

ASM cơ bản

Tài liệu được thu thập copy từ các diễn đàn.(hungvu11)

Bài 1 đầu tiên:

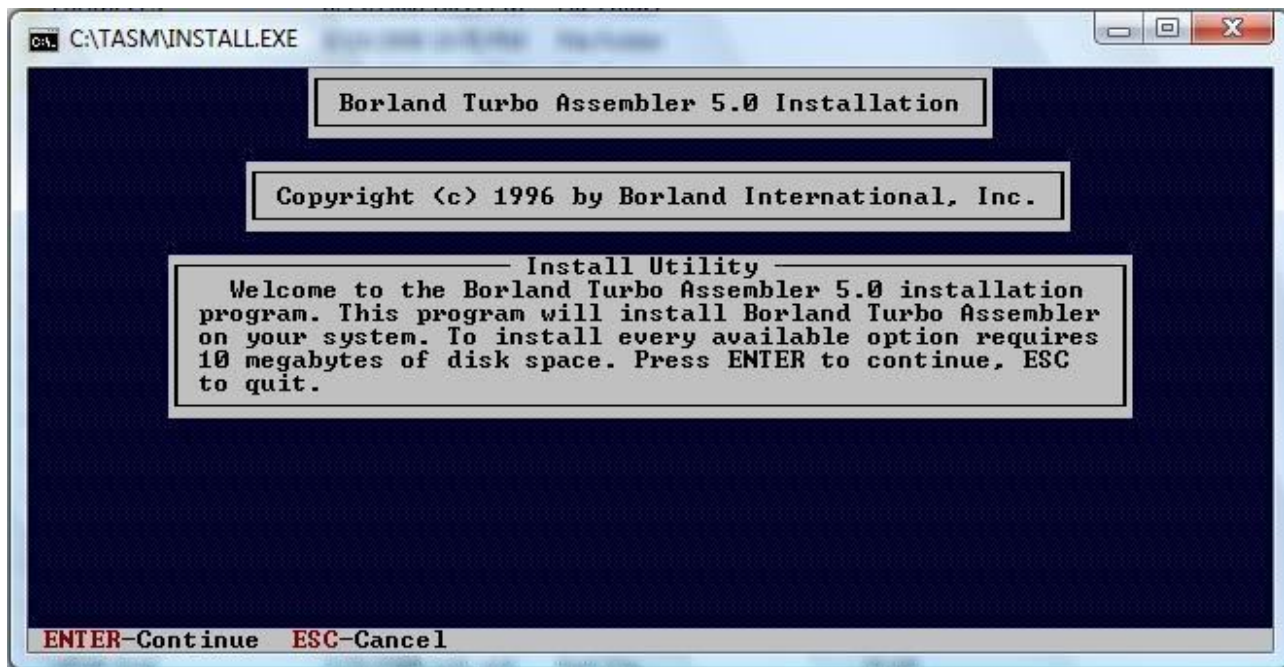
Cài đặt ASM.

Đầu tiên, tải file cài đặt bên dưới về. Sử dụng chương trình giải nén để giải nén. Khi giải nén khuyến cáo bạn giải nén vào thư mục gốc C hoặc D... (để khi cài đặt mọi việc được đơn giản). Sau khi giải nén, bạn vào thư mục vừa giải nén ra (ví dụ của tôi là C:\TASM). Click đúp chuột vào file INSTALL.EXE. Windows sẽ hiện ra cửa sổ cài đặt.

