

LẬP TRÌNH VỚI HỢP NGỮ

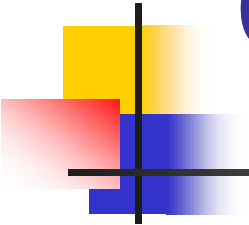
Gv: Lê Minh Triết

Phần 1:

Quy trình tạo và chạy chương trình

- Bộ hợp dịch ASM có hai trình cơ bản là
 - TASM.EXE (trình hợp dịch)
 - TLINK.EXE (trình liên kết)
- Ngoài ra ta còn cần một chương trình dùng để soạn thảo để tạo chương trình nguồn.

! Ta có thể dùng bộ chương trình BorlandC để soạn thảo chương trình nguồn.



Các bước tiến hành lập trình

Soạn thảo chương trình nguồn

Dùng trình hợp dịch TASM.EXE

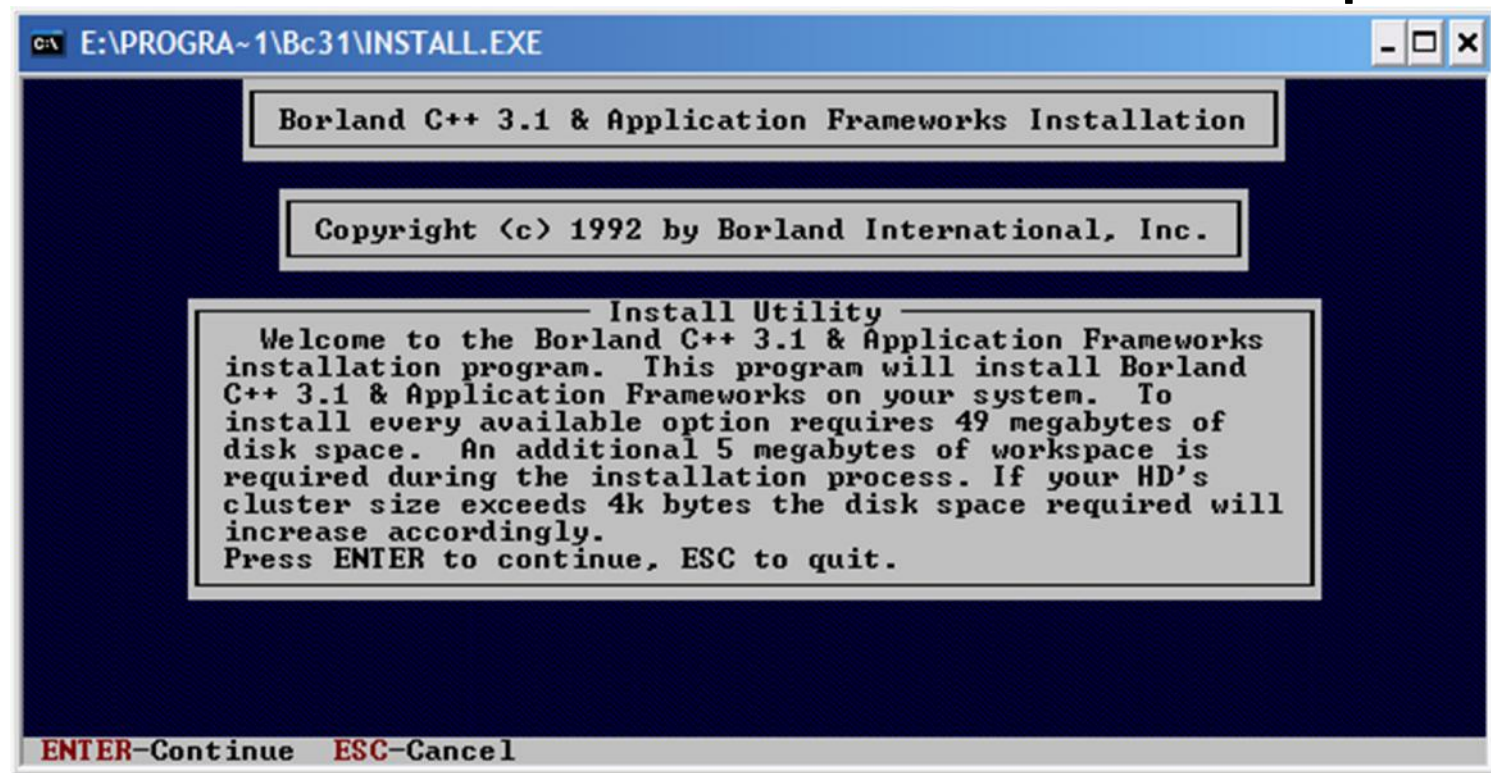
Dùng trình liên kết TLINK.EXE

Thực thi chương trình

Kết quả

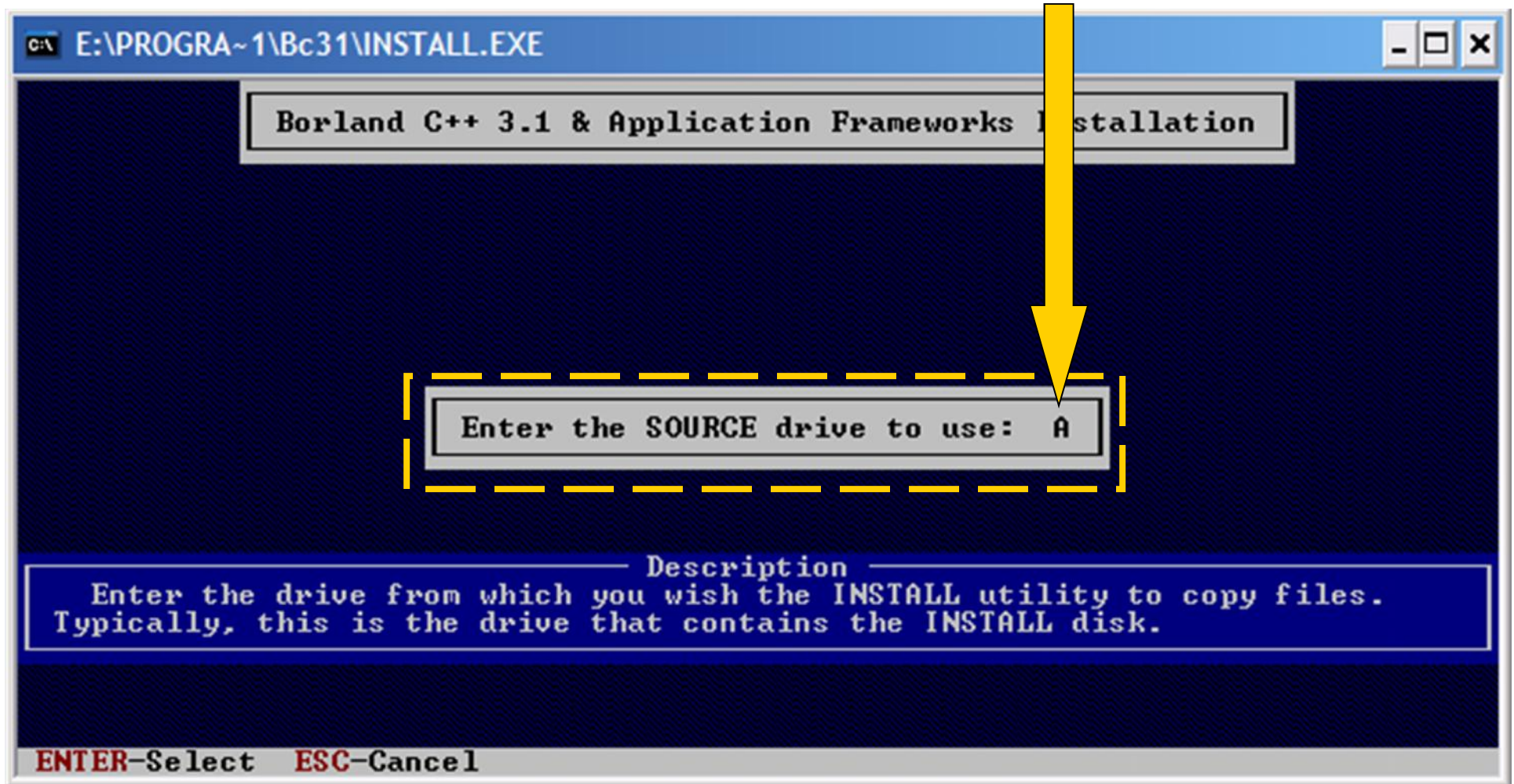
Các bước cài đặt và tạo đường dẫn File biên dịch

1. Chạy file [Install.exe](#) trong thư mục BorlandC (BC)
2. Nhấn nút **Enter** để bắt đầu cài đặt



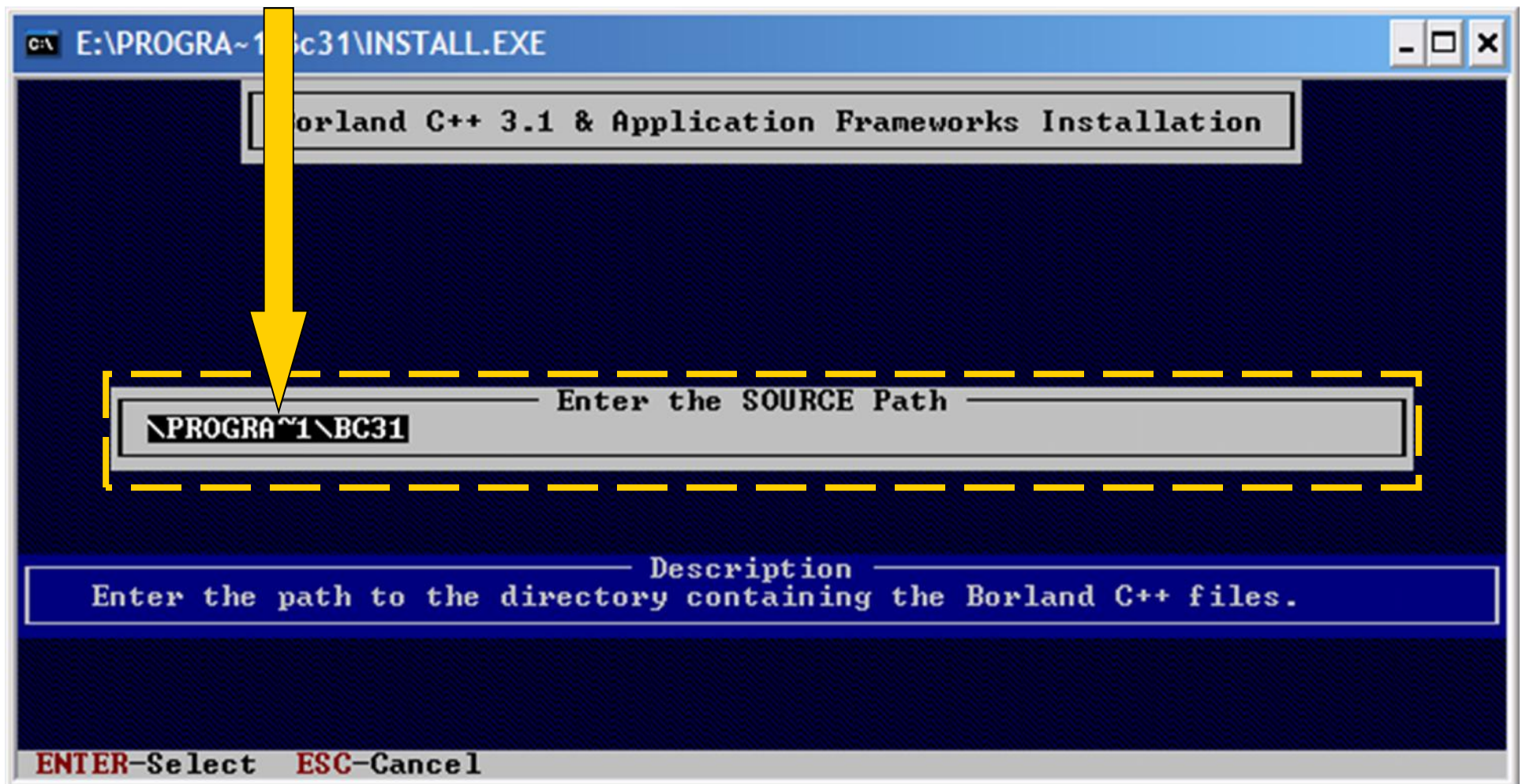
Các bước cài đặt và tạo đường dẫn File biên dịch

