SiSTEM PROGRAMLAMA VE ILERI C UYGULAMALARI (I)

ÖZET NOTLARI

Bu Döküman Kaan Aslan'ın C ve Sistem Programcıları Derneği'nde vermiş olduğu "Sistem Programlama ve Ileri C Uygulamalari (I)" kursunda sınıf içerisinde alınan notlardan oluşturulmuştur. Notlar üzerinde herhangi bir düzeltme yapılmamıştır.

SISTEM PROGRAMLAMADA IŞLENECEK BELLI BAŞLI KONULAR:

1- Fonksiyon göstericiler
2- Gösterici göstricileri
3- Kesmeler
4- Bellekte kalacak programlar
5- Haberleşmenin temelleri (seri haberlerşme paralel haberleşme)
6- Disk işlemleri(fat root)
7- Kendi kendini çağıran fonksiyonlar (recursive fonksiyonlar)
8- Yararlı sistem fonksiyonları kullanımı
9- Korumalı mod hakkında temel bilgiler
10- Parcing algoritmaları
11- İşlemci kavramı ve programlanmaları
12- Veri tabanlarının oluşturulması hakkında bilgiler
13- Özel problemler

İLERİ C

FONKSİYON GÖSTERİCİLERİ:

```
void fonk (void)
{
.....
.....
}
```

Değişkenlerde olduğu gibi fonksiyonlar da memory'de yer kaplamaktadır ve adresleri vardır. Bir fonk başlangıç adresi özel olarak tanımlanmış fonk göstericisinin içerisine yerleştirilebilir.

Fonksiyon Göstericilerinin Genel Biçimi

[fonksiyonun geri dönüş değerinin türü](*gösterici ismi)([param]);

void (*fonk) (void) : *fonk yanındaki () olmasa bir fonksiyonun prototipi olduğu anlamına gelir!!! geri dönüş değeri void türden bir adres
Bir fonksiyon göstericisine her türlü fonksiyonun başlangıç adresi yerleştirilemez geriş dönüş değri ve parametreleri belli türden olan fonksiyonlar yerleştirilebilir.

Örneğin burada p fonksiyon göstericisine geridönüş değeri ve parametresi void olan bir fonksiyonun adresi yerleştirilebilir.

Fonksiyon gösterici bildiriminde parametre parantezinin içerisi boş bırakılırsa o fonksiyon göstericisine atanacak fonksiyon adreslerinde parametre koşulu aranmaz.

(void) olursa parametresi olmamalıdır.
() her tür parametre değeri kabul olur.

FONKSİYONLARIN BAŞLANGIÇ ADRESLERİNİN ELDESİ:

C'de bir fonksiyon ismişnden sonra parantez açılmaz ise yani fonk çağırma operatörü kullanılmaz ise o ifade fonksiyonun başlangıç adresi gösteren adres sabiti olarak ele alınır.

FONKSİYON GÖSTERİCİSİ YOLUYLA FONKSİYONUN ÇAĞIRILMASI:

Aslında () operatörü unary postfix bir operatördür. öncelik tablosunun en yüksek düzeyinde bulunur. görevi programın akışını operamdı olan adrese yönlendirmektir. yani operant bir fonk adresi olmak zorundadır. bu durumda fonksiyon göstericisi yardımı ile bir fonksiyon () opeartörü ile çağırılabilir. fonksiyon gösatericisi ile fonksiyon çağırmanın bir yolu daha vardır. (*p)();
fonksiyon göstericisi iki yolla kullanılsada okunabilirlik açısından çoğu kez (*p)() şeklinde çağırma pnin bir fonksiyon göstericisİ olduğunu açıkça belirttiği için daha çok tercih edilir.

FONKSİYON GÖSTERCİLERİNİN PARAMETRE DEĞİŞKENİ OLARAK KULLANILMASI

Bir fonksiyonun parametre değişkeni parametre değişkeni olabilir. bu durumda fonksiyon bir fonksiyon ismi ile yani fonksiyon adresi ile çağırılmalıdır.

FONKSİYONUN GERİ DÖNÜŞ DEĞERİNİN BİR FONKSİYON ADRESİ OLMASI DURUMU:

Böyle fonksiyonların tanımlanmasında * operatörü fonksiyonun ismi ile birlikte paranteze alınır, parantezimn soluna geri dönüş değerine ilişkin fonksiyonun geri dönüş değeri sağına geri dönüş değerine ilişkin fonksiyonun parametre yapısı yazılır.

```
void (* fonk(void)) (void)
{

}
```

---fonpoin3.c---

```
#include <stdio.h>


void sample(void)
{
   printf("I am sample....\n");
}
void (*fonk(void))(void)
{
   return sample;
}
void main(void)
{
```

```
   void (*p)(void);

   p=fonk();
   p();
}
```

void(*(*sample(void))(void))(void)---> function returning to pointer function returning to pointer to function

Burada sample fonksiyonunun parametresi void geri dönüş değeri, parametresi void geri dönüş değeri, parametresi void geri dönüş değeri void olan bir fonksiyondur.

ÇOK BOYUTLU DİZİLER::

Çok boyutlu bir dizi bildirimi <tür><dizi ismi>[n1][n2] n1 ve n2 sabit ifadesi olmak durumundadır. Genellikle iki boyutlu dizilere rastlanır ve bunlara genellikle matris denir. Bir çok boyutlu dizinin elemanları bellekte ardaşıl bulunur ve organizasyon son boyutu bir dizi biçiminde ele alınarak ardaşıl yerleştirilmesi biçimindedir. (ROWSIZE,COLSIZE) şeklinde tanımlanmış bir matrsin başlangıç adresi matrisle aynı türde olan bir p pointeri içinde olsun;
p göstericisini (row,col) ile belirlenen elemanı gösterecek şekilde ilerletmek için p=p+row*COLSIZE+col ifadesi yazılır. bir matris bildiriminde matrsini ismi matrisin başlangıç adresini tek [] kurulan ifadeler,a[0], gibi matrisin satırlarını gösteren adres sabitleridir.

MATRİSİN BAŞLANGIÇ ADRESİNİN GÖSTERİCİYE ATANMASI

Bir matris tanımlarken satır ve sütün uzunlukları sabit ifadesi biçiminde belirtilmek zorundadır ancak bir matrisin adresini yerleştirercek bir gösterici tanımlamak için matrisin sutün sayısı bilinmek zorundadır. bu türden bir gösterici int (*p)[COLSIZE]; şeklinde tanımlanır. bu bildirimden şu anlaşılır; p bir göstericidir p göstericisinin gösterdiği yerde colsize uzunluğunda int bir dizi vardır.

---matris1.c---

```
#include <stdio.h>
#include <conio.h>

#define COLSIZE 3

void fonk(int (*p)[COLSIZE])
{
   int i,k;

   clrscr();

   for (i = 0; i < 2; ++i)
      for(k = 0; k < 3; ++k)
```

```
        printf("%d%c", p[i][k], ( k == COLSIZE-1)? '\n':' ');

}
void main(void)
{
   int a[2][COLSIZE]={{2,4,6},{8,10,12}};
   fonk (a);
}
```

DİZİLERİN SIRAYA DİZİLMESİ (SORTİNG)

Sıraya dizme küçükten büyüğe veya büyükten küçüğe olabilir. dizileri sıraya dizmek için pek çok algoritmik yöntem vardır.
Belli başlı yötemler:
buble sort
selection sort
quick sort
shell sort
binary tree sort
...

sıraya dizme işleminin süresi dizinin dağılıma göre değişebilir. dağılım rastgele ise çeşitli simulasyon yöntemleri ile en iyi sıralama algoritmasının quicksort olduğu belirtilmiştir.

HER TÜRLÜ DİZİYİ SIRAYA DİZEN BİR FONKSİYONUN TASARIMI

her türlü diziyi sıraya dizebilecek bir fonksiyon aşağıdaki prototiple yazılmaya çalışılabilir.

```
void sort(void *ptr, int size, int type);
```

fonksiyonun birinci parametresi sıraya dizilecek dizinin başlangıç adresini tutacak göstericidir. ikincisi dizinin uzunluğu üçüncüsü ise dizinin türüdür.

```
#define _INT_ 0
#define _LONG_ 1
.....
```

fonksiyon içerisinde type değişkeni switch içerisinde kontrol edilerek uygun türden bir gösterici yardımı ile sıraya dizme işlemi yapılır.ancak bu şekilde bir yapı dizisi sıraya dizilemez çünkü yapının elemanları değişken olabilir. oysa "qsort" isimli standart c fonksiyonu yapı dizisi de dahil olmak üzere her türlü diziyi sıraya dizebilmektedir. her türlü sort algoritmasında dizinin iki elemanı algoritmik yönteme göre karşılaştırılır, eğer koşul uygun ise yer değiştirilir. her türlü diziyi sıraya dizen bir fonksiyon iki elemanın karşılaştırılamsını kendisi içeride sağlamaz bunun yerine diziyi sıraya dizecek programcının fonksiyonunu çağırır.
qsort un prototipi void qsort(void *ptr, unsigned size, unsigned width, int (*cmp)())

1. parametre sıraya dizilecek dizinin başlangıç adresidir.
2. parametre dizinin toplam eleman sayısıdır.
3. parametre dizinin bir elemanın uzunluğudur.

4. parametre karşılaştırma fonksiyonunun adresidir.
karşılaştırma fonksiyonu dizinin yerdeğiştirilme sorgulamasının yapıldığı iki elemanın adresi ile çağırılacaktır. dolayısı ile bu fonksiyonun iki parametresi sıraya dizilecek dizi ile aynı türden olan gösterici olacaktır. karşılaştırma fonksiyonu birinci parametre ile belirlenen eleman ikinci parametre ile belirlenenden büyük ise pozitif herhangi bir değere küçük ise negatif herhangi bir değere ve eşitse 0'a geri dönmelidir. algoritmanın temöel fikri iki elemanın yer değiştirmek için elemanın türünü bilmeye gerek yoktur. başlangıç adresleri ve uzunluğu bilinse yeterlidir.

---bsort.c---

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

void bsort(void *ptr, unsigned size, unsigned width, int (*cmp)())
{
   char *_ptr = (char *)ptr;
   int k,i;
   void *temp;

   temp = malloc (width);
   if (temp == NULL) {
      printf("memory allocation error...\n");
      exit(1);
   }
   for (k = 0; k < size-1; ++k)
      for (i = 0; i < size-1 ; ++i)
         if (cmp(_ptr + i * width, _ptr + (i + 1) * width) > 0) {
            memcpy(temp, _ptr + i * width, width);
            memcpy(_ptr + i * width, _ptr + (i + 1) * width, width);
            memcpy(_ptr + (i + 1) * width, temp, width);
         }
   free(temp);
}

int fcompare(int *p1, int *p2);

void main (void)
{
   int a[5] = {3, 8, 4, 2, 7};
   int j;

   clrscr();
   bsort(a, 5, sizeof(int), fcompare);

   for (j = 0; j < 5;++j)
```

```
        printf("%d\n", a[j]);

}

int fcompare(int *p1, int *p2)
{
   if (*p1 > *p2)
      return 1;
   if (*p1 < *p2)
      return -1;
   return 0;
}


void qsort(void *ptr, unsigned size,unsigned width, int(*cmp)());
```

--------qsort.c--------

```
#include <stdio.h>
#include <stdlib.h>

int cmp(int *p1, int *p2);

void main(void)
{
        int a[] = {8, 3, 4, 5, 7, -2, 71, 47, 10, 9};
        int i;

        qsort(a, 10, sizeof(int), cmp);
        for (i = 0; i < 10; ++i)
                printf("%d\n", a[i]);
}

int cmp(int *p1, int *p2)
{
        if (*p1 > *p2)
                return 1;
        if (*p1 < *p2)
                return -1;
        return 0;
}
```

QSORT'UN PARAMETRELERİ

1.parametre dizinin baslanbgıc adresi
2.parametresi dizininin toplam uzunlugu

3.parametresi dizinin bir elemanın uzunlugu
4.parametresi karsılastırma fonksiyonunun adresidir.karsılastırma fonksiyonunun dizi ile aynı turden iki gosterici parametresi olmalıdır.Bu fonksiyon karsılastırma işleminin yani,sıraya dizme işleminin nasıl yapılacagını tanımlamakta kullanılır.karsılastırma fonksiyonunun geri donus degeri int olmak zorundadır.karsılastırma fonksiyonu dizinin iki elmanının adresiyle qsort tarafından cagırılmaktadır.Eger birinci parametre ile belirtilen eleman ikinci parametre ile belirtilen elamandan buyukse + herhangi bir degerle, kucukse - herahngi bir degerle ve iki eleman birbirine eşitse 0 degeriyle geri donmelidir.

QSORT İLE BİR YAPI DİZİSİNİN SIRAYA DİZİLMESİ

----------qsort2.c----------

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _PERSON {
        char name[30];
        int no;
}PERSON;

int cmp(PERSON *p1, PERSON *p2);

void main(void)
{
        PERSON x[] = {{"Ali Serçe", 123}, {"Yücel Gündüz", 153"},
                        {"Kaan Arslan", 518}, {"Mehmet Eraslan", 218},
                        {"Sacit Er", 174}, {"Güray Sönmez", 317},
                        {"Ayşe Er", 817}, {"Can Ak", 714},
                        {"Hilmi Akçaoğlu", 417}, {"Hakan Şen", 405}};
        int i;

        qsort(x, 10, sizeof(PERSON),cmp);
        for (i = 0; i < 10; ++İ)
                printf("%s %d\n", x[i].name, x[i].no);
}

int cmp(PERSON *p1, PERSON *p2)
{
        if (p1 -> no > p2 -> no)/*bunlar yerie return strcmp(p1->name,p2->name);isme gore*/
                return 1;
        if (p1 -> no < p2 -> no)
                return -1;
        return 0;
}
```

C'de her ifdenin bir türü vardır.Ve bi ifade için iki durum söz konusudur.Ya ifade bellekte bir yer kaplar yani nesnedir, ya da değildir.

BİR DİZİN İÇERİSİNDEKİ DOSYALARIN BULUNMASI

Bir dizin içerisindeki dosyalarıo bulabilmek için işletim sisteminin sundugu iki fonksiyonundan faydalanılır.Bu sistem fonksiyonlarının isimleri dos'ta derleyiciye gore değişebilmektedir. Borland derleyicilerinde findfirst, findnext;microsoft derleyicilerinde _findfirst, _findnext isimleriyle bulunur.Windows altında API fonksiyonu olarak FindFirstFile ve FindNextFile fonksiyonlarıyla bu işleri yapmak mümkümdür.

findfirst FONKSİYONU

prototipi:
int findfirst(const char *filename, struct ffblk *finfo, int attrib);

Fonksiyonun prototipi "dir.h" içerisindedir.Bu fonksiyon bir dosyanın (direstory de)bir dizinin içerisinde olup olmadıgına bakar, eger varsa o dosyanın bilgilerini elde eder.

1.parametresi aranan dosya(full path ile).
2.parametresi struct ffblk turunden bir yapı değişkeninin adresini alır.ffblk isimli yapı dir.h içerisinde belirtilmiştir.
3.parametresi hangi ozellige sahip dosyaların aranacagını belirtir.0 yazmak arşiv ve read-only dosyaları aratır.bi genelde 0 yazacagız.

Eger soz konusu dosya bulunmus ise fonksiyon 0 degeriyle bulunamadıysa 0 dısı herhangi bir deger ile geri doner.Bu fonksiyon birden fazla dosya arastırmak için de kullanılabilir.Bu durumda dosya ismi olarak joker karakterleri girilebilir.Kosulu saglayan birden cok dosya varsa ilkini bulacaktır.Kosulu saglayan diger dosyalar findnext ile bulunur.

findnext FONKSİYONU

prototipi:
int findnext(struct ffblk *finfo);

findnext fonksiyonu her cagırılıdgında kosulu saglayan bir sonraki dosyanın bilgilerini elde eder.Geri donus degeri 0 ise basarılı, 0 dısı ise basarısız demektir.tipipk bir dizin arama algoritması soyledir:

```
result = findfirst(.....);
while(!result) {
        ....
        ....
        ....
        result = findnext(...);
}
```

----------findfn.c----------

```
#include <stdio.h>
#include <dir.h>
#include <conio.h>

#define MAXPATH 80

void main(void)
{
   char path[MAXPATH];
   struct ffblk info;
   int i, result;

   clrscr();

   printf("path=");
   gets(path);
   result = findfirst(path, &info, 0);
   while(!result) {
      printf("%-20s %ld\n", info.ff_name, info.ff_fsize);
      result = findnext(&info);
   }
}
---------------------------
```

BİR DİZİN'İN İÇERİSİNDEKİ DOSYALARIN SIRALANMASI

Bunun için dizin içerisindeki bütün dosyalar dinamik olarak yaratılan ve büyütülen bir dizi içerisinde saklanır.Sonra dizi qsort fonksiyonuyla sıraya dizilir.

```
-----------dirsort.c------------
#include <stdio.h>
#include <stdlib.h>
#include <dir.h>
#include <string.h>
#include <conio.h>

#define MAXPATH     80

int cmp(struct ffblk *p1, struct ffblk *p2);

void main(void)
{
   struct ffblk *pinfo,f;
```

```
    char path[MAXPATH];
    int size, result, i;

    clrscr();

    printf("Path=");
    gets(path);
    size = 0;
    result = findfirst(path, &f, 0);
    pinfo = (struct ffblk *) malloc(sizeof(struct ffblk));
    if (pinfo == NULL) {
       printf("Cannot allocate memory");
       exit(1);
    }
    if (result) {
       printf("Cannot find file..\n");
       exit(1);
    }
    for(;;) {
       if(result)
          break;
       pinfo[size] = f;
       ++size;
       result = findnext(&f);
       if (result)
          break;
       pinfo = (struct ffblk *) realloc(pinfo, sizeof(struct ffblk)*(size+1));
       if (pinfo == NULL) {
          printf("Cannot allocate memory");
          exit(1);
       }

    }
    qsort(pinfo, size, sizeof(struct ffblk), cmp);
    for (i = 0; i < size; ++i) {
       printf("%-20s %ld\n", pinfo[i].ff_name, pinfo[i].ff_fsize);
       if (i % 24 == 23) {
          printf("Press any key to continue..\n");
          getch();
       }
    }
    printf("%d tane dosya dizdik", size);
}

int cmp(struct ffblk *p1, struct ffblk *p2)
{
    return strcmp(p1 -> ff_name, p2 -> ff_name);/*bu isme gore dizer*/
```

```
/*  if(p1 -> ff_fsize > p2 -> ff_fsize) /*bu uzunluga gore dizer*/
        return 1;
   if(p1 -> ff_fsize < p2 -> ff_fsize)
        return -1;
   retun 0;
*/
}
```

HEAP PROBLEMİ

Dosya bilgilerinin bir diziye alınarak saklanması dos gibi heap alanının küçük ve sınırlı oldugu bir sistemde probleme yol acabilir.Önerilen yöntem heap problemi başladığı zaman bütün dizinin bir dosyaya yazılması ve işlemlere bir dosyada devam edilmesidir.Dosyadaki bilgilerin sort edilebilmesi için etkin bir algoritmik yönteme gereksinim vardır.Örneğin binary tree gibi bir arama ağacı disk üzerinde kurulabilir.

KESMELER(interrupts)

KESME NEDİR?

Kesme mikro işlemcinin üzerinde çalıştığı koda ara vererek başka bir kodu çalıştırması işlemidir. Kesmeler çağırılma kaynaklarına göre 3 kısma ayrılırlar:

1-Yazılım kesmeleri
2-Donanım kesmeleri
3-İçşel kesmeler

Yazılım kesmeleri(Software interrupts)

Yazılım kesmelerinin normal fonksiyon cagırmalarından işlevsel bir farkı yoktur.Bunlar programcı tarafından yazılan INT hh makina komutuyla koda dahil edilirler.

Donanım kesmeleri(Hardware interrupts)

Bunlar gercek işlevsel kesmelerdir.Yazılım kesmeleri programcı tarafından cagırılırken donanım kesmeleri elektriksel yolla cagırılmaktadır.Her mikro işlemcinin ve mikro denetleyicinin donanım kesmesi için bir INT ucu vardır.Bu INT ucu uyarıldıgında (uyarılması elektriksel olarak 5v ya da 0v gerilimle uygulanması anlamına gelir) mikro işlemci o anda çalıştırılmakta olan koda ara verir ve başka bir kodu yugulamaya başlar.Yani bu tur kesmelerde kesmenin oluş mekanizması dışsal ve elektriksel olaylara bağlıdır.

İçsel kesmeler(Internal interrupts)

Mikro işlemcinin bir makina kodunu calıstırırken problemle karsılastıgında kendi kendisini cagırdıgı kesmelerdir.

Donanım Kesmelerinin İncelenmesi

80x86 mimarisinde mikro işlemcinin INT ucuna donanım birimleri dogrudan baglanmamıstır.Donanım birimleriyle INT ucu arasına arabirim gorevi yapan ve ismine kesme denetleyicisi denilen ayrı bir işlemci bulunur.(kesme denetleyicisi=8259 (PIC=programmable interrupt controller))
Kesme denetleyicisinin dışsal devrelere baglanacak 8 ucu vardır.PC mimarisinde 1 tane kesme denetleyicisi vardı,AT'lerle beraber sayısı 2'ye cıkartılmıstır.2 kesme denetleyicisi kullanabilmek için 1. kesme denetleyicisinin bir ucunu 2. kesme denetleyicisinin ozel bir ucuna baglamak gerekir.Bu durumda bugun kullandıgımız sistemlerde donanım kesmesi olusturabilecek 15 dışsal birim vardır.Kesme denetleyicisinin uçları kesme oluturan baska işlemcilere baglıdır. Bu yolla donanım kesmesi olusturulmasına IRQ (interrupt request) denir.Birinci kesme denetleyicisinin ilk ucundan baslayarak her uca bir IRQ numarası verilmiştir. Cascade baglantıda (ilk PIC'in ikinciye baglanması olayı) kullanılan uç birinci kesme denetleyicisinin 2 numaralı ucudur(0 baslayarak numaralandırıyoruz). Bu durumda IRQ 2 tanımlı değildir.Intel işlemcilerinde toplam 256 tane kesme numarası vardır(software + hardware + internal).Yani kesme biçiminde çalıştırılacak 256 tane ayrı kod vardır.Örneğin yazılım kesmelerinde INT makina kodunun yanına bir byte sayı yazılır,bu sayı kac numaralı kesmenin kullanılacagını belirtir.Donanım kesmelerinde çalıştırılacak kesme numarası yine kesme denetleyicisi ile mikro işlemci arasındaki 8 elektriksel yolla belirlenir.Yani mikro işlemci INT ucunun uyarıldıgını gordugunde D0-D7 uclarına bakarak kaç numaralı kesmenin çalıştırılacağını kesme denetleyicisinden alır.

PC MİMARİSİNDE KESME OLUŞTURAN DIŞSAL DEVRELER

IRQ 0

Bu ilk kesme denetleyicisinin ilk ucudur.Intel'in 8254 zamanlayıcı işlemcisine baglıdır (programmable interval timer).Bu zamanlama devresi saniyede 18.2 kere darbe üretecek biçimde programlanmıştır.Yani bir program çalışırken saniyede 18.2 kere darbe oluşarak başka bir kod çalıştırılmaktadır.Bu kesme oluştuğunda mikro işlemci 8 numaralı kesme kodunu (interrupt handler) çalıştırır.

8H nolu kesme kodu ne yapar:

bu kesmenin kodu temelde iki işlemi gerçekleştirir.
1- sistemin saatini çalıştırır. işletim sistemi yüklendiğinde RTC işlemcisinden tarih ve zaman bilgisini alır ve her kesme geldiğinde zaman bilgisini arttırır. (frekansı 18.2 s^-1)

2- floppy motorunun durdurulmasını sağlar. floppy işlemi bitirildiğinde motor birden durdurulmaz çünkü küçük zaman aralıkları ile floppyi kullanan programlar durmamasından fayda sağlarlar. her kesme geldiğinde bu kesmenin içerisinde ayarlanmış bir sayaç bir eksiltilir sayaç sıfır olunca floppy motoru durdurulur.
Mikro işlemcinin hızından bağımsız zamanlama işlemleri:
mikro işlemcinin hızından bağımsız olarak zamanlamayı sağlamak için iki temel yöntem akla gelebilir.
1- mikro işlemcinin hızını tespit ederek işlemleri yürütmek. bu yöntem uygulanması son derece güç ve duyarlı olmayan bir yöntemdir.

2- bilgisayarın hızından bağımsız bir biçimde çalışan zamanlayıcı işlemci devrelerinden faydalanmak PC mimarisinde bir 8254 bir tane de RTC işlemcisi bu işlemi yapmaya adaydır. 8254 yoluyla bu işlemi gerçekleştirmek dosta mümkün olsa da multitasking başka sistemlerde mümkün olmayabilir. RTC işelmcisini programlamak windows altında daha uygun gibi görünmektedir. ancak genel olarak mimari zamanlama konusunda zayıf tasarlanmıştır. ancak önemli zamanlama işlemleri için bu işlemi gerçekleştirecek ayrı kartlar kullanılabilir.

IRQ 1 ve 9 numaralı kesme kodu

Birinci un bir numaralı ucuna klavye denetleyicisi denilen bir işlemci bağlıdır. (keyboard controller).(klavye denetleyici genelde intel 8042 yada benzeri ir işlemci kullanılır.) klavyede bir tulşa basıldığı zaman klavye içindeki yerel işlemci (intel 8048 türevi) basılan tuşun sıra numarasını elde ederek kablo yolu ile seri bir biçimde bilgisayar içerisindeki klavye denetleyicisine yollar, bu bilgiyi saklar ve IRQ 1 donanım kesmesinin çağırılmasına yol açar. IRQ 1 dolayısı ile çağırılan kesme numarası 9 dur, bu kod klavye denetleyicisinden basılan tuşun bilgisini alarak klavye tampon bölgesine yazar (memoryde). programlama dillerinde klavyeden karakter alan fonksiyonlar doğrudan klavye tampon bölgesine bakmaktadır. 9 numaralı kesme kodu basılan tuş üzerinde çeşşitli kontrolleri de yapar örneğin pause tuşu prog bekletir. ctrl+alt+del reset eder. caps ışığı yakar vs....

IRQ 2

Bu irq hattı ikinci 8259 a bağlantı için harcanmıştır.

IRQ 3 VE IRQ 4

IRQ 3 com2 com4
IRQ 4 com1 com3

Bilgisayarın dış dünya ile iletişiminin kolay sağlanması için seri ve paralel portları vardır. Seri portlar seri iletişimde kulanılırlar; seri portlar bilgisayarda 25 ve 9 uçlu erkek connector biçiminde bulunmaktadır. 3 ve 4 numaraları IRQ hatlarına intel 8250 UART işlemcisi bağlıdır. Bu işlemci bütün seri haberleşme işemleri için dış dünya ile bağlantı kurmak üzere tasarlanmıştır. Bu işlemci otomatik bazı işlemlerin sağanabilmesi için donanım kesmesine ihtiyaç duymaktadır.

