与系统内部的动态查找表进行比较，若数据报的 MAC 地址不在查找表中，则将该地址加入查找表中，并将数据报发送给相应的目的端口。

0.104 03750-异构网络互连

1. 异构网络互连的技术基础

虚拟互联网络也就是逻辑互联网络，它的意思就是互连起来的各种物理网络的异构性本来是客观存在的，但是利用协议可以使这些性能各异的网络让用户看起来好像是一个统一的网络。这种协议就是网络层重点讨论的 IP 协议。

2. 异构互联网网络的硬件支持

将网络互连起来肯定需要一些中间设备（又称为中间系统或中继系统），根据中继系统所在的层次，可以有以下 4 种不同的中继系统。

- a. 物理层的中继系统：中继器或集线器。
- b. 数据链路层的中继系统：网桥或交换机。
- c. 网络层的中继系统：路由器。
- d. 网络层以上的中继系统：网关。

当中继系统是中继器或网桥时，一般并不称之为网络互连，因为这仅仅是把一个网络扩大了，而这仍然是一个网络。**互联网都是指用路由器进行互连的网络。**

使用虚拟互联网的好处：当互联网上的主机进行通信时，就好像在同一个网络上通信，而看不见互连的具体的网络异构细节（如超时控制、路由选择协议等）。

0.105 03751-路由与转发

1. 路由器的两大核心功能

- a. 两大核心功能：路由选择（确定哪一条路径）和分组转发（当一个分组到达时所采用的动作）。
- b. 路由选择：根据路由算法确定一个进来的分组应该被传送到哪一条输出线上。如果子网内部使用数据报，那么对每一个进来的分组都要重新选择路径。如果子网内部使用虚电路，那么只有当创建一个新的虚电路时，才需要确定路由路径。
- c. 分组转发：就是路由器根据转发表将用户的 IP 数据报从合适的端口转发出去。

提醒 1：路由表是根据路由选择算法得出的，而转发表是从路由表得出的。

提醒 2：路由器可在网络之间转发分组（即 IP 数据报）。特别是，这些互联的网络可以是异构的。因此，如果是许多相同类型的网络互连在一起，那么用一个很大的交换机（如果能够找得到）代替原来的一些路由器是可以的。**如果这些互联的网络是异构的网络，那么就必须使用路由器来进行互联。**

0.106 03760-静态路由与动态路由

1. 路由算法

a. 分类：路由器转发分组是通过路由表转发的，而路由表是通过各种算法得到的。如果从路由算法能否随网络的通信量或拓扑自适应地进行调整变化来划分，则只有两大类，即**静态路由选择策略**（又称为非自适应路由选择）与**动态路由选择策略**（又称为自适应路由选择）。

b. 静态路由：特点是简单和开销小，但不能及时适应网络状态的变化。对于很小的网络，完全可以采用静态路由选择，自己手动配置每一条路由。

c. 动态路由：选择的特点是能较好地适应网络状态变化，但实现起来比较复杂，开销也较大。因此，动态路由适用于较复杂的网络。现代的计算机网络通常使用动态路由选择算法。动态路由算法又可分为两种基本类型：**距离-向量路由算法**和**链路状态路由算法**。

0.107 03763-层次路由

1. 自治系统

因特网将整个互联网划分为许多较小的自治系统（但是注意一个自治系统不是一个局域网，里面包含很多局域网），每个自治系统有权自主地决定本系统内应采用何种路由选择协议。但是问题出来了，如果

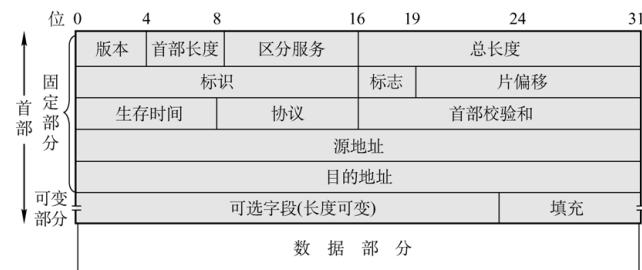
两个自治系统需要通信，并且这两个自治系统内部所使用的路由选择协议不同，那么怎么通信？所以就需要一种在两个自治系统之间的协议来屏蔽这些差异，就是楼下这些协议哦！

2. 路由选择协议的分类

- a. 内部网关协议：一个自治系统内部所使用的路由选择协议，具体的协议有 RIP 和 OSPF 等。自治系统内部的路由选择称为**域内路由选择**。
- b. 外部网关协议：自治系统之间使用的路由选择协议，具体的协议有 BGP。自治系统之间的路由选择称为**域间路由选择**。

0.108 03764-IPv4 分组

1. IP 数据报首部格式分析



1. **版本**。占 4 位，就是说这个 IP 数据报是 IPv4 版本还是 IPv6 版本，通信双方的版本必须一致。
2. **首部长度**。占 4 位，IP 数据报的首部实际上是 60B（但是有 40B 基本从不使用，考试的时候就认为 IP 数据报的首部是 20B，绝对不会错），前面也讲过 IP 数据报首部的长度必须是 4B 的倍数，这样就只要用 15 个标记（每个标记 4 位）就可以表示 60B。
3. **区分服务**。占 1B，从没使用过，不会考，可忽略。
4. **总长度**。占 2B，**千万不要和首部的基本长度弄混淆，这里的基本单位长度是 1B，不再是 4B，并且总长度包括了首部和数据部分**，很明显 16 位可以表示的长度为 65535B（因为 16 位表示的数的范围是 0 ~ 65535）。
5. **标识**。占 2B，它是一个计数器，用来产生 IP 数据报的标识。
6. **标志**。占 3 位，目前只有前 2 位有意义，即 MF 和 DF。
7. **片偏移**。占 13 位，前面已经讲过，但是需要注意的是片偏移是 8B 的整数倍。
8. **生存时间**。占 8 位，如果一个数据报一直在网络中转圈，网络资源就被白白浪费了，所以需要设置生存时间 (Time To Live, TTL)，即数据报在网络中可通过的路由器数的最大值。
9. **协议**。占 8 位，当接收端收到数据报时，肯定要交给传输层的某种协议去处理，是交给传输层的 TCP 协议，还是交给传输层的 UDP 协议，需要此标志给出。
10. **首部校验和**。占 16 位，只需记住只检验数据报的首部，不检验数据部分。
11. **源地址**。发送端主机的 IP 地址，即源地址。
12. **目的地址**。接收端主机的 IP 地址，即目的地址。

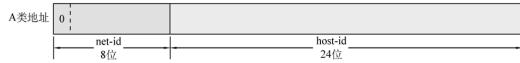
0.109 03765-IPv4 地址

1. IPv4 地址

把整个因特网看作一个单一的、抽象的网络。IP 地址就是给每个连接在因特网上的主机（或路由器）分配一个在全世界范围是唯一的 32 位的标识符。一般将 IP 地址分为 A 类地址、B 类地址、C 类地址、D 类地址和 E 类地址。D 类地址为组播地址，将在 IP 组播中讲解，E 类地址为保留地址，留作以后使用。

2. A 类地址

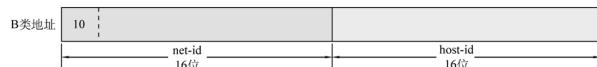
A类地址的网络号为前面8位，并且第一位规定为0，如下图所示。



A类地址可以指派的网络数为 $2^7 - 2$ ，每个A类网络上的最大主机数是 $2^{24} - 2$ 。

3. B类地址

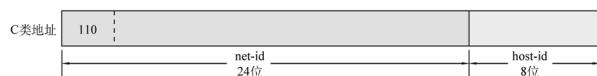
B类地址的网络号为前面16位，并且前面2位规定为10，如下图所示。



B类地址可以指派的网络数为 $2^{14} - 1$ 。每个B类网络上的最大主机数是 $2^{16} - 2$ 。

4. C类地址

C类地址的网络号为前面24位，并且前面3位规定为110，如下图所示。



C类地址可以指派的网络数为 $2^{21} - 1$ 。每个C类网络上的最大主机数是 $2^8 - 2$ 。

5. 六种特殊的IP地址

特殊地址	网络号	主机号	源地址或目的地址
网络地址	特定的	全0	都不是
直接广播地址	特定的	全1	目的地址
受限广播地址	全1	全1	目的地址
这个网络上的这个主机	全0	全0	源地址或默认目的地址
这个网络上的特定主机	全0	特定的	目的地址
环回地址	127	不是全0或全1	源地址或目的地址

0.110 03769-ARP

1. ARP

ARP是解决同一个局域网上的主机或路由器的IP地址和硬件地址的映射问题的。如果所要找的主机和源主机不在同一个局域网上，那么就要通过ARP找到一个位于本局域网上的某个路由器的硬件地址，然后把分组发送给这个路由器，让这个路由器把分组转发给下一个网络，剩下的工作就由下一个网络来做。

注意：ARP请求分组是广播发送的，但是ARP响应分组是普通的单播，即从一个源地址发送到一个目的地址。

2. ARP的四种应用场景

a. 发送方是主机，要把IP数据报发送到本网络上的另一个主机。这时用ARP找到目的主机的硬件地址。

b. 发送方是主机，要把IP数据报发送到另一个网络上的一个主机。这时用ARP找到本网络上的一个路由器的硬件地址，剩下的工作由这个路由器来完成。

c. 发送方是路由器，要把IP数据报转发到本网络上的一个主机。这时用ARP找到目的主机的硬件地址。

d. 发送方是路由器，要把IP数据报转发到另一个网络上的一个主机。这时用ARP找到本网络上的一个路由器的硬件地址，剩下的工作由这个路由器来完成。

既然ARP可以将IP地址转换成物理地址，那么有没有一种设备可以将物理地址转换成IP地址呢？RARP可以转换，但是基本已经被淘汰了，因为物理地址转换成IP地址这种功能已经被集成到了DHCP。

0.111 03770-DHCP

1. DHCP

a. 动态主机配置协议（DHCP）常用于给主机动态地分配 IP 地址。它提供了即插即用连网的机制，这种机制允许一台计算机加入新的网络和获取 IP 地址而不用手工参与。DHCP 是应用层协议，DHCP 报文使用 UDP 传输。

b. DHCP 服务器分配给 DHCP 客户的 IP 地址是临时的，因此 DHCP 客户只能在一段有限的时间内使用这个分配到的 IP 地址。DHCP 协议称这段时间为租用期。

2. DHCP 服务器和 DHCP 客户端的交换过程

1) DHCP 客户机广播“DHCP 发现”消息，试图找到网络中的 DHCP 服务器，服务器获得一个 IP 地址。

2) DHCP 服务器收到“DHCP 发现”消息后，就向网络中广播“DHCP 提供”消息，其中包括提供 DHCP 客户机的 IP 地址和相关配置信息。

3) DHCP 客户机收到“DHCP 提供”消息，如果接受 DHCP 服务器所提供的相关参数，则通过广播“DHCP 请求”消息向 DHCP 服务器请求提供 IP 地址。

4) DHCP 服务器广播“DHCP 确认”消息，将 IP 地址分配给 DHCP 客户机。

DHCP 协议允许网络上配置多台 DHCP 服务器，当 DHCP 客户发出 DHCP 请求时，就有可能收到多个应答信息。这时，DHCP 客户只会挑选其中的一个，通常是挑选“最先到达的信息”。

0.112 03771-ICMP

ICMP 报文分为两种，即 ICMP 差错报告报文和 ICMP 询问报文。

1. ICMP 差错报告报文的分类

- a. 终点不可达：当路由器或主机不能交付数据报时发；
- b. 源站抑制：当路由器或主机由于拥塞而丢弃数据报时发；
- c. 时间超过：当 IP 分组的 TTL 值被减为 0 后发；
- d. 参数问题：当路由器或目的主机收到的数据报的首部中有字段的值不正确时发；
- e. 改变路由（重定向）：有更好路由的时候发。

2. ICMP 询问报文的分类

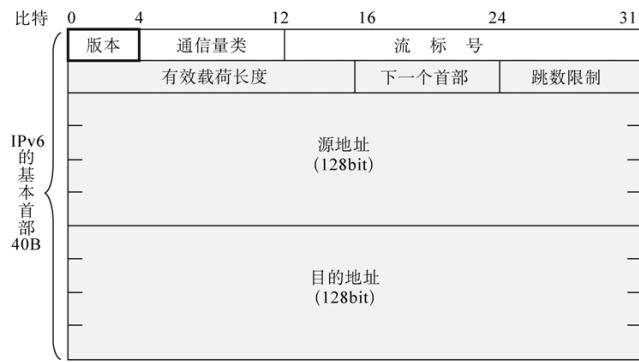
- 1. 有回送请求和回答报文；
- 2. 时间戳请求和回答报文；
- 3. 掩码地址请求和回答报文；
- 4. 路由器询问和通告报文。

3. 不应发送 ICMP 差错报告报文的几种情况

- 1. 对 ICMP 差错报告报文不再发送 ICMP 差错报告报文；
- 2. 对第一个分片的数据报片的所有后续数据报片都不发送 ICMP 差错报告报文；
- 3. 对具有组播地址的数据报都不发送 ICMP 差错报告报文；
- 4. 对具有特殊地址（如 127.0.0.0 或 0.0.0.0）的数据报不发送 ICMP 差错报告报文。

0.113 03773-IPv6 的格式

1. IPv6 的首部格式



- 1) **版本 (version)**。占 4 位，它指明了协议的版本，对于 IPv6，该字段总是 6。
- 2) **通信量类 (Traffic Class)**。占 8 位，这是为了区分不同的 IPv6 数据报的类别或优先级。
- 3) **流标号 (Flow Label)**。占 20 位，“流”是互联网络上从特定源点到特定终点的一系列数据报，“流”所经过的路径上的路由器都保证指明的服务质量。所有属于同一个流的数据报都具有同样的流标号。
- 4) **有效载荷长度 (Payload Length)**。占 16 位，它指明 IPv6 数据报除基本首部以外的字节数（所有扩展首部都算在有效载荷之内），其最大值是 64KB。
- 5) **下一个首部 (Next Header)**。占 8 位，它相当于 IPv4 的协议字段或可选字段。
- 6) **跳数限制 (Hop Limit)**。占 8 位，源站在数据报发出时即设定跳数限制，路由器在转发数据报时将跳数限制字段中的值减 1。当跳数限制的值为零时，就要将此数据报丢弃。
- 7) **源地址**。占 128 位，数据报的发送站的 IP 地址。
- 8) **目的地址**。占 128 位，数据报的接收站的 IP 地址。

2. IPv6 的 3 种地址类型

- 1) **单播**。传统的点对点通信。
- 2) **组播**。数据报交付到一组计算机中的每一个广播可看作是组播的一个特例。
- 3) **任播**。其目的站是一组主机，但数据报在交付时只交付给其中一个，通常是距离最近的那个。

3. IPv6 的地址表示法

为了使地址简洁，通常采用冒号十六进制法表示 IPv6 地址。它把每 16bit 用一个十六进制数表示，各值之间用冒号分隔，如 68E6:8C64:FFFF:0111:1180:960A:FFFF。

通常可以把 IPv6 地址缩写成更紧凑的形式。当 16 位域的开头有连续的 0 时，可以采用缩写法表示，但在域中必须至少有一个数字，如可以把地址 5ED4:0000:0000:0000:EBCD:045A: 000A:7654 缩写成 5ED4:0:0:0:EBCD:45A:A:7654。

当有相继的 0 值域时，还可以采用双冒号表示法进一步缩写。这些域可以用双冒号 (::)。但要注意，双冒号表示法在一个地址中仅可以出现一次，因为 0 值域的个数没有编码，需要从指定的总的域的个数中推算。这样，前述示范地址可以被更紧凑地书写成 5ED4::EBCD: 45A:A:7654。

0.114 03774 路由选择协议分类

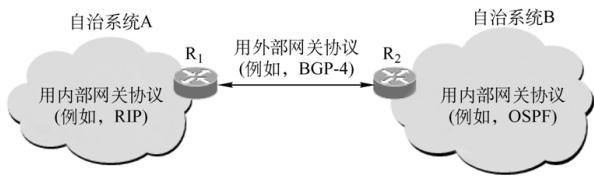
因特网有两大路由选择协议：

1. 内部网关协议 (IGP)

内部网关协议是在一个自治系统内部使用的路由选择协议，它与互联网中其他自治系统选用什么路由选择协议无关。目前这类路由选择协议使用得最多，如 RIP 和 OSPF 路由协议。

2. 外部网关协议 (EGP)

若源站和目的站处在不同的自治系统中，当数据报传到另一个自治系统的边界时（这两个自治系统可能使用不同的内部网关协议），就需要使用一种协议将路由选择信息传递到另一个自治系统中，如下图所示。这样的协议就是外部网关协议，如 BGP-4。



0.115 03775-RIP

1. 距离-向量算法

RIP 认为一个好的路由就是它通过的路由器的数目少，即“距离短”。RIP 允许一条路径最多只能包含 15 个路由器。“距离”的最大值为 16 时即相当于不可达。可见，RIP 只适用于小型互联网（因为比较大的自治系统里面的路由器的数量肯定会大大超过 15 个）。RIP 不能在两个网络之间同时使用多条路由。RIP 选择一个具有最少路由器的路由（即最短路由），哪怕还存在另一条高速（低时延）但路由器较多的路由。

2. RIP 的三要点

- 1) 仅和相邻路由器交换信息。
- 2) 交换的信息是当前本路由器所知道的全部信息，即自己的路由表。
- 3) 按固定的时间间隔（如每隔 30s）交换路由信息。

注意：RIP 选择的路径不一定是时间最短的，但一定是具有最少路由器的路径。因为它是根据最少的跳数进行路径选择的。

3. RIP 的优缺点

- a. **优点：**实现简单、开销小，收敛过程较快。
- b. **缺点：**
 - 1) RIP 限制了网络的规模，它能使用的最大距离为 15（16 表示不可达）。
 - 2) 路由器之间交换的路由信息是路由器中的完整路由表，因而随着网络规模的扩大，开销也就增加了。
 - 3) 当网络出现故障时，RIP 要经过比较长的时间才能将此信息传送到所有的路由器，即“坏消息传播得慢”，使更新过程的收敛时间长。

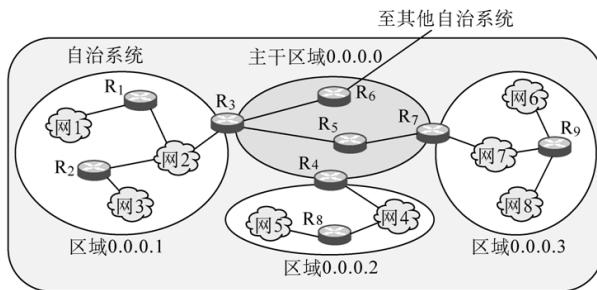
0.116 03776-OSPF

1. 链路状态协议

如果要算路由器 1 到路由器 2、3、4 的最短路径（给出邻接矩阵，即链路状态数据库），就可以将路由器 1 看成是起始结点，然后使用 3 次 Dijkstra 算法分别计算出路由器 1 到路由器 2、3、4 的最短路径，路由表就出来了。一旦网络拓扑又有变化，如以前没有相连的路由器，现在相连了，就又按照这样的步骤去计算路由表，这就是链路状态协议。

2. OSPF 协议介绍

为了使 OSPF 路由协议能够用于规模很大的网络，并且使其收敛得更快，OSPF 路由协议将一个自治系统再划分为若干个更小的范围，称为区域，如下图所示。



3. OSPF 协议三要点

- 1) 向本自治系统中所有路由器发送信息，这里使用的方法是**洪泛法**。
- 2) 发送的信息就是与本路由器相邻的所有路由器的链路状态，但这只是路由器所知道的部分信息。
- 3) “链路状态”就是说明本路由器都和哪些路由器相邻以及该链路的“度量”(metric)。

注意：只有当链路状态发生变化时，路由器才用洪泛法向所有路由器发送此信息。

4. OSPF 的 5 种分组类型

- 1) 类型 1。问候 (Hello) 分组，用来发现和维持邻站的可达性；
- 2) 类型 2。数据库描述分组，向邻站给出自己的链路状态数据库中的所有链路状态项目的摘要信息；
- 3) 类型 3。链路状态请求分组，向对方请求发送某些链路项目的详细信息；
- 4) 类型 4。链路状态更新分组，用洪泛法对全网更新链路状态；
- 5) 类型 5。链路状态确认分组，对链路更新分组的确认。

0.117 03777-RIP 和 OSPF 的比较

1. RIP 和 OSPF 的比较

a. 协议参数

RIP 中用于表示目的网络远近的参数为**跳数**，且参数被限制为最大 15。而 OSPF 协议，路由表中表示目的网络的参数为**费用**（如时延），该参数为一虚拟值，即 OSPF 路由信息不受物理跳数的限制。因此，OSPF 协议适合应用于大型网络。

b. 收敛速度

RIP 周期性地将整个路由表作为路由信息广播至网络中，该广播周期为 30s。在一个较大型的网络中，RIP 会产生很大的广播信息，占用较多的网络带宽资源，并且由于 RIP 30s 的广播周期，影响了 RIP 的收敛，甚至出现不收敛的现象。而 OSPF 是一种链路状态的路由协议，当网络比较稳定时，网络中的路由信息比较少，并且其广播也不是周期性的，因此 OSPF 路由协议在大型网络中也能够较快地收敛。

c. 分层

在 RIP 中，网络是一个平面的概念，并无区域及边界等的定义。在 OSPF 路由协议中，一个网络或者一个自治系统可以划分为很多个区域，每一个区域通过 OSPF 边界路由器相连。

d. 负载平衡

在 OSPF 路由选择协议中，如果到同一个目的网络有多条相同代价的路径，那么可以将通信量分配给这几条路径。这称为多路径间的**负载平衡**。而 RIP 不会，它只能按照一条路径传送数据。

e. 灵活性

OSPF 协议对于不同类型的业务可以计算出不同的路由，十分灵活。而这种灵活性是 RIP 所没有的。

f. 以组播地址发送报文

RIP 使用**广播**报文来发送给网络上的所有设备，而 OSPF 协议采用**组播**地址来发送，只有运行 OSPF 协议的设备才会接收发送来的报文，其他设备不参与接收。

0.118 03778-BGP

1. BGP 的基本原理

每一个自治系统的管理员要选择至少一个路由器（可以有多个）作为该自治系统的“BGP 发言人”。一个 BGP 发言人要与其他自治系统中的 BGP 发言人交换路由信息，就要先建立 TCP 连接，然后在此连接上交换 BGP 报文以建立 BGP 会话，再利用 BGP 会话交换路由信息。各 BGP 发言人互相交换网络可达性的信息后，各 BGP 发言人就可找出到达各自自治系统比较好的路由了。

2. BGP 的特点

- 1) BGP 交换路由信息的结点数量级是自治系统的量级，这要比自治系统中的网络数少很多。
- 2) 每一个自治系统中 BGP 发言人（或边界路由器）的数目是很少的，这样就使得自治系统之间的路由选择不过分复杂。
- 3) BGP 支持 CIDR，因此 BGP 的路由表也就应当包括目的网络前缀、下一跳路由器以及到达该目的网络所要经过的各个自治系统序列。
- 4) 在 BGP 刚刚运行时，BGP 的邻站是交换整个的 BGP 路由表，但以后只需要在发生变化时更新有变化的部分。这样对节省网络带宽和减少路由器的处理开销方面都有好处。

3. BGP 的 4 种报文

- 1) 打开报文。用来与相邻的另一个 BGP 发言人建立关系。
- 2) 更新报文。用来发送某一路由的信息以及列出要撤销的多条路由。
- 3) 保活报文。用来确认打开报文和周期性地证实邻站关系。
- 4) 通知报文。用来发送检测到的差错。

0.119 03779-RIP、OSPF、BGP 最终陈述

主要特点	RIP	OSPF	BGP
网关协议	内部	内部	外部
路由表内容	目的网络，下一跳，距离	目的网络，下一跳，距离	目的网络，完整路径
最优通路依据	跳数	费用	多种有关策略
算法	距离-向量协议	链路状态协议	路径-向量协议
传送方式	传输层 UDP	IP 数据报	建立 TCP 连接
其他	简单、效率低、跳数为 16 不可达； 好消息传得快，坏消息传得慢	效率高、路由器频繁交换信息，难维持一致性；规模大、统一度量为可达性	

0.120 03780-组播的概念

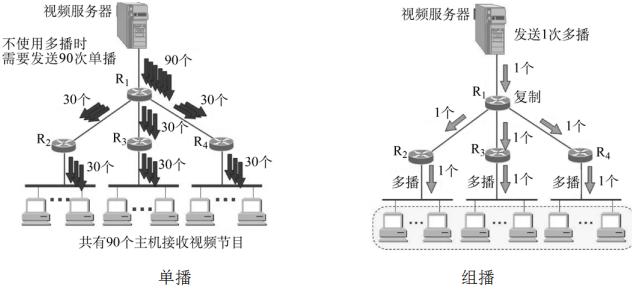
1. 组播地址介绍

因为组播需要一对多的发送，所以组播一定是仅应用于 UDP。而 TCP 是一个面向连接的协议，它意味着分别运行于两台主机（由 IP 地址来确定）内的两个进程（由端口号来确定）之间存在一条连接，所以是一对一的发送。

2. IP 组播的思想

源主机只发送一份数据，该数据中的目的地址为组播的组地址。组地址中的所有接收者都可以接收到同样的数据副本，并且只有组播内的主机可以接收数据，网络中的其他主机不可能收到该数据。

与广播所不同的是，主机组播时仅发送一份数据，组播的数据仅在传送路径分岔时才将数据报复制后继续转发，单播和组播的传播过程如下图所示。采用组播协议可明显地减轻网络中各种资源的消耗。组播需要路由器的支持才能实现，能够运行组播协议的路由器称为组播路由器。



0.121 03783-移动 IP 的概念

1. 移动 IP 的概念

引入背景：随着移动终端设备的广泛使用，移动计算机和移动终端等设备也开始需要接入网络（Internet），但传统的 IP 设计并未考虑到移动结点会在链接中变化互联网接入点的问题。

2. 传统 IP 地址的意义

- 1) 用来标识唯一的主机；
- 2) 作为主机的地址在数据的路由中起重要作用。

但对于移动结点，由于互联网接入点会不断发生变化，所以其 IP 地址在两方面发生分离，一方面是移动结点需要一种机制来唯一标识自己，另一方面是需要这种标识不会被用来路由。而移动 IP 便是为了让移动结点能够分离 IP 地址这两方面功能，而又不彻底改变现有互联网的结构而设计的。

例如，某用户离开北京总公司，出差到上海分公司时，只要简单地将移动结点（如笔记本电脑）连接至上海分公司网络上，那么用户就可以享受到跟在北京总公司里一样的所有操作。用户依旧能使用北京总公司的共享打印机，或者可以依旧访问北京总公司同事计算机里的共享文件及相关数据库资源；而当该计算机移动到外网的话，尽管 IP 地址没变，但是不能再用这个 IP 地址来找寻路由了，而应该申请一个转交地址，由转交地址来实现找路由功能。诸如此类的种种操作，让用户感觉不到自己身在外地，同事也感觉不到你已经出差到外地了。总结来说就是移动 IP 技术可以使移动结点以固定的网络 IP 地址，实现跨越不同网段的漫游功能，并保证了基于网络 IP 的网络权限在漫游过程中不发生任何改变。

0.122 03784-移动 IP 的通信过程

1. 移动 IP 技术的通信过程

- 1) 移动结点在本地网时，按传统的 TCP/IP 方式进行通信（在本地网有固定的地址）。
- 2) 移动结点漫游到一个外地网络时，仍然使用固定的 IP 地址进行通信。为了能够收到通信对端发给它的 IP 分组，移动结点需要向本地代理注册当前的位置地址，这个位置地址就是转交地址。移动 IP 的转交地址可以是外部代理的地址或动态配置的一个地址。
- 3) 本地代理接收来自转交地址的注册后，会构建一条通向转交地址的隧道，将截获的发给移动结点的 IP 分组通过隧道送到转交地址处。
- 4) 在转交地址处解除隧道封装，恢复出原始的 IP 分组，最后送到移动结点，这样移动结点在外网就能够收到这些送给它的 IP 分组了。
- 5) 移动结点在外网通过外网的路由器或者外代理向通信对端发送 IP 数据报。
- 6) 当移动结点来到另一个外网时，只需要向本地代理更新注册的转交地址，就可以继续通信了。

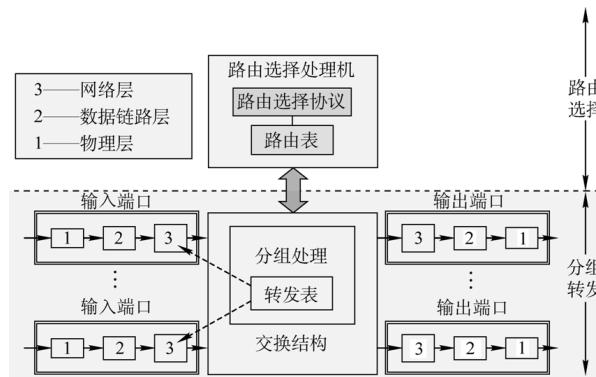
7) 当移动结点回到本地网时, 移动结点向本地代理注销转交地址, 这时移动结点又将使用传统的 TCP/IP 方式进行通信。

总结: 移动 IP 为移动主机设置了两个 IP 地址, 即主地址和辅地址 (转交地址)。移动主机在本地网时, 使用的是主地址。当移动到另外一个网络时, 需要获得一个辅助的临时地址, 但是此时主地址仍然保持不变。当从外网移回本地网时, 辅地址改变或撤销, 而主地址仍然保持不变。

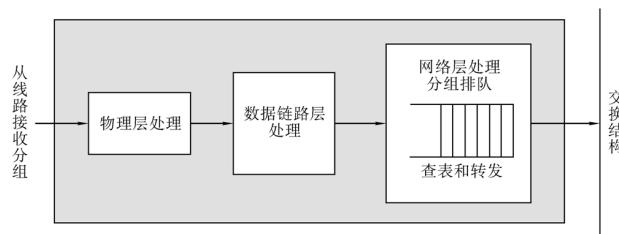
0.123 03786-路由表与路由转发

1. 路由器的工作流程

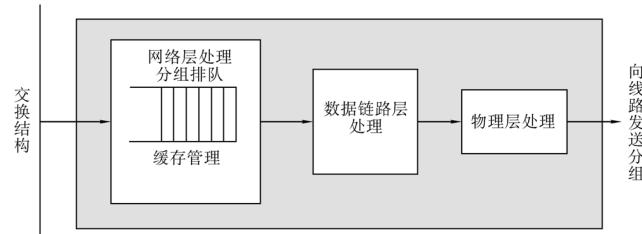
第一步: 首先路由器从线路上接收分组, 也就是下图的输入端口, 经过 1 时进行物理层处理 (进行比特的接收), 经过 2 时进行数据链路层处理 (剥去帧头、帧尾, 得到了 IP 数据报), 然后分组就被送入网络层的模块。



第二步: 网络层模块见下图。若接收的分组是路由器之间交换路由信息的分组 (如 RIP 和 OSPF 分组), 则把这种分组送交路由器的路由选择部分中的**路由选择处理器**。若接收的是数据分组, 则按照分组首部中的目的地址查找转发表, 根据得到的结果, 分组就经过交换结构到达合适的输出端口。当一个分组正在查找转发表时, 后面又紧跟着从这个输入端口收到另一个分组, 这个分组就必须在队列中排队, 因而产生了一定的时延, 见下图。



第三步: 从交换结构传送过来的分组先进行**缓存**, 数据链路层处理模块将给分组加上数据链路层的首部和尾部, 交给**物理层**后发送到外部线路, 如下图所示。



0.124 03787-传输层的功能

1. 传输层的功能

从通信和信息处理的角度看，传输层是 5 层参考模型中的第 4 层，它向上面的应用层提供通信服务。它属于面向通信部分的最高层，同时也是用户功能中的最低层。

传输层为两台主机提供了应用进程之间的通信，又称为端到端通信。由于网络层协议是不可靠的，会使分组丢失、失序和重复等，所以派出传输层为数据传输提供可靠的服务。

功能一：提供应用进程间的逻辑通信

“逻辑通信”的意思是传输层之间的通信好像是沿水平方向传送数据，但事实上这两个传输层之间并没有一条水平方向的物理连接。

功能二：差错检测

对收到报文的首部和数据部分都进行差错检测（网络层只检查 IP 数据报首部，并不检查数据部分）。

功能三：提供无连接或面向连接的服务

根据应用的不同，如有些数据传输要求实时性（如实时视频会议），传输层需要有两种不同的传输协议，即面向连接的 TCP 和无连接的 UDP。TCP 提供了一种可靠性较高的传输服务，UDP 则提供了一种高效率的但不可靠的传输服务。

功能四：复用和分用

复用是指发送方不同的应用进程都可以使用同一个传输层协议传送数据。分用是指接收方的传输层在剥去报文的首部后能够把这些数据正确交付到目的应用进程。

0.125 03789-无连接服务与面向连接服务

1. 传输层的服务介绍

传输层提供了两种类型的服务：**无连接服务**和**面向连接服务**。相应的实现分别是**用户数据报协议 (UDP)** 和 **传输控制协议 (TCP)**。当采用 TCP 时，传输层向上提供的是一条全双工的可靠的逻辑信道；当采用 UDP 时，传输层向上提供的是一条不可靠的逻辑信道。

2. UDP 的主要特点

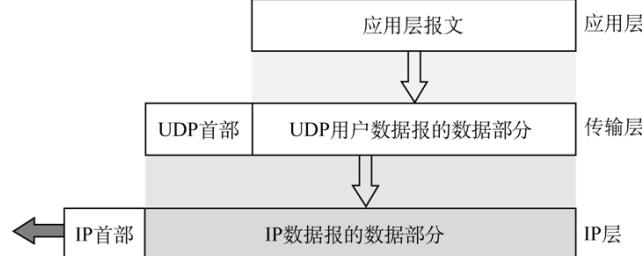
- 1) 传送数据前无需建立连接，数据到达后也无需确认。
- 2) 不可靠交付。
- 3) 报文头部短，传输开销小，时延较短。

3. TCP 的主要特点

- 1) 面向连接，不提供广播或多播服务。
- 2) 可靠交付。
- 3) 报文段头部长，传输开销大。

4. UDP 数据报与 IP 分组的区别

IP 分组要经过互联网中许多路由器的存储转发，但 UDP 数据报是在传输层的端到端抽象的逻辑信道中传送的，**UDP 数据报只是 IP 数据报中的数据部分**（见下图），对路由器是不可见的。



0.126 03790-UDP 报文段

1. UDP 的基本概念

UDP 和 TCP 最大的区别在于它是无连接的，UDP 其实只在 IP 的数据报服务之上增加了端口的功能（为了找到进程）和差错检测的功能。

虽然 UDP 用户数据报只能提供不可靠的交付，但 UDP 在某些方面有其特殊的优点。

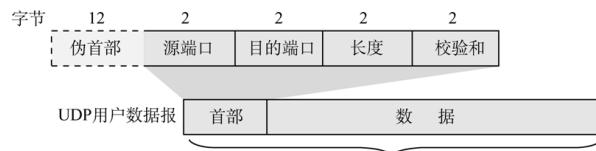
- 1) 发送数据之前不需要建立连接。
- 2) UDP 的主机不需要维持复杂的连接状态表。
- 3) UDP 用户数据报只有 8 个字节的首部开销。

4) 网络出现的拥塞不会使源主机的发送速率降低。

5) UDP 支持一对一、一对多、多对一和多对多的互通通信。

2. UDP 数据报的组成

UDP 数据报有两个字段：数据字段和首部字段。首部字段有 8B，由 4 个字段组成，如下图所示。



1) 源端口。占 2B。前面已经说了使用 16bit 来表示端口号，所以需要 2B 长度。

2) 目的端口。占 2B。

3) 长度。占 2B。

4) 校验和。占 2B。用来检测 UDP 用户数据报在传输中是否有错（既检验首部，又检验数据部分），如果有错，就直接丢弃；若该字段为可选字段，当源主机不想计算校验和时，则直接令该字段为全 0。**检验范围：伪首部、UDP 数据报的首部和数据。其中，在计算校验和时临时生成伪首部。**

0.127 03791-UDP 校验

1. UDP 校验

a. UDP 校验只提供差错检测。

b. 在计算校验和时，要在 UDP 用户数据报之前临时加上 12B 的伪首部。其中，伪首部包括源 IP 地址字段、目的 IP 地址字段、全 0 字段、协议字段（UDP 固定为 17）、UDP 长度字段。

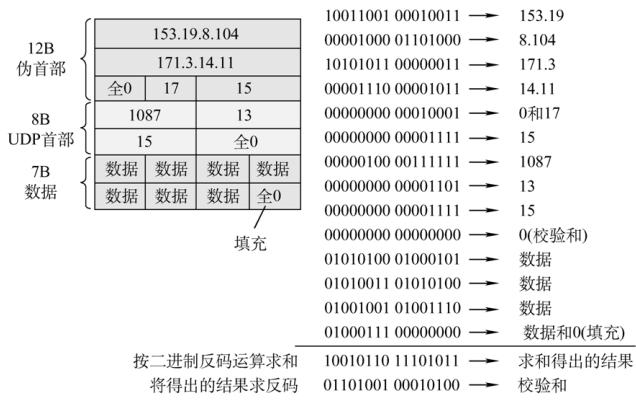
注意：一定要记住伪首部只用于计算和验证校验和，其既不向下传送，也不向上递交。

2. UDP 校验的注意事项

1) 校验的时候若 UDP 数据报数据部分的长度不是偶数字节，则需要填入一个全 0B，如下图所示。但是此字节和伪首部一样，是不发送的。

2) 如果 UDP 校验和校验出 UDP 数据报是错误的，可以丢弃，也可以交付给上层，但是需要附上错误报告，即告诉上层这是错误的数据报。

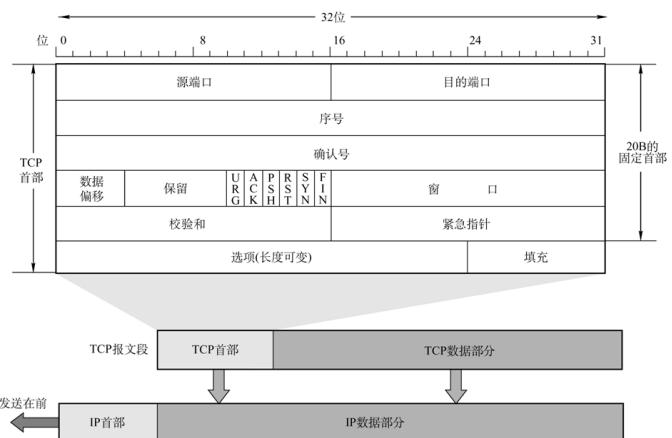
3) 通过伪首部，不仅可以检查源端口号、目的端口号和 UDP 用户数据报的数据部分，还可以检查 IP 数据报的源 IP 地址和目的地址。



按二进制反码计算后，当无差错时其结果应该为全 1；否则就表明有差错出现，接收方就应该丢弃这个 UDP 报文。

0.128 03792-TCP 报文段

1. TCP 报文的首部格式



1) 源端口和目的端口。各占 2B。

2) 序号。占 4B。尽管从应用层交付下来的是 TCP 报文段，但是 TCP 是面向字节流的（就是说 TCP 传送时是按照一个个字节来传送的），所以在一个 TCP 连接中传送的字节流需要编号，这样才能保证按序交付。

3) 确认号。占 4B。TCP 是含有确认机制的，所以接收端需要给发送端发送确认号。

4) 数据偏移。占 4 位。占 4 位可表示 0001 ~ 1111 一共 15 种状态，而基本单位是 4B，所以数据偏移确定了首部最长为 60B。

5) 保留字段。占 6 位。保留为今后使用。

6) 紧急 URG。当 URG=1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送（相当于高优先级的数据）。

7) 确认比特 ACK。只有当 ACK1 时确认号字段才有效。当 ACK0 时，确认号无效。TCP 规定，一旦连接建立了，所有传送的报文段都必须把 ACK 置 1。

8) 推送比特 PSH。TCP 收到推送比特置 1 的报文段，就尽快地交付给接收应用进程，而不再等到整个缓存都填满后再向上交付。

9) 复位比特 RST。当 RST1 时，表明 TCP 连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立传输连接。

10) 同步比特 SYN。同步比特 SYN 置为 1，表示这是一个连接请求或连接接受报文，后面 TCP 连接会详细讲到。

11) 终止比特 FIN。释放一个连接。当 FIN1 时，表明此报文段的发送端的数据已发送完毕，并要求释放传输连接。

12) 窗口字段。占 2B。窗口字段用来控制对方发送的数据量，单位为字节 (B)。记住一句话：窗口字段明确指出了现在允许对方发送的数据量。

13) 校验和字段。占 2B。校验和字段检验的范围包括首部和数据两部分。在计算校验和时，和 UDP 一样，要在 TCP 报文段的前面加上 12B 的伪首部（只需将 UDP 伪首部的第 4 个字段的 17 改为 6，其他和 UDP 一样）。