3. Chọn lại ổ đĩa chứa các tập tin cài đặt



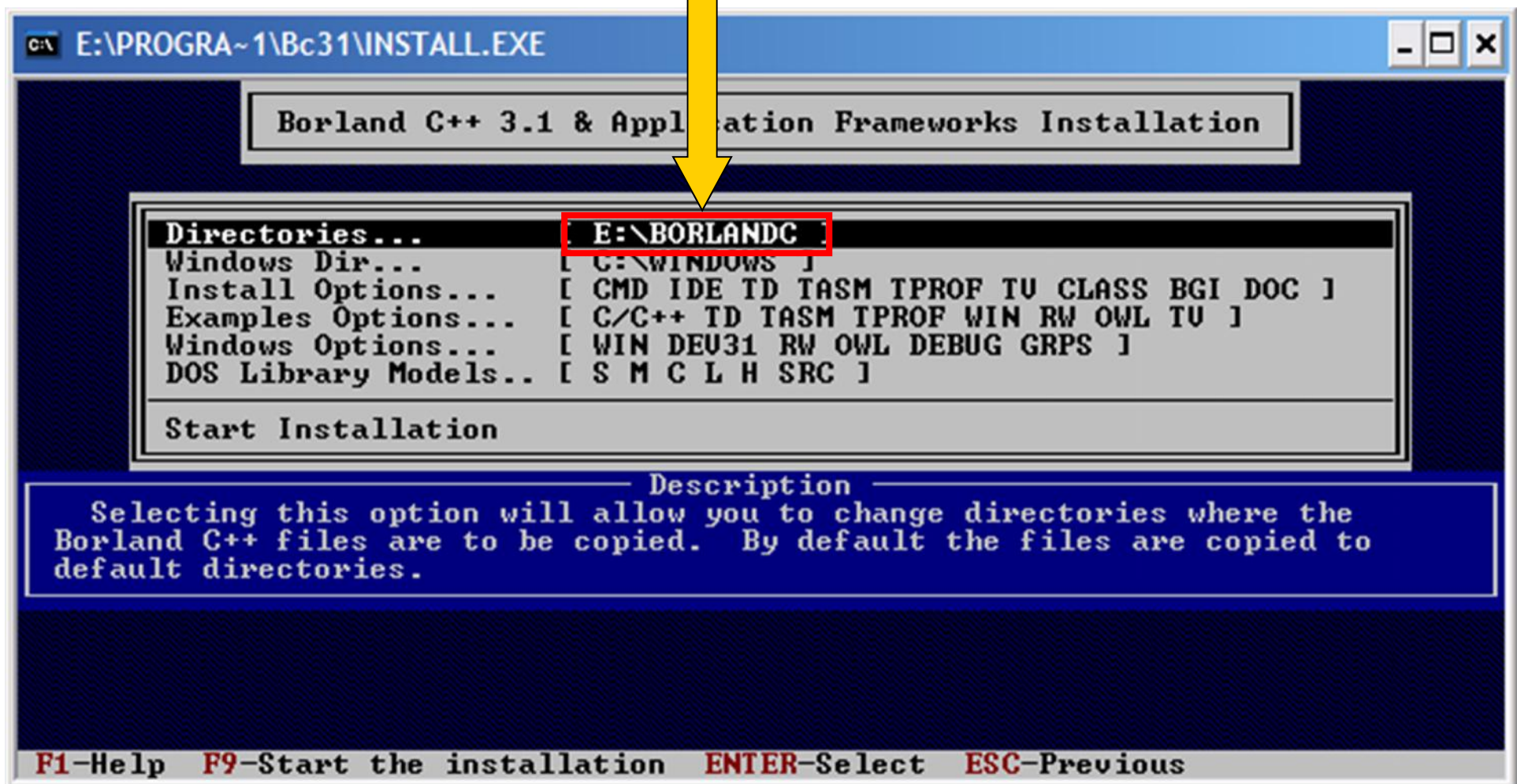
Các bước cài đặt và tạo đường dẫn File biên dịch

Kiểm tra đường dẫn chứa các tập tin cài đặt



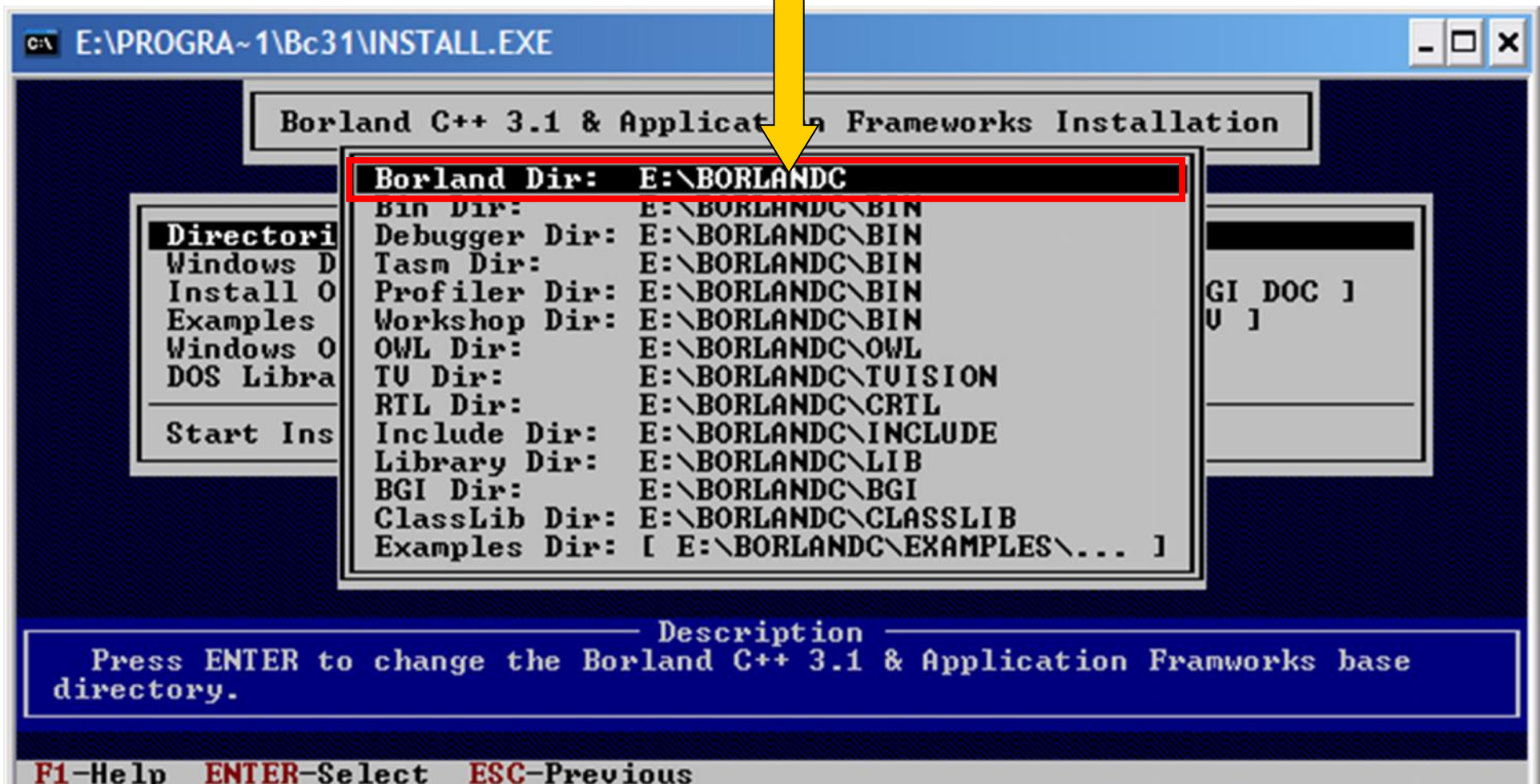
Các bước cài đặt và tạo đường dẫn File biên dịch

4. Sửa lại thư mục cài đặt (nhấn **Enter**)



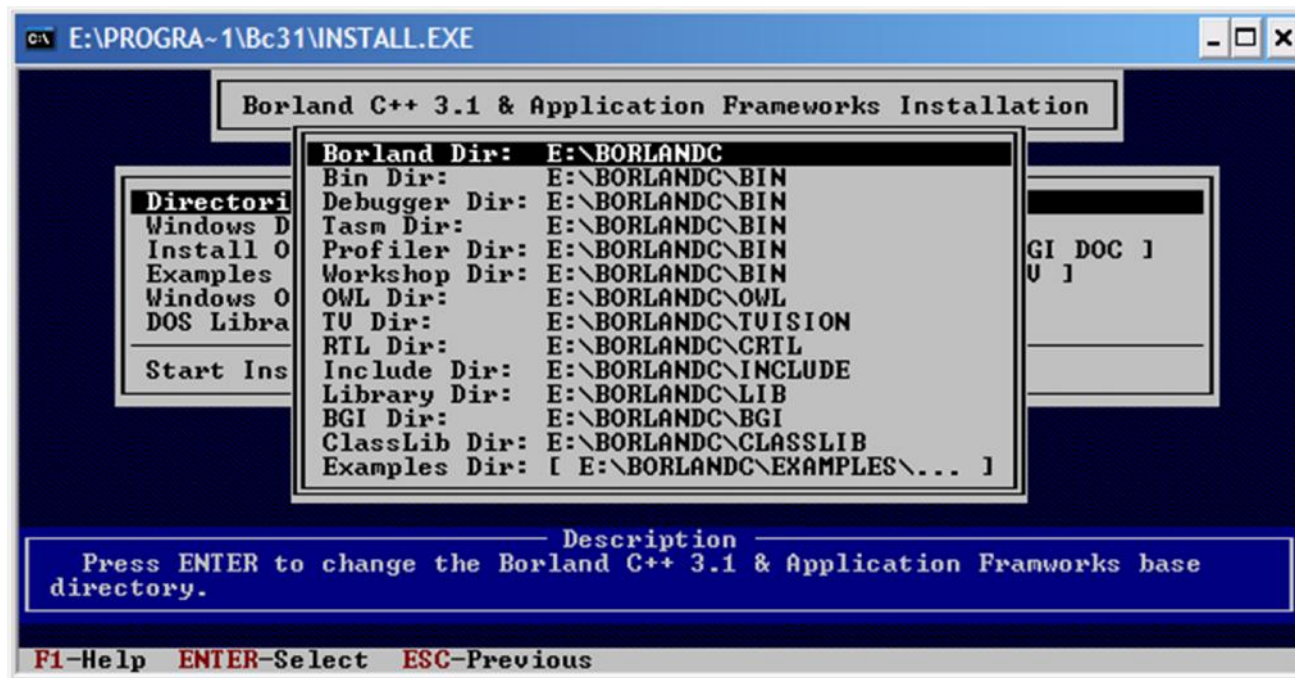
Các bước cài đặt và tạo đường dẫn File biên dịch