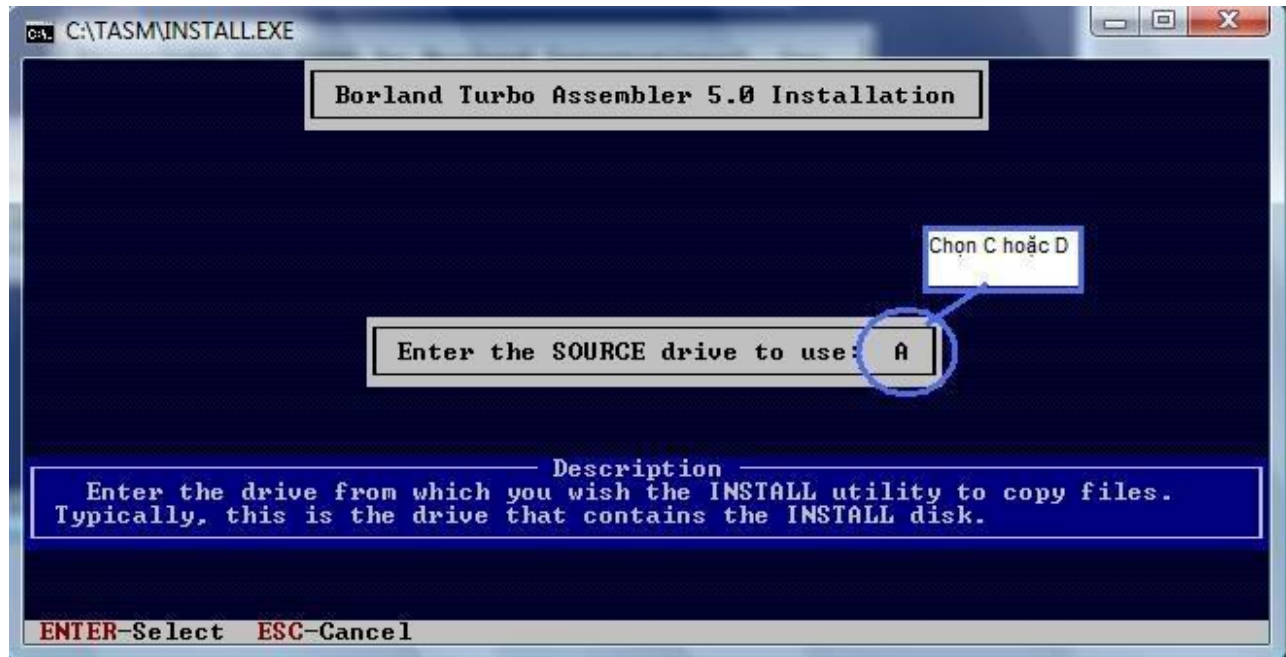
Bài 1 đầu tiên:

Cài đặt ASM.

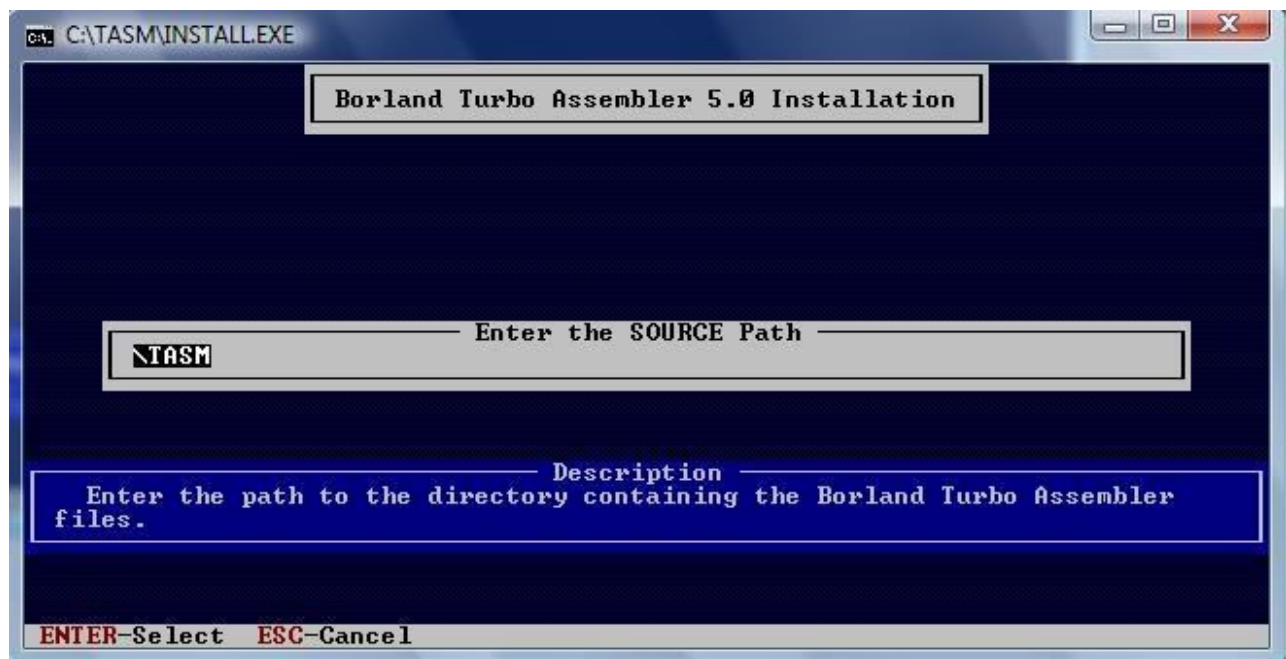
Đầu tiên, tải file cài đặt bên dưới về. Sử dụng chương trình giải nén để giải nén ([tải chương trình giải nén ở đây](#)). Khi giải nén khuyến cáo bạn giải nén vào thư mục gốc C hoặc D... (để khi cài đặt mọi việc được đơn giản). Sau khi giải nén, bạn vào thư mục vừa giải nén ra (ví dụ của tôi là C:\TASM). Click đúp chuột vào file INSTALL.EXE. Windows sẽ hiện ra cửa sổ cài đặt.