14) 紧急指针字段。占 2B。前面已经讲过紧急指针指出在本报文段中的紧急数据的最后一个字节的序号。

15) 选项字段。长度可变。TCP 最初只规定了一种选项，即最大报文段长度 MSS。MSS 告诉对方 TCP：“我的缓存所能接收的报文段的数据字段的最大长度是 MSS 字节。”

16) 填充字段。为了使整个首部长度是 4B 的整数倍。

0.129 03793-TCP 连接管理

1. TCP 连接的建立

TCP 的连接建立采用“三次握手”的方法，目的是为了防止报文段在传输连接建立过程中出现差错。通过 3 次报文段的交互后，通信双方的进程之间就建立了一条传输连接，然后就可以用全双工的方式在该传输连接上正常地传输数据报文段了。

- 1) SYN=1, seq=x。
- 2) SYN=1, ACK=1, seq=y, ack=x+1。
- 3) ACK=1, seq=x+1, ack=y+1。

2. TCP 连接的释放

一旦数据传输结束，参与传输的任何一方都可以请求释放传输连接。在释放连接过程中，发送端进程与接收端进程要通过 4 次 TCP 报文段来释放整个传输连接。

- 1) FIN=1, seq=u。
- 2) ACK=1, seq=v, ack=u+1。
- 3) FIN=1, ACK=1, seq=w, ack=u+1。
- 4) ACK=1, seq=u+1, ack=w+1。

提醒：不管是连接还是释放，SYN、ACK、FIN 的值一定是 1。

0.130 03794-TCP 可靠传输

1. TCP 的数据编号与确认

a. TCP 协议是面向字节的。TCP 将所要传送的报文看成是字节组成的数据流，并使每一个字节对应于一个序号。在连接建立时，双方要商定初始序号。

b. TCP 的确认是对接收到的数据的最高序号表示确认。因此，确认号表示接收端期望下次收到的数据中的第一个数据字节的序号。

c. 在后退 N 帧协议中，如果帧不按序到达，直接丢弃后面的，没有使用选择确认。

d. 在选择重传协议中，先把后面有序的帧存在接收缓冲区中，等到前面失序的帧到达后，一起按序交付，这里就用到选择确认 SACK。

2. TCP 的重传机制

TCP 每发送一个报文段，就对这个报文段设置一次计时器。只要计时器设置的重传时间到了规定时间还没有收到确认，那么就要重传该报文段。TCP 采用了一种自适应的算法，如下：

第一步：记录每个报文段发出的时间以及收到相应的确认报文段的时间。这两个时间之差就是报文段的往返时延。

第二步：将各个报文段的往返时延样本加权平均，就得出报文段的平均往返时延 (RTT)。

第三步：每测量到一个新的往返时延样本，就按下式重新计算一次平均往返时延。

$$RTT = (1-a) \times (\text{旧的 RTT}) + a \times (\text{新的往返时延样本})$$

在上式中， $0 < a < 1$ 。若 a 很接近于 1，表示新算出的平均往返时延 RTT 和原来的值相比变化较大，即 RTT 的值更新较快。若选择 a 接近于 0，则表示加权计算的 RTT 受新的往返时延样本的影响不大，即 RTT 的值更新较慢，一般推荐 a 取 0.125。

计时器的超时重传时间 (RTO) 应略大于上面得出的 RTT，即 $RTO = \beta \times RTT$ (其中 $\beta > 1$)

0.131 03796-TCP 拥塞控制的基本概念

1. 拥塞控制

a. 条件：出现资源拥塞的条件是对资源需求的总和 > 可用资源；

b. 后果：若网络中产生拥塞，网络的性能就要明显变坏，整个网络的吞吐量将随输入负荷的增大而下降

2. 拥塞控制与流量控制的区别

1) 拥塞控制所要做的只有一个前提，就是使得网络能够承受现有的网络负荷。

2) 拥塞控制是一个全局性的过程，涉及所有的主机、所有的路由器以及与降低网络传输性能有关的所有因素。

3) 流量控制往往指在给定的发送端和接收端之间的点对点通信量的控制。

4) 流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。

5) 拥塞控制是很难设计的，因为它是一个动态的（而不是静态的）问题。

6) 当前网络正朝着高速化的方向发展，这很容易出现缓存不够大而造成分组的丢失。但分组的丢失是网络发生拥塞的征兆而不是原因。

7) 在许多情况下，甚至正是拥塞控制本身成为引起网络性能恶化甚至发生死锁的原因，这点应特别引起重视。

3. 拥塞控制的分类

1) 开环控制：在设计网络时事先将有关发生拥塞的因素考虑周到，力求网络在工作时不产生拥塞。

2) 闭环控制：是基于反馈环路的概念。属于闭环控制的有以下几种措施：

监测网络系统以便检测到拥塞在何时、何处发生；

将拥塞发生的信息传送到可采取行动的地方；

调整网络系统的运行以解决出现的问题。

0.132 03797-拥塞控制的 4 种算法

算法一：慢开始算法的原理

步骤一：在主机刚刚开始发送报文段时可先设置拥塞窗口 $cwnd=1$ ，即设置为一个最大报文段 MSS 的数值；

步骤二：在每收到一个对新的报文段的确认后，将拥塞窗口加 1，即增加一个 MSS 的数值；

步骤三：用这样的方法逐步增大发送端的拥塞窗口 $cwnd$ ，可以使分组注入到网络的速率更加合理。

算法二：拥塞避免算法的原理

为防止拥塞窗口 cwnd 的增长引起网络阻塞，还需要一个状态变量，即慢开始门限 ssthresh，其用法如下：

当 $cwnd < ssthresh$ 时，使用慢开始算法。

当 $cwnd > ssthresh$ 时，停止使用慢开始算法，改用拥塞避免算法。

当 $cwnd = ssthresh$ 时，既可以使用慢开始算法，也可以使用拥塞避免算法。

其中，拥塞避免算法的做法是，发送端的拥塞窗口 cwnd 每经过一个往返时延 RTT 就增加一个 MSS 的大小。通常表现为按线性规律增长。

无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有按时收到确认），就要把慢开始门限 ssthresh 设置为出现拥塞时的发送窗口值的一半（但不能小于 2），然后把拥塞窗口 cwnd 重新设置为 1，执行慢开始算法。这样做的目的就是要迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

算法三：快重传算法

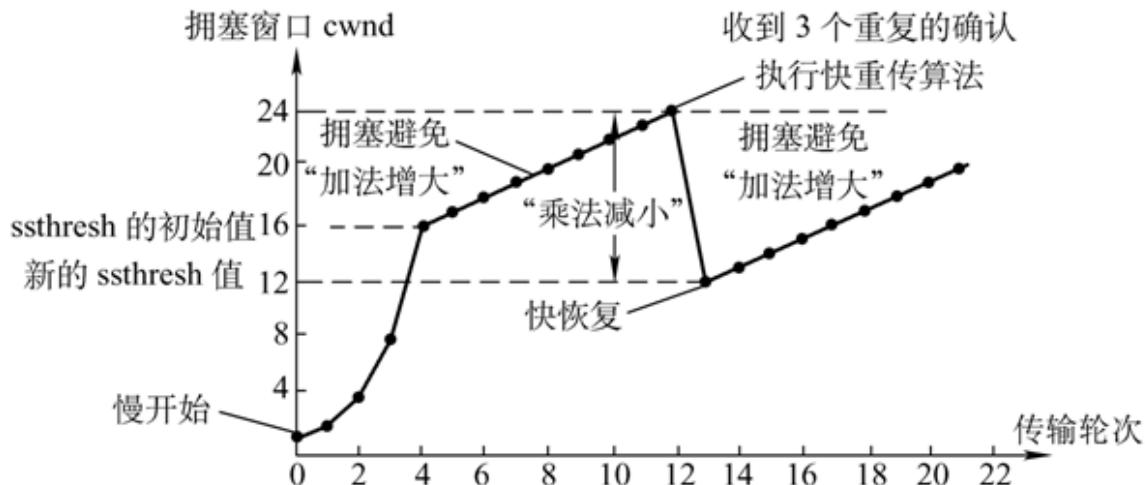
首先要求接收方每收到一个失序的报文段后就立即发出重复确认。这样做可以让发送方及早知道有报文段没有到达接收方。发送方只要连续收到 3 个重复确认就应当立即重传对方尚未收到的报文段。

算法四：快恢复算法

步骤一：当发送端收到连续 3 个重复的确认时，就执行“乘法减小”算法，把慢开始门限 ssthresh 设置为当前拥塞窗口的一半。但接下去不执行慢开始算法；

步骤二：由于发送方现在认为网络很可能没有发生拥塞，所以现在不执行慢开始算法，即拥塞窗口 cwnd 现在不设置为 1，而是将慢开始门限 ssthresh 设置为当前拥塞窗口的一半，然后开始执行拥塞避免算法（“加法增大”），使得拥塞窗口缓慢地线性增大，如下图所示；

步骤三：若发送窗口值还允许发送报文段，就按拥塞避免算法继续发送报文段。



0.133 03798-C-S 模型与 P2P 模型

1. 客户/服务器模型 (C/S 模型)

客户 (Client) 和服务器 (Server) 都是指通信中所涉及的两个应用进程。客户/服务器模型所描述的是进程之间的服务和被服务的关系。在这个模型中，客户是服务的请求方，服务器是服务的提供方。

客户/服务器模型主要特点：

- 1) 网络中各计算机的地位不平等，服务器通过对用户权限的限制来达到管理客户机的目的，使它们不能随意存储数据，更不能随意删除数据，或进行其他受限的网络活动；
- 2) 整个网络的管理工作由少数服务器承担；

3) 可扩展性不佳，由于受服务器硬件和网络带宽的限制，服务器所能支持的客户数比较有限，当客户数增长较快时，会急剧影响网络应用系统的效率。

2. P2P 模型

实际上，P2P 模型从本质上来看仍然是使用客户/服务器方式，只是对等连接中的每一个主机既是客户又是服务器。

P2P 模型带来的好处是，任何一台主机都可以成为服务器，改变了原来需要专用服务器的模式，很显然，多个客户机之间可以直接共享文档。此外，可以借助 P2P 网络模型，解决专用服务器的性能瓶颈问题。

P2P 模型主要特点如下：

1) 繁重的计算机任务可以被分配到各个结点上，利用每个结点空闲的计算能力和存储空间，聚合实现强大的服务。

2) 系统可扩展性好。传统的服务器有连接带宽的限制，只能达到一定的客户端连接数。但是在 P2P 模型中，能避免这个问题。

3) 网络更加健壮，不存在中心结点失效的问题。当一部分结点连接失败之后，其余的结点仍然能形成完整的网络。

0.134 03799-DNS 系统的概念

1. DNS 系统的概念

a. 引入背景：如果现在不允许你通过 www.koudaitiku.com 来访问口袋题库的主页，请问还有什么方式？直接使用存放口袋题库主页的服务器的 IP 地址（115.238.240.66）来访问。当然，如果现在整个网络只有几个网站，人们通过记忆其服务器的 IP 地址来访问还是可以勉强记住的，但是成千上万的网站都需要记忆其服务器的 IP 地址才能正确访问，相信没有几个用户能爱上网络，于是就出现了域名。这样，人们就可以通过便于记忆的域名来访问网站了。但是这个是表面上的，其实真正访问还是要通过 IP 地址，那么我们自然就想到应该存在一个东西，即可以将域名转换成相对应的 IP 地址，于是 DNS 系统就诞生了。

b. DNS 的组成：从概念上可以将 DNS 分为 3 个部分，分别是层次域名空间、域名服务器、解析器。

0.135 03801-域名服务器

1. 域名服务器的概念

因特网的域名系统（DNS）被设计成一个联机分布式的数据库系统，并采用客户/服务器模型。名字到域名的解析是由若干个域名服务器来完成的，域名服务器程序在专设的结点上运行，运行该程序的机器称为域名服务器。

一个服务器所负责管辖的（或有权限的）范围称为区（zone）。每一个区设置相应的权限域名服务器，用来保存该区中的所有主机的域名到 IP 地址的映射。DNS 服务器的管辖范围不是以“域”为单位，而是以“区”为单位，区一定小于或等于域。

2. 域名服务器的分类

1) 根域名服务器（最高层次的域名服务器）。根域名服务器是最重要的域名服务器。所有的根域名服务器都知道所有的顶级域名服务器的域名和 IP 地址。不管是哪一个本地域名服务器，若要对因特网上任何一个域名进行解析，只要自己无法解析，就首先求助于根域名服务器。

注意：根域名服务器用来管辖顶级域名（如.com），它并不直接把待查询的域名转换成 IP 地址，而是告诉本地域名服务器下一步应当找哪一个顶级域名服务器进行查询。

2) 顶级域名服务器。这些域名服务器负责管理在该顶级域名服务器注册的所有二级域名。当收到 DNS 查询请求时，就给出相应的回答。

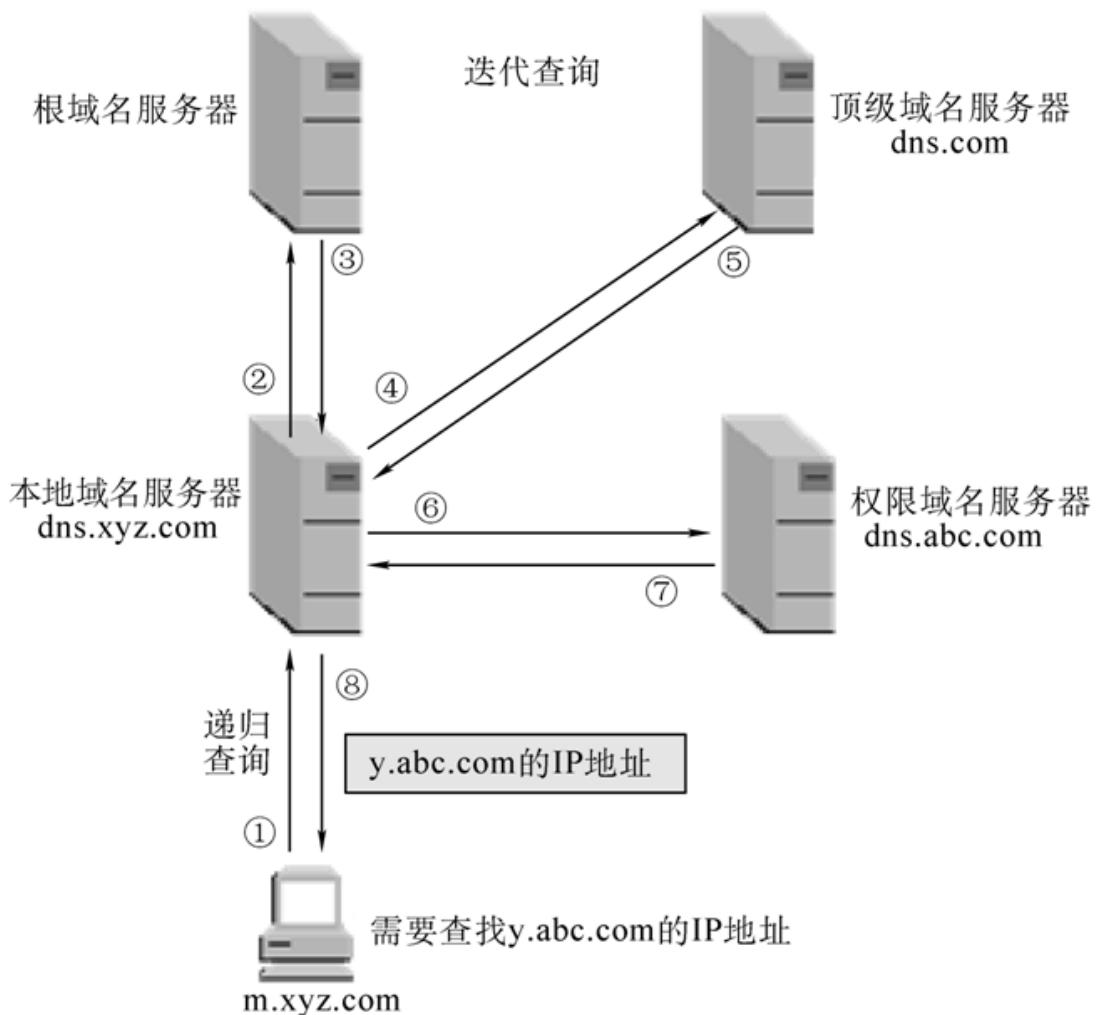
3) 权限域名服务器（授权域名服务器）。负责一个区的域名服务器。当一个权限域名服务器还不能给出最后的查询回答时，就会告诉发出查询请求的 DNS 客户，下一步应当找哪一个权限域名服务器。

4) 本地域名服务器。本地域名服务器对域名系统非常重要。当一个主机发出 DNS 查询请求时，这个查询请求报文就发送给本地域名服务器。每一个因特网服务提供者（ISP）或一个大学，甚至一个大学里的系，都可以拥有一个本地域名服务器，这种域名服务器有时也称为默认域名服务器。人们在使用本地连接时，就需要填写 DNS 服务器，而这个就是本地 DNS 服务器的地址。

0.136 03802-域名解析过程

1. 迭代查询

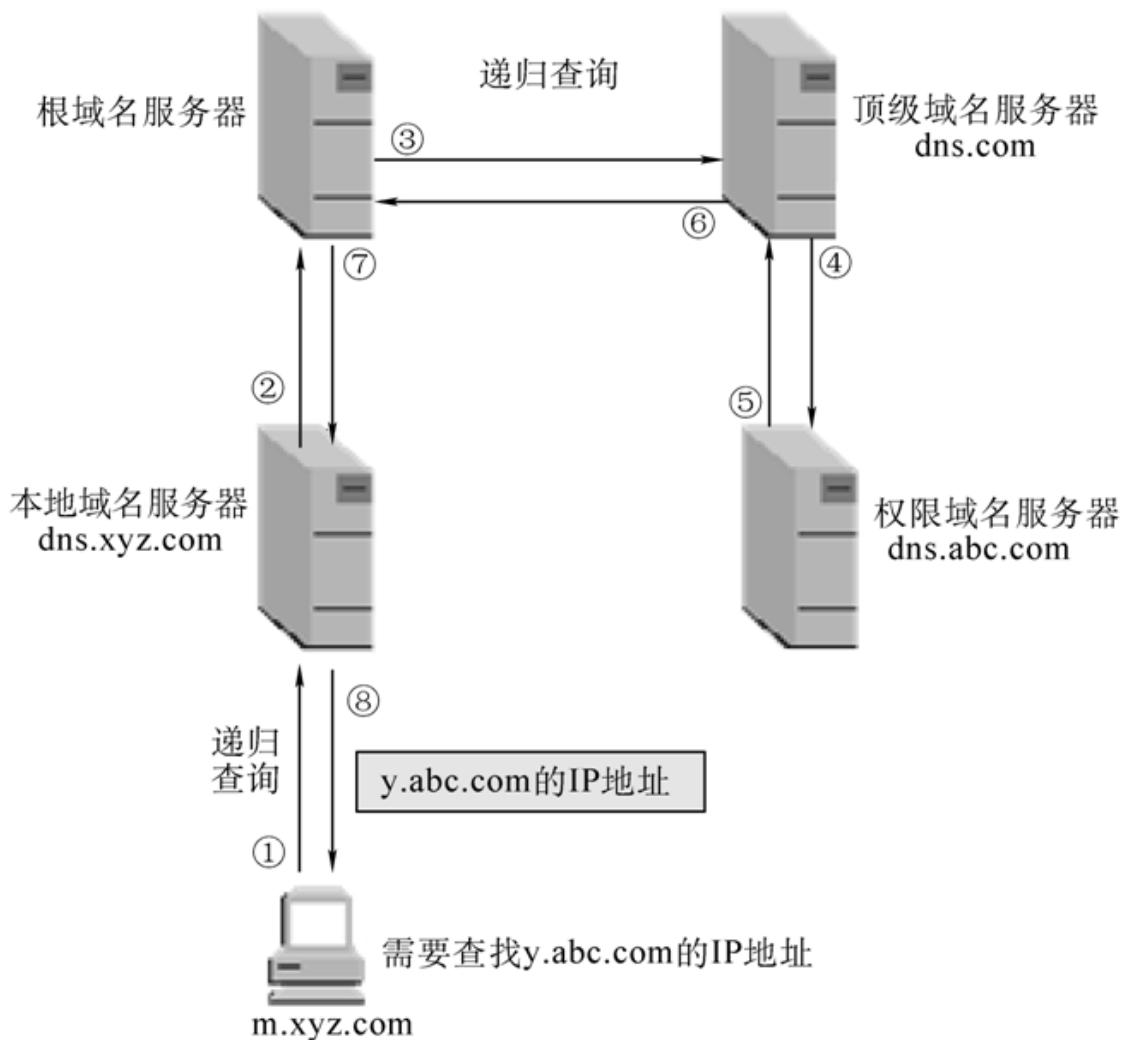
当根域名服务器收到本地域名服务器的迭代查询请求报文时，要么给出所要查询的 IP 地址，要么告诉本地域名服务器“下一步应当向哪一个域名服务器进行查询”，然后让本地域名服务器进行后续的查询，如下图所示。



注意：因为主机向本地域名服务器的查询都是采用递归查询，所以迭代查询又称为递归与迭代相结合的查询方式。相比递归查询，这种方式更常用。

2. 递归查询

递归查询是指本地域名服务器只需向根域名服务器查询一次，后面的几次查询都是在其他几个域名服务器之间进行的（见下图步骤 ~ ）。在步骤 中，本地域名服务器从根域名服务器得到了所需的 IP 地址，最后在步骤 中，本地域名服务器把查询结果告诉主机 `m.xyz.com`。



0.137 03803-FTP 工作原理

1. FTP 协议的基本介绍

a. 基本概念：文件传送协议（FTP）是因特网上使用的最广泛的传送协议。FTP 提供交互式的访问，允许客户指明文件的类型与格式，并允许文件具有存取权限。FTP 屏蔽了各计算机系统的细节，因而适合于在异构网络中任意计算机之间传送文件。

b. 提供的服务：FTP 只提供文件传送的一些基本服务，它使用 TCP 可靠地传输服务。FTP 使用客户/服务器模型。一个 FTP 服务器进程可同时为多个客户进程提供服务。

2. FTP 协议的工作原理

FTP 的服务器进程由两大部分组成：一个主进程负责接收新的请求；另外有若干个从属进程，负责处理单个请求。主进程的工作步骤如下：

1) 打开熟知端口（端口号为 21），使客户进程能够连接上。

2) 等待客户进程发出连接请求。

3) 启动从属进程来处理客户进程发来的请求。从属进程对客户进程的请求处理完毕后即终止，但从属进程在运行期间根据需要还可能创建一些其他子进程。

4) 回到等待状态，继续接收其他客户进程发来的请求。主进程与从属进程的处理是并发进行的。

0.138 03804-控制连接与数据连接

1. FTP 协议的控制与数据连接

a. 在进行文件传输时，FTP 的客户和服务器之间要建立两个 TCP 连接，一个用于传输控制命令和响应，称为**控制连接**；另一个用于实际的文件内容传输，称为**数据连接**。

b. 服务器监听在 21 号端口，等待客户连接，建立在这个端口上的连接称为**控制连接**，客户机可以通过这个连接向服务器发送各种请求，如登录、改变当前目录、切换数据传输模式、列目录内容、上传文件等。当需要传送文件时，服务器和客户机之间要建立另外一个连接，服务器监听在 20 号端口，这个称为**数据连接**。

注意：控制连接在整个会话期间一直保持打开。

0.139 03806-电子邮件格式与 MIME

1. 电子邮件格式

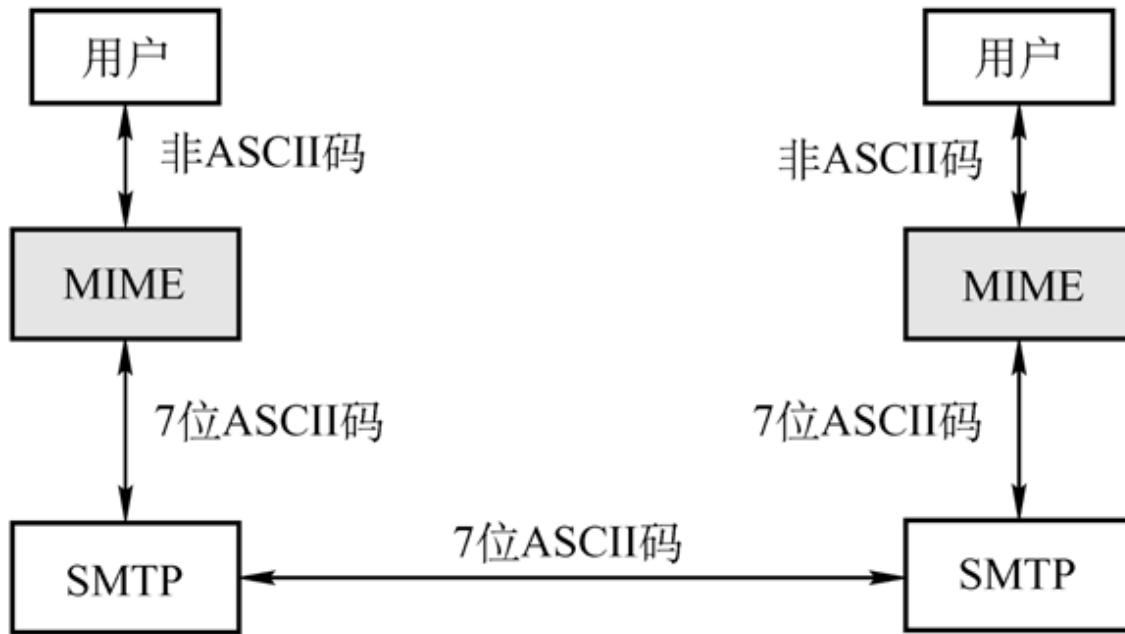
1) 电子邮件由**信封**和**内容**两部分组成。一般只规定了邮件内容中的首部格式，而邮件的主体部分由用户自由撰写。用户写好首部后，邮件系统自动将信封所需的信息提取出来并写在信封上。

2) 邮件内容首部包含一些关键字，后面加上冒号，如“**To:**”是收信人的邮件地址、“**Subject:**”是邮件的主题等。

3) 电子邮件地址的格式：收件人邮箱名 @ 邮箱所在主机的域名，符号“@”读作“at”，表示“在”的意思。例如，电子邮件地址 zhouwei@koudaitiku.com。

2. MIME

由于 SMTP 只限于传送一定长度的 7 位 ASCII 码邮件，于是提出了通用因特网邮件扩充(Multipurpose Internet Mail Extensions, MIME)。MIME 的意图是继续使用目前的 [RFC 822] 格式，但增加了邮件主体的结构，并定义了传送非 ASCII 码的编码规则。MIME 与 SMTP 的关系如下图所示。



0.140 03807-SMTP 与 POP3

1. SMTP

a. 基本介绍：**简单邮件传送协议**（SMTP）所规定的就是在两个相互通信的 SMTP 进程之间应如何交换信息。SMTP 运行在 TCP 基础之上，使用 25 号端口，也使用**客户/服务器模型**。

b. 通信过程：

1) **连接建立**。连接是在发送主机的 SMTP 客户和接收主机的 SMTP 服务器之间建立的。SMTP 不使用中间的邮件服务器。

2) **邮件传送**。

3) **连接释放**: 邮件发送完毕后, SMTP 应释放 TCP 连接。

2. POP3

邮局协议 (POP) 是一个非常简单, 但功能有限的邮件读取协议。现在使用的是它的第三个版本 POP3。POP 也使用客户/服务器的工作方式。在接收邮件的用户计算机中必须运行 POP 客户程序, 而在用户所连接的 ISP 的邮件服务器中运行 POP 服务器程序。

注意: 只要用户从 POP 服务器读取了邮件, POP 服务器就将该邮件删除。

0.141 03808-WWW 的概念和组成结构

1. WWW 的概念

WWW (World Wide Web, 万维网) 简称为 3W, 它并非某种特殊的计算机网络。万维网是一个大规模的、联机式的信息储藏所。它的特点在于用链接的方法能非常方便地从因特网上的一个站点访问另一个站点, 从而主动地按需获取丰富的信息。WWW 还提供各类搜索引擎, 使用户能够方便地查找信息。

2. WWW 的组成结构

a. WWW 把各种信息按照页面的形式组合, 一个页面包含的信息可以有文本、图形、图像、声音、动画、链接等各种格式, 这样一个页面也称为超媒体, 而页面的链接均为超链接。

b. WWW 以客户/服务器模型工作。浏览器就是客户, WWW 文档所驻留的计算机则是 WWW 服务器。

c. WWW 使用统一资源定位符 (URL) 来标志 WWW 上的各种文档。URL 的一般格式为:

< 协议 >: //< 主机 >: < 端口号 >/< 路径 >

其中常见的协议有 HTTP、FTP 等。主机部分是存储该文档的计算机, 可以是域名也可以是 IP 地址, 端口号是服务器监听的端口 (根据协议可以知道端口号, 一般省略), 路径一般也可省略, 并且在 URL 中的字符对大写或小写没有要求。

d. 完整的工作流程:

- 1) Web 用户使用浏览器 (指定 URL) 与 Web 服务器建立连接, 并发送浏览请求;
- 2) Web 服务器把 URL 转换为文件路径, 并返回信息给 Web 浏览器;
- 3) 通信完成, 关闭连接。

0.142 03809-HTTP

1. HTTP 的操作过程

超文本传送协议 (HTTP) 是在客户程序 (如浏览器) 与 WWW 服务器程序之间进行交互所使用的协议。HTTP 是面向事务的应用层协议, 它使用 TCP 连接进行可靠传输, 服务器默认监听在 80 端口。

从协议执行的过程来说, 当浏览器要访问 WWW 服务器时, 首先要完成对 WWW 服务器的域名解析。一旦获得了服务器的 IP 地址, 浏览器将通过 TCP 向服务器发送连接建立请求。每个服务器上都有一个服务进程, 它不断地监听 TCP 的端口 80, 当监听到连接请求后便与浏览器建立连接。TCP 连接建立后, 浏览器就向服务器发送要求获取某一 Web 页面的 HTTP 请求。服务器收到 HTTP 请求后, 将构建所请求的 Web 页的必需信息, 并通过 HTTP 响应返回给浏览器。浏览器再将信息进行解释, 然后将 Web 页显示给用户。最后, TCP 连接释放。

2. HTTP 的工作方式

HTTP 既可以使用非持久连接，也可以使用持久连接。

非持久连接：每一个网页元素对象的传输都需要单独建立一个 TCP 连接（“三次握手”建立）。换句话说，每请求一个万维网文档所需的时间是该文档的传输时间加上两倍往返时间 RTT（一个 RTT 用于 TCP 连接，另一个 RTT 用于请求和接收文档）。

持久连接：万维网服务器在发送响应后仍然保持这条连接，同一个客户和服务器可以继续在这条连接上传送后续的 HTTP 请求和响应报文。**持久连接又分为非流水线（2011 年已经出题）和流水线两种方式。**对于非流水线方式，客户只能在接收到前一个请求的响应后才能发送新的请求。而流水线方式是 HTTP 客户每遇到一个对象引用就立即发出一个请求，因而 HTTP 客户可以一个接一个连续地发出各个引用对象的请求。如果所有的请求和响应都是连续发送的，那么所有引用到的对象共经历一个 RTT 延迟，而不是像非流水线那样，每个引用都必须有一个 RTT 延迟。

0.143 03934-入门知识

可能疑问点：我是零基础的跨专业考生，要不要先看数字电路等基础知识？直接看计算机组成原理教材可以理解吗？

解答：想必以上问题是 95% 跨专业考生必问的。当然，两年前编者作为零基础的跨专业考生，也问过类似的问题。现在编者以一个过来者的身份很肯定地回答你：只需学习一些基础的辅助知识（考研的范围要求）。

在讲解考研知识点之前，此书先给读者介绍一些计算机组成原理的辅助知识，在以后的讲解中就不再重复了。

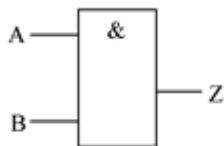
0.144 入门知识 1：了解门电路

在考研知识范围内，门电路不会考查得很复杂，考生只需了解几个基本的门电路即可。

顾名思义，“门”起到开关的作用，如某公司要招聘员工，公司对待招聘员工的要求是既要有技术，又要沟通能力好，因此只要应聘的人同时满足这两个要求就有可能被公司录用。然而，不同的公司对员工有不同的要求，如另外一家公司可能只要技术和沟通能力满足其一即可，那么又可以形成新的“门”。同理，在计算机中，如果有多个输入端，此“门”就可以对这些输入端“提出”要求，如每个输入端都是高电平，“门”才打开；或者多个输入端只要有一个是高电平，“门”就打开。以上就是门电路的基本涵义。

下面介绍常用的 6 种门电路，以下假设都只有两个输入端，实际情况则可能有多个输入端。

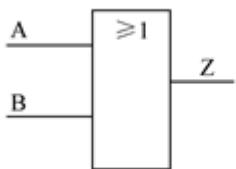
(1) 与门（有假即假）



说明：当所有输入同时为“1”电平时，输出才为“1”电平，否则输出为“0”电平，见下表。

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

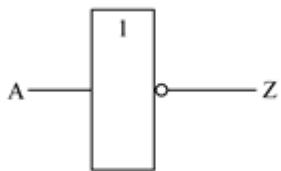
(2) 或门 (有真即真)



说明：多个输入端只要有一个输入端为“1”电平，输出就为“1”电平，只有所有输入端同时为“0”电平，输出才为“0”电平，见下表。

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

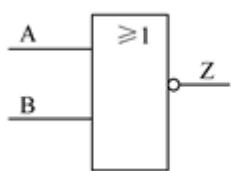
(3) 非门 (取反运算)



说明：输入“1”电平，输出“0”电平；输入“0”电平，输出“1”电平（图中小圈表示取反），见下表。

A	Z
0	1
1	0

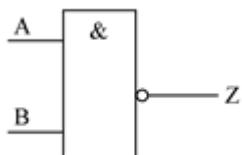
(4) 或非门



说明：和“或”门基本一样，只是将结果取反而已（图中小圈表示取反），见下表。

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

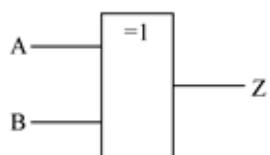
(5) 与非门



说明：和“与”门基本一样，只是将结果取反而已，见下表。

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

(6) 异或门



说明：输入电平相同时，输出“0”电平；输入电平不同时，输出“1”电平。助记：同号相乘为正（0），异号相乘为负（1）。第2章介绍乘法符号的处理时会用到“异或”门，见下表。

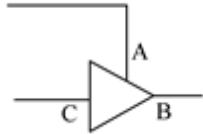
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

注意：在输入端当然也可以使用小圈，只要记住图中小圈表示取反即可。

0.145 入门知识 2：三态门

三态门：指逻辑门的输出端除有高、低电平两种状态外，还有第3种状态——高阻态（电路图参考下图）。高阻态相当于隔断状态（因为实际电路中不可能去断开它，所以设置这样一个状态，使它处于隔断

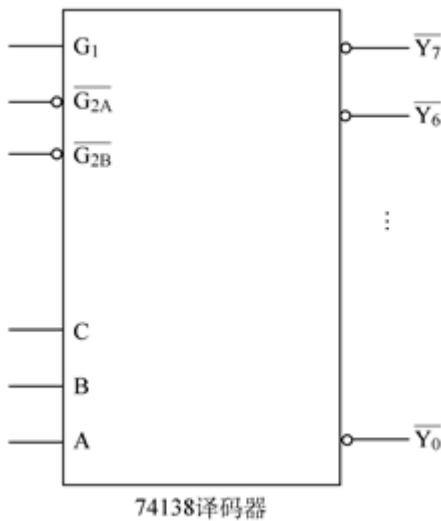
状态), 例如, 内存中的一个存储单元, 当读写控制线处于低电平时, 存储单元被打开, 可以写入数据; 当处于高电平时, 可以读出数据; 但不读不写时, 就要用高阻态, 就像把该存储单元隔离开来一样。更直白的解释是, 高阻态就是一个开关, 当处于高阻态时, 逻辑门什么也不能做。



说明: 当 A 为高电平时, C→B 导通; 当 A 为低电平时, C→B 不导通, 此时为高阻态。

0.146 入门知识 3: 片选译码器

该知识点主要介绍最常用的 3-8 译码器 (或称 74138 译码器, 属于存储器与 CPU 连接中的片选译码器), 其他的译码器 (如 2-4 译码器、4-16 译码器) 原理都相似。常用的 3-8 译码器如下图所示:



先记住一句话, 只要“头上有杠”的信号, 不管是输入端还是输出端都应该加小圈, 表示低电平有效。由于 \bar{Y}_i 的“头上有杠”, 因此输出端必须要加小圈 (若某个 \bar{Y}_i 被选中, 则输出低电平, 即 0), 遇到门电路时再用小圈恢复。但是问题又来了, 有些考生说这个岂不是很麻烦, 直接用高电平有效不就得了? 答案是, 一般都使用低电平有效, 而不使用高电平有效, 这个问题在第 6 章中将会详细介绍。

G_1 端、 $\overline{G_{2A}}$ 端、 $\overline{G_{2B}}$ 端分别表示高电平有效、低电平有效、低电平有效。只有当 G_1 端、 $\overline{G_{2A}}$ 端、 $\overline{G_{2B}}$ 端分别为高电平、低电平、低电平时, 才能使译码器正常工作。其实只需要一个访存控制信号 $MREQ$ 即可, 但是相应教材上一定要同时使用 G_1 端、 $\overline{G_{2A}}$ 端、 $\overline{G_{2B}}$ 端和访存控制信号 \overline{MREQ} , 那只能按照教材上的讲解, 具体连接可参考第 3 章的习题。

可能疑问点: 根据近两年的答疑情况, 很多同学误解了一点。上面讲到只有当 G_1 端、 $\overline{G_{2A}}$ 端、 $\overline{G_{2B}}$ 端分别为高电平、低电平、低电平时, 才能使译码器正常工作, 就是这句话被不少同学误解了, 因为很多同学认为 $\overline{G_{2A}}$ 端是表示低电平有效, 而 $\overline{G_{2A}}$ 端前面又有一个小圈取反, 应该输入高电平才能使其正常工作。其实并不是这样, 不能将 $\overline{G_{2A}}$ 上面的“杠”与小圈进行中和, 不能负负得正。总之记住一句话, 译码器左上的 G_1 、 $\overline{G_{2A}}$ 、 $\overline{G_{2B}}$ 输入端, 没有小圈就输入高电平 1, 有小圈就输入低电平 0。正因为这样, $\overline{G_{2A}}$ 、 $\overline{G_{2B}}$ 经常与低电平的 $MREQ$ 相连, 而 G_1 端经常被接入 +5V 的高电平信号 (可学习完第 3 章后再回来理解)。

接下来就是输入端 C、B、A 和输出端 \bar{Y}_0 、 \bar{Y}_1 、 \dots 、 \bar{Y}_7 之间的对应关系, 见下表:

C	B	A	含 义
0	0	0	$\overline{Y_0}$ 端有效, $\overline{Y_0}$ 对应的存储芯片组被选中
0	0	1	$\overline{Y_1}$ 端有效, $\overline{Y_1}$ 对应的存储芯片组被选中
0	1	0	$\overline{Y_2}$ 端有效, $\overline{Y_2}$ 对应的存储芯片组被选中
0	1	1	$\overline{Y_3}$ 端有效, $\overline{Y_3}$ 对应的存储芯片组被选中
1	0	0	$\overline{Y_4}$ 端有效, $\overline{Y_4}$ 对应的存储芯片组被选中
1	0	1	$\overline{Y_5}$ 端有效, $\overline{Y_5}$ 对应的存储芯片组被选中
1	1	0	$\overline{Y_6}$ 端有效, $\overline{Y_6}$ 对应的存储芯片组被选中
1	1	1	$\overline{Y_7}$ 端有效, $\overline{Y_7}$ 对应的存储芯片组被选中

入门知识 4：那些可怕的专业术语

(1) 系列机

系列机是指一个厂家生产的具有相同系统结构、不同组成和实现的一系列不同型号的机器。它应在指令系统、数据格式、字符编码、中断系统、控制方式、输入/输出控制方式等方面保持统一，从而保证软件的兼容性。

(2) 阿姆代尔定律 (Amdahl's Law)

阿姆代尔定律是指系统优化某部件所获得的系统性能的改善程度，取决于该部件被使用的频率，或占用总执行时间的比例。该定律很好地刻画了改善“系统瓶颈”性能的重要性。

(3) 基准测试程序

基准测试程序是专门用来进行性能评价的一组程序，这些程序能够很好地反映机器在运行实际负载时的性能。可以在不同机器上运行相同的基准测试程序来比较不同机器的运行时间，从而比较其性能。

(4) 最低有效位、最高有效位、最低有效字节、最高有效字节

最低有效位：一个二进制数中的最低位，例如二进制数 1110 中的“0”。

最高有效位：一个二进制数中的最高位，例如二进制数 0111 中的“0”。

最低有效字节：一个二进制数中的最低字节，例如二进制数 1111 1111 0000 0000 1111 0000 中的“1111 0000”。

最高有效字节：一个二进制数中的最高字节，例如二进制数 1111 1111 0000 0000 1111 0000 中的“1111 1111”。

(5) 基数

若进位计数制的“基数”为 R，第 n 位数的权即 R^{n-1} ，则只要将各位数字与它的权相乘，并将其积累加，和数就是十进制数，例如，二进制数的基数是“2”，十进制数的基数为“10”，十六进制的基数为“16”。