E:\BorlandC sửa lại là C:\BC hay C:\BorlandC



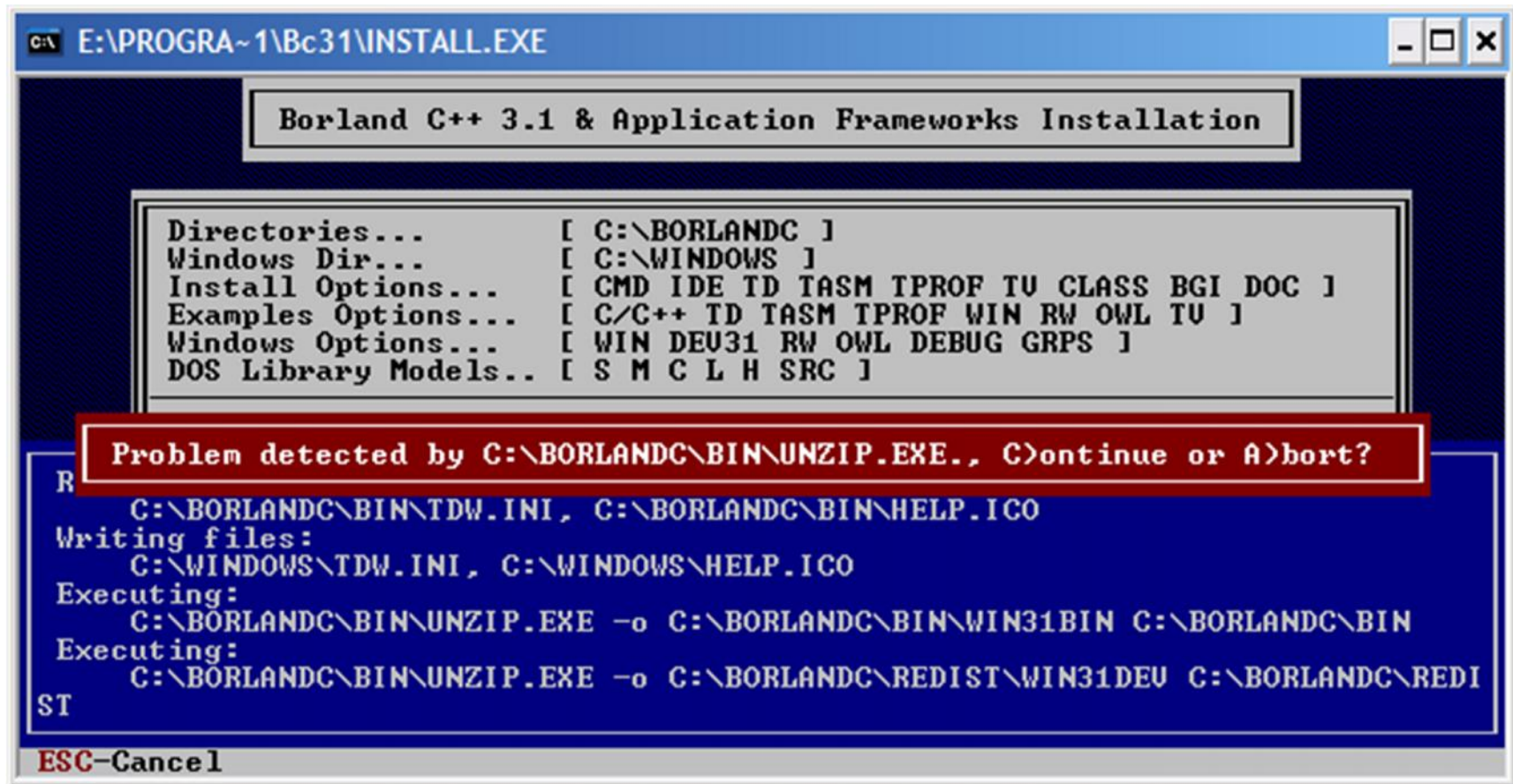
Các bước cài đặt và tạo đường dẫn File biên dịch

- Sau khi sửa xong, nhấn ESC để thoát khỏi màn hình chỉnh sửa đường dẫn.
- Chọn Start Installation để bắt đầu cài đặt.



Các bước cài đặt và tạo đường dẫn File biên dịch

- Khi cài đặt có 1 vài file bị lỗi, ta nhấn nút **C** để tiếp tục



```
E:\PROGRA~1\Bc31\INSTALL.EXE
Borland C++ 3.1 & Application Frameworks Installation

Directories... [ C:\BORLANDC ]
Windows Dir... [ C:\WINDOWS ]
Install Options... [ CMD IDE TD TASM TPROF TU CLASS BGI DOC ]
Examples Options... [ C/C++ TD TASM TPROF WIN RW OWL TU ]
Windows Options... [ WIN DEU31 RW OWL DEBUG GRPS ]
DOS Library Models.. [ S M C L H SRC ]

Problem detected by C:\BORLANDC\BIN\UNZIP.EXE., C>ontinue or A>bort?
R
C:\BORLANDC\BIN\TDW.INI, C:\BORLANDC\BIN\HELP.ICO
Writing files:
C:\WINDOWS\TDW.INI, C:\WINDOWS\HELP.ICO
Executing:
C:\BORLANDC\BIN\UNZIP.EXE -o C:\BORLANDC\BIN\WIN31BIN C:\BORLANDC\BIN
Executing:
C:\BORLANDC\BIN\UNZIP.EXE -o C:\BORLANDC\REDIST\WIN31DEU C:\BORLANDC\REDI
ST
ESC-Cancel
```

Các bước cài đặt và tạo đường dẫn File biên dịch

- Nhấn 1 nút bất kỳ để tiếp tục
 - ESC để tắt màn hình.
- Chú ý tạo đường dẫn (**PATH**)

```
E:\PROGRA~1\Bc31\INSTALL.EXE

Borland C++ 3.1 & Application Frameworks Installation

Direct Window Instal Exampl Window DOS Li
Start
Executing: C:\BORLA
Executing: C:\BORLA
Reading file C:\BORLA
C:\BORLA
Writing files: C:\WINDOWS\BC.ICO, C:\WINDOWS\BC.PIF, C:\WINDOWS\GROUPS.EXE

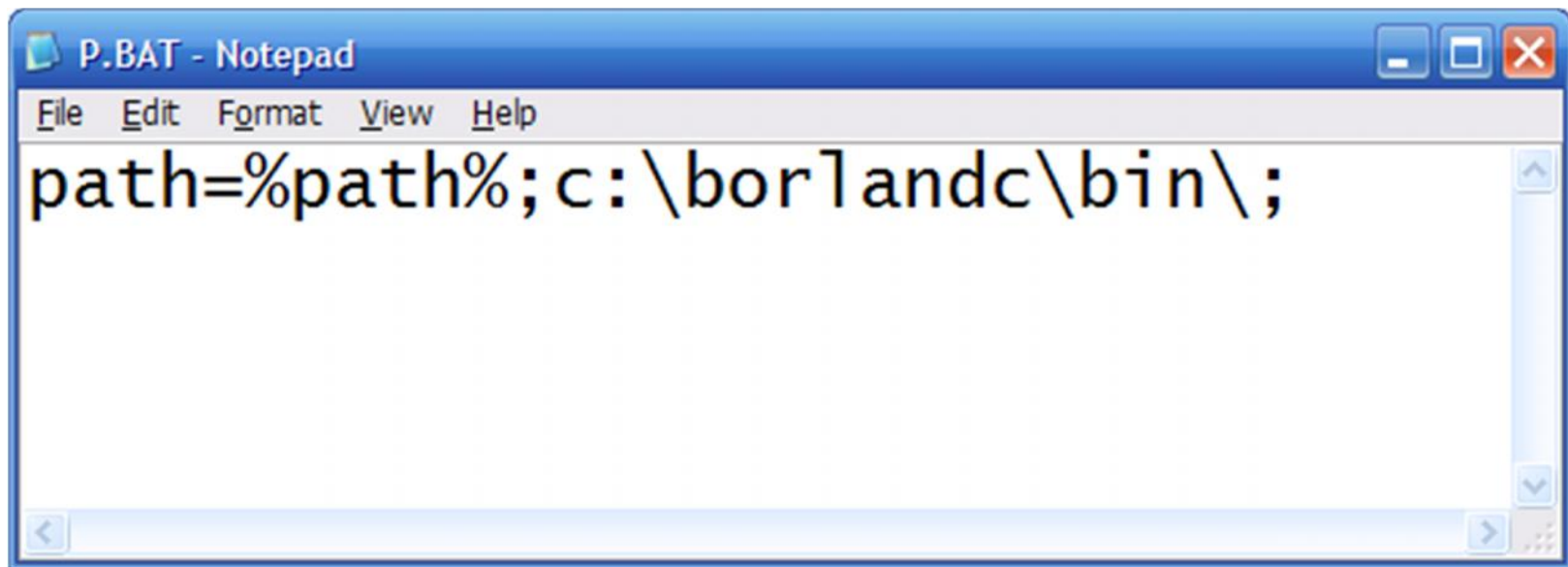
Borland C++ 3.1 & Application Frameworks is now
installed on your system. All the necessary
files have been copied to your hard drive. If
you selected 'YES' to install the command-line
version of the compiler, a configuration file
has been created. Make sure the line:
    FILES = 20
is in your CONFIG.SYS file and C:\BORLANDC\BIN
is in your path. For example:
    PATH=C:\BIN;C:\BORLANDC\BIN
Also during your next Windows session you will
be given the opportunity to create a Borland C++
group.
    Press any key to view the readme file.
```


Tạo file đường dẫn Path

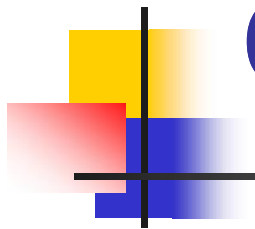


P.BAT

- Tạo 01 file *.bat trong ổ đĩa C:\
(hay ổ đĩa đã cài đặt BorlandC)



```
path=%path%;c:\borlandc\bin\;
```



Các bước tiến hành lập trình

1. Chạy cửa sổ Run
2. Gõ lệnh CMD
3. CD\
4. Chạy File P.bat
5. Chạy chương trình soạn thảo BC.EXE
6. Soạn nội dung chương trình nguồn
7. Lưu lại File với đuôi *.asm
8. Thoát khỏi BorlandC
9. Chạy chương trình hợp dịch TASM.EXE
10. Chạy trình liên kết TLINK.EXE
11. Thực thi chương trình.

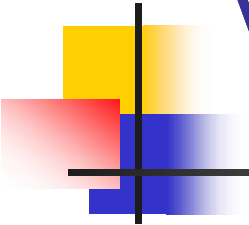
Ví dụ: Soạn chương trình Hello

■ Soạn tập tin Hello.asm

Khai báo dữ liệu bắt đầu bằng .Data

Khai báo đoạn mã chương trình bắt đầu bằng .Code

```
;hello.asm      ghi chú tập tin
dosseg          ; hướng dẫn thứ tự đoạn
.model small    ; chế độ mã và dữ liệu
.stack 100h     ; khai báo Stack 256 byte
.data           ; bắt đầu đoạn dữ liệu
message db "hello, world",0Dh,0Ah,'$' ;khai báo chuỗi
.code           ; bắt đầu đoạn mã chương trình
Begin:
Mov AX,@Data    ; đưa địa chỉ đoạn dữ liệu vào AX
Mov DS,AX       ; và cho DS trỏ tới đó
Mov AH,09h      ; hàm 09h in chuỗi với DX trỏ tới
Mov DX,OFFSET message
Int 21h
Mov AX,4C00h    ; kết thúc chương trình
Int 21h
End Begin
```

Ví dụ: Dịch chương trình Hello

- Gõ lệnh Tasm Hello.asm

```
C:\>tasm hello
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992

Assembling file:      hello.ASM
Error messages:       None
Warning messages:     None
Passes:               1
Remaining memory:     425k

C:\>
```



Thông báo
số lỗi

Ví dụ: Dịch chương trình Hello

- Gõ lệnh Tasm Hello.asm

```
C:\>tasm hello
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992

Assembling file:      hello.ASM
Error messages:      None
Warning messages:    None
Passes:               1
Remaining memory:    425k

C:\>
```

Thông báo
số lỗi

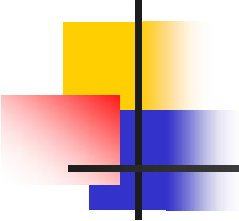
- Gõ tiếp lệnh Tlink Hello.obj
- Chương trình sẽ tạo ra file Hello.exe



Phần 2:

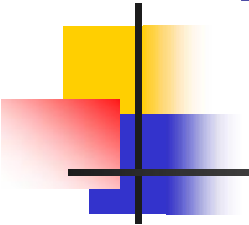
TỔ CHỨC THANH GHI

- Thanh ghi (**register**) là nơi lưu dữ liệu bên trong CPU
 - Tùy theo độ dài 8 hay 16 bit và tùy theo chức năng khi đó thanh ghi được dùng để chứa dữ liệu sẽ thao tác hoặc kết quả các phép tính hoặc các địa chỉ dùng để định vị ô nhớ khi cần thiết.
 - Có tất cả 14 thanh ghi, mỗi thanh ghi dài 16 bit chia thành năm nhóm



1. Nhóm thanh ghi đoạn (segment register)

- Gồm 4 thanh ghi: đoạn mã CS, đoạn dữ liệu DS, đoạn bổ sung ES và đoạn stack SS. Là những thanh ghi chứa địa chỉ segment của các ô nhớ khi cần truy xuất.
 - **Thanh ghi đoạn mã CS (Code Segment):** Lưu địa chỉ segment chứa chương trình ngôn ngữ máy.
 - **Thanh ghi đoạn dữ liệu DS (Data Segment):** Lưu địa chỉ segment của đoạn chứa dữ liệu trong chương trình.
 - **Thanh ghi đoạn bổ sung ES (Extra Segment):** Lưu địa chỉ segment của đoạn dữ liệu bổ sung.
 - **Thanh ghi đoạn Stack SS (Stack Segment):** Lưu địa chỉ segment của đoạn stack.
- 4 thanh ghi này có thể truy xuất dữ liệu trên 4 đoạn khác nhau và 1 chương trình chỉ có thể sử dụng cùng một lúc tối đa 4 đoạn.
- CPU 80386 có 2 thanh ghi tương tự như ES là FS và GS.



