# CHƯƠNG 4 : BỘ NHỚ VÀ HIỂN THỊ KÍ TỰ

## §1. KHÁI NIỆM CHUNG

Trong phần này ta sẽ xem xét việc xử lí hiển thị kí tự bằng cách xâm nhập trực tiếp vào bộ nhớ (direc memory access-DMA) . Ta sẽ tìm hiểu cách xâm nhập trực tiếp màn hình . Cách này nhanh hơn là dùng các hàm của C .

## §2. CÁC TOÁN TỬ BITWISE

**1. Toán tử bitwise and (&)** : C dùng 6 toán tử bitwise được tóm tắt trong bảng sau

| Phép toán | Kí hiệu |
|---|---|
| and | & |
| or | \| |
| xor | ^ |
| dịch phải | >> |
| dịch trái | << |
| đảo | ~ |

Các phép toán này có thể áp dụng cho dữ liệu kiểu int , char nhưng không áp dụng cho số float .

Toán tử & (khác với and logic &&) cần hai toán hạng có kiểu giống nhau . Các toán hạng này được and bit với bit . Toán tử & thường dùng kiểm tra xem một bit cụ thể nào đó có trị là bao nhiêu . Ví dụ để kiểm tra bit thứ 3 cuả biến ch có trị 1 hay 0 ta dùng phép toán :

ch &0x08;

**2. Toán tử or** : Toán tử or (khác or logic | |) thường dùng kết hợp các bit từ các biến khác nhau vào một biến duy nhất . Ví dụ ta có hai biến là ch1 và ch2 và giả sử các bit từ 0..3 của ch1 chứa các trị mong muốn còn các bit 4..7 của ch2 chứa các trị mong muốn . Khi viết :

a = ch1|ch2;

thì cả 8 bit của a đều chứa trị mong muốn .

**3. Toán tử dịch phải >>** : Toán tử này làm việc trên một biến duy nhất . Toán tử này dịch từng bit trong toán hạng sang phải . Sô bit dịch chuyển được chỉ định trong số đi theo sau toán tử . Việc dịch phải một bit đồng nghĩa với việc chia toán hạng cho 2 . Ví dụ : 0 1 1 1 0 0 1 0 dịch sang phải 1 bit sẽ là

0 0 1 1 1 0 0 1

**4. Đổi từ số hex sang số nhị phân** : Ta dùng các toán tử bitwise để đổi một số từ hệ hex sang hệ 2 . Chương trình như sau :

***Chương trình 4-1*** :

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

void main()
  {
    int i,num,bit;
    unsigned int mask;
    char string[10],ch;
    clrscr();
```

```
  do
   {
      mask=0x8000;
      printf("\nBan cho mot so : ");
      scanf("%x",&num);
      printf("Dang nhi phan cua so %x la : ",num);
      for (i=0;i<16;i++)
        {
         bit=(mask&num)? 1 : 0;
         printf("%d",bit);
         if (i==7)
           printf("  ");
         mask >>= 1;
        }
      printf("\nBan muon tinh tiep khong(c/k)?");
      ch=getch();
   }
  while (ch=='c');
  getch();
}
```

Trong chương trình trên ta dùng vòng lặp for để duyệt qua 16 bit của biến nguyên từ trái qua phải . Lõi của vấn đề là các phát biểu :

```
bit = (mask&num)? 1 : 0;
mask >>=1
```

Trong phát biểu đầu tiên mask là biến chỉ có một bit 1 duy nhất ở phía trái nhất . Mask này được & với num để xem bit trái nhất của num có là 1 hay là 0 . Nếu kết quả khác 0 (true) bit tương ứng của num là 1 còn ngược lại bit tương ứng là 0 . Sau mỗi lần & mask được dịch trái 1 bit để xác định bit tiếp theo của num là 0 hay 1 .

**5. Các toán tử bitwise khác** :

***a. Toán tử xor ^*** : Toán tử xor trả về trị 1 khi chỉ có 1 bit chứ không phải 2 bit có trị là 1

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Toán tử xor cần khi lật bit nghĩa là hoán chuyển giữa 1 và 0 vì 1 xor với 1 là 0 và 1 xor với 0 là 1 . Ví dụ để lật bit thứ 3 trong biến ch ta dùng :

