



CHƯƠNG 8

MỘT SỐ CHỦ ĐỀ



MỘT SỐ NGUYÊN TẮC

- Người ta đã viết sẵn cho chúng ta một số các chương trình, chúng ta chỉ việc sử dụng
- Phổ biến chính là các chương trình phục vụ ngắt
- Gọi thông qua các ngắt bằng lệnh : INT n
- Mỗi ngắt có nhiều chức năng khác nhau, mỗi chức năng được đánh số
- Khi thực hiện sẽ tác động lên các dữ liệu nào đó.
- Cho kết quả ở một nơi nào đó đặt ở nơi thích hợp.





- Chính vì vậy trước khi gọi ngắt, phải thực hiện các thao tác chuẩn bị, gồm :
 - Đưa chức năng muốn dùng vào thanh ghi AH
 - Đưa dữ liệu cần tác động vào nơi thích hợp
 - Chuẩn bị nơi chứa kết quả
 - Gọi ngắt : INT n



Một số ngắt thông dụng


- Số ngắt cũng theo qui ước của hệ thống như sau :
 - 00h ÷ 07h : ngắt hệ thống.
 - 08h ÷ 0Fh, 70h ÷ 77h : ngắt cứng.
 - Còn lại : ngắt mềm.
- Một số ngắt thông dụng :
 - INT 10h : màn hình.
 - INT 13h : đĩa.
 - INT 14h : thông tin liên lạc.
 - INT 16h : bàn phím.
 - INT 17h : máy in
 - INT 21h : các phục vụ của MS-DOS.
 - INT 20h : kết thúc chương trình, trở về DOS.

 CHỦ ĐỀ 1 : XUẤT NHẬP CƠ BẢN		
<ul style="list-style-type: none"> ■ Một số ngắt liên quan tới xuất nhập <ul style="list-style-type: none"> ■ Ngắt 21 		
Chức năng	Vào	Ra
1h : Nhập một ký tự từ bàn phím (có hiển thị lên màn hình).	AH = 01h	AL = Mã ASCII của ký tự vừa nhập
2h : Xuất ra màn hình một ký tự	AH = 02h DL = Mã ASCII ký tự	Không
8h : Nhập từ bàn phím một ký tự (không hiển thị ra màn hình)	AH = 08h	AL = Mã ASCII của ký tự vừa nhập.




Ngắt 21

<p>9h : Xuất một chuỗi ký tự ra màn hình</p>	<p>AH = 09h DS:DX= con trỏ tới chuỗi ký tự cần xuất Chú ý : chuỗi ký tự phải kết thúc bằng ký tự \$</p>	<p>Không</p>								
<p>0Ah : Nhập một chuỗi từ bàn phím (có thể chỉnh sửa khi nhập) Chú ý : trước khi gọi hàm này, khai báo vùng đệm (buffer) có dạng sau :</p> <table border="1"> <tr> <th>BYTE</th> <th>Ý nghĩa</th> </tr> <tr> <td>0</td> <td>số ký tự tối đa của chuỗi</td> </tr> <tr> <td>1</td> <td>số ký tự thật sự đã đọc vào, không kể ENTER</td> </tr> <tr> <td>2, ...</td> <td>chứa các ký tự đã nhập vào, kể cả dấu enter</td> </tr> </table>	BYTE	Ý nghĩa	0	số ký tự tối đa của chuỗi	1	số ký tự thật sự đã đọc vào, không kể ENTER	2, ...	chứa các ký tự đã nhập vào, kể cả dấu enter	<p>AH = 0Ah DS:DX = địa chỉ của vùng đệm chứa chuỗi ký tự nhập Ví dụ : Buffer DB 81 DB 0 DB 81 DUP (0)</p>	<p>Không</p>
BYTE	Ý nghĩa									
0	số ký tự tối đa của chuỗi									
1	số ký tự thật sự đã đọc vào, không kể ENTER									
2, ...	chứa các ký tự đã nhập vào, kể cả dấu enter									