(6) 逻辑数据

逻辑数据用来表示命题的“真”和“假”，分别用“1”和“0”来表示。进行逻辑运算时，应按位进行。

0.147 入门知识 5：与存储相关的那些名词

有关存储的概念类包含存储元、存储单元、存储体、存储字和存储字长等，见下表。

情景助记：假设将医院的整个住院部看成一个存储体，住院部由一个个病房组成（假设每个病房的床位数量都相等），那么每个病房就是一个存储单元，病房中的每张床就是一个存储元，每个病房的床位数量就是存储字长。

现假设病床上躺的都是 0 和 1，而每间病房肯定都对应一串二进制代码。如果某病房二进制代码为 10001000，

则这串二进制代码可表示为十进制数 136，也可以表示为十六进制数 88H，当然也直接可以代表 8 位的二进制数。综上所述，可以将每个存储单元中二进制代码的组合看。

存储xx	说 明
存储元	也可称为存储元件和存储基元，用来存放一位二进制信息
存储单元	由若干个存储元组成，能存放多位二进制信息
存储体	许多存储单元可组成存储体（存储矩阵）
存储字	每个存储单元中二进制代码的组合即为存储字，可代表数值、指令和地址等
存储字长	每个存储单元中二进制代码的位数就是存储字长

0.148 入门知识 6：与字、字长相关的那些名词

0.149 $1B=8bit$ ，这个是规定，没有错误。但是很多考生认为一个字等于两个字节，因为他们脑海中有一种概念：一个汉字占用两个字节。计算机中的字和汉字中的字的概念不一样。计算机中的字通常由一个或多个（一般是字节的整数倍）字节构成。现在常用的都是 32 位（4 个字节）字长的机器。

0.150 以前是存储字长等于机器字长，因为机器字长是机器一次能处理的比特数，这样一次取一个等长的存储字便于机器处理。现在机器字长一般大于存储字长。

0.151 一般默认字长 = 机器字长 = 存储字长。

0.152 入门知识 7：与周期相关的那些名词

有关周期的概念主要有以下几种，见下表

名 称	说 明
指令周期	从一条指令的启动到下一条指令启动所经历的时间，通常由多个机器周期组成
时钟周期 (节拍周期)	计算机主频周期，通常将一个时钟周期定义为一个节拍
总线周期	CPU 通过总线对存储器或 I/O 端口进行一次访问所需的时间
机器周期 (CPU 周期)	在计算机中，为了便于管理，常把一条指令的执行过程划分为若干个阶段，每一个阶段完成一项工作，例如：取指令、存储器读、存储器写等，每一项工作称为一个基本操作。完成一个基本操作所需的时间称为机器周期。一般情况下，一个机器周期由若干个时钟周期构成
微指令周期	读出微指令的时间加上执行该条微指令的时间 注意：微指令周期常取成和机器周期相等
存取周期 (存储周期)	需要和存取时间区分开。存取时间又称为存储器的访问时间，指启动一次存储器操作（读或写）到完成该操作所需的时间。存取时间分为读出时间和写入时间两种 存取周期指存储器进行连续两次独立的存储器操作（要么连续两次读操作，要么连续两次写操作）所需的最小间隔时间，通常存取周期大于存取时间

0.153 03937-计算机硬件的基本组成

计算机硬件主要由存储器、运算器、控制器、输入设备和输出设备组成（输入/输出设备统称 I/O 设备），它们之间的关系如图 1-1 所示。

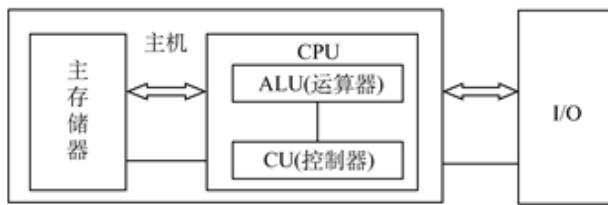


图 1-1 计算机硬件的基本组成

其中，运算器 + 控制器 = CPU，CPU+ 主存储器 = 主机，I/O 设备又称为外部设备。

主存储器（主存）：存放程序和数据的部件。它是计算机实现“存储程序控制”的基础。

运算器：对信息进行处理和运算的部件，经常进行算术和逻辑运算，其核心是算术逻辑单元（ALU）。

控制器：是整个计算机的“指挥中心”，它使计算机各个部件自动协调工作。计算机中有两种信息在流动：一种是控制信息；另一种是数据信息。

输入设备：将人们所熟悉的信息形式转换成计算机可以接收并识别的信息形式的设备，例如键盘，当按下一个键时，此键被翻译成 ASCII 码传输给计算机，而 ASCII 码就是计算机可以接收并识别的信息形式。

输出设备：可将二进制信息转换成人类或其他设备可以接收或识别的信息，如显示器。

0.154 03940-计算机系统的层次结构

就像计算机网络中分层的概念一样，对于某一层次的观察者来说，观察者只需关注此层的一些概念，不用理会下层是如何工作和实现的。同理，现代计算机也不是一种简单的电子设备，而是由硬件与软件结合而成的复杂整体。它通常由 5 个不同的层次组成，在每一层上都能够进行程序设计，如图 1-7 所示。

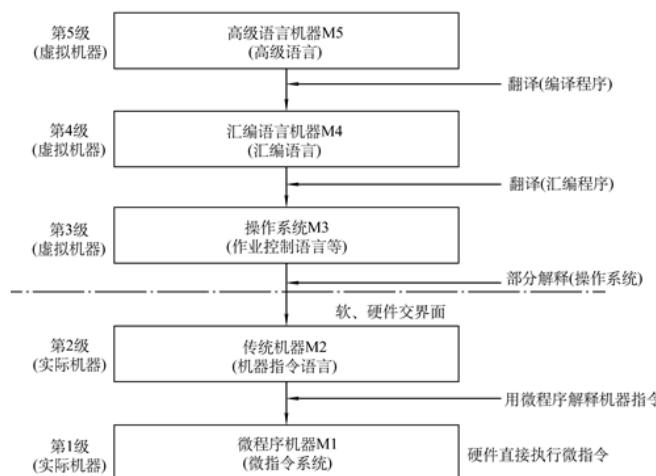


图 1-7 计算机系统的多级层次结构

补充知识点：编译程序、解释程序、汇编程序的区别。

解析：汇编程序是用汇编语言编写的程序，与编译程序、解释程序完全不是一个概念。

总结：

1) 解释程序是高级语言翻译程序的一种，它将源语言书写的源程序作为输入，解释一句就提交给计算机执行一句，并不形成目标程序。

2) 编译程序把高级语言源程序作为输入，进行翻译转换，产生出机器语言的目标程序，然后让计算机去执行这个目标程序，得到计算结果。

0.155 03942-操作系统的概念

1. 用户观点

个人计算机（PC）的操作系统要达到的目的就是方便用户使用，资源利用率显得不是很重要。大型机或者其终端等的操作系统的设计目的就是使资源利用最大化，确保所有资源都能够被充分使用，并且保障稳定性。

2. 系统观点（资源管理的观点）

从计算机的角度来看，操作系统是计算机系统的资源管理程序。

3. 进程观点

这种观点把操作系统看做是由若干个可以独立运行的程序和一个对这些程序进行协调的核心所组成的。操作系统的核心则是控制和协调这些进程的运行，解决进程之间的通信。

4. 虚拟机观点

虚拟机的观点也称为机器扩充的观点。从这一观点来看，操作系统为用户使用计算机提供了许多服务功能和良好的工作环境。用户不再直接使用硬件机器（称为裸机），而是通过操作系统来控制和使用计算机。计算机从而被扩充为功能更强、使用更加方便的虚拟计算机。

0.156 03943-计算机性能指标

考研主要涉及以下计算机性能指标：

(1) 吞吐量

吞吐量指信息流入、处理和流出系统的速率。它取决于 CPU 能够多快地取指令，数据能够多快地从内存取出或存入，以及所得结果能够多快地从内存送到输出设备。这些决定因素中的任一步骤都与主存紧密相关，因此吞吐量主要取决于主存的存取周期。

(2) 响应时间

响应时间指从提交作业到该作业得到 CPU 响应所经历的时间。响应时间越短，吞吐量越大。

(3) 主频

主频是机器内部主时钟的频率，是衡量机器速度的重要参数，其常用单位为 Hz、MHz 等。如果主频为 8MHz，则可以计算出时钟周期为：

即每秒有 8M 个时钟周期。

(4) CPU 周期

CPU 周期又称为机器周期，通常用从内存读取一条指令字的最短时间来定义。一个指令周期常由若干个 CPU 周期构成。

(5) CPU 时钟周期

主频的倒数，是 CPU 中最小的时间单位。

(6) CPI、MIPS 和 FLOPS (三者为衡量运算速度的指标)

CPI：执行一条指令所需要的时钟周期数。

MIPS：每秒可执行百万条指令数，如某机器每秒可以执行 800 万条指令，则记作 8MIPS。

FLOPS：每秒执行的浮点运算次数。

MFLOPS：每秒百万次浮点运算，与 MIPS 类似。

GFLOPS：每秒十亿次浮点运算。

TFLOPS：每秒万亿次浮点运算。

PFLOPS：每秒千万亿次浮点运算。

补充：IPC：CPU 的每一个时钟周期内所执行的指令数。

(7) CPU 执行时间

CPU 执行时间指 CPU 对某特定程序的执行时间，例如，对于程序 A 和程序 B，CPU 执行程序 A 和程序 B 分别使用了 2s 和 4s，则对于程序 A 和程序 B 而言，CPU 执行时间分别是 2s 和 4s。

0.157 03945-操作系统的主要功能和提供的服务

一、操作系统的主要功能

1. 处理器管理

进程控制：负责进程的创建、撤销及状态转换。

进程同步：对并发执行的进程进行协调。

进程通信：负责完成进程间的信息交换。

进程调度：按一定算法进行处理器分配。

2. 存储器管理

内存分配：按一定的策略为每道程序分配内存。

内存保护：保证各程序在自己的内存区域内运行而不相互干扰。

内存扩充：为允许大型作业或多作业的运行，必须借助虚拟存储技术去获得增加内存的效果。

3. 设备管理

设备分配：根据一定的设备分配原则对设备进行分配。

设备传输控制：实现物理的输入输出操作，即启动设备、中断处理、结束处理等。

设备独立性：即用户程序中的设备与实际使用的物理设备无关。

4. 文件管理

文件存储空间的管理：负责对文件存储空间进行管理，包括存储空间的分配与回收等功能。

目录管理：目录是为方便文件管理而设置的数据结构，它能提供按名存取的功能。

文件操作管理：实现文件的操作，负责完成数据的读写。

文件保护：提供文件保护功能，防止文件遭到破坏。

5. 用户接口

命令接口：提供一组命令供用户直接或间接控制自己的作业。

程序接口：也称为系统调用，是程序级的接口，由系统提供一组系统调用命令供用户程序和其他系统程序调用。

图形接口：近年来出现的图形接口（也称图形界面）是命令接口的图形化。

二、操作系统提供的服务

由操作系统的功能可以知道操作系统提供哪些服务：操作系统提供了一个用以执行程序的环境，提供的服务有程序执行、I/O 操作、文件操作、资源分配与保护、错误检测与排除等。

0.158 03946-进位计数制及其相互转换

1. 二进制、八进制、十进制、十六进制的基本概念

十进制：日常生活中的进位计数制都是十进制。

二进制：二进制是计算技术中使用最广泛的一种数制，使用 0 和 1 两个数码来表示。

八进制和十六进制：规则和二进制相似。

2. 二进制、八进制、十进制、十六进制的相互转换

二进制、八进制、十六进制转换为十进制：仅以二进制转换为十进制为例，八进制和十六进制转换为十进制的方法是一样的，只需调整每一位的权重即可。

【例 2-1】将 $(10001.10)_2$ 转换为十进制。

解：每位的位权见表 2-1。

表 2-1 每位的位权

位权	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}
数位	1	0	0	0	1	1	0

每位的位权乘以各自的数位，得到的和相加：

$$2^4 \times 1 + 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 0 + 2^0 \times 1 + 2^{-1} \times 1 + 2^{-2} \times 0 = 17.5$$

至此，得到的 17.5 即为该二进制数对应的十进制数（八进制和十六进制，只需把位权中的基数 2 换为 8 和 16，其余步骤一样）。

十进制转换为二进制、八进制、十六进制：这里仍然以十进制转换为二进制为例。

【例 2-2】将 $(19.6875)_{10}$ 转换为二进制数。

解：在这个转换中，整数部分和小数部分应当分开来计算。整数部分的计算规则可以归纳为“除 2 取余，由下而上”：

$$\begin{array}{ll} 19/2=9 & \text{余 } 1 \\ 9/2=4 & \text{余 } 1 \\ 4/2=2 & \text{余 } 0 \\ 2/2=1 & \text{余 } 0 \\ 1/2=0 & \text{余 } 1 \end{array}$$

（结束的标志为相除之后的商为 0）

然后，将所有余数按照从下往上的顺序写出：10011，此即十进制数 19 转换为二进制数的表示。

小数部分的计算规则可以归纳为“乘 2 取整，由上而下”：

$$\begin{array}{ll} 0.6875 \times 2 = 1.375 & \text{取 } 1 \text{ 余 } 0.375 \\ 0.375 \times 2 = 0.75 & \text{取 } 0 \text{ 余 } 0.75 \\ 0.75 \times 2 = 1.5 & \text{取 } 1 \text{ 余 } 0.5 \\ 0.5 \times 2 = 1 & \text{取 } 1 \text{ 余 } 0 \end{array}$$

（结束标志为取 1 后余数为 0）

然后，将所取的数按照从上往下的顺序写出（整数部分是从下往上，不要混淆）：0.1011，此即十进制数 0.6875 转换为二进制数的表示。

综上所述，转换成二进制数为 $(10011.1011)_2$ 。

0.159 03947-真值和机器数

日常生活中我们经常看到 +5、-8、-0.1、+3.6 等这些带有“+”或者“-”符号的数，称为真值。那么如果要用计算机处理这些数，计算机不认识“+”或者“-”符号怎么办？此时需要具备的思维能力：一想到具有两种状态的事物，都应该联想到二进制的 0 和 1。

恰好“+”“-”是两种状态，于是就可以使用二进制的 0 和 1 来表示。那就再做一个规定：0 表示正号，1 表示负号。这样就可以将一个真值完全数字化了，而被数字化的数就称为机器数（机器数分为原码、补码、反码和移码）。

0.160 03949-字符和字符串

在计算机中，最常用的字符编码是 ASCII 码。基本的 ASCII 码字符集共有 128 个字符，其中有 96 个可打印字符，包括常用的字母、数字、标点符号等，另外还有 32 个控制字符。考生有必要记住一些基本的编码，如常用数字的 ASCII 码，以及大小写字母的 ASCII 码。其实，字母和数字的 ASCII 码的记忆非常简单，只要记住一个字母或数字的 ASCII 码（如记住 A 为 65，0 的 ASCII 码为 48），知道相应的大、小写字母之间差 32，就可以推算出其余字母或数字的 ASCII 码。

虽然标准 ASCII 码是 7 位编码，但由于计算机基本处理单位为字（1B=8bit），因此一般仍以一个字节来存放一个 ASCII 字符。每一个字节中多余出来的一位（最高位）在计算机内部通常保持为 0（在数据传输时可用做奇偶校验位）。

除 ASCII 码以外，还有一些其他的常用编码：扩展 ASCII 码（对 ASCII 码的扩充，字节的最高位保持为 1）、EBCDIC 编码（一些大型主机系统如 MVS、OS/390 等使用的编码）、GB2312 编码（GB 代表“国标”，

中国的标准化组织设计的简体汉字编码)、Unicode 编码(“通用”编码,是统一了很多编码的一个大整合)等。

在了解字符编码的基础上,就不难理解字符串编码了。简单来说,字符串就是字符的“集合”,在计算机的存储中,通常在存储器中占用连续的多个字节空间,每个字节存储一个字符(若是汉字字符串,则是两个字节存储一个汉字)。有一个情况需要知道,当主存字由 2 个或 4 个字节组成时,在同一个主存字中,既可按从低位字节向高位字节的顺序存放字符串的内容,也可按从高位字节向低位字节的顺序存放字符串的内容,这个取决于使用的机器(在第 4 章将会详细讲解高低字节的区别,即大小端)。

注意: 2012 年的考研真题就针对小端方式进行了考查。

0.161 03950-操作系统的形成与发展

1. 手工操作阶段

使用过程大致如下:先将程序纸带(或卡片)装入输入机,然后启动输入机把程序和数据送入计算机,接着通过控制台开关启动程序运行,当程序运行完毕后,由用户取走纸带和结果。

由此可以推断出,这种操作方式具有用户独占计算机资源、资源利用率低以及 CPU 等待等人工操作的特点。

2. 脱机输入输出技术

脱机输入输出技术是为了解决 CPU 和 I/O 设备之间速度不匹配的矛盾而提出的,此技术减少了 CPU 的空闲等待时间,提高了 I/O 速度。

采用脱机输入输出技术后,低速 I/O 设备上数据的输入输出都在外围机的控制下进行,而 CPU 只与高速的输入带及输出带打交道,从而有效地减少了 CPU 等待慢速设备输入输出的时间。

3. 批处理技术

批处理技术是指计算机系统对一批作业自动进行处理的一种技术。计算机系统对磁带上的作业自动地一个接一个进行处理,直至把磁带上的所有作业全部处理完毕,这样便形成了早期的批处理系统。

4. 多道程序设计技术

在早期批处理系统中,每次只将一个用户程序调入内存运行,这种作业运行方式称为单道运行。缺点是每当程序发出 I/O 请求时,CPU 便处于等待 I/O 完成的状态,致使 CPU 空闲。为进一步提高 CPU 的利用率,引入了多道程序设计技术。

多道程序设计技术是“将一个以上的作业存放在主存中,并且同时处于运行状态。这些作业共享处理器、外设以及其他资源”。现代计算机系统一般都基于多道程序设计技术。

在单处理器系统中,多道程序运行的特点:多道、宏观上并行、微观上串行。

5. 操作系统的形成

操作系统是一组控制和管理计算机硬件和软件资源,合理地组织计算机工作流程,以及方便用户的程序的集合。

0.162 03952-校验码

检错编码: 就是通过一定的编码和解码,能够在接收端解码时检查出传输的错误,但不能纠正错误。常见的检错编码有奇偶校验码和循环冗余校验(CRC)码。

1. 奇偶校验码

奇偶校验码就是在信息码后面加一位校验码,分为奇校验和偶校验。

奇校验: 添加一位校验码后,使得整个码字里面 1 的个数是奇数;接收端收到数据后就校验数据里 1 的个数,如果正好为奇数,则认为传输没有出错;如果检测到偶数个 1,则说明传输过程中,数据发生了

改变，要求重发。

偶校验：添加一位校验码后，使得整个码字里面 1 的个数是偶数；接收端收到数据后就校验数据里 1 的个数，如果正好为偶数，则认为传输没有出错；如果检测到奇数个 1，则说明传输过程中，数据发生了改变，要求重发。

2. 循环冗余校验（CRC）码

奇偶校验码的检错率较低，不太实用。目前，在计算机网络和数据通信中，用得最广泛的就是检错率高、开销小、易实现的循环冗余校验码。循环冗余校验码的原理比较简单，相关教材讲解得很细致，这里不再赘述。以下仅给出考生在求解循环冗余校验码过程中遇到的一个最常见问题的解答，即循环冗余校验码中的二进制除法。

试计算 $10110010000/11001$ 。

解析：解题技巧包括以下 3 个方面。

- 1) $0 \pm 1 = 1$, $0 \pm 0 = 0$, $1 \pm 0 = 1$, $1 \pm 1 = 0$ (可以简化为做“异或”运算，在除法过程中，计算部分余数，全部使用“异或”操作，相同则为 0，不同则为 1)。
- 2) 上商的规则是看部分余数的首位，如果为 1，商上 1；如果为 0，商上 0。
- 3) 当部分余数的位数小于除数的位数时，该余数即为最后余数。

$$\begin{array}{r} 1101010 \\ 11001 \sqrt{10110010000} \\ \underline{11001} \\ \underline{\underline{11110}} \\ \underline{11001} \\ \underline{\underline{11110}} \\ \underline{11001} \\ \underline{\underline{11100}} \\ \underline{11001} \\ \underline{\underline{1010}} \end{array}$$

纠错编码：就是在接收端不但能检查出错误，而且能将检查出来的错误纠正。常见的纠错编码为海明码。

海明码又称为汉明码，是在信息字段中插入若干位数据，用于监督码字里的哪一位数据发生了变化，且有一位纠错能力。假设信息位有 k 位，整个码字的长度就是 $k+r$ ；每一位的数据只有两种状态，不是 0 就是 1，有 r 位数据就应该表示出 2^r 种状态。若每一种状态代表一个码元发生了错误，则有 $k+r$ 位码元，就要有 2^{k+r} 种状态来表示，另外还要有一种状态来表示数据正确的情况， $2^{k+r} \geq k+r+1$ 才能检查一位错误，即 $2^r \geq k+r+1$ ，例如，信息数据有 4 位，由 $2^r \geq k+r+1$ 得 $r \geq 3$ ，也就是至少需要 3 位监督数据才能发现并改正 1 位错误。再如，给 8 个学员进行编号，可以用 3 位数来编码：学号为 000, 001, …, 111；也可以用 5 位数来编码：学号为 00000, 00001, 00010, …, 00111，但是没有必要用 5 位，只要能满足编码的要求就可以了，因此只需求出满足条件的最小 k 值即可。

0.163 03953-内核态与用户态

为了避免操作系统及其关键数据（如 PCB 等）受到用户程序有意或无意的破坏，通常将处理器的执行状态分为两种：核心态与用户态。

核心态：又称管态、系统态，是操作系统管理程序执行时机器所处的状态。它具有较高的特权，能执行包括特权指令的一切指令，能访问所有寄存器和存储区。

用户态：又称目态，是用户程序执行时机器所处的状态，是具有较低特权的执行状态，它只能执行规定的指令，访问指定的寄存器和存储区。

划分核心态与用户态之后，用户态程序不能直接调用核心态程序，而是通过执行访问核心态的命令，引起中断，由中断系统转入操作系统内的相应程序，例如，在系统调用时，将由用户态转换到核心态。

特权指令：只能由操作系统内核部分使用，不允许用户直接使用的指令，如 I/O 指令、设置中断屏蔽指令、清内存指令、存储保护指令、设置时钟指令。

0.164 03955-定点数的表示

符号位的处理：一般来讲，符号位的处理有以下两种方式：

1. 无符号数的表示

无符号数指整个机器字长的全部二进制位均为数值位，没有符号位，相当于数的绝对值。若机器字长为 8 位，则数的表示范围为 $0 \sim 2^8 - 1$ ，即 $0 \sim 255$ 。

2. 有符号数的表示

有符号数需要将其符号数字化，即“0”表示正号，“1”表示负号。下面介绍 3 种有符号数的表示方法：原码、补码、反码。

友情提示：不少考生觉得运算器比较复杂难懂，很大程度上是因为分段函数，例如：

$$[x]_{\text{sgn}} = \begin{cases} x & 1 > x \geq 0 \\ 2+x & 0 > x \geq -1 \end{cases}$$

计算真值的原码、补码、反码、移码更简单的方法，根本不用管这个分段函数，就是用下面 3 句话解决问题，简单易懂。

第 1 句：3 种机器数的最高位均为符号位。符号位和数值部分之间可用“.”（对于小数）或“，”（对于整数）隔开。

第 2 句：当真值为正数时，原码、补码和反码的表示形式均相同，即符号位用“0”表示，数值部分与真值相同。

第 3 句：当真值为负数时，原码、补码和反码的表示形式不同，但其符号位都用“1”表示，而数值部分有这样的关系，即补码是原码的“每位求反加 1”，反码是原码的“每位求反”。需要注意的是，上面所谓的“每位求反”均不包括符号位，只是对数值部分进行求反，且原码除了符号位为“1”，数值部分仍然与真值相同。

0.165 03956-中断与异常

中断与异常是一对类似但又有区别的概念。

中断是系统正常功能的一部分，例如，因进程调度使系统停止当前运行的进程转而执行其他进程，或者因缺少所需资源而中断当前操作等待资源到达等，在系统处理完其他事情之后，会继续执行中断前的进程。

异常是由错误引起的，例如文件损坏、进程越界等。

注意：通常异常会引起中断，而中断未必是由异常引起的。

0.166 03958-模块组合结构

操作系统是一个有多种功能的系统程序，可以看成是一个整体模块，也可以看成是若干个模块按一定的结构方式组成的。系统中的每一个模块都是根据它们要完成的功能来划分的，这些功能模块按照一定的结构方式组合起来，协同完成整个系统的功能。

优点：结构紧密、接口简单直接、系统的效率相对较高。

缺点：系统结构不清晰、可扩展性较差、可适应性差。

综上，模块组合结构只适用于系统小、模块少、使用环境比较稳定的系统。

0.167 03962-进程的定义及描述

1. 进程的定义

- a. 进程是程序在处理器上的一次执行过程；
- b. 进程是可以和别的进程并行执行的计算；
- c. 进程是程序在一个数据集合上的运行过程，是系统进行资源分配和调度的一个独立单位；
- d. 进程可定义为一个数据结构及能在其上进行操作的一个程序；
- e. 进程是一个程序关于某个数据集合在处理器上顺序执行所发生的活动。

2. 进程的特征

进程具有以下几个基本特征：**动态性、并发性、独立性、异步性。**

进程的结构特征：为了描述和记录进程的运动变化过程，并使之能正确运行，应为每个进程配置一个进程控制块（Process Control Block, PCB）。这样从结构上看，每个进程都由**程序段、数据段**和一个**进程控制块**组成。

3. 进程和程序的关系

- a. 进程是动态的，程序是静止的；
- b. 进程是暂时的，程序是永久的；
- c. 进程与程序的组成不同：进程的组成包括程序、数据和进程控制块；
- d. 通过多次执行，一个程序可以产生多个不同的进程；通过调用关系，一个进程可以执行多个程序。进程可创建其他进程，而程序不能形成新的程序；
- e. 进程具有并行特性（独立性、异步性），程序则没有。

4. 进程和作业的区别

- a. 作业是用户向计算机提交任务的任务实体。而进程则是完成用户任务的执行实体，是向系统申请分配资源的基本单位；
- b. 一个作业可由多个进程组成，且必须至少由一个进程组成，但一个进程不能构成多个作业。
- c. 作业的概念主要用在批处理系统中。像 UNIX 这样的分时系统则没有作业的概念；而进程的概念则用在几乎所有的多道程序系统中。

5. 进程的组成

进程由进程控制块（PCB）、程序段、数据段、进程标识符（PID）、进程当前状态、进程队列指针、程序和数据地址、进程优先级、CPU 现场保护区、通信信息、家族联系、占有资源清单组成。

0.168 03963-进程的状态与转换

1. 进程的 3 种基本状态

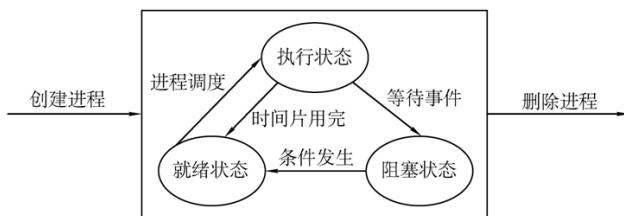
就绪状态：进程已获得了除处理器以外的所有资源，一旦获得处理器，就可以立即执行，此时进程所处的状态为就绪状态。

执行状态（运行状态）：当一个进程获得必要的资源并正在 CPU 上执行时，该进程所处的状态为执行状态。

阻塞状态（等待状态）：正在执行的进程，由于发生某事件而暂时无法执行下去（如等待 I/O 完成），此时进程所处的状态为阻塞状态。当进程处于阻塞状态时，即使把处理器分配给该进程，它也无法运行。

2. 进程状态的相互转换

下图给出了进程的 3 种基本状态以及引起状态转换的典型原因。



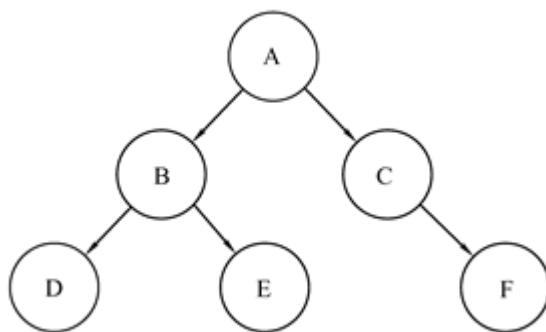
- a. 就绪状态 → 执行状态：一个进程被进程调度程序选中。
- b. 执行状态 → 阻塞状态：请求并等待某个事件发生。
- c. 执行状态 → 就绪状态：时间片用完或在抢占式调度中有更高优先级的进程变为就绪状态。
- d. 阻塞状态 → 就绪状态：进程因为等待的某个条件发生而被唤醒。

0.169 03964-进程的控制

1. 进程的创建

(1) 进程前趋图

一个进程可以创建若干个新进程，新创建的进程又可以创建子进程，为了描述进程之间的创建关系，引入了下图所示的进程前趋图。



(2) 创建原语

在多道程序环境中，只有进程才可以在系统中运行。为了使一个程序能运行，必须为它创建进程。导致进程创建的事件有：用户登录、作业调度、请求服务。

进程创建是通过创建原语实现的。其主要操作过程如下：

- a. 先向系统申请一个空闲 PCB，并指定唯一的进程标识号 PID。
- b. 为新进程分配必要的资源。
- c. 将新进程的 PCB 初始化。为新进程的 PCB 填入进程名、家族信息、程序数据地址、优先级等信息。
- d. 将新进程的 PCB 插入就绪队列。

2. 进程的撤销

导致进程撤销的事件有：进程正常结束、进程异常结束及外界干预等。

撤销原语的功能是撤销一个进程，其主要操作过程如下：

- a. 先从 PCB 集合中找到被撤销进程的 PCB。
- b. 若被撤销进程正处于执行状态，则应立即停止该进程的执行，设置重新调度标志，以便进程撤销后将处理器分配给其他进程。
- c. 对后一种撤销策略，若被撤销进程有子孙进程，还应将该进程的子孙进程予以撤销。
- d. 回收被撤销进程所占有的资源，或者归还给父进程，或者归还给系统。最后，回收它的 PCB。

3. 进程的阻塞与唤醒

阻塞原语的功能是将进程由执行状态转为阻塞状态，而唤醒原语的功能则是将进程由阻塞状态变为就绪状

态。当一个进程期待的某一事件尚未出现时，该进程调用阻塞原语将自己阻塞起来。该进程自身调用原语阻塞自己的，是一种主动行为。

阻塞原语的主要操作过程如下：

- a. 首先停止当前进程的运行。由于该进程正处于执行状态，故应中断处理器。
- b. 保存该进程的 CPU 现场以便之后可以重新调用该进程并从中断点开始执行。
- c. 停止运行该进程，将进程状态由执行状态改为阻塞状态，然后将该进程插入到相应事件的等待队列中。
- d. 转到进程调度程序，从就绪队列中选择一个新的进程投入运行。

对处于阻塞状态的进程，当该进程期待的事件出现时，由发现者进程调用唤醒原语将阻塞的进程唤醒，使其进入就绪状态。

注意：一个进程由执行状态变为阻塞状态，是由这个进程自己调用阻塞原语去完成的；而进程由阻塞状态转变为就绪状态，则是由另一个发现者进程调用唤醒原语实现的，一般这个发现者进程与被唤醒进程是合作的并发进程。

0.170 03965-线程

1. 线程的引入

进程的两个基本属性：

- a. 进程是一个拥有资源的独立单元；
- b. 进程同时又是一个可以被处理器独立调度和分配的单元。

为了使多个程序更好地并发执行，并尽量减少操作系统的开销，操作系统设计者引入了线程，让线程去完成第二个基本属性的任务，而进程只完成第一个基本属性的任务。

2. 线程的定义

线程是进程内一个相对独立的、可调度的执行单元。线程自己基本上不拥有资源，只拥有一点在运行时必不可少的资源（如程序计数器、一组寄存器和栈），但它可以与同属一个进程的其他线程共享进程拥有的全部资源。多线程是指一个进程中多个线程，这些线程共享该进程资源。**但是各线程自己堆栈数据不对其他线程共享。**

3. 线程的实现

- a. **内核级线程：**指依赖于内核，由操作系统内核完成创建和撤销工作的线程。
- b. **用户级线程：**指不依赖于操作系统核心，由应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制的线程。

4. 线程与进程的比较

a. **调度：**在传统的操作系统中，拥有资源和独立调度的基本单位都是进程。而在引入线程的操作系统中，线程是独立调度的基本单位，进程是拥有资源的基本单位。在同一进程中，线程的切换不会引起进程切换。在不同进程中进行线程切换，如从一个进程内的线程切换到另一个进程中的线程中，将会引起进程切换。

b. **拥有资源：**进程是拥有资源的基本单位，而线程不拥有系统资源（也有一点必不可少的资源，并非什么资源都没有），但线程可以访问其隶属进程的系统资源。

c. **并发性：**在引入线程的操作系统中，不仅进程之间可以并发执行，而且同一进程内的多个线程之间也可以并发执行。

d. **系统开销：**由于创建进程或撤销进程时，系统都要为之分配或回收资源，系统开销较大；而线程切换时，只需保存和设置少量寄存器内容，因此开销很小。

5. 多线程模型

- a. **多对一模型：**多对一模型将多个用户级线程映射到一个内核级线程上。只要一个用户级线程阻塞，就会导致整个进程阻塞。
- b. **一对一模型：**一对一模型将内核级线程与用户级线程一一对应。这样做的好处是当一个线程阻塞时，不影响其他线程的运行。
- c. **多对多模型：**多对多模型将多个用户级线程映射到多个内核级线程，采用这样的模型可以打破前两种模型对用户级线程的限制，不仅可以使多个用户级线程真正意义上并行执行，而且不会限制用户级线程的数量。

0.171 03966-进程通信

1. 共享存储器系统

为了传输大量数据，在存储器中划出一块共享存储区，多个进程可以通过对共享存储区进行读写来实现通信。进程在通信前，向系统申请建立一个共享存储区，并指定该共享存储区的关键字。若该共享存储区已经建立，则将该共享存储区的描述符返回给申请者。然后，申请者把获得的共享存储区附接到进程上。这样，进程便可以像读写普通存储器一样读写共享存储区了。

2. 消息传递系统

在消息传递系统中，进程间以消息为单位交换数据，用户直接利用系统提供的一组通信命令（原语）来实现通信。操作系统隐藏了通信的实现细节，简化了通信程序，得到了广泛应用。根据实现方式不同，消息传递系统可以分为以下两类：

- a. **直接通信方式：**发送进程直接把消息发送给接收进程，并将它挂在接收进程的消息缓冲队列上，接收进程从消息缓冲队列中取得消息。
- b. **间接通信方式：**发送进程把消息发送到某个中间实体（通常称为信箱）中，接收进程从中取得消息。这种通信方式又称为信箱通信方式。该通信方式广泛应用于计算机网络中，与之相应的通信系统称为电子邮件系统。

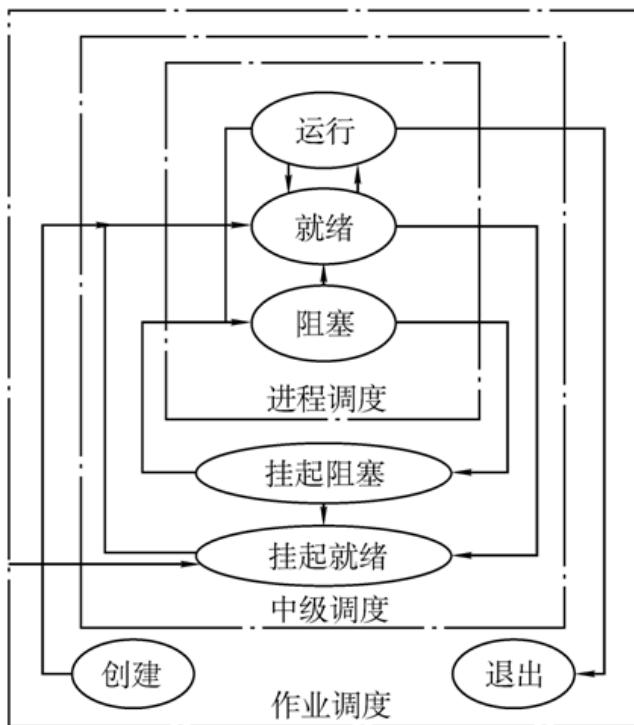
3. 管道通信系统

管道是用于连接读进程和写进程以实现它们之间通信的共享文件，向管道提供输入的发送进程（即写进程）以字符流形式将大量的数据送入管道，而接收管道输出的进程（即读进程）可以从管道中接收数据。

注意：管道是一个共享文件，不能单纯地从字面上仅将管道理解为一个传输通道。

0.172 03967-处理器的三级调度

下图给出了调度层次的示意图。从图中可以看出，一个作业从提交开始直到完成，往往要经历三级调度。



1. 高级调度（作业调度）

高级调度又称为宏观调度、作业调度或者长程调度，其主要任务是按照一定的原则从外存上处于后备状态的作业中选择一个或者多个，给它们分配内存、输入输出设备等必要资源，并建立相应的进程，以使该作业具有获得竞争处理器的权利。

2. 中级调度

中级调度又称为中程调度或者交换调度，其主要任务是按照给定的原则和策略，将处于外存对换区中的具备运行条件的进程调入内存，或者将处于内存中的暂时不能运行的进程交换到外存对换区。中级调度主要涉及内存管理与扩充（其实中级调度可以理解为在换页时将页面在外存与内存之间调度）。

3. 低级调度（进程调度）

低级调度又称为微观调度、进程调度或者短程调度，其主要任务是按照某种策略和方法从就绪队列中选取一个进程，将处理器分配给它。进程调度的运行频率很高，一般隔几十毫秒要运行一次。后面将对此进行详细讲解。

4. 作业调度与进程调度区别

1. 作业调度为进程被调用作准备，进程调度使进程被调用。换言之，作业调度的结果是为作业创建进程，而进程调度的结果是进程被执行。
2. 作业调度次数少，进程调度频率高。
3. 有的系统可以不设置作业调度，但进程调度必须有。

0.173 03968-调度的基本原则

不同调度算法有不同的调度策略，这也决定了调度算法对不同类型的作业影响不同。在选择调度算法时，必须考虑不同算法的特性。为了衡量调度算法的性能，人们提出了一些评价标准。

1. CPU 利用率

CPU 是系统最重要、也是最昂贵的资源，其利用率是评价调度算法的重要指标。

2. 系统吞吐量

系统吞吐量表示单位时间内 CPU 完成作业的数量。对长作业来说，由于它要占用较长的 CPU 处理时间，因此会导致系统吞吐量下降，而对短作业来说则相反。

3. 响应时间

在交互系统中，尤其在多用户系统中，多个用户同时对系统进行操作，都要求在一定时间内得到响应，不能使某些用户的进程长期得不到调用。

4. 周转时间

从每个作业的角度来看，完成该作业的时间是至关重要的，通常用周转时间或者带权周转时间来衡量。

a. 周转时间

周转时间是指作业从提交至完成的时间间隔，包括等待时间和执行时间；

周转时间 T_i 用公式表示为作业 i 的周转时间 $T_i = \text{作业 } i \text{ 的完成时间} - \text{作业 } i \text{ 的提交时间}$ 。

b. 平均周转时间

平均周转时间是指多个作业（例如 n 个作业）周转时间的平均值；

平均周转时间用 T 表示为： $T = (T_1 + T_2 + \dots + T_n) / n$ 。

c. 带权周转时间

带权周转时间是指作业周转时间与运行时间的比；

作业 i 的带权周转时间 W_i 表示为： $W_i = \text{作业 } i \text{ 的周转时间} / \text{作业 } i \text{ 的运行时间}$ 。

d. 平均带权周转时间

与平均周转时间类似，平均带权周转时间是多个作业的带权周转时间的平均值。

0.174 03969-进程调度

1. 进程调度的功能

- a. 记录系统中所有进程的有关情况以及状态特征；
- b. 选择获得处理器的进程；
- c. 处理器分配。

2. 引起进程调度的原因（重点之重点）

- a. 当前运行进程运行结束；
- b. 当前运行进程因某种原因从运行状态进入阻塞状态；
- c. 执行完系统调用等系统程序后返回用户进程；
- d. 在采用抢占调度方式的系统中，一个具有更高优先级的进程要求使用处理器；
- e. 在分时系统中，分配给该进程的时间片已用完。

3. 进程调度的方式

进程调度方式是指当某一个进程正在处理器上执行时，若有某个更为重要或紧迫的进程需要进行处理（即有优先级更高的进程进入就绪队列），此时应如何分配处理器。通常有两种进程调度方式。