IRQ 5

Bu IRQ hattı pc lerde harddisk kontrol işlemcisine bağlıydı. Ancak AT'lerle beraber LPT2 ya da 2 numaralı paralel port kullanımına bırakılmıstır. Bilgisayarda paralel haberleşme için 25000 dişi konnektör kullanılmaktadır. Genellikle printer kullanımı için bulundurulur. Özellikle printer kullanımında otomaik baz işlemlerin sağlanabilmesi için IRQ gereksinimi vardır.

IRQ 6

Bu IRQ hattı Intel 8272 floppy denetleyici işlemcisine bağlıdır(Floppy controller). Bir floppy işlemi bitirildiğinde o anda çalıştırılmakta olan programın haberdar edilmesnde kullanılır.

IRQ 7

Birinci paralel port'a atanmıştır.(LPT1)

IRQ0-7 arasındaki hatlar için çağırılan kesme numaraları 08-0F arasındadır. Intel 8086 işlemcisi 00-07 arasındaki kesme numaralarını içsel kesmeler için kullanmaktadır. Ancak 286'larla beraber içsel kesme sayısı 8'den 32'ye çıkarılmıştır. Yani bugün kullandığımız mimaride 8 içsel kesme 8 IRQ kesmesiyle çakışmaktadır.

IRQ 8

RTC devresine bağlıdır

IRQ 9

Bu kesem dışsal kullanımlara ayrılmıştır.Bu IRQ'nuın kesme numarası 70h'tır.

IRQ 10(0A). 11, 12

Dışsal kullanıma ayrılmıştır

IRQ 13

Matematik işlemciye 80287 ve 80387 matematik işlemcilerine bağlıdır.

IRQ 14,15

PC lerde dışsal kullanımlara ayrılmıştır. PS2 modellerinde harddisk denetleyici işlemcisine bağlıdır.

YAZILIM KESMELERİ

Yazılım kesmeleri programcının program koduna dahil ederek çalıştırdığı kesmelerdir makina dilinde yazılım kesmesi için INT makina komutu kullanılır. INT makina komutu 2 byte tır komutun hangi kesme numarasını çağırılacağını anlatan bir parametresi vardır. yazılım kesmesi için register kavramını bilmek gerekir.

REGISTER NEDİR??

CPU ile RAM bağlantı halindedir. Ram'deki bazı bilgilerin işleme sokulabilmesi için cpu ya çekilmesi gerekir. Bu bilgilerin cpu içinde geçiçi olarak bekletildikleri küçük bellek bölgelerine register denir. bit uzunlukları mikro işlemcinin bir hamlede yapabileceği işlemin uzunluğu ile ilgilidir. 8086 16 bit, 80286 ve sonrası 32 bit mikroişlemcilerdir.

NOT:::
bugünkü pentium işlemcileri 8086 gibi de çalışabilmektedir, işlemci 3 çalışma moduna ayrılmıştır.

1- gerçek mod
2- v86 modu:korumalı modun çeşitli özelliği ile gerçek modun çeşitli özelliklerinin kombine edildiği moddur.
3- protected mode : korumalı mod mikroişlemcinin en yüksek performansta çalıştığı en gelişmiş modudur.
windows altında çalışırken dos penceresi açıldığında yada bir dos prog çalıştığında mik.işl. v86 modunda bir 8086 gibi çalışıyor. bu durumda dos penceresinde çalışmak demek 8086 işlemcisi ile çalışmak anlamına gelmektedir.

Kesme bir fonksiyon gibi çalışır.Yani kesmenin de parametreleri ve geri dönüş değeri söz konusudur.Ancak kesmenin parametreleri register'lar içerisine yazılır,geri dönüş değeri de register'lar içerisinden alınmaktadır.

8086 İŞLEMCİSİNİN REGISTER YAPISI

8086 işlemcisinde 14 tane register vardır.

4 tane genel amaçlı register vardır.Bunların isimleri AX(acumulator register), BX(base register), CX(count register), DX(data register)'dir. Bunlar iki kısma ayrılırlar.Genel amaçlı registerlar kendi aralarında iki parcanın toplamı biçimindedir.

AX = AH + AL
BX = BH + BL
CX = CH + CL
DX = DH + DL

AX	
AH	AL
BX	
BH	BL
CX	
CH	CL
DX	
DH	DL

Bu parçalar bağımsız 8 bitlik registerlar olarak da kullanılabilir. Parçalar ayrı ayrı oluşturulup bütün olarak kullanılabilir.

4 tane segment register vardır:

CS(code segment register), DS(data segment register), SS(stack segment register), ES(extra segment register).

3 tane pointer register vardır:
BP(base pointer register), SP(stack pointer register), IP(instruction pointer register).

2 tane index register vardır:
SI(source index register), DI(destination index register)

1 tane bayrak register vardır(flag register):

Bütün register'lar 16 bittir.

8086 REGISTERLARININ BİR BİRLİK(union) İLE TEMSİL EDİLMESİ

Register yapısını temsil etmek amacıyla iki yapı ile bir birlik kullanılmaktadır. Bu yapılar ve birlikler dos.h dosyası içerisinde bildirilmiştir.

```
strut BYTEREGS {
        unsigned char al, ah, bl, bh, cl, ch, dl, dh;
};
struct WORDReGS {
        unsigned int ab, bx, cx, dx, si ,di, cflag, flags;
};
union REGS {
        struct BYTEREGS h;
        struct WORDREGS x;
};
```

union REGS BİRLİĞİNİN BELLEKTEKİ ELEMAN YERLEŞİMİ

regs.h.al	regs.x.ah	regs.h	regs.x	regs
regs.h.ah	regs.x.ah	regs.h	regs.x	regs
regs.h.bl	regs.x.bx	regs.h	regs.x	regs
regs.h.bh	regs.x.bx	regs.h	regs.x	regs
regs.h.cl	regs.x.cx	regs.h	regs.x	regs
regs.h.ch	regs.x.cx	regs.h	regs.x	regs
regs.h.dl	regs.x.dx	regs.h	regs.x	regs
regs.h.dh	regs.x.dx	regs.h	regs.x	regs
	regs.x.si		regs.x	regs
	regs.x.si		regs.x	regs
	regs.x.di		regs.x	regs
	regs.x.di		regs.x	regs
	regs.x.cflag		regs.x	Regs
	regs.x.cflag		regs.x	regs

	regs.x.flags		regs.x	regs
	regs.x.flags		regs.x	regs

C'de kesme çağırabilmek için INT makina kodunu dogrudan kullanmak yerine kesme çağıran bir fonksiyon kullanılır.En önemli kesme çağıran fonksiyon int86'dır.

int86 fonksiyonunun prototipi:

int int86(int intno, union REGS *inregs, union REGS *outregs);

Bu fonksiyon ikinci parametresiyle belirtilen adresteki register bilgilerini cpu register'larına kopyalayarak birinci parametresiyle belirtilen kesmeyi çağırır. Kesmenin çalışması bittikten snra cpu register'larının içerisindeki değerleri tekrar üçüncü parametresiyle belirtilen bellek bolgesine kopyalar.Fonksiyon başarı durumunda 0 degerine basarısızlık durumunda 0 dısı bir degere geri doner.Bir kesmenin basarısız olması soz konusu olmadıgı için geri donus degerinin pek bir anlamı yoktur.

KESMELERİN İŞLEVLERİ VE SINIFLANDIRILMASI

Kesmeler gercekte makina dilinde yazılmıs bir cesit fonksiyonlardır. Cagırılması için bir programlama diline gereksinim yoktur makina dilinde cagırılabilir.Kesmeler işlevleri,ne gore intel sisteminde 3 kısımda incelenebilir.

1-)BIOS(basic input-output system) kesmeleri
2-)DOS kesmeleri
3-)Özel kesmeler

BIOS KESMELERİ

1 mb adres alanının son 64 k'sı olan(F0000-FFFFF) EPROM içerisinde bulunan makina dilinde yazılmıs kodlardır.EPROM içerisindeki kesme kodlarının bulundugu bolgeye BIOS denir.Kodu burda bulunan kesmeler çeşitli kontrol kartlarının programlayarak işletim sisteminden bagımsız cok temel işlemleri gercekleştirmektedir.BIOS kesmeleri video, aşağı seviyeli disk işlemleri, klavye işlemleri gibi işletim sistemine bile gereksinim duymayan çok temel işlemleri gercekleştirmektedir.

DOS KESMELERİ

DOS'un bellege yuklenmesiyle yaratılırlar, butun DOS işlemleri birtakım DOS kesmelerinin cagırılmasıyla saglanmaktadır.21h numaralı kesme DOS'un temel butun fonksiyonlarını yerine getirmektedir.

OZEL KESMELER

Ozel bazı programların bellege yukledigi kesme kodlarıdır.Ornegin mouse işlemleri için 33h kesmesi kullanılır(dos'ta).Ancak mouse.com programının yuklenmesiyle bu kesme kodları bellege yuklenir.

Toplam kesme sayısı 256 olmasına karsın bir kesme koduna gectikten sonra o kod içersinde ismine fonksiyon denilen alt kodlar bulunabilir.Fonksiyonlar da alt fonksiyonlara ayrılabilirler. Genellikle fonksiyon numaraları AH alt fonksiyon numaraları da AL register'ının içersinde kesme çağırılmadan once yazılır.Bazı kesmelerin cok sayıda fonksiyon ve alt kesmeleri oldugu halde bazı kesmelerin hiç fonksiyonu ya da alt fonksiyonları yoktur.Bir kesme öğrenebilmek için:

1-)Kesmenin ne iş yaptıgını bilmek gerekir.
2-)Kesmenin numarasını, fonksiyon ve alt fonksiyon numarasını öğrenmek gerekir.
3-)Kesmenin parametrelerinin neler oldugu ve hangi register'lara yerleştirilmesi gerektigi ogrenilmelidir.
4-)Geri donus degerinin ne anlama geldigi ve nerelere yerlestirileceginin bilinmesi gerekir.

BAZI BIOS KESMELERİNDEN ÖRNEKLER

10h KESMESİ

10h kesmesinin fonksiyonları video işlemlerini yapar.

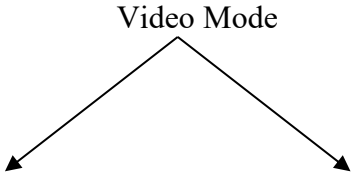
VIDEO KARTLARININ EVRİMİ

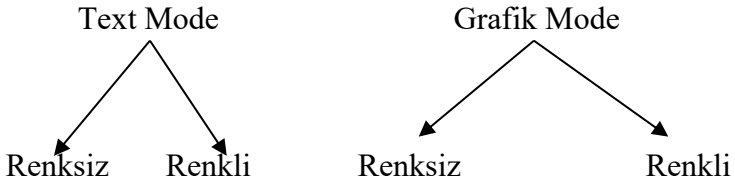
Kullanılan ilk grafik kartı MDA idi.Bu kartlar grafik gösteremiyordu.Aynı zamanda bu kartlar renksizdi.Daha sonra grafik calısmayı saglayabilen 4 renkli CGA kartları(320x280) tasarlandı.EGA kartlarıyla birlikte çözünürlük 640x480'e aynı anda gosterilen renk sayısı da 16'ya yukseltilmiştir.Daha sonra VGa kartlarıyla aynı anda verilebilen renk sayısı 256'ya yukseltilmiştir.Bugun SVGA kartlarında aynı anda 2^16, 2^24 renk görüntülenebilmektedir.EGA da dahil olmak uzere eskiden ekran, ekran kartına 9 pin'lik seri bir baglantı kablosuyla baglanıyordu.VGA kartlarla beraber kart ile monitor arasındaki bilgi aktarımı analog sisteme donusturulmustur.Bugun kullanılan monitörlere RGB(red-green-blue) monitor denir. Bu monitörlerde 3 tane elektron tabancası ile her türlü renk elde edilmektedir.

VIDEO MODE KAVRAMI

VIDEO MODE NEDİR?

Grafik kartı ya da ekran kartının belirli çözünürlük ve renk özelliklerinin etkin duruma getirildiği bir çalışma modudur.Video modun iki bileşeni vardır:Birincisi çözünürlüktür.İkincisi video modun aynı anda gösterebileceği toplam renk sayısıdır.Video mod gerçekte bu çalışma biçimini anlatan 0-255 arası bir sayıdır.





Text mode'da ekrana basılabilecek en kucuk birim bir karakterdir.Ve karakterler bir kalıp olarak basılabilir.Oysa grafik mode'da en kucuk birim 1 noktadır.Bu noktaya pixel denir.
Monitör ve video mode durumuna göre elde edilebilecek görüntü sistemi:

Video mod	Monitör	Sonuç
Renksiz	Renksiz	Renksiz
Renksiz	Renkli	Renksiz
Renkli	Renksiz	Gri tonlar
Renkli	Renkli	Renkli

VİDEO MODUN DEĞİŞTİRİLMESİ

10h,F:0
parametreler
AH->0
AL->video mode

80*25 video mode ilişkin 3 tane mode vardır.

video mode'lar

Video Mode	Çözünürlük	Text/Grafik	Renk sayısı
0	40*25	text	mono
1	40*25	text	16
2	80*25	text	mono
3	80*25	text	16(default olan mode)
4	320*200	grafik	4
5	320*200	grafik	mono
6	640*200	grafik	2
7	80*25	text	mono
8-12	reserved	------	----
13	320*200	grafik	16
14	640*200	grafik	16
15	640*350	grafik	mono
16	640*350	grafik	16
17	640*480	grafik	mono
18	640*480	grafik	16

CURSOR'UN KONUMLANDIRMASI

int 10h

F:2 AH->2

parametreler: DH->satır no
DL->sutun no
BH->0 konulacak sayfa numarası
geri dönüş değeri yok

bu kesme 0,0 orijininden başlar ve 0,0 sol üst köşedir.[24,79 sağ alt noktadır.]
cursorun görünmememsini sağlamak için tek yol menzil dışına çıkartmaktır. örneğin 25*80 de cursor görünmez.

GRAFİK MODDA EKRANA BİR PİXEL BASILMASI

INT 10h
F:0c

AH:0c
AL:color
bh:0
cx:col
dx:row

geri dönüş değeri yok

DİSK İŞLEMLERİ

Bellek nedir?

Bilgilerin saklandığı birimleri bellek denir.İkiye ayrılırlar:

1-Ara bellek (RAM ve ROM)
RAM(random access memory)
Rom(read only memory)
EPROM
EEPROM(electrically erasable memory)(Bilgi yazarken yavaş)
Yarı iletken teknoloji, entegre devre.
2-İkicil bellkeler(Disk, Floppy disk, Cd-rom, Magnetic tape..)
İkincil bellekler güç gereksinimi olmadan bilgileri tutmak için kullanılırlar.
Disk ve floppy disk'lerin çalışmaları birbirlerine cok benzerdir. Temel kavramları aynıdır. Bu nedenle disk dendiğinde aynı şey anlaşılmalıdır.

Disk İle İlgili Temel Kavramlar

Disklerde bir okuyucu kafa vardır, bu kafa yüzeye çok yakın bir biçimde step motor vasıtasıyla hareket eder. Disk bir motor taraında döndürülmektedir. Bilgiler disk üzerinde dairesel yollara yazılırlar, bu yollara track denir. Sabit disklerde bir çubku üzerine monte edilmiş birden fazla yüzey(platter) bulunabilir. Her yüzeyden okuma yapabilmek için bir okuma kaası gereklidir.

Yüzey sayısı eşittir kafa sayıdır. Disk ile cpu arasında doğrudan bir bağlantı olduğu düşünülmemelidir, disk işlemlerinin yürütülmesi yerel bir işlemci tarafından yapılır. CPU bu işlemciyi programlar işlemci disk mekaniğini hareket ettirerek bilginin disk'ten okunamasını ya da disk'e yazılmasını sağlar, eğer diskten okuma söz konusuysa elde edilen bilgi ismine DMA denilen başka bir işlemci tarafından belleğe aktarılır. Yani disk ile bellek arasındaki transfer işlemi CPU tarafından yapılmaz.

(Şekil 1)

Disk aynı zamanda mantıksal olarak sektör denilen pasta dilimlerine ayrılmıştır. Bu dilimlere sektör dilimi denir. Bir tarck'in sektör dilimi içerisinde kalan parçasına sektör denir. Bir sektör transer edilebilecek en küçük disk birimidir. Bir sektörüm kaç byte'tan oluşacağı formatlama sırasında belirlenebilir. Bu değer birkaç değer arasından seçilmektedir. Ancak DOS gibi pek çok işletim sistemi bir sektör=512 byte olarak format yapmaktadır. Sektörlerin disk üzerindeki yeri ne olursa olsun her sektörde eşit sayıda bilgi vardır. Bir disk'in formatlandıktan sonraki toplam kapasitesi şu çarpım sonucu bulunabilir:

kapasite=yüzey sayısı * track sayısı * sektör dilim sayısı * 512 (=byte)

Disketlerin Parametrik Bilgileri

1-Küçük disket(3.5")
1-a)HD =2(kafa sayısı) * 80 (track) * 18 (sektör dilimi) * 512 =1.44 Mb
1-b)DD =2 * 80 * 9 * 512 = 720 Kb
2-Büyük disket(5.25")
2-a)HD =2 * 80 * 15 * 512 = 1.2 Mb
2-b)DD =2 * 40 * 9 * 512 = 360 Kb

Teknolojik gelişim = 2-b, 1-b, 1-a, 2-a

Donanım seviyesinde en temel komut bir sektörün okunması ya da yazılmasına ilişkindir. Disk'i yöneten işlemci okunacak sektörün koordinatlarını üç parametreyle alır. Hangi yüzey, hangi track, hangi sektör dilimi. Bir sektörün yerinin bu üç parametreyle belirtilmesine fiziksel koordinat sistemi denir. Donanım fiziksel kooordinat sisteminden anlamaktadır. Yüzey sayısı 0'dan başlayar. Disketlerde 0 ön yüz 1 arka yüzdür. Track numarası da 0'dan başlar n'e kadar gider. En dıştaki track 0 numaralı track'tir. Sektör dilimi 1'den başlar. Bir sektörün sektör diliminin numarası ardışıl gitmek zorunda değildir ve hangi sektörün 1 numaralı sektör olduğu disk üzerinde sektör gap denilen bölgeye yazılmaktadır.

Disk Okuma Ve Yazma Zamanları

Bir sektörün okunma ya da yazılma emrinin verilmesiyle bu işlemin gerçekleştirilmesi arasında geçen zamandır. Disk işlemcisinin ardışıl sektör okuma ve yazma komutları vardır. Bu komutlarla ardışıl sektörler daha hızlı bir biçimde okunup yazılabilirler. Ancak bir hamlede track başına düşen sektör dilimi sayısından daha fazla sektör transfer edilemez.

Disk İle Bellek Arasında Transfer İşlemi

En düşük disk işlemi disk ile bellek arasında n ardışıl sektörün transfer edilmesi işlemidir. Bir sektörün transfer edilebilmesi için disk ve DMA işlemcilerinin programlanması gerekir. Ancak kolaylık olsun diye aşağı seviyeli disk işlemleri BIOS kesmesi biçiminde EPROM içerisine yazılmıştır. 13h kesmesinin tüm fonksiyonları aşağı seviyeli disk işlemleri için kullanılmaktadır. Ancak C derleyicileri 13h kesmesini çağırarak transfer işlemini yapan özel bir fonksiyonda barındırmaktadır. Bu fonksiyon Borland derleyicilerinde biosdisk Microsoft derleyicilerinde _biosdisk olarak adlandırılır.

biosdisk Fonksiyonunun Parametrelerinin İncelenmesi

int biosdisk(int cmd, int drive, int head, int track, int sector, int nsects, void *buf);

Fonksiyonun prototipi bios.h içerisindedir.

Parametreleri:

cmd=>	13h'ın fonksiyon numarasıdır. 2 nolu fonksiyon sektör okumak içindir,3 nolu fonksiyon sektör'e yazmak içindir.
drive=>	Hangi sürücüye ilişkin sektör okunacaktır. 0=>A, 1=>B, 0x80(128)=>birinci HDD, 0x81(129)=>ikinci HDD
head, track, sector=>	Okunacak sektörün fiziksel koordinatları
nsects=>	Ardışıl kaç sektör okunacağıdır. En az 1 en fazla sektör dilim sayısı kadardır.
buf=>	Bu parametre bellek transfer adresini belirtir. Kuşkusuz okuma durumunda bu adresten başlayan n byte bölgenin dizi tanımlamasıyla ya da dinamik bellek yönetimiyle tahsis edilmiş olması gerekir.

Fonksiyonun geri dönüş değeri işlem başaılıysa 0, başarısızsa 0 dışı bir değerdir. Bu 0 dışı değer başarısızlığın nedeni hakkında bilgi de vermektedir. Bu fonksiyon ile başarısızlığın tescillenmesi için en az 3 kere deneme yapılması gerekir. Çünkü işlemin baarısı biraz da mekanik etkenlere bağlıdır.

Üç kere deneme algoritması:

```
#include <stdio.h>
#include <stdlib.h>
#include <bios.h>

#define TRYNUM      3

void main(void)
{
```

```
    int result;
    int i;

    for (i = 0; i < TRYNUM,; ++i) {
        result = biosdisk( ........);
        if(!result)
            break;
    }
    if(result)
        printf("cannot read or write disk...\n");
        exit(1);
}
```

Örnek:Bir sektörün ekranda görüntülenmesi.

```
--------------biosdisk.c--------------
#include <stdio.h>
#include <stdlib.h>
#include <bios.h>

#define TRYNUM          3
#define SECTOR_SIZE     512

typedef unsigned char BYTE;

void main(void)
{
    BYTE buf[SECTOR_SIZE];
    int i, result;

    for (i = 0; i < TRYNUM; ++i) {
        result = biosdisk(2, 0, 0, 0, 1, 1, buf);
        if(!result)
            break;
    }
    if(result)
        printf("cannot read or write disk...\n");
        exit(1);
    for (i = 0; i < 128; ++i)
        printf("%02%c", buf[i], i % 32 == 31 ? '\n' : ' ');
}
-----------------------------------
```

SINIF ÇALIŞMASI

Bir yazının 1.44 formatlı bir diskette aranması:

norton disk edit ile bu ulaşımı kolaylıkla yapılabiliyor.
dos altında çok biyik yerel diziler açmamak gerekir çünkü steak sistemi 64 k ile sınırlıdır ve karmaşık programlarda steak problemi ile karşılaşılabilir.
bir byte dizisi içerisinde başka bir byte dizisinin aranması için strstr fonksiyonunu kullanamayız. çünkü bu fonksiyon null karakteri görünce işlemi keser. bu durumda bu fonksiyonun yaptığını yapacak başka bir fonksiyon yazmak gerekir. bu fonksiyonu yazarken strncmp fonksiyonu değil memcmp fonksiyonunu kullanmalıyız.

int memcmp(const void *s1, const void *s2, unsigned n)

bu fonksiyon biirinci adreste bulunan bilgi ile ikinci adreste bulunan bilginin ilk n byte ını karşılaştırır. eğer iki bilgi birbirine eşitse 0 değerine; 1>2 ise pozitif bir değere; 2>1 ise negatif herhangi bir değere geri döner.

----------search.c----------

```
#include <stdio.h>
#include <bios.h>
#include <string.h>
#include <stdlib.h>

/*symbolic constants*/
#define NTRACK      80
#define NHEAD       2
#define NSECTOR     18
#define SECTSIZE    512
#define NTRY        3


/*typedefs*/
typedef unsigned char BYTE;



/*function prototypes*/
int sfindstr(const BYTE *buf, const char *str, int nsect, int *last);

int sfindstr(const BYTE *buf, const char *str, int nsect, int *plast)
{
   int i;
   int len;

   len = strlen(str);
   if (*plast)
      if (!memcmp(buf, str + *plast, len - *plast))
         return 1;
      else
```

```
            *plast = 0;
    for(i = 0; i < nsect * SECTSIZE - len + 1; ++i)
        if (!memcmp(&buf[i], str, len))
            return i / 512 + 1;
    for (i = nsect * SECTSIZE - len; i < nsect * SECTSIZE; ++i) {
        if (!memcmp(&buf[i], str, --len)) {
            *plast = len + 1;
            return 0;
        }
    }
    return 0;
}


void main(int argc, char *argv[])
{
    int h, t;
    int result;
    int last = 0;
    int i;
    BYTE *buf;

    if (argc == 1) {
        printf("Disket search utility version 1.0 \n");
        printf("Usage : search <string> \n");
        exit(1);
    }
    if (argc > 2) {
        printf("Too many parameters....\n");
        exit(1);
    }
    buf = (BYTE *) malloc (SECTSIZE * NSECTOR);
    if(buf == NULL) {
        printf("cannot allocate memory....\n");
        exit(1);
    }
    for (t = 0; t < NTRACK; ++t) {
        for (h = 0; h < NHEAD; ++h){
            for (i = 0; i < NTRY; ++i){
                result = biosdisk(2, 0, h, t, 1, NSECTOR, buf);
                if(!result)
                    break;
            }
            if(result){
                printf("Cannot read track:%d\n",t);
                exit(1);
            }
```

```
            printf("Head: %d Track: %d\r", h, t);
            result = sfindstr(buf, argv[1], NSECTOR, &last);
            if(result){
               printf(" string found : Head = %d track = %d sects = %d\n", h, t, result);
               exit(0);
            }
         }
      }
      printf("could not found string ...\n");
}
```

DOSYA SİSTEMİ KAVRAMI VE DOSYALAR

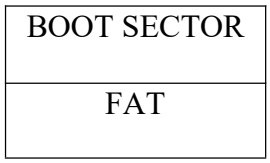
Dosya Nedir?

