# MEMÓRIA ESTENDIDA

## PARTE V

> **O termo Virtual 86 (ou mais precisamente Virtual 8086) tem a sua justificação etimológica no facto de ser possível uma coexistência pacífica e em simultâneo de vários ambientes 8086 no mesmo CPU.**

Tal como ficou prometido – e claro que para nós o prometido é devido – vamos dissertar um pouco neste número sobre o Modo Virtual 86. Para apoio ao nosso discurso, incluímos na disqueta **Spooler** um programa que dá pelo nome de VIRT86.EXE (bem como o respectivo código fonte), o qual aplica amplamente toda a teoria do Modo Virtual 86. Como o Modo Virtual 86 funciona como uma Tarefa de Modo Protegido, podemos desde já avançar que o programa VIRT86.EXE ilustra também essa matéria no âmbito de um sistema utilizando o Mecanismo de «Paging». Ambos os tópicos foram, como estarão lembrados, tratados teoricamente no artigo anterior da nossa revista.

Mais do que «matar vários coelhos com uma cajadada», podemos afirmar, sem muito medo de errar, que o «modus faciendi» para a temática que é abordada pelo VIRT86.EXE, em particular o Modo Virtual 86 e o Mecanismo de «Paging», tem sido muito pouco (para não dizer nada) divulgado ao público até hoje. Por esse motivo estamos certos que o VIRT86.EXE será um contributo muito importante de estudo para os leitores mais interessados, ou apenas motivados por uma ou outra razão, em obter uma melhor compreensão de grande parte do que está por «detrás» de certo *software* «milagroso» dos nossos dias e referimo-nos em particular a gestores de memória como o QEMM, a plataformas de lançamento de programas como o Windows ou a sistemas operativos como o OS/2. Mas antes de nos debruçarmos sobre o funcionamento do programa VIRT86.EXE e também sobre algumas pequenas precauções a ter antes de o testar, vamos primeiro à doutrina, ou melhor, vamos à apresentação do:

### MODO VIRTUAL 86

É um dos modos de funcionamento dos processadores Intel 80386, 80486 e Pentium, o qual é caracterizado pelo facto de os programas de aplicação executarem (ou antes, sejamos parcimoniosos, quase sempre o fazerem) como se o processador se encontrasse em Modo Real (o qual é característico, como se sabe, dos velhos processadores 8086).

O termo Virtual 86 (ou mais precisamente Virtual 8086) tem a sua justificação etimológica no facto de ser possível uma coexistência pacífica e em simultâneo de vários ambientes 8086 no mesmo CPU. Notavelmente, e ao contrário do Modo Real, o Modo Virtual 86 é compatível com o Modo Protegido de trabalho dos processadores Intel de 32 bits (e muito naturalmente, com os de 64 bits também). O Modo Virtual 86 funciona como uma Tarefa dentro do ambiente de Modo Protegido. Como tal (e conferir por favor com o artigo sobre memória estendida publicado na **Spooler** Nº 26) essa Tarefa de Modo Virtual 86 terá de dispor obrigatoriamente de um *TSS Descriptor* na *Global Descriptor Table*.

Para o estabelecimento de uma Tarefa de Modo Virtual 86 é necessário, antes de mais, a existência de um Programa Monitor funcionando em Modo Protegido ao qual competirá inicializar o funcionamento do processador em Modo Virtual 86 e, subseqüentemente, intervir para o tratamento dos *Interrupts*, *Exceptions* e instruções de *Input/Output* que vão ocorrendo a todo o momento.

Esse Programa Monitor, que pode fazer parte ou não de um Sistema Operativo, executa sempre em Nível de Privilégio 0 dispondo, por conseguinte, de todas as faculdades para manipular convenientemente os Mecanismos de Segmentação e de *Paging* do Modo Protegido de 32 bits.

