PROGRAMAR EM ASSEMBLY (PARTE II)

> **...um computador pessoal pode ser entendido como um** sistema **de cinco unidades funcionais...**

Afirmámos no artigo anterior que não existe melhor alternativa para aprofundar o nosso conhecimento sobre um computador do que o estudo da sua linguagem *Assembly*. Uma das razões é que a programação em *Assembly* depende em absoluto, e é indissociável, da realidade física desse computador.

Reservámos o artigo de hoje, exactamente, para fazer uma pequena apresentação desse«sujeito» que dá pelo nome de PC (e entendamos aqui por PC, o IBM PC e seus compatíveis e descendentes) pois muitas das suas peculiaridades são decisivas para se saber como «falar» com ele através do *Assembly*. Se já não tem muitas dúvidas de que é mesmo importante aprender *Assembly*, então encha-se de coragem e avance na leitura.

A ARQUITECTURA DOS PC

1- Visão geral

Muito simplificadamente, um computador pessoal pode ser entendido como um **sistema** de cinco unidades funcionais: a **unidade de entrada**, a **unidade de memória**, a **unidade aritmética**, a **unidade lógica** e a **unidade de saída**.

As unidades de entrada e saída estão em contacto com o mundo exterior ao sistema e esse «mundo» é constituído, ou pode sê-lo, pelo teclado, o monitor, o rato, o disco rígido e as *drives* de disquete, a impressora, o *modem*, uma placa de fax, um CD-ROM e, enfim, qualquer outro **periférico** presente ou que o futuro nos possa trazer.

As unidades de entrada e saída comunicam com o resto do sistema através de **portos** (em inglês diz-se *Input/Output ports*), os quais não devem ser confundidos com as conhecidas «portas» de comunicação série e paralela que todos os PC normalmente possuem e que nos permitem acrescentar ao sistema, periféricos tais como impressoras ou *modems*.

Nos PC as unidades aritmética e lógica estão agrupadas naquilo que se convencionou chamar a Unidade Central de Processamento, abreviadamente **CPU** (do inglês *Central Processing Unit*) mas mais vulgarmente **microprocessador** (ou simplesmente **processador**).

Para o PC nos ser útil necessita de actuar sobre um tipo de informação que dá pelo nome de **programas** os quais são constituidos por comandos que ele entende, e designados **instruções** as quais, quando passadas para o microprocessador de um modo **sequencial** (sequencialmente não significa obrigatoriamente que a instrução anterior esteja ordinalmente posicionada antes), vão actuar sobre outro tipo de informação designada por **dados**.

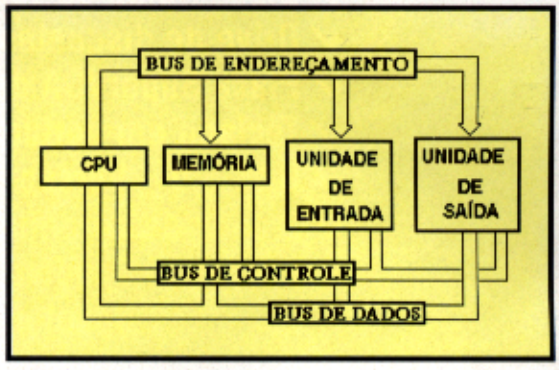
Os programas e os dados são introduzidos no sistema pela unidade de entrada e armazenados na **memória**. Os PC guardam indistintamente na «mesma» memória, tanto os programas como os dados, cabendo ao programador a responsabilidade de fazer o computador diferenciar entre os dois tipos de informação.

Todo este esquema de funcionamento foi idealizado há muitos anos por um senhor chamado Von Neumann, e por isso podemos dizer que um PC é uma máquina classificável como tendo arquitectura de Von Neumann.

Parte da memória designa-se por **RAM** (*Random Acess Memory*) e outra parte designa-se por **ROM** (*Read Only Memory*). RAM é memória que permite leitura e escrita e pode ser também chamada de RWM (*Read/Write Memory*), embora este termo tenha caido um pouco em desuso. ROM é a memória que só permite leitura. O ROM normalmente já vem de fábrica com todos os programas e dados que irá conter durante toda a sua vida (como excepção à regra existe um tipo especial de ROM, designado EPROM, cuja informação pode ser substituída com o auxílio de equipamento especial). O ROM *também é* Random Acess Memory mas não é RAM no mesmo sentido que se dá hoje a este termo!