Ngắt 16h

Chức năng	Vào	Ra
00h : Đọc ký tự từ bàn phím (không hiển thị lên màn hình)	AH = 00h	AL = Mã ASCII của ký tự vừa nhập (nếu AL = 0 thì AH = mã ASCII mở rộng) AH = Mã quét (scan code) hay mã ASCII mở rộng của ký tự vừa nhập
01h : Đọc ký tự từ bàn phím (không hiển thị lên màn hình), dành cho bàn phím 101 phím	AH = 01h	AL = mã ASCII của ký tự vừa nhập (nếu AL = 0 thì AH = mã ASCII mở rộng) AH = mã quét (scan code) hay mã ASCII mở rộng của ký tự nhập




Ngắt 10h

Chức năng	Vào	Ra
0eh : Xuất ký tự ASCII tại vị trí con trỏ và dịch chuyển con trỏ tới vị trí tiếp theo	AH = 0Eh AL = Mã ASCII của ký tự BH = trang, BL = màu nổi	Không

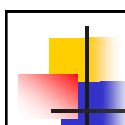
Chú ý :

- Muốn đổi mã ASCII của các số 0 – 9 sang chữ số, ta trừ mã ASCII cho 48 (30H)
- Khi nhập số dùng chức năng 01h của ngắt 21h, nhớ đổi sang số rồi mới tính
- Trước khi in số nhớ đổi từ số sang mã ASCII



Ví dụ 1

- Viết chương trình hiện chuỗi ký tự 'Hello TASM !' ra màn hình.
- **Cách 1 :** Dùng chức năng 9h của ngắt 21h.
- .MODEL small
- .STACK 100h
- .DATA
 - Message DB 'Hello TASM ! \$' ;khai báo biến chuỗi ký tự
- .CODE
- Begin :
 - MOV AX, @DATA
 - MOV DS,AX ; đưa đ/chỉ offset của đoạn dữ liệu vào DS
 - MOV DX, OFFSET Message ; DX chứa đ/chỉ offset của biến chuỗi ký tự
 - MOV AH, 09H ; chức năng 09h
 - INT 21H ; của ngắt 21h hiển thị Message lên màn hình
 - MOV AH, 1H ; chức năng chờ nhấn 1 ký tự
 - INT 21H
 - MOV AH, 4CH ; trở về DOS
 - INT 21H
 - END Begin



Cách 2 :

- Stackseg SEGMENT STACK 'STACK'
- DB 100 DUP(?)
- Stackseg ENDS
- DATA SEGMENT
- Message DB 'Hello TASM !',0
- DATA ENDS
- CODE SEGMENT
- ASSUME CS:CODE, DS:DATA, SS:Stackseg
- Begin :
 - MOV AX, DATA
 - MOV DS, AX
 - MOV SI, OFFSET Message ; DS:SI trở tới thành phần đầu tiên của chuỗi
 - CLD ; DF=0 : lấy ký tự theo chiều từ trái sang
- L1:
 - LODSB ; đưa ký tự được trỏ bởi DS:SI vào AL
 - AND AL,AL ; kiểm tra xem đã tới ký tự 0 chưa, cờ ZF = 1 chưa



- JZ Stop ; nếu đã tới ký tự 0 thì nhảy tới Stop
- MOV AH, 0Eh ; chức năng 0eh của ngắt 10h
- INT 10H
- JMP L1
- Stop :
- MOV AH, 1H
- INT 21H
- MOV AH, 4CH
- INT 21H
- CODE ENDS
- END Begin



Ví dụ 2 :

- Viết chương trình nhận vào từ bàn phím một chuỗi tối đa 80 ký tự và hiện ra màn hình theo dạng chữ nhỏ (lowercase). Đổi chữ lớn (hoa) ra chữ nhỏ (chữ thường) bằng cách set bit 5 trong mã ASCII tương ứng của ký tự (cộng thêm cho 32)
- _STACK SEGMENT STACK 'STACK'
- DB 100H DUP(?)
- _STACK ENDS
- DATA SEGMENT
- M1 DB 'Nhập Vào Chuỗi Ký Tự : \$'
- M2 DB 10,13,'Chuỗi Ký Tự Sau Khi Đổi La :\$'
- Buff DB 80
- DB 0
- DB 80 DUP(?)
- DATA ENDS
- CODE SEGMENT
- ASSUME CS:CODE, DS:DATA, SS:_STACK



```

■ Start:
■     MOV AX, DATA
■     MOV DS, AX
■     MOV AH, 09H           ; in ra chuỗi thương bao M1
■     MOV DX, OFFSET M1
■     INT 21H
■     MOV AH, 0AH           ; nhập chuỗi vào Buff
■     MOV DX, OFFSET Buff
■     INT 21H
■     MOV DX, OFFSET M2     ; in ra chuỗi M2
■     MOV AH, 09H
■     INT 21H
■     MOV BX, OFFSET Buff   ; BX trở tới đầu Buff
■     INC BX                ; BX trở tới ô nhớ chứa số ký tự thực sự được nhập
■     XOR CH,CH             ; CH = 0
■     MOV CL, [BX]          ; CL = Số ký tự nhập vào
■     INC BX                ; BX trở tới ô nhớ chứa ký tự đầu tiên của chuỗi