抢占方式：又称为可剥夺方式。这种调度方式是指当一个进程正在处理器上执行时，若有某个优先级更高的进程进入就绪队列，则立即暂停正在执行的进程，将处理器分配给新进程。

非抢占方式：又称为不可剥夺方式。这种方式是指当某一个进程正在处理器上执行时，即使有某个优先级更高的进程进入就绪队列，仍然让正在执行的进程继续执行，直到该进程完成或因发生某种事件而进入完成或阻塞状态时，才把处理器分配给新进程。

0.175 03970-常见调度算法

1. 先来先服务调度算法

适用范围：可用于作业调度和进程调度。

基本思想是按照进程进入就绪队列的先后次序来分配处理器。先来先服务调度算法采用**非抢占的调度方式**。

2. 短作业优先调度算法

适用范围：可用于作业调度和进程调度。

基本思想是**把处理器分配给最快完成的作业（或进程）**。

3. 优先级调度算法

适用范围：可用于作业调度和进程调度。

基本思想是**把处理器分配给优先级最高的进程**。进程优先级通常分为两种：

a. 静态优先级：**是指优先级在创建进程时就已经确定了，在整个进程运行期间不再改变。**

b. 动态优先级：**是指在创建进程时，根据进程的特点及相关情况确定一个优先级，在进程运行过程中再根据情况的变化调整优先级。**

4. 时间片轮转调度算法

适用范围：主要用于进程调度。

基本思想是**处于就绪队列中的进程就可以依次轮流获得一个时间片的处理时间，然后重新回到队列尾部排队等待执行，如此不断循环，直至完成。**

5. 高响应比优先调度算法

适用范围：主要用于作业调度。

基本思想是**每次进行作业调度时，先计算就绪队列中的每个作业的响应比，挑选响应比最高的作业投入运行**。响应比的计算公式如下：

响应比 = 作业响应时间 / 估计运行时间，或者

响应比 = (作业等待时间 + 估计运行时间) / 估计运行时间

6. 多级队列调度算法

适用范围：主要用于进程调度。

基本思想是根据进程的性质或类型，将就绪队列划分为若干个独立的队列，每个进程固定地分属于一个队列。每个队列采用一种调度算法，不同的队列可以采用不同的调度算法。

7. 多级反馈队列调度算法

多级反馈队列调度算法是时间片轮转调度算法和优先级调度算法的综合与发展。具体讲解手机上观看不方便哦，请前往《操作系统高分笔记》的详细讲解。

0.176 03971-进程同步的基本概念

1. 两种形式的制约关系

间接相互制约关系（互斥）：两进程需要互斥使用临界资源。

直接相互制约关系（同步）：两进程需要合作完成同一任务。

2. 临界资源与临界区

a. **临界资源：**同时仅允许一个进程使用的资源称为临界资源。许多物理设备都属于临界资源，如打印机、绘图机等。

b. **临界区：**每个进程中访问临界资源的一段代码。

3. 互斥的概念与要求

为了禁止两个进程同时进入临界区，软件算法或同步机构都应遵循以下准则：

- a. **空闲让进：**当没有进程处于临界区时，可以允许一个请求进入临界区的进程立即进入自己的临界区。
- b. **忙则等待：**当已有进程进入其临界区时，其他试图进入临界区的进程必须等待。
- c. **有限等待：**对要求访问临界资源的进程，应保证能在有限的时间内进入自己的临界区。
- d. **让权等待：**当一个进程因为某些原因不能进入自己的临界区时，应释放处理器给其他进程。

4. 同步的概念与实现机制

一般来说，一个进程相对另一个进程的运行速度是不确定的。也就是说，进程之间是在异步环境下运行的。但是相互合作的进程需要在某些关键点上协调它们的工作。所谓进程同步，是指多个相互合作的进程在一些关键点上可能需要互相等待或互相交换信息，这种相互制约关系称为进程同步。

0.177 03972-互斥实现方法

1. 软件方法

软件方法较为繁琐，不适合在手机端展示其所有的讲解思路，建议参考《操作系统高分笔记》；

2. 硬件方法

硬件方法主要有两种：一种是中断屏蔽；另一种是硬件指令。

与软件实现方法相比，由于硬件方法采用处理器指令能够很好地把检查和修改操作结合成一个不可分割的整体，因此具有明显的优点。与此同时，也有一些缺点。

硬件方法的优点：

- a. 适用范围广：硬件方法适用于任何数目的进程，在单处理器和多处理器环境中完全相同。
- b. 简单：硬件方法的标志设置简单，含义明确，容易验证其正确性。
- c. 支持多个临界区：当一个进程内有多个临界区时，只需为每个临界区设立一个布尔变量。

硬件方法的缺点：

- a. 进程在等待进入临界区时要耗费处理器时间，不能实现“让权等待”（需要软件配合进行判断）；
- b. 进入临界区的进程的选择算法用硬件实现有一些缺陷，可能会使一些进程一直选不上，从而导致“饥饿”现象。

0.178 03973-信号量

1. 信号量及同步原语

```

struct semaphore {
    //信号量由二元组构成，整型变量和等待队列
    int count;
    queueType queue;
};

wait (semaphore s) //P操作
{
    s.count--;
    if(s.count<0)
    {
        阻塞该进程;
        将该进程插入等待队列s.queue;
    }
}

signal (semaphore s) //V操作
{
    s.count++;
    if(s.count<=0)
    {
        从等待队列s.queue取出第一个进程P;
        将P插入就绪队列;
    }
}

```

P、V 操作均为不可分割的原子操作，这保证了对信号量进行操作过程中不会被打断或阻塞。P 操作相当于申请资源，V 操作相当于释放资源。P 操作和 V 操作在系统中一定是成对出现，但未必在一个进程中，可以分布在不同进程中。

2. 信号量的分类

a. 整型信号量：整型信号量是一个整型量 s，除初始化外，仅能通过标准的原子操作 P 和 V 来访问。整型信号量引入了 P、V 操作，但是在进行 P 操作时，如果无可用资源，则进程持续对该信号量进行测试，存在“忙等”现象，未遵循“让权等待”准则。

b. 记录型信号量（资源信号量）

为了解决整型信号量存在的“忙等”问题，添加了链表结构，用于链接所有等待该资源的进程，记录型信号量正是由于采用了记录型的数据结构而得名。

当进程对信号量进行 P 操作时，若此时无剩余资源可用，则进程自我阻塞，放弃处理器，并插入到等待链表中。可见，该机制遵循“让权等待”准则。当进程对信号量进行 V 操作时，若链表中仍有等待该资源的进程，则唤醒链表中的第一个等待进程。

3. 信号量的应用

a. 实现进程同步：假设存在并发进程 P1 和 P2。P1 中有一条语句 S1，P2 中有一条语句 S2，要求 S1 必须在 S2 之前执行。这种同步问题使用信号量就很好解决。

```

semaphore N=0; //设置信号量并设置初值为0
P1()
{
    S1;
    V(N);
}
P2();
{
    P(N);
    S2;
}

```

b. **实现进程互斥**: 假设有进程 P1 和 P2，两者有各自的临界区，但系统要求同时只能有一个进程进入自己的临界区，这里使用信号量可以很方便地解决临界区的互斥进入。设置信号量 N，初值为 1（即可用资源数为 1），只需要将临界区放在 P(N) 和 V(N) 之间即可实现两进程的互斥进入。

```

semaphore N=1;
P1()
{
    P(N);
    P1的临界区代码;
    V(N);
}
P2()
{
    P(N);
    P2的临界区代码;
    V(N);
}

```

当有两个或者多个进程需要互斥访问某资源时，可以设置一个初值为 1 的信号量，在这些进程的访问资源的代码前后分别对该信号量进行 P 操作和 V 操作，即可保证进程对该资源的互斥访问。

0.179 03974-经典同步问题

1. 生产者-消费者问题

生产者-消费者问题是著名的进程同步问题。它描述的是一组生产者向一组消费者提供产品，他们共享一个有界缓冲区，生产者向其中投入产品，消费者从中取走产品。生产者-消费者问题的同步程序结构描述如下：

```

Semaphore full=0; //满缓冲区数目
Semaphore empty=n; //空缓冲区数目
Semaphore mutex=1; //对有界缓冲区进行操作的互斥信号量
Producer()
{
    while(true)
    {
        Produce an item put in nextp; //nextp为临时缓冲区
        P(empty); //申请一个空缓冲区
        P(mutex); //申请使用缓冲池
        将产品放入缓冲池;
        V(mutex); //缓冲池使用完毕，释放互斥信号量
        V(full); //增加一个满缓冲区
    }
}
Consumer()
{
    while(true)
    {
        P(full); //申请一个满缓冲区
        P(mutex); //申请使用缓冲池
        取出产品;
        V(mutex); //缓冲池使用完毕，释放互斥信号量
        V(empty); //增加一个空缓冲区
        Consumer the item in nextc; //消费掉产品
    }
}

```

2. 读者-写者问题

有一个许多进程共享的数据区，这个数据区可以是一个文件或者主存的一块空间；有一些只读取这个数据区的进程（读者）和一些只往数据区写数据的进程（写者），此外还需要满足以下条件：

- 任意多个读者可以同时读这个文件。
- 一次只能有一个写者可以往文件中写（写者必须互斥）。
- 如果一个写者正在进行操作，禁止任何读进程读文件和其他任何写进程写文件。

需要分多种情况实现该问题：读者优先、公平情况和写者优先。这里只给出读者优先算法，其他两种情况请参考《操作系统高分笔记》；读者-写者问题的读者优先算法的同步程序结构描述如下：

```

semaphore rmutex=1; //初始化信号量rmutex，保证对于readcount的互斥访问
semaphore mutex=1; //初始化信号量mutex，保证对于数据区的写互斥
int readcount=0; //用于记录读者数量，初值为0
reader()
{
    while(true) //循环执行这段代码
    {
        P(rmutex); //申请readcount的使用权
        if(readcount==0) //如果此为第一个读者，要阻止写者进入
            P(mutex);
        readcount++; //读者数量加1
        V(rmutex); //释放readcount的使用权，允许其他读者使用
        进行读操作;
        P(rmutex); //申请readcount的使用权，要对其进行操作
        readcount--; //读者数量减1
        if(readcount==0)
            V(mutex); //如果没有读者了，则允许写者进入
        V(rmutex); //释放readcount的使用权，允许其他读者使用
    }
}
writer()
{
    while(true) //循环执行这段代码
    {
        P(mutex); //申请对数据区进行访问
        进行写操作;
        V(mutex); //释放数据区，允许其他进程读写
    }
}

```

3. 哲学家进餐问题

5个哲学家围绕一张圆桌而坐，桌子上放着5根筷子，每两个哲学家之间放一根；哲学家的动作包括思考和进餐，进餐时需要同时拿起他左边和右边的两根筷子，思考时则同时将两根筷子放回原处。哲学家进餐问题可以看做是并发进程执行时处理临界资源的一个典型问题。哲学家进餐问题的同步程序结构描述如下：

规定奇数号的哲学家先拿左边筷子，然后拿右边筷子；偶数号的哲学家则相反。

```
semaphore Fork[5]={1,1,1,1,1};  
philosopher(int i) //i=1,2,3,4,5  
{  
    while(true)  
    {  
        思考;  
        想吃饭;  
        If(i % 2 !=0) //判断是否为奇数号哲学家  
        {  
            //若为奇数号哲学家，则先拿左边筷子  
            P(Fork[i]);  
            P(Fork[(i + 1) % 5]);  
            进餐;  
            V(Fork[i]);  
            V(Fork[(i + 1) % 5]);  
        }  
        else //若为偶数号哲学家，则先拿右边筷子  
        {  
            P(Fork[(i + 1) % 5]);  
            P(Fork[i]);  
            进餐;  
            V(Fork[(i + 1) % 5]);  
            V(Fork[i]);  
        }  
    }  
}
```

4. 理发师问题

理发店有一位理发师、一把理发椅和若干供顾客等候用的凳子（这里假设有 n 个凳子）。如果没有顾客，则理发师在理发椅上睡觉。当一个顾客到来时，他必须先叫醒理发师；如果理发师正在给顾客理发，则如果有空凳子，该顾客等待；如果没有空凳子，顾客就离开。要为理发师和顾客各设计一段程序来描述其活动。理发师问题的同步程序结构描述如下：

```
int chairs=n + 1; //为顾客准备的凳子和理发椅的数量  
semaphore ready=0; //表示等待理发的顾客数量，初值为0  
semaphore finish=1; //理发师初始状态为空闲  
semaphore mutex=1; //互斥信号量  
barber()  
{  
    while(true)  
    { //理完一人，还有顾客吗  
        P(ready); //看看有没有顾客，如果没有就阻塞  
        理发;  
        P(mutex); //理发结束，对chairs进行操作  
        chairs++; //顾客走掉，座位空余出一个  
        V(mutex); //允许其他进程访问chairs  
        V(finish); //理发师空闲，可以为下一个顾客理发  
    }  
}  
customer()  
{  
    P(mutex); //申请使用chairs变量  
    if (chairs > 0) //如果当前有空余座位  
    {  
        chairs--; //占用一个位置  
        V(mutex); //允许其他进程访问chairs  
        V(ready); //等待理发，唤醒理发师  
        P(finish); //当理发师空闲时开始理发  
    }  
    else //没有空余座位，准备离开  
        V(mutex); //释放mutex，允许其他进程访问chairs  
}
```

0.180 03975-关于 P、V 问题的解题思路

步骤一：确定题目涉及的若干进程中哪些进程是并发的，分析清楚若干进程间的制约关系（互斥或同步）；

步骤二：根据刚才对制约关系和并发关系的分析，确定进程流程，依据进程流程和进程间的制约关系设置相关信号量或变量以及它们的初值（这里要明确每种信号量和变量的物理含义，以及初值的含义）；

步骤三：将设置好的信号量添加到进程流程的适当位置；

步骤四：根据添加好信号量的进程流程写出伪代码；

步骤五：检查伪代码是否正确完整。

步骤六：（可选）最后设计一个简单的用例测试一下伪代码的正确性。

0.181 03976-管程

1. 管程的基本概念

管程定义了一个数据结构和能为并发进程所执行的一组操作，这组操作能同步进程和改变管程中的数据。由管程的定义可知，管程由局部于管程的共享数据结构说明、操作这些数据结构的一组过程以及对局部于管程的数据结构设置初值的语句组成。

2. 管程的基本特征

- a. 局部于管程的数据只能被局部于管程内的过程所访问。
- b. 一个进程只有通过调用管程内的过程才能进入管程访问共享数据。
- c. 每次仅允许一个进程在管程内执行某个内部过程，即进程互斥地通过调用内部过程进入管程。其他想进入管程的过程必须等待，并阻塞在等待队列。

0.182 03977-死锁的概念

1. 死锁的基本概念

当多个进程因竞争系统资源或相互通信而处于永久阻塞状态时，若无外力作用，这些进程都将无法向前推进。这些进程中的每一个进程，均无限期地等待此组进程中某个其他进程占有的、自己永远无法得到的资源，这种现象称为死锁。

2. 发生死锁的条件

- a. 参与死锁的进程至少有两个。
- b. 每个参与死锁的进程均等待资源。
- c. 参与死锁的进程中至少有两个进程占有资源。
- d. 死锁进程是系统中当前进程集合的一个子集。

0.183 03978-死锁产生的原因和必要条件

1. 资源分类

可剥夺资源：是指虽然资源占有者进程需要使用该资源，但另一个进程可以强行把该资源从占有者进程处剥夺来归自己使用。

不可剥夺资源：是指除占有者进程不再需要使用该资源而主动释放资源，其他进程不得在占有者进程使用资源过程中强行剥夺。

2. 死锁产生的原因

- a. 死锁产生的原因是竞争资源；
- b. 虽然资源竞争可能导致死锁，但是资源竞争并不等于死锁，死锁产生的原因是系统资源不足和进程推进顺序不当；
- c. 系统资源不足是产生死锁的根本原因。

3. 死锁产生的必要条件

- a. 互斥条件；
- b. 不剥夺条件；

- c. 请求与保持条件；
- d. 环路等待条件；

要产生死锁，这 4 个条件缺一不可，因此可以通过破坏其中的一个或几个条件来避免死锁的产生。

0.184 03979-处理死锁的基本方法

目前用于处理死锁的方法主要有以下 4 种：

1. **鸵鸟算法**：指像鸵鸟一样对死锁视而不见，即不理睬死锁。
2. **预防死锁**：通过设置某些限制条件，去破坏产生死锁的 4 个必要条件中的一个或几个来预防死锁的产生。
3. **避免死锁**：在资源的动态分配过程中，用某种方法防止系统进入不安全状态，从而避免死锁的产生。
4. **检测及解除死锁**：通过系统的检测机构及时地检测出死锁的发生，然后采取某种措施解除死锁。

0.185 03980-死锁的预防

预防死锁的发生，只需破坏死锁产生的 4 个必要条件之一即可。下面具体分析与这 4 个条件相关的技术。

1. 互斥条件

为了破坏互斥条件，就要允许多个进程同时访问资源。但是这会受到资源本身固有特性的限制，有些资源根本不能同时访问，只能互斥访问，如打印机就不允许多个进程在其运行期间交替打印数据，只能互斥使用。由此看来，通过破坏互斥条件来防止死锁的发生是不大可能的。

2. 不剥夺条件

为了破坏不剥夺条件，可以制定这样的策略：对于一个已经获得了某些资源的进程，若新的资源请求不能立即得到满足，则它必须释放所有已经获得的资源，以后需要资源时再重新申请。

3. 请求与保持条件

为了破坏请求与保持条件，可以采用预先静态分配方法。预先静态分配法要求进程在其运行之前一次性申请它所需要的全部资源，在它的资源未满足前，不把它投入运行。一旦投入运行后，这些资源就一直归它所有，也不再提出其他资源请求，这样就可以保证系统不会发生死锁。

4. 环路等待条件

为了破坏环路等待条件，可以采用有序资源分配法。有序资源分配法是将系统中的所有资源都按类型赋予一个编号（例如打印机为 1，磁带机为 2），要求每一个进程均严格按照编号递增的次序请求资源，同类资源一次申请完。

0.186 03981-死锁的避免

在避免死锁的办法中，所施加的限制条件较弱，有可能获得较好的系统性能。在该方法中把系统的状态分为安全状态和不安全状态，只要能使系统始终处于安全状态，便可以避免死锁的发生。

1. 安全状态与不安全状态

安全状态：按某方案分配资源系统一定不会进入死锁。

不安全状态：按某方案分配资源，系统有可能进入死锁。

安全状态一定不会导致死锁，不安全状态不一定会导致死锁，但是有可能导致死锁。

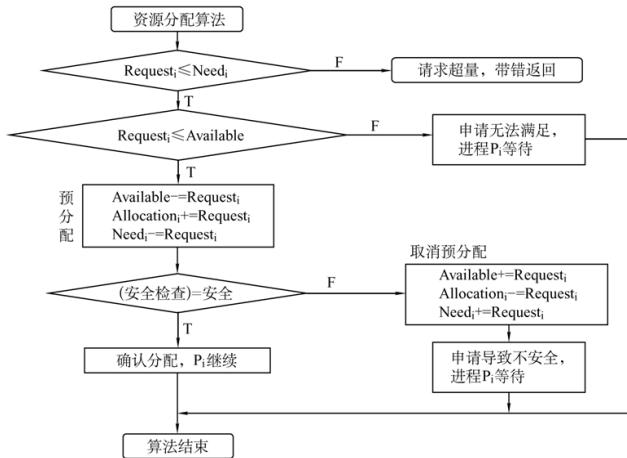
2. 银行家算法

银行家算法中使用的数据结构如下：

- 可利用资源向量 Available;
- 最大需求矩阵 Max;
- 分配矩阵 Allocation;
- 需求矩阵 Need。

“矩阵三兄弟”具有如下关系： $Need[i][j] = Max[i][j] - Allocation[i][j]$ ；

银行家算法的流程图如下：



0.187 03982-死锁的检测和解除

1. 死锁检测

a. 资源分配图

进程的死锁问题可以用有向图更加准确而形象地描述，这种有向图称为系统资源分配图。

b. 死锁定理

可以用简化资源分配图的方法来检测系统状态 S 是否是死锁状态。简化方法如下：

首先，在资源分配图中，找出一个既不阻塞又非孤立的进程节点 P_i 。因进程 P_i 获得了所需要的全部资源，它能继续运行直到完成，然后释放其占有的所有资源；

其次，进程 P_i 释放资源后，可以唤醒因等待这些资源而阻塞的进程，原来阻塞的进程可能变为非阻塞进程；

最后，重复前两步的简化过程后，若能消去图中所有边，使所有进程成为孤立节点，则称该图是可完全简化的；若不能通过任何过程使该图完全简化，则称该图是不可完全简化的。

S 为死锁状态的条件是：当且仅当 S 状态的资源分配图是不可完全简化的，该定理称为死锁定理。

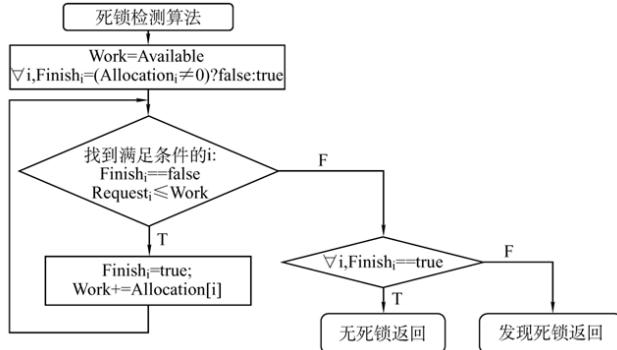
2. 死锁检测算法

(1) 基本思想是：获得某时刻 t 系统中各类可利用资源的数目向量 $available(t)$ ，对于系统中的一组进程 $\{P_1, P_2, \dots, P_n\}$ ，找出那些对各类资源请求数目均小于系统现有的各类可利用资源数目的进程。这样的进程可以获得它们所需要的全部资源并运行结束，当运行结束后，它们会释放所占有的全部资源，从而使可用资源数目增加，将这样的进程加入到可运行结束的进程序列中，然后对剩下的进程再进行上述考查。如果一组进程中有几个不属于该序列，那么它们会发生死锁。

(2) 与银行家算法和安全性算法类似，死锁检测算法也需要几个数据结构。

- a. Available: 表示当前可用资源的向量;
- b. Allocation: 表示已经分配的资源矩阵;
- c. Request: 表示进程请求资源的矩阵;
- d. 临时变量: Work 与 Finish 两个向量, 其作用和安全性算法中的相同。

(3) 死锁检测算法如下图所示。



3. 死锁解除

一旦检测出系统中出现了死锁, 就应使陷入死锁的进程从死锁状态中解脱出来。常用的解除死锁方法有两种:

- a. 剥夺资源: 从其他进程中抢占足够的资源给死锁的进程以解除其死锁状态;
- b. 撤销进程: 撤销一些进程, 直到有足够的资源分配给其他进程, 解除死锁状态。

0.188 03983-死锁与饿死

1. 进程饿死的概念

由于分配不到资源使某一进程长期处于等待, 并且等待时间给进程推进和响应带来明显影响时, 则称此时发生了**进程饥饿**, 当饥饿到一定程度的进程所赋予的任务即使完成也不再具有实际意义时, 称该**进程被饿死**。

2. 饿死与死锁的关系

饿死与死锁有一定联系: 二者都是由于竞争资源而引起的, 但又有明显差别, 主要表现在以下几个方面。

- a. 从进程状态考虑, 死锁进程都处于等待状态; 忙时等待 (处于运行或就绪状态) 的进程并非处于等待状态, 但却可能被饿死;
- b. 死锁进程等待的是永远不会被释放的资源; 而饿死进程等待的是会被释放但却不会分配给自己的资源, 表现为等待时间没有上界 (排队等待或忙时等待);
- c. 死锁一定发生了循环等待, 而饿死则不然。这也表明通过资源分配图可以检测死锁存在与否, 但却不能检测是否有进程饿死;
- d. 死锁一定涉及多个进程, 而饥饿或被饿死的进程可能只有一个。

饥饿和饿死与资源分配策略有关, 因而可从公平性方面考虑防止饥饿与饿死, 以确保所有进程不被忽视, 如多级反馈队列调度算法。

0.189 03984-定点数的运算

定点数的移位运算

(1) 逻辑移位

逻辑移位规则：逻辑左移时，高位移丢，低位补 0；逻辑右移时，低位移丢，高位添 0，例如，寄存器的内容为 10001010，逻辑左移为 00010100，逻辑右移为 01000101。

(2) 算术移位

算术移位规则总结如下表：

	码制	添补代码
正数	原码、补码、反码	0
负数	原码	0
	补码	左移添 0 右移添 1
	反码	1

原码定点数的加/减运算

1) 加法规则：先判断符号位，若相同，绝对值相加，结果符号位不变；若不同，则做减法，绝对值大的数减去绝对值小的数，结果符号与绝对值大的数相同。

2) 减法规则：两个原码表示的数相减，首先将减数符号取反，然后将被减数与符号取反后的减数按原码加法进行运算。

补码定点数的加/减运算

(1) 补码加法

两个数的补码相加，符号位参加运算，且两数和的补码等于两数的补码之和，公式是如下：

$$[x+y]_{\text{补}} = [x]_{\text{补}} + [y]_{\text{补}}$$

(2) 补码减法

$$[x-y]_{\text{补}} = [x+(-y)]_{\text{补}} = [x]_{\text{补}} + [-y]_{\text{补}}$$

$$\text{故 } [x-y]_{\text{补}} = [x]_{\text{补}} + [-y]_{\text{补}}$$

溢出的概念和判别方法

(1) 溢出产生的原因

溢出的概念：假设机器字长固定，不妨设为 8 位（包含一位符号位），根据前面掌握的知识，补码的取值范围应该是 -128 ~ 127，若现在两个数相加大于 127，或者小于 -128，则称为溢出，其中两数相加大于上界 127，称为上溢或者正溢出；两数相加小于下界 -128，称为下溢或者负溢出。定点小数的情况相同，若两个定点小数相加大于或等于 1，则称为上溢；若两个定点小数相加小于 -1，则称为下溢。

(2) 计算机是怎么判断溢出的呢？一共有以下 3 种方法。

1) 方法一：从两个数的符号位出发。

讲解之前先介绍两个必须知道的概念：

对于加法，只有在正数加正数或负数加负数两种情况下才有可能出现溢出，符号不同的两个数相加是不会溢出的。

对于减法，只有在正数减负数或负数减正数两种情况下才有可能出现溢出，符号相同的两个数相减是不会溢出的。

由于减法运算在机器中是用加法器实现的，因此可得出如下结论：不论是做加法，还是做减法，只要

实际参加操作的两个数（减法时即为被减数和“求补”以后的减数）符号相同，结果又与原操作数的符号不同，即认为溢出。

2) 方法二：仍然是从两个数的符号位出发。

在计算机中，一般都是通过数值部分最高位的进位（或者称为最高有效位）和符号位产生的进位进行“异或”操作，然后按照“异或”的结果进行判断（“异或”就是两数不同时为 1，两数相同时为 0）。若“异或”结果为 1，则为溢出；若“异或”结果为 0，则无溢出。

3) 方法三：用两位符号位判断溢出。

变形补码判断溢出的原则：当两位符号位不同时，表示溢出，否则，无溢出。无论是否发生溢出，高位符号位永远代表真正的符号位。再深入分析一下：假设现在运算结果的符号位为 01，而高位符号代表的是真正的符号位，可以判断这个结果一定是一个正数，既然是正数则肯定是正溢出；反之，运算结果的符号位是 10，则是负溢出。

定点数的乘除法内容较多，不适合手机观看，详见《计算机组成原理高分笔记》。

0.190 03985-浮点数的表示

在现代计算机中，为了便于软件移植，一般均采用 IEEE 754 标准来表示浮点数。在介绍 IEEE 754 标准前，有必要先介绍一下浮点数的表示形式。

既然尾数和阶码分别是定点小数和定点整数，即尾数和阶码都是有符号位的，那么就可以写出浮点数 N 的一般格式，如图 2-1 所示。

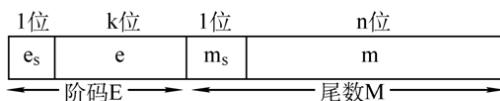


图 2-1 浮点数 N 的一般格式

从上图中可知：

- 1) 浮点数阶码的底 r 省略（一般容易出选择题）。
- 2) 阶符和阶码的位数 k 合起来反映浮点数的表示范围及小数点的实际位置。
- 3) 尾数 M 的位数 n 反映了浮点数的精度。
- 4) 尾数的符号为 m_s ，它也是整个浮点数的符号位，表示了该浮点数的正负。

在大多数机器中，尾数为纯小数，常用原码或补码表示；阶码为定点整数，常用补码或移码表示。

下面就开始介绍 IEEE 754 标准。

IEEE754 标准

采用 IEEE 754 标准来表示浮点数，格式如图 2-2 所示。



图 2-2 采用 IEEE 754 标准表示的浮点数

如数符为 s ，阶码为 e ，尾数为 M

则真值 $x = (-1)^s \times 1.M \times 2^{e-127}$ (32 位浮点数)

按照 IEEE 754 标准，常用的浮点数有以下 3 种，见下表：

	符号位 S	阶码	尾数	总位数	最大指数	最小指数	指数偏移量
短实数	1	8	23	32	+127	-126	+127
长实数	1	11	52	64	+1023	-1022	+1023
临时实数	1	15	64	80	+16383	-16382	+16383

0.191 03986-浮点数的加-减运算

对于浮点数的加/减运算，可以总结为以下 4 个步骤：

对阶，使两数的小数点位置对齐。

尾数求和，将对阶后的两尾数按定点加/减运算规则求和或者求差。

规格化，为增加有效数字的位数，提高运算精度，必须将求和或求差后的尾数规格化。

舍入，为提高精度，要考虑尾数右移时丢失的数值位。

当然，以上 4 个步骤完成后，还需要加上一步，即检查一下最后的结果是否溢出，由于浮点数的溢出完全是用阶码来判断的，假设阶码采用补码来表示，溢出就可以使用双符号位判断溢出的方式来判断此浮点数是否溢出，过程如下：

```
if (阶符 == 01)  
    上溢，需做中断处理；  
else if (阶符 == 10)  
    下溢，按机器零处理；  
else  
    结果正确；
```

0.192 03987-串行加法器和并行加法器

1. 全加器

介绍串行加法器和并行加法器之前，先大致了解一下全加器的结构。全加器是一个加法单元，而一个加法单元是一个三端输入、两端输出的加法网络，其结构如下图所示。

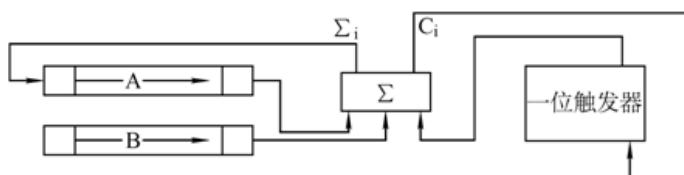


图 2-3 加法单元

其中， A_i 、 B_i 、 C_{i-1} 分别代表被加数 A_i 、加数 B_i 和低位传来的进位； C_i 代表本位向高位的进位， S_i 代表和 S_i 。很容易得出一个结论：当 A_i 、 B_i 、 C_{i-1} 组成的 3 位二进制数中 1 的个数为奇数时， $S_i=1$ ；当 A_i 、 B_i 、 C_{i-1} 组成的 3 位二进制数中 1 的个数大于或等于 2 时， $S_i=0$ 。

2. 串行加法器

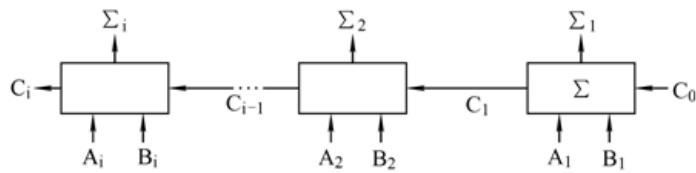
只设一个全加器的加法器称为串行加法器。两个操作数分别放在两个移位寄存器中，并且由移位寄存器从低位到高位串行地提供操作数进行相加。如果操作数长 16 位，就需要分成 16 步进行，每步产生一位和，串行地送入结果寄存器，而产生的进位信号只需要一位触发器，每完成一步，用新的进位覆盖旧的进位。串行加法器的结构如下图所示。



3. 并行加法器

(1) 并行加法器之串行进位链

并行加法器由若干个全加器构成，如下图所示。



(2) 并行加法器之并行进位链

通常将各位之间传递进位信号的逻辑连接构成的进位线路称为进位链。要想提高运算速度，一定要改善进位链，因此接下来主要从进位链着手来解决问题。

并行加法器中的进位信号是同时产生的，称为并行进位链。并行进位链又分为两种：**单重分组跳跃进位链**和**双重分组跳跃进位链**。

0.193 03988-算术逻辑单元的功能和结构

数字电路一般可分为组合逻辑电路和时序逻辑电路。

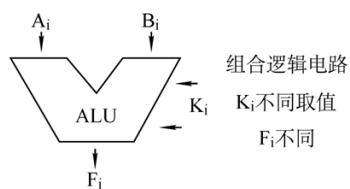
组合逻辑电路在逻辑功能上的特点是任意时刻的输出仅仅取决于该时刻的输入，与电路原来的状态无关。也就是说，组合逻辑电路没有“记忆”，运算后的结果要立刻送入寄存器保存。

时序逻辑电路在逻辑功能上的特点是任意时刻的输出不仅取决于当时的输入信号，还取决于电路原来的状态。也就是说，时序逻辑电路具有记忆元件，即触发器（能够存储一位信号的基本单元电路），可以记录前一时刻的输出状态（CPU就是一种复杂的时序逻辑电路）。

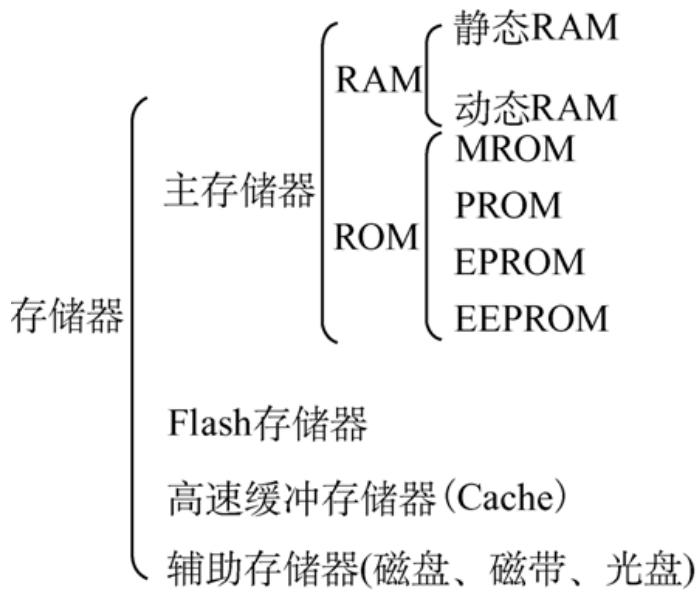
ALU是一种**组合逻辑电路**，因此在实际使用ALU时，其输入端口A和B必须与锁存器相连，而且在运算的过程中锁存器的内容是不变的，其输出也必须送至寄存器保存。

ALU的主要功能：ALU的功能不仅是执行各种算术（加、减、乘、除）和逻辑运算（“与”、“或”、“非”、“异或”等）操作的部件，还具有先行进位逻辑。其实在并行加法器之并行进位链里面就使用了ALU，只是当时还没有介绍ALU的概念，下面将会和前面的内容联系起来讲解。

ALU的电路框架如下图所示。

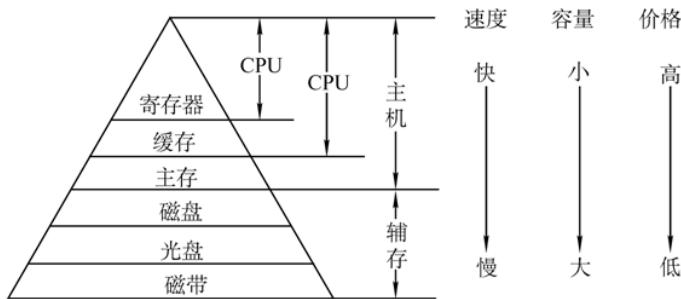


0.194 03989-存储器的分类

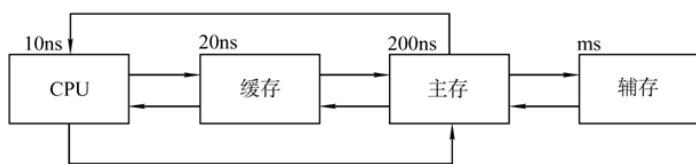


0.195 03990-存储器的层次化结构

存储器主要有 3 个性能指标：速度、容量、价格。一般来说，速度越高，价格就越高；容量越大，价格就越低，而且容量越大，速度必定越低，如下图所示。而理想的存储器应该是大容量、高速度、低价格。



存储系统的层次结构主要体现在缓存-主存和主存-辅存这两个存储层次上，如下图所示。显然，CPU 和缓存、主存都能直接交换信息；缓存能直接和 CPU、主存交换信息；主存可以和 CPU、缓存、辅存交换信息。



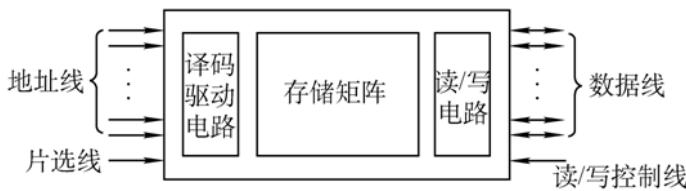
主存-辅存层次主要解决存储系统的容量问题。主存和辅存之间的数据交换是由硬件和操作系统共同完成的。

0.196 03991-半导体随机存取存储器基本概念

介绍 SRAM 和 DRAM 之前，首先了解半导体存储芯片的基本结构及其译码驱动方式。

1. 半导体存储芯片的基本结构

半导体存储芯片主要由存储矩阵、译码驱动电路和读/写电路组成，如下图所示。



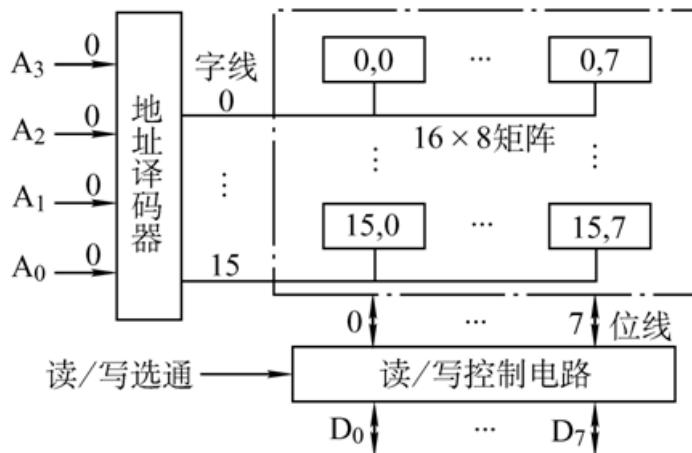
从上图中可以看出，地址线是单向的，数据线是双向的，剩下的属于控制线，控制线有读/写控制线和片选线两种。读/写控制线用来进行读/写操作，片选线用来选择存储芯片。由于一般半导体都是由很多的芯片组成的，因此需要用片选信号来选择要读或写哪一个芯片。

2. 半导体存储芯片的译码驱动方式

半导体存储芯片的译码驱动方式分为两种：线选法和重合法。

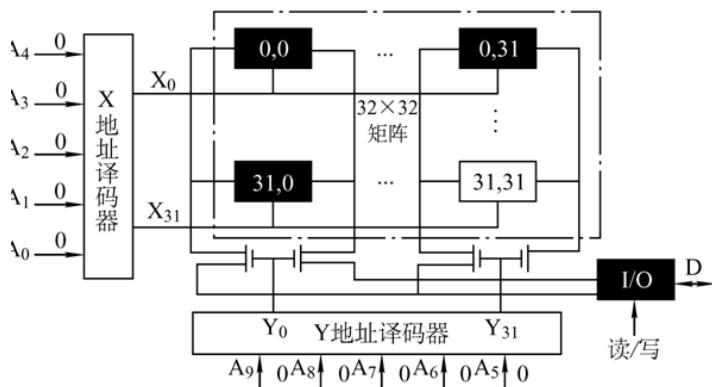
(1) 线选法（单译码）

首先假设该矩阵有 N 行，然后就可以通过公式 $\lceil \log_2 N \rceil$ 算出地址线所需要的根数。以下图为例，矩阵有 16 行，需要 4 根地址线 A_0, A_1, A_2, A_3 ，值 0000, 0001, 0010, ..., 1111 共 16 个数，分别代表了该矩阵的 16 行。由于图 3-5 中 A_0, A_1, A_2, A_3 的值都为 0，因此选中了第 0 行。选中之后再由读/写控制电路进行读写操作即可。另外，由于矩阵每行有 8 位，因此需要 8 根数据线。



(2) 重合法 (双译码)

线选法是选中矩阵的一行（在计算机中称为选中一个字），而重合法比线选法更细，它可以选中矩阵的某一个元素（在计算机中称为选中一位）。如下图所示，存储矩阵可以看成是 32×32 的矩阵。由线性代数可知，要想在矩阵中定位一个元素，需要行列的坐标，故此时不但需要行地址线，而且还需要列地址线。 32×32 矩阵里选中一行需要 5 根地址线， 32×32 矩阵里选中一列也需要 5 根地址线，一共需要 10 根地址线。图 3-6 中 10 根地址线全为 0，就选中了矩阵的 $(0, 0)$ 元素。