```
ch ^ 0x08
```

***b. Toán tử dịch phải <<*** : Toán tử này tương tự toán tử dịch trái . Giá trị của bit chèn vào bên phải luôn luôn bằng 0 . Dịchphải tương ứng với việc nhân số đó cho 2 .

***c. Toán tử đảo*** : Toán tử này là toán tử một ngôi . Nó tác động lên các bit của toán hạng và đảo trị của bit từ 1 sang 0 và từ 0 sang 1 . Đảo 2 lần một số ta lại nhận được số cũ .

## §3. BỘ NHỚ MÀN HÌNH

**1. Khái niệm chung** : Màn hình thông thường có 25 dòng và mỗi dòng có 80 kí tự . Như vậy toàn bộ màn hình có 2000 kí tự . Mỗi kí tự trên màn hình tương ứng với một địa chỉ cụ thể trong bộ nhớ màn hình . Mỗi kí tự dùng 2 byte của bộ nhớ này : byte thứ nhất chứa mã ASCII của kí tự (từ 0 đến 255 nay từ 0 đến ff)và byte thứ 2 chứa thuộc tính của nó . Như vậy để hiển thị 2000 kí tự , bộ nhớ màn hình phải cần 4000 byte . Bộ nhớ màn hình đơn sắc bắt đầu tại B000h và kết thúc tại B0F9F . Nếu ta có màn hình màu thì vùng nhớ cho chế độ văn bản sẽ bắt đầu từ B8000h và kết thúc tại B8F9F . Khi muốn hiển thị kí tự trên màn hình , đoản trình thư vện C sẽ gọi đoản trình ROM-BIOS để đặt kí tự cần hiển thị vào địa chỉ tương ứng trong bộ nhớ nàm hình. Nếu muốn có tốc độ nhanh , hãy chèn trực tiếp các giá trị trên vào vùng nhớ màn hình .

**2. Con trỏ far** : Khi biết địa chỉ , cách thông dụng để đưa giá trị vào bộ nhớ là dùng con trỏ . Do vậy nếu ta cần đưa kí tự vào vị trí đầu tiên của vùng nhớ màn hình thì ta viết :

```
int *ptr ;
ptr = 0xB800;
*(ptr)=ch;
```

Đoạn chương trình trên có vẻ hợp lí nhưng lại không làm việc vì biến con trỏ thông thường có hai byte trong khi địa chỉ B0000h lại dài 5 chữ số (2,5 byte) . Lí do dẫn đến tình trạng này là do con trỏ thường dùng để chứa đại chỉ nằm trong một đoạn duy nhất mà thôi . Trong họ 8086 , một đoạn dài 10000h hay 65535 byte . Bên trong các đoạn địa chạy từ 0h đến FFFFh . Thông thường các dữ liệu của chương trình C nằm trong một đoạn nên để thâm nhập các địa chỉ nằm ngoài đoạn ta phải dùng một cơ chế khác . Bên trong 8086 , tình trạng này được khắc phục bằng cách dùng các thanh ghi gọi là thanh ghi đoạn . Các địa chỉ nằm ngoài đoạn đucợ tạo lập bằng tổ hợp địa chỉ đoạn và địa chỉ offset .

| | | | | |
|---|---|---|---|---|
| B | 0 | 0 | 0 | |
| | 0 | 7 | D | 0 |
| B | 0 | 7 | D | 0 |

Trong hình trên địa chỉ đoạn B000h được dịch trái 4 bit rồi cộng với địa chỉ offset 07D0 tạo ra địa chỉ tuyệt đối B07D0h.

**3. Dùng địa chỉ đoạn : offset trong C** : Như vậy khi địa chỉ nằm bên ngoài đoạn dữ liệu , C dùng tổ hợp đoạn : offset và yêu cầu dạng biểu diễn 32 bit(4 byte , 8 chữ số hex) với 4 chữ số cho địa chỉ đoạn và 4 chữ số cho địa chỉ offset . Do vậy C coi địa chỉ tuyệt đối B07D0 là 0xB00007D0 (B000 và theo sau là 07D0) . Trong C con trỏ 32 được tính bằng cách dịch địa chỉ đoạn sang trái 16 bit và cộng với địa chỉ offset . Do con trỏ thông thường không thể cất giữ địa chỉ dài 32 bit nên ta phải dùng con trỏ far Con trỏ này cất giữ địa chỉ dài 4 byte . Vì vậy chương trình sẽ là :