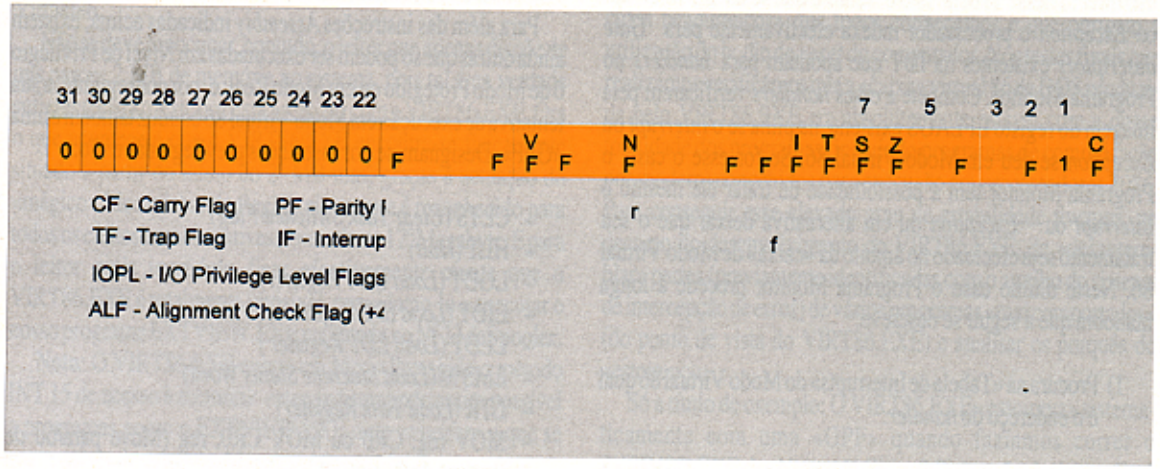
A Tarefa de Modo Virtual 86 propriamente dita executa, contudo, em Nível de Privilégio 3 (o nível mais baixo possível). Aqui, e tal como em Modo Real, o Mecanismo de Segmentação não irá actuar para o isolamento dos diversos segmentos entre si, mas actuará e ao mais alto nível (através do Programa Monitor), para isolar a Tarefa Virtual 86 das restantes Tarefas que eventualmente possam coexistir no mesmo CPU. Por sua vez, o Mecanismo de *Paging* permitirá, entre outras coisas, que o sistema:

- Crie várias Tarefas distintas de Modo Virtual 86. Cada uma dessas Tarefas pode mapear os endereços lineares do primeiro megabyte para quaisquer outros endereços da memória física;
- Emule o «dar a volta» no primeiro megabyte. Isto pode ser necessário porque nos velhos processadores 8086 o endereço 100000H coincidia, como muitos leitores saberão, com o endereço 0H.

E muitos programas hoje em dia ainda testam se a linha A20 se encontra aberta (e consequentemente se existe acesso à memória

estendida) através da comparação dos primeiros bytes a partir do endereço linear 0H com os primeiros bytes a partir do endereço linear 100000H.

Posto do modo mais simples possível, pode afirmar-se que um sistema inicialmente em Modo Protegido entra em Modo Virtual 86 quando o *flag* VM do registo EFLAGS passa a assumir o valor 1. Foque a sua atenção na figura seguinte onde apresentamos um esquema do registo EFLAGS do CPU dos processadores Intel de 32 bits e repare que os primeiros 16 bits são idênticos aos do registo FLAGS dos processadores de 16 bits. E o leitor pode concluir desde já, e bem, que só o facto de o *flag* VM ocupar a 18ª posição, isso implica de imediato que o Modo Virtual 86 (tal como foi concebido pela Intel) só é possível de ser realizado em processadores a partir de 32 bits.




Existem tão só e apenas dois procedimentos para colocar o *flag* VM com o valor 1 dando com isso início à execução do processador em Modo Virtual 86 e que são:

1) Uma Mudança de Tarefa (*Task Switch*) a qual fará carregar a quase totalidade dos registos do processador, incluindo o registo EFLAGS, a partir de um TSS (*Task State Segment*). Se a imagem do registo EFLAGS nesse TSS contiver na posição correspondente à *flag* VM o valor 1, então o processador saberá que no contexto da nova Tarefa os endereços lineares (dados pelo dueto segmento/deslocamento) deverão ser interpretados à maneira dos velhos processadores 8086.