2. Nhóm thanh ghi đa dụng (general register)

- Gồm bốn thanh ghi **AX, BX, CX, DX**. Các thanh ghi này có thể xem như một thanh ghi 16 bit hoặc hai thanh ghi nhỏ:

$$AX = AH + AL$$

$$BX = BH + BL$$

$$CX = CH + CL$$

$$DX = DH + DL$$

CPU 80386 có thể kéo dài đến 32 bit tạo thành thanh ghi **EAX, EBX, ECX, EDX**.

- Thanh ghi tích lũy AX (Accumulator register): thường dùng để lưu số nhân, số chia trong các phép toán nhân, chia, các phép tính số học, logic và chuyển dữ liệu.

VD: **MUL BH** ; $AX \leftarrow AL * BH$

- Thanh ghi cơ sở BX (Base register): thường dùng để định vị bộ nhớ.

VD: **MOV [BX], AX** ; Lấy nội dung thanh ghi AX đưa vào ô nhớ
; có địa chỉ segment là DS và địa chỉ offset BX.

- Thanh ghi đếm CX (Count register): dùng để định số lần lặp của vòng lặp.
- Thanh ghi dữ liệu DX (Data register): dùng để lưu kết quả của các phép toán nhân và chia, định địa chỉ cổng trong các lệnh nhập xuất cổng.

VD: **MOV AL, 62** ; $AL \leftarrow 62$

MOV DX, 1000 ; $DX \leftarrow 1000$

OUT DX, AL ; Đưa nội dung của AL (tức 62) ra cổng 1000

Phần 3:

1. Cú pháp lệnh hợp ngữ

- Một chương trình hợp ngữ gồm các Statement (mệnh đề) được viết liên tiếp nhau , mỗi Statement được viết trên 1 dòng. Một Statement có thể là:
 - 1 lệnh (Instruction) : được chuyển thành mã máy.
 - 1 chỉ dẫn (Assembler directive) : không chuyển thành mã máy

- Các lệnh gồm 4 trường :

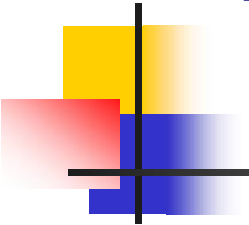
Name	Operation	Operand(s)	Comment
------	-----------	------------	---------

Các trường cách nhau ít nhất 1 khoảng trắng hoặc 1 TAB

Ví dụ: **START :** **MOV CX,5** ; khởi tạo thanh ghi CX

- Hay một chỉ dẫn của ASM :

Ví dụ: **MAIN** **PROC** ; tạo một thủ tục có tên là MAIN



1.1 Trường Tên (Name Field)

- Trường tên dùng cho nhãn lệnh, tên thủ tục và tên biến. ASM sẽ chuyển tên thành địa chỉ bộ nhớ .
 - Tên có thể dài từ 1 đến 31 ký tự .
 - Trong tên chứa các ký tự từ **a-z**, các số và các ký tự đặc biệt sau: **?** , **@** , **_** , **\$** và dấu.
 - Không được phép có ký tự trống trong phần tên.
 - Tên không được bắt đầu bằng một số .
 - ASM không phân biệt giữa ký tự viết thường và viết hoa .
- Các ví dụ về tên hợp lệ và không hợp lệ trong ASM.

Tên hợp lệ

COUNTER1

@CHARACTER

SUM_OF_DIGITS

DONE?

.TEST

Tên không hợp lệ

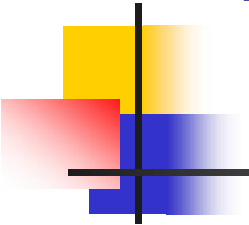
TWO WORDS

2ABC

A45.28

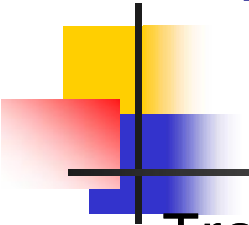
YOU&ME

ADD-REPEAT



1.2 Trường toán tử (operation field)

- Đối với 1 lệnh trường toán tử chứa ký hiệu (Symbol) của phép toán (Operation code = OPCODE). ASM sẽ chuyển ký hiệu phép toán thành mã máy .
- Thông thường ký hiệu mã phép toán mô tả chức năng của phép toán. Ví dụ: **ADD, SUB, INC, DEC, INT...**
- Đối với chỉ dẫn của ASM, trường toán tử chứa một opcode giả (pseudo operation code = pseudo-op). ASM không chuyển pseudo-op thành mã máy mà hướng dẫn ASM thực hiện một việc gì đó ví dụ tạo ra một thủ tục, định nghĩa các biến ...



1.3 Trường các toán hạng (Operand(s) field)

- Trong 1 lệnh, trường toán hạng chỉ ra các số liệu tham gia trong lệnh đó.

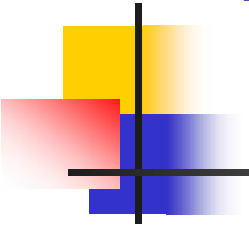
- 1 lệnh có thể không có toán hạng , có 1 hoặc 2 toán hạng .

Ví dụ: **NOP** ; không có toán hạng

INC AX ; 1 toán hạng

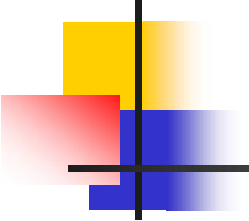
ADD WORD1,2 ; 2 toán hạng,
; cộng 2 với nội dung của từ nhớ WORD1

- Trong các lệnh 2 toán hạng toán hạng đầu là toán hạng đích (destination operand). Toán hạng đích thường là thanh ghi hoặc vị trí nhớ dùng để lưu trữ kết quả. Toán hạng thứ hai là toán hạng nguồn. Toán hạng nguồn thường không bị thay đổi sau khi thực hiện lệnh .
- Đối với một chỉ dẫn của ASM, trường toán hạng chứa một hoặc nhiều thông tin mà ASM dùng để thực thi chỉ dẫn .



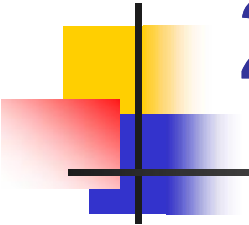
1.4 Trường chú thích (Comment field)

- Trường chú thích là một tùy chọn trong ngôn ngữ ASM. Lập trình viên dùng trường chú thích để thuyết minh về câu lệnh. Điều này là cần thiết vì ngôn ngữ ASM là ngôn ngữ cấp thấp (low level) vì vậy sẽ rất khó hiểu chương trình nếu nó không được chú thích một cách đầy đủ và rõ ràng. Tuy nhiên không nên có chú thích đối với mọi dòng của chương trình, kể cả những lệnh mà ý nghĩa của nó đã rất rõ ràng.



2. Các kiểu số liệu trong chương trình hợp ngữ

- CPU chỉ làm việc với các số nhị phân. Vì vậy ASM phải chuyển tất cả các loại số liệu thành số nhị phân.
- Trong một chương trình hợp ngữ cho phép biểu diễn số liệu dưới dạng nhị phân, thập phân hoặc thập lục phân.



2.1 Các số

- 1 số nhị phân là 1 dãy các bit **0**, **1** và kết thúc bằng **b** hoặc **B**
- 1 số thập phân là 1 dãy các **chữ số thập phân** và kết thúc bởi **d** hoặc **D** (có thể không cần)
- 1 số hex phải bắt đầu bởi 1 **chữ số thập phân** và phải kết thúc bởi **h** hoặc **H**.
- Ví dụ các biểu diễn số hợp lệ và không hợp lệ trong ASM :

Số

10111

10111b

-2183D

1B4DH

1B4D

FFFFH

0FFFFH

Loại

thập phân

nhị phân

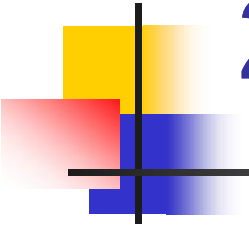
thập phân

hex

số hex không hợp lệ

số hex không hợp lệ

số hex

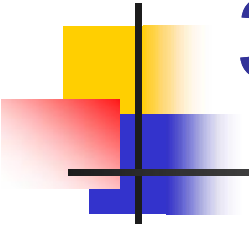


2.2 Các ký tự

- Ký tự và một chuỗi các ký tự phải được đóng giữa hai dấu ngoặc đơn hoặc hai dấu ngoặc kép.

Ví dụ: 'A' và "HELLO".

- Các ký tự đều được chuyển thành mã ASCII.
- Do đó trong một chương trình ASM xem khai báo 'A' hay 41h (mã ASCII của A) là giống nhau



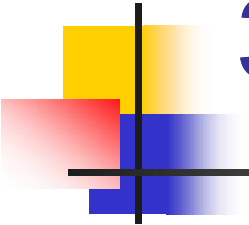
3. Các biến (Variables)

- Trong ASM biến đóng vai trò như trong ngôn ngữ cấp cao. Mỗi biến có một loại dữ liệu và nó được gán một địa chỉ bộ nhớ sau khi dịch chương trình. Bảng sau đây liệt kê các toán tử giả dùng để định nghĩa các kiểu số liệu.

PSEUDO-OP

STANDS FOR

DB	define byte
DW	define word (doublebyte)
DD	define doubleword (2 từ liên tiếp)
DQ	define quadword (4 từ liên tiếp)
DT	define tenbytes (10 bytes liên tiếp)



3.1. Kiểu byte

- Để định nghĩa biến kiểu byte cú pháp như sau:

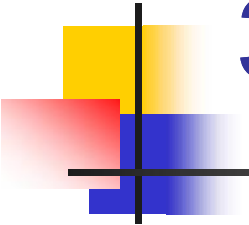
NAME **DB** **initial_value**

Ví dụ : **ALPHA** **DB** **4**

- Chỉ dẫn này sẽ gán tên ALPHA cho một byte nhớ trong bộ nhớ mà giá trị ban đầu của nó là 4.
- Nếu giá trị của byte là không xác định thì đặt dấu chấm hỏi (?) vào giá trị ban đầu.

Ví dụ : **BYT** **DB** **?**

- Đối với biến kiểu byte vùng giá trị nó lưu trữ được là từ -128 đến 127 đối với số có dấu và 0 đến 255 đối với số không dấu .



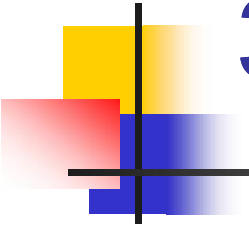
3.2. Kiểu word

- Định nghĩa một biến kiểu Word như sau:

NAME	DW	initial_value
------	----	---------------

Ví dụ :	WRD	DW	-2
---------	-----	----	----

- Có thể dùng dấu ? để thay thế cho biến từ có giá trị không xác định. Vùng giá trị của biến từ là -32768 đến 32767 đối với số có dấu và 0 đến 56535 đối với số không dấu .



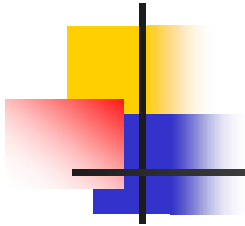
3.3 Mảng (arrays)

- Mảng là một loạt các **byte** nhớ hoặc **word** nhớ liên tiếp nhau. Ví dụ để định nghĩa 1 mảng 3 byte là **B_ARRAY**, giá trị ban đầu là **10h**, **20h** và **30h** ta có thể viết :

B_ARRAY DB 10h,20h,30h

- **B_ARRAY** là tên được gán cho byte đầu tiên
 - **B_ARRAY+1** là tên của byte thứ hai
 - **B_ARRAY+2** là tên của byte thứ ba
- Nếu ASM gán địa chỉ offset là 0200h cho mảng **B_ARRAY** thì nội dung bộ nhớ sẽ như sau :

SYMBOL	ADDRESS	CONTENTS
B_ARRAY	200h	10h
B_ARRAY+1	201h	20h
B_ARRAY+2	202h	30h

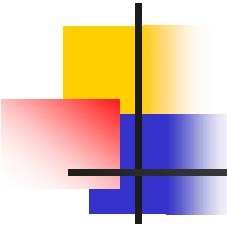


- Chỉ dẫn sau đây sẽ định nghĩa một mảng 4 phần tử có tên là W_ARRAY:

W_ARRAY DW 1000,40,29887,329

- Giả sử mảng bắt đầu tại 0300h, bộ nhớ như sau:

SYMBOL	ADDRESS	CONTENTS
W_ARRAY	300h	1000d
W_ARRAY+2	302h	40d
W_ARRAY+4	304h	29887d
W_ARRAY+6	306h	329d

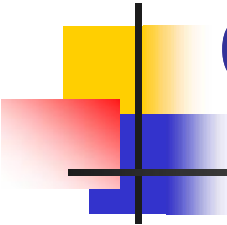


Byte thấp và byte cao của một word

- Đôi khi chúng ta cần truy xuất tới byte thấp và byte cao của một biến Word.
- Giả sử chúng ta định nghĩa :

WORD1 DW 1234h

- Byte thấp của WORD1 chứa 34h
- Còn byte cao của WORD1 chứa 12h
- Ký hiệu địa chỉ của byte thấp là WORD1
- Còn ký hiệu địa chỉ của byte cao là WORD1+1 .