```
int far *ptr ;
ptr = 0xB8000000;
*(ptr)=ch;
```

**4. Dùng một kí tự để tô màn hình** : Chúng ta dùng con trỏ far để ghi lên màn hình 2000 bản sao của một kí tự . Điều này tương tự như dùng putch() . Chương trình kết thúc ghi gõ x

***Chương trình 4-2*** :

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

```
#include <stdio.h>
#define length         2000
void main()
  {
    int far *fptr;
    int add;
    char ch;
    clrscr();
    printf("Go vao mot ki tu , go lai de thay doi");
    fptr=(int far*)0xB8000000;
    while((ch=getche())!='x')
      for (add=0;add<length;add++)
          *(fptr+add)=ch|0x0700;
  }
```

Trong chương trình phát biểu :

```
*(fptr+add)=ch|0x0700;
```

dùng để điền đầy kí tự lên màn hình . Biến ch là kí tự muốn đặt vào bộ nhớ . Hằng số 0x0700 là byte thuộc tính , có nghĩa là không chớp nháy , không đậm , chữ trắng trên nền đen .

Phát biểu khác cần giải thích :

```
fptr=(int far*)0xB8000000;
```

Ta dùng dấu ngoặc vì hằng 0xB8000000 và biến int far fptr có kiểu khác nhau : hằng có vẻ là số nguyên dài còn fptr lại là con trỏ chỉ đến kiểu int . Do đó để tránh nhắc nhở của trình biên dịch ta cần biến đổi kiểu làm hằng trở tthành con trỏ far chỉ đến int. Màn hình có thể được xem như một mảng hai chiều gồm các hàng và cột . Địa chỉ tương ứng trong bộ nhớ được tính từ phép nhân số hiệu hàng với số lượng cột trong một hàng rồi cộng kết quả và số hiệu cột với địa chỉ bắt đầu của vùng nhớ màn hình . Ta có chương trình sau :

***Chương trình 4-3*** :

```
#include <conio.h>
#include <stdio.h>
#define rowmax 25
#define colmax 80

void main()
  {
    int far *fptr;
    int row,col;
    char ch;
    clrscr();
    printf("Go vao mot ki tu , go lai de thay doi");
    fptr=(int far*)0xB8000000;
    while((ch=getche())!='x')
      for (row=0;row<rowmax;row++)
          for (col=0;col<colmax;col++)
            *(fptr+row*colmax+col)=ch|0x0700;
  }
```

**5.Trình xử lí văn bản theo dòng**: Để biết thên về sự tiện lợi của con trỏ far ta xét thêm một trình xử lí văn bản theo dòng . Trình xử lí này chỉ làm việc trên một dòng văn bản . Ta sẽ tiến hành theo 2 bước : đầu tiên là một chương trình cho phép người dùng gõ vào một dòng

và di chuyển con nháy tới lui . Có thể xoá kí tự nhờ di chuyển con nháy tới đó và ghi đè lên nó . Chương trình như sau :

***Chương trình 4-4*** :