```



```

■ L1:
■     MOV DL, [BX]          ; đưa ký tự đầu tiên vào DL
■     CMP DL, ' '           ; nếu là khoảng trắng thì in ra bình thường
■     JE L2
■     CMP DL, 97             ; nếu là chữ hoa đổi ra chữ thường (DL > 'A')
■     JGE L2
■     ADD DL, 32
■ L2:
■     MOV AH, 02H
■     INT 21H
■     INC BX
■     LOOP L1
■ Stop:
■     MOV AH, 1
■     INT 21H
■     MOV AH, 4CH
■     INT 21H
■ CODE ENDS
■ END Start

```



CHỦ ĐỀ 2

MACRO VÀ THỦ TỤC



Macro

- Macro là 1 khối văn bản có tên, dùng để thay thế cho một đoạn chương trình có dạng giống nhau được lặp lại nhiều lần.
- Khi trình biên dịch gặp tên Macro, nó sẽ chèn khối văn bản tương ứng vào chương trình.
- Khi macro được định nghĩa, trong C.T nguồn thay vì viết một dãy các câu lệnh đã được định nghĩa bởi macro, ta chỉ cần viết tên của macro đó.
- Dùng Macro làm cho chương trình nguồn trở nên rõ ràng, gọn, dễ đọc nhưng không có lợi về bộ nhớ chương trình (nghĩa là chương trình có dùng macro hay không khi dịch ra sẽ có kích thước không đổi).



Định nghĩa macro :

<Tên> MACRO [<danh sách tham số>,...]

Các lệnh trong phần thân

ENDM

- Macro gồm 3 phần :

- Phần header : để khai báo macro

<Tên> MACRO [<danh sách tham số>,...]

- Tên : tên macro cần định nghĩa
 - Danh sách tham số : dãy các tham số cách nhau bởi dấu phẩy
 - Phần thân : tập các câu lệnh mà macro cần định nghĩa
 - Phần cuối : là chỉ dẫn ENDM, đánh dấu điểm kết thúc của macro



Phân biệt giữa macro và chương trình con :

- Chương trình con : các lệnh của chương trình con khi dịch chỉ xuất hiện một lần duy nhất trong chương trình, khi cần gọi chương trình này ta chỉ cần dùng lệnh CALL
- Đối với macro : khi dịch, sẽ thay thế mỗi tên của macro trong chương trình bằng các câu lệnh mà macro đó đã định nghĩa.



Ưu điểm và nhược điểm của macro

- Có thể thay đổi các tham số truyền cho macro dễ dàng
- Macro được thực hiện nhanh hơn chương trình con, vì CPU không cần thực hiện lệnh CALL và RET
- Các macro có thể lập thành một thư viện macro
- Khuyết điểm : mỗi lần macro được dùng, assembler thay tên macro bằng các câu lệnh tương ứng mà macro đã định nghĩa, làm cho chương trình mã máy dài hơn, tốn nhiều bộ nhớ hơn.



Gọi macro

- <tên macro> <thông số thực> , . .
 - Nếu số <thông số thực> nhiều hơn số <thông số hình thức> thì các thông số dư bị bỏ qua. Ngược lại thì thông số hình thức nhận được trị rỗng (NULL).