Bạn làm theo hình vẽ(gõ C hoặc D hay ổ đĩa bạn muốn cài đặt TASM) và ấn Enter



Tiếp tục ấn Enter



Ấn Enter đến lúc nào hoàn thành thì thôi :-)

Link download: <http://www.mediafire.com/?tjw322pbbwr>

Bài 2: Hợp ngữ - Assembly

Nội dung môn học:

Phần I: Ngôn ngữ ASM và cách lập trình.

Phần II: Liên kết ngôn ngữ bậc cao với ASM

Phần III: Lập trình hệ thống

Phần I

1. Mở đầu

- ASM là ngôn ngữ lập trình bậc thấp.

- Ưu điểm:

- * Viết chương trình chạy nhanh.

- * Chiếm ít vùng nhớ.

- Nhược điểm:

- * Khó viết.

- * Khó Debug

- * Non Portability (không có khả năng chạy trên nhiều nền hệ điều hành khác nhau).

- Ứng dụng:

- * Viết lõi (Kernel) của HĐH.

- * Viết Virus.

- * Viết trò chơi.

- * Các chương trình đo, điều khiển trong hệ thống máy móc công nghiệp.

2. Cài đặt chương trình

3. Các bước thực hiện một chương trình ASM trên PC

- * Sử dụng một chương trình soạn thảo bất kỳ (NC, C, NotePad...) và lưu lại dưới dạng tệp tin ASM.

- * Dịch từ tệp tin ASM sang tên tin OBJ.

- * Chuyển từ tệp tin OBJ sang tệp tin thực thi EXE hoặc COM.

- * Chạy chương trình.

4. Hỗ trợ của hệ thống cho lập trình ASM

***Nói qua về các thành phần của một hệ thống vi tính:**

Một hệ thống máy vi tính điển hình bao gồm: khối hệ thống (System unit), bàn phím (Keyboard), màn hình (Display Screen) và các ổ đĩa (Disk drives). Khối hệ thống được xem như là “máy tính” bởi nó tập trung các bảng vi mạch của máy vi tính. Bàn phím, màn hình, các ổ đĩa gọi là

các thiết bị vào ra (I/O devices) hay thiết bị ngoại vi (Peripheral devices).

Về mặt chức năng, vi mạch máy tính gồm 3 phần:

+ Đơn vị xử lý trung tâm (CPU).

+ Bộ nhớ.

+ Các mạch vào ra.

- Bộ nhớ: thông tin xử lý trong máy tính được lưu trữ trong bộ nhớ của nó, mỗi phần tử vi mạch nhớ có thể chứa một bit dữ liệu.