0.197 03992-SRAM

存储器的工作分为三大部分：保持存储信息、读数据和写数据。以下围绕这三大部分去阐述 SRAM 的工作原理。

(1) 保持存储信息

SRAM 主要使用六管静态 MOS 存储单元电路，利用触发器来保存信息（触发器在第 2 章中介绍过，即能够存储一位信号的基本单元电路），如图 3-7 所示。

规定： T_1 通、 T_2 止，存 “0”； T_1 止、 T_2 通，存 “1”。

符号介绍：符号 \diagdown 表示初始为高电平，加了信号就变成低电平；符号 \diagup 表示初始为低电平，加了信号就变成高电平。 \bar{W} 线用做读 “0” 写 “0”，即当要读 “0” 写 “0” 时， \bar{W} 线是高电平；W 线用做读 “1” 写 “1”，即当要读 “1” 写 “1” 时，W 线是低电平。

Z 表示字线，连接的是地址线； \bar{W} 和 W 表示位线，连接的是数据线。另外， $T_1 \sim T_6$ 为 MOS 管，如果为高电平，MOS 管就连通；如果为低电平，MOS 管就截止。

只要不访问存储单元，信息就可以一直保持下去。不访问是什么意思？符号 \diagup 表示初始为低电平，不访问就肯定不会改变低电平的性质，这样， T_5 和 T_6 就是低电平，即 T_5 和 T_6 截止了。这样， \bar{W} 和 W 就完全和记忆单元隔离了，即高阻状态。三态门的有关知识可参考辅助知识点 2。既然读/写线都被隔离了，那么信息就肯定不会改变了，也就保持住了。

(2) 读数据

读数据只要送地址并且发读命令就可以了，以读 “0” 信号为例，如图 3-8 所示。首先送地址，此时 Z 线变成高电平，这样， T_5 和 T_6 就导通了， \bar{W} 和 W 就和记忆单元连起来了。由于此记忆单元存储的是信号 “0”，根据 T_1 通、 T_2 止，存 “0”，此时 T_1 是通的， T_2 是截止的。

由于 T_1 是导通的，且 T_1 那端接地了， \bar{W} 为高电平，因此 \bar{W} 就和地产生了电势差（物理知识，记住就好），有电势差就有电流通过，于是产生读电流，读数据成功。

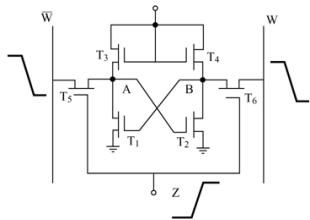


图 3-7 SRAM 的基本单元电路

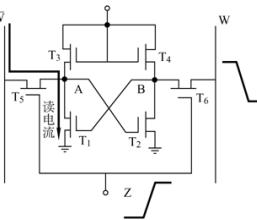


图 3-8 SRAM 读数据过程

(3) 写数据

写数据肯定要分为两部分：一部分是写在哪里；另一部分是写什么。此时，不但需要传输地址信号，还需要传输数据信号。以写 “1” 信号为例，首先传送地址，Z 线变成高电平，此时， T_5 和 T_6 就导通了，W 是读 “1” 写 “1” 线，应该将待写数据 “1” 送到 W 线，W 线就变成低电平。由于 T_6 是导通的，因此 B 点就为低电平，B 点低电平就使得 T_1 低电平，使 T_1 截止了；由于 \bar{W} 没有数据送入，因此一直是高电平。 \bar{W} 是高电平，且 T_5 导通，就使得 A 点是高电平，A 点高电平就使得 T_2 高电平，使 T_2 就导通了。 T_1 止、 T_2 通，就将 “1” 存进去了。

SRAM 的时序电路图不太重要，然而需要从中了解一些内容，即读周期和读时间之间的关系以及写周期和写时间之间的关系。

0.198 03993-DRAM

1. DRAM 的工作原理

DRAM 分为三部分即保持存储信息、读数据和写数据。

2. DRAM 存储器的刷新

通常有 3 种刷新方式：集中刷新、分散刷新和异步刷新。

集中刷新

把刷新操作集中到一段时间内集中进行（集中“歼灭”）。一般来说，电容上的电荷基本只能维持 2ms，在 2ms 内必须要刷新一次。将 2ms（2000us）看成一个刷新周期。假设存取周期为 0.5us，那么在一个刷新周期里有 4000 个存取周期。假设该存储矩阵有 32 行，则对 32 行集中刷新需要 16us。刷新的时候是不能进行读/写操作的，故称刷新这段时间为“死时间”，又称为访存“死区”。可以计算出死区的占用比例为 $32/4000=0.8\%$ （称为“死时间”率）。

分散刷新

将刷新操作分散进行，周期性地进行（分散“歼灭”）。分散刷新需要重点讲解，因为里面有一个不太容易理解的知识点。在分散刷新中，存储周期已经不再是传统的存储周期了。也就是说，此时存储周期不再等于读（写）周期，而这里扩展了操作的定义，即：

存储周期 = 读或写周期 + 刷新一行的时间

从这个公式可以看出，此时存储周期为读或写周期的两倍。此处刷新一行的时间又看成是等于存取周期的。

异步刷新

是一个折中方案，既不会像集中刷新那么大费周章，产生集中的固定时间，也不会像分散刷新那么频繁地刷新，而是有计划地刷新，时间分配十分合理。异步刷新是把存储矩阵的每行分散到 2ms 时间内刷新，但不是集中刷新，而是平均地分配。这就能保证当刷新完第一行后，再过 2ms 又可完成对第一行的下一次刷新。这样就不会像分散刷新那样每个存取周期都刷新某一行。

0.199 03996-主存储器与 CPU 的连接

要掌握主存储器与 CPU 的连接之前，首先需要了解一个考研必考的知识点，即**存储器容量的扩充**。一般存储器容量的扩充通常有 3 类：**位扩充、字扩充和字位扩充**。

1. 位扩充（增加 $a \times b$ 后面的 b）

位扩充指增加存储字长，例如，现要将 $1K \times 4$ 位的芯片组成 $1K \times 8$ 位的存储器。

2. 字扩充（增加 $a \times b$ 前面的 a）

字扩充是增加存储单元的个数，例如，现要将 $1K \times 8$ 位的芯片组成 $2K \times 8$ 位的存储器。

3. 字位扩充（增加 $a \times b$ 中的 a 和 b）

字位扩充既增加了存储单元的个数，又增加了存储字长，例如，要用 $1K \times 4$ 位的芯片组成 $4K \times 8$ 位的存储器。

注意：在进行字位扩充时，一定是先进行位扩充，再进行字扩充。

另外，记住下面这个芯片数量计算公式：

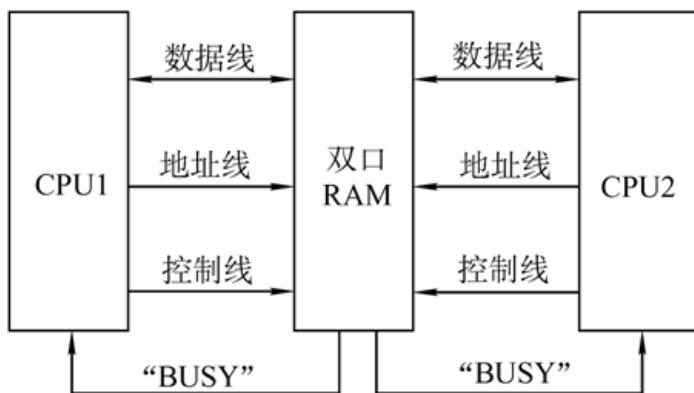
如果要求将容量为 $a \times b$ 的芯片组成容量为 $c \times d$ 的芯片，假设需要芯片的数量为 n，则 $n = (c * d) / (a * b)$ （该公式就是整个存储器的容量除以单个芯片的容量）。

以上掌握之后，只需要根据需要组成的芯片计算地址线和数据线的数量，再使用译码器（根据芯片的数量来选择，一般有 2-4 译码器和 3-8 译码器）连接起来即可。

0.200 03997-双口 RAM 和多模块存储器

双口 RAM

具有两组相互独立的地址线、数据线和读/写控制线，如下图所示。由于它可以进行并行的独立操作，因此是一种高速工作的存储器。很有可能在同一时间两个端口同时操作存储器的同一存储单元，这样就发生了冲突。为了解决此问题，特设置了 BUSY 标志。在这种情况下，当某存储单元被某端口访问时，就对另一个端口设置 BUSY 延迟，另一个端口就无法访问该存储单元。



单体多字存储器

单体多字存储器的使用前提是指令和数据在主存内必须连续存放，否则效果会很差。单体多字存储器把存储器的存储字字长增加 n 倍，以存放 n 个指令字或数据字，于是单体多字存储器的最大带宽比单体单字存储器的最大带宽提高 n 倍。因为程序使用指令字和数据字也存在一定的随机性，因此，一次读取的 n 个字很有可能是最近不需要的，正常情况下不可能达到最大带宽。缺点：必须是凑齐了 n 个数据字之后才能作为一个存储字一次写入存储器。

多模块存储器

多体并行存储器就是采用多个模块组成的存储器，每个模块有相同的容量和存取速度，各模块都有独立的地址寄存器、数据寄存器、地址译码器和读/写电路。多体并行存储器分为两种：高位交叉编址的多体存储器和低位交叉编址的多体存储器。

- 1) **高位交叉编址**。高位交叉编址是高位地址表示体号，低位地址来定位体内地址。按这种方式，可以在同一时间使得不同的请求源同时访问不同的体（如在某一时刻，CPU 在和第 0 个体交换数据，而此时第 1 个体正在和 I/O 交换数据），进而实现个体的并行工作。
- 2) **低位交叉编址**。低位地址可用来表示体号，高位地址可用来定位体内地址。这样，连续地址分布在相邻的不同模块内，而同一个模块的地址都是不连续的。因此，低位交叉编址存储器可以实现多模块流水线式并行存取，大大提高存储器的带宽。

假设模块存取周期为 T，总线传送周期为 τ ，且存储器由 m 个模块组成，若采用低位交叉编址的存储器，连续读取 n 个字所需要的时间 $t_1=T+(n-1)\tau$ ；若采用高位交叉编址的存储器，连续读取 n 个字所需要的时间 $t_2=nT$ 。

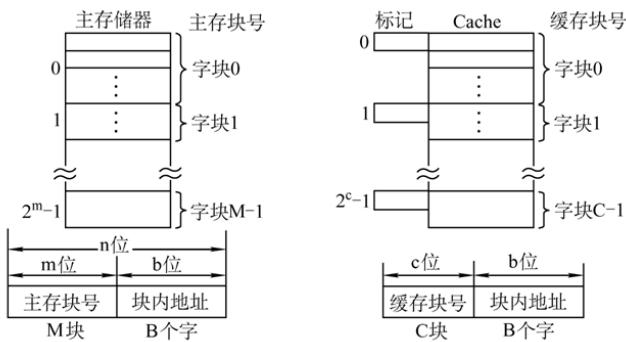
注意：高位交叉编址中的并行体现在不同的请求源并行地访问不同的体；低位交叉编址中的并行体现在同一请求源并行地访问不同的体。

0.201 03998-Cache 的基本工作原理

局部性原理：通过大量典型程序的分析，发现 CPU 从主存取指令或取数据，在一定时间内，只是对主存局部地址区域的访问（如循环程序、一些常数）。于是人们就想到一个办法，将 CPU 近期需要的程序提前存放到 Cache 中。这样 CPU 只需访问 Cache 就可以得到所需要的数据了。一般 Cache 采用高速的 SRAM 制作（主存一般使用 DRAM），其价格比主存高，容量远比主存小。

主存和 Cache 的编址

下图为 Cache-主存存储空间的基本结构。



(1) 主存

从上图中可以看出，主存由一个个的字块组成，当然每个字块包含 N 个字。主存的地址应该分为两部分：一部分用来寻找某个字块；另一部分用来寻找该字块中的字或字节（至于是字还是字节，需要看是哪种寻址方式，在后面例题中会详细讲解）。从上图中可以看出，主存的地址被分为两部分：高 m 位表示主存的块地址，低 b 位表示其块内的字或字节数，则 $2^m = M$ 表示主存的总块数。

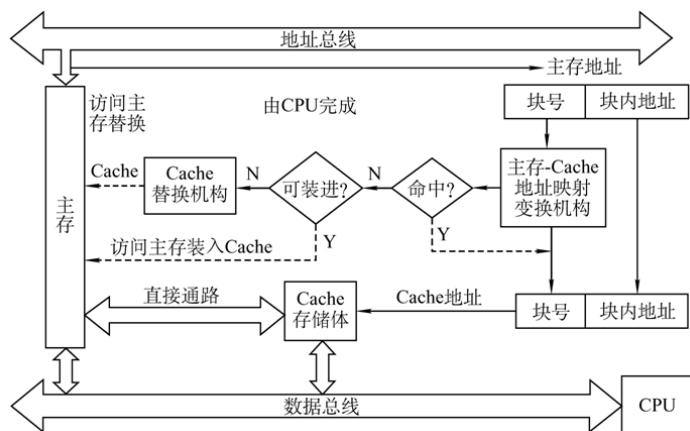
(2) Cache

同样，Cache 的地址也应该分为两部分：高 c 位表示 Cache 的块号，低 b 位表示其块内的字或字节数，则 $2^c = C$ 表示 Cache 的总块数，当然 Cache 的块数 C 应当远远小于主存块数 M。

既然 C 远远小于主存块数 M，一个缓存块不能唯一地、永久地对应一个主存块（因此在上图中给 Cache 设置了标记，相当于主存块的编号），那么肯定会存在一种情况，即某时刻 CPU 要访问的信息不在 Cache 中了，那应该怎么办？这种情况称为 Cache 不命中，或者 Cache 缺失。通常使用“命中率”或者“缺失率”来衡量 Cache 的效率。

Cache 的基本结构

讲解之前需要分析一下，Cache 的基本结构应该由哪几大部分组成？首先，CPU 送来的主存地址怎么能转换成 Cache 地址？这需要一个地址映射变换机构（手动滑一下，参考后面的知识点讲解）。其次，如果 Cache 内容已满，无法接受来自主存的块时，怎么去给 Cache 腾出位置来？这需要一个替换机构（手动滑一下，参考后面的知识点讲解），见下图。

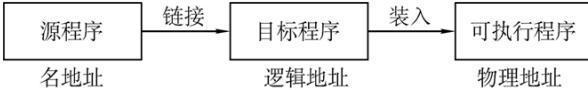


0.202 03999-内存管理概述

1. 内存管理的功能

- 内存的分配和回收；
- 地址变换；
- 扩充内存；
- 存储保护。

2. 应用程序的编译、链接与装入

- a. 从源程序到执行的进程，经历了编译、链接、装入 3 个步骤；
- b. 地址转换将逻辑地址转换为物理地址，这个过程叫做重定位。不同地址的变换过程如下图所示。

```
graph LR; A["源程序  
名地址"] -- 链接 --> B["目标程序  
逻辑地址"]; B -- 装入 --> C["可执行程序  
物理地址"]
```
- c. 程序的链接有 3 种方式：静态链接、运行时动态链接、装入时动态链接；
- d. 程序的装入也有 3 种方式：绝对装入、可重定位装入、动态运行装入。

3. 物理地址和逻辑地址

- a. 逻辑地址是指由程序产生的与段（与页无关，因为只有段对用户可见）相关的偏移地址部分。
- b. 物理地址是指出现在 CPU 外部地址总线上的寻址物理内存的地址信号，是逻辑地址变换后的最终结果地址，物理地址空间是指内存中物理地址单元的集合。

4. 内存保护

内存保护是为了防止一个作业有意或无意地破坏操作系统或其他作业。常用的存储保护方法有界限寄存器方法和存储保护键方法。

- a. **界限寄存器方法：**有两种，分别是上、下界寄存器方法和基址和限长寄存器方法。上、下界寄存器方法采用上、下界寄存器分别存放作业的结束地址和开始地址。在作业运行过程中，将每一个访问内存的地址都同这两个寄存器内容进行比较，如超出范围便产生保护性中断；基址和限长寄存器方法采用基址和限长寄存器分别存放作业的起始地址及作业的地址空间长度。当作业执行时，将每一个访问内存的相对地址和现场寄存器比较，如果超过了限长寄存器的值，则发出越界中断信号，并停止作业的运行。
- b. **存储保护键方法：**存储保护键方法是给每个存储块分配一个单独的保护键，它相当于一把“锁”。此外，进入系统的每个作业也被赋予一个保护键，它相当于一把“钥匙”。当作业运行时，检查“钥匙”和“锁”是否匹配，如果二者不匹配，则系统发出保护性中断信号，停止作业运行。

0.203 04001-连续分配管理方式

1. 单一连续分配

- a. 基本概念：单一连续分配是一种最简单的存储管理方式，通常只能用于单用户、单任务的操作系统中。这种存储管理方式将内存分为两个连续存储区域，其中的一个存储区域固定地分配给操作系统使用，通常放在内存低地址部分，另一个存储区域给用户作业使用。
- b. 技术采用：单一连续分配方式采用静态分配，适合单道程序，可采用覆盖技术。作业一旦进入内存，就要等到其结束后才能释放内存。因此，这种分配方式不支持虚拟存储器的实现，无法实现多道程序共享主存。
- c. 优点：管理简单，只需要很少的软件和硬件支持，且便于用户了解和使用，不存在其他用户干扰的问题。
- d. 缺点：是只能用于单用户、单任务的操作系统，内存中只装入一道作业运行，从而导致各类资源的利用率都很低。单一连续分配会产生内部碎片。

2. 固定分区分配

- a. 基本概念：固定分区分配（也称为固定分区存储管理）方法是最早使用的一种可运行多道程序的存储管理方法，它将内存空间划分为若干个固定大小的分区，每个分区中可以装入一道程序；
- b. 技术采用：固定分区分配中，程序通常采用静态重定位方式装入内存；
- c. 分区大小可以相等也可以不相等：

分区大小相等：缺乏灵活性，造成内存空间的浪费，当程序太大时，一个分区又不足以装入该程序，导致程序无法运行；

分区大小不相等：可把内存区划分成含有多个较小的分区、适量的中等分区及少量的大分区。可根据程序的大小为之分配适合的分区；

- d. 优点：可用于多道程序系统最简单的存储分配；
- e. 缺点：不能实现多进程共享一个主存区，利用率较低，会产生内部碎片。

3. 动态分区分配

a. 基本概念：动态分区分配又称为可变式分区分配，是一种动态划分存储器的分区方法。这种分配方法并不事先将主存划分成一块块的分区，而是在作业进入主存时，根据作业的大小动态地建立分区，并使分区的大小正好满足作业的需要。因此，系统中分区的大小是可变的，分区的数目也是可变的。

b. 分区分配算法：主要有以下四种，分别是首次适应算法、下次适应算法、最佳适应算法、最差适应算法；

- c. 优点：实现了多道程序共用主存；管理方案相对简单；实现存储保护的手段比较简单。

d. 缺点：主存利用不够充分，存在外部碎片；无法实现多进程共享存储器信息；无法实现主存的扩充，进程地址空间受实际存储空间的限制。

4. 内部碎片与外部碎片

- a. 内存碎片：分配给进程的分区中未被利用的碎片称为内部碎片；
- b. 外部碎片：而系统中剩余的无法利用的小块存储空间称为外部碎片。

0.204 04002-Cache 和主存之间的映射方式

地址映射变换机构是将 CPU 送来的主存地址转换为 Cache 地址。由于主存和 Cache 的块大小相同，块内地址都是相对于块的起始地址的偏移量（即低位地址相同），因此地址变换主要是主存块号与 Cache 块号之间的转换。地址变换主要有 3 种转换方式，即直接映射、全相联映射和组相联映射。

1. 直接映射

图 3-28 为直接映射方式主存与 Cache 中字块的对应关系，其中 Cache 分为 8 行，主存分为 256 行。由于 Cache 被分为 8 块，因此在主存中可以将每 8 块看成一个“轮回”，这样主存就可以被分为 32 个“轮回”。而每个“轮回”中的第 i 块只能映射到 Cache 的第 i 块，类似于上面生活场景助解中的个位数为 n 的学生只能坐到讲台上的 n 号位置。如果要用一个数学表达式来表示，则很快会想到用取模来对应，如下：

$$i = j \bmod C$$

其中，i 为 Cache 中的块号，j 为主存中的块号，C 为 Cache 的块数（图 3-28 中 C 等于 8）。上面的公式表示将主存第 j 块内容复制到 Cache 的 i 块中。

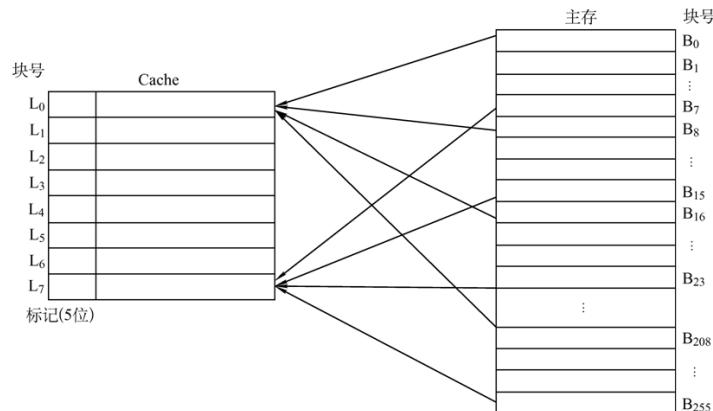


图 3-28 直接映射方式主存与 Cache 中字块的对应关系

2. 全相联映射

图 3-30 为全相联映射方式主存与 Cache 中字块的对应关系。全相联映射允许主存中每一个字块映射到 Cache 中的任何一块的位置上。《计算机组成原理高分笔记》书籍里面讲过，如果是全相联映射，那么每个人需要举着两位数号码的牌子才能识别这个人。在图 3-30 中，主存有 256 块，Cache 需要 8 位 ($2^8 = 256$) 来作为标记位，这样才能识别每一个主存块。回到图 3-28，因为直接映射只需要识别每个组号即可，所以主存大小是 Cache 的 32 倍（也就是说，主存需要分为 32 组），即 Cache 需要 5 位来作为标记位，这样才能识别该块属于哪一组。

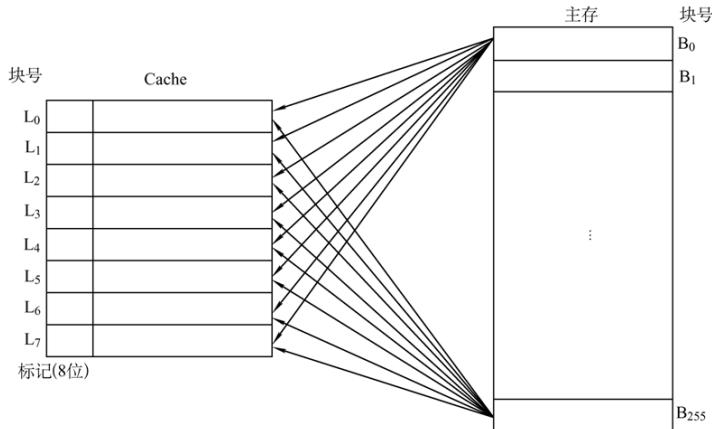


图 3-30 全相联映射方式主存与 Cache 中字块的对应关系

3. 组相联映射

图 3-32 为组相联映射方式主存与 Cache 中字块的对应关系。可以看出，组相联映射是对直接映射和全相联映射进行折中的一种方式。假设把 Cache 分为 Q 组，每组有 R 块，现在考生需要做的事情是把组相联映射的一组看作直接映射中的一块。同理，可以得到和直接映射中一样的公式为： $i=j \bmod Q$

其中， i 为 Cache 中的组号， j 为主存中的块号， Q 为 Cache 的组数（图 3-32 中 Q 等于 4）。通俗地说，上面的公式就是主存第 j 块内容复制到 Cache 的 i 组中，至于是第 i 组的哪一块，那就可以随意放了。

由于 Cache 分为 4 组，因此主存的 256 块应该分成 $256/4=64$ 个“轮回”，故需要 6 位 tag 来表示是哪一个“轮回”。

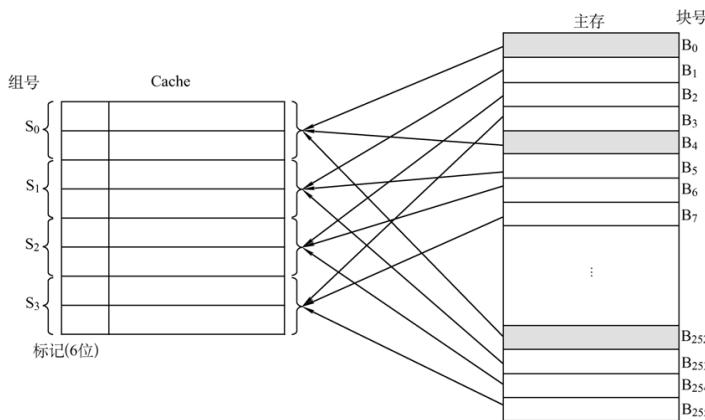
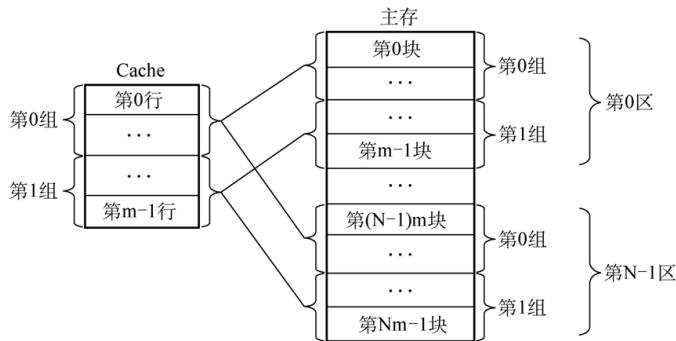


图 3-32 组相联映射方式主存与 Cache 中字块的对应关系

千万注意：组相联还有另外一种映射方式，因为 12 年第 17 题考查过，用上面的组相联无法解出此题，另一种组相联见下图：



该方式是先将主存块按 Cache 大小分区，再将各个分区中的块进行分组，同样 Cache 内也分组，组内分块。主存中不同区的相同序号的组和 Cache 同序号的组采用直接映射（例如主存的第 X 区的第 0 组只能映射到 Cache 的第 0 组），主存和 Cache 同序号的组内各块采用全相联映射，不同序号的组没有映射关系。

0.205 04004-Cache 写操作策略

由于 Cache 的内容只是主存部分内容的副本，因此它应当与主存内容保持一致。而 CPU 对 Cache 的写入更改了 Cache 的内容，就会导致 Cache 的内容和主存的内容不一致。如何能让 Cache 的内容与主存的内容保持一致就是 Cache 写操作策略需要完成的事情。Cache 写操作策略有如下 3 种形式。

1. 写回法

写回法要求：当 CPU 写 Cache 命中时，只修改 Cache 的内容，而不立即写入主存，只有当此行被换出时才写回主存，这种方式可以减少访问主存的次数。问题来了，那当换出此块的时候怎么能知道此块被修改过？实现这种方式时对 Cache 的每行都必须设置一个修改位（或者称为“脏位”）。当某行被换出时，根据此行的修改位是 0 还是 1（可以规定 1 代表修改过，0 代表没有修改），来决定将该行内容写回主存还是简单弃去。

注意：上面考虑的是 Cache 命中时，那不命中呢？如果 CPU 要对 Cache 中某块的某字进行修改，此时恰好此字不在 Cache 中，就需要从主存中找出包含此字的数据块。千万注意，CPU 不会在主存中直接修改，而是找到之后直接复制到 Cache 中进行修改，等从 Cache 中换出此块时，再复制到主存。

此知识点可设置综合题的细节题，如当使用写回法时，求 Cache 的位数。此时，一些考生可能不会加上修改位（隐含条件，有多少行就加多少修改位）。

2. 全写法

全写法要求：当写 Cache 命中时，Cache 与主存同时发生写修改，因而较好地保持了 Cache 与主存内容的一致性。很明显，此时 Cache 不需要每行都设置修改位。当写 Cache 未命中时，直接在主存中修改（和写回法不同）。至于在主存中修改后需不需要复制到 Cache 中，这个视情况而定，可以复制也可以不复制。

3. 写一次法

写一次法是基于写回法并结合全写法的写策略的一种形式（这种情况好像看得比较多，每次都是先介绍两种方式，第 3 种就采取折中方式，如 Cache 的映射方式就是如此）。写命中与写未命中的处理方法与写回法基本相同，仅仅是第一次写命中时要同时写入主存。

0.206 04005-虚拟存储器的基本概念

虚拟存储器的相关概念归纳如下：

- 1) 虚拟存储器是一个逻辑模型，并不是一个实际的物理存储器。

2) 虚拟存储器必须建立在主存-辅存结构基础上，但两者是有差别的：虚拟存储器允许使用比主存容量大得多的地址空间，并不是虚拟存储器最多只允许使用主存空间；虚拟存储器每次访问时，必须进行虚实地址变换，而非虚拟存储器则不必。

3) 虚拟存储器的作用是分隔地址空间，解决主存的容量问题和实现程序的重定位。

4) 虚拟存储器和 Cache 都基于程序局部性原理。

两者的相同点：都把程序中最近常用的部分驻留在高速的存储器中；一旦这部分程序不再常用，把它们送回到低速存储器中；这种换入、换出操作是由硬件或操作系统完成的，对用户透明；都力图使存储系统的性能接近高速存储器，而价格却接近低速存储器。

两者的不同点：Cache 用硬件实现，对操作系统透明，而虚拟存储器用操作系统与硬件相结合的方式实现；Cache 是一个物理存储器，而虚拟存储器仅是一个逻辑存储器，其物理结构建立在主存-辅存结构基础上。其实，此知识点中更需要掌握的是为了实现这种虚拟存储管理而需要的技术手段，如请求分页存储管理、请求分段存储管理、请求段页式存储管理，但这些知识点都属于操作系统的部分，可以参考操作系统对于此块知识点的讲解。

0.207 04006-页式虚拟存储器

页式虚拟存储器就是将其基本单位划分为页，且将主存的物理空间划分为与虚拟存储器等长的页。划分的页称为页面。主存的页称为实页，虚拟存储器的页称为虚页。

系统基本信息的传送单位是定长的页，需要通过地址变换机构实现访存过程，当访问页面不在主存时，通过页面置换算法将需要的页面调入主存。

优点：由于页面的起点、终点地址是固定的，因此页表简单，调入方便，主存空间浪费小。

缺点：由于页面不是逻辑上的独立实体，因此处理、保护和共享都不如段式虚拟存储器方便。

0.208 04012-请求分页存储管理方式

1. 请求分页原理

a. 基本概念：在分页存储管理的基础上，增加请求调页功能、页面置换功能所形成的一种虚拟存储系统。在请求分页存储管理中，作业运行之前，只要将当前需要的一部分页面装入主存，便可以启动作业运行。在作业运行过程中，若所要访问的页面不在主存中，则通过调页功能将其调入，同时还可以通过置换功能将暂时不用的页面置换到外存上，以便腾出内存空间。

b. 理解方式：请求分页 = 基本分页 + 请求调页功能 + 页面置换功能。

2. 页表结构

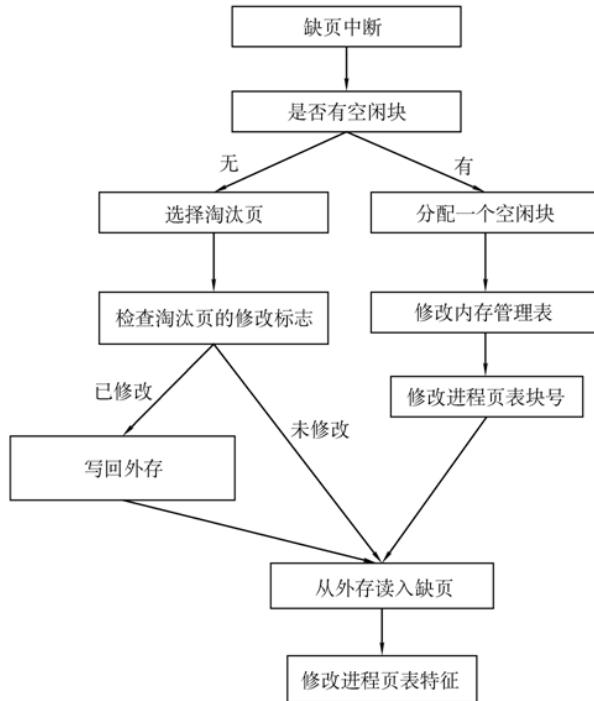
扩充后的页表项如下图所示。

页号	物理块号	状态位	访问字段	修改位	外存地址
----	------	-----	------	-----	------

- a. 页号和物理块号：其在分页存储管理中已经定义过，这两个信息是进行地址变换所必需的；
- b. 状态位（存在位）：用于判断页面是否在主存中；
- c. 访问字段：用于记录页面在一段时间内被访问的次数，或最近已有多久未被访问，以供置换算法在选择换出页面时参考；
- d. 修改位：用于表示页面调入内存后是否被修改过；
- e. 外存地址：用于指出页面在外存上的存放地址，供调入页面时使用。

3. 缺页中断与地址变换

a. 流程图：如下图所示。



b. 缺页中断与一般中断的区别：在指令的执行期间产生和处理缺页中断；一条指令可以产生多个缺页中断。

4. 请求分页管理方式的优缺点

a. 优点：

可以离散储存程序，降低了碎片数量；
提供虚拟存储器，提高了主存利用率，有利于多道程序运行，方便用户。

b. 缺点：

必须有硬件支持；
有些情况下系统会产生抖动现象；
程序最后一页仍然存在未被利用的部分空间。

0.209 04013-外存储器

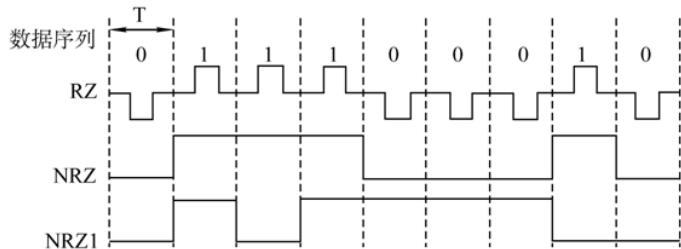
1. 硬盘存储器数据记录方式

记录数据的方式主要有 6 类，但考生只需掌握 3 种即可，如下图所示。

1) 归零制 (RZ)。记录“1”时，通正向脉冲电流；记录“0”时，通反向脉冲电流。“0”和“1”信息之间驱动电流归零。

2) 不归零制 (NRZ)。记录“1”时，通正向脉冲电流；记录“0”时，通反向脉冲电流。只有当相邻信息代码不同时，电流才改变方向，故称为“见变就翻”。

3) “见 1 就翻”的不归零制 (NRZ1)。只有记录“1”时，电流才改变方向，如下图所示。



2. 硬盘存储器的技术指标

(1) 记录密度

记录密度通常是指单位长度内所存储的二进制信息量。硬盘存储器一般需要用道密度和位密度一起来表示。

道密度指磁盘沿半径方向单位长度的磁道数（就是单位长度内有多少个同心圆）。相邻磁道之间的距离称为道距。

位密度（或称线密度）指单位长度磁道能记录二进制信息的位数。

注意：磁盘的所有磁道记录的信息量一定是相等的，并不是圆越大，记录的信息就越多。既然大圆和小圆记录的信息是一样多，那么每个磁道的位密度都是不同的（很明显，圈越大，位密度越小）。另外，一般题目中给出的磁盘位密度都是指最大位密度，也就是最内层圈的位密度。

(2) 存储容量

存储容量指外存所能存储的二进制信息总数量，以位或字节为单位。以硬盘存储器为例，存储容量可按下式计算： $C=n \times k \times s$

式中， C 为存储总容量； n 为存放信息的盘面数； k 为每个盘面的磁道数； s 为每条磁道上记录的二进制代码数。

(3) 平均寻址时间

平均寻址时间 = (最大寻道时间 + 最小寻道时间)/2+(最大等待时间 + 最小等待时间)/2

(4) 数据传输率

数据传输率指单位时间内磁表面存储器向主机传输数据的位数或字节数，它与记录密度和磁盘转动的速度有关。

(5) 误码率

如果从磁盘读出 N 位数据，有 M 位出错，那么误码率为 M/N 。为了减少误码率，硬盘存储器通常采用循环冗余码来校验数据。

3. 磁盘阵列

RAID 指由多个小容量磁盘代替一个大容量的磁盘。

1) RAID 0。最简单的磁盘阵列架构。写入时将资料分成数个小块，再同时送到不同的磁盘内存储，读取资料时也需要从不同的磁盘内读取，然后重新组合。正是由于 RAID 将资料分块存储，使得存储速度大大加快。它的缺点也显而易见，如果某一个磁盘的数据损坏，将会因资料不完整而无法读取，造成系统停止，甚至毁掉所有硬盘资料。

2) RAID 1。又称为镜像备份，意思就是可以将资料如镜子里的成像一样，在两个硬盘上各保持一份完全相同的备份，使得资料完全一样。写入时，相同的资料会同时写到磁盘阵列中的每一个磁盘；读取时仅从其中一个读取即可。由于每个硬盘都有一个镜像的硬盘，因此当某个硬盘出现损坏，并不会影响整个系统的运行。可以说 RAID 1 的容错性是相当好的。

4. 光盘存储器

(1) 只读型光盘

只读型光盘内的数据和程序由厂家事先写入，用户只能读出，不能修改或写入新的内容。非常类似于 ROM 的特性，故又将其称为 CD-ROM。

(2) 只写一次型光盘

只写一次型光盘允许用户写入信息，写入后可多次读出，但不能再修改，故称其为“写一次型”。

(3) 可擦写型光盘

可擦写型光盘由于是激光照射在磁性介质上进行读/写，因为这种光盘类似于磁盘，可以重复读写。

0.210 04014-页面置换算法

1. 最佳置换 (OPT) 算法

在预知一个进程的页面号引用串的情况下，每次都淘汰以后不再使用的或以后最迟再被使用的页面，这种算法就是最佳置换算法。

2. 先进先出 (FIFO) 算法

FIFO 算法是最简单的页面置换算法，每次总是淘汰先进入内存的页面，也就是淘汰在内存驻留时间最长的页面。

3. 最近最少使用 (LRU) 算法

选择最近最长时间没有被使用的页面予以淘汰，其思想是用以前的页面引用情况来预测将来会出现的页面引用情况，也就是假设一个页面刚被访问，那么不久该页面还会被访问。即最佳置换算法是“向后看”，而 LRU 算法则是“向前看”。

4. 时钟置换 (CLOCK) 算法

时钟置换 (CLOCK) 算法也称为最近未使用算法 (NRU)，是 LRU 和 FIFO 的折中。作为 LRU 的近似算法，CLOCK 算法给每个页面设置一个访问位，用以标识该页最近有没有被访问过。CLOCK 维护一个内存中所有页面的循环链表，当程序需要访问链表中存在的页面时，该页面的访问位就被置位为 1；否则，若程序要访问的页面没有在链表中，那就需要淘汰一个内存中的页面，于是一个指针就从上次被淘汰页面的下一个位置开始顺序地去遍历这个循环链表，当这个指针指向的页面的访问位为 1 时，就把该访问位清零，指针再向下移动，当指针指向的页面的访问位为 0 时，就选择淘汰掉这一页面，若遍历了一遍链表仍没找到可以淘汰的页面，那么就继续遍历下去。

改进型 CLOCK 算法，它考虑了页面载入内存后是否被修改的问题，增加了修改位。在访问位同为 0 的进程间优先淘汰没有修改过的页面，因为没有修改过的页面可以被直接淘汰掉，而修改过的页面需要写回到外存中。与简单 CLOCK 算法相比，该算法可减少磁盘 I/O 次数，但会增加扫描次数。

5. 其他页面置换算法

a. **最不常用置换 (LFU) 算法：**选择到当前时间为止访问次数最少的页面淘汰。该算法要求为每页设置一个访问计数器，每当页面被访问时，该页的访问计数器加 1。发生缺页中断时，淘汰计数值最小的页面，并将所有计数器清零。

b. **页面缓冲 (PBA) 算法：**PBA 算法是对 FIFO 算法的发展，通过建立置换页面的缓冲，找回刚被置换的页面，从而减少系统 I/O 的消耗。

0.211 04017-指令的基本格式

一条指令首先应该告诉机器，用户要干什么？例如加/减/乘/除或其他操作（由操作码实现）。确定操作后就要知道对谁进行操作（由地址码实现）。因此，一条指令应该由操作码和地址码两部分组成，如下图所示。



操作码：分为定长操作码和不定长操作码（也可称为扩展操作码或变长操作码）。一般将操作码放在每条指令的前一个字节或前多个字节，当读出操作码后就可以马上判定，这是一条零地址指令（如停机指令），还是单操作数指令（如求反、求补等操作），或者是双操作数指令（如加、减、乘、除等）。