Tanto a RAM como o ROM são constituídos por uma larga sequência de **localizações** cujo conteúdo é 1 **byte** (8 bits). Cada uma dessas localizações tem um **endereço**. O endereço é expresso por um número que define unívocamente o byte dessa localização, mas também podemos juntar 2 bytes seguidos (nos PC chama-se um **word**) ou 4 bytes seguidos (nos PC chama-se um **doubleword**) e acedê-los simultaneamente com um único endereçamento.

Nos PC as várias unidades são ligadas por um conjunto de circuitos eléctricos designados por **BUS** (há quem diga barramento ou via), os quais são constituídos por **linhas de sinal**, cada uma delas podendo transportar 1 bit de informação... Existe o Bus de Controle, o Bus de Endereçamento e o Bus de Dados.



Qualquer dado que «viaje» do microprocessador para a memória ou para as unidades de entrada e saída, ou entre qualquer das outras partes componentes do sistema, fá-lo sempre através do Bus de Dados.

Mas antes desse dado viajar, o seu endereço, no caso de se tratar de uma célula de memória ou de um porto, é colocado préviamente no Bus de Endereçamento.

Por sua vez, o Bus de Controle ajuda a coordenar «toda essa louca correria» de um lado para o outro. Para toda esta «orquestra» acertar o passo existe também um **metrónomo**, isto é, um gerador de frequências constantes designa-

do *clock generator* (traduzido normalmente por relógio).

O microprocessador e todos os outros componentes trabalham a uma frequência que é sempre uma fracção da frequência do clock.

Naturalmente que todos estes conceitos «dão pano para mangas» e a sua explicação em pormenor daria garantidamente para se escrever um livro bastante volumoso. Porém, se conseguiu apreender o que foi dito até aqui (se é que não sabia já), então possui uma base mínima para poder prosseguir.

Existem ainda outros componentes do sistema com certa importância para o programador, mas vamos deixar a sua abordagem para uma outra ocasião.

Para a iniciação à programação em *Assembly* é essencial, contudo, ter uma ideia um pouco mais aprofundada sobre dois dos componentes do sistema: o CPU (que integra as unidades lógica e aritmética) e o modo como ele está organizado, e a memória e a sua estrutura interna.

No artigo de hoje falaremos precisamente sobre o primeiro deles, o CPU.

2 – O CPU

O CPU ou microprocessador é constituído por pequenos locais de armazenamento designados **registos.** Do ponto de vista do programador o melhor enfoque a dar ao CPU é precisamente o de vê-lo como um conjunto de registos.

O microprocessador do PC original foi baptizado de **8088** e era caracterizado por ter **registos internos de 16 bits**, mas utilizar um Bus de Dados de apenas 8 bits.

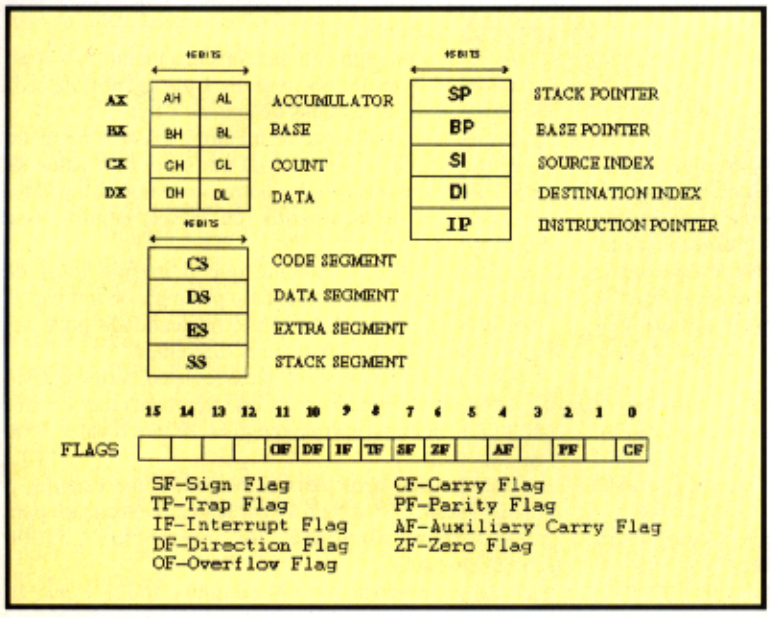
Pouco tempo depois apareceu o **8086** que era muito semelhante ao 8088, mas podia usar um Bus de Dados de 16 bits.

Na altura algumas pessoas referiam-se ao 8088 como um falso processador de 16 bits (o que lembra a polémica dos nossos dias entre o 80386DX e o 80386SX).