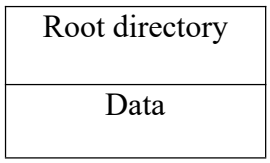
Dosya donanımla ilişkisi olmayan genel bir terimdir. İçerisinde bilgilerin bulunduğu disk alanlarına dosya denir. Bir dosyanın parçaları nihayet sektörler içerisinde bulunmaktadır. Dosyanın parçalarınn sektörlere nasıl yerleştirildiğini dosyaların nasıl ele alınıp işleme sokulduğu, dosyaların hangi özelliklere sahip olduğu işletim sisteminden işletim sistemine göre değişen bir bilgidir. İşletim sistemlerinin dosya işlelerine ilişkin belirlemelerine ve kodlarına dosya sistemi(file system) denir. Dosya kavramsal olarak ardışıl byte bilgilerden oluşmaktadır. Oysa gerçekte disk üzerinde çeşitli sektörlere yayılmış bir biçimde bulunur. İşletim sistemlerinin birbirlerine benzer ya da farklı dosya sistemleri olabilir. DOS ve WIN 3.x'in dosya sistemleri tamamen aynıdır, bu dosya sisteine genellikle FAT sistemi denir. WIN 95 ve 98'in dosya sistemleri DOS'a çok benzer, bu sistemlere çeşitli küçük ekler yapılmıştır. Örneğin uzun dosya isimleri ve 32 bit FAT algoritması gibi... WIN NT'nin dosya sistemine NTFS denir, DOS'un dosya sistemine benzemekle birlikte farklılıkları vardır. UNIX'in dosya sistemi diğer gruptan tamamen farklıdır.

DOS İŞLETİM SİSTEMİNİN DOSYA SİSTEMİ

DOS İŞLETİM SİSTEMİNİN DOSYA SİSTEMİ

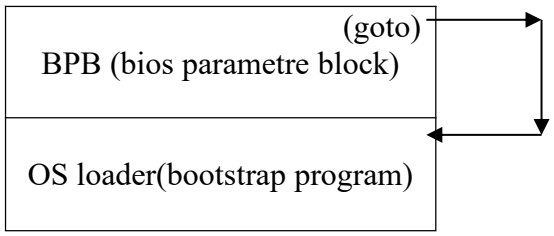
DOS'un dosya sistemi Windows 3.1'inkiyle tamamen aynıdır. WIN95/98 sistemlerinin dosya sistemleriyse DOS'unkinden çok az farklıdır(Uzun dosya isimleri ve 32 bit FAT sistemi). İlgili partion'ın herhangi bir işletim sisteminin dosya sisemine uydurulması için o sistemde formatlama işlemine sokulması gerekir. DOS'un dosya sisteminde bir disk 4 bölümden oluşmaktadır.





BOOT SECTOR

BOOT SECTOR diskin çeşitli parametrik bilgilerinin tutulduğu ve sistemi yükleyen programın saklandığı 512 byte uzunluğunda bir sektördür. BOOT SECTOR disetlerde sıfırıncı head sıfırıncı track ve birinci sektöründedir(0h:0t:1s). Yani disketin ilk sektörüdür. Disklerde BOOT SECTOR ilgili disk bölmesinin (partition) mantıksal ilk setörüdür. BOOT SECTOR içerik bakımından iki bölümden oluşur. BOOT SECTOR'ün düşük anlamlı byte'larına bios parametre block(BPB) denir, diğer bölüm işletim sistemini yükleyecek yükleyici programın bulunduğu bölümdür.



BPB Bölümü

BPB bölümünde diskin kullanımına ilişkin önemli parametrik bilgiler bulunmaktadır. BPB bölümünün içeriği şöyledir.

Offset	Uzunluk	Anlamı
0(0)	3 byte	JMP code
3(3)	8 byte	OEM name and version
B(11)	1 word	Sektördeki byte sayısı
D(13)	1 byte	Clusterdaki sektör sayısı
E(14)	2 byte	Ayrılmış sektörler
10(16)	1 byte	FAT kopyalarının sayısı
11(17)	1 word	Root directory girişleri sayısı
13(19)	1 word	Toplam sektör sayısı
15(21)	1 byte	Ortam belirleyicisi
16(22)	1 word	FAT'in kaç sektör olduğu
18(24)	1 word	Track'teki sektör dilimi sayısı
1A(26)	1 word	Kafa sayısı
1C(28)	1 word	Saklı sektörlerin sayısı

JMP code: İşletim sistemini yükleyen program BPB bloğunun ötesinde bulunmaktadır. Yükleme işleminin gerçekleşmesi için BPB bloğunun bir çeşit GOTO komutuyla atlanması gerekir. Burada 3 byte uzunluğunda bir GOTO komutu bulunmaktadır. Intel işlemcilerinde GOTO komutuna JMP denir.

OEM name: Buraya bilgi format programı tarafından yorum amaçlı olarak yazılmaktadır. Genellikle diskin hangi işletim siseminin hangi version'ı tarafından formatlandığını belirlemek için kullanılır. Buradaki bilginin değiştirilmesi hiçbir probleme yol açmaz.

Sektördeki byte sayısı: Burada sektör başına düşen byte sayısı bulunmaktadır. Madem bir sektör 512 byte uzunluğa ayarlanmaktadır o halde buradaki bilgi de 00 02 biçiminde bulunmalıdır.

Cluster kavramı: DOS'ta dosyanın parçaları data bölümüne rastgele yerleştirilir. Bir cluster dosyanın saklanabileceği en küçük disk birimidir. Bir cluster ardışıl n sektörden olumak zorundadır(2'nin kuvveti olmak zorundadır).

Yani DOS depolama birimi olarak sektör değil cluster düşünür(dosyanın parçalarını rastgele clusterlar içerisinde tutulmaktadır).

Cluster Sisteminin Avantajları ve Dezavantajları:

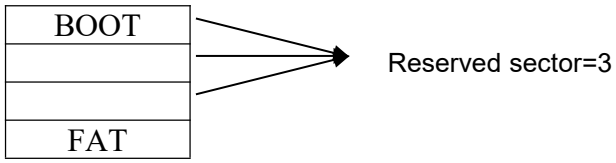
1-)Cluster sistemiyle dosyanın disk üzerindeki parçaları azaltılmaktadır. Bu da diskin kafa hareketinin azaltılması anlamına gelir.
2-)Dosyanın parçalarının yerleri FAt denilen bir bölgede tutulmaktadır. Eğer cluster kavramı olmasaydı bu FAt bölgesi daha büyük bir yer kaplardı.
3-)Her dosyanın son cluster'ında kullanılmayan bölümler oluşur. Buna "internal fragmentation" (içsel bölünme) denir(Diskin kullanım oranını düşürür).

Sürekli Ve Rastgele Yerleşim Sistemleri

Diske ya da belleğe yüklemeler ardışıl bir biçimde yapılsadı belirli bir süre yükleme boşaltma işlemlerinden sonra kullanılamayan küçük boş bölgeler oluşurdu. Örneğin boş bölgelerin toplamı çok büyük olabildiği halde ardışıl olarak bulunmadığı için tahsis edilemezdi. Oysa ardışıl tahsis etme zorunluluğu ortadan kaldırılarak bu küçük parçalar da tahsisatın bir parçası haline getirirlebilmektedir. Buna rastegele yerleşim denir. Ardışıl yerleşimdeki küçük kullanılmayan bölümler oluşmasına bölünme durumu(fragmentation) denilmektedir. Rastgele yerleşimde disk ya da RAM belirli uzunluktaki bloklara ayrılır ve tahsis edilecek birim bu bloklara rastgele atanır. Tabii hangi bloklarda hangi tahsisat parçasının bulunduğu ayrı bir tabloda tutulmak zorundadır. Ancak rastgele yerleşim sisteminde bölünme durumu tamamen giderilemez. Her bloğun sonunda kullanılmayan küçük boş alanlar kalır, bu durmuna içsel bölünme durumu(internal fragmentation) denir.Peki bir blok ya da bir cluster ne kadar uzunlukta olmalıdır? Bir bloğun uzunluğu çeşitli amaçlar göz önünde bulundurularak ayarlanabilir. Intel işlemcilerinde bu bloklar 4 k uzunluğundadır ve değişitirilemez. DOS işletim sistemi disk'in toplam uzunluğuna bakarak bir cluster'ın kaç sektör olacağını otomatik bir biçimde belirler. Bazı tür UNIX sistemlerinde bir bloğun kaç sektör olacağı yükleyen kişiye default değeri bildirilerek

sorulur. 1.44 3.5" disketlerde 1 cluster=1 sektör 720 k 3.5" disketlerde 1 cluster=2 sektördür. Sonuçta bir clusterın kaç sektör olacağı formatlama sırasında otomatik olarak belirlenmektedir.

Ayrılmış sektörler:BOOT SECTOR içerisindeki yükleyici program büyük ise FAT bölgesi BOOT SECTOR'den hemen sonra başlamaz. Reserved sector sayısı BOOT SECTOR ve FAT bölgesine kadar uzatılmış bölgenin aç sektör uzunlugunda olduğunu gösterir. FAT hemen BOOT SECTOR'den sonra başlıyorsa reserved sector 1 değerindedir. Reserved sector sayısı FAT öncesindeki sektör sayısını belirtmektedir.



Bu durumda FAT bölgesinin başlangıcı reservered sector alanında yazılan sektördedir.

FAT kopyalarının sayısı: FAT bölgesi istenildiği kadar kopyadan oluşabilir. DOS standart olarak 2 kopya FAT tutmaktadır. FAT'in birden fazla kopyasının bulundurulması ciddi anlamda bir fayda sağlamamaktadır. Çünkü FAT bölgesinin güncellenmesi her iki kopya üzerinde aynı anda yapılmaktadır.

Root directory girişleri sayısı:Root directory'deki dosya bilgileri root directory alanında saklıdır. Bir dosya bilgisi 32 byte yer kaplar. Yaani burada yazılan sayı 32 ile çarpılır ve 512'ye bölünürse root directory bölgesinin kaç sektör uzunluğunda oludğu bulunur. Root bölgesinin sektör uzunluğu data bölgesinin yerini tespit etmek amacıyla kullanılır.

Toplam sektör sayısı: Burada diskteki toplam sektör sayısı bulunmaktadır. Malesef DOS 4.01 version'ına kadar bu bölge 2 byte olarak küçük bir biçimde tasarlanmıştır. Buraya yazılabilecek en büyük sayı 65535'dır. 65525*512=33 Mb'tır. DOS 4.01'e kadar en büyük DOS partition'ı 33 mb olabilir. Ancak bu alan yetersiz kaldığı için DOS 4.01 version'ında BPB bloğu büyütülmüş ve 20h offsetinde 4 byte uzunluğunda yeni bir toplam sektör sayısı alanı tanımlanmıştır. Bugünkü DOS version'ları önce eski 13h offsetindeki eski word alanına bakarlar, eğer buradaki sayı 0 değilse toplam sektör sayısının bu alanda yazılan olduğunu düşünürler. Eğer buradaki sayı 0 ise 20h offsetinden 4 byte'lık değeri okurlar.

Ortam belirleyicisi: Buradaki 1 byte'lık değer genel olarak disk birimi hakkında bilgi vermektedir. Buraddaki değerler ve anlamları dağıtılan notlarda belirtilmiştir.(2.37(disk id byte))

FAT'in kaç sektör olduğu: Burada bir FAT'te FAT'in kaç sektör bilgisinden oluştuğu bilgisi yer almaktadır. Bu durumda ROOT DIRECTORY bölgesinin hangi sektörden başlayacağı FAT kopyalarının sayısının bu değeriyle çarpımı sonucu hesaplanabilir.

Track'teki sektör dilimi sayısı: Bir track'te kaç sektör dilimi oldğu bilgisidir.

Kafa sayısı: Diskin kaç yüzeyden oluştuğu bilgisini verir.

Saklı sektörlerin sayısı: Saklı sektör kullanılmayan sektör olarak düşünülmüştür. Ancak bu değer genellikle 0'dır Saklı sektörlerin ne amaçla kullanıldığı kesin olarak bilinmemektedir.

İŞLETİM SİSTEMLERİNİN OTOMATİK OLARAK YÜKLENMESİ

Bilgisayar açıldığında mikro işlemci reset edilir ve çalışma otomatik olarak 1 Mb bellek alanının son 64 kb'si içerisinde bulunan EPROM bölgesinden başlar. EPROM, board üzerinde ayrı bir bölgede bulunur. Ancak işlemci o adres bölgesine erişmeye çalıştığında SIMM(single inline memory module) modülüne değil EPROM modülüne yönlendirilmektedir. EPROM'daki program bilgisayara bağlı olan aygıtların belirlenmesi ve test edilmesi işlemlerini yapar. Örneğin floppy sürücü kontrol edilir, ne kadar fiziksel RAM olduğu test edilir vb... Bugün bu self testin yanı sıra BIOS programı aynı zamanda PNP (plug and play) testi de yapmaktadır. Yani bilgisayara takılı olan bir kart ya da bir chip PNP teknlojisine uygun olarak yapılmışsa dışarıdan bu kartın hangi amaçlarla bilgisayara takıldığı, bilgisayarın hangi kaynaklarını kullandığı tespit edilebilir. PNP kartları kaynak kullanım parametreleri dışarıdan programlama yoluyla değiştirilecek biçimde tasarlanmıştır. İşletim sistemi gerekirse çeşitli çakışmaları çözebilir. Self-test sonrasında elde edilen bilgiler bir biçimde belirli bir yerde saklanmaktadır. Self-test sonrasında ismine ROM yükleyisi programı denilen(rom bootstrap program) özel bir program çalışmaktadır. Buradaki program sırasıyla A ve C disklerine bakar, oranın boot sektörünü belleğin 7C00 bölgesine yükleyerek kontrolü boot sektördeki programa bırakır. Yani işletim sistemi gerçekte boot sektördeki program sayesinde yüklenmektedir. Yani bütün işletim sistemleri boot sektör üzerindeki program tarafından yüklenmektedir.

Mantıksal Sektör Kavramı

Fiziksel olarak bir sektör h:t:s parametreleriyle belirtilmektedir. Oysa DOS işletim sisteminde bir sektörün yerini test etmek için 3 parametreli sistem yerine ismine mantıksal sektör denilen(mutlak sektör=absolute sector) tek parametreli bir koordinat sistemi kullanılmaktadır. Bu koordinat sistemine göre tüm sektörlere 0'dan başlayarak ardışıl birer sayı karşılık getirilmektedir. Önce sektör dilimleri daha sonra kafalar daha sonra trackler değiştirilerek her sektöre ardışıl bir numara verilmiştir. Örneğin bir disketeki numaralandırma şöyledir.

Fiziksel sektör	Mantıksal sektör
0:0:1	0
0:0:2	1
.....	…..

.....	…..
0:0:18	17
1:0:1	18
.....	…..
.....	…..
1:0:18	35
0:1:1	36
.....	…..
.....	…..

Bu koordinat sisteminin amacı parametre sayısını azaltmaktır. Bu durumda eğer diskin parametreleri bilinirse (BPB alanında yazmaktadır) mantıksal sektörle fiziksel sektör arasında dönüşüm kolaylıkla yapılabilir. DOS işletim sisteminin absread ve abswrite isimli iki sistem fonksiyonu vardır. Bu fonksiyonlar parametre olarak mantıksal sektör numarasını alırlar kendi içlerinde fiziksel koordinat sistemine dönştürerek okuma/yazma işlemini gerçekleştirirler.

absread ve abswrite FONKSİYONLARININ biosdisk FONKSİYONUNDAN FARKLARI

1-Bu fonksiyon kendi içerisinde 3 kere denemeyi yapmaktadır.
2-absread ve abswrite fonksiyonları ile istenilen sayıda sektör okunabilir.
3-Bu fonksiyonlar partition duyarlıdır. Yani her DOS partition'ının ilk sektörünü 0 kabul ederek işlem yaparlar.

absread abswrite Fonksiyonları

Prototipleri:

```
int absread(int drive, int nsects, long lsect, void *buf);
int abswrite(int drive, int nsects, long lsect, void *buf);
```

prototipi dos.h içerisindedir.

drive: 0=>A
1=>B
2=>C
3=>D
4=>E

nsect: Ardışıl kaç sektör okunacağıdır.

lsect: Hangi mantıksal sektörden başlanacağıdır.

buf: Transfer adresidir.

Bu fonksiyonların geri dönüş değeri başarılıyla 0 başarısızsa -1 değerine geri döner. Hatanın sebebini -1 değerine geri dönüldüyse "errno" global değişkeninin içerisine yazar.

Sınıf çalışması:BPB bloğunu okuyarak disk parametre bilgilerini ekrana yazdıran program. Bunun için önce BPB alanı bir yapı içerisine kopyalanır ve o yapıdan çekilerek kullanılır. Bu işlem için bootinfo isminde bir fonksiyon yazılacak ve bu fonksiyon yazılarak işlemler yapılacak.

```
/*------------disk.h-------------------*/
/*-------------------------------------------------------------------------
Disk Header File
-------------------------------------------------------------------------*/

#ifndef _DISK_H_

/* Symbolic contants */

#define SECT_SIZE   512

/* typedefs */

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned long int DWORD;

/* Boot sector */

typedef struct _BOOT {
   WORD fat_len;      /*fat length as sector (A)*/
   WORD root_len;     /*root length as sector*/
   WORD fat_num;      /*number of fat(A)*/
   DWORD total_sec;   /*total sector (A)*/
   WORD bps;          /*byte per sector(A)*/
   WORD spc;          /*sector per cluster(A)*/
   WORD reserved_sect; /*reserved sector(A)*/
   BYTE media_descriptor;/*media descriptor byte(A)*/
   WORD spt;          /*sector per track(A)*/
   WORD root_ent;     /*root entry(A)*/
   WORD head_num;     /*number of heads(A)*/
   WORD hid_num;      /*number of hidden sector(A)*/
   WORD tph;          /*track per head*/
   WORD fat_or;       /*fat director location*/
   WORD root_or;      /*root directory location*/
   WORD data_or;      /*first data sector location*/
} BOOT;

/* Function Prototypes */

int boot_info(BOOT *info, int drive);
```

```
#endif

/*---------------------------------------*/

/*-------------bootinfo.c-----------------*/
/*-----------------------------------------------------------------------
Disk Implementatin File
-----------------------------------------------------------------------*/
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <ctype.h>
#include "disk.h"

/* Functions */

int boot_info(BOOT *info, int drive)
{
   BYTE btr[SECT_SIZE];

   if (absread(drive, 1, 0, btr) == -1)
      return -1;
   info -> fat_len = *(WORD *)(btr + 0x16);
   info -> root_len = *(WORD *)(btr + 0x11) * 32 / 512;
   info -> fat_num = *(BYTE *)(btr + 0x10);
   if (*(WORD *)(btr + 0x13))
      info -> total_sec = *(WORD *)(btr + 0x13);
   else
      info -> total_sec = *(DWORD *)(btr + 0x20);
   info -> bps = *(WORD *)(btr + 0xB);
   info -> spc = *(BYTE *) (btr +0xD);
   info -> reserved_sect = *(WORD *)(btr + 0xE);
   info -> media_descriptor = *(btr + 0x15);
   info -> spt = *(WORD *)(btr + 0x18);
   info -> root_ent = *(WORD *)(btr + 0x11);
   info -> head_num = *(WORD *)(btr + 0x1A);
   info ->hid_num = *(WORD *)(btr + 0x1C);
   info -> tph = (WORD) (info -> total_sec / info -> spt / info -> head_num);
   info -> fat_or = info -> reserved_sect;
   info -> root_or = info -> fat_len * info -> fat_num + info -> reserved_sect;
   info -> data_or = info -> root_or + info -> root_len;
   return 0;
}


#if 1
```

```
void main(int argc, char *argv[])
{
   BOOT boot;
   int result;

   if (argc != 2) {
      printf("Wrong number of arguments...\n");
      exit(1);
   }
   if (!isalpha(argv[1][0]) || argv[1][1] != ':') {
      printf("Invalid drive: %s\n", argv[1]);
      exit(1);
   }
   printf("Ok...\n");
   result = boot_info(&boot, toupper(argv[1][0]) - 'A');
   if (result == -1) {
      printf("Cannot get boot record...\n");
      exit(1);
   }
   printf("Bytes per sector: %u\n", boot.bps);
   printf("Sector per cluster: %d\n", boot.spc);
   printf("Number of Head: %u\n", boot.head_num);
   printf("Fat origin: %u\n", boot.fat_or);
   printf("Root origin: %u\n", boot.root_or);
}




#endif

/*-----------------------------------------------*/
```

FAT BÖLGESİ

FAT bölgesi bir dosyanın parçalarının hangi cluster'larda olduğunu tutan bir bölgedir. Genellikle 2 kopyası bulunur. FAT bölgesinin hangi mantıksal bölgeden başladığı ve kaç sektör uzunlukta olduğu BOOT SECTOR'ndeki BPB bloğu incelenerek yorumlanabilir. DOS dosya sisteminde data bölgesi cluster numaralarına ayrılmıştır. Data bölgesinin ilk n sektörü iki numaralı cluster sonraki n sektörü üç numaralı cluster vesaire olarak ardışıl olarak sıralandırılmıştır.
Data bölgesi DOS'a göre sektörlerden değil cluster'lardan oluşmaktadır. İlk cluster numarası 2 olarak belirlenmiştir.

data bölgesi
```
------------------
|2  |3  |4  |5  |6  |
```

```
------------------
|7  |8  |9  |10|11|
------------------
....
...
...
----------------
```
Örneğin data bölgesinin n'incş cluster'ının hangi mantıksal sektörden başladığı
(n-2) * spc + data_or
ifadesi ile hesaplanbilir.

spc   =>sektör başına cluster sayısı
data_or=>data bölgesinin hangi mantıksal sektörden başladığı

Örneğin n'inci cluster bilgisini elde etmek için
1. bootinfo fonksiyonu ile bpb bloğu elde edilir.
2. (n - 2) * sc + data_or ifadesi ile ilgili cluster'ın hangi mantıksal sektörden başladığı bulunur.
3.absread fonksiyonu ile spc kadar sektör okunur.

Sınıf çalışması: İstenilen bir cluster bilgilerini okuyan fonksiyonun tasarlanması

Fonksiyonun parametresi:

```
int read_cluster(long cno, const BOOT *bpb, void *buf);
```

----------disk.h----------

```
/*------------------------------------------------------------------------
Disk Header File
------------------------------------------------------------------------*/

#ifndef _DISK_H_

/* Symbolic contants */

#define SECT_SIZE   512

/* typedefs */

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned long int DWORD;

/* Boot sector */

typedef struct _BOOT {
   WORD fat_len;       /*fat length as sector (A)*/
   WORD root_len;      /*root length as sector*/
```

```
   WORD fat_num;       /*number of fat(A)*/
   DWORD total_sec;    /*total sector (A)*/
   WORD bps;           /*byte per sector(A)*/
   WORD spc;           /*sector per cluster(A)*/
   WORD reserved_sect; /*reserved sector(A)*/
   BYTE media_descriptor;/*media descriptor byte(A)*/
   WORD spt;           /*sector per track(A)*/
   WORD root_ent;      /*root entry(A)*/
   WORD head_num;      /*number of heads(A)*/
   WORD hid_num;       /*number of hidden sector(A)*/
   WORD tph;           /*track per head*/
   WORD fat_or;        /*fat director location*/
   WORD root_or;       /*root directory location*/
   WORD data_or;       /*first data sector location*/
} BOOT;

/* Function Prototypes */

int boot_info(BOOT *info, int drive);

#endif
----------------------------------------

----------clust.c-------------------
/*-------------------------------------------------------------------
Disk Implementatin File
-------------------------------------------------------------------
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <ctype.h>
#include <conio.h>
#include "disk.h"


/* Functions */

int read_cluster(int drive, long cno, const BOOT *bpb, void *buf)
{
   long sect_num;

   sect_num = (cno - 2) * bpb -> spc + bpb -> data_or;
   return absread(drive, bpb -> spc, sect_num, buf);
}

int boot_info(BOOT *info, int drive)
{
```

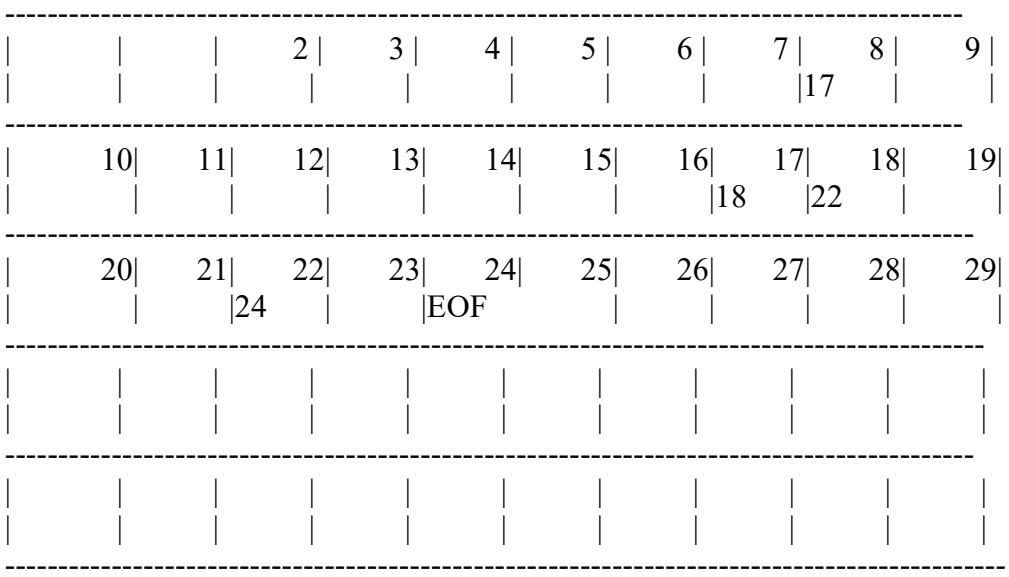
```
   BYTE btr[SECT_SIZE];

   if (absread(drive, 1, 0, btr) == -1)
      return -1;
   info -> fat_len = *(WORD *)(btr + 0x16);
   info -> root_len = *(WORD *)(btr + 0x11) * 32 / 512;
   info -> fat_num = *(BYTE *)(btr + 0x10);
   if (*(WORD *)(btr + 0x13))
      info -> total_sec = *(WORD *)(btr + 0x13);
   else
      info -> total_sec = *(DWORD *)(btr + 0x20);
   info -> bps = *(WORD *)(btr + 0xB);
   info -> spc = *(BYTE *) (btr +0xD);
   info -> reserved_sect = *(WORD *)(btr + 0xE);
   info -> media_descriptor = *(btr + 0x15);
   info -> spt = *(WORD *)(btr + 0x18);
   info -> root_ent = *(WORD *)(btr + 0x11);
   info -> head_num = *(WORD *)(btr + 0x1A);
   info ->hid_num = *(WORD *)(btr + 0x1C);
   info -> tph = (WORD) (info -> total_sec / info -> spt / info -> head_num);
   info -> fat_or = info -> reserved_sect;
   info -> root_or = info -> fat_len * info -> fat_num + info -> reserved_sect;
   info -> data_or = info -> root_or + info -> root_len;
   return 0;
}


#if 0/*
      ileride yorum satırı ile kesmek yerine preprocessor kodu ile
      cıkarabilmek için #if kullandık.ardındaki sayı 0 ise #endif'e kadar
      olan kısım kod'dan cıkarılacaktır
    */
void main(int argc, char *argv[])
{
   BOOT boot;
   int result;

   if (argc != 2) {
      printf("Wrong number of arguments...\n");
      exit(1);
   }
   if (!isalpha(argv[1][0]) || argv[1][1] != ':') {
      printf("Invalid drive: %s\n", argv[1]);
      exit(1);
   }
   printf("Ok...\n");
   result = boot_info(&boot, toupper(argv[1][0]) - 'A');
```

```
    if (result == -1) {
        printf("Cannot get boot record...\n");
        exit(1);
    }
    printf("Bytes per sector: %u\n", boot.bps);
    printf("Sector per cluster: %d\n", boot.spc);
    printf("Number of Head: %u\n", boot.head_num);
    printf("Fat origin: %u\n", boot.fat_or);
    printf("Root origin: %u\n", boot.root_or);
}

#endif

#if 1

void main(void)
{
    BOOT bpb;
    BYTE *buf;
    int i;
    int result;

    clrscr();
    result = boot_info(&bpb, 0);
    if (result == -1) {
        printf("Cannot read boot sector.. Disk unusable...\n");
        exit(1);
    }
    buf = (BYTE *)malloc(bpb.bps * bpb.spc);

    result = read_cluster(0, 10, &bpb, buf);
    if (result == -1) {
        printf("Cannot read cluster...\n");
        exit(1);
    }
    for(i = 0; i < 32; ++i)
        printf("%02x%c", buf[i], i % 16 == 15 ? '\n' : ' ');
    free(buf);
}

#endif
```

FAT BÖLGESİNİN YAPISI

FAT bölgesi cluster numaralarından oluşmuş bir bağlı liste yapısı biçiminde tasarlanmıştır. FAT bölgesi 12 bit, 16 bit, 32 bit omak üzere 3 çeşittir.

