حالت حفاظت شده

قسمت اول

این مقاله در دو قسمت بوده. هدف از ترجمه این مقاله آشنایی ابتدایی با مبحث Protected Mode میباشد.

در این آموزش فرض بر این است که شما برنامه نویسی در حالت واقعی را میدانید.
این آموزش با بحث در مورد حافظه در حالت حفاظت شده شروع میشود و بعد از آن در رابطه با دستورالعمل های اسمبلی بحث میکند و بعد از آن دررابطه با DPMI.
دو حالت حفاظت شده وجود دارد flat pmode و segmented pmode.

در این آموزش در مورد حالت حفاظت شده بصورت کلی بحث میشود این آموزش یک آموزش سریع در مورد حالت حفاظت شده میباشد.

در حالت واقعی همانطور که میدانید حافظه بصورت ۱۶ بیتی در دسترس میباشد یعنی در قطعات ۱۶ بیتی و افستهای ۱۶ بیتی این حالت به شما فقط اجازه دسترسی به ۱ مگابایت از حافظه را میدهد(البته بدون استفاده از نرم افزارهای XMS/EMS). در حالت محافظت شده شما میتوانید به ۴ گیگابایت حافظه دسترسی پیدا کنید.
برای بدست آوردن آدرس حافظه ای در حالت حفاظت شده از حالت واقعی از فرمول زیر میتوان استفاده کرد:

PmodeAddress = SEG * 16 + Offset

مثال:
فرض کنیم که شما از DOS4GW استفاده میکنید(نرم افزاری که به شما اجازه میدهد فقط به تمامی RAM موجود در سیستم از طریق DOS دسترسی پیدا کنید و دیگر خاصیتهای حالت حفاظت شده را دارا نمیباشد.)
و میخواهید یک فونت ۸x۸ را از حافظه بخوانید. فونت در حالت واقعی در آدرس F000:FA6E میباشد.برای اینکه آدرس را در حالت حفاظت شده بدست بیاوریم داریم:

F000 * 16 + FA6E = FFA6Eh

یک مثال دیگر:

Video Segment A000:0000
Pmode Address A0000

شما میتوانید ۴بایت به چپ شیفت کنید(کمی به جوابها توجه کنید حتماً متوجه میشوید!).
اسمبلی در حالت حفاظت شده کمی متفاوت میباشد در حالت محافظت شده دستورات بیشتر و چیزهای بیشتری در اختیار اسمبلی میباشد.ما زیاد وارد جزیئات نمیشویم و فقط در رابطه با مدل جدید برنامه نویسی بحث خواهیم کرد.
قدم اول شما نمیتوانید ثباتهای قطعه(segement register) را دستکاری کنید:

ES , DS , CS , FS , GS

برای دسترسی به حافظه در قطعه برنامه تان داریم:

```
MOV EDI,0A0000h
MOV [EDI], AL
```

اگر شما از داس استفاده میکنید تمامی ثباتهای شما اکثر اوقات به یک قطعه اشاره میکنند(ES=CS=DS) پس دیگر شما نباید نگران مقدار دهی آنها باشید . اگر شما در حال طراحی سیستم عامل هستید باید جزئیات بیشتری را در نظر بگیرید.

DPMI(Dos Protected Mode Interrupt)

نرم افزارهای جانبی مانند DOS4GW از DPMI استفاده میکنند.
شما نمیتوانید وقفه های ۱۶ بیتی که حاوی کدهای ۱۶ بیتی میباشند را بکار بگیرید شما باید از ساختار ثبات (structure register) استفاده کرده و در ضمن حافظه سطح پائین داس را به آن تخصیص دهید.
یک وقفه که حافظه را به عنوان ورودی و یا برای برگشت خروجی خود اشغال میکند باید بوسیله وقفه 31h صدا زده شود.بعضی از وقفه ها را شما باید بصورت مستقیم صدا بزنید مانند وقفه هایی که از حافظه برای مقاصد خود استفاده نمیکنند.
مثال:

```
MOV AX,13h
INT 10h
```

ست کردن مود ۱۳ برای وقفه 10h میتواند مانند بالا استفاده شود.

ساختار در C :

```
/* Real Mode Registers Structure */
typedef struct rmode_type {
  long EDI;
  long ESI;
  long EBP;
  long reserved;
  long EBX;
  long EDX;
  long ECX;
  long EAX;
  short flags;
  short ES, DS, FS, GS, IP, CS, SP, SS;
} RMI, *RMIPtr;
Function 1 Subfuntion 0h
  Allcoate Low Dos Memory.
```

این کد قطعه ای را به حافظه تخصیص داده شده برمیگرداند.
افست حافظه همیشه عدد 0 خواهد بود.

```
  MOV EBX, Size Of Memory To Be Allocated
  MOV EAX, 100h   ; Allocate Dos Memory
  ADD EBX, 15     ; Add 15 To The Size
  SHR EBX, 4      ; & Shift Over By 4 (Memory Can only be alloced in 16 Byte
                  ; Pages. Thus if you wanted 1 byte of memory, you would
                  ; actually allocated 16 bytes
                  ; 1 + 15 = 10h >>4 = 1 page (16 Bytes).
  INT 31h         ; Call Interrupt 31h
  JNC SHORT Done  ; No Carry, then Allocated
  XOR AX, AX      ; No Memory Free (Set AX = 0)
```

اگر ثبات AX برابر با عدد 0 شود این بدین معنا میباشد که حافظه ای تخصیص داده نشده است.
ثبات DX انتخابگر(selector) برای حافظه (بعداً در رابطه با انتخابگر توضیح داده خواهد شد) انتخابگر را ذخیره کرده چرا که هنگامی که حافظه را از تخصیص دربیاوریم به آن احتیاج خواهیم داشت.