Chuỗi các ký tự (character strings)

- Một mảng các mã ASCII có thể được định nghĩa bằng một chuỗi các ký tự .

Ví dụ : **LETTERS DW 41h,42h,43h**

tương đương với

LETTERS DW 'ABC '

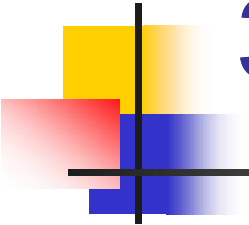
- Bên trong một chuỗi , ASM sẽ phân biệt chữ hoa và chữ thường . Vì vậy chuỗi 'abc' sẽ được chuyển thành 3 bytes : **61h, 62h và 63h.**

- Ta cũng có thể tổ hợp các ký tự và các số trong một định nghĩa.

Ví dụ : **MSG DB 'HELLO', 0AH, 0DH, '\$'**

tương đương với

MSG DB 48H,45H,4CH,4Ch,4FH,0AH,0DH,24H



3.4 Các hằng (constants)

- Trong một chương trình các hằng có thể được đặt tên nhờ chỉ dẫn EQU (equates). Cú pháp của EQU là :

NAME EQU constant

Ví dụ : LF EQU 0AH

sau khi có khai báo trên thì LF được dùng thay cho 0Ah trong chương trình. Vì vậy ASM sẽ chuyển các lệnh :

MOV DL,0Ah

và MOV DL,LF

thành cùng một mã máy.

- Cũng có thể dùng EQU để định nghĩa một chuỗi

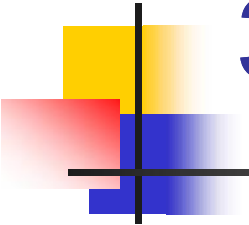
Ví dụ: PROMPT EQU 'TYPE YOUR NAME '

Sau khi có khai báo này, thay cho

MSG DB 'TYPE YOUR NAME '

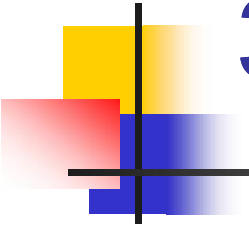
chúng ta có thể viết

MSG DB PROMPT



3.5. Các lệnh cơ bản

- CPU 8086 có rất nhiều lệnh, trong chương này, chúng ta sẽ xem xét 7 lệnh đơn giản của 8086 mà chúng thường được dùng với các thao tác di chuyển số liệu và thực hiện các phép toán số học.
- Trong phần sau đây, **WORD1** và **WORD2** là các biến kiểu **word**, **BYTE1** và **BYTE2** là các biến kiểu **byte** .



3.5.1 Lệnh MOV và XCHG

- Lệnh MOV dùng để chuyển số liệu giữa các thanh ghi, giữa 1 thanh ghi và 1 vị trí nhớ hoặc để di chuyển trực tiếp một số đến một thanh ghi hoặc một vị trí nhớ.
- Cú pháp của lệnh MOV là :

MOV Destination , Source

- Ví dụ : MOV AX,WORD1

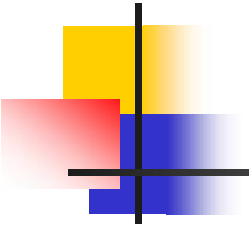
; lấy nội dung của WORD1 đưa vào thanh ghi AX

MOV AX,BX

; AX lấy nội dung của BX, BX không thay đổi

MOV AH,'A'

; AX lấy giá trị 41h



! Chú ý lệnh MOV

- Lệnh Mov không làm ảnh hưởng thanh ghi cờ hiệu
- Không thể chuyển dữ liệu trực tiếp giữa 2 toán hạng mà phải dùng 1 thanh ghi trung gian

Ví dụ: Chuyển dữ liệu 16bit từ Var1 vào Var2

Mov AX,Var1

Mov Var2,AX

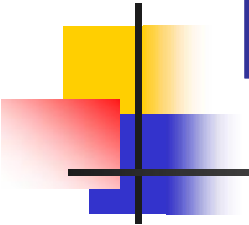
- Không thể chuyển trực tiếp 1 hằng vào 1 thanh ghi đoạn, muốn chuyển ta phải dùng 1 thanh ghi trung gian

Ví dụ: Muốn chuyển giá trị B800h vào thanh ghi DS

Mov AX,0B800h

Mov DS,AX

- Không thể chuyển trực tiếp giữa hai thanh ghi đoạn



Lệnh XCHG

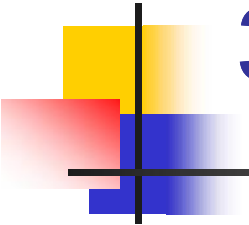
- Lệnh XCHG (Exchange) dùng để trao đổi nội dung của 2 thanh ghi hoặc của một thanh ghi và một ô nhớ.

Ví dụ : **XCHG AH,BL**

XCHG AX,WORD1

; trao đổi nội dung của thanh ghi AX và biến WORD1.

- Không thể dùng lệnh này với các thanh ghi đoạn



3.5.2 Lệnh ADD, SUB, INC, DEC

- Lệnh ADD và SUB được dùng để cộng và trừ nội dung của 2 thanh ghi, của một thanh ghi và một vị trí nhớ, hoặc cộng (trừ) một số với một thanh ghi hoặc một vị trí nhớ.
- Cú pháp:

ADD Destination , Source

SUB Destination , Source

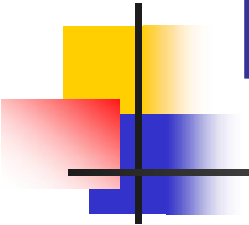
Ví dụ :

ADD WORD1, AX

ADD BL , 5

SUB AX,DX ; AX=AX-DX

- Không thể cộng trực tiếp thanh ghi đoạn



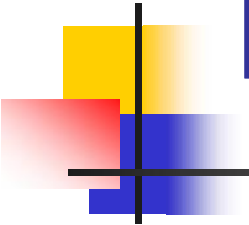
Lệnh ADD, SUB

- Không được phép cộng hoặc trừ trực tiếp giữa 2 vị trí nhớ.

Để giải quyết vấn đề này người ta phải di chuyển byte (word) nhớ đến một thanh ghi sau đó mới cộng hoặc trừ thanh ghi này với một byte (word) nhớ khác.

Ví dụ:

```
MOV      AL, BYTE2  
ADD      BYTE1, AL
```



Lệnh INC, DEC

- Lệnh INC (increment) để cộng 1 vào nội dung thanh ghi hoặc một vị trí nhớ.
- Lệnh DEC (decrement) để giảm bớt 1 khỏi một thanh ghi hoặc 1 vị trí nhớ.

Cú pháp:

INC Destination

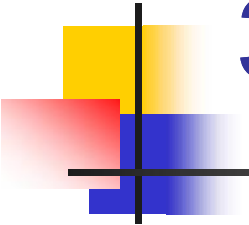
DEC Destination

Ví dụ :

INC WORD1

INC AX

DEC BL



3.5.3 Lệnh NEG (negative)

- Lệnh NEG để đổi dấu của một thanh ghi hoặc một vị trí nhớ.

Cú pháp :

NEG **destination**

Công dụng để thực hiện phép trừ 1 hằng với 1 toán hạng (hằng không thể đứng trước). Ví dụ:

SUB **100,AL** ; ASM không cho phép nên ta viết lại như sau:

SUB **AL,100**

NEG **AL**

3.6 Chuyển ngôn ngữ cấp cao thành ngôn ngữ ASM

- Giả sử A, B là 2 biến word. Anh/chị chuyển các mệnh đề sau ra ngôn ngữ ASM.

1.6.1 Mệnh đề B=A

MOV	AX,A	; đưa A vào AX
MOV	B,AX	; đưa AX vào B

1.6.2 Mệnh đề A=5-A

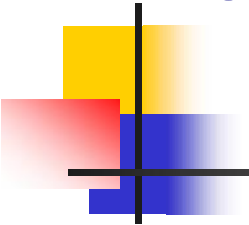
MOV	AX,5	; đưa 5 vào AX
SUB	AX,A	; AX=5-A
MOV	A,AX	; A=5-A

cách khác :

NEG	A	;A=-A
ADD	A,5	;A=5-A

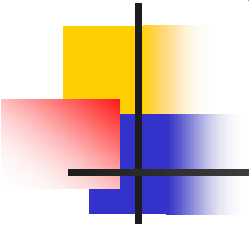
1.6.3 Mệnh đề A=B-2*A

MOV	AX,B	;Ax=B
SUB	AX,A	;AX=B-A
SUB	AX,A	;AX=B-2*A
MOV	A,AX	;A=B-2*A



3.7 Cấu trúc của một chương trình hợp ngữ

- Một chương trình ngôn ngữ máy bao gồm:
 - mã (code)
 - số liệu (data)
 - ngăn xếp (stack).
- Mỗi một phần chiếm một đoạn bộ nhớ. Mỗi một đoạn chương trình là được chuyển thành một đoạn bộ nhớ bởi ASM.



3.7.1 Các kiểu bộ nhớ (memory models)

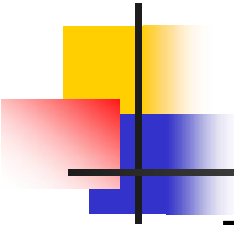
- Độ lớn của mã và số liệu trong một chương trình được quy định bởi chỉ dẫn MODEL nhằm xác định kiểu bộ nhớ dùng với chương trình.

Cú pháp của chỉ dẫn MODEL như sau :

.MODEL **memory_model**

- Bảng sau cho thấy các kiểu bộ nhớ :

MODEL	DESCRIPTION
TINY	Code, data nằm trong 1 đoạn
SMALL	Code nằm trong 1 đoạn, data nằm trong 1 đoạn
MEDIUM	Code nhiều hơn 1 đoạn , data trong 1 đoạn
COMPACT	Code trong 1 đoạn, data nhiều hơn 1 đoạn
LARGE	Code, data lớn hơn 1 đoạn, array không quá 64KB
HUGE	Code, data lớn hơn 1 đoạn, array lớn hơn 64KB



3.7.2 Đoạn số liệu

- Đoạn số liệu của chương trình chứa các khai báo biến, khai báo hằng...Để bắt đầu đoạn số liệu chúng ta dùng chỉ dẫn DATA với cú pháp như sau :

.DATA

;khai báo tên các biến , hằng và mảng

Ví dụ :

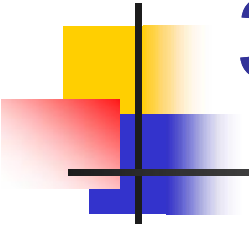
.DATA

WORD1 DW 2 ; gán 2 vào word1

WORD2 DW 5

MSG DB 'THIS IS A MESSAGE '

MASK EQU 10010010B ; hằng



3.7.3 Đoạn ngăn xếp

- Mục đích của việc khai báo đoạn ngăn xếp là dành một vùng nhớ (vùng **stack**) để lưu trữ cho stack.

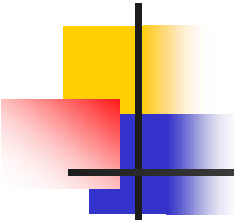
Cú pháp:

.STACK size

nếu không khai báo size thì 1KB được dành cho vùng stack.

.STACK 100h

;dành 256 bytes (=100h) cho vùng stack



3.7.4 Đoạn mã

- Bắt đầu đoạn mã chứa các lệnh của chương trình:

.CODE

- Bên trong đoạn mã các lệnh thường được tổ chức thành thủ tục (procedure), có cấu trúc như sau:

name **PROC**

;thân của thủ tục

name **ENDP**

- Cấu trúc của 1 chương trình, phần CODE là thủ tục có tên MAIN:

.MODEL SMALL

.STACK 100h

.DATA

.CODE **; định nghĩa số liệu tại đây**

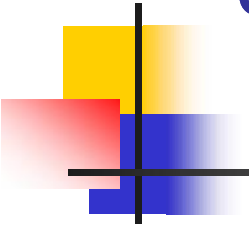
MAIN **PROC**

;thân của thủ tục MAIN

MAIN **ENDP**

; các thủ tục khác nếu có

END **MAIN**



3.8 Các lệnh vào ra

Lệnh INT (interrupt)

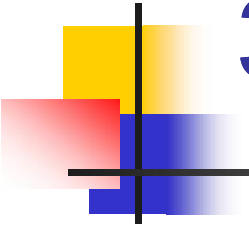
- Để gọi các chương trình con của BIOS và DOS có thể dùng lệnh INT với cú pháp như sau :

INT interrupt_number

ở đây **interrupt_number** là một số mà nó chỉ định một routine.