■ Ví dụ :

```


Doi      MACRO    bien1,bien2
          MOV AX,bien1
          XCHG     AX,bien2
          MOV bien1,AX

```

ENDM



- Nếu gọi : *Doi tri,tri_so* thì MASM thay Macro bằng các lệnh :
 MOV AX,tri
 XCHG AX,tri_so
 MOV tri,AX
- Thông số dùng trong macro không có thuộc tính nhất định mà tùy thuộc vào thông số thực khi gọi. Miễn sao khi thay macro và dịch không bị lỗi cú pháp là được.
- Ví dụ : có thể gọi Doi BX,tri_so
- Nhưng không thể gọi Doi BX,2



Các chỉ dẫn dùng trong macro

- **Chỉ dẫn LOCAL**
 LOCAL <tên> [, ...]
- Chỉ dùng trong MACRO và đi trước các chỉ thị khác.
- Dùng để khai báo các nhãn địa phương dùng trong macro để tránh tình trạng báo lỗi khi có sự lặp lại nhãn trong các lần gọi macro.
- **Ví dụ :** Ta có định nghĩa macro
 Wait MACRO count
 PUSH CX
 MOV CX, count
 Next :
 LOOP Next
 POP CX
 ENDM

Ví dụ dùng Macro Wait trong chương trình

- Macro Wait chỉ có thể được gọi tối đa một lần
- Giải pháp là dùng chỉ dẫn LOCAL : các nhãn được khai báo bởi LOCAL được đổi thành tên nhãn mới trong chương trình mỗi khi macro được gọi, tên nhãn mới có dạng : ??Number (Number có tầm từ 0000h – FFFFh)

■ Ví dụ :

```
Wait      MACRO      count
          LOCAL      Next
          PUSH CX
          MOV CX, count
```

Next :

```
          LOOP Next
          POP CX
```

ENDM

Giải thích :

Khi gọi macro : Wait 1000 lần đầu, assembler sẽ dịch macro Wait như sau :

```
PUSH CX
MOV CX, 1000
```

??0000 :

```
LOOP ??0000
POP CX
```

- Nhãn Next được thay bằng tên mới là ??0000


Khi gọi macro : Wait 3000 lần thứ 2, assembler sẽ dịch macro Wait như sau :

```
PUSH CX
MOV CX, 3000
```

??0001 :

```
LOOP ??0001
POP CX
```

- Nhãn Next được thay bằng nhãn mới ??0001




Ví dụ :

- Macro tính lũy thừa của một số

```

Power  MACRO      Factor, Exp      ; Exp=lũy thừa, factor=cơ số
    LOCAL cont, zero
    XOR    DX,DX
    MOV    CX, Exp
    MOV    AX, 01
    JCXZ   Zero
    MOV    BX, Factor
    Cont :
        MUL    BX
        LOOP   Cont
    Zero :
    ENDM
  
```

- Để tính 2^5 , ta gọi macro như sau : Power 2, 5 ; kết quả lưu trong cặp thanh ghi DX:AX




Thư viện Macro

- Các macro chương trình tham chiếu đến có thể chứa trong 1 file riêng, tạo ra một thư viện các macro.
- Chỉ dẫn INCLUDE <tên đầy đủ của file chứa các macro>
- **Ví dụ 1:** Viết các macro hiện một xâu ký tự kết thúc bằng \$ ra màn hình.

```

Hienstring      MACRO xau
    PUSH AX DX
    MOV DX, OFFSET xau
    MOV AH, 09h
    INT 21h
    POP DX AX
ENDM
  
```




Ví dụ 2 :


- Viết chương trình hiện một xâu ký tự kết thúc bằng \$ lên màn hình
- **Cách 1 :** Dùng macro khai báo trực tiếp trong chương trình.

```
HienString    MACRO XAU
                PUSH AX DX
                MOV DX, OFFSET XAU
                MOV AH,09H
                INT 21H
                POP DX AX
            ENDM


SEGSTACK      SEGMENT    STACK        'STACK'
                DB 100H DUP(?)
SEGSTACK      ENDS
```




- SEGDATA SEGMENT
- M1 DB 'Thong Bao Cho Moi Nguoi \$'
- M2 DB 0AH, 0DH, '\$'
- SEGDATA ENDS
- SEGCODE SEGMENT
- ASSUME CS:SEGCODE, DS:SEGDATA, SS:SEGSTACK
- START:
- MOV AX, SEGDATA
- MOV DS, AX
- HienString M1
- HienString M2
- MOV AH, 4CH
- INT 21H
- SEGCODE ENDS
- END START