地址码：有些书中将地址码称为操作数字段。下面分析地址码到底需要做什么。

- 1) 需要指出操作数的地址，即用哪里的数来操作。
- 2) 需要指出操作后的结果放在哪里，即给出结果存放的地址。
- 3) 需要指出该条指令执行结束后怎么办，即需要指出下一条指令的地址。

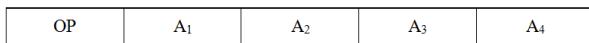
综上分析, 地址码应该指出操作数(一般称源操作数)的地址、运算结果需存放的地址、下一条指令的地址。由于操作数可以在主存, 也可以在寄存器, 因此以上所说的地址既可以是主存地址, 也可以是寄存器地址, 甚至还可以是 I/O 设备地址(I/O 设备地址在第 7 章中介绍)。

0.212 04018-定长操作码指令格式

1. 地址码

地址码用来指出该指令的源操作数的地址(一个或两个)、结果的地址及下一条指令的地址。这里的“地址”既可以是主存地址, 也可以是寄存器地址, 甚至还可以是 I/O 设备地址。但为了讲解方便, 一般都假设“地址”为“主存地址”。

(1) 四地址指令



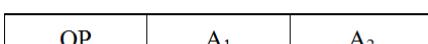
在上图中, OP 表示操作码; A₁、A₂ 分别为第一操作数和第二操作数地址; A₃ 为存放运算结果的地址; A₄ 为下一条指令的地址。

(2) 三地址指令



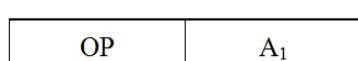
上图的指令可以完成操作 $(A_1)OP(A_2) \rightarrow A_3$, 后续指令的地址隐含在程序计数器中。

(3) 二地址指令



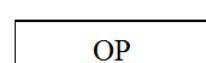
上图的指令可以完成操作 $(A_1)OP(A_2) \rightarrow A_1$ 或 $(A_1)OP(A_2) \rightarrow A_2$ 。A₁ 或 A₂ 既代表源操作数的地址, 又代表存放本次运算结果的地址。

(4) 一地址指令



如果其中一个操作数能隐藏在运算器的 ACC 就好了, 这样取其中一个源操作数就可以直接在 ACC 中进行了, 而且仅仅需要访问一次存储器取另外一个源操作数就够了。

(5) 零地址指令



确实有些指令还是可以进行优化的, 如停机指令、空操作指令等是不需要地址码的, 因此零地址指令产生了。如上图。

2. 操作码

上一个知识点中提到过, 操作码被分为定长操作码和不定长操作码。

定长操作码指令是在指令字的最高位部分分配固定的若干位表示操作码。对于具有 n 位操作码字段的指令系统, 最多能够表示 2^n 条指令。

0.213 04019-不定长操作码指令格式

不定长操作码指令格式就是操作码的长度不固定, 操作码的长度随地址码个数的减少而增加, 不同的地址数的指令可以具有不同长度的操作码。这样, 在满足需要的前提下, 有效地缩短了指令字长。在设计操作码指令格式时, 必须注意以下两点:

1) 不允许较短的操作码是较长操作码的前缀，例如，某条指令的操作码是 11，而另外一条指令的操作码是 111，11 是 111 的前缀，这样译码就会出现歧义，故不采纳（可以联系数据结构中的赫夫曼树编码来理解）。

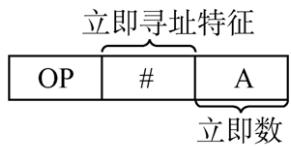
2) 各条指令的操作码一定不可以重复。

通常情况下，对使用频率较高的指令，分配较短的操作码；对使用频率较低的指令，分配较长的操作码，从而尽可能减少指令译码和分析的时间。

0.214 04021-常见寻址方式

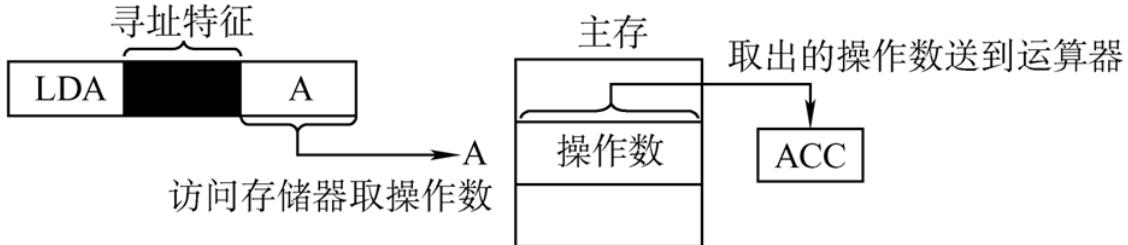
1. 立即寻址

这个寻址方式直接给出操作数，不需要给出地址去其他地方找操作数。也就是说，下图中的 A 不是操作数的地址，而就是操作数本身。通常把“#”符号放在立即数前面，以表示该寻址方式为立即寻址，如 #20H。其他寻址方式则不用特殊符号来表示。



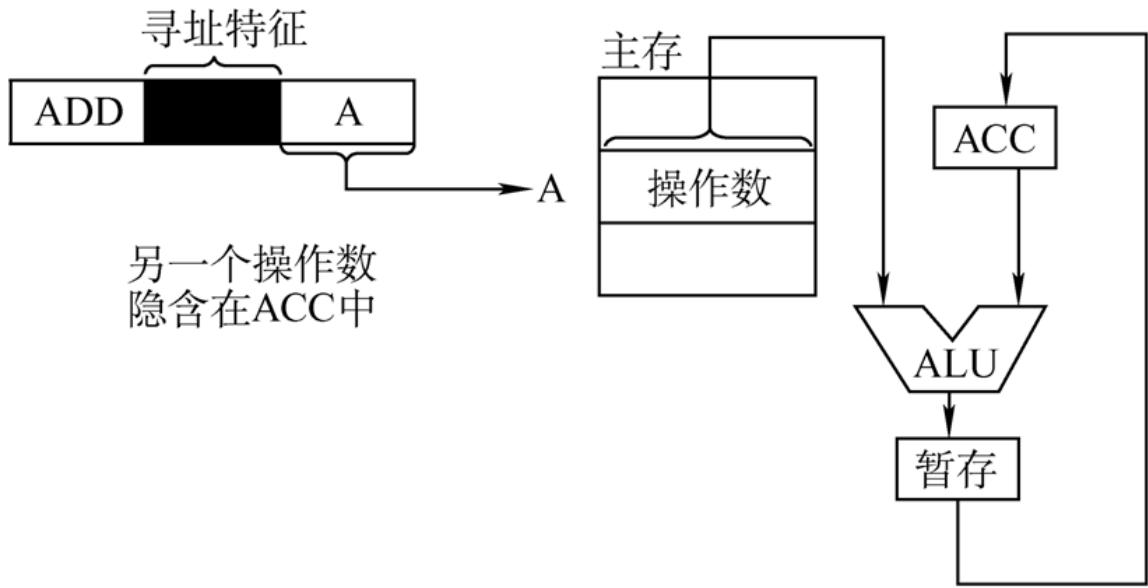
2. 直接寻址

首先将一个生活实例作为铺垫。就好像从淘宝购物，有些快递是直接送上门的，就类似于立即寻址，而有些快递是送到一个固定地点，然后发短信告诉客户，客户再去取。短信上的地址就是直接寻址中给出的地址，通过这个地址客户就可以拿到他们的物品（操作数）。也就是说，取到操作数之后再将操作数送往运算器或其他地方。可见，直接寻址在执行阶段需要访问一次存储器去取操作数，如下图所示。



3. 隐含寻址（了解即可）

隐含寻址指指令字中不明显地给出操作数地址，其操作数地址隐含在操作码或者某个寄存器中。其中最典型的例子就是一地址格式的加法指令，如下图所示。操作码显示的是 ADD，说明肯定至少需要两个操作数才能做加法运算，而地址码仅仅给出了一个操作数的地址，那另外一个操作数呢？没错，就在 ACC（累加器）中（这个没有为什么，就是规定）。换句话说，一地址格式的算术运算指令的另外一个操作数隐含在 ACC 中。



4. 间接寻址（非常重要）

直接寻址是直接给出了操作数的有效地址，即直接可以通过该地址找到操作数，但间接寻址指令给出的地址是操作数有效地址的地址。间接寻址又分为一次间接寻址和多次间接寻址。

5. 寄存器寻址

寄存器寻址比较简单，基本和直接寻址类似。在直接寻址的指令字中，地址码字段给出的是主存的地址，而在寄存器寻址的指令字中，地址码字段直接给出了寄存器编号 R_i ，则操作数的有效地址 $EA=R_i$ 。

6. 寄存器间接寻址

和寄存器寻址的不同之处在于， R_i 的内容不是操作数，而是操作数所在主存单元的地址号（很容易出选择题），即有效地址 $EA=(R_i)$ 。

7. 基址寻址

字面意义就是操作数的有效地址需要通过某个基础地址来形成。这个基础地址放在哪里呢？需要设置一个基址寄存器（BR），其操作数的有效地址 EA 等于指令字中的形式地址 A 与基址寄存器中的内容（称为基址）相加，即 $EA=A+(BR)$ 。

8. 变址寻址

变址寻址的有效地址 EA 等于指令字中的形式地址 A 与变址寄存器 IX 的内容相加之和，即 $EA=A+(IX)$ 。

在变址寻址中，变址寄存器的内容是由用户设定的，在程序执行过程中其值可变，而指令字中的形式地址 A 是不可变的。这点恰好和基址寄存器相反（注意出选择题）。

9. 相对寻址

相对寻址基于程序局部性原理（注意考查选择题）。相对寻址的有效地址是将程序计数器（PC）的内容与指令字中的形式地址 A 相加而成，如下： $EA=(PC)+A$ 。

0.215 04022-CISC 和 RISC 的基本概念

1. RISC 的主要特点总结

- 1) 选取简单指令，让复杂指令由简单指令的组合来实现；
- 2) 指令长度固定，指令格式种类少，寻址方式种类少；

- 3) 只有取数/存数指令访问存储器;
- 4) CPU 中有多个通用寄存器;
- 5) 采用流水线技术;
- 6) 控制器采用组合逻辑控制;
- 7) 采用优化的编译程序。

2. CISC 的主要特点总结

- 1) 指令系统复杂庞大;
- 2) 指令长度不固定, 指令格式种类多, 寻址方式种类多;
- 3) 可以访存的指令不受限制;
- 4) 由于 80CISC 各指令的使用频率差距太大。
- 5) 各种指令执行时间相差很大, 大多数指令需多个时钟周期才能完成。
- 6) 控制器大多数采用微程序控制。
- 7) 难以用优化编译生成高效的目标代码程序。

3. RISC 与 CISC 的比较 (了解即可)

- 1) RISC 比 CISC 更能提高计算机的运算速度, 例如, 由于 RISC 寄存器多, 因此就可以减少访存次数, 其次, 由于指令数和寻址方式少, 因此指令译码较快。
- 2) RISC 比 CISC 更便于设计, 可降低成本, 提高可靠性。
- 3) RISC 能有效支持高级语言程序。

0.216 04023-CPU 的功能

程序一旦被送出存储器, 控制器就要开始工作了, 负责协调并控制计算机各部件执行程序的指令序列, 其基本功能是取指令、分析指令和执行指令。除了以上三大基本功能外, 控制器还必须能控制程序的输入和运算结果的输出 (即控制主机与 I/O 设备交换信息) 以及对总线的管理, 甚至能处理机器运行过程中出现的异常情况 (如掉电) 和特殊请求 (如打印机请求打印一行字符), 即处理中断的能力。控制器的功能总结如下:

- 1) 控制器能自动地形成指令的地址, 并能发出取指令的命令, 将对应此地址的指令取到控制器中, 称为指令控制。
- 2) 取到指令之后, 应该产生完成每条指令所需要的控制命令, 称为操作控制。
- 3) 控制命令产生后, 需要对各种控制命令加以时间上的控制, 称为时间控制。
- 4) 在执行的过程中, 可能需要进行算术运算和逻辑运算, 称为数据加工。
- 5) 最后当然还有处理中断的能力, 称为中断处理。

0.217 04024-CPU 的基本结构

通过以上分析可知, 指令控制、操作控制、时间控制由控制单元 (CU) 完成; 数据加工由 ALU 完成; 中断处理由中断系统完成, 最后再加上一些寄存器, CPU 就被制作出来了, 如图 5-2 所示。将图 5-2 所示的 CPU 进行细化, 可以得到图 5-3 所示的 CPU 的结构。

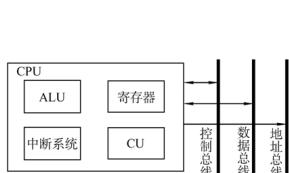


图 5-2 使用系统总线的 CPU

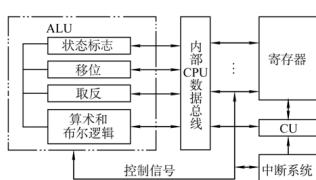


图 5-3 CPU 的内部结构

0.218 04025-CPU 中的主要寄存器

在 CPU 中至少要有 6 类寄存器。这些寄存器一般用来暂存一个计算机字，有时候也可以进行扩展，例如，某条指令是双字长，那么存放该指令的寄存器就必须扩展为双字长。下面详细介绍这些寄存器的功能与结构。

(1) 数据缓冲寄存器 (DR)

数据缓冲寄存器用来暂时存放由主存读出的一条指令或一个数据字；反之，当向主存存入一条指令或一个数据字时，也暂时将它们存放在数据缓冲寄存器中。

(2) 指令寄存器 (IR)

指令寄存器用来保存当前正在执行的指令。当执行一条指令时，先把它从内存取到数据缓冲寄存器中，然后传送至指令寄存器。

(3) 程序计数器 (PC)

为了保证程序能够连续地执行下去，CPU 必须采取某些手段来确定下一条指令的地址，而程序计数器正是起到了这种作用，所以通常又将程序计数器称为指令计数器。

(4) 地址寄存器 (AR)

地址寄存器用来保存当前 CPU 所访问的内存单元的地址。由于在内存和 CPU 之间存在着操作速度上的差别，因此必须使用地址寄存器来保持地址信息，直到内存的读/写操作完成为止。

(5) 累加寄存器 (ACC)

累加寄存器通常简称为累加器，它是一个通用寄存器。其功能是：当运算器的算术逻辑单元 (ALU) 执行算术或逻辑运算时，为 ALU 提供一个工作区。

(6) 状态条件寄存器 (PSW)

状态条件寄存器保存由算术指令和逻辑指令运行或测试的结果建立的各种条件码内容，如运算结果进位标志 (C)、运算结果溢出标志 (V)、运算结果为零标志 (Z)、运算结果为负标志 (N) 等。这些标志位通常分别由一位触发器保存。

0.219 04026-抖动现象与缺页率

1. Belady 异常

a. 定义：FIFO 置换算法的缺页率可能会随着所分配的物理块数的增加而增加，这种奇怪的现象就是 Belady 异常。

b. 原因：FIFO 算法的置换特征与进程访问内存的动态特征相矛盾，即被置换的页面并不是进程不会访问的。

注意：LRU 算法和最佳置换算法永远不会出现 Belady 异常，被归类为堆栈算法的页面置换算法也不可能出现 Belady 异常。

2. 抖动现象

a. 定义：若选用的页面置换算法不合适，可能会出现抖动现象：刚被淘汰的页面，过后不久又要访问，并且调入不久后又调出，如此反复，使得系统把大部分时间用在了页面的调入调出上，而几乎不能完成任何有效的工作，这种现象称为抖动（或颠簸）。

b. 原因：在请求分页系统中每个进程只能分配到所需全部内存空间的一部分。

3. 缺页率

a. 定义：假定一个作业共有 n 页，系统分配给该作业 m 页的空间 ($m < n$)。如果该作业在运行中共需要访问 A 次页面（即引用串长度为 A），其中所要访问页面不在内存，需要将所需页调入内存的次数为 F，则缺页率定义为 $f=F/A$ ，命中率即为 $1-f$ 。

b. 缺页率会受置换算法、分配的页面数量、页面大小等因素的影响。

c. 缺页率对于请求分页管理系统是很重要的，如果缺页率过高，会直接导致读取页面的平均时间增加，会使进程执行速度显著降低。因此，如何降低缺页率是一项非常重要的工作。

0.220 04027-请求分段存储管理系统

1. 请求分段存储管理系统

a. 定义：用户提供了一个比主存可用空间大得多的虚拟存储器。同样，虚拟存储器的实际容量由计算机的地址结构确定。

b. 原理：请求分段存储管理系统中，作业运行之前，只要将当前需要的若干段装入主存，便可启动作业运行。在作业运行过程中，如果要访问的分段不在主存中，则通过调段功能将其调入，同时还可以通过置换功能将暂时不用的分段置换到外存上，以便腾出内存空间。

c. 段表结构图：如下图所示。

段号	段长	内存始址	访问字段	修改位	状态位	外存地址
----	----	------	------	-----	-----	------

变换过程：段号、段长和内存始址 3 个信息是进行地址变换所必需的，其他字段的含义与请求分页存储管理相同。当段在内存中时，地址变换过程与分段存储管理相同；当段不在内存中时，应先将该段调入内存，然后再进行地址变换。

缺段处理过程：被访问的段不在主存中时，将产生一个缺段中断信号。操作系统处理该中断时，在主存中查找是否有足够大的分区存放该段。如果没有这样的分区，则检查空闲分区容量总和，确定是否需要对分区进行拼接，或者调出一个或几个段后再装入所需段。

0.221 04028-内存管理方式之间的比较

1. 三种离散分配方式的比较

在内存管理方式中，离散分配管理方式比连续分配管理方式重要得多，而且也是历年研究生考试的考查重点，因此将三种离散分配方式单独拿出来进行比较，见下表。

对比及联系 内存管理方式	分页存储管理		分段存储管理		段页式存储管理	
有无外部碎片	无		有		无	
有无内部碎片	有		无		有	
优点	内存利用率高，基本解决了内存零头问题		段拥有逻辑意义，便于共享、保护和动态链接		兼有两者的优点	
缺点	页缺乏逻辑意义，不能很好地满足用户	内存利用率不高，难以找到连续的空闲区放入整段	多访问一次内存			

2. 几种内存管理方式之间的比较

比较的方面	单一连续分配	分区		分页		基本分段	基本段页式
		固定分区	可变分区	基本分页	请求分页		
内存块的分配	连续	连续		离散		离散	离散
通用环境	单道	多道		多道		多道	多道
地址维数	一维	一维		一维		二维	二维
是否需要全部程序段在内存	是	是		是	否	是	是
扩展内存	交换	交换		交换	虚拟存储器	交换	交换
内存分配单位	整个内存的用户可用区	分区		页		段	页
地址重定位	静态	静态	动态	静态	动态	静态	静态
重定位机构	装入程序	装入程序	重定位寄存器	页表 页表控制寄存器 加法器	段表 段表控制寄存器 加法器	段表 页表 段表控制寄存器 加法器	
信息共享	不能	不能		可以，但限制多		可以	可以

0.222 04029-内存管理计算中地址的处理

1. 地址进制之间的转换

通常题目给出的地址形式分为两种：十进制与其他进制（通常是十六进制、八进制或二进制）。当题目中给出的地址是十进制时，通常地址是不会特别说明或者不带后缀的，例如“访问 7105 号单元”；而当给出的地址是其他进制时，通常会特别说明或者用符号后缀，十六进制、八进制与二进制对应的后缀分别

为字母 H、O、B，例如“访问 1A79H 号单元”就是十六进制地址，其中字母 H 表示该地址是以十六进制给出的。而在答题过程中，通常会进行进制之间的转换，在转换之后可以将转换后的地址加括号并加注下标来表明转换后的进制，例如将 17ACH（十六进制）转化为二进制，则可以表示为 $17ACH = 0001\ 0111\ 1010\ 1100$ 。

2. 逻辑地址转换为物理地址的过程

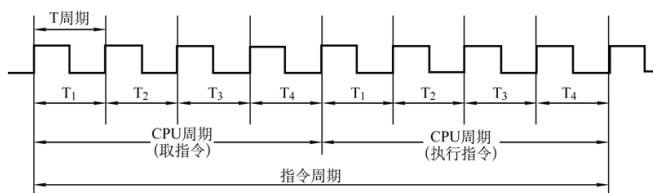
在请求分页系统中，若将逻辑地址转换为物理地址，则处理过程如下：

- 将其他进制转化为二进制，方便处理。
- 求出页号，页号为逻辑地址与页面大小的商，二进制下为地址高位。
- 求出页内位移，页内位移为逻辑地址与页面大小的余数，二进制下为地址低位。
- 根据题意产生页表，通过查找页表得到对应页的内存块号或页框号（页框号为把物理块地址除去页内位移若干位后剩下的地址高位，也可以简单理解为“物理地址的页号”）。
- 如果给出的是内存块号，则用内存块号乘以块大小，加上基址，再加上页内位移得到物理地址（给出这种条件的题目通常会给出物理地址的基址或者起始地址）。
- 如果给出的是页框号，则用页框号与页内位移进行拼接（页框号依然是高位，页内位移是低位，与逻辑地址的页号和页内位移构成类似），得到物理地址。
- 将二进制表示的物理地址根据题目要求转换为十六进制或者十进制。

0.223 04031-指令周期

CPU 每取出并执行一条指令所需的全部时间，即 CPU 完成一条指令的时间，称为**指令周期**。

指令周期被划分为几个不同的阶段，**每个阶段所需的时间称为机器周期**，又称为 CPU 工作周期或基本周期，**通常等于取指时间（或访存时间）**。时钟周期是时钟频率的倒数，也可称为节拍脉冲或 T 周期，是处理操作最基本的单位。一个指令周期由若干个机器周期组成，每个机器周期又由若干个时钟周期组成，如下图所示。



当 CPU 采用中断方式实现主存与 I/O 交换信息时，CPU 在每条指令的执行周期结束前，都要发出**中断查询信号**，以检测是否有 I/O 提出请求。如果有请求，则 CPU 要进入**中断响应阶段**，又称为**中断周期**。这样，一个完整的指令周期应包括取指、间址、执行和中断 4 个子周期，如图 5-7 所示。**完整的指令周期流程**如图 5-8 所示。

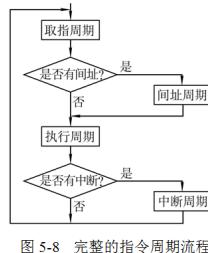
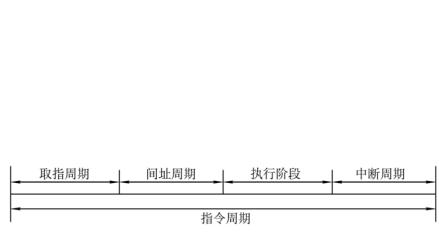


图 5-7 完整的指令周期

图 5-8 完整的指令周期流程

0.224 04032-请求分页管理方式中有效访问时间的计算

1. 访问的页在主存中，且访问页在快表中

在快表中就表明在内存中，则 $EAT = \text{查找快表时间} + \text{根据物理地址访存时间} = a+t$ 。

2. 访问的页在主存中，但不在快表中

EAT= 查找快表时间 + 查找页表时间 + 修改快表时间 (题目未给出则忽略不计, 如果给出, 通常与访问快表时间相同) + 根据物理地址访存时间 = $a+t+a+t=2(a+t)$ 。

3. 访问的页不在主存中

既然不在主存中, 就肯定不在快表中, 即发生缺页, 设处理缺页中断的时间为 T (包括将该页调入主存, 更新页表和快表的时间), 则 EAT= 查找快表时间 + 查找页表时间 + 处理缺页时间 (通常包括了更新页表和快表时间) + 查找快表时间 + 根据物理地址访存时间 = $a+t+T+a+t=T+2(a+t)$ 。

接下来加入缺页率和命中快表几率, 将上述 3 种情况组合起来, 形成完整有效访问时间计算公式。假设命中快表的几率为 d, 缺页率为 f, 则:

EAT= 查找快表时间 + $d \times$ 根据物理地址访存时间 + $(1-d) \times [$ 查找页表时间 + $f \times (\text{处理缺页时间} + \text{查找快表时间} + \text{根据物理地址访存时间}) + (1-f) \times (\text{修改快表时间} + \text{根据物理地址访存})] = a + d \times t + (1-d)[t + f(T+a+t) + (1-f)(a+t)]$ 。

4. 快表访问和修改时间

有些题目会说明系统中有快表, 如果没说明, 则视为没有快表, 将上述公式的命中率和访问时间变为 0 就可以了; 有些题目会说明忽略访问和修改快表时间, 则将访问时间 a 变为 0 就可以了。

5. 关于处理缺页中断时间 T 的计算

如果题目中没有说明被置换出的页面是否被修改, 则缺页中断时间统一为一个值 T。如果题目中说明了被置换的页面分为修改和未修改两种不同的情况, 假设被修改的概率为 n, 处理被修改的页面的时间为 T1, 处理未被修改的页面的时间为 T2, 则处理缺页时间 $T=n \times T1 + (1-n) \times T2$ 。

0.225 04033-指令的执行过程与信息流

(1) 取指周期

取指周期需要解决两个问题: 一个是 CPU 到哪个存储单元取指令; 另一个是如何形成后继指令地址。指令的地址由程序计数器 (PC) 给出。因此, 取指周期的操作为: 按 PC 内容取出指令, 并将 PC 内容递增。当出现转移情况时, 指令地址在执行周期被修改。

取指周期信息流如下:

1. (PC)→MAR //将要执行指令的地址放到地址缓冲寄存器
2. 1→R //发出读命令 (固定写法), 但是这个也可以不写, 后面会详细讲解这种细节问题
3. M(MAR)→MDR //将要执行的指令从存储器中读到数据缓冲寄存器, 其中 (MAR) 表示地址缓冲寄存器中的内容, 所以 M(MAR) 就表示在主存中此地址的内容, 即欲执行指令本身
4. (MDR)→IR //将要执行的指令打入指令寄存器
5. OP(IR)→CU //IR 表示指令本身, OP(IR) 表示指令的操作码, AD(IR) 表示指令的地址码
6. (PC)+1→PC //形成下一条指令的地址

(2) 间址周期 (并不是所有指令的执行过程中都会有间址周期)

间址周期是为了取出操作数的有效地址, 操作数的地址存放在指令所对应的存储器 (或者寄存器) 中, 然后到其所对应的存储器中去取操作数。

间址周期信息流如下:

1. AD(IR)→MAR //将指令字中的地址码 (形式地址) 打入地址缓冲寄存器
2. 1→R //发出读命令
3. M(MAR)→MDR //将有效地址从主存打入数据缓冲寄存器

(3) 执行周期

不同指令的执行周期操作命令不一样，所以没有统一的格式。

(4) 中断周期

执行周期结束后，CPU 需要查询是否有请求中断的事件发生，如果有则进入中断周期。中断隐指令保存的断点存在哪里？怎么寻找中断服务程序入口地址？只有这两个问题确定了，才能写出微指令序列。

前提：现假设程序断点保存至主存的“0”号单元，且采用硬件向量法寻找入口地址。中断周期的微指令序列如下：

1. $0 \rightarrow \text{MAR}$ //将主存“0”号单元的地址送入主存地址寄存器
2. $1 \rightarrow W$ //启动存储器写
3. $(\text{PC}) \rightarrow \text{MDR}$ //将 PC 的内容（程序断点）送入主存数据寄存器
4. $(\text{MDR}) \rightarrow M (\text{MAR})$ //将主存数据寄存器的内容写入 MAR 所指示的主存单元
5. 向量地址 $\rightarrow \text{PC}$ //将向量地址形成部件的输出送至 PC
6. $0 \rightarrow \text{EINT}$ //关中断，将允许中断触发器清零

以上就是中断周期的全部微指令操作。如果断点不是存入主存，而是存入堆栈，那么微程序指令又是什么？很简单，只需将上述步骤 1 改为：

1. $(\text{SP}) -1 \rightarrow \text{SP}$, 且 $(\text{SP}) \rightarrow \text{MAR}$ //这里假设先修改指针，后存入数据

0.226 04034-文件的基本概念

1. 文件的概念

在计算机中，大量的数据和信息是通过文件存储和管理的。文件系统负责管理文件，并为用户提供对文件进行存取、共享及保护的方法。

文件：有文件名的一组相关元素的集合，在文件系统中是一个最大的数据单位，它描述了一个对象集，每个文件都有一个文件名，用户通过文件名来访问文件。

2. 文件的属性

- a. 名称：文件名唯一，以容易读取的形式保存；
- b. 标识符：系统内文件的唯一标签，通常为数字，对用户来说是透明的；
- c. 文件类型：被支持不同类型的文件系统所使用；
- d. 文件位置：指向文件的指针；
- e. 文件的大小、建立时间、用户标识等。

3. 文件的分类

- a. 按用途分类：系统文件、库文件、用户文件；
- b. 按保护级别分类：只读文件、读写文件、执行文件、不保护文件；
- c. 按信息流向分类：输入文件、输出文件、输入输出文件。
- d. 按数据形式分类：源文件、目标文件、可执行文件。

4. 文件的操作

- a. 基本的文件操作：创建文件、删除文件、读文件、写文件、截断文件、设置文件的读/写位置等。
- b. 文件的打开操作：是指系统将文件的属性从外存复制到内存，并设定一个编号（或索引）返回给用户。以后当用户要对该文件进行操作时，只需利用编号（或索引号）向系统提出请求即可。这样避免了系统对文件的再次检索，节约了检索开销，也提高了对文件的操作速度。
- c. 文件的关闭操作：是指系统将打开的文件的编号（或索引号）删除，并销毁其文件控制块。如果文件被修改，则需要将修改保存到外存。

0.227 04036-文件的逻辑结构

1. 无结构的流式文件

a. 基本概念：由若干个字符组成，**可以看做是一个字符流，称为流式文件**。可以将流式文件看成记录式文件的特例。在 UNIX 系统中，所有文件都被视为流式文件，系统不对文件进行格式处理。

2. 有结构的记录式文件（顺序文件）

a. 基本概念：顺序文件又称为连续结构，是一种最简单的文件结构，**其将一个逻辑文件的信息连续存放**。以顺序结构存放的文件称为顺序文件或连续文件。

b. 分类：按照记录**是否定长**，顺序文件分为**定长记录顺序文件**和**变长记录顺序文件**。按照文件中记录**是否按照关键字排序**，顺序文件又分为**串结构**和**顺序结构**：串结构中各记录之间的顺序与关键字无关，而顺序结构中所有记录按照关键字顺序排序。

c. 优点：顺序文件**存取时速度较快**；当文件为定长记录文件时，还可以根据文件起始地址及记录长度进行**随机访问**。

d. 缺点：因为文件存储要求连续的存储空间，所以会产生碎片，同时也不利于文件的扩充。

3. 有结构的记录式文件（索引文件）

a. 基本概念：**索引结构为一个逻辑文件的信息建立一个索引表**。索引表中的表目存放文件记录的长度和所在逻辑文件的起始位置，因此逻辑文件中不再保存记录的长度信息。索引表本身是一个定长文件，每个逻辑块可以是变长的，索引表和逻辑文件两者构成了索引文件。

b. 优点：可以进行随机访问，也易于进行文件的增删。

c. 缺点：索引表的使用增加了存储空间的开销，另外，索引表的查找策略对文件系统的效率影响很大。

4. 有结构的记录式文件（索引顺序文件）

a. 基本概念：**索引顺序文件是顺序和索引两种形式的结合**。索引顺序文件将顺序文件中的所有记录分为若干个组，为顺序文件建立一张索引表，并为每组中的第一个记录在索引表建立一个索引项，其中含有该记录的关键字和指向该记录的指针。

索引表中包含**关键字**和**指针**两个数据项，索引表中索引项按照关键字顺序排列。索引顺序文件的逻辑文件（主文件）是一个顺序文件，每个分组内部的关键字不必有序排列，但是组与组之间的关键字是有序排列的。

b. 优点：大大提高了顺序存取的速度。

c. 缺点：仍然需要配置一个索引表，增加了存储开销。

5. 直接文件和散列（Hash）文件

a. 基本概念：**建立关键字和相应记录物理地址之间的对应关系**，这样就可以直接通过关键字的值找到**记录的物理地址**，也就是说，关键字的值决定了记录的物理地址，这种结构的文件称为直接文件。这种映射结构不同于顺序文件或索引文件，没有顺序的特性。

b. 优点：通过散列函数对关键字进行转换，转换结果直接决定记录的物理地址。**散列文件有很高的存取速度**。

c. 缺点：由于不同关键字的散列函数值相同而引起冲突。

0.228 04037-目录结构

1. 文件目录

a. 基本概念：文件的组织可以通过目录来实现，文件说明的集合称为文件目录。**目录最基本的功能就是通过文件名存取文件**。

b. 基本功能：实现“按名存取”、提高检索速度、允许文件同名、允许文件共享。

注意：文件目录也作为一个文件来处理，称为目录文件。由于文件系统中一般有很多文件，文件目录也很大，因此文件目录并不放在主存中，而是放在外存中。

2. 文件控制块和索引节点

a. **文件控制块：**文件控制块主要由文件名、文件的结构、文件的物理位置、存取控制信息、管理信息等组成。

b. **索引节点：**有些系统采用了文件名与文件描述信息分开的方法，将文件描述信息单独形成一个索引节点，简称为 i 节点。

每个文件都有唯一的磁盘索引节点，主要包括以下内容：文件主标识符、文件类型、文件存取权限、文件物理地址、文件长度、文件链接计数、文件存取时间。

当文件被打开时，磁盘索引节点被复制到内存的索引节点中，以便使用。存放在内存中的索引节点称为内存索引节点，其增加了以下内容：索引节点编号、状态、访问计数、逻辑设备号、链接指针。

3. 单级目录结构

a. 基本概念：单级目录结构（或称为一级目录结构）是最简单的目录结构。在整个文件系统中，**单级目录结构只建立一张目录表，每个文件占据其中的一个表目。**

b. 优点：易于实现，管理简单。

c. 缺点：不允许文件重名、文件查找速度慢。

4. 二级目录结构

a. 基本概念：**二级目录结构将文件目录分成主文件目录和用户文件目录。**系统为每个用户建立一个单独的用户文件目录，其中的表项登记了该用户建立的所有文件及其说明信息。主文件目录则记录系统中各个用户文件目录的情况，每个用户占一个表目，表目中包括用户名及相应用户目录所在的存储位置等。这样就形成了二级目录结构。

b. 优点：解决文件重名问题，并可以获得较高的查找速度。

c. 缺点：二级目录结构缺乏灵活性，特别是当用户需要在某些任务上进行合作和访问其他文件时会产生很多问题。

5. 树形目录结构

a. 基本概念：为了便于系统和用户更灵活、方便地组织管理和使用各类文件，**将二级目录的层次关系加以推广，便形成了多级目录结构**，又称为树形目录结构。树形目录结构中引入了以下概念：路径名、当前目录。

b. 优点：方便对文件进行分类，层次结构清晰，也能够更有效地进行文件的管理和保护。

c. 缺点：在树形目录中查找一个文件，需要按照路径名逐级访问中间节点，增加了磁盘访问次数，进而影响了查询速度。

6. 图形目录结构

a. 基本概念：树形目录结构便于实现文件分类，但是不便于实现文件共享，**为此在树形目录结构的基础上增加了一些指向同一节点的有向边，使整个目录成为一个有向无环图。**这就是图形目录结构，引入这种结构的目的是实现文件共享。

b. 优点：方便实现文件的共享。

c. 缺点：系统的管理变得复杂。

0.229 04038-文件共享

1. 共享动机

- a. 多用户操作系统中不同的用户间需要共享一些文件来共同完成任务；
- b. 网络上不同的计算机之间需要进行通信，需要远程文件系统的共享功能的支持。

2. 基于索引节点的共享方式（硬链接）

a. 实现方法：一个共享文件只有一个索引节点，如果不同文件名的目录项需要共享该文件，只需目录项中的指针都指向该索引节点即可。在索引节点中再增加一个计数值来统计指向该索引节点的目录项的个数，这样就需要删除该文件时可以判断计数值，只有计数值为 1 时才删除该索引节点，若计数值大于 1，则把计数值减 1 即可。

- b. 优点：能够实现文件的异名共享。
- c. 缺点：当文件被多个用户共享时，文件拥有者不能删除文件。

3. 利用符号链实现文件共享（软链接）

- a. 实现方法：新建一个链接文件，文件里面存储需要共享文件的路径名。
- b. 优点：解决了基于索引节点共享方法中文件拥有者不能删除共享文件的问题。
- c. 缺点：当其他用户要访问共享文件时，需要逐层查找目录，开销较大。

0.230 04039-数据通路的功能和基本结构

1. 数据通路的功能

建立数据通路的功能就是实现 CPU 内部的运算器和寄存器，以及寄存器之间的数据交换。

2. 数据通路的基本结构

数据通路的基本结构主要有以下两种方式：

1) **CPU 内部总线方式**。将所有寄存器的输入端和输出端都连接到一条或多条公共的通路上，这种结构比较简单，但是数据传输存在较多的冲突现象，性能较低。如果连接各部件的总线只有一条，则称为单总线结构。如果 CPU 中有两条或多条总线，则构成双总线结构和多总线结构。在双总线或多总线结构中，数据的传递可以同时进行。

2) **专用数据通路方式**。根据指令执行过程中的数据和地址的流动安排连接线路，避免使用共享的总线，性能比较高，但硬件量较大。

0.231 04042-目录的实现

1. 线性表

最为简单的目录实现方法是使用存储文件名和数据块指针的线性表（数组、链表等）。创建新文件时，必须首先搜索目录表以确定没有同名的文件存在，接着在目录表后增加一个目录项。若要删除文件，根据给定的文件名搜索目录表，接着释放分配给它的空间。采用链表结构可以减少删除文件的时间，其优点在于实现简单，不过由于线性表需要采用顺序方法查找特定的项，故运行比较费时。

2. 散列表

散列表根据文件名得到一个值，并返回一个指向线性表中元素的指针。这种方法大大缩短了查找目录的时间，插入和删除也比较简单，不过需要一些措施来避免冲突（两个不同名文件的散列函数值相同）。这种方法的特点是散列表长度固定以及散列函数对表长的依赖性。

0.232 04043-文件的实现

1. 文件的实现

a. 基本概念：文件的实现主要是指文件在存储器上的实现，即文件物理结构的实现，包括外存分配方式与文件存储空间的管理。

2. 外存分配方式（连续分配）

a. 基本概念：把逻辑文件中的记录顺序地存储到相邻的物理盘块中，这样所形成的文件结构称为顺序文件结构，此时的物理文件称为顺序文件。

b. 优点：查找速度比其他方法快（只需要起始块号和文件大小），目录中关于文件物理存储位置的信息也比较简单。

c. 缺点：容易产生碎片，需要定期进行存储空间的紧缩。这种分配方法不适合文件随时间动态增长和减少的情况，也不适合用户事先不知道文件大小的情况。

3. 外存分配方式（链接分配）

a. 隐式链接：用于链接物理块的指针隐式地放在每个物理块中。

b. 显式链接：用于链接物理块的指针显式存放在内存的一张链接表中。

c. 优点：简单（只需起始位置），文件创建与增长容易实现。

d. 缺点：不能随机访问盘块，链接指针会占用一些存储空间，而且存在可靠性问题。

4. 外存分配方式（索引分配）

a. 基本概念：系统为每个文件分配一个索引块，索引块中存放索引表，索引表中的每个表项对应分配给该文件的一个物理块。

b. 单级索引分配：单级索引分配方法就是将每个文件所对应的盘块号集中放在一起，为每个文件分配一个索引块（表），再把分配给该文件的所有盘块号都记录在该索引块中，因而该索引块就是一个包含多个盘块号的数组。

c. 两级索引分配：当文件较大，一个索引块放不下文件的块序列时，可以对索引块再建立索引，这样构成二级索引。

d. 混合索引分配：所谓混合索引分配，是指将多种索引分配方式相结合而形成的一种分配方式。

5. 文件存储空间管理（空闲文件表）

a. 基本概念：为所有空闲文件单独建立一个目录，每个空闲文件在这个目录中占一个表目。表目的内容包括第一个空闲块号、物理块号和空闲块数目。

b. 优点：当文件存储空间中只有少量空闲文件时，这种方法有较好的效果。

c. 缺点：如果存储空间中有大量的小空闲文件，则空闲文件目录将变得很大，其效率将大为降低。这种管理技术仅适用于连续文件。

6. 文件存储空间管理（空闲块链表）

a. 基本概念：将文件存储设备上的所有空闲块链接在一起，形成一条空闲块链，并设置一个头指针指向空闲块链的第一个物理块。也可以将链表中的空闲盘块改为空闲盘区（每个空闲盘区包含若干个连续的空闲盘块），这样的链称为空闲盘区链。

7. 文件存储空间管理（位示图法）

a. 基本概念：为文件存储器建立一张位示图（称它是图，其实就是一连串的二进制位），以反映整个存储空间的分配情况。在位示图中，每一个二进制位都对应一个物理块，若某位为 1，表示对应的物理块已分配；若为 0，表示对应的物理块空闲。