Ví dụ: **INT 16h**

; gọi routine thực hiện việc nhập số liệu từ Keyboard.



3.8.1 Lệnh INT 21h

- **INT 21h** dùng để gọi một số lớn các các hàm (function) của DOS. Tùy theo giá trị mà chúng ta đặt vào thanh ghi **AH**, **INT 21h** sẽ gọi chạy một routine tương ứng .
- Trong phần này ta sẽ quan tâm đến 2 hàm:

FUNCTION NUMBER

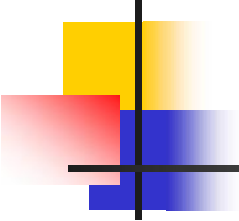
ROUTINE

1

Nhập từ bàn phím một ký tự

2

Xuất ra màn hình 1 ký tự



FUCNTION 1 :

Nhập từ bàn phím một ký tự

Input : **AH=1**

Output: **AL= mã ASCII của ký tự vừa nhập**

AL=0 nếu không nhấn phím

Để gọi routine này thực hiện các lệnh sau:

MOV AH,1 ; input key function

**INT 21h ; AL chứa mã ASCII của ký tự
vừa nhập và hiển thị ra màn hình**

FUNCTION 2 :

Xuất ra màn hình 1 ký tự

Input : **AH=2**

DL= mã ASCII của ký tự cần xuất

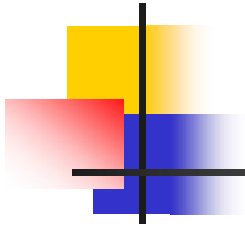
Output: **AL= mã ASCII của ký tự cần xuất**

Ví dụ: **MOV AH,2**

MOV DL,'?' ; ký tự '?'

INT 21H ; hiển thị ký tự '?'

Nếu DL chứa ký tự điều khiển thì khi gọi INT 21h, ký tự điều khiển sẽ được thực hiện



ASCII code (Hex)

FUNCTION

7

beep

8

b.space – nhập b.phím 1 ký tự 0 h.thị

9

tab – xuất một chuỗi ký tự ra màn hình

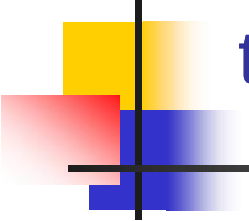
A

line feed - nhập chuỗi từ bàn phím

D

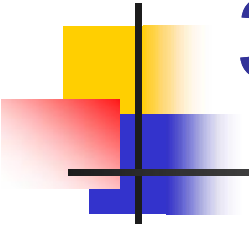
carriage return (Enter)

3.9 Chương trình ví dụ: chương trình đọc một ký tự từ bàn phím và in nó trên đầu dòng mới



```
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
; display dấu nhắc
MOV AH,2
MOV DL,'?'
INT 21H
; nhập 1 ký tự
MOV AH,1 ; hàm đọc ký tự
INT 21H ; ký tự được đưa vào AL
MOV BL,AL ; cất ký tự trong BL
```

```
; nhảy đến dòng mới
MOV AH,2 ; hàm xuất 1 ký tự
MOV DL,0DH ; ký tự carriage return
INT 21H ; thực hiện c.r.
MOV DL,0AH ; ký tự line feed
INT 21H ; thực hiện line feed
; xuất ký tự
MOV DL,BL ; đưa ký tự vào DL
INT 21H ; xuất ký tự
; trở về DOS
MOV AH,4CH ; hàm thoát về DOS
INT 21H ; exit to DOS
MAIN ENDP
END MAIN
```



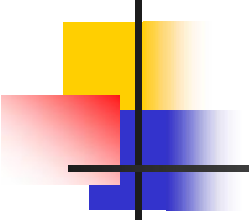
3.10 Xuất một chuỗi ký tự

- Trong chương trình trên đây ta đã dùng INT 21H hàm 2 và 4 để đọc và xuất một ký tự.
- Hàm 9 ngắt 21H có thể dùng để xuất một chuỗi ký tự.

Input : DX=địa chỉ của chuỗi

Chuỗi kết thúc bằng dấu '\$'

- Ký tự \$ ở cuối chuỗi sẽ không được in lên màn hình. Nếu chuỗi có chứa ký tự điều khiển thì chức năng tương ứng sẽ thực hiện.

- 
- Chương trình in lên màn hình chuỗi “HELLO!”. Chuỗi HELLO được định nghĩa như sau:

MSG DB ‘HELLO!\$’

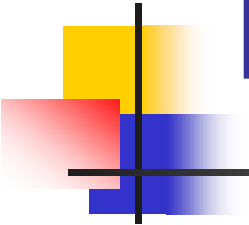
- Lệnh LEA (Load Effective Address) nạp địa chỉ

LEA destination , source

- Ngắt 21h, hàm số 9 sẽ xuất một chuỗi ký tự ra màn hình với điều kiện địa chỉ của biến chuỗi phải ở trên DX . Có thể thực hiện điều này bởi lệnh :

LEA DX,MSG

; đưa địa chỉ offset của biến MSG vào DX



PRINT STRING PROGRAM

```
.MODEL      SMALL
.STACK      100H
.DATA
    MSG     DB      'HELLO!$'
.CODE
MAIN        PROC
; initialize DS
    MOV     AX,@DATA
    MOV     DS,AX
; display message
    LEA     DX,MSG
    MOV     AH,9
    INT     21H
; return to DOS
    MOV     AH,4CH
    INT     21H
MAIN        ENDP
END MAIN
```

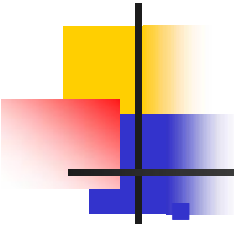
CASE COVERT PROGRAM

```
.MODEL    SMALL
.STACK    100H
.DATA
    CR     EQU     0DH
    LF     EQU     0AH
    TB1     DB      'Nhập vào một ký tự thường:$'
    TB2     DB      0DH,0AH,'Ký tự vừa nhập viết hoa là : '
    CHAR     DB      ?, '$' ; định nghĩa biến CHAR có giá trị ban đầu chưa xác định
.CODE
MAIN      PROC
; INITIALIZE DS
    MOV     AX,@DATA
    MOV     DS,AX
;PRINT PROMPT USER
    LEA     DX,TB1           ; lấy thông điệp số 1
    MOV     AH,9
    INT     21H              ; xuất nó ra màn hình
;nhập vào một ký tự thường và đổi nó thành ký tự hoa
    MOV     AH,1             ; nhập vào 1 ký tự
    INT     21H              ; cất nó trong AL
    SUB     AL,20H           ; đổi thành chữ hoa và cất nó trong AL
    MOV     CHAR, AL         ; cất ký tự trong biến CHAR
; xuất ký tự trên dòng tiếp theo
    LEA     DX, TB2          ; lấy thông điệp thứ 2
    MOV     AH,9
    INT     21H              ; xuất chuỗi 2, vì TB2 0 kết thúc bởi ký tự $ nên tiếp tục xuất ký tự có trong biến CHAR
;dos exit
    MOV     AH,4CH
    INT     21H              ; dos exit
MAIN      ENDP
END       MAIN
```

- Yêu cầu nhập 1 ký tự, sau đó thay đổi ký tự đã nhập thành ký tự đi liền trước theo thứ tự ASCII và in ra màn hình

CASE PREVIOUS PROGRAM

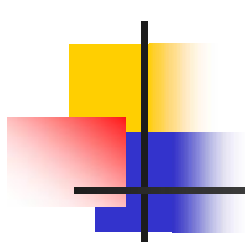
```
.MODEL    SMALL
.STACK    100H
.DATA
    CR     EQU     0DH
    LF     EQU     0AH
    MSG1    DB      'Nhập vào 1 ký tự:$'
    MSG2    DB      0DH,0AH,'Ký tự liên trước nó là : '
    CHAR    DB      '?,$' ; định nghĩa biến CHAR có giá trị ban đầu chưa xác định
.CODE
MAIN      PROC
; INITIALIZE DS
    MOV     AX,@DATA
    MOV     DS,AX
;PRINT PROMPT USER
    LEA     DX,MSG1          ; lấy thông điệp số 1
    MOV     AH,9
    INT     21H              ; xuất nó ra màn hình
;nhập vào một ký tự thường và đổi nó thành ký tự hoa
    MOV     AH,1             ; nhập vào 1 ký tự
    INT     21H              ; cất nó trong AL
    DEC     AL                ;
    MOV     CHAR, AL         ; cất ký tự trong biến CHAR
; xuất ký tự trên dòng tiếp theo
    LEA     DX, MSG2         ; lấy thông điệp thứ 2
    MOV     AH,9
    INT     21H              ; xuất chuỗi 2, vì MSG2 0 kết thúc bởi ký tự $ nên tiếp tục xuất ký tự có trong biến CHAR
;dos exit
    MOV     AH,4CH
    INT     21H              ; dos exit
MAIN      ENDP
END       MAIN
```

3.1 Ví dụ về lệnh nhảy

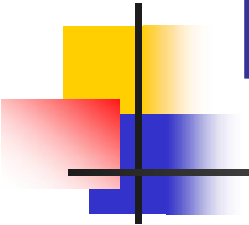
Để hình dung được lệnh nhảy làm việc như thế nào chúng ta hãy viết chương trình in ra toàn bộ tập các ký tự IBM .

```
.MODEL      SMALL
.STACK      100H
.CODE
MAIN                PROC
    MOV      AH,2          ; hàm xuất ký tự
    MOV      CX,256        ; số ký tự cần xuất
    MOV      DL,0          ; DL giữ mã ASCII của ký tự NUL
PRINT_LOOP :
    INT      21H           ;display character
    INC      DL
    DEC      CX
    JNZ      PRINT_LOOP    ;nhảy đến print_loop nếu CX# 0
;DOS EXIT
    MOV      AH,4CH
    INT      21H
MAINENDP
END MAIN
```



Nhảy có dấu

SYMBOL	DESCRIPTION	CONDITION FOR JUMPS
JG/JNLE	jump if greater than	ZF=0 and SF=OF
	jump if not less than or equal to	
JGE/JNL	jump if greater than or equal to	SF=OF
	jump if not less or equal to	
JL/JNGE	jump if less than	
	jump if not greater or equal	SF<>OF
JLE/JNG	jump if less than or equal	ZF=1 or SF<>OF
	jump if not greater	

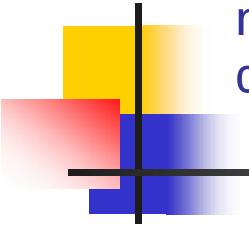


Lệnh CMP (Compare)

- Các lệnh nhảy thường lấy kết quả của lệnh Compare như là điều kiện. Cú pháp của lệnh CMP là :

CMP destination, source

- Lệnh này so sánh toán hạng nguồn và toán hạng đích bằng cách tính hiệu $Dest - Src$. Kết quả không được cất giữ . Lệnh CMP giống như lệnh SUB, chỉ khác: lệnh CMP toán hạng đích không thay đổi.