Contudo, a verdade é que o 8088 é idêntico ao 8086 a nível de registos internos e deve ser considerado um processador de 16 bits, apesar de a transferência de words de e para a memória ser efectuada através de 2 acessos consecutivos.

Tanto o 8088 como o 8086 têm um Bus de Endereçamento de 20 linhas (numeradas de 0 a 19) e a sua capacidade de endereçar memória é assim de 2 elevado a 20, o que corresponde a 1 megabyte, 1024 kilobytes ou 1048576 bytes.

Os registos do 8088/8086 são mostrados na gravura seguinte:



Na linha de evolução dos microprocessadores 8088 e 8086, e devido ao sucesso dos computadores que os tomaram por base, a Intel desenvolveu o **80286**, o qual também é constituído por registos de 16 bits e um Bus de Dados de 16 bits, mas possui um Bus de Endereçamento de 24 linhas.

O 80286 possui mais alguns registos que o 8088/8086 o que, aliado à capacidade de endereçamento de 16 megabytes, lhe permite trabalhar naquilo que se designa por **modo protegido.**

Contudo, mantém plena compatibilidade a nível dos mesmos registos já existentes no 8088/8086, quando trabalha em **modo real**.

Depois do 80286 apareceram os 80386 e 80486 que são microprocessadores de 32 bits e tem um Bus de Dados de 32 bits para acesso à memória e, eventualmente, também as unidade de entrada e de saída (se em arquitectura EISA), e um Bus de Endereçamento também de 32 bits. O 80386 SX possui um Bus de Dados de apenas 16 bits para acesso à memória e um Bus de Endereçamento de 24 bits, mas considera-se também um microprocessador de 32 bits pois é internamente idêntico aos outros e integralmente compatível com eles.

Todos os modelos 80386 (incluindo o SX) e 80486, e só a titulo de curiosidade, possuem um espaço de endereçamento lógico de até 64 triliões (2 elevado à 46ª potência) de bytes em estrutura segmentada, ou um máximo de 4 biliões de bytes num único segmento. E não é necessário haver correspondência em memória física para esse endereço ser produzido, o espaço em disco também serve, como se vê pelas versões 3 do Windows, porque este «milagre» é conseguido através de um mecanismo assaz complicado e que se designa globalmente por **virtualização de memória.**

Apesar destes microprocessadores melhorarem em muito a capacidade de actuação em modo protegido comparativamente ao 80286 e implementarem ainda um novo modo de trabalho designado por **modo virtual 86** (parecido com o modo real e na prática indiferenciável pelos programas executáveis sobre o MS-DOS, pelo que tudo o que dissermos sobre o modo real também se aplica ao modo virtual), acima de tudo mantiveram também a compatibilidade a nível de registos com os «velhinhos» 8088/8086 quando trabalham no modo real.

O máximo divisor comum de todos os processadores da família é, como já verificou, o trabalho em modo real. E, como nos nossos dias, 95% dos programas existentes foram feitos para trabalhar apenas em modo real, é extremamente importante começar por conhecer como trabalham os microprocessadores desta família nesse modo.

Deixemos apenas a salvaguarda que os 80386 e 80486 possuem ainda outros registos internos que podem ser utilizados em modo real e que não existiam nos modelos anteriores; contudo, na prática, os programas que pretendam correr em todos os computadores da família não deverão fazer uso deles.

Para efeito do nosso estudo os registos podem ser divididos em 4 grupos tal como se vê na figura à esquerda, e assim teremos:

a) Registos de utilização geral

Designam-se por **registos de utilização geral** os registos AX, BX, CX e DX.

Cada um deles é por sua vez subdivisível em dois registos, com-

> **Nos PC as unidades aritmética e lógica estão agrupadas naquilo que se convencionou chamar a Unidade Central de Processamento**

preendendo quer a parte dos últimos 8 bits (parte alta) quer a parte dos primeiros 8 bits (parte baixa) do registo original.

Por exemplo, à parte alta do registo AX dá-se o nome de registo AH (*Accumulator High*) e à sua parte baixa dá-se o nome de registo AL (*Accumulator Low*). E de modo idêntico seria para os outros registos deste grupo.

O programador é relativamente livre de escolher, com alguma arbitraridade, qualquer um destes registos para muitas operações aritméticas e lógicas, mas existem contudo grupos de instruções em que tal já não pode acontecer. Isto significa que, embora haja uma certa flexibilidade, os registos de utilização geral têm uma certa vocação.

Assim o AX é vocacionado para multiplicar e dividir, o BX é o único neste grupo que pode conter um endereço de memória para aceder a dados, o CX é utilizado como contador de iterações em várias instruções e o DX é o único que pode especificar o endereço de um «porto» nas instruções que permitem o seu acesso.