- **Cách 2 :** Đưa các khai báo macro vào một file, giả sử macro HienString được chứa trong file có tên là Lib1.asm đặt trong ổ đĩa C
- INCLUDE C:\Lib1.asm
- SEGSTACK SEGMENT STACK 'STACK'
- DB 100H DUP(?)
- SEGSTACK ENDS
- SEGDATA SEGMENT
- M1 DB 'Thong Bao Cho Moi Ngui \$'
- M2 DB 0AH, 0DH, '\$'
- SEGDATA ENDS
- SEGCODE SEGMENT
- ASSUME CS:SEGCODE, DS:SEGDATA, SS:SEGSTACK




- START:
- MOV AX, SEGDATA
- MOV DS, AX
- HienString M1
- HienString M2
- MOV AH, 4CH
- INT 21H
- SEGCODE ENDS
- END START



Thủ Tục (Procedure)

- **Các lợi ích khi tổ chức chương trình dạng thủ tục**
 - Giảm số lượng mã, vì thủ tục có thể được gọi từ bất cứ nơi nào trong segment mã.
 - Giúp cho việc tổ chức chương trình được tốt hơn
 - Tạo dễ dàng cho việc chạy từng bước một chương trình do các lỗi kỹ thuật có thể được cách ly rõ ràng.
 - Dễ dàng bảo trì các chương trình do bởi việc sửa chữa các thủ tục được thực hiện không mấy khó khăn.
- **Khai báo thủ tục**


```
<Name> PROC [Near| Far]
.....
Thân thủ tục
RET
<Name > ENDP
```



Thư viện các thủ tục

- Nếu không có thuộc tính thì xem là NEAR.
- PROC giúp cho chương trình dễ đọc, dễ phân biệt các chương trình con.
- Thuộc tính của PROC ảnh hưởng lên cách dịch các lệnh CALL và lệnh RET
- Ta có thể đặt các khai báo thủ tục vào một file riêng, sau đó sử dụng chỉ dẫn INCLUDE để khai báo cho chương trình chính biết lấy thủ tục ở đâu.
- **Ví dụ :** Viết chương trình nhập hai số từ bàn phím, tính tổng và in kết quả ra màn hình.
- **Cách 1 : tạo thư viện các thủ tục riêng.**


```
INCLUDE C:\LT\LIB1.ASM
```

```

■ .MODEL SMALL
■ .STACK 100H
■ .DATA
■     M1 DB 'NHAP VAO SO THAP PHAN 1 :$'
■     M2 DB 0DH, 0AH, 'NHAP VAO SO THAP PHAN 2 :$'
■     M3 DB 0DH,0AH,"TONG 2 SO LA : $"
■     X1 DW 0
■     X2 DW 0
■     KQ DW 0
■     BUFF1  DB 6
■             DB 0
■             DB 5 DUP(?)
■     BUFF2  DB 6
■             DB 0
■             DB 5 DUP(?)
■
■ .CODE


```



```

■ BEGIN:
■     MOV AX,@DATA
■     MOV DS, AX
■     HIENSTRING M1
■     MOV AH,0AH
■     MOV DX,OFFSET BUFF1
■     INT 21H


```



```

HIENSTRING M2
■ MOV AH,0AH
■ MOV DX,OFFSET BUFF2
■ INT 21H
;DOI CHUOI TRONG BUFF1 VA BUFF2 RA SO
■ MOV BX,OFFSET BUFF1
■ CALL CHUOI_SO
■ MOV X1, AX
■ MOV BX,OFFSET BUFF2
■ CALL CHUOI_SO
■ MOV X2,AX


```



```

;TINH TONG HAI SO ROI IN RA
■ HIENSTRING M3
■ MOV AX,X1
■ ADD AX,X2
■ CALL SO_CHUOI
;TRO VE DOS
■ MOV AH,4CH
■ INT 21H
■ INCLUDE C:\TASM\LIBPROC.ASM
■ END BEGIN