16 Bit FAT



örnek bir dosyanın FAT'teki sıralaması(8-17-18-22-24)

Önceki elemanın sonraki elamanı gösterdiği listelere bağlı liste denir.(linked list) FAT bölgesi FAT elemanlarından oluşur, bir FAT elemanı 16 FAT'lerde 16 bit 32 bit FAt'lerde 32 bit ve 12 bit FAT'lerde 12 bitten oluşur. İşletim sistemi bir dosyanın parçalarının hangi cluster'larda olduğunu şöyle bulur. Dosyanın ilk cluster'ının yerini FAt bölgesine bakmadan önce bilmek zorundadır. Her FAT elemanında bir sonraki cluster'ın hangi FAT elemanında olduğu bilgisi vardır. Zincir özel bir sayı ile bitirilir. İlk iki FAT elemanı reserve edilmiştir ve içerisinde özel bir değer vardır.

FAT elemanının içerisinde yazılan sayıların yorumlanması

Değer	Anlamı
0000	Boş
0001-FFEF	Sonraki cluster
FFF0-FFF6	Reserved cluster
FFF7	Bad cluster
FFF8-FFFF	End of file

Bir FAT elemanında 0000 varsa o cluster DOS tarafından istenildiği zaman tahsis edilebilir yani boştur.Eğer FAT elamında FFF(-FFFF arası bir değer varsa bu işlem FAT zincirinin bittiğini belitir. (Genellikle buradafi sayı FFFF biçimindedir.) Eğer FAT elemanında FFF7 değeri varsa ilgili cluster fiziksel bir kusurdan dolayı kullanılamaz bir durumdadır. FAT bölgesi formatlama işlemi sırasında 0 değerleriyle doldurulur. Ancak format programı eğer

fiziksel kusurlu sektörler tespit ederse bu sektörlere ilişkin cluster'ları "bad cluster" olarak belirler ve bu bad clusterlara ilişkin FAT elemanlarını FFF7 ile doldurur. Böylece işletim sistemi bu bölgeye yükleme yapmaz.

16 bit FAT'te işlemler

FAT bölgesinin BYTE *fat; gibi bir göstericinin içerisinden başlsadığını varsayalım. Yani fat isimli göstericinin içerisinde diskteki FAT bölgesinin bellekteki görüntüsü bulunmaktadır. Bu durumda n numaralı FAT elamını çekmek için *(WORD *)(fat + n *2) denklemi kullanılabilir. Ya da bu adres WORD türünden bir göstericiye atanıp işlemler kolalaştırılabilir:

```
WORD *fwfat;
pwfat = (WORD *)fat;
pwfat[n];
```

12 bit FAT yapısı

12 bit FAT'te 12 bit uzunluğundadır. Yani 3 hex digit yer kaplamaktadır. Bu durumda FAT bölgesi BYTE *fat; biçiminde bir gösterici tarafından belirleniyorsa n'inci FAT elemanının içerisindeki değer şu algoritmayla bulunabilir:

```
BYTE *fat;
WORD w;

if (n % 2 == 1) {
        w = *(WORD*)(fat + n * 3 / 2);
        w = w >> 4;/*w'nun düşük anlamlı 4 bitinin 0'lanması için*/
}
else {
        w = *(WORD *)(fat + n * 3 / 2);
        w = w & 0x0FFF);/*w'nun yüksek anlamlı 4 bitinin 0'lanması için*/
}
```

Eğer DOS formatlı bir diskte toplam sektör sayısı 32680'den küçükse kesinlikle12 bit FAT büyükse 16 bit FAT vardır demektir. (BPB alanında toplam sektör sayısı yaazmaktadır.)

12 bit FAT sisteminde FAT elemanının alacağı değerler

Değer	Anlam
000	Boş
001-FEF	Sonraki cluster
FF0-FF6	Reserved cluster
FF7	Bad cluster
FF8-FFF	Zincir sonu

FORMATLAMA İŞLEMİ

Formatlama işlemi iki aşamada yürütülen bir işlemdir. Birinci aşamaya aşağı seviyeli formatlama denir. Aşağı seviyeli formatlama işletim sistemine bağlı değildir. Aşağı seviyeli formatlama sırasında sektör boşlukları (sector gap) düzenlenir ve içlerine baz kritik bilgiler yazılır. Gerçekte sektör dilimleri birbirlerini izlemez, her sektör dilimiyle diğeri arasında sektör boşluğu dediğimiz bir alan vardır. 2 Mb'lik disket formatlandığında bu sebepten dolayı 1.44 Mb'a düşer. Aşağı seviyeli formatlamadan sonra işletim sisteminin disk organizasyonu için organizasyon bilgilerinin yazılması için yüksek seviyeli formatlama yapılır. Örneğin DOS için bu işlem BOOT ve BPB alanının oluşturulması FAT ve ROOT bölgelerinin 0'lanması biçimindedir. Harddisklerde aşağı seviyeli formatlama işlemi disk'i satanlar tarafından ilk elden yapılmıştır. Disketler için format programının birkaç seçeneği vardır. /u (unconditional) ve /q (quick). Aşağı seviyeli formatlama işleminden sonra bilgi kurtarılması söz konusu değildir.

DİSK ORGANİZASYON BOZUKLUKLARI

Bu tür bozukluklarda fiziksel bir bozukluk söz konusu değildir. Yalnızca disk üzerindeki organizasyona ilişkin bilgilerde bozukluk vardır. DOS/WINDOWS için şu tür organizasyon bozuklukları söz konusu olabilir.

1-)Kayıp cluster'lar(lost clusters):

Bu durumda ortada bir dosyaya ilişkin FAT zinciri vardır. Ancak directory bölgesinde bu zincir için bir giriş yoktur. Bu tür durumlarda onaran programlar zincir için directory girişinde bir dosya girişi oluştururlar.

2-)Karşılıklı bağlı dosyalar(cross linked file):

Bu durmda iki ayrı FAT zinciri belli bir noktadan sonra birleşerek devam etmektedir. Onaran programlar birleşim yerinden sonraki cluster'ların bir kopyasını çıkartarak iki zinciri birbirinden ayırır.

3-)Tahsisat hatası(alocation error, size adjusted):

Bu durumda directory girişindeki dosya uzunluğu ile o dosya için FAT zincirinde tahsis edilmiş olan cluster uzunluğu uyuşmamaktadır. Onaran programlar FAT zincirini esas alarak directory girişindeki dosya uzunluğunu güncellerler.

4-)FAT bozuk(File allocation table bad..):

FAT bölgesinin ilk iki elemanı uygun değerlerde değilse ya da FAT bölgesi aşırı derecede bozuksa söz konusudur.

5-)Genel okuma/yazma hatası(general failure error read/write):

Burada BOOT sector BPB bölgesi anlamlı bir biçimde oluşturulmamıştır. Ya disk foratlı değildir ya da DOS formatlı bir disk söz konusu değildir. Onaran programın yapabileceği birsey yoktur.

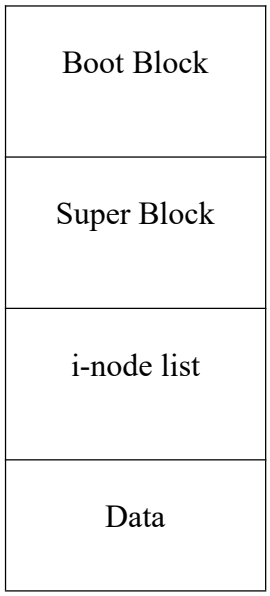
Mademki bir diskin en önemli bölgesi 0. track içerisinde bulunmaktadır. Eğer bu track bozuksa diskin tamamı kullanılamaz duruma düşer(track 0 bad, disk unusable). İşletim sisteminin boot etmek için gereken dosyalar directory girişinde yazı olan normal bir takım dosyalardır. Örneğin DOS işletim sisteminde msdos.sys, io.sys ve command.com dosyaları sistem dosyalarıdır. Sistem dosyalarının bir bölümü disk üzerinde önceden belirlenmiş olan clusterlarda

bulunmak zorundadır. Örneğin DOS'ta msdos.sys ve io.sys dosyaları data bölgesinin ilk clusterlarında olmak zorundadır. Bu zorunluluk boot sectr içerisindeki yükleyici programın FAT ve ROOT bölgelerinde inceleme gereğini ortadan kaldırmaktadır.

UNIX İŞLETİM SİSTEMİNİN DOSYA SİSTEMİ

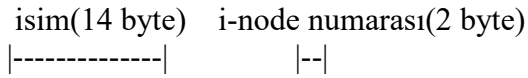
Unix AT&T kurumuna ilişkin tescillli bir isimdir. değişik firmalar değişik unix sistemleri yazmışlardır. Bunlardan bazıları:
UNIX
HPUNIX
SCOUNIX
XENIX
LINUX

Unix'in dosya sistemi ve disk organizasyonu DOS'unkinden oldukça farklıdır. Cluster kavramı yerine block kavramı vardır. Bir block n tane sektörden oluşur. Bir unix disk'i aşağıdaki gibi organize edilmiştir.



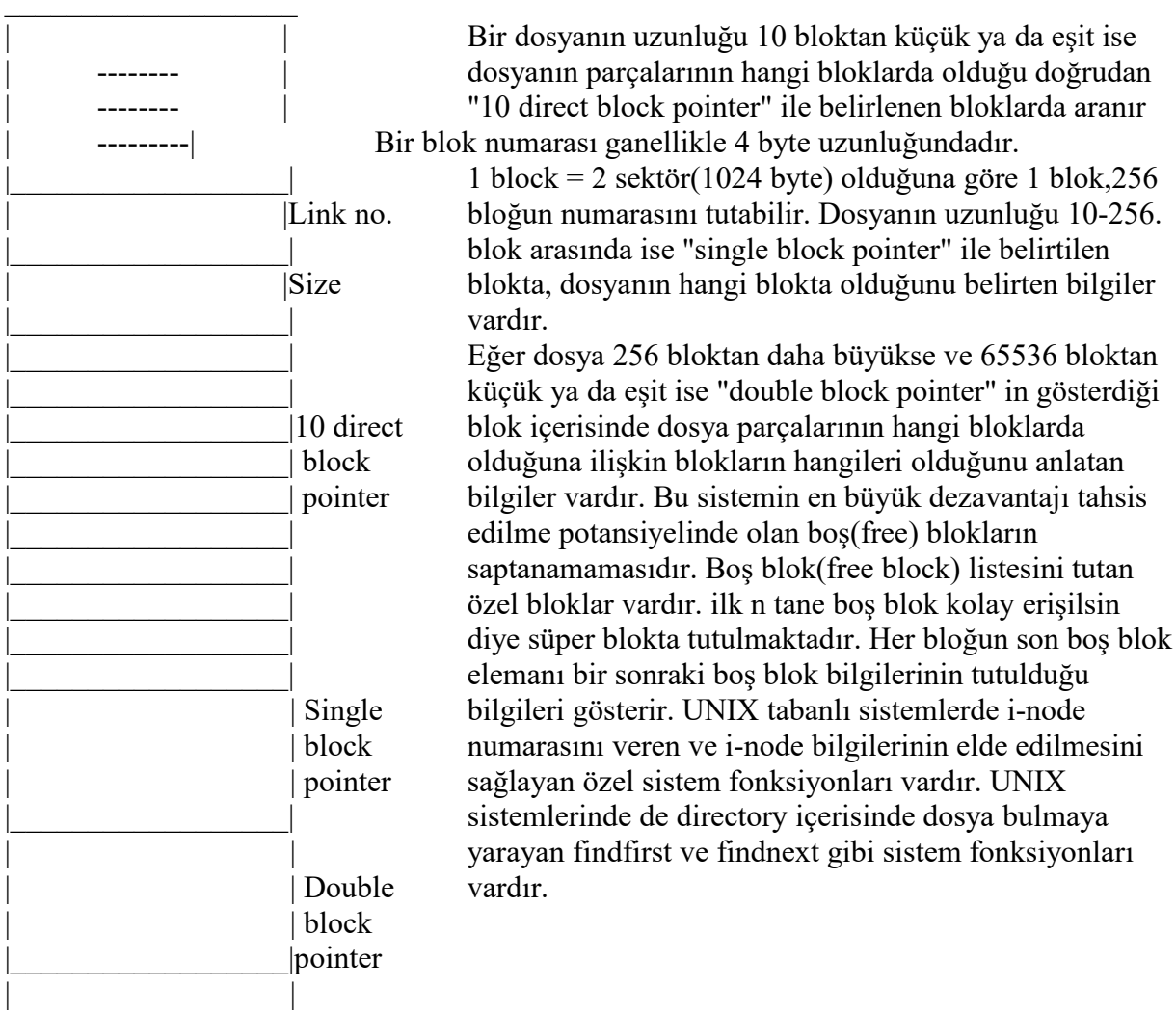
Bir block'un kaç sektör olduğu kurulum sırasında kuran kişiye sorularak belirlenir. Genelikle 2-4-8 değeleri kullanılır. BOOT BLOCK bir block uzunluğundadır. Ve DOS'ta olduğu gibi işletim sisteminin yüklenmesiyle ilgilidir. Super block da bir block uzunluğundadır ve DOS'un BPB alanına karşılık gelir. Yani kritik format bilgileri burada tutulur. Örneğin toplam block sayısı, bir block'un kaç sektörden oluştuğu vb.. i-node list n block uzunluğundadır ve i-node elemanlarından oluşur. i-node elemanlarının unix sisteminde unix sistemine değişebilir. i-node elemanlarının her biri bir dosyaya ilişkindir. i-node elemanlarının her birinin içerisinde dosyanın güvenlik bilgileri, dosyanın disk üzerinde hangi blocklarda olduğu vb. bilgiler bulunur. i-node list bölgesinin uzunluğu değiştirilemez. Yani dosya sistemindeki dosyaların belli bir sayısı olmak zorundadır. Dosyaların isimleri yani directory bilgileri data bölgesinde sıradan bir dosyaymış gibi tutulur. Bir directory dosyası dosya bilgi elemanlarında oluşur. Dosya bilgi elemanları dosya isimlerinden(14 byte) ve i-node numaralarından(2 byte) oluşur. i-node numarası

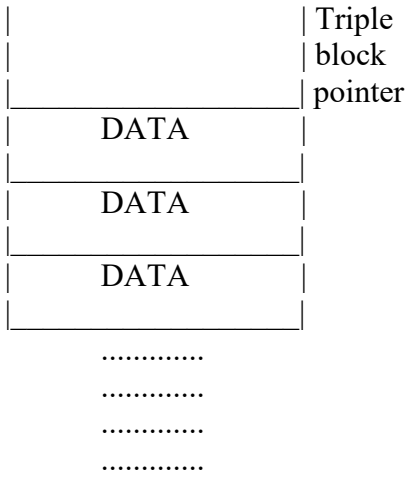
o dosyaya ilişkin bilgilerin i-node list bölgesi içerindeki hangi i-node elemanında olduğunu gösterir.



Root directory'e ilişkin dosya bilgileri sıfırıncı i-node elemanında tutulmaktadır. Alt dizinler normal bir dosyaymış gibi organize edilirler.

i-node Elemanlarının Yapısı





DİSK BÖLGE TABLOSU (DISC PARTITION TABLE)

Bir hard disk değişik işletim sistemlerinin yüklenmesine olanak verecek şekilde organize edilmiştir. Disk üzerinde ardışıl sektörlerden oluşmuş ve her bir tanesi farklı dosya sistemleri ile organize edilen bölgeler oluşturulabilir. Bu bölgelere 'partition' denir. Söz konusu bu disk bölgelerine ilişkin bilgiler, ismine disk bölge tablosu denilen bir tablo ile tutulurlar. Disk bölge tablosu içerisinde bölgelerin hangi kafa-iz-sektör dilimlerinde başladığı, hangi kafa- iz- sektör dilimlerinde bittiği, kaç sektör uzunluğunda bittiği gibi bilgiler vardır.

Disk bölge tablosu 4 bölgenin yerini tutacak şekilde 4 elemanlıdır. Her bir giriş 16 byte uzunluğundadır. Disk bölge tablosu her hard-diskin 0. yüz 0. iz ve 1. sektöründedir. Bu sektörün 1beH (Decimal 446) offsetinden başlar. Yani disk bölge tablosu diskin ilk sektörünün son 16 * 4 + 2 byte'lık bölgesindedir. Son 2 byte disk bölge tablosunun geçerli olup olmadığını gösteren kontrol byte'larıdır. Bu byte'ların 55 AA biçiminde olması gerekir. Fdisk programı aslında bu 64 byte uzunluğundaki bölge üzerinde işlemler yapar. Hard-diskin ilk sektörüne MBR (Master Boot Record) denir. Her disk bölgesinin ilk sektörü o dosya sisteminin boot sektörü olmak zorundadır.

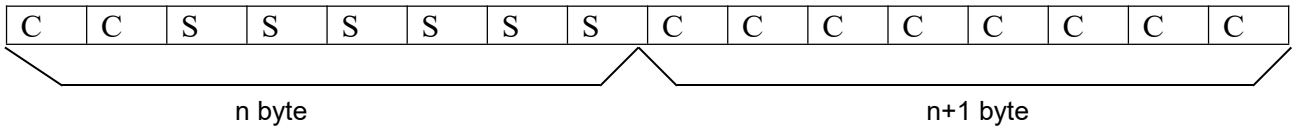
DİSK BÖLME TABLOSUNUN FORMATI

OFFSET	UZUNLUK	ANLAMI
0(0)	BYTE	Bölge durumu:burada ilgili partianın aktif olup olmadığı bilgisi vardır. Buradaki bilgi ya 00h yada 80h durumundadır.4 bölgeden yalnızca biri 80h değerine sahiptir. 80h bülgenin aktif olduğunu gösterir. Aktif bölge hangi bölgenin default işletim sistemine sahip olduğunu anlatır
1(1)	BYTE	Başlangıç kafa numarası ilgili bölgenin hangi yüzden başladığını gösterir.
2(2)	word	Başlangıç sektör ve track
4(4)	byte	Bölge türü o bölgedeki işletim sisteminin hangisi olduğunu belirtir.
5(5)	byte	Bitiş kafa numarası
6(6)	word	Bitiş sektör ve track

8(8)	dword	Başlangıç mantıksal sektör no
c(12)	dword	Bölgedeki toplam sektör sayısı

hangi disk bölgesinin aktif olduğunu tespit eden ve onun boot sektörünü yükleyip çalıştıran program MBR sektöründe bulunmaktadır. yani bios taki program kontrolu MBR içindeki programa bırakır. MBR daki ise kontrolu aktif bölgenin boot sektörüne bırakmaktadır. bir takım bölge yönetim programları MBR yerine yüklenmiş olabilir. bu durumda açılış sırasında hangi işletim sisteminin yükleneceği sorulur; MBR daki yeni program ilgili bölgedeki işletim sisteminin yüklenmesini sağlar.

BAŞLANGIÇ VE BİTİŞ SEKTÖR VE TRACK BİLGİSİNİN HESAPLANMASI



Eğer bölge türü 5 ise bu bölgenin ayrı bir disk alanı gibi değerlendireleceği anlamı çıkar buna extended partian denir diğerlerine primary partian denir extended partian ayrı disk gibi ele alınır extended partianın da ilk sektöründe yine bir disk bölge tablosu vardır. Ancak extended partiana ilişkin bölge tablosunda iki giriş vardır.

!!!!!!!!!! KENDİ KENDİNİ ÇAĞIRAN FONKSIYONLAR !!!!!!!!!

RECURSİVE FUNCTİONS

Bir fonksiyon hatta main bile kendini çağırabilir. tabii bu çağırma işleminin belli bir sayısı olmalıdır. eğer bir fonksiyon sürekli kendini çağırırsa stack taşması oluşur ve sistemin bütünlüğü bozulur. bir fonksiyonun ne kadart sayıda kendini çağırabileceği çalışılan sistemdeki mikro işlemci, işletim sistemi, fonksyın içindeki yerel değişkenlerin taoplam uzunluğu gibi faktörlere bağlıdır. kendi kendini çağıran bir fonksiyondaki yerel değişkenler ne kadar fazla yer kaplarsa kendi kendini çağırma sayısı o kadar azalır. böyle fonksiyonların tasarımında büyük yerel dizilerin kullanılması tavsiye edilmez.

KENDİ KENDİNİ ÇAĞIRAN FONKSIYONLARIN FAYDALARI

Yazılımda kullanılan algoritmalar kendi kendini çağırma açısından 3 bölümde incelenebilir.

1- Kenci kendini çağırmayan fonk larla kullanılan algoritmalar: algoritmaların çoğu bu türdendir.

2- hem kendi kendini çağıran hemde çağırmayan fonklarla sağlanabilen algoritmalar: bu tür algoritmaların iki yazım biçimide oalbilir.

3- yalnızca kendi kendini çağıran fonksiyonlarla çağırılabilen algoritmalar: bu tür algoritmaların kendi kendini çağırmayan fonksiyonlarla tasarımı imkansız veya imkansız denecek kadar zordur.

```
/*---------fakt.c-------------*/
/*--recursive değil--*/
long factorial (int n)

{
   long i;
   long f = 1;

   for (i = 1; i <= n; ++i)
      f *=i;

   return f;
}

main (void)
{
   printf("%ld\n", factorial(8));
}
/*--------------------------------*/
```

Örnek:Faktoriyel algoritmasının kendi kendini çağıran bir algoritmayloa tasarımı.
bir fonksiyon kendini çağırınca yerel değişkenler tekrar yaratılırlar.

```
/*-------fakt_rek.c--------------*/
/*---------recursive------------*/
long factorial (int n)

{
   long temp;

   if (n == 0)
      return 1;
   temp = n * factorial (n-1);
   return temp;
}

main (void)
{
```

```
   printf("%ld\n", factorial(8));
}
/*--------------------------------*/
```

Örnek:Bir sayının tam üssünü alan fonksiyon tasarımı.

```
/*-----------pow_rek.c------------*/
/*recursive*/
long rpow(double a, int b)
{
   if (b == 0)
      return 1;
   return a * rpow(a, b - 1);
}

main (void)
{
   printf("%ld", rpow(2,8));
}
/*--------------------------------*/
```

Örnek:Bir diziyi sıraya dizen recursive bir fonksiyonun tasarımı.(seçerek sıralama yöntemiyle)

Böyle bir tasarımda dizinin başlangıç adresi ve uzunluğu fonksiyona parametre olarak verilir. Fonksiyon dizinin en büyük elemanını bularak sonundaki elemanla yer değiştirir ve kendi kedisini bir eksik uzunluk ile çağırır. Uzunluk 1 olduğunda işlem bitirilir.

```
/*-----------rsort.c----------------*/
void rsort(int *p, int n)
{
   int i, max, indis;

   if (n == 1)
      return;
   max = p[0];
   indis = 0;
   for (i = 1; i < n; ++i)
      if(max < p[i]){
         max = p[i];
         indis = i;
      }
   p[indis] = p[n-1];
   p[n-1] = max;
   rsort(p, n-1);
}

main(void)
```

```
{
   int s[5] = {4, 2, 8, 6, 7};
   int i;

   clrscr();
   rsort(s, 5);
   for (i = 0;i < 5; ++i)
      printf("%d\n", s[i]);
}
/*-----------------------------------*/
```

Örnek:Sadece putchar fonksiyonu kullanarak int bir değerin ekrana yazılması.

```
/*---------printd.c-----------*/
/*---------print_re.c---------*/
```

Örnek:Bir karakter dizisini ters yüz eden fonksiyonun recursive olarak tasarımı.

```
/*----------swapelem.c-------------*/
#include <string.h>
#include <stdio.h>

void swapelem(char *str, int l, int r)
{
   char temp;

   if (l >= r)
      return;
   temp = str[l];
   str[l] = str[r];
   str[r] = temp;
   swapelem(str, l + 1, r - 1);
}

void mystrrev(char *str)
{
   swapelem(str, 0, strlen(str) - 1);
}

void main(void)
{
   char *text = "Merhaba Dünya";

   puts(text);
   mystrrev(text);
   puts(text);
}
```

```
/*---------------------------------*/
```

Dizin Arama İşlemleri(Directory Search And Traversing)

findfirst fonksiyonunun üçüncü parametresi dizin içerisinde hangi özellikteki dosyaların aranacağını gösterir. Özellikler dos.h içerisinde yalnızca bir bit'i 1 olan sembolik sabitler biçiminde tanımlanmıştır.

```
FA_RDONLY
FA_HIDDEN
FA_SYSTEM
FA_LABEL
FA_DIREC
FA_ARCH
```

Bu sembolik sabitler bit OR işlemine sokulur. Örneğin

```
#define FA_ALL      (FA_DIREC | FA_ARCH | FA_HIDDEN | FA_SYSTEM)
```

ifadesi sayesinde FA_ALL kullanılarak tüm attribute'lar için arama yapılabilir.