Các vi mạch được tổ chức thành các nhóm có thể chứa 8 bit dữ liệu

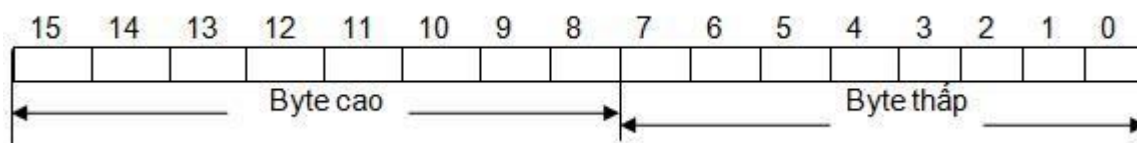
Mỗi chuỗi 8 bit này được gọi là 1 byte. Mỗi một byte được xác định bởi 1 con số được gọi là địa chỉ (address)

- Vị trí bit: Các vị trí được đánh số từ phải qua trái, bắt đầu từ 0. Trong một từ (word) 2 byte, các bit từ 0 đến 7 tạo thành byte thấp và các bit từ 8 đến 15 tạo nên byte cao. Để chứa một từ này trong bộ nhớ, byte thấp được ghi vào byte nhớ với địa chỉ thấp, còn byte cao ghi vào byte nhớ với địa chỉ cao hơn.

Vị trí các bit trong byte:



Vị trí các bit trong từ 2 byte:



Thao tác với bộ nhớ: Đọc nội dung của ô nhớ và ghi dữ liệu vào ô nhớ. Khi đọc, bộ VXL chỉ lấy ra bản sao của dữ liệu còn nội dung nguyên thủy của nó không thay đổi. Khi ghi dữ liệu, dữ liệu được viết vào sẽ trở thành nội dung mới của ô nhớ và dữ liệu nguyên thủy của nó bởi thế sẽ mất đi.

4.1 Các thanh ghi

Là vùng nhớ đặc biệt (dạng RAM) nằm trong CPU. Mỗi thanh ghi có tên hủý (kí hiệu) và việc thâm nhập vào chúng thông qua tên hủý.

Phân loại

Máy tính 16 bits: 14 thanh ghi.

Máy tính 32 bits: 16 thanh ghi.

Máy tính 16 bits:

Thanh ghi cờ 16 bits

				O	D	I	T	S	Z		A		P		C
--	--	--	--	---	---	---	---	---	---	--	---	--	---	--	---

C: Carry

P: Parity

A: Auxiliary

Z: Zero

S: Sign

T: Trap

I: Interrupt

D: Direction

O: Overflow

Lập trình ASM hay lấy trạng thái các bit cờ làm điều kiện cho các lệnh nhảy có điều kiện.

Thanh ghi đa năng 16 bits

AH	AL
BH	BL
CH	CL
DH	DL
SI	
DI	
BP	
SP	

Thanh ghi AX (accumulator register): là thanh ghi được sử dụng nhiều nhất trong các lệnh số học, logic và chuyển dữ liệu bởi chúng tạo ra mã máy ngắn nhất.

Thanh ghi BX (Base register): đồng thời đóng vai trò là thanh ghi địa chỉ.

Thanh ghi CX (Count register): Việc xây dựng các chương trình lặp được thực hiện dễ dàng bằng cách sử dụng thanh ghi CX, trong CX đóng vai trò là bộ đếm số vòng lặp.

Thanh ghi DX (Data register): được sử dụng trong các thao tác nhân và chia, nó cũng được sử dụng trong các thao tác vào ra.

Thanh ghi SP (Stack pointer): được sử dụng kết hợp với thanh ghi SS để truy cập đoạn ngăn xếp.

Thanh ghi DI (Destination Index): thực hiện các chức năng tương tự như thanh ghi SI. Có một số lệnh gọi là các thao tác chuỗi sử dụng DI để truy cập đến ô nhớ được định địa chỉ bởi đoạn SS.

Thanh ghi BP (Base pointer): được sử dụng để truy cập dữ liệu trong ngăn xếp. Khác với SP, BP cũng có thể sử dụng để truy cập dữ liệu trong các đoạn khác.