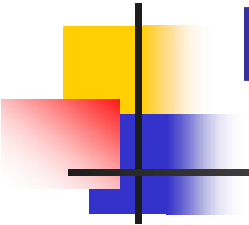


Hiển thị câu hỏi “Is it after 12 noon (Y/N)? và chờ ta bấm một phím, nếu ta bấm y hay Y thông báo “Good afternoon!”, còn bấm n hay N (hay bất kỳ phím) thông báo “Good morning!”

```
.MODEL    SMALL
.STACK    100h
.DATA
    TimePrompt      DB      'Is it after 12 noon (Y/N)?$
    GoodMorningMsg   LABEL BYTE
                        DB      0Dh,0Ah, 'Good morning!',0Dh,0Ah,'$'
    GoodAfternoonMsg LABEL BYTE
                        DB      0Dh,0Ah, 'Good afternoon!',0Dh,0Ah,'$'
.CODE
    Begin:
        MOV     AX, @DATA
        MOV     DS, AX
        LEA     DX, TimePrompt
        MOV     AH, 09h
        INT     21h
        MOV     AH, 01h      ;Nhập một ký tự chứa vào AL
        INT     21h
        CMP     AL, 'y'      ; Ký tự đã gõ là y?
        JZ      IsAfternoon  ; Đúng, nhảy tới IsAfternoon
        CMP     AL, 'Y'      ; Ký tự gõ là Y?
        JNZ     IsMorning    ; Không nhảy tới IsMorning
```

```
IsAfternoon:
;Không gõ y hay Y
    LEA     DX, GoodAfternoonMsg
    JMP     DisplayGreeting
IsMorning:
    LEA     DX, GoodMorningMsg
DisplayGreeting:
    MOV     AH, 09h
    INT     21h

    MOV     AX, 4C00h
    INT     21h
END        Begin
```



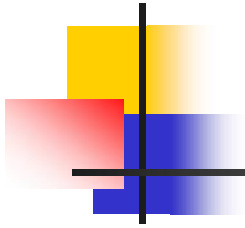
Diễn dịch lệnh nhảy có điều kiện

Ví dụ trên đây về lệnh CMP cho phép lệnh nhảy sau nó chuyển điều khiển đến nhãn BELOW các lệnh

CMP AX,BX

JG BELOW

có nghĩa là nếu $AX > BX$ thì nhảy đến nhãn BELOW

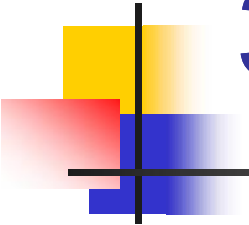


Mặc dù lệnh CMP được thiết kế cho các lệnh nhảy. Nhưng lệnh nhảy có thể đứng trước 1 lệnh khác, chẳng hạn :

DEC AX

JL THERE

có nghĩa là nếu AX trong diễn dịch có dấu < 0 thì điều khiển được chuyển cho THERE



3.11 Lệnh JMP

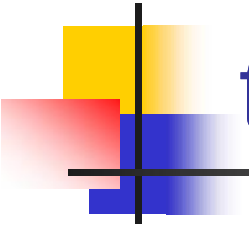
- Lệnh JMP (jump) là lệnh nhảy không điều kiện.

Cú pháp:

JMP **destination**

Trong đó **destination** là một nhãn ở trong cùng 1 đoạn với lệnh JMP.

Lệnh JMP dùng để khắc phục hạn chế của các lệnh nhảy có điều kiện (không quá 126 bytes kể từ vị trí của lệnh nhảy có điều kiện)



Ví dụ chúng ta có đoạn chương trình sau :

TOP:

; thân vòng lặp

DEC CX

JNZ TOP ; nếu CX>0 tiếp tục lặp

MOV AX,BX

giả sử thân vòng lặp chứa nhiều lệnh mà nó vượt khỏi 126 bytes trước lệnh JNZ TOP . Có thể giải quyết tình trạng này bằng các lệnh sau :

TOP:

; thân vòng lặp

DEC CX

JNZ BOTTOM ; nếu CX>0 tiếp tục lặp

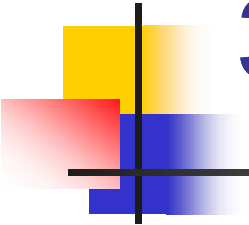
JMP EXIT

BOTTOM:

JMP TOP

EXIT:

MOV AX,BX



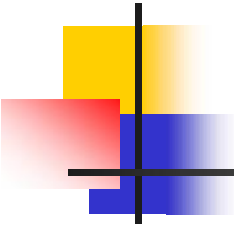
3.12 Cấu trúc rẽ nhánh

- Trong ngôn ngữ cấp cao cấu trúc rẽ nhánh cho phép một chương trình rẽ nhánh đến những đoạn khác nhau tùy thuộc vào các điều kiện . Trong phần này chúng ta sẽ xem xét 3 cấu trúc

a. IF-THEN

b. IF_THEN_ELSE

c. CASE



a. IF-THEN

Cấu trúc IF-THEN có thể diễn đạt như sau :

IF condition is true

THEN

execute true branch statements

END IF

Ví dụ : Thay thế giá trị trên AX bằng giá trị tuyệt đối của nó

IF AX<0

THEN

replace AX by -AX

END-IF

Có thể mã hoá như sau :

; if AX<0

CMP AX,0

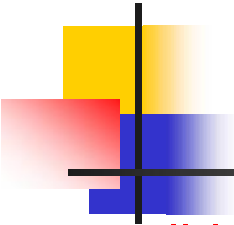
JNL END_IF ; no, exit

;then

NEG AX

; yes, đổi dấu

END_IF :



b. IF_THEN_ELSE

IF condition is true

THEN

execute true branch statements

ELSE

execute false branch statements

END_IF

Ví dụ : giả sử AL và BL chứa ASCII code của 1 ký tự. Hãy xuất ra màn hình ký tự trước (theo thứ tự ký tự). Thuật toán

IF AL <= BL

THEN

display AL

ELSE

display character in BL

END_IF

Có thể mã hoá như sau :

MOV AH,2

; chuẩn bị xuất ký tự

;if AL <= BL

CMP AL,BL ;AL <= BL?

JNBE ELSE_

; nếu không nhỏ hơn hay = BL

;then

MOV DL,AL

JMP DISPLAY

ELSE_:

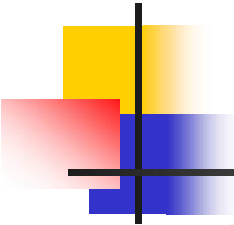
MOV DL,BL

DISPLAY:

INT 21H

END IF :

Biên soạn: Lê Minh Triết



c. CASE

Case là một cấu trúc rẽ nhánh nhiều hướng. Cấu trúc của CASE như sau :

CASE expression

value_1 : Statements_1

value_2 : Statements_2

...

value_n : Statements_n

Ví dụ : Nếu $AX < 0$ thì đặt -1 vào BX

Nếu $AX = 0$ thì đặt 0 vào BX

Nếu $AX > 0$ thì đặt 1 vào BX

Thuật toán :

CASE	AX
< 0	put -1 in BX
$= 0$	put 0 in BX
> 0	put 1 in BX

Có thể mã hoá như sau :

; case AX

CMP AX,0 ;test AX

JL NEGATIVE ;AX<0

JE ZERO ;AX=0

JG positive ;AX>0

NEGATIVE:

MOV BX,-1

JMP END_CASE

ZERO:

MOV BX,0

JMP END_CASE

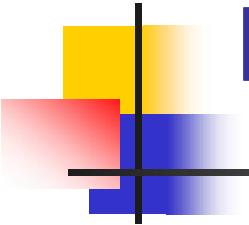
POSITIVE:

MOV BX,1

JMP END_CASE

END_CASE :

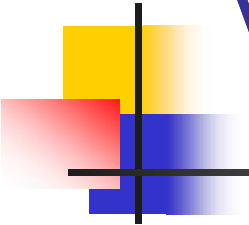
Biên soạn: Lê Minh Triết



Rẽ nhánh với tổ hợp các điều kiện

- Đôi khi tình trạng rẽ nhánh trong các lệnh IF ,CASE cần một tổ hợp các điều kiện dưới dạng :

Condition_1	AND	Condition_2
Condition_1	OR	Condition_2



Ví dụ về điều kiện AND :

- Đọc 1 ký tự và nếu là ký tự hoa thì in ra MH.

Thuật toán :

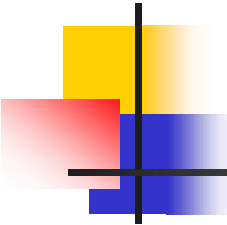
Read a character (into AL)

IF ('A' <= character) AND (character <= 'Z')

THEN

display character

END_IF



Ví dụ về điều kiện AND :

Đọc 1 ký tự và nếu là ký tự hoa thì in ra MH.

; read a character

MOV AH,1

INT 21H ; character in AL

; IF ('A' <= character) AND (character <= 'Z')

CMP AL,'A' ; char >='A'?

JNGE END_IF ;no, exit

CMP AL,'Z' ; char <='Z'?

JNLE END_IF ; no exit

; then display it

MOV DL,AL

MOV AH,2

INT 21H

END_IF :

Ví dụ về điều kiện OR : Đọc một ký tự , nếu ký tự đó là 'Y' hoặc 'y' thì in nó lên màn hình , ngược lại thì kết thúc chương trình .

Thuật toán

Read a character (into AL)

IF (chr ='Y') OR (chr='y')

THEN

display it

ELSE

terminate the program

END_IF

Code

;read a character

MOV AH,1

INT 21H

; character in AL

;IF (chr ='y') OR (chr = 'Y')

CMP AL,'y' ; chr ='y'?

JE THEN ;=, in Chr

CMP AL,'Y' ; chr ='Y'?

JE THEN ;=, in Chr

JMP ELSE_ ;<> , terminate

THEN :

MOV DL,AL

MOV AH,2

INT 21H

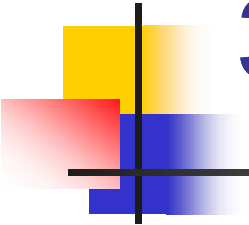
JMP END_IF

ELSE_:

MOV AH,4CH

INT 21h

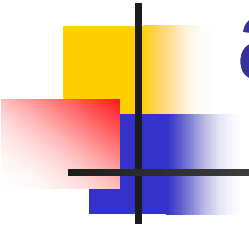
END_IF :



3.13 Cấu trúc lặp

Một vòng lặp gồm nhiều lệnh được lặp lại, số lần lặp phụ thuộc điều kiện.

- a. Vòng FOR
- b. Vòng WHILE
- c. Vòng REPEAT



a. Vòng FOR

Lệnh LOOP có thể dùng để thực hiện vòng FOR.

Cú pháp :

LOOP **destination_label**

Số đếm cho vòng lặp là thanh ghi CX mà ban đầu nó được gán 1 giá trị nào đó . Khi lệnh LOOP được thực hiện CX sẽ tự động giảm đi 1. Nếu CX chưa bằng 0 thì vòng lặp được thực hiện tiếp tục. Nếu CX=0 lệnh sau lệnh LOOP được thực hiện

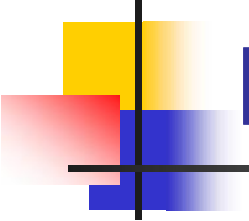
Dùng lệnh LOOP , vòng FOR có thể thực hiện như sau :

; gán cho cho CX số lần lặp

TOP:

; thân của vòng lặp

LOOP TOP



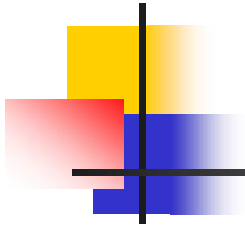
Ví dụ : Dùng vòng lặp in ra 1 hàng 80 dấu '*'

```
MOV CX,80      ; CX chứa số lần lặp
MOV AH,2        ; hàm xuất ký tự
MOV DL,'*'      ; DL chứa ký tự '*'
```

TOP:

```
INT 21h        ; in dấu '*'
LOOPTOP        ; lặp 80 lần
```

Lưu ý rằng vòng FOR cũng như lệnh LOOP thực hiện ít nhất là 1 lần. Do đó nếu ban đầu CX=0 thì vòng lặp sẽ thực hiện lặp đến 65535 lần. Để tránh tình trạng này, lệnh JCXZ (Jump if CX is zero) phải được dùng trước vòng lặp.



Lệnh JXCZ có cú pháp như sau :

JCXZ destination_label

Nếu $CX=0$ điều khiển được chuyển cho destination_label. Các lệnh sau đây sẽ đảm bảo vòng lặp không thực hiện nếu $CX=0$

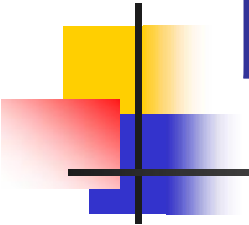
JCXZ SKIP

TOP :

; thân vòng lặp

LOOPTOP

SKIP :



b. Vòng WHILE

Vòng WHILE phụ thuộc vào 1 điều kiện. Nếu điều kiện đúng thì thực hiện vòng WHILE. Vì vậy nếu điều kiện sai thì vòng WHILE không thực hiện gì cả .

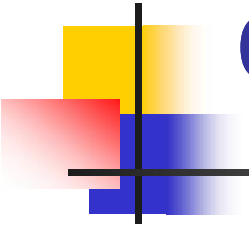
Ví dụ : Viết đoạn mã đếm số ký tự được nhập vào trên cùng một hàng .

```
MOV    DX,0           ; DX để đếm số ký tự
MOV    AH,1           ; hàm đọc 1 ký tự
INT     21h           ; đọc ký tự vào AL
```

WHILE_:

```
CMP     AL,0DH         ; có phải là ký tự CR?
JE      END_WHILE      ; đúng , thoát
INC     DX             ; tăng DX lên 1
INT     21h           ; đọc ký tự
JMP     WHILE_         ; lặp
```

END_WHILE :



c. Vòng REPEAT

Cấu trúc REPEAT: **repeat** **statements**
 until **condition**

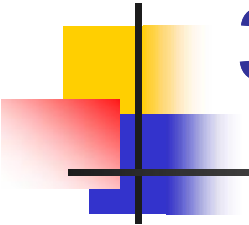
Trong cấu trúc repeat mệnh đề được thi hành đồng thời điều kiện được kiểm tra. Nếu điều kiện đúng thì vòng lặp kết thúc.