```




Thư viện chứa thủ tục chuỗi_so và so_chuoi

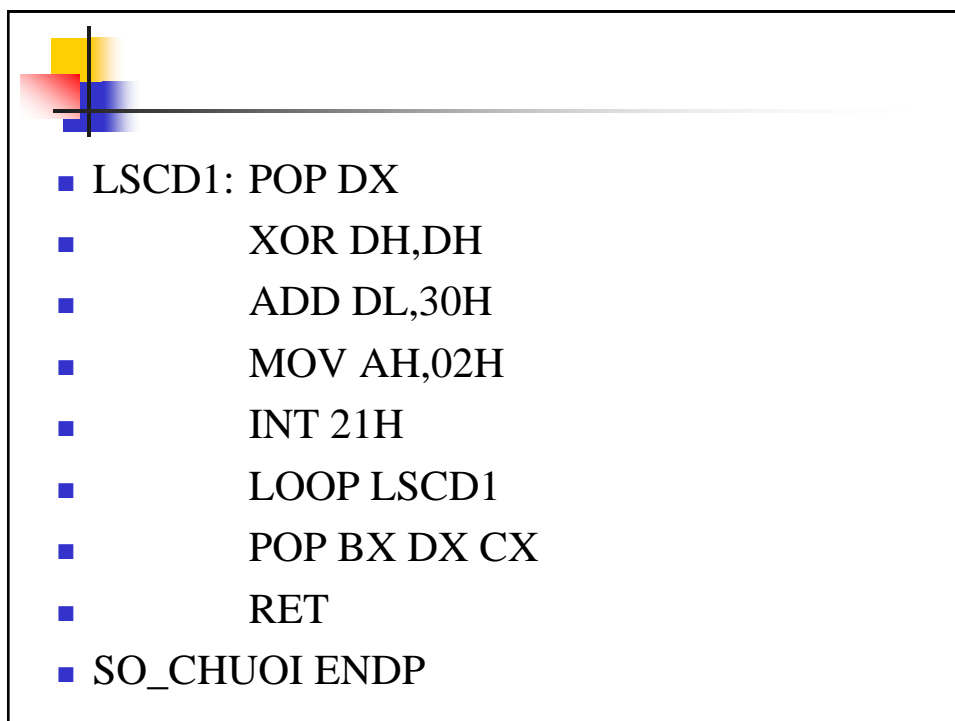
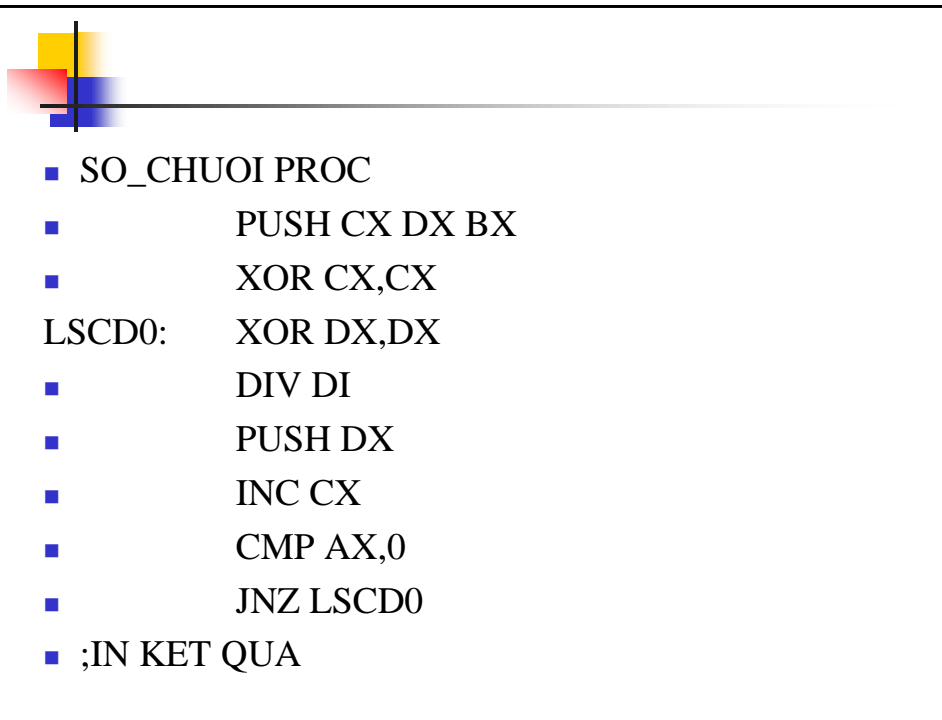
- CHUOI_SO PROC
- PUSH CX DX
- XOR CX,CX
- XOR AX,AX
- XOR DX,DX