Thanh ghi SI (Source Index): được sử dụng để trỏ tới các ô nhớ trong đoạn dữ liệu được định địa chỉ bởi thanh ghi DS. Bằng cách tăng SI chúng ta có thể dễ dàng truy cập đến các ô nhớ liên tiếp.

Thanh ghi con trỏ lệnh IP|PC:

Cho biết phần địa chỉ offset của vùng nhớ RAM chứa mã máy của chương trình.

Thanh ghi phân đoạn 16 bits

A diagram showing four horizontal rectangular boxes stacked vertically, representing 16-bit segment registers. Each box contains a label on its right side. From top to bottom, the labels are CS, DS, ES, and SS. A small square icon with a cross is located to the left of the top box.

CS
DS
ES
SS

Cho biết phần địa chỉ segment của vùng nhớ RAM

CS: chứa mã máy.

DS, ES: chứa dữ liệu.

SS: chứa ngăn xếp

Với máy tính 32 bits: có 16 thanh ghi. Trong đó các thanh ghi cơ, đa năng, con trỏ lệnh là các thanh ghi 32 bits với các kí hiệu thêm chữ R phía trước.

Các thanh ghi segment vẫn là 16 bits, và có thêm 2 thanh ghi FS, GS để hỗ trợ cho dữ liệu.

Bài 3: Chương trình đầu tiên

Chương trình đầu tiên của chúng ta sẽ rất đơn giản, hiện lên một xâu ký tự quen thuộc cho những người lập trình chúng ta "Hello World!".

Dưới đây là đoạn code của chương trình:

Code:

```

.model small
.stack
.data
message db "Hello world!!!", "$"
.code
main proc
mov ax,seg message
mov ds,ax
lea dx,message
mov ah,09
int 21h
mov ax,4c00h
int 21h
main endp
end main

```

Sử dụng bất cứ trình soạn thảo văn bản nào bạn có và gõ đoạn code chương trình trên vào và lưu thành tệp tin FirstApp.asm

Tải tệp tin source: FirstApp.ASM

Sử dụng các tệp tin TASM.EXE và TLINK.EXE của Turbo Assembly để biên dịch chương trình thành FirstApp.exe

Nào, hãy đi sâu vào nghiên cứu đoạn code trên.

.model small(Chú ý có dấu chấm " . " đầu dòng): đây là lệnh điều khiển xác định mô hình bộ nhớ cho một chương trình ASM. Ở đây ta sử dụng small có nghĩa là chương trình không cần nhiều bộ nhớ lắm.

.stack: lệnh này thông báo cho Assembly biết rằng chúng ta bắt đầu sử dụng "stack" segment tại đây. Stack được dùng để lưu trữ các dữ liệu tạm thời, nó có thể không được sử dụng trong chương trình nhưng khi biên dịch chương trình cần phải dùng đến cấu trúc stack vì thế một lời khuyên là bạn nên khai báo Stack trong mọi trường hợp.

.data : đánh dấu điểm khởi đầu của vùng nhớ chứa số liệu (data segment). Chúng ta phải đặt việc khai báo các biến nhớ sẽ sử dụng tại đây.


.code : đánh dấu điểm khởi đầu của vùng nhớ chứa mã lệnh (code segment).

main proc : Các mã lệnh đều phải được đặt trong một thủ tục giống như ngôn ngữ C hay bất cứ ngôn ngữ lập trình nào khác. Dòng lệnh này cho biết một thủ tục proc được gọi tại đây. **main endp** cho biết thủ tục đã kết thúc. Một thủ tục BẮT BUỘC phải có khởi đầu và kết thúc. **end main** : cho biết chương trình đã kết thúc.

message db "xxxx" : DB (Define Byte) khai báo biến có độ dài 1 Byte. Byte này chứa thông tin ở giữa 2 dấu nháy kép (""). "Message" là tên biến.

mov ax, seg message : đưa giá trị của biến message vào thanh ghi AX.

mov ds,ax : đưa địa chỉ (phần segment) của biến nhớ vào thanh ghi DS

lea dx, message : đưa địa chỉ (phần Offset) của biến nhớ message vào thanh ghi DX. Như vậy chúng ta có địa chỉ của biến message sẽ là DS  X.

mov ah, 09

int 21h : ngắt 21h, đưa chuỗi ký tự ra màn hình.

mov ax, 4c00h :

int 21h : Trở về DOS.

