MEMÓRIA ESTENDIDA

PARTE II

> "No artigo de hoje e no próximo vamos tentar explicar um pouco aos leitores interessados nestas matérias, mas ainda desconhecedores, os príncipios de funcionamento dos processadores Intel de 32 bits em modo protegido."

No número anterior da nossa revista iniciámos esta nova série de artigos versando a temática da **memória estendida** e do **modo protegido de 32 bits**.

E tivemos nessa altura oportunidade de apresentar um método de acesso a dados da **memória estendida**, designado normalmente por método INT 15H, ou mais correctamente por serviço do BIOS *Protected-Mode Data Move*. E porque este serviço é providenciado pelo BIOS de todos os computadores de modelos compatíveis com os IBM AT e PS/2 ele está, por conseguinte, implicitamente ao inteiro dispor de todos nós.

Esse método contudo não é de utilização muito fácil para efeitos da transferência de dados, o que até poderia ser um mal menor, não fosse o facto de apresentar também vários outros inconvenientes, como se terão apercebido os leitores que analizaram o código fonte do programa EXTVIEW.EXE que ilustrava o artigo. Apesar disso é um método utilizado ainda por muitos programas nos nossos dias. Tudo dependerá por conseguinte do objectivo que se pretenda atingir.

No artigo de hoje e no próximo vamos tentar explicar um pouco aos leitores interessados nestas matérias, mas ainda desconhecedores, os príncipios de funcionamento dos processadores Intel de 32 bits em **modo protegido**, modo esse que é, como se sabe, o único modo de funcionamento do processador em que é possível o acesso a dados e a execução de programas acima do primeiro Megabyte de memória (mais correcto será afirmar, como estarão lembrados, 64 KB menos 16 bytes acima do primeiro Megabyte).

Como este assunto não é assim muito trivial, alguns pequenos conhecimentos de linguagem *Assembly*, embora não absolutamente essenciais, ajudarão um pouco na compreensão geral do conteúdo do texto que se seguirá.

Mas a compreensão do funcionamento e análise do pequeno programa que preparámos para ilustrar o tema do artigo e que consta da disquete **Spooler** desta edição, o PROTECT.EXE, exigirá contudo um conhecimento razoável de linguagem *Assembly* (e também algumas luzes de linguagem C já que o prólogo e o epílogo foram construídos nessa linguagem por razões de produtividade).

O PROTECT.EXE é em si mesmo um programa sem utilidade prática e que não faz nada de especial, excepto enviar duas mensagens do lado de lá (isto é do segundo Megabyte) informando-nos o que está a fazer e onde. Contudo, o estudo do seu funcionamento a partir do código fonte ajudará bastante, segundo cremos, a solidificar algumas conceitos base que aqui iremos expor.

MEMÓRIA SEGMENTADA

Em **modo protegido** a palavra **segmentação** não tem o mesmo sentido que se dá a esse termo em **modo real**. Em **modo real**, como se sabe, a memória acedível por um programa em dado momento é dada por um «dueto» que consiste no valor existente num registo de segmento e por um deslocamento (ou *offset*) relativo ao valor existente no mesmo registo de segmento.

Os registos de segmento percorrem a memória a «passadas» de 16 bytes (designado por um parágrafo) de cada vez e o deslocamento por sua vez é limitado a um número máximo de 16 bits, isto é pode ir de 0 até 65535 (ou 0FFFFh em hexadecimal).

Considere-se por exemplo a instrução Assembly: MOV AL, ES:[1234h]

Após a sua execução, o registo AL do processador ficará carregado com o valor do byte existente na localização de memória identificada pelo endereço 1234h dentro do segmento ES.