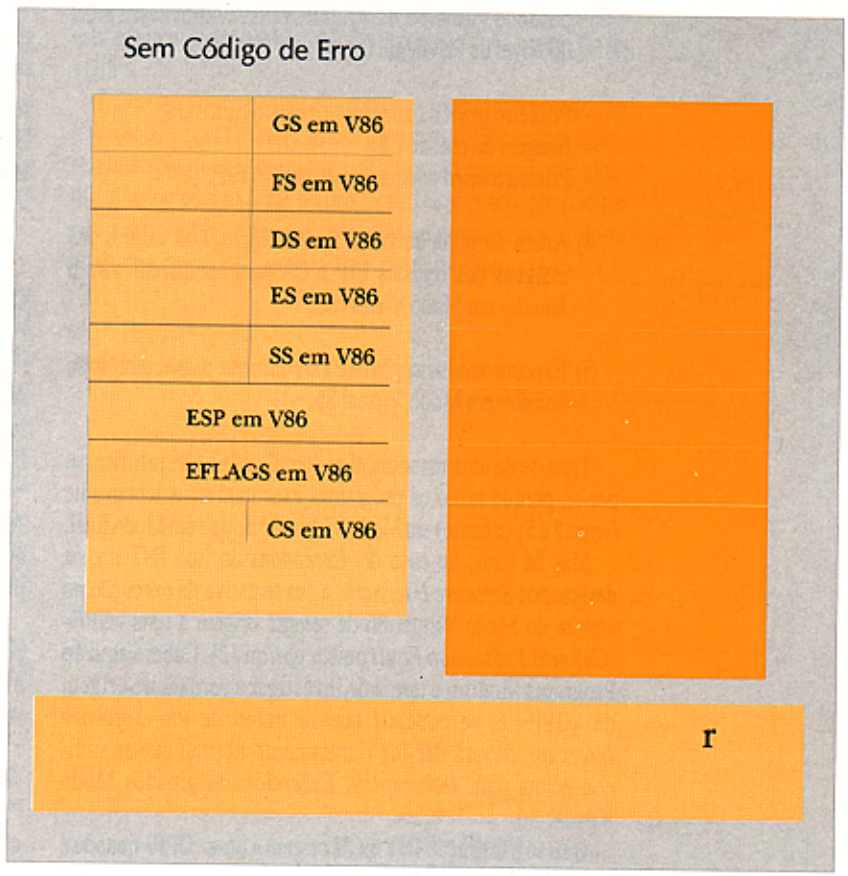
2) Uma instrução «IRETD» no contexto de uma Tarefa executando em Modo Protegido fará carregar o registo EFLAGS com um *doubleword* existente na pilha.

Se esse *doubleword* contiver na posição correspondente à *flag* VM o valor 1, então o processador devolverá o controlo para uma rotina que executará em Modo Virtual 86.

O processador abandona o Modo Virtual 86 e prossegue a execução em Modo Protegido quando se verificar uma das duas situações seguintes:

1) Uma Mudança de Tarefa veículada por uma *Task Gate* aquando da ocorrência de um *Interrupt* ou *Exception* (ao qual corresponda na *Interrupt Descriptor Table* um *Task Gate Descriptor*) carrega o registo EFLAGS (a partir de um TSS naturalmente) com um *doubleword* no qual a posição correspondente à *flag* VM apresente o valor 0.

2) Um *Interrupt* ou *Exception* chama uma rotina com Nível de Privilégio 0 via uma *Interrupt Gate* ou *Trap Gate*. Ao fazê-lo, o processador guarda automaticamente na pilha (e pela ordem que se indica) o conteúdo dos registos de segmento GS, FS, DS, ES, SS, o valor dos registos ESP e EFLAGS, o do registo de segmento CS e o do registo EIP. No caso de se tratar de uma *Exception* 8 ou entre 10 e 14, será ainda gerado um código de erro o qual é igualmente colocado na pilha. O aspecto da pilha após a reentrada em Modo Protegido é ilustrado pela figura seguinte:



Seguidamente o processador coloca o valor zero em todos os registos de segmento e também no *flag* VM do registo EFLAGS passando a executar de imediato em Modo Protegido de 32 bits.