- INC BX ; BX chỉ tới ô nhớ chứa số ký tự trong chuỗi
- MOV CL,[BX] ; CL chứa số ký tự có trong chuỗi
- INC BX ; BX chỉ tới ký tự đầu tiên

- MOV AL,[BX] ; đưa ký tự đầu vào AL
- SUB AL,30H ; đổi thành số
- DEC CX ; giảm CX
- MOV DI,10 ; DI=10
- CMP CX,0 ; so sánh CX với 0
- JZ L2



- L1:
- INC BX
- MUL DI
- MOV DL,[BX]
- SUB DL,30H
- ADD AX,DX
- LOOP L1
-
- L2:
- POP DX CX
- RET
- CHUOI_SO ENDP






Chủ đề 3 :

XỬ LÝ TẬP TIN




Một số chức năng ngắt 21h liên quan

Chức năng - Mô tả	Vào	Ra
3Ch : Mở tệp đã có hoặc tạo một tệp mới (nếu chưa có) và mở	AH = 3Ch DS:DX trỏ tới chuỗi tên tệp (chuỗi tên tệp kết thúc bằng 0) CL = chứa byte thuộc tính	Nếu thành công thì AX chứa thẻ tệp Nếu có lỗi (CF=1) thì AX chứa mã lỗi (3, 4, 5)
39h : Tạo thư mục	AH = 39h DS:DX = SEG:OFFSET = tên biến chuỗi chứa tên thư mục (chuỗi phải kết thúc bằng 0)	Nếu có lỗi tạo thư mục thì CF = 1
3Dh : Mở một file đã có	AH = 3Dh DS:DX = trỏ tới chuỗi tên tập tin kết thúc bằng byte 0 = mã truy cập (0 : mở để đọc, 1 : mở để ghi, 2 : mở để đọc và ghi)	Nếu thành công thì AX chứa thẻ tệp. Nếu có lỗi (CF=1) thì AX chứa mã lỗi (2, 4, 5, 12)



3Fh : Đọc tệp	AH = 3Fh BX = thẻ tệp CX = số lượng byte cần đọc DS:DX = trỏ tới vùng đệm chứa các ký tự đọc được	AX = chứa số byte đã đọc được Nếu AX = 0 hoặc AX < CX file đã kết thúc. nếu có lỗi (CF=1) thì AX chứa mã lỗi (5, 6)
3Eh : Đóng tệp đã mở	AH = 3Eh BX = thẻ tệp	Nếu có lỗi (CF=1) thì AX chứa mã lỗi (6)
40h : Ghi tệp	AH = 40h BX = thẻ tệp CX = số lượng byte cần ghi DS:DX trỏ tới vùng đệm chứa các ký tự sẽ ghi vào tệp	AX = chứa số byte đã ghi được. Nếu AX < CX lỗi đầy đĩa. Nếu có lỗi (CF=1) AX chứa mã lỗi (5, 6)



42h : Di chuyển con trỏ để truy cập trực tiếp ở vị trí mới	AH = 42h, BX = thẻ tệp CX:DX = độ dài cần chuyển = phương thức truy nhập Nếu : = 0 : di chuyển so với đầu tệp = 1 : di chuyển so với vị trí hiện tại của con trỏ tệp. = 2 : di chuyển so với cuối tệp.	Nếu thành công thì DX:AX = vị trí mới của con trỏ tệp. Nếu có lỗi (CF=1) thì AX chứa mã lỗi.
43h : Lấy/Thay đổi thuộc tính file	AH = 43h DS:DX = địa chỉ của chuỗi ASCII (chuỗi tên file kết thúc bằng 0) = 0 : để lấy thuộc tính file = 1 : để thay đổi thuộc tính file CX = thuộc tính file mới (nếu =1)	Nếu thành công thì CX = thuộc tính hiện thời (khi =0). Nếu CF=1 thì có lỗi, và AX chứa mã lỗi (2, 3, 5)



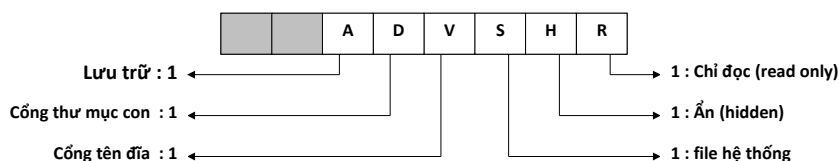
Lỗi thao tác tập tin

- Có nhiều khả năng gây ra lỗi với các hàm quản lý tệp của int 21h của DOS. Hệ điều hành xác định lỗi thông qua mã lỗi được đặt trong thanh ghi AX. Mỗi khi có lỗi, bit cờ CF = 1. Sau đây là 1 số mã lỗi thường gặp.