برای ذخیره یک شی در حافظه در زبان C شما باید یک struct از نوع آن شی داشته باشید.برای مثال شی ما در اینجا Thing نام دارد.
(توجه داشته باشید که زبان C شی گرا نبوده و در اینجا منظور از شی همان متغیرها معمول برنامه نویسی میباشد).

```
struct Thing *ThingObject;
short Selector,Segment;

Segment = AllocateDosMemory(&Selector,sizeof(Thing));
ThingObject = Segment <<4;
```

حال ما قطعه را ۴ خانه شیفت داده و مقدار آن را در ThingObject جایگزین کردیم و به یاد داریم Seg*16 + Offset و افست شما 0 خواهد شد.

هنگامی که حافظه را از تخصیص درمی‌آوریم باید انتخابگر را ذخیره کنیم.
خب حال این اطلاعات را در ساختار RMI مینویسیم:
فرض کنیم ThingObject ساختار اطلاعات VESA (نوعی حالت گرافیکی و همچین اسلات های موجود بر روی مادربورد که امروزه استفاده نمیشود) میباشد و شما میخواهید تابع 4Fh زیر تابع 1 از وقفه 10h را صدا بزنید:

```
RMI X;

 X.EAX = 0x4F01;  // Set the Function/Subfunction
 X.ECX = 0x101;   // CX = SVGA Mode
 X.EDI = 0;       // ES:DI Points to VESA structure
 X.ES = Segment;
```

بصورت بالا ساختار را مقدار دهی کرده

```
MOV EDI, Address of X
 MOV EBX, 10h     ; Real Mode Interrupt To Call

 PUSH ES          ; Save ES
 MOV EAX, 300h    ; Function 3h SubFunction 0 - Call Real Mode INT
 XOR ECX, ECX
 INT 31h
 JNC SHORT NoError ; Carry Flag Set? There is Error.
 XOR EAX, EAX     ; Error, Set EAX = 0
 JMP SHORT Done
 NoError: MOV EAX, 1 ; No Error, set EAX != 0
 Done:
 POP ES

 If EAX = 0, Then there was an error.
```

خب حال ساختار ما اطلاعات وقفه ای که به آن معرفی کردیم را درون خود نگه میدارد.

برای خالی کردن حافظه شما باید از انتخابگر استفاده کنید:

```
 MOV DX, Selector   ; The Selector From the allocation
 MOV EAX, 101h      ; Function 1h Sub Function 1
 INT 31h            ; Cal DPMI
 JNC SHORT NoError  ; If Carry Set, Then Error
 XOR EAX, EAX
 JMP SHORT Done
 NoError: MOV EAX, 1
 Done:

 If EAX = 0, Then Error Deallocating Memory.
```

به خاطر داشته باشید کد بالا فقط حافظه پایین داس را تخصیص میدهد.هدف از تخصیص از حافظـه پـایین داس بـدین دلیـل میباشد که اگر ما بخواهیم از وقفه های داس در حالت واقعی استفاده کنیم باید از حافظه پایین داس استفاده کنیم.

انتخابگر یک عدد میباشد که برای ما مشخص میکند که کدام مدخل از LDT و یا GDT را انتخاب کرده ایم.
LDTوGDT اطلاعاتی درباره حافظه تخصیص شده را نگه داری میکنند(برای مثال پایه و اندازه).
به همین دلیل هنگامی که ما میخواهیم حافظه را از تخصیص دربیاوریم باید حتماً انتخابگر را ذخیره کنیم.

LDT سرنام Local Descriptor Table
GDT سرنام Global Descriptor Table
IDT سرنام Interrupt Descriptor Table

این جداول اطلاعاتی دررابطه با حافظه را درخود نگه میدارند.
GDT در تمامی برنامه ها سراسری و قابل استفاده میباشد. این جدول برای سیستم عامل میباشد.
LDT هر برنامه یک نسخه از جدول خود را دارا میباشد.
IDT برای وقفه ها بوده و در حافظه میباشد.

این جداول مانند IVT(Interrupt Vector Table) در داس عمل میکنند ولی با این تفاوت که آنهـا قابلیـت ایـن را دارا میباشند که بتوانند از تخصیص درآمده و دوباره حافظه ای به آنها تخصیص داده شود.

این آموزش فقط در سطح پایه بوده برای اطلاعات بیتشر میتوانید از کتاب زیر استفاده کنید:

Protected Mode Software Artitecture
By Addison-Wesley Publishing Company
(Part of the PC Architecture Series)
ISBN - 0-201-55447-X

خب اینهم کد وارد شدن به حالت حفاظت شده بطور سریع و مستقیم!!!

```
MOV EAX, CR0  ; Get Control Register 0
OR AL, 1      ; Set Pmode Bit
MOV CR0, EAX  ; Restore Control Register 0
```

ولی قبل از اجرای این کد حتماً IDT,LDT,GDT را تنظیم کرده و نوع CPU را حتماً تشخیص دهید.

پایان قسمت اول

Translated by NETSPC
Persian OS group(_LOVE_CODER_,MASTER,NETSPC)
Please contact us at

os@persiasecure.com
http://groups.google.com/group/Persian_OS
http://www.persiasecure.com/OS

Released in 2006 April