```
#include <conio.h>
#include <dos.h>
#define colmax 80
#define rarrow 77
#define larrow 75
#define video 0x10
#define ctrlc '\x03'
int col=0;
int far *fptr;
union REGS reg;

void main()
  {
    char ch;
    void clear(void);
    void cursor(void);
    void insert(char);
    fptr=(int far*)0xB8000000;
    clear();
    cursor();
    while((ch=getch())!=ctrlc)
      {
          if (ch==0)
            {
              ch=getch();
              switch (ch)
                {
                    case rarrow : if (col<colmax)
                                  ++col;
                                  break;
                    case larrow : if (col>0)
                                  --col;
                                  break;
                }
            }
          else
            if (col<colmax)
              insert(ch);
          cursor();
      }
  }

void cursor()
  {
    reg.h.ah=2;
    reg.h.dl=col;
    reg.h.dh=0;
```

```
  reg.h.bh=0;
  int86(video,&reg,&reg);
 }

void insert(char ch)
 {
  *(fptr+col)=ch|0x0700;
  ++col;
 }

void clear()
 {
  int j;
  for (j=0;j<2000;j++)
    *(fptr+j)=0x0700;
 }
```

Để xoá màn hình ta điền số 0 vào vùng nhớ màn hình với thuộc tính 07 . Sau đó con nháy được di chuyển về đầu màn hình nhờ phục vụ ấn định vị trí con nháy như sau :

ngắt 10h
ah=0;
dh=số hiệu dòng
dl= số hiệu cột
bh=số hiệu trang , thường là 0

Phát biểu switch dùng để đoán nhận các phím được nhận là phím thường hay phím chức năng . Phím mũi tên dùng tăng giảm col và gọi hàm cursor() để di chuyển con nháy tới đó . Nếu kí tự gõ vào là kí tự thường , nó được chèn vào nhờ hàm insert() .

**6. Byte thuộc tính** : Một kí tự trên màn hình được lưu giữ bởi 2 byte : một byte là mã của kí tự và byte kia là thuộc tính của nó . Byte thuộc tính được chia làm nhiều phần , bit nào bằng 1 thì thì thuộc tính tương ứng được bật . Bit thứ 3 điều khiển độ sáng còn bit thứ 7 điều khiển độ chớp nháy . Các bit còn lại là : 6 - thành phần đỏ của màu nền ; 5 - thành phần green của màu nền ; 4 - thành phần blue của màu nền ; 2 - thành phần đỏ của màu chữ ; 1 - thành phần green của màu chữ ; 0 - thành phần blue của màu chữ . Ta lập một chương trình để điền đầy màn hình bằng các kí tự chớp nháy .

***Chương trình 4-5*** :

```
#include <conio.h>
#include <stdio.h>
#define rowmax 25
#define colmax 80

void main()
 {
  int far *fptr;
  int row,col;
  char ch;
  clrscr();
  printf("Go vao mot ki tu , go lai de thay doi");
  fptr=(int far*)0xB8000000;
  while((ch=getche())!='x')
   for (row=0;row<rowmax;row++)
```

```
        for (col=0;col<colmax;col++)
          *(fptr+row*colmax+col)=ch|0x8700;
  }
```

Để bật chớp nháy ta để bit thứ 7 thành 1 , 3 bit màu nền 0 , 1 và 2 được đặt trị 1 nên nền sẽ là đen . Byte thuộc tính lúc này là 10000111 = 87h.

**7. Chương trình điền thuộc tính** : Để hiểu sâu hơn thuộc tính của kí tự ta xét chương trình sau

***Chương trình 4-6*** :