Agora perguntar-se-á o leitor mais atento, e que já se apercebeu do maior busílis da questão: «Como é possível que um *Interrupt* ou *Exception* que ocorre em Modo Virtual (que como se disse é compatível com o Modo Real) faça chamadas a *handlers* do tipo *Task Gate*, *Interrupt Gate* ou *Trap Gate* que fazem parte da *Interrupt Descriptor Table* do Modo Protegido?» A explicação – e confessemos que não é nada evidente – é a seguinte:

Naturalmente que os programas que executam em Modo Virtual 86 têm de ter uma «Tabela de *Interrupts*» que se inicia no endereço linear 0, por razões óbvias de compatibilidade com o Modo Real. Contudo, em Modo Virtual 86 o processador não faz uso directo dessa Tabela. Isto é, sempre que se dá um *Interrupt* ou *Exception* o processador indexa efectivamente para *Gate descriptors* existentes na IDT que apontam para *handlers* no Programa Monitor. Compete a esses *handlers* verificarem pela imagem do registo EFLAGS existente na pilha se o *Interrupt* ou *Exception* se deu em Modo Virtual 86. Se for esse o caso, o Programa Monitor tem a possibilidade de tratar ele mesmo o *Interrupt* ou *Exception* ou em alternativa deixar que o seu tratamento seja efectuado no âmbito da «caixa» de Modo Virtual 86. Neste último caso o Programa Monitor procede à longa manobra que a seguir se descreve:

**Naturalmente que os programas que executam em Modo Virtual 86 têm de ter uma «Tabela de *Interrupts*» que se inicia no endereço linear 0, por razões óbvias de compatibilidade com o Modo Real.**

1) Procura na «Tabela de Interrupts» do Modo Virtual 86 qual é o endereço do *handler*.

2) Procura na pilha do Nível de Privilégio 0, pelo conteúdo dos registos SS e ESP em Modo Virtual 86.

3) Fazendo contas com base nesses registos, guarda na pilha do Modo Virtual 86, os seguintes valores existentes na pilha do Nível de Privilégio 0:

- Primeiro *word* da imagem do registo EFLAGS;
- Imagem do registo CS;
- Primeiro *word* da imagem do registo EIP.

4) Altera na pilha do Nível de Privilégio 0 os valores das imagens dos registos EIP e CS para apontarem para o *handler* em Modo Virtual 86.

5) Executa uma instrução IRETD com o que passa controlo ao *handler* em Modo Virtual 86.

Trata-se de uma manobra algo complicada e que justifica, só por si, porque razão os programas executam mais lentamente (entre 2 e 5 por cento) em Modo Virtual 86 do que em Modo Real.

Mas há mais, no caso de *Exceptions* do tipo INT nn, os designados *Software Interrupts*, a sua tentativa de execução no âmbito do Modo Virtual 86 dá sempre origem a uma «GPF» (*General Protection Fault* ou *Exception 13*). Caberá aqui ao Programa Monitor a tarefa de investigar a verdadeira origem da «GPF» e, se concluir que se tratou de um *Software Interrupt*, deverá dar-lhe o tratamento normal que se indicou acima para *Interrupts* e *Exceptions* originados Modo Virtual 86.

Não só a instrução INT **nn** dá origem a uma «GPF» quando é executada em Modo Virtual 86. As seguintes instruções *Assembly* são também susceptíveis de a produzir:

- CLI (*Clear Interrupt Flag*)
- STI (*Set Interrupt Flag*)
- PUSHF (*Push Flags*)
- POPF (*Pop Flags*)
- IRET (*Return from Interrupt*)

Como o Nível de Privilégio em Modo Virtual 86 é sempre 3, se o campo IOPL do registo EFLAGS for menor que 3, qualquer tentativa de execução das instruções *Assembly* indicadas acima gera sempre uma «GPF», o que pode ter utilidade designadamente para dar oportunidade ao Programa Monitor de assumir o controlo e proceder ao seu tratamento conveniente.