Se nesse momento o segmento ES contivesse o valor 2000h, o **endereço físico** (endereço absoluto a partir do ínicio da memória) do byte na memória seria:

2000h x 10h + 1234h = 21234h

Mas os registos de segmento são também registos de 16 bits, portanto o máximo endereço físico de memória que é possível aceder em **modo real** é, seguindo o mesmo raciocínio:

0FFFFh x 10h + 0FFFFh = 10FFEFh
(ou seja 1114095 em decimal)

Essa posição de memória corresponde precisamente a 64 KB menos 16 bytes acima do primeiro Megabyte. Esta é a grande limitação do funcionamento do processador em **modo real**.

Em **modo protegido** o processador faz a gestão do endereçamento à memória de modo significativamente diferente.

Os programas não endereçam directamente a memória, mas sim um modelo designado por **memória virtual** (não confundir com o modo de trabalho designado por virtual 86). Cabe a dois mecanismos internos ao processador desfazerem a ambiguidade. Um deles, designado por **mecanismo de segmentação**, possibilita a existência de múltiplos espaços de endereçamento independentes; o outro designado por **mecanismo de *paging*** permite a existência de um grande espaço de endereçamento em memória dispondo-se de relativamente pouca memória RAM mas em compensação suficiente espaço em disco. Cada um dos mecanismos isoladamente, ou ambos simultâneamente, podem ser activados por certos programas (normalmente pertencendo ao sistema operativo ou então que se substituem ao sistema operativo nessas missões).

Um endereço emitido dentro de um programa é um **endereço lógico** e compete, em primeira instância, ao mecanismo de segmentação a oportunidade de traduzir

esse endereço lógico num endereço absoluto designado por **endereço linear**.

Se o mecanismo de *paging* não estiver activo, o endereço linear corresponde de imediato ao endereço físico. Se estiver activo, será o mecanismo de *paging* que determinará a partir do endereço linear qual é o endereço físico.

O mecanismo de segmentação permite gerir segmentos de qualquer tamanho, deste 1 byte até 4 Gigabytes de tamanho e não, tal como acontecia em modo real, apenas segmentos de 64 KB.

Dentro de um dado programa é normal criarem-se vários segmentos independentes com características muito bem definidas. Haverá segmentos reservados ao código executável do programa, haverá outros reservados aos seus dados e um ou mais às suas pilhas (ou *stacks*).

Dentro de cada segmento e um pouco à semelhança do que aconteceria em modo real, o endereço linear é determinado pelo deslocamento dentro de um segmento, mas com uma diferença muito importante: os registos de segmento não contém valores em parágrafos correspondentes directamente a um certo endereço físico de memória, tal como aconteceria em modo real, mas sim valores designados por **selectores** (veja por favor a figura seguinte).

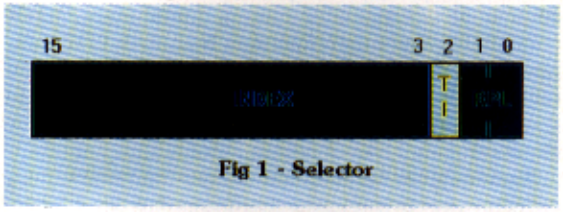

Fig 1 - Selector

Os selectores contêm índices (cujo valor é dado pelos bits 3 a 15) relativos a tabelas em RAM, designadas por *Descriptor Tables*. As *Descriptor Tables* contêm *Segment Descriptors* ou simplesmente *Descriptors* (que serão detalhados mais à frente).

Um programa terá sempre de efectuar a sua escolha entre duas alternativas de *Descriptor Tables*, a qual é formalizada através do bit 2 do selector (designado por *Table Indicator* ou **TI**).

Há que optar entre a *Global Descriptor Table* ou **GDT** (seleccionada com TI=0) e a *Local Descriptor Table* ou **LDT** (seleccionada com TI=1).

No sistema está disponível uma única GDT para todos os programas, e uma LDT para cada programa em execução. Contudo, podem ser concebidos sistemas operativos em que todos os programas compartilhem uma única LDT (o Windows 3.xx comportando-se como uma extensão ao sistema operativo MS-DOS actua desse modo). Pode também ser concebido um sistema sem LDT, em que todos os programas utilizem apenas a GDT.