```
#include <conio.h>
#include <stdio.h>
#define rowmax 25
#define colmax 80
void main()
  {
    int far *fptr;
    char ch,attr=0x07;
    void fill(char,char);
    clrscr();
    printf("Go n cho chu binh thuong,\n");
    printf("Go b cho chu xanh nuoc bien,\n");
    printf("Go i cho chu sang,\n");
    printf("Go c cho chu chop nhay,\n");
    printf("Go r cho chu dao mau\n");
    while((ch=getche())!='x')
      {
        switch (ch)
          {
            case 'n':attr=0x07;
                     break;
            case 'b':attr=attr&0x88;
                     attr=attr|0x01;
                     break;
            case 'i':attr=attr^0x08;
                     break;
            case 'c':attr=attr^0x80;
                     break;
            case 'r':attr=attr&0x88;
                     attr=attr|0x70;
                     break;
          }
        fill(ch,attr);
      }
  }
void fill(char ch,char attr)
  {
    int far *fptr;
    int row,col;
    fptr=(int far*)0xB8000000;
    for (row=0;row<rowmax;row++)
```

```
    for (col=0;col<colmax;col++)
        *(fptr+row*colmax+col)=ch|attr<<8;
 }
```

Trong hàm fill() ta có lệnh

```
*(fptr+row*colmax+col)=ch|attr<<8;
```

vì attr là kí tự nên phải dịch trái 8 bit trước khi kết hợp với ch .

**8. Trở lại xử lí văn bản** : Bây giờ chúng ta đã biết thuộc tính của kí tự và ta sẽ mở rộng chương trình xử lí văn bản bằng cách thêm vào việc chèn và huỷ bỏ kí tự ,đổi màu .

***Chương trình 4-7*** :

```
#include <conio.h>
#include <dos.h>
#define colmax 80
#define rarrow 77
#define larrow 75
#define video 0x10
#define norm 0x07
#define blue 0x01
#define bkspc 8
#define altu 22
#define ctrlc '\x03'
int col=0;
int length=0;
int far *fptr;
union REGS reg;

void main()
 {
  char ch,attr=norm;
  void clear(void);
  void cursor(void);
  void insert(char,char);
  void del(void);
  fptr=(int far*)0xB8000000;
  clear();
  cursor();
  while((ch=getch())!=ctrlc)
   {
     if (ch==0)
      {
       ch=getch();
       switch (ch)
        {
          case rarrow : if (col<length)
                          ++col;
                        break;
          case larrow : if (col>0)
                          --col;
                        break;
          case altu   : attr=(attr==norm)? blue:norm;
        }
```

```
          }
        else
          switch (ch)
            {
              case bkspc: if (length>0)
                            del();
                          break;
              default : if (length<colmax)
                          insert(ch,attr);
            }
          cursor();
      }
  }

void cursor()
  {
    reg.h.ah=2;
    reg.h.dl=col;
    reg.h.dh=0;
    reg.h.bh=0;
    int86(video,&reg,&reg);
  }

void insert(char ch,char attr)
  {
    int i;
    for (i=length;i>col;i--)
      *(fptr+i)=*(fptr+i-1);
    *(fptr+col)=ch|attr<<8;
    ++length;
    ++col;
  }

void del()
  {
    int i;
    for (i=col;i<=length;i++)
      *(fptr+i-1)=*(fptr+i);
    --length;
    --col;
  }