0.233 04044-控制单元的功能

1. 微操作命令的分析

前面已经详细讲解过取指周期、间址周期的微操作命令，下面详细讲解各指令的执行周期和中断周期。

(1) 执行周期（讲解一个典型指令）

加法指令。加法有太多的不确定性，如操作数可以在寄存器、累加器、主存等，这些微操作命令都是不一样的，以下假设一个前提。

前提：假设一个操作数在累加器，一个操作数在主存 A 单元，并且运算结果送至累加器，请写出具体的微操作指令。

思路：首先要从主存中取出数，然后再和累加器 ACC 的内容相加送入 ACC 即可。微程序序列如下：

Ad (IR) → MAR

//将指令的地址码送入主存地址寄存器

1→R

//启动存储器读

M (MAR) → MDR

//将 MAR 所指的主存单元中的内容（操作数）经数据总线读到 MDR

(2) 中断周期

前提：现假设程序断点保存至主存的“0”号单元，且采用硬件向量法寻找入口地址。中断周期的微指令序列如下：

0→MAR

//将主存“0”号单元的地址送入主存地址寄存器

1→W

//启动存储器写

(PC) → MDR

//将 PC 的内容（程序断点）送入主存数据寄存器

(MDR) → M (MAR)

//将主存数据寄存器的内容写入 MAR 所指示的主存单元

向量地址 → PC

//将向量地址形成部件的输出送至 PC

0→EINT

//关中断，将允许中断触发器清零

以上就是中断周期的全部微指令操作。如果断点不是存入主存，而是存入堆栈，那么微程序指令又是什么？很简单，只需将上述步骤 改为：

(SP) -1→SP，且 (SP) → MAR

//这里假设先修改指针，后存入数据

2. 控制单元的功能

输入 CU 的内容如下：

1) 指令寄存器。将指令的操作码送入 CU 进行译码。

2) 标志。有时候控制单元需要根据上条指令的结果来产生相应的控制信号。因为“标志”也是控制单元的输入信号。

3) 时钟。每个操作完成需要多少时间？每个操作之间的执行按照什么样的先后顺序？怎么去解决？自然想到时钟信号，通过时钟脉冲来控制。

4) 来自系统控制总线的控制信号。中断请求、DMA 请求等信号的输入。

输出 CU 的内容如下：

1) CPU 内的控制信号。主要用于 CPU 内寄存器之间的传送和控制 ALU 实现不同的操作。

2) 送至系统控制总线的信号。命令主存或者 I/O 读/写、中断响应等输出信号。

3. 控制方式

(1) 同步控制方式

任何一条指令或指令中任何一个微操作的执行，都由事先确定且有统一基准时标的时序信号所控制的方式叫做同步控制方式。

(2) 异步控制方式

异步控制方式不存在基准时标信号，没有固定的周期节拍和严格的时钟同步，执行每条指令和每个操作需要多少时间就占用多少时间。

0.234 04045-磁盘结构

1. 磁盘的物理结构

磁盘上的一系列同心圆称为磁道，磁道沿径向又分成大小相等的多个扇区，盘片上与盘片中心有一定距离的所有磁道组成了一个柱面。因此，**磁盘上的每个物理块可以用柱面号、磁头号和扇区号表示**。

2. 磁盘结构中的信息

- a. 引导控制块：通常为分区的第一块，如果该分区没有操作系统，则为空；
- b. 分区控制块：包括分区的详细信息，如分区的块数、块的大小、空闲块的数目和指针等；
- c. 目录结构：采用目录文件组织；
- d. 文件控制块：包括文件的信息，如文件名、拥有者、文件大小和数据块位置等。

3. 磁盘的访问时间 Ta

磁盘的访问时间 Ta 表示为：**访问时间 = 寻道时间 + 旋转延迟 + 传输时间**。

a. **寻道时间 Ts**：磁盘接收到读指令后，磁头从当前位置移动到目标磁道位置，所需时间为寻道时间 Ts。该时间是启动磁臂的时间 s 与磁头移动 n 条磁道所花费时间的总和，m 为每移动一个磁道所需时间，即 $Ts=m \times n + s$ ；

b. **旋转延迟 Tr**：旋转磁盘、定位数据所在的扇区所需的时间为旋转延迟 Tr。设磁盘的旋转速度为 r，则 $Tr=(1/r)/2=1/(2r)$ ；

c. **传输时间 Tt**：从磁盘上读取数据的时间为传输时间 Tt。传输时间取决于每次读写的字节数 b 和磁盘的旋转速度，即 $Tt=b/(rN)$ ，式中，r 为转速；N 为一个磁道上的字节数。

0.235 04046-调度算法

1. 先来先服务（FCFS）算法

FCFS 算法是一种最简单的磁盘调度算法。该算法按进程请求访问磁盘的先后次序进行调度。该算法的特点是合理、简单，但未对寻道进行优化。

2. 最短寻道时间优先 (SSTF) 算法

SSTF 算法选择与当前磁头所在磁道距离最近的请求作为下一次服务的对象。该算法的寻道性能比 FCFS 算法好，但不能保证平均寻道时间最短，并且可能会使某些进程的请求总被其他进程的请求抢占而长期得不到服务（这种现象称为“饥饿”）。

3. 扫描算法 (SCAN) 或电梯调度算法

SCAN 算法在磁头当前移动方向上选择与当前磁头所在磁道距离最近的请求作为下一次服务的对象。由于这种算法中磁头移动的规律颇似电梯的运行，故也称为电梯调度算法。SCAN 算法具有较好的寻道性能，又避免了“饥饿”现象，但其对两端磁道请求比较不公平（通常两端请求都是最后得到服务）。

4. 循环扫描 (CSCAN) 算法

CSCAN 算法是对 SCAN 算法的改良，它规定磁头单向移动，例如，自里向外移动，当磁头移到最外磁道时立即返回到最里磁道，如此循环进行扫描。该算法消除了对两端磁道请求的不公平。

0.236 04047-磁盘管理

1. 磁盘格式化

一个新的磁盘只是一个含有磁性记录材料的空白盘。在磁盘能存储数据前，它必须分成扇区以便磁盘控制器能进行读和写操作，这个过程称为低级格式化。低级格式化为磁盘的每个扇区采用独特的数据结构。每个扇区的数据结构通常由头、数据区域（通常为 512B）和尾部组成。头部和尾部包含了一些磁盘控制器所使用的信息。为了使用磁盘存储文件，操作系统还需要将自己的数据结构记录在磁盘上：

a. 将磁盘分为由一个或多个柱面组成的分区（就是常见的 C 盘、D 盘等分区）。

b. 对物理分区进行逻辑格式化（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和已经分配的空间以及一个初始为空的目录。

2. 引导块

计算机启动时需要运行一个初始化程序（自举程序），它初始化 CPU、寄存器、设备控制器和内存等，接着启动操作系统。为此，该自举程序应找到磁盘上的操作系统内核，装入内存，并转到初始地址，从而开始操作系统的运行。

自举程序通常保存在 ROM 中，为了避免改变自举代码需要改变 ROM 硬件的问题，只在 ROM 中保留很小的自举装入程序，将完整功能的自举程序保存在磁盘的启动块上，启动块位于磁盘的固定位。拥有启动分区的磁盘称为启动磁盘或系统磁盘。

3. 坏块

由于硬件有移动部件且容错能力差，因此容易导致一个或多个扇区损坏。根据所使用的磁盘和控制器，对这些块有多种处理方式。

- a. 简单磁盘：坏扇区可手工处理。
- b. 复杂的磁盘：其控制器维护一个磁盘坏块链表。该链表在出厂前进行低级格式化时就初始化了，并在磁盘的整个使用过程中不断更新。低级格式化将一些块保留作为备用，对操作系统透明。控制器可以用备用块来逻辑地替代坏块，这种方案称为扇区备用。

0.237 04048-I-O 设备的分类与 I-O 管理的任务

1. I/O 设备的分类

- a. 按设备的使用特性分类：存储设备和 I/O 设备；

- b. 按信息交换单位分类：字符设备和块设备；
- c. 按传输速率分类：低速设备、中速设备和高速设备；
- d. 按设备的共享属性分类：独占设备、共享设备和虚拟设备。

2. I/O 管理的任务和功能

- a. 设备分配：按照设备类型和相应的分配算法决定将 I/O 设备分配给哪一个进程。
- b. 设备处理：设备处理程序用以实现 CPU 和设备控制器之间的通信。
- c. 缓冲管理：设置缓冲区的目的是为了缓和 CPU 与 I/O 设备速度不匹配的矛盾。
- d. 设备独立性：设备独立性又称为设备无关性，是指应用程序独立于物理设备。

0.238 04049-I-O 控制方式

1. 设备控制器的组成

设备控制器由设备控制器与处理器的接口、设备控制器与设备的接口及 I/O 逻辑三部分组成。

2. 设备控制器的功能

- a. 接收和识别来自 CPU 的各种指令；
- b. 实现 CPU 与设备控制器、设备控制器与设备之间的数据交换；
- c. 记录设备的状态供 CPU 查询；
- d. 识别所控制的每个设备的地址。

3. 程序直接控制方式

- a. 基本概念：早期的计算机系统中，没有中断系统，所以 CPU 和 I/O 设备进行通信、传输数据时，由于 CPU 的速度远远快于 I/O 设备，因此 CPU 需要不断地测试 I/O 设备，这种控制方式又称为轮询或忙等。
- b. 优点：程序直接控制方式的工作过程非常简单。
- c. 缺点：CPU 的利用率相当低。

4. 中断控制方式

- a. 基本概念：CPU 与外设并行工作，CPU 不用等待外设，当外设准备好数据后通过中断通知 CPU，CPU 停下当前的工作来处理外设的数据。
- b. 优点：与程序直接控制方式相比，有了中断的硬件支持后，CPU 和 I/O 设备间可以并行工作了，CPU 只需收到中断后处理即可，大大提高了 CPU 利用率。
- c. 缺点：但这种控制方式仍然存在一些问题，例如每台设备每输入输出一个数据，都要求中断 CPU，这样在一次数据传送过程中的中断次数过多，从而消耗了大量 CPU 时间。
- d. 中断处理程序的处理过程如下：

- 1) 唤醒被阻塞的驱动（程序）进程
- 2) 保护被中断进程的 CPU 环境
- 3) 转入想要的设备处理程序
- 4) 中断处理
- 5) 恢复被中断进程的现场

5. DMA 控制方式

- a. 基本概念：在外设和内存之间开辟直接的数据交换通路。在 DMA 控制方式中，设备控制器具有更强的功能，在其控制下，设备和内存之间可以成批地进行数据交换，而不用 CPU 干预。
- b. 特点：数据传输的基本单位是数据块，而且数据是单向传输，从设备到内存或者相反。
- c. 优点：DMA 控制方式下，设备和 CPU 可以并行工作，同时设备与内存的数据交换速度加快，并且不需要 CPU 干预。

d. 缺点：DMA 控制方式仍然存在一定局限性，如数据传送的方向、存放输入数据的内存起始地址及传送数据的长度等都由 CPU 控制，并且每台设备都需要一个 DMA 控制器，当设备增加时，多个 DMA 控制器的使用也不经济。

6. 通道控制方式

a. 基本概念：是一种以内存为中心，实现设备与内存直接交换数据的控制方式。与 DMA 控制方式相比，通道所需要的 CPU 干预更少，而且可以做到一个通道控制多台设备，从而更进一步减轻了 CPU 负担。**通道本质上是一个简单的处理器**，通道的指令系统比较简单，一般只有数据传送指令、设备控制指令等。

b. 优点：解决了 I/O 操作的独立性和各部件工作的并行性。通道把中央处理器从繁琐的输入输出操作中解放出来。采用通道技术后，不仅能实现 CPU 和通道的并行操作，而且通道与通道之间也能实现并行操作，各通道上的外设也能实现并行操作，从而可达到提高整个系统效率的根本目的。

c. 缺点：由于需要更多硬件（通道处理器），因此成本较高。通常应用于大型数据交互的场合。

d. I/O 通道与一般处理器的区别：I/O 通道的指令类型单一，其所能执行的命令主要局限于与 I/O 操作有关的指令；通道没有自己的内存，通道所执行的通道程序放在主机的内存中，也就是说通道是与 CPU 共享内存的。

0.239 04050-I-O 软件层次结构

1. 层次结构概述

下图总结了 I/O 软件的层次结构及每一层的主要功能。



2. 中断处理程序

a. **基本概念**：控制输入输出设备和内存与 CPU 之间的数据传送的主要方式。当完成 I/O 操作时，设备便向 CPU 发送一个中断信号，CPU 响应中断后便转入中断处理程序。

b. **中断过程**：唤醒被阻塞的驱动程序进程；保护被中断进程的 CPU 环境；分析中断原因；进行中断处理；恢复被中断进程的现场。

3. 设备驱动程序

a. **基本概念**：接受来自上层的设备独立性软件的抽象请求，将这些请求转换成设备控制器可以接受的具体命令，再将这些命令发送给设备控制器，并监督这些命令正确执行。**设备驱动程序是操作系统中唯一知道设备控制器中设置了多少个寄存器以及这些寄存器有何用途的程序**。

b. **设备驱动程序的处理过程**：将抽象要求转换为具体要求；检查 I/O 请求的合法性；读出和检查设备的状态；传送必要参数；设置工作方式；启动 I/O 设备。

4. 设备独立性软件

a. **基本概念**：实现一般设备都需要的 I/O 功能，并向用户空间软件提供一个统一的接口。设备独立性软件通常应实现的功能包括设备驱动程序的统一接口，设备命名，设备保护，提供与设备无关的逻辑块，缓冲、存储设备的块分配，独占设备的分配和释放，出错处理。

5. 用户层软件

一般来说，大部分 I/O 软件都包含在操作系统中，但是仍有一部分是由与用户程序链接在一起的库函数，甚至运行于内核之外的程序构成的。通常的系统调用包括 I/O 系统调用，是由库函数实现的。SPOOLing 系统也处于这一层上。

0.240 04053-设备分配与回收

1. 设备管理中的数据结构

- a. DCT (设备控制表): 用于记录设备的特性以及 I/O 控制器的连接情况;
- b. COCT (控制器控制表): 用于反映设备控制器的使用状态以及和通道的连接情况等;
- c. CHCT (通道控制表): 用于反映通道的状态等;
- d. SDT (系统设备表): 整个系统只有一张系统设备表, 它记录了已连接到系统中的所有物理设备的情况, 每个物理设备占用一个表目。

2. 设备分配策略

a. 设备的使用性质

在分配设备时, 应考虑设备的使用性质, 设备分配可以采用以下 3 种不同的方式。

- 1) 独享设备: 又称为独占设备, 应采用独享分配方式, 即在将一个设备分配给某进程后便一直由其独占, 直至该进程完成或释放设备后, 系统才能再将设备分配给其他进程。
- 2) 共享分配: 对于共享设备, 系统可将其同时分配给多个进程使用。
- 3) 虚拟分配: 当进程申请独享设备时, 系统给它分配共享设备上的一部分存储空间; 当进程要与设备交换信息时, 系统就把要交换的信息存放在这部分存储空间中; 在适当的时候, 将设备上的信息传输到存储空间中或将存储空间中的信息传送到设备。

b. 设备分配算法

- 1) 先来先服务: 根据请求的时间顺序构成队列, 总是把设备优先分配给队首进程;
- 2) 优先级高者优先: 按照优先级的高低进行设备分配, 若优先级相同, 则按照先来先服务算法进行分配。

c. 设备分配的安全性

在设备分配中应保证不发生进程的死锁。在进行设备分配时, 可采用静态分配方式和动态分配方式。

- 1) 静态分配: 在用户作业开始执行之前, 由系统一次性分配该作业所需要的所有设备、设备控制器和通道, 一旦分配, 则一直占用, 直至作业撤销。静态分配虽然不会出现死锁, 但设备使用效率较低。
- 2) 动态分配: 在进程执行过程中根据执行需要进行设备分配。在进程需要设备时申请, 用完后立即释放。动态分配方式有利于提高设备的利用率, 但如果分配算法不当, 则可能造成死锁。

3. 设备独立性

- a. 基本概念: 应用程序独立于具体使用的物理设备, 它可以提高设备分配的灵活性和设备的利用率。
- b. 实现方法: 引入逻辑设备和物理设备这两个概念, 而系统中需要设置一张逻辑设备表, 其中每个表项中都有逻辑设备名、物理设备名和设备驱动程序入口地址。在设备驱动程序之上设置一层设备独立性软件, 用来执行所有 I/O 设备的公用操作, 并向用户层软件提供统一接口。
- c. 带来的好处: 设备分配时的灵活性、易于实现 I/O 重定向。

4. 设备分配程序

a. 单通路 I/O 系统的设备分配

当某一进程提出 I/O 请求后, 系统的设备分配程序可以按照下述步骤进行设备分配: 分配设备 → 分配设备控制器 → 分配通道。在分配时如遇到对应设备忙的情况, 则将进程插入到对应的等待队列中。

b. 多通路 I/O 系统的设备分配

步骤如下:

- 1) 根据设备类型, 检索系统设备控制表, 找到第一个空闲设备, 并检测分配的安全性, 如安全则分配, 反之插入该类设备的等待队列。
- 2) 设备分配后, 检索设备控制器控制表, 找到第一个与已分配设备相连的空闲设备控制器, 若无空闲, 则返回步骤 1) 查找下一个空闲设备。

3) 设备控制器分配后，同样查找与其相连的通道，找到第一个空闲通道，若无则返回步骤 2) 查找下一个空闲设备控制器。若有空闲通道，则此次设备分配成功，将相应的设备、设备控制器和通道分配给进程，并启动 I/O 设备，开始信息传输。

5. 设备的回收

当进程使用完对应的 I/O 设备后，释放所占有设备、设备控制器及通道，系统进行回收，修改对应的数据结构，以便下次分配时使用。

0.241 04054-假脱机技术

1. 假脱机技术

系统中独占设备的数量有限，往往不能满足系统中多个进程的需要，从而成为系统的“瓶颈”，使许多进程因等待而阻塞。另一方面，分配到独占设备的进程，在整个运行期间往往占有但不经常使用设备，使设备利用率偏低。为克服这种缺点，**人们通过共享设备来虚拟独占设备，将独占设备改造成共享设备，从而提高了设备利用率和系统的效率**，该技术称为假脱机 (SPOOLing) 技术。

2. SPOOLing 技术

a. 基本概念

是低速输入输出设备与主机交换的一种技术，其核心思想是以联机的方式得到脱机的效果。低速设备经通道和设在主机内存的缓冲存储器与高速设备相连，该高速设备通常是辅存。为了存放从低速设备上输入的信息，在内存中形成缓冲区，在高速设备上形成输出井和输入井，传递时信息从低速设备传入缓冲区，再传到高速设备的输入井，再从高速设备的输出井传到缓冲区，再传到低速设备。

b. SPOOLing 系统的组成

1) 输入井和输出井：输入井和输出井是在磁盘上开辟出来的两个存储区域。**输入井模拟脱机输入时的磁盘，用于收容 I/O 设备输入的数据。输出井模拟脱机输出时的磁盘，用于收容用户程序的输出数据。**

2) 输入缓冲区和输出缓冲区：输入缓冲区和输出缓冲区是在内存中开辟的两个缓冲区。**输入缓冲区用于暂存由输入设备传递过来的数据，然后再传送到输入井。输出缓冲区用于暂存从输出井传递过来的数据，然后再传送到输出设备。**

3) 输入进程和输出进程：**输入进程模拟脱机输入时的外围控制机**，将用户要求的数据从输入设备通过输入缓冲区再传递到输出井。当需要输入数据时，CPU 直接将数据从输入井读入内存。**输出进程模拟脱机输出时的外围控制机**，把用户要求输出的数据先从内存送到输出井，等输出设备空闲时，再将输出井中的数据经过输出缓冲区送到输出设备上。

c. SPOOLing 技术的特点

- 1) 提高了 I/O 速度；
- 2) 设备并没有分配给任何进程；
- 3) 实现了虚拟设备功能；
- 4) SPOOLing 除了是一种速度匹配技术外，也是一种虚拟设备技术。

0.242 04055-控制单元的设计

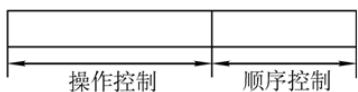
实现控制单元 (CU) 的方式有两类 (2009 年第 19 题已经考过)：

1) **组合逻辑控制 (或称为硬布线逻辑控制)**：由基本的门电路组合实现。这种方式实现的控制器的处理速度快，但电路庞杂，制造周期长，不灵活，可维护性差。

2) **微程序控制**：仿照程序设计的方法编制每个机器指令对应的微程序，每个微程序由若干条微指令构成，各微指令包含若干条微命令。所有指令对应的微程序放在只读存储器中。当执行到某条指令时，取出对应微程序中的各条微指令，译码产生对应的微命令，送到机器相应的地方，控制其动作。

1. 微指令的基本格式

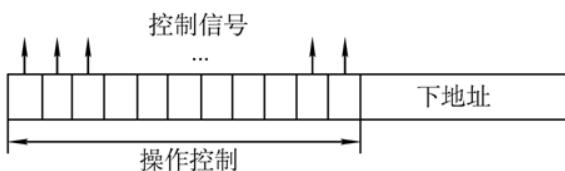
微指令的基本格式如下图所示，共分两个字段，一个为操作控制字段，该字段发出各种控制信号；另一个为顺序控制字段，该字段可指出下条微指令的地址（简称下地址），以控制微指令序列的执行，这个其实类似于 PC。



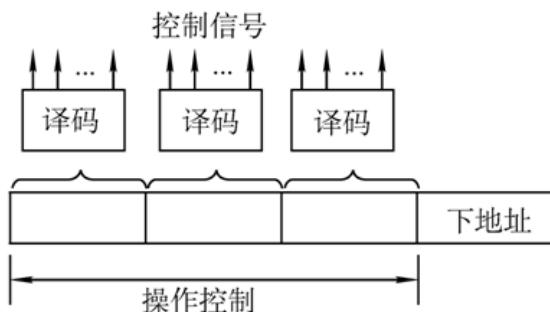
2. 微指令的编码方式

微指令的编码方式又称为微指令的控制方式。它是指如何对微指令的控制字段进行编码，以形成控制信号。

1) **直接编码（直接控制）方式**。在微指令的微命令字段中每一位都代表一个微命令。设计微指令时，选用或不选用某个微命令，只要将表示该微命令的对应位设置成 1 或 0 就可以了。因此，微命令的产生不需要译码，如下图所示。



2) **字段直接编码方式**。将微指令的微命令字段分成若干小字段，把互斥性微命令（在同一微指令周期中不能同时出现的微命令称为互斥性微命令）组合在同一字段中，把相容性微命令组合在不同的字段中。每个字段独立编码，每种编码代表一个微命令且各字段编码含义单独定义，与其他字段无关，这就是字段直接编码方式，如下图所示。



3) **字段间接编码方式**。一个字段的某些微命令需由另一个字段中的某些微命令来解释，由于不是靠字段直接译码发出的微命令，故称为字段间接编码，又称隐式编码。这种方式可进一步缩短微指令字长，但因削弱了微指令的并行控制能力，因此通常作为字段直接编码方式的一种辅助手段。

4) **混合编码方式**。混合编码方式由直接编码与字段（直接或间接）编码混合使用。

3. 微指令序列地址的形成

后续微指令的地址主要考查两种形成方式。

第一方式：后续微指令的地址可以由微指令的下地址字段直接给出，这种方式又称为断定方式（2014 年统考真题已经考查）。

第二方式：后续微指令的地址还可以根据机器指令的操作码形成。微地址形成部件实际是一个编码器，其输入为指令操作码。即将机器指令的操作码送入地址译码器进行译码，输出结果就是对该机器指令微程序的首地址。然后拿着首地址去微命令寄存器找到相应的微命令。

4. 微指令格式

微指令格式与微指令的编码方式有关，其通常分为**水平型微指令**和**垂直型微指令**两种。水平型微指令与垂直型微指令的比较见下表。

比赛项目	水平型微指令	垂直型微指令
并行性	好	不好
执行指令需要的微指令数	少	多
和机器指令的相似度	差别很大	相似
最后陈述	用较短的微程序结构换取较长的微指令结构	用较长的微程序结构换取较短的微指令结构

5. 微操作的节拍和安排设计步骤

每一条机器指令要完成的操作是固定的，因此不论是组合逻辑设计还是微程序设计，对应相同的 CPU 结构，两种控制单元的微操作命令和节拍安排是极其相似的。微程序控制单元在取指周期发出的微操作命令及节拍安排如下：

T0 (PC) → MAR, 1→R

T1 M (MAR) → MDR, (PC) +1→PC

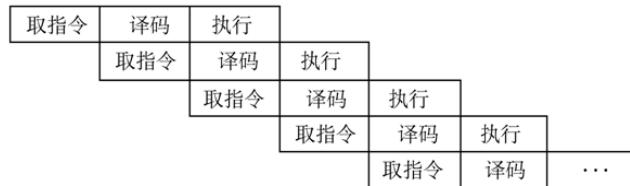
T2 (MDR) → IR, OP (IR) → 微地址形成部件

0.243 04056-指令流水线的基本概念

一条指令的执行需要经过 3 个阶段：取指令、译码、执行；每个阶段都要花费一个时钟周期，如果没有采用流水线技术，那么执行 N 条这样的指令就需要 $3N$ 个时钟周期，如下图所示。



当第 $N-2$ 条指令在执行的时候应该对第 $N-1$ 条指令进行译码，当第 $N-1$ 条指令在译码时，可以将第 N 条指令取出来，这样就缩短了每条指令的平均执行周期。这就是指令流水线的思想，如下图所示。



当使用指令流水线时，执行 N 条指令需要的时钟周期数为 $N+2$ 。当 N 较大时， $N+2$ 远远小于 $3N$ 。

0.244 04057-指令流水线的基本实现

要使得流水线具有良好的性能，必须使流水线畅通流动，不发生断流。但由于流水过程中会出现以下 3 种相关冲突，因此实现流水线的不断流是困难的。这 3 种相关是资源相关（结构相关）、数据相关和控制相关。

1. 资源相关（结构相关）

所谓资源相关是指多条指令进入流水线后在同一机器时钟周期使用了同一个功能部件所发生的冲突。假定一条指令流水线由 5 段组成，分别为取指令 (FI)，指令译码 (ID)、计算有效地址或执行 (EX)、访存取数 (MEM) 和结果写寄存器 (WB)，见下表。

时钟 指令	1	2	3	4	5	6	7	8
I ₁	FI	ID	EX	MEM	WB			
I ₂		FI	ID	EX	MEM	WB		
I ₃			FI	ID	EX	MEM	WB	
I ₄				FI	ID	EX	MEM	WB
I ₅					FI	ID	EX	MEM

在第 4 个时钟，第₃条的 MEM 段（访存取数）与第₄条的 FI 段（访存取指令）都要访问存储器。当数据和指令放在同一个存储器且只有一个访问口时，便发生两条指令争用存储器资源的相关冲突，有两个办法解决冲突：

方法 1：第 I₄ 条的 FI 段停顿一个时钟再启动。

方法 2：增加一个存储器，将指令和数据分别放在两个存储器中。

2. 数据相关（重中之重）

(1) 写后读相关

字面理解就是应该先写入再读取，而现在是没有写入就已经读了（即读了旧数据），出现错误。

(2) 读后写相关

字面理解就是应该先读取再写入，而现在是写入后再读取（本来应读取旧数据，现在却读到了新数据），出现错误。

(3) 写后写相关

字面上理解是前面一条指令先写数据，然后后面一条指令再写，而现在恰好相反了，导致数据错误。

3. 控制相关

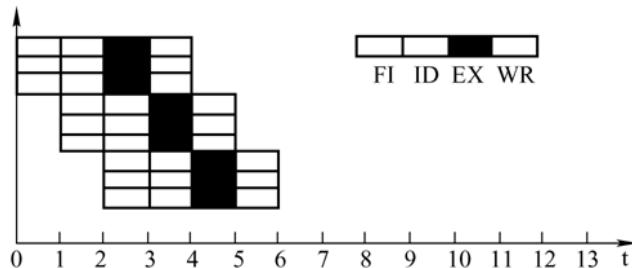
控制相关冲突是由转移指令引起的。当执行转移指令时，依据转移条件的产生结果，可能顺序执行下一条指令；也可能转移到新的目标地址取指令，从而使流水线发生断流。

解决方式：采用“猜测法”技术，机器先选定转移分支中的一个，按它取指并处理，条件码生成后，如果猜测正确，那么流水线继续进行下去；如果猜测错误，那么之前预取的指令失效。

0.245 04058-超标量和动态流水线的基本概念

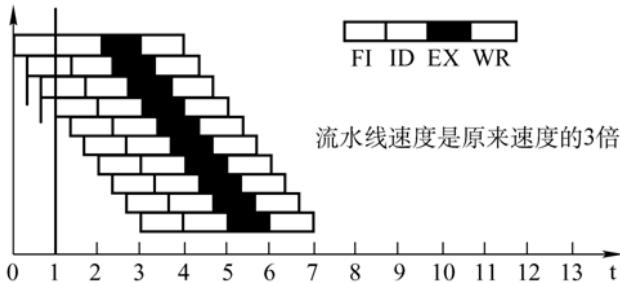
1. 超标量技术

超标量技术的每个时钟周期不像以前的普通指令流水线只能执行一条指令的某个阶段，而是可以并发执行多条独立指令，为此就需要配置多个功能部件，如下图所示。



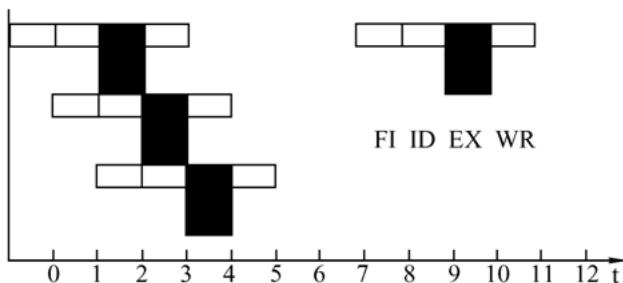
2. 超级流水线

典型的流水线是将每一条机器指令分成 5 步，即取指、译码、取操作数、执行和回写。在理想条件下，平均每个时钟周期可以完成一条指令。而所谓的超级流水线是将机器指令划分为更多级的操作，以减轻每一级的复杂程度。在流水线的每一步中，如果需要执行的逻辑操作少一些，那么每一步就可以在较短的时间内完成，如下图所示。



3. 超长指令字

由编译程序挖掘出指令潜在的并行性，将多条能并行操作的指令组合成一条具有多个操作码字段的超长指令字（可达几百位），为此需要采用多个处理部件，如下图所示。



4. 动态流水线

动态流水线就是多种运算可以同时进行，而静态流水线只能是一种运算进行完再进行下一种运算。

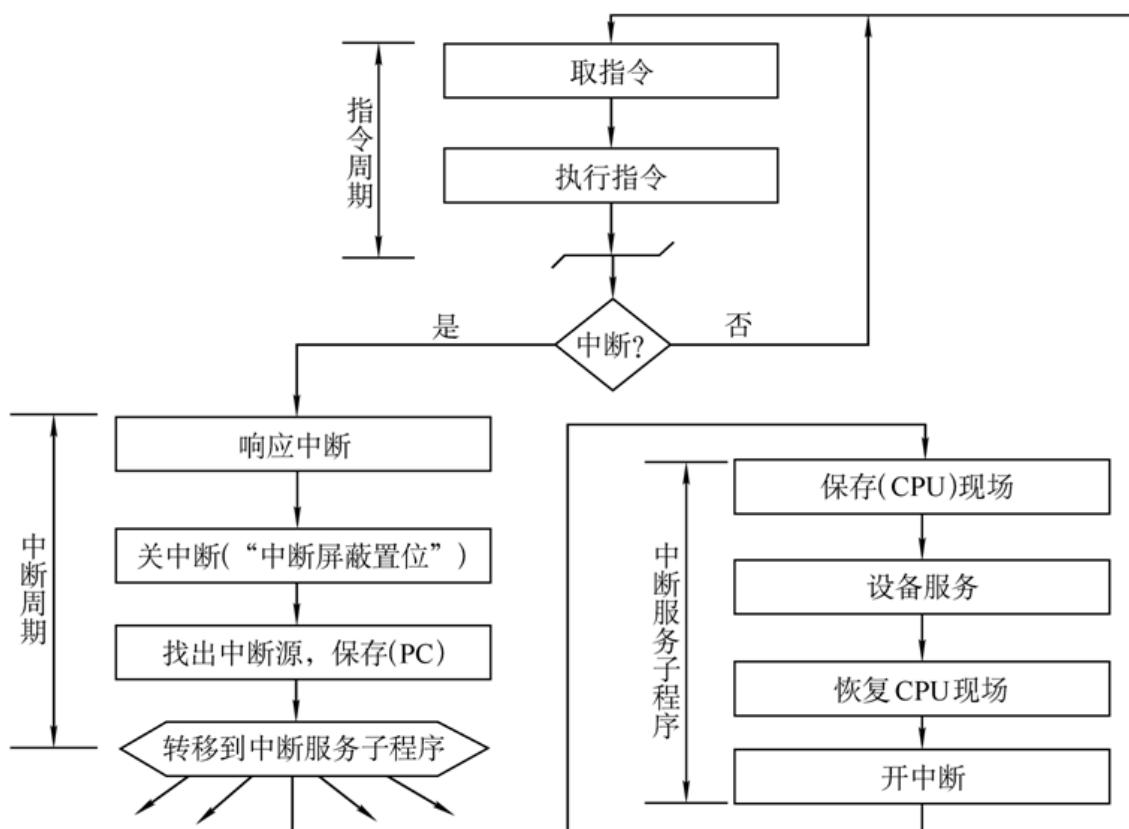
0.246 04059-中断系统

中断概念的出现是计算机系统结构设计中的一个重大变革。

第 7 章会讲到程序中断方式，某一外部设备的数据准备就绪后，它“主动”向 CPU 发出请求中断的信号，请求 CPU 暂时中断目前正在执行的程序而进行数据交换。

当 CPU 响应这个中断时，便暂停运行主程序，并自动转移到该设备的中断服务程序。

当中断服务程序结束以后，CPU 又回到原来的主程序。其实，计算机在运行过程中，除了会遇到 I/O 中断外，还有许多意外事件发生，如突然断电，机器硬件突然出现故障等。中断处理过程的详细流程，如下图所示。



图解：当 CPU 执行完一条现行指令时，如果外部设备向 CPU 发出中断请求，那么 CPU 在满足响应条件下，将发出中断响应信号，与此同时关闭中断（“中断屏蔽”触发器置“1”），表示 CPU 不再受理另外一个设备的中断。这时，CPU 将寻找中断请求源是哪一个设备，并保存 CPU 自己的程序计数器（PC）的内容。然后，它将转移到处理该中断源的中断服务程序。CPU 在保存现场信息、设备服务（如交换数据）以后，将恢复现场信息。这些动作完成以后，开放中断（“中断屏蔽”触发器置“0”），并返回到

原来被中断的主程序的下一条指令。

关于中断系统的讲解，详见书籍《计算机组成原理高分笔记》。

0.247 04060-总线的基本概念

背景知识：为什么要使用总线结构？**早期的计算机**大都采用分散连接方式，它是以存储器为中心的结构，由于现在的外部设备太多，如果使用分散连接方式，则随时增减外部设备不易实现，因此衍生出了**总线**的结构。

1) 总线的传输周期。

指 CPU 通过总线对存储器或 I/O 端口进行一次访问所需的时间，包括**总线申请阶段、寻址阶段、传输阶段和结束阶段**（在总线操作和定时里将详细讲解这 4 个阶段）。

2) 总线宽度。

总线实际上是由许多传输线或通路组成的，每条线可一位一位地传输二进制代码，一串二进制代码可在一段时间内逐一传输完成。若干条传输线可以同时传输若干位二进制代码，如 16 条传输线组成的总线可同时传输 16 位二进制代码。

3) 总线特性。

这个不重要，记住每个功能特性大致做什么就行，总结如下。

机械特性：尺寸、形状。

电气特性：传输方向和有效的电平范围。

功能特性：每根传输线的功能（是传送地址？还是传送数据？或是传送控制信号？）。

时间特性：信号的时序关系（哪根线在什么时间内有效）。

0.248 04061-总线的分类

总线按连接部件的不同划分的三大分类（属于考试重点）。

按连接部件不同可以分为**片内总线、系统总线、通信总线**等。

片内总线：顾名思义是指芯片内部的总线，如在 CPU 芯片内部，寄存器与寄存器之间，寄存器与算术逻辑单元 ALU 之间都是由片内总线连接的。

系统总线：连接五大部件之间的信息传输线。按系统总线传输信息的不同，又可分为 3 类：**数据总线、地址总线和控制总线**。

数据总线：用来传送各功能部件之间的数据信息，它是双向传输总线。一般数据总线有 8 位、16 位或 32 位。数据总线的位数称为数据总线宽度。如果数据总线的位数为 8 位，而指令字长为 16 位，那么 CPU 在取指令阶段必须进行两次访问。

地址总线：它是单向传输总线。地址总线主要用来指出数据总线上的源数据或目的数据在主存单元的地址或 I/O 设备的地址，地址总线上的代码用来指明 CPU 欲访问的存储单元或 I/O 设备的地址，由 CPU 给出。

控制总线：由于数据总线和地址总线是被所有部件共享的，如何使各部件能在不同时刻占有总线使用权，还需要依靠控制总线来完成，因此控制总线是用来发出各种控制信号的传输线。

通信总线：用于计算机系统之间或计算机系统与其他系统之间的通信。**通信总线按照数据传输方式可分为串行通信和并行通信**。前面讲解过，串行通信就是单车道的高速公路；并行通信就是多车道的高速公路。一般来说，并行通信适合近距离的数据传输，通常小于 30m；串行通信适宜于远距离传送。

0.249 04062-总线的组成及性能指标

(1) 总线的组成

总线通常由一组控制线、一组数据线和一组地址线组成。但是也有例外，某些总线没有单独的地址线，地址信息也通过数据线来传送，这种情况称为**数据线和地址线复用**。

(2) 性能指标

1. **总线宽度**。通常是指数据总线的根数，用位（bit）表示，如8位、16位（即8根、16根）等。
2. **总线带宽**。单位时间内总线上传输数据的位数（可以理解成某段高速公路在单位时间内所通过的车辆数）。
3. **总线复用**：在前面讲解过，即地址总线和数据总线共用一组线。
4. **信号线数**：地址总线、数据总线和控制总线3种总线数的总和。

0.250 04063-总线的结构

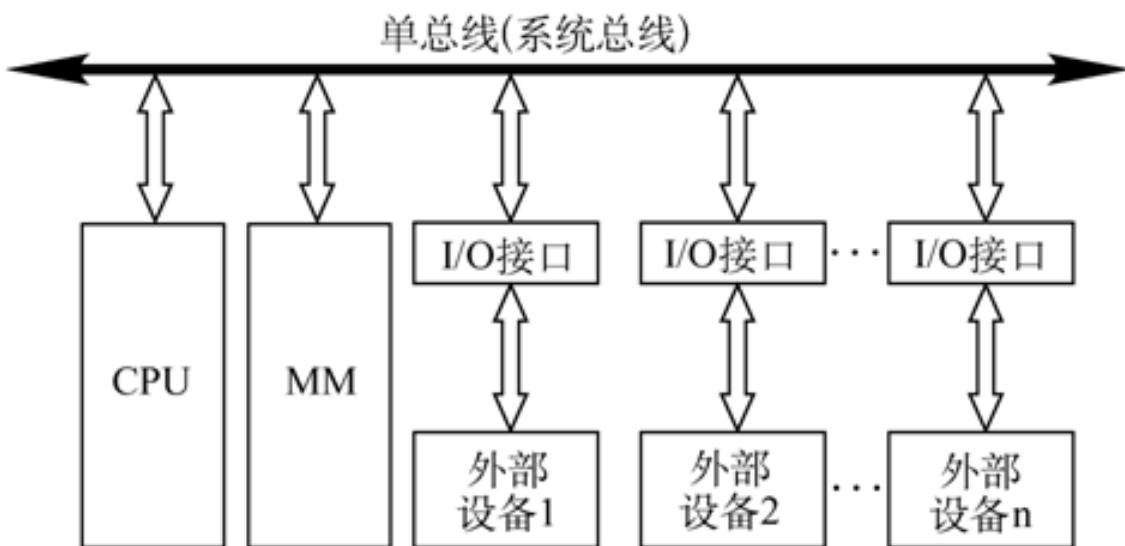
常用的总线结构通常可分为**单总线结构、双总线结构和三总线结构**。

(1) 单总线结构

单总线结构是将CPU、主存、I/O设备（通过I/O接口）都连接在一组总线上，允许I/O设备之间、I/O设备与CPU之间或I/O设备与主存之间直接交换信息，这种总线结构简单，很容易扩充外部设备，如下图所示。但是缺点也是显而易见的，所有的信息传送都通过这组共享总线，即不允许两个以上的部件在同一时刻向总线传输信息，这就必然会影响系统的工作效率。