Para além das instruções *Assembly* indicadas acima, existem ainda outras que só podem ser executadas em Nível de Privilégio 0 de Modo Protegido ou em Modo Real, e, por conseguinte, a sua tentativa de execução em Modo Virtual 86 conduz também a uma «GPF». Designam-se por instruções previlegiadas e são:

- CLTS (*Clear Task Switched Flag*)
- HLT (*Halt*)
- LGDT (*Load GDT Register*)
- LIDT (*Load IDT Register*)
- LLDT (*Load LDT Register*)
- LMSW (*Load Machine Status Word*)
- LTR (*Load Task Register*)
- MOV reg, CR0 ou MOV CR0, reg (Move para/ou de Registo de Controlo 0)
- MOV reg, DRn ou MOV DRn, reg (Move para/ou de Registo de Debug n)
- MOV reg, TRn ou MOV TRn, reg (Move para/ou de Registo de Teste n)

As instruções *Assembly* de *Input/Output* por seu lado não são sensíveis em Modo Virtual 86 ao campo IOPL do registo EFLAGS, mas sim ao *I/O Permission Bitmap* já referido no artigo anterior.

E de teoria cremos que já basta, vamos agora passar à prática.

## O PROGRAMA VIRT86.EXE

Antes de executar este programa retire, primeiro que tudo, do seu CONFIG.SYS, caso lá se encontrem, *device drivers* que instalem gestores de memória, tais como o HIMEM.SYS, EMM386.SYS ou o QEMM386.SYS.

Retire também, quer do CONFIG.SYS quer do AUTOEXEC.BAT, e isto é mesmo «muito importante», quaisquer *caches* de disco como por exemplo o SMARTDRV.EXE ou o PC-Cache.

Se considera o procedimento anterior muito confuso, pode preferir a alternativa de «fazer o Rename» do seu CONFIG.SYS e AUTOEXEC.BAT para, por exemplo, Config.sss e Autoexec.bbb.

Seguidamente faça o rearranque do sistema com a conhecida saudação dos três dedos (isto é, Ctrl + Alt + Del).

**Nota**: Se possuir o DOS 6.0, pode conseguir o mesmo efeito carregando na tecla F5 durante o rearranque do sistema.

Quando estiver de novo no *prompt* do DOS, já pode restaurar os anteriores CONFIG.SYS e AUTOEXEC.BAT. E pode também não ser má ideia fazer executar, nesta fase, se dispuser de teclado português, o comando externo do DOS «KEYB PO» por razões algo evidentes. Após estes preparativos poderá dirigir-se para o directório onde se encontra o VIRT86.EXE e executá-lo.

O VIRT86.EXE mostra-lhe no *prompt* um indicativo (eventualmente a cores se tiver instalado o ANSI.SYS) de que foi instalado. Experimente agora saber qual o espaço ocupado em memória pelo VIRT86.EXE. Com o utilitário MEM.EXE do DOS (ou qualquer outro equivalente) faça MEM /c. Conclusão: o VIRT86.EXE desapareceu sem deixar rasto.

Mas não obstante não estar visível, o VIRT86.EXE encontra-se em execução no 17º Megabyte de memória, exercendo orgulhosamente as suas funções de Programa Monitor do Modo Virtual86.

Não se deve preocupar, contudo, o leitor que apenas dispõe de uns parcos 2 MB de memória no sistema, pois tal é na verdade suficiente para o teste que está a realizar com o VIRT86.EXE. Ora o que acontece é que o VIRT86.EXE encontra-se fisicamente situado a partir do início do 2º MB mas, graças à actuação do «mágico» Mecanismo de *Paging* ele actua e é reconhecido para todos os efeitos como se estivesse situado no 17º Megabyte. E para o leitor mais incrédulo ficar efectivamente ciente que o VIRT86.EXE é totalmente invisível, convidamo-lo a executar o nosso programa EXTVIEW.EXE publicado no Nº 23 da **Spooler**.

**Nota:** O VIRT86.EXE emula em Modo Protegido o método INT 15 de acesso à memória *extended*, tornando assim possível a programas como o EXTVIEW.EXE irem «cheirar» o que se passa acima do primeiro Megabyte de memória.