O selector contém ainda um campo constituído pelos bits 0 e 1 e que é designado por *Request Privilege Level* ou **RPL**.

E antes de continuarmos vamos analisar um pouco o que se entende por «privilégios», pois daí deriva precisamente um pouco da explicação (mas não toda) do nome «modo protegido».

O processador dispõe de um dispositivo de protecção que reconhece 4 níveis (ou *rings*) de privilégio numerados de 0 a 3. Quanto maior o número, menor o nível de privilégio.

Uma «General Protection Exception» (*Exceptions* ou «Excepções» como se queira, serão tratadas no próximo número) é gerada sempre que um programa tente aceder a um segmento utilizando um nível de privilégio inferior ao que se aplica a esse segmento. O controle do nível de privilégio é efectuado por três estruturas:

1) Os bits 0 e 1 dos segmento CS (ou SS). Esse valor indica ao processador o *Current Privilege Level* ou **CPL**. O CPL contém o nível de privilégio do programa em execução. Normalmente o CPL é igual ao nível de privilégio do segmento de código (isto é, o segmento em que estão a ser lidas as instruções do programa). Se o programa transferir controlo para um segmento de código menos priviligiado, o CPL é ajustado para esse nível. O programa normalmente não pode transferir controle para segmentos de código mais priviligiados, a menos que esses segmentos sejam do tipo ***conforming***. Um segmento de código *conforming* é executado com o nível de privilégio da rotina que o chamou, mas aqui o CPL não será alterado (nos dois primeiros bits do registo CS), mesmo que explicitamente se altere o RPL na execução de instruções JMP FAR ou CALL FAR.
2) Os bits 45 e 46 do *Segment Descriptor* contêm um campo designado por *Descriptor Privilege Level* ou **DPL**, o qual representa o nível de privilégio do segmento.
3) Finalmente, o RPL que vimos acima e que é coincidente com o CPL nos segmentos de código e pilha. Se o RPL para um selector de dados for menos priviligiado (valor numérico maior) que o CPL ,o acesso à memória é efectuado ao nível de privilégio do segmento de dados. Por conseguinte, um programa só pode aceder a um segmento se o DPL do segmento for o mesmo ou menos priviligiado que o menos priviligiado dos CPL e RPL.

Vimos atrás que os selectores indexam *Descriptors* em *Descriptor Tables*.

Pela Figura 2, vemos que cada *Descriptor* consta de 64 bits divididos por vários campos, os quais dizem não só da localização e tamanho dos segmentos, mas também contêm vária informação de controlo e *status*. Vamos ver esses campos um a um, pedindo desde já as nossas mais sentidas desculpas pelo amplo uso que iremos fazer (e que fizemos também até aqui) dos termos originais em inglês, mas efectivamente só iríamos aumentar a confusão presente (e especialmente futura) do leitor se nos atrevêssemos a inventar traduções para termos já internacionalmente bem firmados.

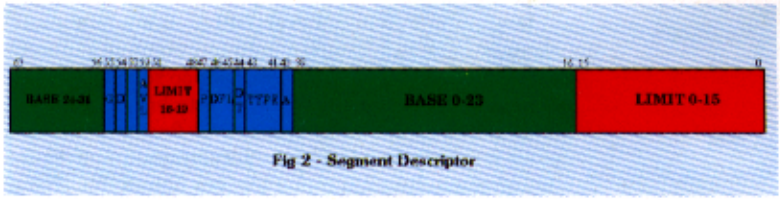

Fig 2 - Segment Descriptor

LIMIT – Define o tamanho do segmento. O processador concatena os dois campos de Limite e forma um valor de 20 bits. Esse valor pode ser interpretado de 2 modos:

- Se o bit de Granularidade G (bit 55) for 0 (*Byte Granularity*) o segmento terá um tamanho compreendido entre 1 byte e 1 megabyte (2^{20} bytes).
- Se o bit de Granularidade for 1 (*Page Granularity*) o segmento terá um valor compreendido entre 4 KB e 4 Gigabytes. Por conseguinte, aqui o incremento processa-se 4 KB (designado por uma página) de cada vez..