Örnek:Recursive olarak dizin dosyaları listelenmesi

```
/*--------------dirsearc.c-------------------*/
#include <stdio.h>
#include <dos.h>
#include <dir.h>

void main(void)
{
   struct ffblk X;
   int result;

   result = findfirst("c:/*.*", &X, FA_DIREC);
   while(!result){
      if(X.ff_attrib & FA_DIREC)
         printf("%s\n", X.ff_name);
      result = findnext(&X);
   }
}
/*-----------------------------------------*/
```

Örneğimizdeki walk_tree fonksiyonu kendi kendini çağırarak bütün dizinleri dolaşmaktadır. walk_tree fonksiyonu bir dizin dosyası bulduğunda global pth dizisini o dizini belirtecek biçimde günceller ve kendini çağırır.

```
/*--------------dirlist.c-------------------*/
#include <stdio.h>
```

```
#include <dir.h>
#include <dos.h>
#include <stdlib.h>
#include <string.h>

/*Symbolic Constants*/

#define MAXPATH     80

/*Function Prototype*/

void walk_tree(void);
void disp_dir(const char *str);
void next_path(const char *str);
void prev_path(void);
void dirtree(int drive);

/*Global Variables*/

char pth[MAXPATH] = "C:/*.*";

/*Functions*/

void next_path(const char *str)
{
   *(pth + strlen(pth) - 3) = '\0';
   strcat(pth, str);
   strcat(pth, "/*.*");
}

void prev_path(void)
{
   char *str = pth;
   int n;

   n = strlen(pth);
   str += n -5;
   while(*str != '/')
      --str;
   *str = '\0';
   strcat(str, "/*.*");
}

void disp_dir(const char *str)
{
   /*int num = 0;*/
   int i/*, len*/;
```

```
/*  while (str[num])
      ++num;*/
   for(i = 0; i < /*num*/ strlen(str) - 4; ++i)
      putchar(str[i]);
   putchar('\n');
}

void dirtree(int drive)
{
   pth[0] = drive + 'A';
   walk_tree();
}

void walk_tree(void)
{
   struct ffblk fdirec;
   int result;

   result = findfirst(pth, &fdirec, FA_DIREC | FA_SYSTEM | FA_HIDDEN);
   while(!result) {
      if (!strcmp(fdirec.ff_name, ".")) {
         result = findnext(&fdirec);
         result = findnext(&fdirec);
         continue;
      }
      if(fdirec.ff_attrib & FA_DIREC){
         next_path(fdirec.ff_name);
         disp_dir(pth);
         walk_tree();
      }
      result = findnext(&fdirec);
   }
   prev_path();
}
#if 1

void main (void)
{
   dirtree(2);
}
#endif
/*---------------------------------*/
```

Örnek:sector per cluster büyüklüğü yüzünden kullanılamayan yeri bulan programın tasarlanması.

```
/*-------------tsize.c---------------*/
#include <stdio.h>
#include <dir.h>
#include <dos.h>
#include <stdlib.h>
#include <string.h>

/*Symbolic Constants*/

#define MAXPATH     80

/*Function Prototype*/

void walk_tree(void);
void next_path(const char *str);
void prev_path(void);
long GetAllocSize(int drive);

/*Global Variables*/

char pth[MAXPATH] = "C:/*.*";

long size = 0;

/*Functions*/

void next_path(const char *str)
{
   *(pth + strlen(pth) - 3) = '\0';
   strcat(pth, str);
   strcat(pth, "/*.*");
}

void prev_path(void)
{
   char *str = pth;
   int n;

   n = strlen(pth);
   str += n -5;
   while(*str != '/')
      --str;
   *str = '\0';
   strcat(str, "/*.*");
}

long GetAllocSize(int drive)
```

```
{
   pth[0] = drive + 'A';
   walk_tree();
   return size;
}

void walk_tree(void)
{
   struct ffblk fdirec;
   int result;

   result = findfirst(pth, &fdirec, FA_DIREC | FA_SYSTEM | FA_HIDDEN);
   while(!result) {
      if (!strcmp(fdirec.ff_name, ".")) {
         result = findnext(&fdirec);
         result = findnext(&fdirec);
         continue;
      }
      if(fdirec.ff_attrib & FA_DIREC){
         next_path(fdirec.ff_name);
         walk_tree();
      }
      else
         size += fdirec.ff_fsize % (16 * 512);
      result = findnext(&fdirec);
   }
   prev_path();
}
#if 1

void main (void)
{
   printf("%ld\n", GetAllocSize(2));
}
#endif
/*--------------------------------------*/
```

Çok İşlemli Sistemlerde Kaynak Paylaşımı

Çok işlemli(multi-processing) sistemlerde n tane program(process) zaman paylaşımlı olarak değiştirmeli bir biçimde çalıştırılmaktadır. Genellikle işletim sistemlerinde dönerli çizelgeleme (round rolling scheduling) sistemi kullanılır. Bu sistemde her bir program sırasıyla belli bir süre(quantum) çalıştırılır. Bazı sistemlerde(örneğin WINDOWS 95/98/NT) aynı zamanda bir öncelik derecesi sistemi de kullanılır. Bu sisteme göre öncelik derecesi yüksek olanlar tam olarak bitirildikten sonra diğerlerine çizelgeleme uygulanır. Tabii işletim sistemi klavye, mouse gibi dışsal birimleri kullanmak isteyen programları bu istekler karşılanana kadar

çizelgeleme dışı bırakırlar. Bu duruma bloke edilme(process blocking) denir. Dışsal olay gerçekleştiğinde işletim sistemi tekrar programı çizelgelemeye devam eder.

Çok işlemli sistemlerde bir programın çalıştırılma süresi dolduğunda diğer bir program çalıştırılır. Bu işleme görevler arası geçiş(task switch) işlemi denir. Bir program paylaşılan bir kaynağı kullanmaya başladığında bir görevler arası geçiş durumu oluşursa başka bir program da bu kaynağa eriştiğinde problemli bir durum ortaya çıkar. Çünkü önceki program bir takım işlemlere başlamış olabilir, oysa geçilen program yeni baştan bir takım işlemleri başlatmış olabilir. Programın çalışma sırası eski programa geldiğinde bu programın oluşturduğu eski değerler kaybolacağından sistem çöker. Sistemin çökmemesi için paylaşılan kaynağın bırakılana kadar tek program tarafından kullanılması sağlanmalıdır.

Paylaşılan bir kaynağı kullanan koda kritik kod(critical section) denir. Kritik kodlara girişler ve bu kodlardan çıkışlar işletim sisteminin özel fonksiyonları çağırılarak belirtilmelidir. Eğer bir program kritik bir koda girdiyse ve başka bir program da aynı kaynağa ulaşmak istemişse o program işletim sistemi tarafından çizelgeleme dışı bırakılır. İlk program kritik kodu bitirdiğinde diğer program çizelgeleme dahil edilerek kritik koda sokulur. Kritik kod yaratma işlemine semafor ya da seri hale getirme işlemi denir.

HATA AYIKLAMA VE ASSERT MAKROSUNUN KULLANIMI

BUG NEDİR??

Bir kodun bazı durumlarda hatalı çalışmasına yol açan olaylar. Bir programın yanlış çalışmasına yol açan gizli yazılım hatalarıdır. Bug lar programın satır sayısının artması ile doğru orantılı bir şekilde artarlar. Bir programda bug olup olmadığı büyük firmalarda içsel test bölümlerince test edilir testin son aşaması gerçek kullanım testi(beta) testi aşamasıdır. bugların ayıklanması ve tespit edilmesi için özel yazılımlara gereksinim duyulabilir. Bir programın hatasız olmasını sağlamanın en önemli yolu programı hatasız olarak yazmaya çalışmaktır. Hatasız program yazmak için alınacak ilk önlem uyarı mesajlarını gözden geçirmektir. Çünkü uyarı mesajlarının %95'i programcınını gerçekten yapmış olduğu hataları bildirmektedir. derleyicilerin uyarı mesajlarının bazıları kurulum sırasında default olarak pasif duruma getirilmiştir. Bunların hepsinin aktive edilmesi hata yakalama açısından önemlidir. Uyarı mesajları ya da uyarıyı gerektiren durumlar standart olarak belirlenmemiştir ama error gerektiren durumlar dilin kuralları ile ilgili olduğu için standardize edilmiştir.

BUGLARIN PROGRAMIN YAZILIMI SIRASINDA TESPİT EDİLMESİ

Bir programı yazarken hataların tespit edilmesinin en iyi yolu bir ipucunun elde edilmesidir. Bu ipucunun sağlanması için program içinde hata yapılma olasılığı olan tüm kodlara if kontrolleri yerleştirilir. if deyimi içerisine bir mesaj yazılarak program sonlandırılabilir. Bu mesaj bir ipucu niteliğindedir. Buradan hareketle bug'a yol açan kod bulunabilir. Her türlü hata olasılığı içeren kodlara kontrol yerleştirmek iyi bir çözümdür. Ancak gereksiz kontroller programı büyütür ve yavaşlatır. Yapılacak en iyi şey program deneme(debug) ve gerçek (release) versiyonlarına ayırmaktır. Deneme versiyonu kontrollü ve yavaş çalışan verisyondur, ancak hatalar bu sayede tespit edilebilir. Oysa release versiyon kontrollerin silinmesi ile elde edilmiş versiyondur, satılcak olan da release versiyonu satılacaktır. İşte en önemli nokta tek bir kod üzerinde bu kontrollerin eklenip çıkartılmasına olanak sağlayacak bir yöntem belirlemektir.

Özetle program deneme versiyonu ile geliştirilip test edilir gerçek versiyonu ile satılır. Bu işlemin de pratik bir biçimde yapılması gerekir.

DEBUG KONTROLLERİ NEREYE VE HANGİ DURUMLARDA YERLEŞTİRİLMELİDİR??

1- Gösterici parametresi alan fonksiyonlarda geçirilen adresin NULL gösterici olup olmadığının testi. Yani gösterici parametresi alan tüm fonksiyonların blok başlarında bu durum test edilmelidir.

2- Sınır değerlerinin test edilmesi için konulan değerler.

3- Verify işlemlerinde kullanılabilir. Olması kesin gözüyle bakılan durumların test edilmesine yönelik kontrollerin konulması.

Hata kontrolü problemi tespit ederse problemin kaynak kod içerisinde nerede olduğunu da belirtmesi gerekir. Örneğin kursoru konumlandıran pos fonksiyonu binlerce yerde çağırılmış olabilir, bu fonksiyona yanlış parametre geçilmesinden dolayı tespit edilen hatanın yeri belirlenmelidir. Yerleştirilecek kontrollerin sayısı istenildiği kadar çok olabilir, ancak gereksiz kontrollerin yerleştirilmesi doğru değildir.

abort ve exit Fonksiyonları

exit fonksiyonu bir takım işlemleri geri alarak(programın yüklenmesi sırasında yapılmış bazı işlemler) programdan çıkmak amacıyla kullanılır. Bu fonksiyonla programdan çıkmak için programın bütünlüğünün bozulmamış olması gerekir. Oysa abort fonksiyonu geri alma işlemlerini yapmadan ani bir biçimde çıkışı sağlar. Yani programın bütünlüğü bozulduuğunda bir takım işlemlerin geri alınmasına bile olanak kalmamışsa abort fonksiyonu tercih edilmiştir. Her iki fonksiyonun prototipi de stdlib.h 'tadır. abort fonksiyonu parametresizdir ve programı sonlandırırken ekrana "Abnormal program termination" yazısını yazar. Programın deneme sürümündeki hata yakalama kodlarından çıkış abort fonksiyonuyla yapılmalıdır.

assert Makrosu

assert isimli makro ANSI standartlarında tanımlanmış olan ve bildirimi assert.h içerisinde olan programın deneme sürümünde kontrol yerleştirmesini sağlayan bir makrodur. Bu makro yerine programcının isteği doğrultusunda çeşitli makrolar da yazılabilir.

Standart assert Makrosunun Kullanılması

assert makrosu bir karşılştırma ifadesiyle çağırılmadır. Örnek:

```
assert(p != NULL);
```

Bu makroyla program içerisine ön işleci tarafından bir kontrol kodu yerleştirilir. assert ifadesi olması gereken iddiayı anlatmalıdır. Eğer bu iddia geçersizse yani assert ifadesinin sayısal değeri 0'sa kontrol kodu tarafından ekrana "Assertion failed" yazısı basılarak abort fonksiyonu ile program sonlandırılır. Eğer assert.h dosyasından önce NDEBUG ifadesi define edilmişse assert makrosu koddan tamamen çıkartılır. Bu durumda deneme sürümünde bolbol assert kullanılır ancak NDEBUG sebmbolik sabiti define edilmez. Programın bug'sız çalıştığına emin olduktan

sonra NDEBUG sabiti define edilerek program dahil edilir. Bu işlemle bütün assert kontrolleri koddan çıkarılmış olur.

```
/*----------assert.c----------*/
#include <stdio.h>
#include <assert.h>

void myputs(const char *str)
{
   assert(str != NULL);
   while(*str != '\0'){
      putchar(*str);
      ++str;
   }
   putchar('\n');
}

void main(void)
{
   char s[] = "Ankara";
   char *p;

   p = strchr(s ,'x');
   myputs(p);
}
/*----------------------------*/
```

assert ile yakalanan nokta bug'ın oluştuğu nokta değildir. Bug'ın olumsuzluklara yol açtığı noktalardan biridir. Bu noktadan hareketle bug'ın yeri tespit edilmelidir.

ASSERT MAKROSUNUN YAZILMASI

Assert makrosunun ASSERT biçiminde tekrar yazmayı deneyelim. yazacağımız bu versiyonda DEGUB isimli sembolik sabit define edilmemiş ise ASSERT ifadeleriş koddan çıkartılır. edilmişse koda eklenir (orjinal assert işlemlerinde NDEBUG sembolik sabiti define edilmniş ise assert ifadeleri koddan çıkartılmaktadır).
1. versiyon:

```
#ifdef DEBUG
   #define ASSERT(f)   \
      if (f) { }      \
      else {          \
         printf("Assert failed\n");\
         abort();    \
      }
#else
   #define ASSERT(f)
```

```
#define DEBUG
#include "myassert.h"

void fonk (const char *p)
{
  ASSERT (p != NULL);                                /*kontrol olayı*/
}

if (p != NULL) {}
else {
    printf("Assertion failed...\n");
    abort();
}
```

2.versiyon:
hata yakalandığında verilecek mesaj ve abort fonksiyonunun çağırılma kodu başka bir fonksiyon içerisine yazılabilir. böylece her çağırmada mesaj stringi için tekrar tekrar koda eklenmez.

```
#ifdef DEBUG
void _Assert (void)
{
   printf("Assertion failed...\n");
   abort();
}
#endif

#ifdef DEBUG
   #define ASSERT(f)       \
      if (f) { }          \
      else               \
         _Assert();
#else
   #define ASSERT(f)
#endif
```

3. versiyon:
Bu assert işlemlerini daha etkin hale getirmek için hatanın yakalandığı dosya isminin ve satır numarasının da rapor edilmesi gerekir.

SOnuç Assert işlemleri ön işlemci komutları ile koda eklenip çıkartılmaktadır. Orjinal Assert makrosu Standart bir biçimde tüm C derleyicilerinde bulunurlar. doğrudan bu makro kullanılabilir.

PORT KAVRAMI VE İŞLEMCİLER

İşlemci nedir???

İşlemci programlanabilen yani komut çalıştırabilen elektronik devrelerdir. işlemciler bnu günkü teknoloji ile entegre biçiminde üretilirler. İşlemciler Çeşitli uçlara sahiptir. işlemcilere komut gönderilmesi elektriksel işaretlerle yapılır. sayısal bilgisayarlarda kullanılan işlemciler. ikilik sistemde çalışırlar. İşlemcinin bir ucuna iletilen gerilim seviyesi sayısal olarak 0 veya 1 biçiminde algılanır. örenğin 5 volt civarında bir gerilim 1, 0 civarında bir gerilim ise 0 biçiminde yorumlanır. işlemciyi programlamak yada komut göndermek demek işlemcinin n tane ucuna elektrikdsel işaretler uygulamak demektir. işlemci bu n uçtan aldığı değerleri sayısal olarak yorumlayarak ne yapması gerektirğini anlar. işlemcinin işlemi yapması için belli bir zamana ihtayaçı vardır. bu zaman sonrasında elde edilen değerler işlemcinin m tane ucundan okunabilir. tabi m tane uctan okunan değerler m bit sayı biçiminde yorumlanmalıdır. bir işlemciye komutla rve komutun parametreleri 8-16-32 bit topluluklar halinde gönderilebilir. böyle işlemcilere 8-16-32 bit işlemciler denir. bir bilgisayar sisteminde kendi yerel bölgesinde sorumlu pek çok işlemci vardır. bu işlemcilere komutları merkezi bir işlemci gönderir. böyle işlemcilere de CPU denir. tabi cpu da programlanabikir bir işlemcidir. onun hangi işlemcileri programlaması gerektiğine sonucta programcı karar verir. cpu ile ram ve işlemciler ik grup uç ile bağlanmışlardır. Veri yolu denilen bir grup uç işlemcilere ve ram e bilgi göndermek yada onlardan bilgi almak için kullanılır, adres yolu denilen bir grup uç ise istenilen bir işlemciyi seçmek yada ramden bir kutucuğu seçmek amacı ile kullanılır. bir işlemcinin port numarası o işlemcinin seçilmesini sağlayan bir sayıdır. bire işlmecinin n tane port numarası olabilir. bütün port numaraları çeşitli işlemciler tarafından paylaştırılmıştır. örneğin bir port bir bilgi gönderildiğinde o port numarası hangi işlemciye bağlı ise bilgi o işlemciye gönderilmiş olur. bir işlemciyi programlayabilmek için şunları bilme gerekir.

1- işlemcinin port numarasını
2- işlemcinin hangi komutlara karşı ne yapacağını
3- bu işlemlerin programlama içinden nasıl yapılacağı

Bir cpu nun en fazla kaç portsa bilgi gönderebileceği tasarımda tespit edilir. örneğin intel işlemcileri en fazla 65536 porta erişebilir. klasik bir at mimarisinde ilk 4096 port kullanılmaktadır. yani işlemcilerin port numaraları 000-3FF arasındadır. bir işlemcinin port numaraları eski sistemde elektriksel bağlantılarla yani değiştirilemez bir biçimde belirleniyordu. oysa son yıllarda geliştirilen pnp teknolojisi ile işlemcilerin port numarası da dışarıdan yazılım yolu ile değiştirilebilmektedir. bu sistem mimariye yeni bir kart yada işlemci eklenmesinde oluşacak çatışmaları gidermek amaçlı düşünülmüştür.

Portlara Değer Gönderilmesi ve Portlardan Değer Alınması

Aslında portlara bilgi gönderip portlardan bilgi alma işlemi OUT ve IN makine komutlarıyla yapılır. Tabii bu komutları kullanabilmek için sembolik makine dilini bilmek gerekir. Ancak bunun yerine prototipleri dos.h içerisinde bulunan bir grup fonksiyondan da faydalanılabilir. Bu fonksiyonlar 8 bit ve 16 bit işlemciler için düşünülmüştür.

1-İşlemcilere 1 Byte Bilgi Gönderen Fonksiyonlar

void outportb(int portadr, unsigned char value);
int outp(int portid, int value); (makro)

Bu fonksiyonların işlevleri tamamen aynıdır. Birinci parametreleri bilginin gönderileceği port numarası, ikinci parametresi ise gönderilecek değerdir.

2-İşlemcilerden 1 Byte Bilgi Okuyan Fonksiyonlar

unsigned char inportb(int portadr);
int inp(int portid); (makro)

Bu fonksiyonlar parametresiyle belirtilen porttan bilgiyi okuyarak geri dönüş değeri olarak bildirirler.

3-Portlara 2 Byte Bilgi Gönderen ve Okuyan Fonksiyonlar

void outport(int portadr, int value);
unsigned inport(int portadr);

Portların Kullanılmasına İlişkin Bir Örnek

Bir işlemcinin port numaraları işlemci üretildiğinde göreli olarak belirtilir. Onlara gerçek numaraları tasarımcı vermektedir. Birinci 8259 20H ve 21H portlarını kullanmaktadır. 8259'un 21H portuna 1 byte bilgi gönderilince bu bilgi 8259'un interrupt mask register denilen bir register'ına yazılır. 8259 interrupt mask register içerisindeki 1 olan bitlere ilişkin uçlardan gelen kesmeleri görmezlikten gelir. 21H portu hem okunabilir hem yazılabilir bir porttur.

Zamanlayıcı kesmesini devre dışı bırakmak için yapılacak işlemler:

Bunun için önce interrupt mask register okunur(Yani 21H portu okunacak), sonra 8254 0 numaralı uca bağlı olduğuna göre diğer bitlere dokunulmadan 0 numaralı bit 1 yapılır(1 ile OR işlemi) ve elde edilen değer geri yazılır.

UYARI: Çok işlemli işletim sistemlerinde(Win, Unix vesaire) koruma mekanizması uygulanmaktadır. Koruma mekanizması gereği normnal programcılar portlara erişip onları programlayamazlar.

------outp.c-------
```
#include <dos.h>
#include <stdio.h>

void main(void)
{
   char x;

   x = inp(0x21);
   x = x | 1;
```

```
   outp(0x21, x);
}
```

PC'LERDE ZAMANLAMA İŞLEMLERİ

PC mimarisinde zamanlama konusuyla ilgili iki işlemci bulunmaktadır.
1-8254 PIT(programmable interval timer)
2-RTC(real time clock)

8254 PIT İŞLEMCİSİ

Bu işlemci kendi içersinde 3 ayrı darbe üretebilen sayaca sahiptir. Bu üç ayrı sayaç mimaride çeşitli amaçlarla kullanılır. Örneğin sayaçlardan birisi 8259 kesme denetleyicisi işlemcisine bağlıdır ve saniyede 18,2 kere darbe üreterek 8H donanım kesmesinin çağırılmasına yol açar. 8254'ün 3 bağımsız sayacının da darbe üretme frekansının tespit edilmesinde kullanılan bir yazmacı(register) vardır. Programlama sırasında bu yazmaca 16 bit uzunluğunda bir sayı yerleştirilir. İşlemcinin herbir saat darbesinde bu yazmaç içerisindeki sayı bir azaltılır. Bu sayı sıfırlanınca darbe üretilir ve ilk sayı tekrar yazmaç içerisine yerleştirilir. PC mimarisinde 8254 işlemcisinin kristal frekansı .....(tam değer gelecek). 8H kesmesi için kullanılan sayaca ilişkin yazmaç değeri en büyük değer olan 65535'tir. Bu durumda 8H kesmesinin oluşma frekansı ...../65535 = ...'dır. Bu durumda 8254 işlemcisi programlanarak 8H kesmesinin çağırılma hızı arttırılabilir ama azaltılamaz. CPU çalışma frekansından bağımsız gecikme sağlayacak bir fonksiyon bu işlemcinin programlanmasıyla yazılabilir. Tabii böyle bir fonksiyon windows ve unix gibi sistemler altında kullanılamaz(bu işletim sistemleri bu işlemciden görevler arası geçiş mekanizması için faydalanırlar ve koruma mekanizması içinde bu işlemciye erişim engellenmektedir). Yani böyle bir fonksiyon sadece DOS altında yazılabilir. Doğal olarak istenirse ayrı bir kart biçiminde başka bir 8254 işlemcisi sisteme eklenebilir.

RTC İŞLEMCİSİ

Bu işlemci tamamen bir saatin yaptığı işlemleri yapmak amacıyla tasarlanmıştır. O anda bulunulan tarih ve zaman bilgisi RTC işlemcisine sorularak elde edilebilir. RTC işlemcisi AT'lerle birlikte mimariye katılmıştır ve bilgisayar kapatılsa bile bir pil tarafından beslenerek çalışmaya devam eder. Bu işlemcinin aynı zamanda küçük bir RAM bölgesi vardır. Setup bilgileri bu bölgede tutulur. Bu işlemci tarih bilgisine ilişkin yıl bilgisini 4 digit halinde tutmaktadır.

C'de Tarih ve Zaman Bilgisinin Elde Edilmesi

En temel yöntem time ve localtime fonksiyonlarının kullanımasıdır. time fonksiyonu 01-01-1970'den itibaren geçen saniye sayısını verir. localtime fonksiyonu bu değeri alarak bunu tarih ve zaman bilgisine dönüştürür. Ancak bu fonksiyonlar tarihe ilişkin yıl bilgisini 2 digit olarak verirler. Sonuç olarak tarihe ilişkin yıl bilgisinin 4 digit olarak elde edilmesi için C'de RTC işlemcisinin okunması gerekir(diğer programlama dillerinde kimi fonksiyonlar yada komutlar RTC işlemcisni okuyarak yıl bilgisini 4 digit olarak zaten verirler). RTC işlemcisi ile

ilişki kurmak için 2 yöntem kullanılabilir:
1-Doğrudan işlemcinin port numaraları kullanılarak erişilebilir.Ancak bunun için RTC'nin programlanmasını bilmek gerekir.
2-1A BIOS kesmesinin çeşitli kesmeleri RTC işlemcisini programlayarak zaten temel işlemleri yapmaktadır. Yani bu kesme çağırılarak işlemler yürütülebilir.

1AH Kesmesinin Fonksiyonları

Bu kesmenin 4 önemli fonksiyonu vardır.
F:2 Get time
F:3 Set time
F:4 Get date
F:5 Set date
Bu fonsiyonlar BCD(binary coded decimal) sayı sistemini kullanmakatdır.

BCD Sayı Sistemi

Onluk sistemde yazılmış olan her bir digit'e 4 bit karşılık düşürülürse o onluk sayı BCD sisteminde ifade edilmiş olur.
1913 == 0001 1001 0001 0011
1001 001 == 91

1AH F:2
Parametreler AH=>2
Geri dönüş değeri CH=>saat, CL=>dakika, DH=>saniye(BCD yorumu)
1AH F:3
Parametreler AH=>3, CH=>saat, CL=dakika, DH=>saniye(BCD)
1AH F:4
Parametreleri AH=>4
Geri dönüş değeri CH=>yüzyıl - 1, CL=>yıl, DH=>ay, DL=>gün(BCD)
1AH F:5
Parametreler AH=>5, CH=>yüzyıl - 1, CL=>yıl, DH=>ay, DL=>gün(BCD)

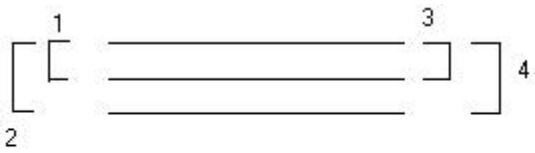
Bu fonksiyonlar eğer RTC bulunmadığından dolayı başarısızlığa uğrarsa cflag elamanı 1 olmaktadır.