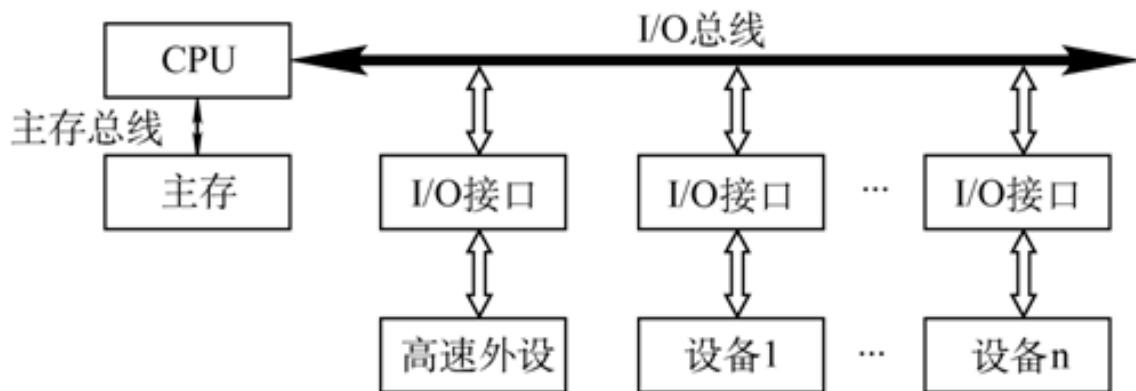
所以，这类总线多数被**小型计算机或微型计算机采用**。另外，使用单总线结构肯定不能解决CPU、主存、I/O设备之间传输速率不匹配问题，故必须使用多总线结构。

单总线的特点：由于I/O设备与主存共用一组地址线，因此主存和I/O设备是统一编址的，CPU就可以像访问内存一样访问外部设备。



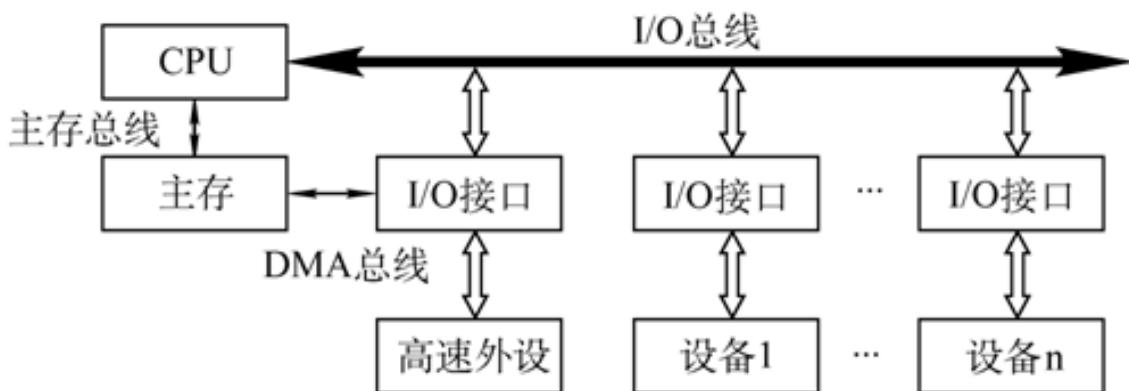
(2) 双总线结构

双总线的结构特点是将速度较低的I/O设备从单总线中分离出来，形成**主存总线与I/O总线分开的结构**，如下图所示。**但双总线结构也有问题**，比如现在有三间房，主存住在最左边，CPU住在中间，I/O住在最右边，每次主存和I/O设备交换信息都要经过CPU，那CPU的工作效率就自然会降低，所以引进了**三总线结构**。



(3) 三总线结构

由下图可以看出，与双总线结构相比，三总线结构增加了一条小路（DMA 总线），专门用于 I/O 高速设备与主存之间直接交换信息。DMA 将在第 7 章的相关知识点讲解，这里只需知道这种结构即可。

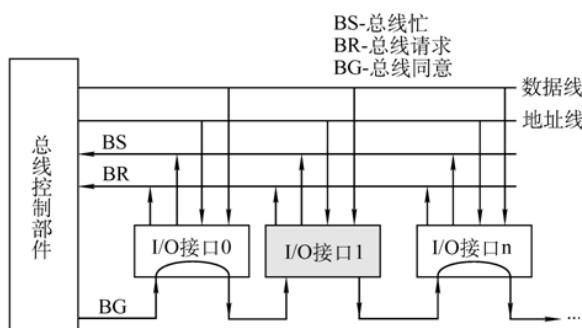


0.251 04064-集中仲裁方式

常见的集中控制优先权仲裁方式有以下 3 种：

1. 链式查询方式

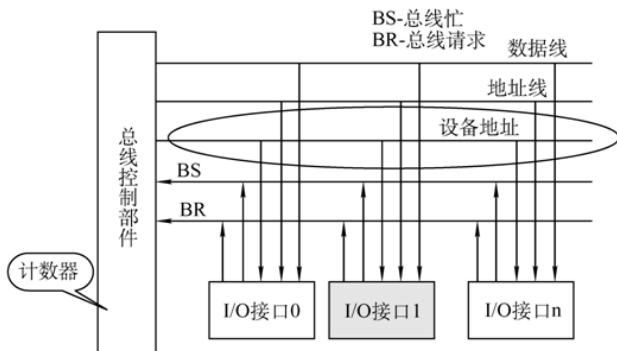
链式查询方式如下图所示。



链式查询优先级判别方式：离总线控制器越近的部件，其优先级越高；离总线控制器越远的部件，其优先级越低。

2. 计数器查询方式

计数器定时查询方式采用一个计数器控制总线使用权（计数器在总线控制部件里面）。相对于链式查询方式多了一组设备地址线，少了一根总线同意线。各个设备仍共用一条请求线，计数器查询方式如下图所示。



计数器查询优先级判别方式：当总线控制器收到总线请求信号判断总线不忙时，计数器开始计数，计数值通过一组地址线发向各个部件。当地址线上的计数值与请求使用总线设备的地址一致时，该设备获得总线控制权。同时，终止计数器的计数及查询工作。

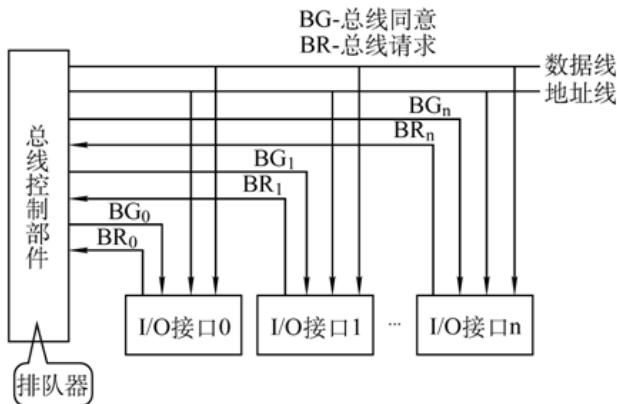
计数器有两种计数方式：

计数器每次判优都从“0”开始，此时一旦设备的优先顺序被固定后，设备的优先级就按 $0, 1, 2, \dots, n$ 的顺序降序排列，永远不能改变；

计数器也可以从上一次的终点开始计数，即是一种循环方法，此时所有设备使用总线的优先级相等。计数器的初值当然也可以由程序设置，故优先级顺序可以改变。

3. 独立请求方式

独立请求方式如下图所示。每一个设备均有一对总线请求信号 BR_i 和总线同意信号 BG_i 。



独立请求优先级判别方式：当总线上的部件需要使用总线时，经各自的总线请求线发送总线请求信号，在总线控制器中排队（总线控制部件中有一个排队器）。当总线控制器按一定的优先顺序决定批准某个部件的请求时，则给该部件发送总线响应信号，该部件接到此信号就获得了总线使用权，开始传输数据。

0.252 04065-分布仲裁方式

分布仲裁方式不需要中央仲裁器，每个主模块都有自己的仲裁号和仲裁器，多个仲裁器竞争使用总线。当它们有总线请求时，把它们各自唯一的仲裁号发送到共享的仲裁总线上，每个仲裁器将从仲裁总线上得到的仲裁号与自己的仲裁号进行比较。若仲裁总线上的号优先级高，则它的总线请求不予响应，并撤销它的仲裁号。最后，获胜者的仲裁号保留在仲裁总线上。

0.253 04066-总线周期的概念

通常将完成一次总线操作的时间称为**总线周期**，其可分为以下4个阶段：

1) **申请分配阶段**。由需要使用总线的主模块（或主设备）提出申请，经总线仲裁机器决定下一个传输周期的总线使用权授予某一申请者。

2) **寻址阶段**。取得了使用权的主模块通过总线发出本次要访问的从模块(或从设备)的地址及有关命令，启动参与本次传输的从模块。

3) **传送数据阶段**。主模块和从模块进行数据交换，数据由源模块发出，经数据总线流入目的模块。

4) **结束阶段**。主模块的有关信息均从系统总线上撤除，让出总线使用权。

如果该系统只有一个主模块，就无须申请、分配和撤除了，总线使用权始终归它所有。

总线通信控制主要解决通信双方如何获知传输开始和传输结束，以及通信双方如何协调和如何配合。通常，**总线通信方式分为4类**：同步通信(同步定时方式)、异步通信(异步定时方式)、半同步通信和分离式通信。

0.254 04067-同步定时方式

同步定时方式指系统采用一个统一的时钟信号来协调发送和接收双方的传送定时关系。**时钟信号通常由CPU的总线控制器部件发出**，然后送到总线上的所有部件。

同步通信的优点：传送速度快，具有较高的传输速率。

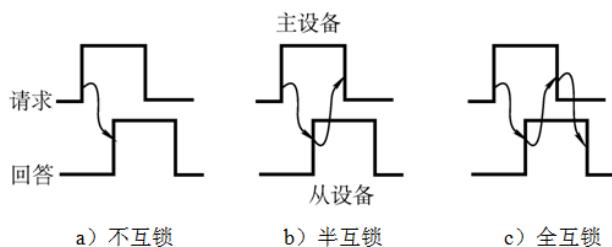
同步通信的缺点：同步通信必须按最慢的模块来设计公共时钟，当总线上的模块存取速度差别很大时，便会大大损失总线效率，并且不知道被访问的外部设备是否真正响应，故可靠性较低。

同步通信的使用范围：总线长度较短、总线所接部件的存取时间应该都比较接近。

0.255 04068-异步定时方式

异步通信没有公共的时钟标准，不要求所有部件严格地统一操作时间，而是采用**应答的方式(或称握手方式)**，即当主模块发出请求信号时，一直等待从模块反馈“响应”信号后才开始通信。当然，这就要求主、从模块之间增加两条应答线。

异步通信的应答方式又可分为**不互锁、半互锁和全互锁**3种类型，如下图所示。



1. 不互锁方式

特点：主模块的请求信号和从模块的回答信号没有互相的制约关系。

2. 半互锁方式

特点：主模块的请求信号和从模块的回答信号有简单的制约关系。

3. 全互锁方式

特点：主模块的请求信号和从模块的回答信号有完全的制约关系。

0.256 04069-总线标准

目前，流行的总线标准可以分为两大类：**系统总线标准和设备总线标准**。

系统总线标准包括以下4种：

- 1) ISA。
- 2) EISA。
- 3) VESA (VL-BUS)。局部总线标准。

4) PCI (最常用)。支持即插即用并且可对数据和地址进行奇偶校验。

设备总线标准包括以下 3 种：

- 1) IDE。用于处理器和磁盘驱动器之间。
- 2) AGP。专用于连接主存和图形存储器。
- 3) USB 接口。这个应该是考生最熟悉的了，可实现外部设备的快速连接。

0.257 04070-I-O 系统基本概念

1. I/O 系统的历史演变过程

(1) 早期阶段（程序查询方式）

在早期阶段，主存、CPU、I/O 系统都买不起房子，三个兄弟挤在一套房子里。CPU 住在主存和 I/O 系统的中间。每次 I/O 系统和主存交换信息的时候都需要经过 CPU 才能交换完成。可想而知，这样极其浪费时间。

(2) 接口模块和 DMA 阶段（中断方式和 DMA 方式）

在该阶段 I/O 设备通过接口模块与主机相连，且采用了总线连接的方式。虽然这仅仅是简单的一句话，却基本解决了以上的两个问题。

(3) 具有通道结构的阶段

通道是用来负责管理 I/O 设备以及实现主存与 I/O 设备之间交换信息的部件，可以视为一种具有特殊功能的处理器。通道具有专用的通道指令，能独立地执行用通道指令所编写的输入/输出程序，但不是一个完全独立的处理器。它依据 CPU 的 I/O 指令进行启动、停止或改变工作状态，是从属于 CPU 的一个专用处理器。

(4) 具有 I/O 处理器的阶段

该部分不属于计算机组成原理中的讲解内容，考研基本不会涉及。

2. I/O 系统的基本概念

I/O 系统主要由两部分组成：I/O 软件和 I/O 硬件。

(1) I/O 软件

对于采用接口模块方式，要使得 I/O 设备与主机协调工作，必须靠 I/O 指令来完成。对于采用通道管理方式，不仅需要 I/O 指令，还需要通道指令。

(2) I/O 硬件

输入/输出系统的硬件组成是多种多样的，在带接口的 I/O 系统中，I/O 硬件包括接口模块和 I/O 设备两大部分；在具有通道或 I/O 处理器的 I/O 系统中，I/O 硬件包括通道（或称为处理器）、设备控制器和 I/O 设备等。

0.258 04071-I-O 设备分类

I/O 设备根据其特点可以分为以下 3 类：

输入设备：能将人们熟悉的信息形式变换成计算机能接收并识别的信息形式。

输出设备：能将计算机运算结果的二进制信息转换成人类或者其他设备可以接收和识别的信息形式。

输入/输出兼用设备：既可以作为输入设备，也可以作为输出设备，例如硬盘，既可以存数据，也可以取数据。

0.259 04072-输入设备

1. 键盘

键盘是应用最普遍的输入设备，可以通过键盘上的各个键，按某种规范向主机输入各种信息，如汉字、字母、数字、特殊符号等。

键盘输入信息分为以下 3 个步骤：

- 1) 按下一个键。
- 2) 查出按了哪个键。
- 3) 将此键翻译成 ASCII 码传给计算机。

2. 鼠标

早期的鼠标底座装有一个金属球，在光滑的表面上摩擦，使金属球转动，球与 4 个方向的电位器接触，就可以测量出上、下、左、右 4 个方向的相对位移量。

0.260 04073-输出设备

一、显示器

1. 显示器的分类

按显示设备所用的显示器件分类：阴极射线管（CRT）显示器（重点）、液晶显示器（LCD）、等离子显示器。

按所显示的信息内容分类：字符显示器、图形显示器、图像显示器。

2. 阴极射线管（CRT）显示器

(1) CRT 显示器的分类

CRT 显示器按扫描方式的不同，可分为光栅扫描和随机扫描；按分辨率的不同，又可分为高分辨率和低分辨率。

(2) 分辨率和灰度级

分辨率指显示器所能表示的像素个数，像素越密，分辨率越高，图像越清晰。

灰度级指黑白显示器中所显示的像素点的亮暗差别，在彩色显示器中则表现为颜色的不同，灰度级越高，图像层次越清楚逼真。

(3) 刷新和刷新存储器

CRT 发光是由电子束打在荧光粉上引起的，电子束扫过之后，其发光亮度只能维持几十毫秒便会消失。为了使人眼能看到稳定的图像显示，必须使电子束不断地重复扫描整个屏幕，这个过程称为刷新。按人的视觉生理，刷新频率大于 30 次/s 时才不会感到闪烁。

为了不断提高刷新图像的信号，必须把图像信息存储在刷新存储器（也称为视频存储器）。其存储容量由图像分辨率和灰度级决定，分辨率越高，灰度级越高，刷新存储器容量越大。假如分辨率为 1024×1024 像素，256 级灰度的图像（需要 8 位二进制数来表示）储存容量为 $1024 \times 1024 \times \log_2 256 = 1MB$ 。

二、打印机

打印输出是计算机最基本的输出形式，与显示器输出相比，打印输出可产生永久性记录，因此打印设备又被称为硬拷贝设备。

0.261 04074-I-O 接口基础知识

为什么要设置接口？原因分析如下。

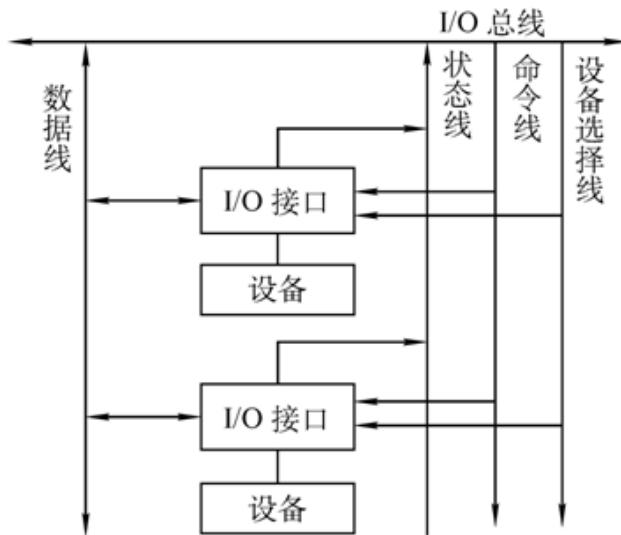
- 1) 前面讲过，某些 I/O 设备速度较慢，与 CPU 速度相差可能很大，通过接口可以实现数据缓冲。
- 2) 一台机器通常配有多台 I/O 设备，它们各自有其设备编号，通过接口可以实现 I/O 设备的选择。

3) 某些设备是一位位地串行传送数据，而 CPU 一般为并行传送，通过接口可实现数据串—并格式的转换。

0.262 04075-I-O 接口的功能和基本结构

1. I/O 接口与 I/O 设备的关系

每一台 I/O 设备都是通过 I/O 接口连接到系统总线上的，并且此总线包括数据线、设备选择线、命令线和状态线，如下图所示。



数据线（双向）：用做 I/O 设备与主机之间传送数据。

设备选择线（单向）：用来传送设备码。

命令线（单向）：用来传输 CPU 向设备发出的各种命令信号，如启动、停止、读、写等信号。

状态线（单向）：将 I/O 设备的状态向主机报告的信号线。

2. I/O 接口的功能

(1) 选择设备功能

由于 I/O 总线与所有设备的接口电路相连，但 CPU 究竟选择哪台设备，还得通过设备选择线上的设备码来确定，该设备码将送至所有设备的接口，因此当设备选择线上的设备码与本设备码相符合时，应发出设备选择信号 SEL，这种功能可通过接口内的设备选择电路来实现。

(2) 传送命令功能

当 CPU 向 I/O 设备发出命令时，要求 I/O 设备能作出响应，如果 I/O 接口不具备传送命令信息的功能，那么设备将无法响应，故通常在 I/O 接口中设有存放命令的命令寄存器和命令译码器。

(3) 传送数据功能

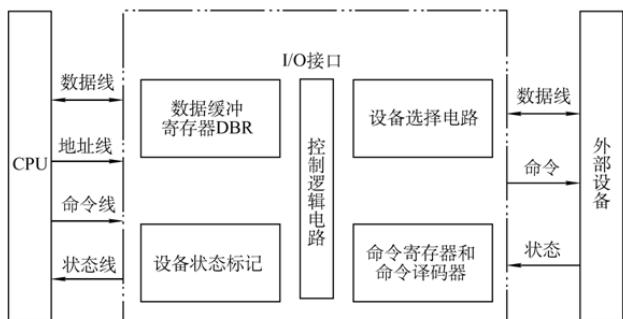
由于 I/O 接口处于主机和 I/O 设备之间，因此主机和 I/O 设备之间传送数据必须通过接口才能实现。这就要求接口中必须具有数据通路，完成数据传送。

(4) 反映 I/O 设备的工作状态

并不是数据发过来就能接收的，因为可能此时还没有准备好，所以接口内必须设置一些反映设备工作状态的触发器。

3. I/O 接口的基本结构

I/O 接口的基本组成如下图所示。



现代计算机一般都采用了**中断技术**，因此接口电路中一般还设有中断请求触发器（INTR），当其为“1”时，表示该I/O设备向CPU发出中断请求。接口内还有屏蔽触发器（MASK），它与中断请求触发器配合使用，完成设备的屏蔽功能。

0.263 04076-I-O 端口及其编址

1. I/O 端口

I/O 端口和 I/O 接口是两个不同的概念。端口指可以由 CPU 进行读或写的寄存器。这些寄存器分别用来存放数据信息、控制信息和状态信息，分别被称为数据端口、控制端口和状态端口。若干个端口加上相应的控制逻辑电路就组成了接口。

2. I/O 端口的编址

I/O 设备与主机交换信息和 CPU 与主存交换信息有很多的不同点，例如，CPU 如何对 I/O 编址就是其中待解决的问题。

一般将 I/O 设备看作**地址码**。

I/O 地址码的编址一般采用两种方式：**统一编址**和**不统一编址**。

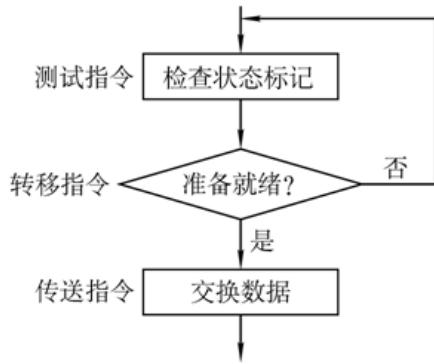
前面讲过了存储器地址，如果能将 I/O 的地址码和主存的地址码统一起来就方便多了，即统一编址，例如，在 64KB 地址的存储空间中，划出 8KB 地址作为 I/O 设备的地址，凡是在这 8KB 地址范围内的访问，就是对 I/O 设备的访问，所用的指令和访存指令相似。显然，统一编址占用了存储空间，减少了主存容量，但无须专用的 I/O 指令。

不统一编址指 I/O 地址和存储器地址是分开的，所有对 I/O 设备的访问必须有专用的 I/O 指令。不统一编址由于不占用主存空间，故不影响主存容量，但需要设置 I/O 专用指令。因此，设计机器时，需根据实际情况衡量考虑选取何种编址方式。

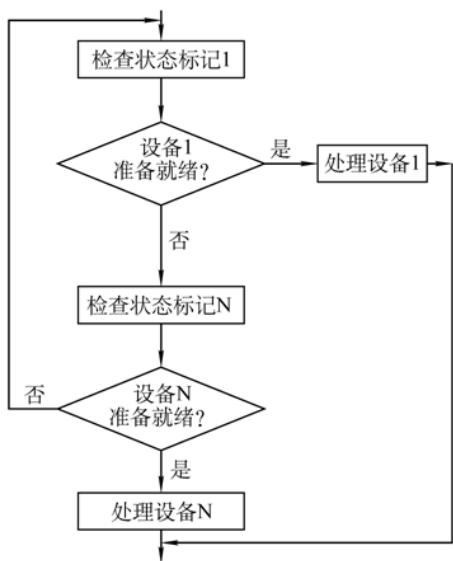
0.264 04077-程序查询方式

程序查询方式又称为**程序控制 I/O 方式**。

在这种方式中，数据在 CPU 和外部设备之间的传送完全靠计算机程序控制，是在 CPU 主动控制下进行的。当需要输入/输出时，CPU 暂停执行主程序，转去执行设备输入/输出的服务程序（指由 I/O 指令编写成的程序），根据服务程序中的 I/O 指令进行数据传送。程序查询方式的主要缺点在于每时每刻需要不断查询 I/O 设备是否准备就绪。下图为单个设备的查询流程。



当 I/O 设备较多时, CPU 需要按各个 I/O 设备在系统中的优先级别进行逐级查询, 其流程如下图所示。下图中设备的优先顺序按 1 ~ N 降序排列。



从单个设备的查询流程可以看出, 为了正确完成这种查询, 通常要执行以下 3 条指令。

- 1) 测试指令。用来查询 I/O 设备是否准备就绪。
- 2) 转移指令。若 I/O 设备未准备就绪, 则执行转移指令, 即转到测试指令, 继续测试 I/O 设备的状态。
- 3) 传送指令。若 I/O 设备准备就绪, 则执行传送指令。

0.265 04078 程序中断方式

1. I/O 中断的定义

CPU 启动 I/O 后, 不必停止现行程序的运行。而 I/O 接到启动命令后, 进入自身的准备阶段。当准备就绪时, 向 CPU 提出请求, 此时 CPU 立即中断现行程序 (不看书了), 并保存断点 (用书签做记号), 转至执行中断服务程序 (打球去了), 为 I/O 服务。中断服务程序结束后, CPU 又返回到程序的断点处, 继续执行原程序。

一次中断处理的过程可简单归纳为 5 个阶段:

- 1) 中断请求。设置中断请求触发器。
- 2) 中断判优。设置中断判优, 用硬件或者软件都可以实现。
- 3) 中断响应。
- 4) 中断服务。

5) 中断返回。

2. 为了处理 I/O 中断，在 I/O 接口电路中必须配置如下相关的硬件线路：

1) 中断请求触发器和中断屏蔽触发器。

2) 排队器。

3) 中断向量地址形成部件。

3. 中断服务程序的完整流程。

(1) 保护现场

保护现场有两个含义：其一是保存程序的断点；其二是保存通用寄存器和状态寄存器的内容。前者由中断隐指令完成，后者可由中断服务程序完成。

(2) 中断服务（设备服务）

这是中断服务程序的主体部分，不同的中断请求源其中断服务操作内容是不同的。

(3) 恢复现场

这是中断服务程序的结尾部分，要求在退出服务程序前，将原程序中断时的“现场”恢复到原来的寄存器中。

(4) 中断返回

中断服务程序的最后一条指令通常是一条**中断返回指令**，使其返回到原程序的断点处，以便继续执行原程序。

0.266 04079-DMA 方式

1. DMA 方式的特点

DMA 方式是一种完全由硬件进行成组信息传送的控制方式，具有程序中断方式的优点，即在数据准备阶段，CPU 与外设并行。它还降低了 CPU 在传送数据时的开销，这是因为信息传送不再经过 CPU，而在外设与内存之间直接进行，因此称为直接存储器存取方式。由于数据传送不经过 CPU，也就不需要保护、恢复 CPU 现场等繁琐操作。这种方式适用于磁盘、磁带等高速设备大批量数据的传送。

2. DMA 的传送方法

如果出现高速 I/O (通过 DMA 接口) 和 CPU 同时访问主存怎么办？这个时候 CPU 就得将总线 (如地址线、数据线等) 的占有权让给 DMA 接口使用，即 DMA 采用**周期窃取**的方式占用一个存储周期。通常，DMA 与主存交换数据时采用以下 3 种方法。

(1) 停止 CPU 访问主存

当外设需要传送一片数据时，由 DMA 接口向 CPU 发一个信号，要求 CPU 放弃地址线、数据线和有关控制线的使用权，DMA 接口获得总线控制权后，开始进行数据传送。在数据传送结束后，DMA 接口通知 CPU 可以使用主存，并把总线控制权交还给 CPU。在这种传送过程中，CPU 基本处于不工作状态或保持原始状态。

(2) 周期挪用

当 I/O 设备没有 DMA 请求时，CPU 按程序的要求访问主存，一旦 I/O 设备有 DMA 请求，就会遇到 3 种情况：第一种情况是 CPU 不在访存 (CPU 正在执行加法指令)，故 I/O 的访存请求与 CPU 未发生冲突；第二种情况是 CPU 正在访存，必须等待存储周期结束后，CPU 再将总线占有权让出；第三种情况是 I/O 和 CPU 同时请求访存，出现了访存冲突，此刻 CPU 要暂时放弃总线占有权，由 I/O 设备挪用一个或几个存储周期。

(3) DMA 与 CPU 交替访问

这种方式适用于 CPU 的工作周期比主存存取周期长的情况，例如，CPU 的工作周期是 1.2s，主存的存取周期小于 0.6s，那么可将一个 CPU 周期分为 C1 和 C2 两个周期，其中 C1 专供 DMA 访存，C2 专供 CPU 访存。

3. DMA 接口的功能

利用 DMA 方式传送数据时，数据的传输过程完全由 DMA 接口电路控制，故 DMA 接口又称为 DMA 控制器。DMA 接口应具有以下几个功能：

- 1) 向 CPU 申请 DMA 数据传送。
- 2) 在 CPU 允许 DMA 工作时，处理总线控制权的转交，避免因进入 DMA 工作而影响 CPU 正常活动或引起总线竞争。
- 3) 在 DMA 期间管理系统总线，控制数据传送。
- 4) 确定数据传送的起始地址和数据长度，并修正数据传送过程中的数据地址和数据长度。
- 5) 在数据块传送结束时，给出 DMA 操作完成的信号。

DMA 的传送过程分为预处理、数据传送和后处理 3 个阶段。

0.267 04081-加密算法

1. 对称加密算法

a. 基本概念：对称加密算法是应用较早的加密算法，技术较为成熟。在对称加密算法中，**数据发送方将明文和加密密钥一起经过加密算法处理后，使其变成密文发送出去**。接收方收到密文后，若想解读原文，则需要使用加密中使用过的密钥及相同加密算法的逆算法对密文进行解密，才能使其恢复成明文。**由于加密和解密使用的密钥只有一个，因此叫做对称加密算法**。

b. 特点：算法公开、计算量小、加密速度快、安全性较差；

c. 常见算法：DES 算法、3DES 算法、TDEA 算法、RC2、RC4 等。

2. 非对称加密算法

a. 基本概念：非对称加密算法又称为“公开密钥加密算法”，之所以叫做非对称，是这种加密算法需要**两个密钥：公开密钥和私有密钥**。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。

b. 非对称加密算法实现机密信息交换的基本过程：

甲方生成一对密钥，并将其中一把作为公用密钥向其他方公开；如果乙方要发送信息给甲方，则使用这个公用密钥对信息进行加密后将密文发送给甲方；甲方再用自己保存的另一把专用密钥对密文进行解密。相反地，甲方可以使用乙方的公用密钥对机密信息进行加密后再发送给乙方；乙方再用自己的专用密钥对加密后的信息进行解密。**简单来说，就是用公用密钥进行加密，采用专用密钥解密**。

非对称加密算法解决了收发双方交换密钥的问题。只要某人将自己的公用密钥公布，任何一方要发送数据给他，只需要采用这个公用密钥对数据进行加密发送给他即可，省去了事先商定密钥的过程。

c. 特点：算法复杂，加密速度慢，安全性依赖于算法与密钥，安全性较好。

d. 常见算法：RSA、Elgamal、ECC 等。

0.268 04084-FDDI 环

1. FDDI 环

a. FDDI 环称为**光纤分布式数据接口**。只需要把 FDDI 环看成是令牌环形网络的一种就行了，只不过其传输介质是多模光纤以及其他一些功能的改进。

b. FDDI 环仍然基于 IEEE 802.5 令牌环标准的 MAC 协议，上层仍采用与其他局域网相同的逻辑链路控制 LLC 协议，分组最大长度为 4500B。FDDI 环主要是用做校园环境的主干网。

c. FDDI 使用了比令牌环更复杂的方法访问网络。和令牌环一样，也需在环内传递一个令牌，而且允许令牌的持有者发送 FDDI 帧。和令牌环不同，FDDI 网络可在环内传送多个帧。因为令牌接受了传送数据帧的任务以后，FDDI 令牌持有者可以立即释放令牌，把它传给环内的下一个站点，无需等待数据帧完成在环内的全部循环。这意味着，第一个站点发出的数据帧仍在环内循环的时候，下一个站点可以立即开始发送自己的数据，故 FDDI 网络可在环内传送多个帧。

注意：由光纤构成的 FDDI 环，其基本结构为逆向双环。一个环为主环，另一个环为备用环。一个顺时针传送信息，另一个逆时针传送信息。当主环上的设备失效或光缆发生故障时，通过从主环向备用环的切换可继续维持 FDDI 的正常工作。这种故障容错能力是其他网络所没有的。

0.269 04087-浮点数的乘除法运算

1. 浮点数乘除法的运算规则

运算规则：两个浮点数相乘，乘积的阶码应为相乘两数的阶码之和，乘积的尾数应为相乘两数的尾数之积。两个浮点数相除，商的阶码为被除数的阶码减去除数的阶码，尾数为被除数的尾数除以除数的尾数所得的商，下面可以用数学公式来描述。

假设有两个浮点数 x 和 y：

$$\&x=Sx \cdot r^{5E7Bjx7D} \quad \&y=Sy \cdot r^{5E7Bjy7D}$$

$$\&y=Sy \cdot r^{5E7Bjy7D} \quad \&y=Sy \cdot r^{5E7Bjy7D}$$

那么有，

$$\&x \cdot y = (Sx \cdot Sy) \cdot r^{5E7Bjx + 5E7Bjy7D} \quad (1)$$

$$\begin{aligned} &\frac{x}{y} = \frac{Sx}{Sy} \cdot r^{\{5E7Bjx - 5E7Bjy\}} \end{aligned}$$

从式 1 和式 2 可以看出，浮点数乘除运算不存在两个数的对阶问题，故比浮点数的加减法还要简单。

2. 浮点数乘除法的运算步骤

第一步：0 操作数检查。

对于乘法：检测两个尾数中是否一个为 0，若有一个为 0，则乘积必为 0，不再作其他操作；若两尾数均不为 0，则可进行乘法运算。

对于除法：若被除数 x 为 0，则商为 0；若除数 y 为 0，则商为，另作处理。若两尾数均不为 0，则

可进行除法运算。

第二步：阶码加减操作。

在浮点乘除法中，对阶码的运算只有 4 种，即 +1、-1、两阶码求和以及两阶码求差。当然，在运算的过程中，还要检查是否有溢出，因为两个同号的阶码相加或异号的阶码相减可能产生溢出。

第三步：尾数乘/除操作。

对于乘法：第 2 章讲解了非常多的定点小数乘法算法，两个浮点数的尾数相乘可以随意选取一种定点小数乘法运算来完成。

第四步：结果规格化及舍入处理。

可以直接采用浮点数加减法的规格化和舍入处理方式。主要有以下两种：

1) 第一种：无条件地丢掉正常尾数最低位之后的全部数值。这种办法被称为截断处理，其好处是处理简单，缺点是影响结果的精度。

2) 第二种：运算过程中保留右移中移出的若干高位的值，最后再按某种规则用这些位上的值进行修正尾数。这种处理方法被称为舍入处理。

当尾数用补码表示时具体规则是：

- 1) 当丢失的各位均为 0 时，不必舍入。
- 2) 当丢失的各位数中的最高位为 0，且以下各位不全为 0 时，或者丢失的最高位为 1，以下各位均为 0 时，舍去丢失位上的值。
- 3) 当丢失的最高位为 1，以下各位不全为 0 时，执行在尾数最低位加 1 的修正操作。
- 4) 舍入处理。

0.270 18659-CIDR

1. CIDR

a. 作用：无分类编址（CIDR）是为解决 IP 地址耗尽而提出的一种措施。CIDR 消除了传统的 A 类、B 类和 C 类地址以及划分子网的概念，因而可以更加有效地分配 IPv4 的地址空间。

b. 地址格式：CIDR 使用各种长度的“网络前缀”来代替分类地址中的网络号和子网号。于是，IP 地址又从三级编址回到了两级编址，其地址格式为

IP 地址 ::= {

为了区分网络前缀，通常采用“斜线记法”（又称 CIDR 记法），即 IP 地址/网络前缀所占位数。

2. CIDR 地址块

将网络前缀都相同的连续的 IP 地址组成“CIDR 地址块”。一个 CIDR 地址块可以表示很多地址，这种地址的聚合常称为路由聚合（也称构成超网），它使得路由表中的一个项目可以表示很多个原来传统分类地址的路由，因此可以缩短路由表，减小路由器之间选择信息的交换，从而提高网络性能。CIDR 中同样使用了掩码来确定其网络前缀。对于“/20”的地址块，其掩码由连续的 20 个“1”和后续 12 个“0”组成。可以看出，“1”对应于网络前缀，“0”对应于主机号。

在使用 CIDR 时，路由表中的每个项目由网络前缀和下一跳地址组成。这样就会导致查找路由表时可能会得到不止一个匹配结果。应当从匹配结果中选择具有最长网络前缀的路由，因为网络前缀越长，其地址块就越小，路由就越具体。最长前缀匹配原则又称为最长匹配或最佳匹配。

0.271 18660-子网划分与子网掩码

1. 子网划分

聪明的人类想出了“**子网号字段**”，使得两级的 IP 地址变为三级的 IP 地址，这种做法叫做子网划分。子网划分属于一个单位内部的事情，单位对外仍然表现为没有划分子网的网络。

划分子网的基本思路：从主机号借用若干个比特作为子网号，而主机号也就相应减少了若干个比特，网络号不变。于是三级的 IP 地址可记为

IP 地址:: = {

凡是从其他网络发送给本单位某个主机的 IP 分组，仍然根据 IP 分组的目的网络号先找到连接在本单位网络上的路由器，然后此路由器在收到 IP 分组后，再按目的网络号和子网号找到目的子网。最后将该 IP 分组直接交付给目的主机。

2. 子网掩码

子网划分与否是看不出来的，如果要告诉主机或路由器是否对一个 A 类、B 类、C 类网络进行了子网划分，则需要**子网掩码**。

子网掩码是一个与 IP 地址相对应的 32 位的二进制串，它由一串 1 和 0 组成。其中，1 对应于 IP 地址中的网络号和子网号，0 对应于主机号。因为 1 对 1 进行与操作，结果为 1；1 对 0 进行与操作，结果为 0。所以使用一串 1 对网络号和子网号进行与操作，就可以得到网络号。

现在的因特网标准规定，所有网络都必须有一个**子网掩码**。如果一个网络没有划分子网，就采用默认子网掩码。A 类、B 类、C 类地址的默认子网掩码分别是 255.0.0.0、255.255.0.0 和 255.255.255.0。

总结：不管网络有没有划分子网，只要将子网掩码和 IP 地址进行逐位的“与”运算，就一定能立即得出网络地址。

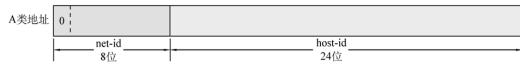
0.272 18661-IPv4 地址

1. IPv4 地址

把整个因特网看作一个单一的、抽象的网络。IP 地址就是给每个连接在因特网上的主机（或路由器）分配一个在全世界范围是唯一的 32 位的标识符。一般将 IP 地址分为 A 类地址、B 类地址、C 类地址、D 类地址和 E 类地址。D 类地址为组播地址，将在 IP 组播中讲解，E 类地址为保留地址，留作以后使用。

2. A 类地址

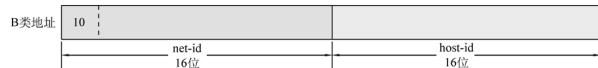
A 类地址的网络号为前面 8 位，并且第一位规定为 0，如下图所示。



A 类地址可以指派的网络数为 $2^7 - 2$ ，每个 A 类网络上的最大主机数是 $2^{24} - 2$ 。

3. B 类地址

B 类地址的网络号为前面 16 位，并且前面 2 位规定为 10，如下图所示。



B 类地址可以指派的网络数为 $2^{14} - 1$ 。每个 B 类网络上的最大主机数是 $2^{16} - 2$ 。

4. C 类地址

C 类地址的网络号为前面 24 位，并且前面 3 位规定为 110，如下图所示。



C 类地址可以指派的网络数为 $2^{21}-1$ 。每个 C 类网络上的最大主机数是 2^8-2 。

5. 六种特殊的 IP 地址

特殊地址	网络号	主机号	源地址或目的地址
网络地址	特定的	全 0	都不是
直接广播地址	特定的	全 1	目的地址
受限广播地址	全 1	全 1	目的地址
这个网络上的这个主机	全 0	全 0	源地址或默认目的地址
这个网络上的特定主机	全 0	特定的	目的地址
环回地址	127	不是全 0 或全 1	源地址或目的地址

0.273 20845-时间复杂度与空间复杂度分析基础

1. 时间复杂度

将算法中基本操作的执行次数作为算法的时间复杂度，多数情况下取最深层循环内的语句所描述的操作作为基本操作。

假设基本操作所执行的次数是 n 的一个函数 $f(n)$ ，那么时间复杂度函数 $T(n)=O(f(n))$ 中增长最快的项/此项的系数)

常见的复杂度有：

$O(1)$ $O(\log_2(n))$ $O(n)$ $O(n\log_2(n))$ $O(n^2)$ $O(n^3)$

2. 空间复杂度

运行时所需额外内存空间的度量。

0.274 20848-数据物理结构

数据的物理结构（或存储结构）是数据的逻辑结构在计算机中的物理表示。它包括数据的表示和关系的表示。比如一个链表结点，结点包含值域和指针域。值域就是数据的表示，指针域就是关系的表示。

在数据结构中有 4 种常用的存储方法：

1. 顺序存储方法：把逻辑上相邻的结点存储在物理位置上相邻的存储单元中。
2. 链式存储方法：不要求逻辑上相邻的结点在物理位置上相邻，结点间的逻辑关系是由附加的指针字段表示的。
3. 索引存储方法：通过建立附加的索引表来标识结点的地址。
4. 散列（或哈希）存储方法：根据结点的关键字直接计算出该结点的存储地址。

0.275 20849-算法

算法可以理解为有基本运算及规定的运算顺序所构成的完整的解题步骤。

1. 算法的特性

- 1) 有穷性；
- 2) 确定性；
- 3) 输入；
- 4) 输出；
- 5) 可行性。

2. 算法的设计目标

- 1) 正确性；
- 2) 可读性；
- 3) 健壮性；

4) 高效率与低存储量需求（即更低的时间、空间复杂度）。