void clear()
  {
    int j;
    for (j=0;j<2000;j++)
      *(fptr+j)=0x0700;
  }
```

Khi gõ tổ hợp phím Alt+U sẽ lật biến attr qua lại giữa norm(thuộc tính 07) và blue (cho chữ màu xanh - thuộc tính 01) . Hàm insert(0 có vòng lặp for dùng để thâm nhập trực

tiếp bộ nhớ và con trỏ far để dịch các kí tự sang trái khi cần chèn . Tiến trìmh dịch phải bắt đầu từ cuối câu để tránh ghi đè lên .

## §4. CÁC KIỂU BỘ NHỚ TRONG C

**1. Địa chỉ đoạn và offset** : Trong C kiểu bộ nhớ là khái niệm để chỉ về lượng các phần bộ nhớ khác nhau mà chương trình có thể chiếm . C cho phép 6 kiểu bộ nhớ là tiny , small , compact , medium , large và huge . Kiểu bộ nhớ mặc định là small .

Bộ vi xử lí dùng các thanh ghi 16 bit để ghi địa chỉ . Thanh ghi 16 bit lưu được ffffh byte hay 65536 hay 64 Kb địa chỉ . Vùng nhớ có kích thước này gọi là đoạn . Để truy cập địa chỉ nằm ngoài đoạn , bộ vi xử lí phải dùng hai thanh ghi là thanh ghi đoạn và thanh ghi offset . Địa chỉ thực được tính bằng cách dịch địa chỉ của thanh ghi đoạn sang trái 4 bit rồi cộng với thanh ghi offset . Làm như vậy ta đánh địa chỉ được fffffh hay 1048576 = 1Mb .

**2. Hai loại chỉ thị của bộ vi xử lí** : Bộ vi xử lí dùng hai kĩ thuật khác nhau để tham chiếu dữ liệu trong bộ nhớ . Nếu vị trí cần tham chiếu nằm trong đoạn 64Kb và đoạn này đã được chỉ định trong thanh ghi đoạn thì bộ vi xử lí chỉ cần dùng một lệnh duy nhất để truy cập dữ liệu . Cách này tương ứng với việc dùng con trỏ near trong C và thực hiện rất nhanh . Trái lại nếu bộ vi xử lí cần tham chiếu ô nhớ nằm ngoài đoạn thì đầu tiên nó phải thay đổi địa chỉ đoạn và sau đoa là địa chỉ offset . Điều này tương ứng với việc dùng con trỏ far trong C và thực hiện khá chậm .

**3. Các kiểu Compact , small , medium và large** : Có 4 loại chỉ thị của bộ vi xử lí ứng với 4 kiểu bộ nhớ trong C

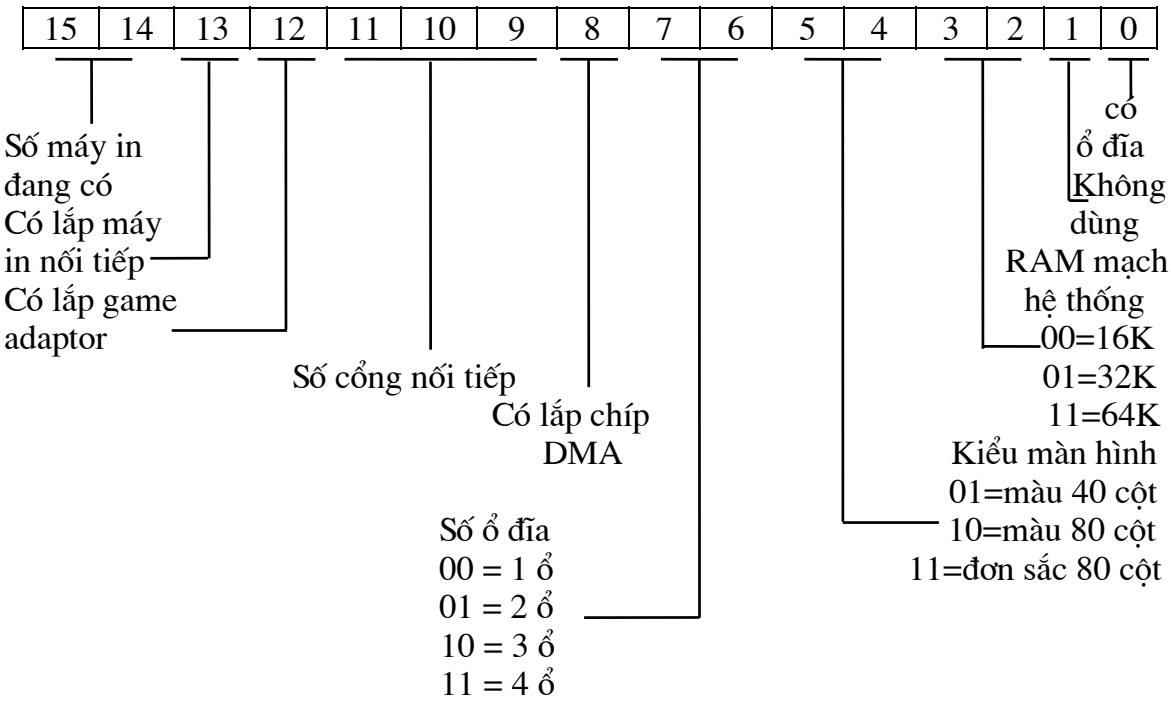
| Kiểu | Mã | Dữ liệu |
|---|---|---|
| small | near | near |
| medium | far | near |
| compact | near | far |
| large | far | far |