Hi vọng bạn có thể hiểu qua về cấu trúc một chương trình ASM.

Assembly Fundamental ! Cơ bản về assembly

Bài viết này có tham khảo tài liệu Intel. Khi tìm hiểu về Assembly, chắc hẳn rất mệt mỏi. Tại sao có nhiều người nói asm khó. Thực sự nó không khó mà vì nó quá đơn giản. Vì nó quá đơn giản mà ta suy nghĩ cao siêu về nó nên ta không hiểu về nó. Cách đơn giản học asm là suy nghĩ đơn giản. -----Keep your mind simple !----- Muốn tìm hiểu asm trước hết cần phải tìm hiểu về con 80386.

-----Bắt đầu từ CPU 386-----CPU là thiết bị thực hiện hàng loạt những chuỗi lệnh. Những lệnh mà CPU thực hiện rất đơn giản. Những lệnh này cần thanh ghi làm toán hạng của nó. 386 có 8 thanh ghi đa dụng: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP. Tất cả thanh ghi đa dụng đều dài 32 bit

EAX, EBX, ECX, EDX có thể chia ra đoạn 16, và 8 bit chẳng hạn như :

31 7 0

+-----+

EAX + | AH | AL +

+-----+

AX

Trong thanh ghi EAX sẽ được chia thành hai đoạn :

Thanh ghi đoạn:

CS, SS, DS, ES, FS, GS dài 16 bit

Thanh ghi floating point:

8 thanh ghi từ ST0->ST7 dài 80 bit.

Ngoài ra các CPU sau này có thêm 8 thanh ghi mm0->mm7 dài 64 bit của MMX

và 8 thanh ghi xmm0 --> xmm7 dài 128 bit của SSE.

Những thanh ghi này chủ yếu dùng để move dữ liệu vòng quanh bộ vi xử lý.

Là toán hạng cho các instruction.

Các thanh ghi đa dụng giống như biến trong C nó có thể lưu số nguyên. Nó có thể lưu địa chỉ giống như con trỏ.

Các thanh ghi floating point dùng để thực hiện phép toán dấu chấm động.

Thanh ghi EFLAGS và EIP. Tác dụng của thanh ghi này sẽ bàn đến sau.

-----Chế độ thực (real mode)-----

Trong chế độ này, bộ nhớ bị giới hạn 1 MB bộ nhớ (2^{20} bytes) địa chỉ bộ nhớ từ 00000 --> FFFFF. Địa chỉ bộ nhớ này cần 20 bit mà 8086 không có thanh ghi nào dài đến 20 bit cả. Nó giải quyết bằng cách sử dụng hai thanh ghi để diễn đạt 20 bit. Thanh ghi thứ nhất gọi là selector, thanh ghi thứ 2 gọi là offset. Nên địa chỉ vật lý có thể xác định bằng :

$selector * 16 + offset$

Nhân 16 trong số hex khá dễ vì chỉ thêm số 0 bên phải selector. Giá trị selector thường nằm trong thanh ghi CS.

Một selector có thể đánh địa chỉ cho 64KB bộ nhớ. Điều gì sẽ xảy ra khi chương trình cần hơn 64KB bộ nhớ. Giá trị CS không thể sử dụng trong suốt chương trình. Chương trình trong bộ nhớ bị cắt ra nhiều đoạn (segment) nhỏ hơn 64KB. Khi chương trình chạy giá trị CS sẽ thay đổi. Điều này có thể gây khó chịu.

Mỗi byte trong bộ nhớ thực sự không có một địa chỉ duy nhất. Ví dụ địa chỉ vật lý 04808 có ba cách diễn đạt 047C:0048, 047E:0028, 047B:0058. Điều này gây ra sự phức tạp khi so sánh địa chỉ.