Ví dụ : Viết đoạn mã đọc vào các ký tự cho đến khi gặp ký tự trống .

```
          MOV    AH,1                    ; đọc ký tự
REPEAT:
          INT     21h                    ; ký tự trên AL
;until
          CMP    AL,' '                ; AL=' '?
          JNE     REPEAT
```

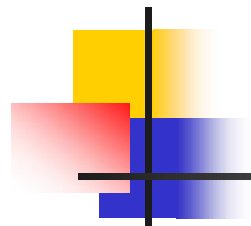
Lưu ý: REPEAT tiến hành ít nhất 1 lần, trong khi đó WHILE có thể không tiến hành lần nào nếu từ đầu điều kiện bị sai.

Biên soạn: Lê Minh Triết



3.14 Lập trình với cấu trúc cấp cao

- Bài toán : Viết chương trình nhắc người dùng gõ vào một dòng văn bản. Đếm số ký tự vừa nhập vào và in ra kết quả
- Kết quả chạy chương trình sẽ như sau :
Nhập vào 1 dòng ký tự:
SU PHAM
Số ký tự nhập vào = 7



```
.MODEL  SMALL
.STACK  100h
.DATA
```

```
    Tbao  DB      'Nhap vao 1 dong ky tu:$'
    Tbao2 DB      0Dh,0Ah, "So ky tu ban nhap vao la:"
    Sockt  DB      ?, '$'
```

```
.CODE
```

```
    Begin:
```

```
        MOV     AX, @DATA
        MOV     DS, AX
        LEA     DX, Tbao
        MOV     AH, 09h
        INT     21h
```

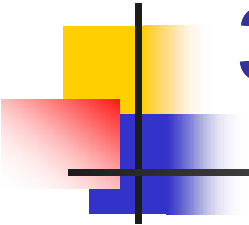
```
        MOV     DX, 0           ; DX để đếm số ký tự
        MOV     AH, 1          ; hàm đọc 1 ký tự
        INT     21h            ; đọc ký tự vào AL
```

```
WHILE_:
        CMP     AL, 0DH
; có phải là ký tự CR?
        JE      END_WHILE;
        đúng , thoát
        INC     DX
; tăng DX lên 1
        INT     21h
; đọc ký tự
        JMP     WHILE_
; lặp
END_WHILE :
        MOV     Sockt, DX

        LEA     DX, Tbao2
        MOV     AH, 09h
        INT     21h

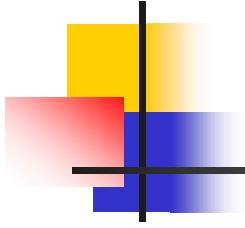
        MOV     AX, 4C00h
        INT     21h
END      Begin
```

Biên soạn: Lê Minh Triết



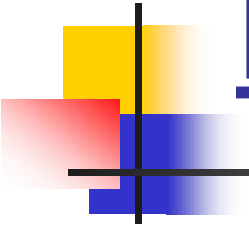
3.14 Lập trình với cấu trúc cấp cao

- Bài toán : Viết chương trình nhắc người dùng gõ vào một dòng văn bản. Trên 2 dòng tiếp theo in ra ký tự viết hoa đầu tiên và ký tự viết hoa cuối cùng theo thứ tự alphabetical. Nếu người dùng gõ vào một ký tự thường, máy sẽ thông báo 'No capitals'
- Kết quả chạy chương trình sẽ như sau :
Type a line of text :
TRUONG DAI HOC SU PHAM
First capital = A
Last capital = U



- Để giải bài toán này ta dùng kỹ thuật lập trình TOP-DOWN, nghĩa là chia nhỏ bài toán thành nhiều bài toán con. Có thể chia bài toán thành 3 bài toán con như sau :

1. Xuất 1 chuỗi ký tự (lời nhắc)
2. Đọc và xử lý 1 dòng văn bản
3. In kết quả



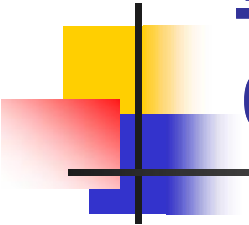
Bước 1: Hiện dấu nhắc .

Bước này có thể mã hoá như sau :

```
MOV AH,9           ; hàm xuất chuỗi  
LEA DX,PRMOPT      ; lấy địa chỉ chuỗi vào DX  
INT 21H            ; xuất chuỗi
```

Dấu nhắc có thể mã hoá như sau trong đoạn số liệu .

```
PROMPT DB 'Type a line of text:',0DH,0AH,'$'
```



Bước 2 : Đọc và xử lý một dòng văn bản

Đọc các ký tự từ bàn phím , tìm ra ký tự đầu và ký tự cuối , nhắc nhở người dùng nếu ký tự gõ vào không phải là ký tự hoa.

Có thể biểu diễn bước này bởi thuật toán sau :

Read a character

WHILE character is not a carriage return DO

IF character is a capital (*)

THEN

IF character precedes first capital

Then

first capital= character

End_if

IF character follows last character

Then

last character = character

End_if

END_IF

Read a character

END_WHILE

Trong đó dòng (*) có nghĩa là điều kiện để ký tự là hoa là điều kiện AND

IF ('A' <= character) AND (character <= 'Z')

Biên soạn: Lê Minh Triết

```

MOV    AH,1      ; đọc ký tự
INT    21H       ; ký tự trên AL
WHILE :
;trong khi ký tự gõ vào không phải là CR thì
    thực hiện
    CMP    AL,0DH    ; CR?
    JE     END_WHILE ;yes, thoát
; nếu ký tự là hoa
    CMP    AL,'A'    ; char >='A'?
    JNGE   END_IF
;không phải ký tự hoa thì nhảy đến END_IF
    CMP    AL,'Z'    ; char <= 'Z'?
    JNLE   END_IF
; không phải ký tự hoa thì nhảy đến END_IF
; thì
; nếu ký tự nằm trước biến FIRST (giá trị ban
    đầu là '[' : ký tự sau Z )
    CMP    AL,FIRST ; char < FIRST ?
    JNL    CHECK_LAST; >=
; thì ký tự viết hoa đầu tiên = ký tự
    MOV    FIRST,AL  ;FIRST=chr.
;end_if

```

```

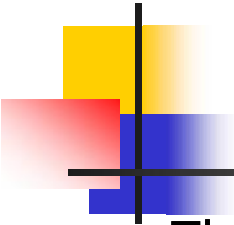
CHECK_LAST:
; nếu ký tự là sau biến LAST ( giá trị ban đầu là
    '@': ký tự trước A)
    CMP    AL,LAST  ; char > LAST ?
    JNG    END_IF   ; <=
;thì ký tự cuối cùng = ký tự
    MOV    LAST, AL ;LAST = character
;end_if
END_IF :
; đọc một ký tự
    INT    21H      ; ký tự trên AL
    JMP    WHILE_   ; lặp
END_WHILE:
Các biến FIRST và LAST định nghĩa như sau:
FIRST    DB    '[' $    ; '[' là ký tự sau Z
LAST     DB    '@ $ '   ; '@' là ký tự trước A

```

```

MOV    AH,1    ; đọc ký tự
INT     21H    ; ký tự trên AL
;trong khi ký tự gõ vào không phải là CR
;thì thực hiện
CMP     AL,0DH ; CR?
JE      END_IF ; thoát
;Xet ktu in hoa
CMP     AL,'A'  ; char >='A'?
JNGE    END_IF
CMP     AL,'Z'  ; char <= 'Z'?
JNLE    END_IF ; không phải
; ký tự hoa thì nhảy đến END_IF
; nếu thỏa thì
;Thong bao ra MH la kt in hoa
Else_:
;Thong bao ra MH o la kt in hoa
END_IF :

```



Bước 3 : In kết quả

Thuật toán

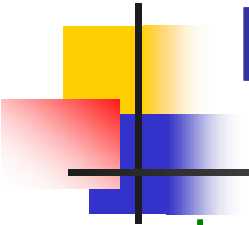
```
IF no capital were typed
THEN
    display 'No capital'
ELSE
    display first capital and last capital
END_IF
```

Bước 3 sẽ phải in ra các thông báo :

NOCAP_MSG nếu không phải chữ in
CAP1_MSG chữ in đầu tiên
CAP2_MSG chữ in cuối cùng

Chúng được định nghĩa như sau trong đoạn số liệu .

NOCAP_MSG	DB	0DH,0AH,'No capitals \$'
CAP1_MSG	DB	0DH,0AH,'First capital= '
FIRST	DB	'[\$ '
CAP2_MSG	DB	0DH,0AH,'Last capital= '
LAST	DB	'@ \$'



Bước 3 có thể mã hoá như sau :

;in kết quả

MOV AH,9 ; hàm xuất ký tự

; IF không có chữ hoa nào được nhập thì FIRST = '['

CMP FIRST,'[' ; FIRST='[' ?

JNE CAPS ; không , in kết quả

;THEN

LEA DX,NOCAP_MSG

INT 21H

CAPS:

LEA DX,CAP1_MSG

INT 21H

LEA DX,CAP2_MSG

INT 21H

; end_if

Chương trình có thể viết như sau:

```
.MODEL    SMALL
.STACK    100h
.DATA
PROMPT    DB    'Type a line of text', 0DH, AH, '$'
NOCAP_MSG DB    0DH,0AH, 'No capitals $'
CAP1_MSG  DB    0DH,0AH, 'First capital='
FIRST     DB    '[ $'
CAP2_MSG  DB    'Last capital = '
LAST      DB    '@ $'
.CODE
MAIN      PROC
; khởi tạo DS
    MOV    AX,@DATA
    MOV    DS,AX
; in dấu nhắc
    MOV    AH,9                ; hàm xuất chuỗi
    LEA    DX,PROMPT           ; lấy địa chỉ chuỗi vào DX
    INT    21H                ; xuất chuỗi
```

;đọc và xử lý 1 dòng văn bản

MOV AH,1 ; đọc ký tự

INT 21H ; ký tự trên AL

WHILE :

;trong khi ký tự gõ vào không phải là CR thì thực hiện

CMP AL,0DH ; CR?

JE END_WHILE ;yes, thoát

; nếu ký tự là hoa

CMP AL,'A' ; char >='A'?

JNGE END_IF ;0 phải K.tự hoa, nhảy đến END_IF

CMP AL,'Z' ; char <= 'Z'?

JNLE END_IF ; 0 phải k.tự hoa, nhảy đến END_IF

; thì

; nếu ký tự nằm trước biến FIRST

CMP AL,FIRST ; char < FIRST ?

JNL CHECK_LAST ; >=

; thì ký tự viết hoa đầu tiên = ký tự

MOV FIRST,AL ; FIRST=character

;end_if

CHECK_LAST:

; nếu ký tự là sau biến LAST

 CMP AL, LAST ; char > LAST ?

 JNG END_IF ; <=

;thì ký tự cuối cùng = ký tự

 MOV LAST, AL ;LAST = character

;end_if

END_IF :

; đọc một ký tự

 INT 21H ; ký tự trên AL

 JMP WHILE_ ; lặp

END_WHILE:

;in kết quả

MOV AH,9 ; hàm xuất ký tự

; IF không có chữ hoa nào được nhập thì FIRST = '['

CMP FIRST,'[' ; FIRST='[' ?

JNE CAPS ; không , in kết quả

;Then

LEA DX,NOCAP_MSG

INT 21H

CAPS:

LEA DX,CAP1_MSG

INT 21H

LEA DX,CAP2_MSG

INT 21H

; end_if

; dos exit

MOV AH,4CH

INT 21h

MAIN ENDP

END MAIN

- Bài tập1:

- Viết chương trình nhập vào 2 số (0-9), tính tổng, hiệu 2 số đó.

- Bài tập2:

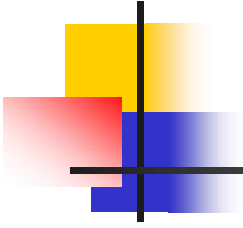
- Nhập vào 1 số nguyên n, không dấu (1-9)
- Tính tổng $= 1 + 2 + \dots + N$

- Bài tập3:

- Nhập vào 1 dãy số Vd: 23145. Đếm số ký số có trong dãy Vd: Dãy có 5 ký số.

- Bài tập4:

- Nhập vào 1 ký tự hoa ('A' <= giả sử <= 'Z').
In dãy ký tự từ đó đến Z



Mul desc

- Desc là thanh ghi bất kỳ, nhân với giá trị trong thanh ghi AL
- Kết quả lưu vào AX
- AAM
- AH,AL