Outra acção do VIRT86.EXE é reduzir a memória *extended* em 1 MB. Se o leitor tinha 3 MB ficará apenas com 2 MB. Naturalmente que não provocámos nenhuma avaria nos módulos SIMM. O que se passa é que o VIRT86.EXE intercepta qualquer acesso ao CMOS (o qual é em última análise efectuado através de instruções *Assembly* de *Input/Output*) e subtrai à memória *extended* relatada pelo CMOS o valor de 1 MB antes de a informação ser passada de volta para o programa inquirente.

**Nota:** E para completar a «mentirola» sobre a diminuição de memória, tivemos também que actuar em duas outras frentes:

- através de uma emulação do serviço 88H do INT 15H.
- sobre a informação existente na estrutura *List of Lists* (obtida via serviço 52H do INT 21H) relativa ao valor da memória *extended* e que é recolhida pelo DOS durante o arranque do sistema.

Isto feito testámos todos os grandes programas capazes de analisar este tipo de assuntos e todos engoliram a «grande mentira». Claro que podíamos ter feito o inverso e relatar ao sistema que dispúnhamos de mais memória. Deixamos contudo esse exercício ao leitor que pretenda surpreender os amigos.

Outra característica do VIRT86.EXE é a sua capacidade para criar UMB (*Upper Memory Blocks*) ao iniciar a sua execução. Por exemplo, se o leitor teclar «VIRT86 RAM=D000-DFFF» toda a zona de memória compreendida entre os endereços D000:000 e E000:0000 ficará habilitada para leitura e escrita, o que pode ser facilmente verificado com o auxílio do utilitário do DOS de nome DEBUG.EXE (e executando o comando «d d000:0000»)

O VIRT86.EXE pode ainda providenciar ou não o acesso à HMA (*High Memory Area*).

Por omissão tem-se sempre acesso à HMA (ou melhor, a uma pseudo HMA), mas com «VIRT86 NOHMA» então verifica-se «o dar a volta» que se mencionou mais acima e não há HMA para ninguém.

O VIRT86.EXE dispõe também de alguns comandos internos. Por exemplo, o comando «sai» (que exactamente por ser interno pode ser digitado directamente no *prompt* do DOS em qualquer directório ou partição) conduz-nos de imediato de volta ao Modo Real. Sobre os restantes comandos internos convidamos muito amigavelmente o leitor, e a título de exercício, a tentar descobri-los pela leitura do código fonte do programa.

Tentámos dotar o VIRT86.EXE de uma sistema de intercepção de erros tão robusto quanto possível de modo a evitar os «aborrecidos penduranços» que todos conhecemos e que obrigam à utilização do botão de *reset* do computador. Nos testes finais que realizámos verificámos que a «caixa» virtual 86 de DOS produzida pelo VIRT86.EXE era absolutamente estável com a quase totalidade dos programas (incluindo programas de comunicação por modem) concebidos para execução em processadores 8086. Para conseguir esse elevado grau de estabilidade tivemos, no decurso do desenvolvimento do VIRT86.EXE, de «desassemblar» partes (previamente identificadas com o auxílio do sistema de intercepção de erros) de vários programas «mal comportados» (do ponto de vista do VIRT86.EXE) e analisar os porquês do «estoiranço».

Só a título de exemplo: O VIRT86.EXE «estoirava» sistematicamente com uma «GPF» quando fazíamos correr o SYSINFO.EXE das Norton Utilities 7.0. A razão devia-se afinal à execução por parte deste programa da instrução privilegiada MOV EAX, CR0 a qual é muitas vezes utilizada em Modo Real para verificar se o processador está em Modo Protegido. Naturalmente que tivemos de encarregar o Programa Monitor de emular esta e outras instruções em Modo Protegido após a geração da «GPF».

Inúmeros outros pequenos «truques» foram incorporados no VIRT86.EXE para o fazer cumprir a sua missão tão condignamente quanto possível, «truques» esses que estão devidamente identificados no código fonte do programa. Esperamos que o leitor estudioso destes temas tire pelo menos tanto proveito deste programa «invisível», quanto a nós prazer nos deu criá-lo.

**“Tentámos dotar o VIRT86.EXE de uma sistema de intercepção de erros tão robusto quanto possível de modo a evitar os «aborrecidos penduranços».”**

José Páscoa