-----Chế độ bảo vệ 16bit 286(protect mode)-----

Trong chế độ bảo vệ 286, những thanh ghi đoạn không dùng để định địa chỉ nữa như real mode nữa. Trong real mode nó dùng làm định địa chỉ vật lý theo đoạn. Trong protect mode, nó là con trỏ trỏ đến descriptor table. Ở cả hai chế độ, chương trình vẫn bị chia thành nhiều đoạn. Tuy nhiên

trong protect mode những đoạn này không cố định trong bộ nhớ vật lý .Thực tế nó không phải lúc nào cũng ở trong bộ nhớ .Nó có cơ chế bộ nhớ ảo.Nghĩa là giữ một phần bộ nhớ của chương trình được lưu trên đĩa cứng ,phần còn lại trong bộ nhớ RAM để chạy chương trình. trong protect mode,mỗi đoạn có một descriptor chứa những thông tin về chính nó.Những thông tin này gồm :nó hiện diện trong bộ nhớ không,access permission,địa chỉ vật lý của nó.Điều làm cho các chương trình không thể ghi đè lên nhau trong bộ nhớ được và nó bảo vệ vùng nhớ hđh.Điều này rất khác so với CPU đời trước đó.

Nhưng nó vẫn còn nhược điểm rất lớn.Offset của nó vẫn là 16 bit

-----32 bit protect mode-----

Có hai điểm khác biệt lớn giữa 386 32bit protect mode và 286 16 bit protect mode:

- 1.Offset mở rộng ra 32 bit.Vì thế một đoạn có thể mở rộng lên 4GB
2. Đoạn có thể chia thành 4K đơn vị gọi là page.Bộ nhớ ảo hoạt động với trang thay vì đoạn.Điều có nghĩa một phần của đoạn trong bộ nhớ có thể nằm trong ổ cứng.Trong 286 ,toàn bộ đoạn phải nằm trong bộ nhớ hoặc trên ổ cứng khi paging.Mà điều này thì rất vô lý trong 386 protect mode.

Hiện nay đa số các hệ điều hành điều hành chạy trong protect mode .Linux và Window đều chạy ở protect mode .

-----Ngôn ngữ Assembly-----

Chương trình Assembly(Asm) thì lưu ở dạng text cao hơn ngôn ngữ máy.Mỗi chỉ lệnh (instruction) là một chỉ lệnh của máy(machine instruction). Ví dụ :

add eax,ebx

chỉ lệnh này có opcode là 03 ,clocks = 2

Từ "add" là một mnemonic cho chỉ lệnh của máy. Câu lệnh asm có cú pháp chung chung như sau:

mnemonic toán_hạn_1,toán_hạn_2

có thể có chỉ lệnh 1-2-3 toán hạn hoặc không có toán hạn nào hết.

Trình biên dịch sẽ convert mã text asm --> mã máy (machine code).Những ngôn ngữ lập trình bậc cao sẽ phức tạp hơn khi biên dịch vì mỗi lệnh không tương đương với một chỉ lệnh mã máy.

Toán hạn có thể là thanh ghi (register) , địa chỉ ô nhớ(memory), giá trị trực tiếp(immediate) và giá trị bao hàm trong chỉ lệnh.

Ví dụ : inc eax

Toán hạng là thanh ghi eax,giá trị bao hàm trong chỉ lệnh là 1 vì chỉ lệnh này sẽ cộng 1 vào eax.

Đối với Intel syntax :

câu lệnh mov sẽ có cú pháp : mov dest , src

dữ liệu trong src sẽ copy vào dest .Sơ đồ : src-->dest

Chương trình ASM đơn giản. Chương trình asm này thực hiện việc cộng hai số.

-----simple.asm-----

%include "asm_io.inc"

segment .data

I1 db "Số thu nhất : ",0

I2 db "Số thu hai : ",0

I3 db "Tổng hai số là : ",0

segment .bss

temp resd 1

```
segment .text
global _asm_main
asm_main:
enter 0,0
pusha
mov eax, l1
call print_string
call read_int
mov [temp], eax
mov eax, l2
call print_string
call read_int
add eax,[temp]
mov ebx,eax
mov eax, l3
call print_string
mov eax,ebx
call print_int
popa
leave
ret
-----end-----
```

biên dịch nasm -f coff simple.asm