Nếu mã chương trình nằm gọn trong một đoạn 64 K và mã dữ liệu nằm gọn trong một đoạn 64 K khác thì kiểu bộ nhớ small là thích hợp . Nếu mã chương trình lớn hơn 64 K và mã dữ liệu nằm gọn trong một đoạn 64 K khác thì hãy dùng kiểu bộ nhớ medium. Nếu mã chương trình nhỏ hơn 64 K và mã dữ liệu lớn hơn 64 K thì hãy dùng kiểu bộ nhớ compact. Nếu cả mã chương trình và mã dữ liệu lớn hơn 64 K thì hãy dùng kiểu bộ nhớ large .

**4. Kiểu tyni và kiểu huge** : Kiểu tyni được dùng trong các trường hợp đặc biệt với lượng bộ nhớ cho cả mã chương trình lẫn mã dữ liệu nằm gọn trong một đoạn . Kiểu này được dùng để tạo ra tập tin dạng *.com . Kiểu huge được dùng chô một mục dữ liệu (thường là mảng ) mà bản thân nó lớn hơn 64K .

## §5. TỪ CHỨA DANH MỤC THIẾT BỊ

Đây là một vùng bộ nhớ dài 2 byte nằm trong vùng nhớ thấp có địa chỉ tuyệt đối là 410h chứa thông tin về thiết bị được nối với máy tính. Để truy cậo từ này ta dùng con trỏ far . Con trỏ sẽ chỉ tới đoạn 0000 , địa xhỉ offset là 0410h và được biểu diễn trong C là 00000410 hay 0x410




Để xem xét từng bit và nhóm bit trong từ này chúng ta sẽ dùng các toán tử bitweise . Nói chung ta sẽ dịch từ chứa danh mục thiết bị sang phải và đưa các bit cần quan tâm vào phía phải của từ và che các bit không quan tâm ở phái trái bằng toán tử and . Ngoài từ chứa danh mục thiết bị ta có thể đọc từ chứa kích thước bộ nhớ tại địa chỉ 413h .

***Chương trình 4-8*** :

```
#define eqlist 0x410
#define memsiz 0x413
#include <conio.h>
#include <stdio.h>
void main()
  {
    int far *fptr;
    unsigned int eq,data;
    clrscr();
    fptr=(int far *)eqlist;
    eq=*(fptr);
    data=eq>>14;
    printf("So may in la : %d\n",data);
    if (eq&0x2000)
      printf("Co may in noi tiep\n");
    data=(eq>>9)&7;
    printf("So cong noi tiep la :%d\n",data+1);
    if (eq&1)
      {
          data=(eq>>6)&3;
          printf("So dia mem la :%d\n",data);
      }
    else
```

```
  printf("Khong co dia mem\n");
 data=(eq>>4)&3;
 switch (data)
  {
      case 1: printf("Man hinh mau 40 cot\n");
              break;
      case 2: printf("Man hinh mau 80 cot\n");
              break;
      case 3: printf("Man hinh don sac 80 cot\n");
  }
 fptr=(int far *)memsiz;
 printf("Dung luong bo nho :%dKbyte\n",*(fptr));
 getch();
}
```