PC'LERDE HABERLEŞME

SAYISAL HABERLEŞMENİN TEMELLERİ

Haberleşme temel olarak gönderici tarafın hatta bir gerilim uygulaması alıcı tarafın da bu gerilime bakarak hangi bitin gönderildiğini tespit etmesi esasına dayanır. Yani haberleşmede bilgiler bitlerine ayrıştırılır, bit bit iletilir. Haberleşmede bir gönderici bir de alıcı taraf söz konusudur. Eğer tek yönlü bir haberleşme sistemi kullanılacaksa iki hat yeterlidir. (Hatlarsan biri toprak gerilim iki uca uygulanacaktır) Eğer çift yönlü bir haberleşme isteniyorsa yani iki tarafın

da hem göndermesi hem alması söz konusuysa hat sayısı üçe çıkartılmalıdır. Gönderme ve alma durumlarına göre haberleşme sistemleri bir kaç gruba ayrılır.



1:Gönderme
2:Alma
3:Alma
4:Gönderme

Simplex: Tek taraflı haberleşme
Half Dublex: İki taraflı ama gönderme ve alma işlemi aynı anda yapılamaz.
Full Dublex: Her iki taraf da gönderip alma işlemini eş zamanlı yapabilir.

t anında 1 bit ya da n bitlik bir bilginin hatta bırakılması durumuna göre haberleşme seri haberleşme ve paralel haberleşme olarak iki bölüme ayrılabilir. Paralel haberleşme için daha fazla hat gerekir ve uygulamada tercih edilmemektedir. Seri ya da paralel haberleşmenin prensipleri arasında bir farklılık yoktur. (Paralel haberleşmeye byte seviyesinde seri haberleşme de denir)

Haberleşmenin Problemleri

1-)Eş zamanlılık problemi(senkronizasyon)

Gönderici ve alıcı tarafın eşit hızlarda işlemlerini yapması gerekir. En önemli problem gönderici beklemeye başladığında alıcının bu beklemeyi tespit etmesidir. Gönderen tarafın byte'lar arasında hiçbir bekleme olmadan bütün bitleri eş zamanlı göndermesi durumuna senkron seri haberleşme sistemi denir. Byte'lar arasında rast gele bekleme zamanları olabilir ancak byte'lar eşit zaman aralıklarıyla gönderilebiliyorsa bu haberleşme sistemine asenkron seri haberleşme sistemi denir. Uygulamada hemen her zaman asenkron seri haberleşme sistemi kullanılır.

2-)Alıcı tarafın doğru bilgiyi alıp almadığı bir biçimde test edilmelidir.

Eğer alıcı taraf bilgiyi aldıysa hiç olmazsa bilginin yanlış alındığının bilinmesi gerekir.

3-)Mesafe sorunu

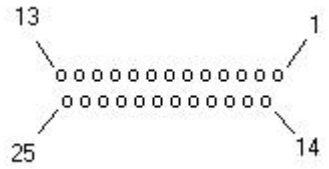
Gönderici ve alıcı taraf birbirinden çok uzaksa bilgi telefon haberleşme sistemiyle (telefon tellerinden modem aracılığıyla) gönderilip alınabilir.

Bilgisayardaki Haberleşme Portları

PC'lerde ismine seri ve paralel port denilen iki grup haberleşme portu bulunur. Paralel portlar 25 pinli bilgisayar tarafı dişi olan konnektörlerden oluşur. Seri portlar 9 ya da 25 pinli olabilirler. Bilgisayar tarafı erkek biçimindeki konnektörlerden oluşur. Normal olarak PC'ler en fazla 4 seri 4 paralel portu desteklemektedirler. Satın alındığında genelde iki seri bir paralel portu bulunur. Fakat özel kartlarla bu sayı fazlalaştırılabilir.

Paralel Portlar

Pinlerin İsimleri ve Anlamları



Pin isimleri bilgisayar printer haberleşmesi düşünülerek konulmuştur. Tabii paralel port yalnızca printer haberleşmesinde değil daha pek çok konuda kullanılabilir. Bu durumda isimlerin bizim için ciddi bir anlamı yoktur.

1: STROPE
2: D0
3: D1
4: D2
5: D3
6: D4
7: D5
8: D6
9: D7
10: ACK
11: BUSY
12: PE
13: SLCT
14: AUTOFEED
15: ERROR
16: INIT
17: SLCT IN
18-25: GND(Toprak)

Bizim için paralel port 25 pinden oluşmuş bir devre olarak değerlendirilebilir. Paralel portun bazı pinlerine programlama yoluyla 5 Volt ya da 0 Volt gerilim uygulanabilir. Yine paralel portun bazı pinlerindeki oluşturulmuş olan gerilimler okunabilir. Paralel portun uçlarına bilgi göndererek bu porta takılan çeşitli cihazlar takılabilir. Paralel portun çıkışları TTL seviyesindedir. Yani doğrudan logic devreler sürülebilir. Paralel portun 2-9 nolu uçlarına yalnızca işaret gönderilebilir. Yani bu uçlardan okuma yapılamaz. (Bazı eski paralel portlarda bu uçlar çift yönlü çalışabilmektedir) Paralel portun yalnızca 5 ucundan okuma yapılabilir. Bu pinler 10,11,12,13,15 nolu pinlerdir. Diğer pinler okuma ya da yazma amaçlı kullanılamazlar. Sonuç olarak 8 bit gönderme 5 bit okuma yapılabilir. Paralel portun uçları paralel port işlemcisi denilen özel bir işlemciye bağlıdır.

Paralel Port İşlemcisinin Kullanımı

Bu .işlemcinin 3 tane 8 bitlik register'ı vardır. Bu register'lara DATA, STATUS, COMMAND register'ları denir. Bu 3 register'ın da birer port adresi vardır. Bu portların adresleri normal olarak BIOS haberleşme alanında yazmaktadır. Bunların adresleri

0040:0008 => LPT1
0040:000A => LPT2
0040:000C => LPT3
0040:000E => LPT4

Ancak sistemlerin çok büyük çoğunluğunda taban adresler şu biçimdedir.

LPT1 => 378H
LPT2 => 278H
LPT3 => 3BCH

Bu durumda paralel port işlemcisinin register'ları taban port adresine göre şu biçimdedir.

Taban + 0 = Data register
Taban + 1 = Status register
Taban + 2 = Command register

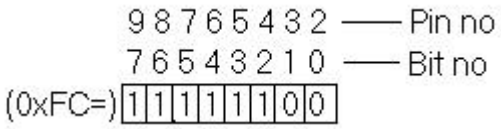
Örneğin LPT1 için:

Data register için 378H,
Status register için 379H,
Command register için ise 37AH'ta bulunur.

Data Register

Bu register'ın LPT1 için port numarası 378H, LPT2 için port numarası 278H'tır.Bu register yalnızca yazma amaçlı kullanılabilir. Okuma amaçlı kullanılamaz.

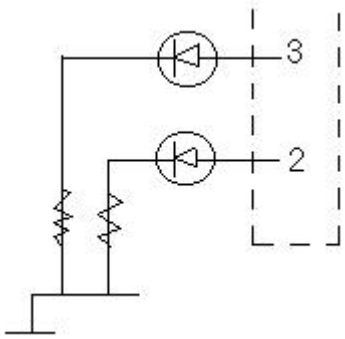
```
outp(0x378, 0xFC);    /*LPT1'in data register'ına 0xFC bilgisi gönderilmesi*/
```



Data register'a bir bilgi yazıldığında yazılan bilginin bitleri paralel portun D0 ve D7 uçları arasında gerilim seviyesi olarak görünür. Bu uçlar 2-9 numaralı uçlardır. Örneğin paralel portun 2

ve 3 numaralı pinlerinde 5 Volt oluşturabilmek için data register'a 3 değerinin yazılması lazımdır. (3 = 0011)

Led yakıp söndürme:(led.c)



Status Register

Bu register LPT1 için 379H, LPT2 için 279H port adreslerindedir. Yalnızca okunabilen bir register'dır. Bu register'dan okunan bilgiler 10, 11, 12, 13, 15 nolu uçların gerilim seviyelerini göstermektedir. 7 numaralı bitten elde edilen değer 11 numaralı pinin tersidir. Yani bu bit 1 olarak okunmuşsa 11 numaralı pin 0 Volta çekilmiştir.

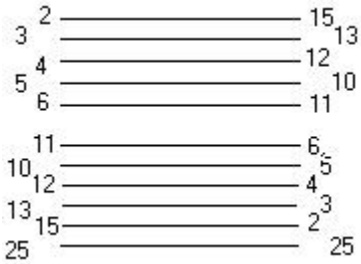
Örnek:13 nolu pin ile 25 nolu pin kısa devresinde okuma.....(read.c)

Command Register

LPT1 için 37A LPT2 için port adresleri 27A'dır. Bu register belli bir olay meydana geldiğinde ya da belli durumlarda kullanılır. Yani okuma ya da yazma amaçlı kullanılamaz. Normal olarak write-only bir register'dır. Ancak bu register'a bilgi yazıldığında paralel port işlemcisi çeşitli amaçlarla programlanır. Yani burada yazılan değerler işlemciyi programlamak amacıyla kullanılmaktadır.

Bilgisayarlar Arası Bilgi Aktarımı

İki bilgisayar arası paralel portlar kullanılarak bir bilgi aktarımı yapılacaksa ismine laplink denilen bir kablo hazırlanması gerekir. Laplink kablosu bağlantı şeması:



2:D0
3:D1

4:D2
5:D3
6:D4
15:error
13:SLCT
12:PE
10:ACK
11:BUSY
25:GND

Mademki paralel portun 5 ucu alma 8 ucu gönderme için kullanılıyor, o halde bilgi transferi de olsa olsa bit-bit yapılabilir. Yani bir byte karşı tarafa iki hamlede gönderilebilir. Laplink kablosunda gönderen tarafın 4 data ucu alan tarafın 4 alıcı ucuna karşılıklı bağlanır. Gönderen taraf bir byte'lık bilgiyi 4 bit 4 bit ayırarak gönderir. 4 bit bilgi göndermek için gönderici taraf 4 biti data register'a yazar, alıcı taraf status register'ı okuyarak bu 4 biti elde eder. Tabi bu işlemin bir biçimde eş zamanlı olarak yapılması gerekir. Bu eş zamanlılığı sağlamak için beşinci uç kullanılmaktadır.
Başlangıç durumu: Alıcı taraf 1'de bekliyor.
1.adım: Gönderen 4 biti hatta bırakıyor; hattı sıfıra çekiyor. Bu sırada alıcı taraf döngüde 1 olduğu müddetçe bekliyor.
2.adım: Gönderen hat 0'da olduğu sürece bekliyor. Alıcı gönder demek için hattı 1'e çekiyor ve alıcı 1 olduğu müddetçe bekliyor.

Eş zamanlılık için kullanılan 6-11 bağlantısı ters işaretlidir. Tabii bu bağlantıda 6-11 ve 11-6 olmak üzere çift taraflıdır.

Transfer Programı Hakkında Açıklama

Programda sendchr ve receive_chr olmak üzere iki fonksiyon yazılmıştır. Sendchr 1 byte uzunluğunda bilgiyi 4 bitlik iki parçaya ayırarak karşı tarafa gönderir. Receive_chr eş zamanlılık belirlemelerine göre bunu alır. Tabii bu fonksiyonların tek başlarına kullanılması sonsuz beklemeye yol açar. Yani bir bilgisayar sendchr ile bir byte bilgi göndermişse işlemin devam etmesi için karşı
tarafın receive_chr ise bunu alması gerekir. Örneğimizdeki sendchr ve receive_chr fonksiyonları ayrı birer fonksiyon olarak tasarlanmıştır. Bu fonksiyonlar dosya transferinin ötesinde endüstriyel uygulamalarda PC'lerin ucuz haberleşmesini sağlamak amacıyla kullanılabilir. Paralel laplink kablosu en fazla 3-4 metre uzunluğunda olmalıdır. Örnek programda gönderen ve alan iki ayrı program vardır. Gönderen taraf önce dosya ismi ve uzunluğunu gönderir. Alan taraf dosya ismini aldıktan sonra yazma modunda dosyayı açar. Dosya uzunluğu alan tarafın işlemini bitirmesi için gereklidir.

Paralel portlar hakkında adresler
http://dragon.hearts.ac.uk/data/datasheets/parallel.html
http://www.geocities.com/SiliconValley/Bay/8302/parallel.htm

/*----------sendrece.c----------*/

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <stdlib.h>

#define LPT1DATA   0x378
#define LPT1STAT    0x379

typedef unsigned char BYTE;

int receive_char(void)
{
        BYTE dpos;
        BYTE rchar;

        do {
                dpos = inp(LPT1STAT);
        } while((dpos >> 7) == 1);
        dpos = inp(LPT1STAT);
        dpos = (dpos << 1) >> 4;
        rchar = dpos;
        outp(LPT1DATA, 0x0);
        do {
                dpos = inp(LPT1STAT);
        } while((dpos >> 7) == 0);
        dpos = inp(LPT1STAT);
        dpos = (dpos << 1) >> 4;
        rchar |= dpos << 4;
        outp(LPT1DATA, 0xFF);
        return rchar;
}

void sendchar(char ch)
{
        BYTE dpos;

        do {
                dpos = inp(LPT1STAT);
        } while((dpos >> 7) == 1);/*wait until the receiver is ready*/
        dpos = ch & 0x0F;
        dpos |= 0x10;
        outp(LPT1DATA, dpos);/*send lower 4 bit to receiver*/
        do {
                dpos = inp(LPT1STAT);
        } while((dpos >> 7) == 0);/*wait until the receiver is ready*/
        dpos = ch >> 4;
        dpos &= 0x0F;
```

```
        outp(LPT1DATA, dpos);/*send upper 4 bit to receiver*/
}

#ifdef RECEIVE

void main(void)
{
        FILE *fp;
        char fname[20];
        char flen[20];
        BYTE ch;
        long len;
        long n;
        int i;

        outp(LPT1DATA, 0xFF);
        i = 0; /*Get file name*/
        do {
                ch = receive_char();
                fname[i++] = ch;
        } while(ch != '\0');
        i = 0; /*get file length*/
        do {
                ch = receive_char();
                flen[i++] = ch;
        } while(ch != '\0');
        len = atol(flen);
        printf("Dosya: %s Uzunluk: %ld\n", fname, len);
        if ((fp = fopen(fname, "w+b")) == NULL) {
                printf("Cannot open file..!");
                exit(1);
        }
        for (n = 0; n < len; ++n) { /*get file info*/
                ch = receive_char();
                fputc(ch, fp);
        }
        printf("Dosya aktarıldı..\n");
        fclose(fp);
}

#endif

#ifdef SEND

void main(int argc, char *argv[])
{
        FILE *fp;
```

```
        char flen[30];
        BYTE ch;
        long n;
        long len;
        int i;

        if (argc != 2){
                printf("wrong number of arguments..\n");
                exit(1);
        }
        if ((fp = fopen(argv[1], "r+b")) == NULL) {
                printf("Cannot open file..\n");
                exit(1);
        }
        fseek(fp, 0, 2);/*get file size*/
        len = ftell(fp);
        ltoa(len, flen, 10);
        rewind(fp);
        outp(LPT1DATA, 0);
        i = 0;
        do {                    /*send file name*/
                sendchar(argv[1][i++]);
        } while(argv[1][l++] != '\0');
        for (n = 0; n < len; ++n) { /*send file info*/
                ch = fgetc(fp);
                sendchar(ch);
        }
        fclose(fp);
}

#endif
```

Küçük Chat Programının Açıklaması

Bu programda da sendchar ve receive_char fonksiyonları kullanılmıştır. Gönderen taraf klavyede bir tuşa basar. Basılan tuş sendchar fonk ile yollanır, karşı taraf bunu receive_char fonksiyonu ile alır ve ekranda görüntüler. Örnek program tek dosya halinde verilmiştir. İki tane main #ifdef SEND ve #ifdef RECEIVE ön işlemci bloklarına alınmıştır.

```
/*----------chat.c------------*/
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <stdlib.h>

#define LPT1DATA   0x378
#define LPT1STAT    0x379
```

```
typedef unsigned char BYTE;

int receive_char(void)
{
	BYTE dpos;
	BYTE rchar;

	do {
		dpos = inp(LPT1STAT);
	} while((dpos >> 7) == 1);
	dpos = inp(LPT1STAT);
	dpos = (dpos << 1) >> 4;
	rchar = dpos;
	outp(LPT1DATA, 0x0);
	do {
		dpos = inp(LPT1STAT);
	} while((dpos >> 7) == 0);
	dpos = inp(LPT1STAT);
	dpos = (dpos << 1) >> 4;
	rchar |= dpos << 4;
	outp(LPT1DATA, 0xFF);
	return rchar;
}

void sendchar(char ch)
{
	BYTE dpos;

	do {
		dpos = inp(LPT1STAT);
	} while((dpos >> 7) == 1);/*wait until the receiver is ready*/
	dpos = ch & 0x0F;
	dpos |= 0x10;
	outp(LPT1DATA, dpos);/*send lower 4 bit to receiver*/
	do {
		dpos = inp(LPT1STAT);
	} while((dpos >> 7) == 0);/*wait until the receiver is ready*/
	dpos = ch >> 4;
	dpos &= 0x0F;
	outp(LPT1DATA, dpos);/*send upper 4 bit to receiver*/
}

#ifdef RECEIVE

void main(void)
{
```

```
	BYTE n;

	outp(LPT1DATA, 0xFF);
	do {
		n = receive_char();
		if (n == '\r')
			putchar('\n');
		else
			putchar(n);
	} while(n != 'q');
}
#endif

#ifdef SEND

void main(void)
{
	BYTE ch;

	outp(LPT1DATA, 0);
	for(;;){
		ch = getch();
		if (ch == '\r');
			putchar('\n');
		else
			putchar(ch);
		sendchar(ch);
		if (ch == 'q')
			break;
	}
}

#endif
```

Paralel Port ve IRQ

Paralel portun 10 nolu ACK ucu IRQ oluşturmak için kullanılabilmektedir. Eğer 10 numaralı uç 0 Volta çekilirse (bu uç yalnızca okunabilir ve default olarak 1'dir.) 8259 yoluyla 7 numaralı IRQ çağırılır. Bu IRQ 0F numaralı kesme kodunun çağırılmasına yol açar. Bu durumda transfer işlemi şöyle de yapılabilir. Paralel porta bilgi geldiğinde alıcı taraf için kesme çağırılır, kesme kodu da alma uçlarını okuyarak bilgiyi alır. Bu işlemde dikkat edilecek iki durum vardır:
1-)Normal olarak paralel port işlemcisi 8259'u uyararak kesme oluşmasına yol açmaz. Bunu mümkün hale getirmek için paralel port işlemcisinin command register'ının 4 numaralı biti1 yapılmalıdır.
2-)IRQ oluştuktan sonra bir daha kesmenin gelebilmesi için kesme kodunun kesme denetleyicisine 20H değerini göndermesi gerekir. (Yani 20H portuna 20H değerinin gönderilmesi gerekir.)

3-)Başlangıç konumunda 0F numaralı kesme 8259 tarafından pasif hale sokulmuş olabilir. Bunu aktif hale getirebilmek için 21H portuna 7 numaralı bitin 1 yapılmasını sağlayacak bir değer gönderilmelidir.

Paralel Portun Yetmediği Durumlarda Çıkış Elde Etmenin Yöntemleri

Bu tür durumlarda özel kartlar kullanılabilir. İçinde 8255 işlemcinin bulunduğu pek çok basit kart piyasada bulunmaktadır. Bir 8255 işlemcisi ile 3*8 giriş ya da çıkış ucu elde edilebilir. Piyasada satılan 3, 4, 5... kadar 8255 işlemcisi bulunmaktadır.

Asenkron Seri Haberleşmenin Özellikleri

Asenkron seri haberleşme uygulamada en fazla kullanılan haberleşme yöntemidir. Bu haberleşmede bytelar arası bekleme zamanı rastgele olabilir. Ancak byteların bitlewri eşit hızda gönderilip alınır. Hız saniyede gönderilen bit sayısı ile ölçülür.(bps, baud). En çok kullanılan haberleşme hızları: 1200, 2400, 4800, 9600, 19200, 38400... En önemli iki sorun eş zamanlılığın sağlanması ve bilgi gönderilmeyen zaman aralığının saptanmasıdır. Eş zamanlılık kararlı çalışan işlemcilerle kolaylıkla sağlanabilir. Bilginin gönderilmediği durumda hat 1'de beklemektedir. Alıcı taraf bilginin kodlanmaya başladığını hattın 0'a çekilmesinden anlar. Bundan sonra alıcı taraf belirlenen hızda hattı örneklemeye başlar(hatta bakar) ve data bitlerini elde eder. Bilginin gönderilmeye başladığını anlatan, hattın 0'a çekilmesiyle oluşan ilk 0 biti data bitlerine dahil değildir. Buna start bit denir. Start bit her zaman logic 0 olmak zorundadır. Data bitleri gönderildikten sonra iletişimin döngüsel olarak devam edebilmesi için hattın tekrar 1 durumuna çekilmesi gerekir. Data bitlerinden sonra hattın 1'e çekilmesine yarayan logic 1 seviyeli bit'e stop bit denir. Stop bit start bit gibi mutlaka bulunmak zorundadır. Bu durumda 8 bitlik bir transfer için aslında en az 10 bit gönderilmektedir. Bps cinsinden verilen hız değeri içerisine bu bitler de dahildir. Stop bit sayısı 1 yada 2 olarak belirlene blinir. 2. Stop bit aslında gerekli değildir ancak bytların beklemesiz peşi sıra gönerildiği durumlarda 2 stop bit sistemi kullanılırsa alıcı tarafa daha fazla zaman verilmiş olur böylece iletişim butür durumlarda dahada sağlamlaştırılırç. Uygulamada 2 stop bit fazla kullanılmaz. Bir byte lık bilginin gönderilmesi sonucunda alan tarafın bu bilgiyi doğru alıp almadığı tespit edile bilinir. Seri haberleşmede bunun için parit yöntemi denilen bir yöntem kullanıla bilinir. Tabi iltişimde böyle bir hata yakalama yönteminin kullanılıp kullanılmayacağı isteye bağlıdır. Parity yöntemi kullanılacaksa data bitlerinden sonra ismine parity biti denilen bir bit daha gönderilir. Parity yöntemi tek ve çift olmak üzere ikiye ayrılır. Çift parity yöntemi gönderilen data bitleri içerisindeki 1 olan bitlerin sayısını çift yapacak biçimde bit parity biti gönderme durumudur. Örneğin data bitleri içerisinde 3 tane 1 varsa ve çift parity yöntemi kullanılıyorsa parity biti olarak 1 gönderilir. Tek parity yöntemi bunun tam tersidir. Data bitlerindeki 1'lerin sayısını tek yapacak biçimde parity biti gönderilmesi durumudur. Bu yönteme göre alıcı taraf önce data bitlerini alır, hangi parity bitinin gelmesi gerektiğini hesaplar. Eğer bir uyuşmazlık varsa alınan byte değerinin yanlış olduğunu anlar. Tabii parity yöntemi hataları kesinlikle bulabilecek bir yöntem değildir. Örneğin iki bit bozulmuşsa böyle bir hatayı bulamaz. İletişimde parity kontrolünün yapılıp yapılmayacağının yapılacaksa tek ya da çift mi yapılacağı ortaklaşa tespit edilmelidir. Bu durumda 1 byte'lık bilginin gönderilmesi sırasında sırasıyla şu bitler gönderilmektedir: Start bit, data bitleri, parity biti, stop bitleri. Aslında data

bitleri 8 bit olmak zorunda değildir. Gönderilecek bilginin türüne göre 5, 6, 7 bit de olabilir. Sonuç olarak bir iletişimde iki tarafında şunları kesin olarak bilmesi gerekir:

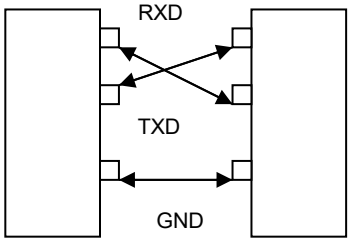
1. Kaç data biti kullanılacak
2. Stop bitlerinin sayısı
3. Parity yönteminin kullanılıp kullanılmayacağı

UART İşlemcileri

Bir bytelık bilginin bitlerine ayrılarak tespit edilen parametreler doğultusunda gönderilmesi ismine UART(universal asynchronous receiver and transmitter) denilen bir işlemci sayesinde yapılmaktadır. UART işlemcisi pek çok firma tarafından aynı özelliklere sahip olacak biçimde üretilmiştir(National 15560, Intel 8250). UART işlemcisi kabaca şöyle kullanılır:

1. Önce data bitlerinin sayısı, stop bitlerinin sayısı, parity kullanılma durumu, örnekleme hızı tespit edilerek işlemci bu parametreler doğrultusunda programlanır.
2. Bilgi UART işlemcisine 1 byte şeklinde gönderilir. İşlemci bunu bitlerine ayırıp karşı tarafa belirlenen parametreler doğrultusunda kodlar.
3. UART işlemcisi karşı taraftan bit bit gelen bilgiyi alarak bunu byte bilgisine dönüştürür ve saklar.

İşlemci 28 pin'e sahiptir. Bu uçlardan 3'ü iletişimde asıl önemli görevi üstlenir. Bu uçları RXD, TXD, GND uçlarıdır. İki UART işlemecisi karşılıklı olarak şöyle bağlanabilir.



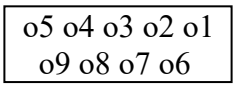
UART işlemcileri full dublex çalışacak şekilde tasarlanmıştır.

Seri Port Kullanımı

Seri port 9 ya da 25 pinli olarak bilgisayarlarda bulunmaktadır. Bilgisayar tarafı erkek konnektör biçimindedir. Zaten 25 pinli konnnektörlerin de yalnızca 9 ucu bağlıdır. Seri portlara UART işlemcisi bağlıdır. Yani seri portların uçları UART işlemcisiyle aymnıdır. UART'ın çıkışları TTL seviyesindedir. Ancak seri porta verilmeden önce bir RS232 dönüştürücü devresinden geçirilir. RS232 gerilim seviyeleri TTL gibi değildir. +- 15 V civarındadır. RS232 dönüştürücüsü çift yönlüdür.

Seri Port Pinlerinin İsimlendirilmesi ve Anlamlandırılması

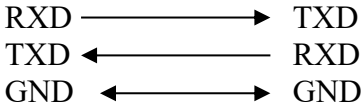
9 pin:



Seri port ve UART pinlerinin isimleri temel olarak modem haberleşmesi temel alınarak verilmiştir.

1=>CD(Carrier detect)
2=>RXD(Receive data)
3=>TXD(Transmit data)
4=>DTR(Data terminalk ready)
5=>GND(Ground)
6=>DSR(Data set ready)
7=>RTS(Request to send)
8=>CTS(Clear to send)
9=> RI(Ring indicator)

Seri porttan bilgi aktaran bir lap link kablosu en ekonomik olarak 3 uçlu olabilir.



UART İşlemcisinin Programlanması

UART işlemcisinin programlanmasının 3 yolu vardır:

1. C derleyicilerinin kütüphanelerinde programlayan, bilgi gönderen ve alan çeşitli fonksiyonlar vardır. O fonksiyonların kullanılması öğrenilir ve çağırılır.
2. BIOS içerisinde(EPROM'da) UART işlemcisini programlayan çeşitli kesme kodları vardır. Bu kesmeler çağırılarak işlemler yürütülebilir. Bu işlem için 14H BIOS kesmesinin fonksiyonları kullanılmaktadır.
3. Doğrudan UART işlmecisinin portlarına bilgi göndererek en aşağı seviyeden programlama işlemini gerçekleştirmek. (Zaten 14H kesmesi de bu biçimde işlemleri yürütmektedir.)

UART İşlemcisinin Aşağı Seviyeli Programlanması

UART işlemcisinin içerisinde haberleşme portlarına bağlı çeşitli register'lar vardır. Bu portlara bilgi gönderildiğinde bilgi register'lara yazılır. Benzer biçimde portlar okunduğunda aslında o register'ların içerisindeki bilgi okunmaktadır. UART işlemcisinin programlama yazılım yoluyla erişilebilen 10 tane register'ı vardır. Register'ların birer port numarası bulunur. Seri portlara bağlı UART işlemcilerinin taban port adresleri BIOS haberleşme alanı içerisinde yazmaktadır. Ancak çok büyük olasılıkla taban port adresleri şöyledir:

COM1=>3F8
COM2=>2F8
COM3=>3E8
COM4=>2E8

Bir Byte Bilginin Gönderilmesi İşlemi

Bir byte'lık bilgi UART işlemcisinin THR register'ına gönderilir. İşlemci bu gönderilme işleminden sonra otomatik olarak bu bilgiyi bitlerine ayırarak TXD ucundan kodlayacaktır. Ancak gönderici taraf çok hızlı bir biçimde bilgileri THR içerisine yazarsa problemli bir durum ortaya çıkar. Yani THR register'ı UART tarafından boşaltıldıktan sonra tekrar bilgi gönderilmektedir.

Bir Byte Bilginin Alınması İşlemi

UART bit bit elde ettiği bilgiyi byte'a dönüştürerek RBR register'ının içerisne yazar. Yani bilginin alınabilmesi için RBR register'ının okunması gerekir. Tabii programcı RBR register'ını yalnızca yeni bir bilgi geldiğinde okumalıdır. O halde UART yeni bir bilgi geldiğini bir biçimde haberdar eder.

LSR REGISTERI (line status register)

Bu register genel olarak 8250'nin içinde bulunduğu durum hakkında bilgi vermektedir. Bu register'ın her bir biti bir durum hakkında bilgi verir. Bitlerin anlamları:

7	6	5	4	3	2	1	0
Xxxxxx	TEMT	THRE	BI	FE	PE	OE	DR

DR biti: Bu bit bir ise RBR registerına gelen bilgi yerleştirilmiştir dolayısı ile bu registerın okunup bilginin alınması gerekllidir. Bu durumda bilgi alma işi için sürekli LSR portu okunur eğer DR biti 1 ise RBR register'ı okunarak bilgi elde edilir.