Mã lỗi (dạng Hex)	Ý nghĩa của lỗi
1	Số hàm không hợp lệ
2	Không tìm thấy tệp
3	Không tìm thấy đường dẫn
4	Mở quá nhiều tệp
5	Từ chối truy nhập (access denied)
6	Thẻ tệp không hợp lệ
C	Mã truy nhập không hợp lệ
F	Ổ đĩa không hợp lệ
10	Đang tìm cách xoá thư mục hiện thời
11	Không cùng thiết bị
12	Không tìm được thêm tệp nào



Byte thuộc tính





Ví dụ 1:


- Viết chương trình tạo lập thư mục với yêu cầu tên thư mục (có thể gồm cả ổ đĩa, đường dẫn và tên thư mục) được nhập vào từ bàn phím (nên chọn phương pháp cho phép sửa sai khi nhập vào nhằm tên thư mục).
- **Cách giải :**
 - Để thoả mãn yêu cầu nhận tên thư mục từ bàn phím và cho phép sửa sai khi vào nhầm, ta dùng chức năng 0Ah của int 21h. Sau khi nhận xong ký tự thì phải chuyển các ký tự đó (từ byte thứ 3 trở đi trong vùng đệm, ví dụ buff) đến vùng nhớ chứa tên thư mục (Dir_name) và phải thêm 0 vào cuối chuỗi.



```

■ INCLUDE C:\LIB1.ASM
■ _STACK      SEGMENT  STACK      'STACK'
■             DB 100 DUP(?)
■ _STACK      ENDS
■ DATA SEGMENT
■ M1          DB  'HAY VAO TEN THU MUC CAN TAO :$'
■ M2          DB  0AH,0DH,'CO LOI KHI TAO THU MUC $'
■ M3          DB  0AH,0DH,'THU MUC DA TAO DUOC $'
■ BUFF        DB 30
■             DB ?
■             DB 30 DUP(?)
■ DIR_NAME    DB  30 DUP(?)
■ DATA ENDS


```

```

■ CODE          SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:_STACK
START:
    ■ MOV AX,DATA
    ■ MOV DS,AX
    ■ CLRSCR
    ■ HienString M1
    ■ CALL GET_DIR_NAME
    ■ LEA DX,DIR_NAME      ; chức năng tạo thư mục
    ■ MOV AH,39H           ;
    ■ INT 21H
    ■ JNC L1
    ■ HienString M2        ; có lỗi khi tạo thư mục
    ■ JMP  STOP


```



```

L1:      HienString M3
STOP:    MOV AH,1
    ■    INT 21H
    ■    MOV AH,4CH
    ■    INT 21H
    ■    ;-----
    ■    ; Thủ tục nhận tên thư mục từ bàn phím
    ■    GET_DIR_NAME  PROC
    ■        PUSH AX DX SI DI
    ■        LEA DX, BUFF      ; DS:DX = trỏ tới vùng đệm
    ■        nhận các ký tự
    ■        MOV AH, 0AH      ; nhận xâu ký tự để vào Buff
    ■        INT 21H          ; và cho phép sửa khi vào


```



- MOV SI, OFFSET BUFF ; SI trỏ tới byte đầu của Buff
- LEA DIR_NAME ; DI trỏ tới byte đầu Dir_name
- INC SI

GDN1:

- INC SI
- MOV AL, [SI] ; đưa ký tự từ Buff vào AL
- CMP AL, 0DH ; có phải là phím enter không ?
- JE GDN2 ; nếu phải thì thì nhảy
- MOV [DI], AL ; nếu không thì đưa ký tự đó vào Dir_name
- INC DI ; trỏ tới byte kế tiếp của Dir_name
- JMP GDN1



GDN2:

- XOR AL, AL ; thêm 0 vào cuối xâu ký tự chưa tên thư mục
- MOV [DI], AL ;
- POP DI SI DX AX
- RET
- GET_DIR_NAME ENDP
- CODE ENDS
- END START