Quase sempre um endereço lógico terá de ter um deslocamento compreendido entre 0 e o valor do Limite. Programas que gerem deslocamentos fora do limite dão origem a *Exceptions*.

Podem contudo definir-se segmentos do tipo *Expand-Down* (ver mais à frente) em que, pelo contrário, os endereçamentos podem ser feitos com qualquer deslocamento excepto valores compreendidos entre 0 e o Limite. Este tipo de segmento é algumas vezes utilizado para *stacks* que contemplem a possibilidade de ser expandidos (para baixo claro).

Nota: A razão pela qual o Limite é formado dentro do *Descriptor* por dois campos em posições diferentes tem a ver com a necessidade de se manter compatibilidade com os processadores 80286. Até ao bit 47 os *Descriptors* são idênticos tanto nos 80286 como nos modelos de 32 bits, mas os bits 48 a 63 são «reservados» nos 80286.

BASE – Define a localização do segmento no espaço de endereçamento de 32 bits (4 Gigabytes). O processador concatena os três campos que definem a Base para formar o valor requerido de 32 bits.

ACCESSED (A) – Se este bit for 0 isso significará que o selector para o *Descriptor* não foi transferido para um registo de segmento. Se for 1 acontece o contrário. Os *Descriptors* inicialmente referem todos os segmentos como *Accessed*, mas se num certo momento se colocarem todos a 0, então será possível a partir daí controlar os segmentos que foram acedidos.

Existe uma certa analogia com o conhecido *Archive* bit dos ficheiros.

TYPE – Este campo define as operações permitidas no segmento. Se se tratar de um segmento de memória a interpretação a dar é a seguinte:

000 – Segmento de Dados, *Read-Only, Expand-Up*
001 – Segmento de Dados, *Read/Write, Expand-Up*
010 – Segmento de Dados, *Read-Only, Expand-Down*
011 – Segmento de Dados, *Read/Write, Expand-Down*
100 – Segmento de Código, *Execute-Only, Nonconforming*
101 – Segmento de Código, *Execute-Read, Nonconforming*
110 – Segmento de Código, *Execute-Only, Conforming*
111 – Segmento de Código, *Execute-Read, Conforming*

Nota: Se não se tratar de um segmento de memória (e se trata de um segmento de sistema ou *task gate*) a interpretação será algo diferente mas inibimo-nos de a apresentar apenas por ultrapassar o âmbito introdutório deste artigo.

Descriptor Type (DT) – Os *Descriptors* de segmentos de memória recebem 1 neste campo. Os *Descriptors* de segmentos de sistema e *gates* recebem 0. Ver por favor a Nota anterior.

Descriptor Privilege Level (DPL) – Já nos referimos a ele mais atrás. Este campo é utilizado, como se explicou, para o processador controlar o acesso ao segmento.

Segment Present Bit (P) – Se este bit estiver a 0 o processador gerará uma *Segment-Not-Present Exception* se um selector para o *Descriptor* for carregado num registo de segmento. Essa *Exception* tem por objectivo alertar o sistema operativo sobre o acesso a segmentos indisponíveis (que podem por exemplo ter sido guardados no disco rígido). Como resultado dessa *Exception* o sistema operativo tem então a oportunidade de providenciar para que esse segmento fique disponível de novo (por exemplo recarregando-o do disco para a memória) ao programa de aplicações e de um modo totalmente transparente para este. Vê-se assim a importância deste bit na gestão da memória virtual.

Available Bit (AVL) – Como a tradução indica «Disponível» este bit pode ser utilizado pelo *software* do sistema para qualquer tipo de controle que entenda ser conveniente.