```
----------------------lsr-dr.c-----------------------
#define BASE       0x3F8          /* Com1 */
#define LSR        (BASE + 5)     /* LSR port numarası */

void fonk(void)
{
   int x;
   char ch;

   do {
      x = inp(LSR);
   } while (!(x & 1));
   ch = inp(RBR);
}
-----------------------------------------------------
```

LSR register'ı her okunduğunda içerisindeki bitler otomatik olarak sıfırlanır. Yani programcının sıfırılamasına gerek kalmaz.

OE biti (overrun error): Bu bit 1 ise şöyle bir olay gerçeklerşmiştir. 8250'ye bir bilgi gelmiş ve RBR de tutulmuştur ancak okunamadan yeni gelen bir bilgi eskisini bozmuştur. Bu overrun error olarak adlanır.

PE biti (parity error): Bu bit 1 ise bilgi parity hatasından dolayı yanlış alınmıştır.

FE biti (framing error): Start bit data bitleri ve parity ve stop bitlerinin hepsine birlikte "frame" denmektedir. Start bitten sonra data ve parity bitleri alınır ve bundan sonra stop bit için hat 1'e çekilmelidir. Hat 1'e çekilmemişse stop bit alınamamıştır. Dolayısı ile frame bozuktur o zaman bu bit 1 olur.

BI biti (brake ıntterupt): Haberleşmede hat koparsa UART sürekli sıfır değerini okur işte start bit data bitleri parity ve stop bitlşeri uzunluğu kadar süre sıfır okunmuş ise UART durumdan şüphelenerek bu biti bir yapar. Yani bu bit özellikle hat kopması gibi durumları tespit etmek amacıyla kullanılır.

THRE biti (transmitter holding register empty): Bilgi gönderirken önceki bilginin THR registerından alınmış olduğunu garantilemek gerekir. Eğer UART THR register'ındaki bilgiyi almışsa bu bit 1'ler. Yani yeni bir bilgi THR register'ına yazabiliriz.

```
----------------------lsr-thre.c--------------------------
#define BASE       0x3F8            /* Com1 */
#define LSR        (BASE + 5)        /* LSR port numarası */


void fonk(void)
{
   int x;
   char ch;

   do {
      x = inp(LSR);
   } while (!(x >> 5) & 1));
   outp(THR, ch);
}
-----------------------------------------------------------
```

TEMT biti (transmitter empty): THRE biti bilginin yanlızca THR register'ından alındığını anlatır. Bu bilgi alındıktan sonra bitlere ayrıştıtılıp gönderilecektir. Oysa bu bitin bir olması aynı zamanda thr register'ı içindeki bilginin alınarak gönderilmiş olduğunu anlatır. Yani bu bit bir ise THR ve TSR register'larının her ikisi birden boştur.

UART'IN HIZININ AYARLANMASI

UART'ın hızının ayarlanması için iki registere'a bilgi gönderilmesi gerekir. Bunlar DLL (divisor Latch Least) ve DLM (divisor latch most) . UART işlemcisinin uçlarına bağülanan kristal frekenası PC sistemlerinde 1843200 hertz dir.
Aşağıdaki denklemden elde edilen 16 bitlik değerin yüksek anlamlı byte'ı DLM düşük olanı DLL içine yerleştirilir.
16 bit register değeri = 1843200 / (16 * baud rate)
bu denklemden Pc sistemleri için UART ın en yüksek ve en düşük çalışma hızları tespit edilebilir. En yüksek hız DLM=00 DLL=01 ile 115200 dür. En düşük hız ise DLM=FF DLL=FF iken yaklşık 1 dir

En çok kullanılan hızları DLL ve DLM değerleri:

HIZ	DLM	DLL
2400	00	30
4800	00	18
9600	00	0C
19200	00	06
38400	00	03
57600	00	02
115200	00	01

DATA STOP VE PARITY İŞLEMLERİNİN AYARLANMASI

Bu işlemler için LCR registerı kullanıllır. Bu registerda 8 bittir. Ve belirlemeler bu bitleri set ederek yapılır.

Düşük anlamlı iki bit data bitlerinin sayısını oluşturmak için kullanılır.
00→5 bit
01→6 bit
10→7 bit
11→8 bit

7	6	5 (sabit parity)	4	3 (Parity kontrol)	2 (Stop bit)	1	0
	Sürekli sıfırdır.	0→s.parity yok 1→s.parity var	0→ tek 1→ çift	1→yapılacak 2→yapılmayacak	0→1stopbit 1→2stopbit	\\\\\\\\\\ \\\\\\\\\\	\\\\\\\\

4nolu bit tek mi yoksa çift mi parity yönteminin kullanılacağını belirlemede işe yarar. Yapılmayacaksa bu bitin anlamı yoktur.
5nolu bit bir ise parity biti olarak sürekli bir yada sıfır gönderilir. Buna sabit parity yöntemi denir.
7 NOLU BITIN ANLAMI: Bu bit bazı registerların kullandığı ortak port numaralarını ayrıştırmak amacı ile kullanılır.eğer BASE+0 portuna bilgi gönderiliyorsa bu bilgi THR DLL registerına gönderiliyordur. İşte LCR nin 7 nolu biti sıfır ise THR bir ise dll registerına gönderilecektir. Benzer bir biçimde base +1 portu hem okunabilir hem yazılabilitr bir porttur. Ve

görünüşte bu porta DLM ve IER registerlarının her ikisi birden bağlıdır. İşte LCR registerının 7 nolu biti bir ise DLM 0 ise IEM registerının kullanımı söz konusudur.

SERİ PORT v IRQ

Seri port yoluyla donanım kesmenin çağırılması sağlanabilir. COM1 ve COM3 için IRQ 4 çağırılır. IRQ 4'ün kesme numarası 0CH'tır. COM2 ve COM4 için IRQ 3 çağırılır. IRQ 3'ün kesme numarası 0BH'tır. Yani birinci 8259'un 4 numaralı ucu COM1 portuna bağlı olan UART'a 3 numaralı ucu ise COM2 portuna bağlı olan UART'a(8250) bağlıdır. Uygulamada pek çok seri port haberleşmesi kesme sistemiyle yapılmaktadır. Yani seri portlara bağlı olan birimler UART işlemcisini uyarırlar, UART işlemcisi ise 8259 işlemcisine kesme isteğini bildirir. Sonuç olarak kesme dışsal birim tarafından çağırtılmış olur. Örneğin mouse devresi böyle bir kesme mekanizmasıyla çalışır. Yani mouse seri porta periyodik olarak IRQ isteği gönderir, kesme kodu da mouse okunu hareket ettirir. UART işlemcisinin kesme çağırması bir takım olayların gerçekleşmesine bağlıdır. Hangi olaylar gerçekleştiğinde kesme çağırılacağı IER register'ına değere yaılarak belirlenir. IER register'ının düşük anlamlı 4 biti kullanılır, diğerleri de 0'dır.

0 Numaralı Bit:

Bu bit 1 ise RBR register'ına yeni bir bilgi geldiğinde kesme çağırılır.

1 Numaralı Bit:

Bu bit 1 ise THR register'ı boşalınca kesme çağırılır.

2 Numaralı Bit:

Bu bir 1 ise UART işlemcisinin durumu değişince kesme çağırılır. Yani LSR register'ındaki bitlerin durumu değişmişse kesme çağırılır.

3 Numaralı Bit:

Bu bit 1 ise MSR register'ındaki herhagi bir bit konum değiştirmişse çağırılır.

Bu bitlerin hepsi 1 yapılırsa kesme pek çok sebepten dolayı gelmiş olabilir. İşte kesme kodu kesmenin nedenini IIR(interrupt identification register) register'ına bakarak anlar. Bu register'ın 1 ve 2 numaralı bitleri kesmenin nedenini açıklar.

00→ MSR değişmesi nedeniyle
01→ THR boşalması nedeniyle
10→ RBR'ye bilgi gelmesi nedeniyle
11→ Hata nedeniyle

DOSYALARIN SERİ PORT KULLANILARAK TRANSFER EDİLMESİ

Gönderen taraf dosyanın karakterleri hızlı bir biçimde yazıyorsa fakat alan taraf yavaş ise bilgiyi doğru bir biçimde alamayacaktır (alan taraf overrun error durumuyla karşılaşır). Yani gönderen hızlı alan yavaşsa sorun oluşacaktır. Bu tür senkronizasyon problemleri başka sebeplerden de ortaya çıkabilir. Bu durumda bir transfer protokolü kullanmak gerekir. Kullanılan en basit transfer protokolü z-modem protokolüdür. Bu protokole göre gönderen taraf alan taraftan onay bekler. Onay bekleme durumu alan tarafın gönderen tarafa bir karakter göndermesi işlemidir. Böylece iki sistem birbirinden farklı hızlarda bile olsa yazılım yoluyla senkronizasyon sağlanmış olur. kısa mesafe bilgisayar haberleşmesi için senkronizasyonu donanım yoluyla

sağlamak da mümkündür. Özellikle RTS/CTS uçları karşılıklı bağlanarak donanım yoluyla senkronizasyon sağlanabilir.

MODEMLER HAKKINDA KISA BİLGİLER

Modemler seri porta bağlanarak çalıştırılırlar. İçsel ve dişsal olmak üzere iki tür modem vardır. Dışsal modemler bir seri kablo ile bilgisayara bağlanırlar. Oysa içsel modemler genişletme yuvalarına takılırlar. İçsel modemler yine kendi içersinde seri porta sahiptirler. Yazılımda kullanım farklılıkları yoktur. Yani açıkça seri porta bağlanmış olmasalar da bağlanmış varsayılarak programlamalıdırlar. Modemler sayısal bilgiyi normal telefon haberleşmesine çevirerek uzak iletişimi sağlarlar. Mademki modem UART işlemcisiyle bilgisayara bağlanmıştır. O halde modeme bilgi göndermek için aslında bilgi UART işlemcisine gönderilir. Benzer biçimde modem bilgiyi elde ettiğinde UART işlemcisine kendisi yazar, biz de işlemciden bilgiyi alırız.

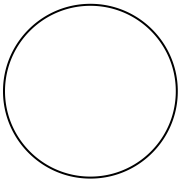
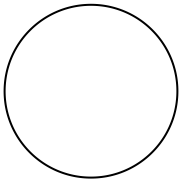
MODEMİN PROGRAMLANMASI

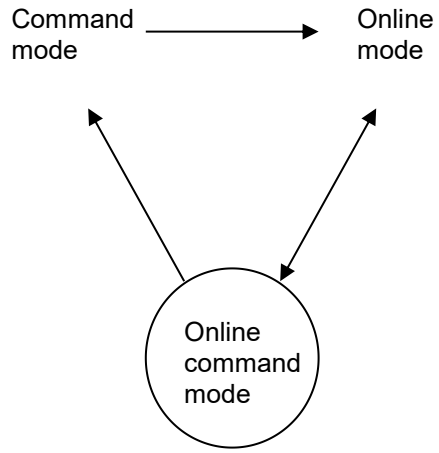
Modemin programlanması ve kullanılması yüksek seviyeli komutlarla yapılır. Bir numarayı arama, aramaya cevap verme, bilgi gönderme ve bilgi alma işlemleri yazı biçiminde komutlarla yürütülür. Bu türe standart komutlara AT komutları denir. Başlıca AT komutları şunlardır(bütün komutların başı AT ile başlar daha sonraki karakter komutun ne olduğunu açıklar):

Komut	İşlevi
ATA	Aramaya yanıt verir(answer). Telefon çaldığı zaman telefonun çaldığı tespit edilince bağlantının kurulması için bu komut gönderilir.
ATD	Arama komutu(dial). Karşı tarafı aramak için kullanılır.
ATZ	Modemi reset etmek amacıyla kullanılır.

MODEMİN ÇALIŞMA MODLARI

Bir modemin 3 modu vardı:	
1. Command mode	Modem reset edildiğinde command moddadır. Bu mode'da modeme AT komutları gönderilebilir. Örneğin bu modda arama yapılabilir, aramaya cevap verilebilir. Bağlantı yapıldıktan sonra modem otomatik olarak online moda geçer.
2. Online mode	Bu modda modeme bir bilgi gönderildiğinde modem bunu komut olarak kabul etmez, karşı tarafa gönderir. Yine bu modda modemden alınan bilgi karşı tarafın gönderdiği bilgidir. Haberleşme bittikten sonra modem yine command moda geçirilir ve sonra bağlantı kesilir. Online moddan online-command moduna geçiş yapılabilir.
3. Online-command mode	Bu modda bağlantı kesilmez ancak modem AT komutlarını alma moduna geçer.



Modemin command modda iki durumu vardır. Echo mode denilen modda modeme her gönderilen komut karşılığında modem önce gönderilen komutun kendisini yollar. Sonra komut karşılığında yapılan işlemi anlatan küçük bir cevap yazısı gönderir. Echo mode kapatılırsa modem gönderilen komutu bir daha göndermez. Yalnızca cevap yazısını gönderir.

bioscom FONKSİYONUNUN KULANIMI

Seri porta bilgi gönderip almak için aşağı seviyeli programlama yerine doğrudan 14H kesmesi çağırılabilir ya da Borland derleyicilerinde bioscom isimli fonksiyon kullanılabilir. Prototipi bios.h içerisindedir.

int bioscom(int cmd, char abyte, int port);

cmd→ 0 veya _COM_INIT
cmd→ 3 veya _COM_STATUS
cmd→ 1 veya _COM_STATUS
cmd→ 2 veya _COM_RECEIVE

Önce com portu cmd 0 ile initialize etmek gerekir. Üçüncü parametre com port numarasını göstermektedir. 0 ise com1, 1 ise com2. İkinci parametre eğer com port set edilecekse set değerlerini alır. Eğer bilgi gönderilecekse gönderilcek bilgiyi temsil eder. Eğer bilgi alınacaksa ikinci parametrenin hiçbir önemi yoktur. Örneğin:

bioscom(_COM_INIT, _COM_CHR8 | _COM_STOP1 | _COM_NOPARITY | _COM_9600, 0);	Com portu 9600, noparity, 8 data biti ve 1 stop bitine set eder.
bisocom(_COM_SEND, 'x', 0);	x karakterini com1 portuna gönderir
y = bioscom(_COM_RECEIVE, 0, 0);	Com portundan bilgi alır ve y'ye aktarır.
State = bioscom(_COM_STATUS, 0, 0);	UART'ın LSR ve MSR register değerleri alınır.

Örnek:Bir string'i com porta gönderen program.

```
/*-------------bioscom.c-----------------*/
void writecomstr(const char * str)
{
   char stat;

   while(*str != '\0'){
      do {
         stat = bioscom(_COM_STATUS, 0, 0);
      }while(!((stat >> 14) & 1));
      bioscom(_COM_SEND, *str, 0);
      ++str;
   }
}

void main(void)
{
   writecomstr("ATD2883520\r");/*com port'a modem için dial komutu numarası yollar*/
}
/*-----------------------------------------*/
```

Ancak comporta bilgi göndermenin en kolay yolu önce com portu bioscom fonksiyonu ile set etmek, daha sonra fprintf gibi dosya fonksiyonlarıyla FILE adresi yerine stdaux vererek kullanmaktır. Örneğin:

```
fprintf(stdaux,"ATD2883520\r");
```

AT komutları gönderildikten sonra komutun bittiğini göstermek için '\r' karakterini göndermek gerekir.

İLERİ DÜZEY EKRAN İŞLEMLERİ

Standart Text Mode

80x25 çözünürlüğe sahip text moda standart text mode denir. Standart text mode neredeyse bütün bilgisayarlar tarafından desteklenmektedir. Bu bölümdeki yalnızca standart text mode'u içermektedir. Standart text mode'a ilişkin 3 tane video mode vardır. Bunlar 2, 3, 7 ekran modlarıdır. 2 ve 7 numaralı modlar mono modlardır, 3 renkli text moddur. Mono modlarda renk gözükmez. Renkli modda her karakter matrisinin şekil ve zemin rengi söz konusudur. 16 şekil 8 zemin rengi vardır ve bu renkler önceden tanımlanmıştır. Zemin renkleri şekil renklerinin ilk 8 tanesidir. Yani aynı anda en fazla 16 renk görülebilir. Yüksek sayıda renk ancak grafik modda elde edilebilir.

Standart Text Mode'da Ekran Belleğinin Organizasyonu

Bugün kullanılan VGA grafik kartlarının üzerinde ismine ekran belleği denilen bir RAM bölgesi vardır. Ekran belleği DOS bölgesinin bitiminden başlar(A0000) ve 1 MB adres alanı

içerisindedir. Yani bu RAM bölgesi ekran kartının üzerinde olmasına karşın sanki normal simm bloklarının içerisindeymiş gibi işlem görür. Bu bölgeye uzak göstericilerle erişebiliriz. Ekran belleği A0000 adresinden başlamasına karşın renkli standart text mode'da belleğin aktif olan bölgesi B8000'dan, renksiz text mode'da ise B0000'dan başlar. Ekran kartı ekran belleğinin aktif bölgesindeki bilgileri monitöre belirlenen bir hızda gönderir. Monitörde bnu bilgileri ekranın belirlenen bölgesine basar. Yani ekranda bir bilginin çıkmasını sağlamak için tek yapılacak şey bilgiyi bu ekran belleğine yazmaktır. Bilgi buraya yazıldıktan sonra ekran kartı tarafından monitöre yollanır ve monitör tarafından gösterilir. PC'lerde hangi programlama dili söz konusu olursa olsun sonuçta ekrana yazma yapan fonksiyonlar ya da komutlar bilgiyi ekran belleğine yazmak zorundadırlar. Ekran belleğindeki her iki byte ekrandaki bir matrisel hücreyi temsil eder. Ekran belleğinin aktif bölgesini gösterecek bir gösterici tanımlanabilir. Renkli text mode için bu göstericinin içerisine segmen ve offset olarak şu değerler yerleştirilmelidir. scrp ekran belleğinin aktif bölgesini gösteriyor olsun.

```
char far *scrp = (char far *)0xB8000000;
```

Bu göstericinin (row,col) hücresini göstermesi için yapılması gereken:

```
scrp = scrp + row * 160 + col * 2;
```

Bu iki byte'ın birinci byte'ı ekranın belirlenen bölgesinde çıkacak ascii karakterini, ikinci byte'ı ise rengini belirlemekte kullanılır. Örneğin aşağıdaki fonksiyon ekranın istenilen bölgesine isenilen karakteri basar.

```
/*------------ekran1.c-----------------*/
void _writec(int row, int col, char ch)
{
   char far *scrp = (char far *)0xB8000000;

   scrp += row * 160 + col * 2;
   *scrp = ch;
}

void main(void)
{
    _writec(10, 10, 'x');
}
/*---------------------------------*/
```

Renk Bilgisinin Oluşturulması

Düşük anlamlı 4 bit şekil daha sonraki 3 bit zemin ve en yüksek anlamlı bit ise yanıp sönme bilgisini içerir.

B	Z	Z	Z	Ş	Ş	Ş	Ş

Renkler:

```
BLACK       0x0
BLUE        0x1
GREEN       0x2
CYAN        0x3
RED         0x4
MAGENTA     0x5
BROWN       0x6
WHITE       0x7
GRAY        0x8
LBLUE       0x9
LGREEN      0xA
LCYAN       0xB
LRED        0xC
LMAGENTA    0xD
YELLOW      0xE
LWHITE      0xF

/*------------ekran2.c-----------------*/
void _writec(int row, int col, char ch, int color)
{
   char far *scrp = (char far *)0xB8000000;

   scrp += row * 160 + col * 2;
   *scrp++ = ch;
   *scrp = color;
}

void main(void)
{
   clrscr();
   _writec(10, 10, 'x', 0x1F);
}
/*----------------------------------------*/
```

conio Kütüphanesinin Kullanımı

Prototipleri conio.h içerisinde bulunan ve temel ekran işlmelerini yapan bir grup fonksiyon vardır. Bu fonksiyonlar borland derleyicilerine özgüdür ve yalnızca DOS altında kullanılabilir.

textattr, textbackground, textcolor Fonksiyonlar

Bu fonksiyonlar cprintf ile yazılan yazının rengini belirlemekte kullanılırlar. cprintf fonksiyonu burada belirlenmiş olan renklere bakarak yazma işlemini yapar ve kullanımı printf fonksiyonunun aynısıdır.

textattr fonksiyonunun tek parametresi vardır. Bu parametre şekil zemin ve blink bilgisini içerir.

textbackground şekil rengine dokunmadan sadece zemin rengini değiştirmede kullanılır.

textcolor yalnızca şekil rengini değiştirmek amacıyla kullanılır.

Renk İfadesinin Düzenli Bir Biçimde Yapılması

Şekil ve zemin renkleri düzenli bir biçimde sembolik sabit olarak belirlenirse bit OR işlemiyle renkler kolaylıkla belirlenebilir. Örneğin şekil renkleri ve zemin renkleri şöyle tasarlanabilir.

```
0x01    0x0
0x02    0x10
0x03    0x20
..      ..
0x0F    0x70
```

Bu durumda aşağıdaki gibi bir kullanım çok kullanışlı olur.

```
BLUE_ | WHITE       /*      zemin rengi mavi şekil rengi beyaz    */
```

Ekran belleği ile daha düzenli çalışabilmek için ekran belleğindeki her hücre bilgisi bir yapıyla temsil edilebilir.

```
/*-----------ekran3.c-------------*/
typedef struct _CHR{
   char chr;
   char attr;
}CHR;

void _writec(int row, int col, int ch)
{
   CHR far *scrp = (CHR far *)0xB8000000;

   scrp += row * 80 + col;
   scrp -> chr = ch;
}

void main(void)
{
   _writec(10, 20, 'y');
}
/*---------------------------------*/
```

Kütüphane Oluşturma

Sürekli yinelenen işlemlerin kolay yapılmasını sağlamak için çeşitli fonksiyonları yazmak ve biriktirmek gerekir. Bu fonksiyonların birbirlerini çağırmasıyla yüksek seviyeli işlemler basit birkaç fonksiyona indirgenebilir. Böylece büyük projeler önceden oluşturulmuş kütüphaneler oluşturularak kolayca hazırlanabilir. Kütüphane oluşturma işlemleri için disk üzerinde iyi bir organizasyon oluşturulmalıdır. Bir kütüphane içindeki fonksiyonları düzenli bir dokümantasyonunun yapılması gerekir. Yani fonsiyonların parametrelerinin ne anlamlara geldiği, amaçlarının ne olduğu, geri dönüş değerlerinin ne olduğu açıkça dokümante edilmelidir.

Bir kütüphane içerisinde 3 seviye fonksiyon bulunmalıdır.

1-)Aşağı seviyeli fonksiyonlar
2-)Orta seviyeli fonksiyonlar
3-)Yüksek seviyeli fonksiyonlar

Aşağı seviyeli fonksiyonlar en temel işlemleri yapan fonksiyon grubunu oluşturur. Orta seviyeli fonksiyonlar aşağı seviyeli fonksiyonları kullanarak orta seviyeli işlemler yaparlar. Yüksek seviyeli fonksiyonlar kullanıcının çağırdığı çok yüksek seviyeli işlemleri yapan fonksiyonlardır.

Kütüphane Dosyalarının Tasarlanması

Kütüphane dosyaları C ve H uzantılı olmak üzere iki şekilde tasarlanmalıdır. C uzantılı dosya içerisinde fonksiyon tanımlamaları bulunur. H uzantılı dosya içerisinde ise genel olarak nesne yaratmayan bildirimler yerleştirilir. Yani sembolik sabitler, yapı, birlik, bit alanı bildirimleri, typedef bildirimleri, fonksiyon prototipleri gibi bildirimler bulunmalıdır. Bu H uzantılı dosya C uzantılı dosya içerisinden include edilir. Artık istenirse bu dosya derlenerek LIB uzantılı dosyaların içerisine katılabilir. Burada oluşturulan H uzantılı dosya yalnızca bu kütüphaneyi derlemek amacıyla değil bu kütüphanenin kullanıldığı durumlarda da include edilmelidir.

Scroll İşlemleri

Dikdörtgensel bir bölge içerisindeki tüm karakterlerin sağa sola yukarı aşağı kaydırılmasına scroll işlemi denir. Scroll işlemleri 10H kesmesinin 6 ve 7 numaralı fonksiyonlarıyla yaptırılır.

AH	6, 7 (6 aşağı, 7 yukarı)	
AL	1	CH→row1
		CL→col1
BH	7	DH→row2
		DL→col2

Mademki scroll_up ve scroll_down fonksiyonları tüm ekran için kullanılır, o halde yazım işlemini kolaylaştırmak için iki makro tasarlanabilir.