Bit 53 – Este campo é reservado pela Intel para futuros microprocessadores. Deverá ser mantido a zero, até se saber mais pormenores dessas intenções.

Default Bit (D) – Este bit determina qual é por omissão o tamanho do operando e do endereço nas instruções executadas. Se o segmento de programa for de 32 bits, o valor deste bit deve ser 1. Para os segmentos de dados e pilha este bit recebe o nome de «Big Bit» (B).

Granularity Bit (G) – Já foi referido atrás quando tratamento do campo de Limite.

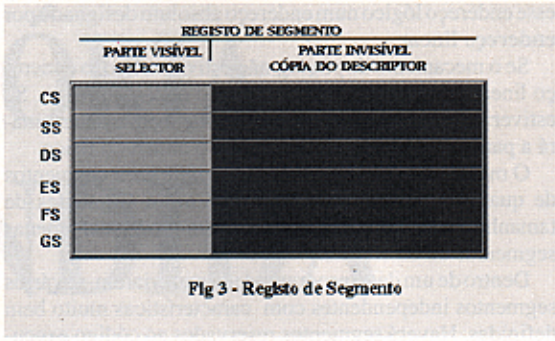

Fig 3 - Registo de Segmento

Os registos de segmento são constituídos por uma parte visível e acessível por *software* e uma parte invisível (Fig 3). Quando um selector é carregado num registo de segmento, o processador carregará automáticamente a parte invisível do mesmo registo de segmento com a informação constante do *Descriptor*. Este pormenor é importante, pois significa que o processador a partir daí procederá exactamente em conformidade com o conteúdo do *Descriptor* e sem qualquer perda de desempenho.

Resta-nos referir ainda que a Base e Limite da GDT constam de um registo do processador de nome GDTR (*Global Descriptor Table Register*). No caso de LDT, a sua Base, Limite e selector da GDT que contém o *Descriptor* do segmento onde se encontra a LDT constam de um registo do processador de nome LDTR (*Local Descriptor Table Register*).

Para visualizar melhor como toda esta engrenagem joga, os leitores que pela primeira vez lêem sobre este assunto, deverão investir algum tempo na análise do conteúdo da Figura 4.

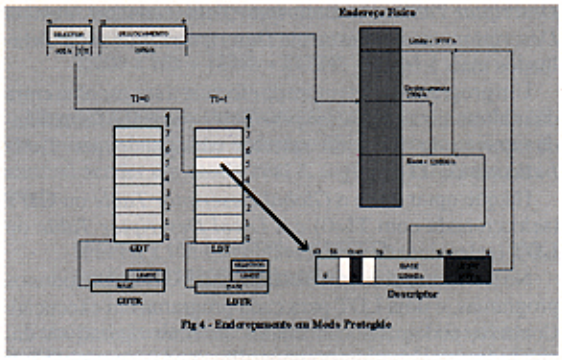

Fig 4 - Endereçamento em Modo Protegido

MODELO «FLAT»

O modelo «FLAT» (ou de Base 0) de memória é um caso particular do modelo de memória segmentada, mas em que (parece contraditório mas não é) pura e simplesmente a segmentação desapareceu. Esse efeito consegue-se mapeando todos os segmentos para o mesmo endereço físico de memória. A Base dos segmentos será 0 e o Limite 4 Gigabytes. Este modelo é interessante para sistemas operativos como o Unix que não suportam segmentação mas podem suportar *paging*.

O modelo «FLAT» pode também ser «protegido» se os Limites forem estabelecidos para se fixarem apenas em zonas para as quais existem efectivamente endereços físicos. Esta situação designa-se por modelo *Protected FLAT*.

E no próximo número cá estaremos de novo para completar a introdução a esta interessante mas indiscutívelmente delicada matéria. E naturalmente contamos aqui com os nossos leitores, em particular o núcleo duro dos que não desistem à primeira.

José Páscoa