```
#define scrup()      scroll_up(0, 0, 24, 79)
#define scrdown()    scroll_down(0, 0, 24, 79)
```

Ekranın Silinmesi

Int 10H F:6

AH 6
AL 0
BH 7
CH row1
CL col1
DH row2
DL col2

Bu fonksiyon belirli bir dikdörtgensel bölgenin içini silmekte kullanılır. Ekran kütüphanesinde clear isimli makro clearscr fonksiyonunu çağırarak tüm ekranı silmektedir.

```
#define clear()          clearscr(0, 0, 24, 79)
```

Cursor'un Yerinin Tespiti

Cursor'un bulunduğu satır ve sütunun numarasını almak için 10H kesmesinin 3 numaralı fonksiyonu kullanılır. Geri dönüş değeri

DH satır no
DL sütun no

getcur Fonksiyonu

getcur fonksiyonu cursor'un konumunu tek parça int biçiminde alır. Öyleki yüksek anlamlı byte değeri satır, düşük anlamlı byte değeri sütun belirtir.

setcur Fonksiyonu

getcur ile alınan değer setcur ile konumlandırılabilir.

```
row = getrow();        cpos = getcur();
col = getcol();        ....
....                   .....
....                   ...
pos(row, col);         setcur(cpos);
```

Sayfa Kavramı

80x25 text mode'da bir ekranlık görüntü 80 * 25 * 2 = 4000 byte ile temsil edilir. 4000 bilgisayar için iyi bir sayı değildir. Ekran belleğindeki herbir 4096 byte bilgiye bir sayfa denir. Default sayfa sıfırdır. Ama bir kesme ile bir sayfa aktif hale geitirilebilir. Bir sayfanın aktif hale

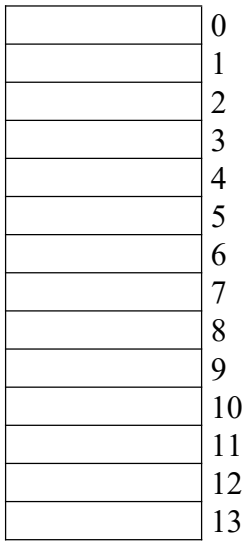
getirilmesi ile ekran belleğinin aktif bölgesi değiştirilmiş olur. Toplam sayfa sayısı ekran kartına ve özellikle de kart üzerindeki bellek miktarına bağlı olarak değişir. Birden fazla ekran sayfasının kullanılması yaygın bir özellik değildir. Donanımsal olarak sayfa değiştirmek için int 10H kesmesinin 5 numaralı fonksiyonu kullanılır.

AL→sayfa numarası

Her sayfanın ayrı bir cursor'u vardır. Aktif sayfayı donanımsal olarak değiştiren set_active_page fonksiyonu tanımlanmıştır. Sayfanın donanımsal olarak değiştirilmesi pagemno global değişkenini etkilemediği için birçok fonksiyon yanlış(başka bir sayfa için) çalışır. Bunun için change_active_page fonksiyonu tasarlanmıştır. Bu fonksiyon önce donanımsal değişiklik yapar, daha sonra pageno global değişkeninin değerini değiştirir. Yani biz kütüphane içerisinde change_active_page fonksiyonunu çağırmalıyız.

Cursur'un Boyunun Değiştirilmesi

Cursor 0-13'lük bir dikdörtgenin herhangi bir kesiti olabilir.



Cursor'un boyu iki değişkenle tespit edilir: min ve max. Normal cursor durumunda min = 12, max = 13, maximum cursor durumunda ise min = 0, max = 13'tür. Cursor'un boyunun deiştirilmesi int 10H kesmesibnin 1 nolu fonksiyonu ile yapılır.
AH→1
CH→min
CL→max

Kütüphanede set_cursor_size fonksiyonu bu işi yapar. İşlemleri kolaylaştırmak için iki makro yazılmıştır.

```
#define maxcur()      set_cursor_size(0, 13)
#define normcur()     set_cursor_size(12, 13)
```

Klavye İşlemleri

getkb() ve keypressed() fonksiyonları getch() ve kbhit() fonksiyonlarının karşılığı sayılabilir.

Ekran Belleği İle Doğrudan Çalışan Aşağı Seviyeli Fonksiyonların Tasarımı

Ekran belleğinde herbir hücreyi temsil eden bir yapı belirlemek işleri kolaylaştırır. Global bir değişkende ekran belleğinin başlangıç adresi tutulur.

```
typedef struct _CHR {
        char chr;
        char attr;
} CHR;

char far *vp = (char far *)0xb8000000;
```

initvideo() fonksiyonu

Mademki ekran belleğinin aktif bölgesi mono ve renkli modlar için farklıdır. O halde sistemin otomatik olarak çalışabilmesi için global vp değişkeninin set edilmesi gerekir. Bu fonksiyon önce o andaki video modu alır, eğer bu video mod 2 ve 7 numaralı modlardan bir tanesi ise ekran belleğinin başlanıç adresini 0xb0000000 adresine değilse 0xb8000000 adresine set eder. Bu kütüphaneyi kullanan her türlü kod başlangıçta initvideo fonksiyonunu çağırmalıdır.

Buradaki temel fonksiyonlar genellikle row ve col parametresi alırlar. Genellikle her fonksiyon global vp değerini sayfa numarası ile birleştirerek aktif bölgeyi yerel bir scrp göstericisine alır.

```
CHR far *scrp = (CHR far *)(vp + pageno * PGSIZE);
```

Fonsiyon isimlendirmesinde bazı isimler _ ile başlatılmıştır. Bu isimli fonksiyonlar renk bilgisini dikkate almazlar.

Temel Ekran Fonksiyonlarının İşlevleri

_writec	İstenen bir ekran bölgesine tek bir karakteri renk bilgisine karışmadan basar.
writec	Yukarıdaki fonksiyon gibidir anck renk bilgisini de kullanır.
_writes	Belirlenen ekran bölgesine renk bilgisini dikkate almadan bir yazıyı basar
writes	Yukarıdai fonksiyon gibidir ancak renk bilgisini de kullanır.
_fillc	Belirlenen ekran bölgesinden itibaren renk bilgisini dikkate almadan belirlenen bir karaketeri n defa basar.
fillc	Yukarıdaki fonksiyon gibidir ancak renk bilgisini de kullanır.
_vfillc	Ekranın belirlenen bir bölgesinden başlayarak belirlenen bir karakteri düşey olarak renk bilgisini dikkate almadan basar.
vfillc	Yukarıdaki fonksiyon gibidir ancak renk bilgisini de kullanır.
_writens	Bir yazıyı belirlenen bir bölgeden başlayarak n defa renk bilgisini dikkate almdan

	basar.
writens	Yukarıdaki fonksiyon gibidir ancak renk bilgisini de kullanır.
_frame	Bu fonksiyon rengi dikkate almadan sol üst ve sağ alt köşegenleriyle belirtilen çerçeveyi çizer
frame	Yukarıdaki fonksiyon gibidir ancak renk bilgisini de kullanır.
_framed	_frame gibidir ama çift çizgili çerçeve çizer.
framed	frame gibidir ama çift çizgili çerçeve çizer.

Çerçeve Çizimler

Çerçeve yalnızca dikdörtgensel bir çizim belirtir. Oysa pencere kavramı bölgesel bir kavramdır. Çerçeve çizmek için özel çerçeve karakterleri kullanılır. Bütün çerçeve karakterlerinin çift çizgili olanları da vardır. Çerçeve çizmek için writec, fillc ve vfillc fonksiyonları kullanılır. Önce köşe karakterleri writec fonksiyonlarıyla yerleştirirlir. Sonra fillc ile yatay çizgiler, vfillc ile dikey çizgiler yerleştirilir.

setwnd Fonksiyonu

wnd isimli gösterici Window türünden dinamik olarak büyütülen bir dizinin başlangıç adresini tutar.

chwnd Fonksiyonu

Bu fonksiyon bir pencere açıksa kapatır, kapalıysa açar. Eğer pencere açılıyorsa arka plan bilgi setwnd tarafından tahsis edilmiş olan bloğa yazılır, kapatılıyorsa bu bilgi ekrana geri basılır. Ancak pencere kapanırken dikket edilmesi gereken bir durum vardır. Pencere açıkken bir takım görüntüsel değişiklikler yapılmış olabilir. Arka plan bilgiyi ekrana basmadan önce değiştirilmiş olan pencere bilgisinin saklanması gerekir. Saklama işleminden sonra arka plan bilgi basılır. Artık yeni bilgi arka plan bilgi olacaktır ve ilgili değişiklikler yapılır. Fonksiyona setwnd'den alınan handle değeri parametre olarak geçirilir. Pencere açıksa kapatılır, kapalıysa açılır. Tabii bu pencere sisteminde son açılan pencere ilk kapatılmak zorundadır. Herhangi bir pencerenin kapatılacağı durumda daha değişik bir organizasyon yapılmalıdır.

freewnd Fonksiyonu

Bir pencere artık kullanılmayacaksa freewnd fonksiyonu ile serbest bırakılmalıdır. Çünkü bu fonksiyon arka plan bilgi için tahsis edilmiş alanı free fonksiyonu ile serbest bırakır.

clearwnd Fonksiyonu

Bu fonksiyon programı bitirirken sadece 1 kez çağırılmalıdır. Çünkü dinamik olarak büyütülen Window dizisini serbest bırakmakta kullanılır.

Pencere Sisteminin Geliştirilmesi Konusundan Tavsiyeler

- Bir pencere açıldıktan sonra pencerenin içerisine birşeyler yazmak için her defasında koordinat dönüşümü yapmak gerekir. Oysa bu işlem bir grup fonksiyon tarafından otomatik

olarak da yapılabilir. Örneğin ekrana bilgi yazan bütün fonksiyonların W versiyonları olur. Bu fonksiyon kullanıldığında hangi pencere aktifse onun sol üst köşegeni 0, 0 kabul edilir. Aktif pencerenin otomatik olarak bulunması yani örneğin bir pencere kapatıldıktan sonra hangi pencerenin aktif olacağının bilineceği stack algritması ile mümkün olabilir. Tabii bu işlemler manuel olarak da bir fonksiyon çağırılarak yapılabilir. Örneğin void set_active_window(int handle);

- Oluşturduğumuz pencere sisteminde son açılan pencere ilk kapatılmak zorundadır. Oysa değişik bir düzenlemeyle herhangi bir pencerenin kapatılması sağlanabilir.

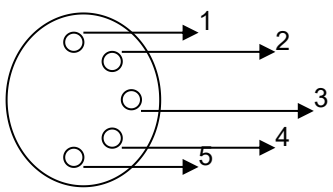
Klavye Kullanımı

1980 yılında piyasaya sürülen ilk PC'lerde 83 tuşlu klavye kullanılıyordu. Bu klavyeye PC klavyesi denilmektedir. Daha sonra bu klavye sistemine bir tuş daha eklenmiştir ve 84 tuşlu yapılmıştır. Ancak 1985 yıllarından sonra klavyede çok ciddi bir değişiklik yapılmıştır ve 101/102 tuş sistemine geçilmiştir. Yapılan değişiklik hem tuşların yerleriyle hem de sayılarıyla ilgilidir. Ok tuşları, home, end gibi özel tuşlardan birer tane daha eklenmiştir. Son yıllarda özellikle WINDOWS işletim sistemi için klavyeye birkaç tuş daha eklenmiştir. Ancak bu tuşlar WINDOWS dışında kullanılmamaktadır.

Klavye Devresi

Klavyenin mekanik kısmı birbirini kesen matrisel bir bağlantı sistemine oturtulmuştur. Bir tuşa basıldığında yatay ve düşey eksenlerdeki yollar kısa devre yapılır. Böylece elektriksel olarak hangi tuşa basıldığı anlaşılır. Bu matris biçimindeki yapının yatay ve dikey uçları ismine klavye işlemcisi denilen özel bir işlemciye bağlıdır. Bu işlemci Intel 8048 ya da türevi bir işlemci(Holtek HT6547D) olabilir. Klavye işlemcisi basılan tuşun kaç numaralı tuş olduğunu tespit eder ve bunu bit bit kodlayarak iletir. Keyboard işlemcisine göre her tuşun bir sıra nosu vardır. Bu sayıya klavye tarama kodu(keyboard scan code) denir. Basılan tuşun klavye tarama kodu klavye işlemcisinden bit bit okunarak alınabilir. Basılan tuşun bilgisi seri haberleşme yoluyla bit bit işlemcinin data ve clock uçları kullanılarak alınabilir. Clock ucu normalde logic 1 seviyesindedir. Bir tuşa basıldığında işlemci hangi tuşa basıldığını anlar ve clock ucunu 0'a çekerek basılan tuşun klavey tarama kodunu vermeye başlar. Bilgiyi alacak kişi clock ucuna bakarak data hattını örneklemelidir. Tuşa basıldığında işlemci yalnızca klavye tarama(KTK) kodu yollar. İşlemci tuştan elin çekildiğini de anlar. Bunu bildirmek için önce bir F0 sonra da tuşa ilişkin KTK değerini yollar. KTK bilgisi de 1 byte uzunluğunda bir bilgidir. Numlock capslock gibi tuşlara basıldığında işlemci normal olarak bu tuşların ışıklarını yakmaz. Bu ışıkların yaktırılması bu işlemcinin dışarıdan programlanmasıyla yapılır. Yani işlemci hem bilgi göndermekte hem de bilgi alarak programlanmaktadır.

Klavye bilgisayar bağlantısında 5 uçlu bir konnektör kullanılır. Bu uçlar:



1. Clock
2. Data
3. Reset
4. Ground
5. +5V

Klavyeden gönderilen bilgi bu konnektör yoluyla ismine klavye işlemcisi denilen bir işlemci tarafından elde edilir. Klavye işlemcisi olarak Intel 8042 gibi işlemciler kullanılmaktadır. Klavye denetleyicisi klavye tarama kodunu ismine sistem tarama kodu(system scan code) denen başka bir tarama koduna çevirir. Klavye denetleyicisi 8259 kesme denetleyicisine bağlıdır. Bir tuşa basıldığında ve çekildiğinde 1 numaralı IRQ'nun çağırılmasına yol açar. Bu IRQ sayesinde 9 numaralı kesme kodu çalıştırılmaktadır. Klavye denetleyicisinin iki tane register'ı vardır. Bu register'lar doğrudan 60H ve 64H portlarına bağlıdır. 60H portuna DATA portu denir. Eğer bu porta bağlı register okunursa son basılan ya da bırakılan tuşa ilişkin sistem tarama kodu elde edilir. Sistem tarama kodu klavye tarama kodundan farklıdır. Tuştan el çekildiğinde çekilme işlemini anlatmak için F0 byte'ı gönderilmez, sistem tarama kodunun en yüksek anlamlı biti 1'lenir. Bir tuşa basıldığında tuşa ilişkin sistem tarama kodu klavey denetleyicisi tarafından denetlkenir ve IRQ 1 çağırılır.

```
/*-----klavye.c-----*/
#include <stdio.h>
#include <dos.h>
#include <conio.h>

void main(void)
{
   char ch;

   for(;;) {
      printf("%02X\r", inp(0x60));
      if (kbhit()) {
         ch = getch();
         if (ch == 'q')
            break;
      }
   }
}
/*----------------------*/
```

IRQ 1 (9H kesmesi) Ne Yapar?

Kesme kodu 60H portunu okuyarak basılan tuşun bilgisini alır ve bunu ismine klavye tampon bölgesi denilen bir bölgeye dönüştürerek yazar. Kesme kodunun klavye tampon bölgesine yerleştirdiği tarama koduna genişletilmiş tarama kodu(extended scan code) denir. Genişletilmiş tarama kodu sistem tarama kodundan ctrl ve alt tuşlarının kullanımı açısından farklıdır. Ctlr ve alt tuşuna başka bir tuşla beraber basıldığında aslında bunların iki ayrı sistem

tarama kodu olduğu halde tek bir kod olarak birleştirilmektedir. Yani örneğin ctrl+c tuşlarına basıldığında sanki tek bir tuşa basılmış gibi tek bir genişletilmiş tarama kodu yazılır.

Tarama kodu çevirme özeti: Klavye içerisindeki klavye işlemcisi dışarıya klavye tarama kodunu verir. Klavye denetleyicisi bunu sistem tarama koduna çevirir ve 60H portundaki register'a yazar. 9H kesme kodu bunu genişletilmiş tarama koduna çevirir ve klavye tampon bölgesine yazar.

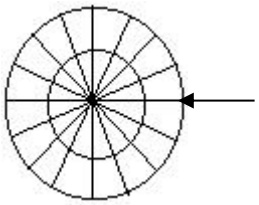
Programlama dillerindeki bütün klavyeden tuş alan fonksiyonlar tuş bilgisini nihai yer olan klave tampon bölgesinden almaktadır.

Klavye Tampon Bölgesi(keyboard buffer)

Klavye tampon bölgesi 0000:041E bölgesinde bulunmaktadır. Bu tampon bölge en fazla 15 tuşun bilgisini tutan döngüsel bir yapıdadır. Klavye bufferı'nın durumunu ismine head ve tail pointer denilen iki pointer gösterir. Bu iki pointer aynı değerdeyse tampon bölge boştur. Bir tuşa basıldığında 9H kesme kodu tuş bilgisini tampon bölgeye yerleştirir ve tail pointer'ı ilerletir. Şu anda tamponda 1 karakter vardır.

Head pointer

Tail pointer



Klavye tampon bölgesindeki herbir giriş 2 byte uzunluğundadır. Bu iki byte'ın düşük anlamlı byte'ı basılan tuşun ASCII kodunu yüksek anlamlı bye'ı genişletilmiş tarama kodunu tutar. Özel tuşların ASCII kodu 0'dır. Onlkar ancak genişletilmiş tarama koduyla teşhis edilebilirler. Klavye tampon bölgesinde bulunan bilgiler 16H kesmesinin fonksiyonlarıyla alınabilirler. Yani programcı klavye tampon bölgesinin yapısını bilmek zorunda değildir. Çünkü buradan bilgiyi 16H kesmesinin fonksiyonlarını çağırarak alabilir. Programlama dillerindeki klavyeye ilişkin tüm fonksiyonlar aslında 16H kesmesini çağırarak bu bilgiyi tampon bölgeden alırlar. Örneğin getch gibi bir fonksiyon 16H kesmesini çağırmaktadır. Ancak genişletilmiş tarama kodunu vermemektedir.

16H Kesmesinin Bazı Fonksiyonları

Tamponda bulunan tuşa ilişkin ASCII ve genişletilmiş tarama kodunun alınması

INT 16H
F:0 ve F:10H

Geri dönüş değeri:

AX
AH→ Tarama kodu
AL→ ASCII kodu

Bu kesmeyi kullanarak ASCII ve genişletilmiş tarama kodunu alan bir C fonksiyonu yazılabilir.

```
/*-----klavye2.c-----*/
#include <stdio.h>
#include <dos.h>

int getkb(void)
{
   union REGS regs;

   regs.h.ah = 0x0;
   int86(0x16, &regs, &regs);
   return regs.x.ax;
}

void main(void)
{
   int ch;

   while((ch = getkb()) != 0x011b)
      printf("%04x\n", ch);
}
/*----------------------*/
```

Tabii bu kesme eğer tampon bölge boşsa doluncaya kadar bekler. Yani 16H kesme kodu aşağıdaki gibi çalışır:

```
16H()
{
        while(tampon boş)
                ;
        return tampondaki_bilgiler;
}
```

Tamponun boş olduğunu varsayalım. 16H döngü içerisinde tampona bakarak bekler.bir tuşa basıldığında programın akışı donanımn kesmesiyle kesme koduna geçer. Kesme kodu tamponu doldurur. Kesme kodunun çıkışında döngüsel durum tampon dolduğu için sonlanır. Bütün özel tuşların ASCII ve scan kodları birlikte “scr.h” header dosyasında tanımlanmıştır. Sembolik sabit isimlendirilmesinde KP ile başlayan grup sağ taraftaki tuş grubudur. Örneğin KP8 yukarı ok tuşu, KP2 aşağı ok tuşudur.

```
/*-----klavye3.c-----*/
```

Klavyeden String Alan Gelişmiş Fonksiyonlar

Standart C'de kullandığımız gets() ve scanf() fonksiyonları ciddi projelerde kullanılmayacak kadar ilkeldir. Bu fonksiyonlar hiçbir özel tuş için özel davranış göstermezler. Örneğin insert ve delete tuşlarına duyarlı değildirler. gets() fonklsiyonu yerine çok daha gelişmiş getstr() fonksiyonu yazılacaktır.

getsr() Fonksiyonundan Beklenilenler

- n karakter alma gibi bir sınır belirleme durumu olmalıdır.İstense bile daha fazla karakter alınamamalıdır.
- DEL, INS, HOME, END gibi tuşlara uygun davranışları göstermelidir.
- İstenilen bir tuşa basıldığında ver giriş alanından çıkılması gerekir.

getstr() Fonksiyonunun Parametreleri

getstr(void *str, int count, int (*exfunc)(int), const char *init);

- Fonksiyonun birinci parametresi edit alanı içerisindeki bilginin yerleştirileceği adresi belirtir.
- İkinci parametre en fazla kaç karakter okunacağını beliritir.
- Üçüncü parametre edit alanının terk etmek için kullanılacak tuşun belirleneceği fonksiyonu çağırır. Bu fonksiyon getstr() içerisinden basılan tuş parametresiyle çağırılır. Eğer fonksiyon 0 dışı bir değere geri dönerse edit alanı terk edilir, 0 değeriyle dönerse terk edilmez. Fonksiyon şöyle tasarlanabilir.

```
int exfunc(int ch)
{
   switch(ch) {
      case CR:
      case CTRL_C:
         return 1;
   }
   return 0;
}
```

- Dördüncü parametre edit alanına girildiğinde çıkacak yazıyı belirlemede kullanılır. Eğer bu parametre NULL ise hiçbir yazı çıkartılmaz.

Fonksiyon hangi tuş nedeniyle edit alanından çıklmışsa ona geri döner.

Fonksiyonun Tasarımı

Algoritmanın çekirdeği döngü içerisinde getkb() fonksiyonuyla bir tuş alınması, bu tuşun switch içerisinde hangi tuş olduğunun belirlenmesi ve uygun işlemleri yapılması işlemlerine dayanır. Edit penceresi cursor neredeyse orada açılır.

Fonksiyonda Kullanılan Yerel Değişkenler

row, col	O anda cursor'un bulunduğu hücrenin koordinat bilgilerini tutar.
col_init	Edit alanının başlangıç kolon bilgisini tutar.
nchr	O anda edit alanında girilmiş toplam karakterlerin sayısını belirtir.
n	Cursor'un bulunduğu sütun bilgisini edit alanının başından itibaren belirten bir orijinde tutar.
updatenchr()	İlk girişte edit alnında bulunan toplam karakterlerin sayısına geri döner.
getnscr()	Edit alanının içerisindeki bilginin elde edilmesi için kullanılır.

Bütün tuş kontrollerinde eğer gerekliyse n, col, nchr değişkenleri güncellenmelidir. Eğer basılan tuş özel bir tuş değilse switch değiminin default kısmına gelinirlir. Bu durmda basılan tuş edit alanının sonuna eklenir.

Kod Organizasyonu

getstr() fonksiyonu iki dosya halinde yazılmıştır(getstr.h ve getstr.c). Getstr.h içerisinde fonksiyon parametreleri ve sembolik sabitler vardır. Getstr.c içerisinde ise getstr() ve ilgili diğer fonksiyonların tanımlamaları bulunur. getstr() fonksiyonu yazılırken scr.c modülündeki fonksiyonlar kullanılmaktadır. Bunu sağlamak için bir project dosyası oluşturulur. Project dosyasının içerisine scr.c ve getstr.c dosyalrı konulur.

Kütüphane Oluşturma

Uzantısı .lib olan dosyalara kütüphane dosyaları denir. Kütüphane dosyaları obj modüllerden oluşur. Obj modüller derlenmiş fonksiyonlardan oluşur. Örneğin n tane fonksiyon kütüphane içerisine yerleştirilecekse:

1. n tane fonksiyon bir kaynak dosya içerisine yazılır. Bu kaynak dosya x.c olsun.
2. Bu dosya derlenerek obj modül elde edilir. X.obj oluştu.
3. Bu obj modül lib dosyanın içerisine yerleştirilir.

Bir kütüphaneye obj modül eklemek ya da çıkarmak için

TLIB.EXE(Borland)
LIB.EXE(Microsoft)
ARC(Unix tabanlı)

Kullanımı:

Tlib <.lib name> +/- <.obj name>

Örnek:

```
Tlib a.lib + b.obj        /*b.obj'u a.lib'e dahil eder*/
Tlib a – b        /*a.lib içerisinden b.obj dosyası çıkarılır*/
Tlib a,con        /*a.lib içerisindeki tüm obj modülleri ve fonksiyonları ekrana listeler*/
Tlib a,b          /*a.lib içerisindeki tüm obj modülleri ve fonksiyonları b.lst dosyasına yazar*/
```

İlgili lib dosyası yoksa yaratır. Lib dosyaları link aşamasında aranır. İstenilen bir .lib dosyasına linker'ın bakması için o lib dosyasının project dosyasına dahil edilmesi gerekir. Tabii derleme aşamasında kütüphanede bulunan fonksiyonları prototipleri için header dosyalarının include edilmesi gerekir.

String Fonksiyonlarının Geliştirilmesi

1. Scroll içeren bir version'u yazılabilir. Yani fonksiyondaki count edit alanının uzunluğunu belirtir, ancak count sayısından daha fazla karakter girişi yapılabilir.
2. Tarih alan tam sayı ve gerçek sayı alan özel getstr() fonksiyonları yazılabilir.
3. Ve nihayet bütün bu fonksiyonlar tek bir fonksiyon içerisine entegre edilebilirler.