> **Designam-se por** registos de utilização geral **os registos AX, BX, CX e DX**

b) Registos Ponteiro e Registos de Indexação

Quando a Intel iniciou o projecto do 8088, partiu da premissa que as linguagens de programação de alto nível com futuro nos PC seriam aquelas que, como o Pascal (a linguagem C também, mas na altura ainda era pouco popular), passam os dados às «subrotinas» através do **stack** (também designado por **pilha**) e que usam também esse stack para a colocação das variáveis locais da subrotina.

Nessa linha de pensamento tinha muita lógica que o microprocessador tornasse a «vida mais fácil» aos compiladores dessas linguagens, disponibilizando registos suficientes para uma correcta manipulação dos dados existentes no stack.

Para quem ande um pouco «arredado destas coisas»: nos PC o stack é uma área da memória que o sistema operativo ou o programa em execução reservou primáriamente para o microprocessador poder guardar os endereços de retorno das subrotinas, mas que pode também servir para as finalidades acima indicadas.

O stack funciona como uma «pilha de pratos que estão para lavar na cozinha». O que chegou em último lugar está no topo da pilha e lavar-se-á primeiro (e o primeiro da pilha pode ter muito que esperar até ser lavado).

Curiosamente, nos PC a pilha está virada ao contrário e o seu **topo** tem, por conseguinte, o endereço de memória mais baixo. Para saber qual é o topo do stack o processador utiliza um registo ponteiro de nome SP (*Stack Pointer*).

E para saber qual é o endereço de uma área dentro do stack que contém os dados passados à subrotina (ou as variáveis locais), o processador socorre-se de outro registo ponteiro de nome BP (*Base Pointer*).

Uma das primeiras acções que muitas subrotinas produzidas em linguagens de alto nível executam (em programa já compilado naturalmente) é passarem para o registo BP o valor do topo do stack, de modo a poderem aceder aos dados no stack, e fazem isso copiando para esse registo o conteúdo de SP.

Mas, muitas outras vezes, nós não queremos manipular dados no stack, mas sim em áreas da memória especificamente reservadas para dados.

Podemos, por exemplo, desejar efectuar uma movimentação de um grupo de dados entre duas dessas áreas. Para essa finalidade existem dois registos classificados como registos de indexação e que são o SI (*Source Index*) e o DI (*Destination Index*).

A razão do nome «indexação» será posta em evidência quando estudarmos as instruções de movimentação de *strings*.

Tanto os registos Ponteiro como os de Indexação trabalham em conjunto com os registo de Segmento, que serão explicados mais abaixo.

Por último, e ainda neste grupo, temos o IP (*Instruction Pointer*) que guarda o endereço da instrução que vai ser executada dentro do registo de segmento de código CS.

O IP faz parte da unidade de controlo do microprocessador e os programas não podem alterar directamente o seu valor, embora possam contudo fazê-lo indirectamente com instruções que alterem a sequência da execução de um programa, isto é, instruções que provoquem saltos para outros locais dentro do programa.

c) Registos de Segmento

Muitas pessoas desistem à primeira do estudo da linguagem *Assembly*, não por qualquer eventual dificuldade no entendimento do conjunto de instruções *Assembly*, que é extraordinariamente simples e muito mais reduzido do que o de qualquer linguagem de alto nível. Também não é sequer pela necessidade de ter de conhecer as directivas que é necessário dar ao Assemblador para que ele entenda bem o que nós pretendemos.

O que muitas pessoas tem mais dificuldade em assimilar é que a memória é tratada pelos processadores desta família como estando dividida em compartimentos de 64 KB (em modo real, pois em modo protegido podem ser maiores ou menores), designados, muito naturalmente, por **segmentos**.

A necessidade dos segmentos radica num facto muito simples.

A Intel pretendia que o microprocessador 8088/8086 podesse endereçar 1 MB (que corresponde a 2 elevado à 20ª potência) de memória, mas este processador dispunha apenas de registos de 16 bits que, no máximo, permitem construir um endereço compreendido entre 0 e 65535 (isto é uma quantidade de endereços correspondentes a 2 elevado à 16ª potência).

Como naquele tempo não se tornava viável a construção de um processador com alguns registos de 20 bits (pelo menos os que endereçam memória), encontrou-se uma solução alternativa e que, embora de «grande engenho e arte», não deixa de ser um pouco confusa mesmo para quem já tenha alguma prática de programação em *Assembly* noutros tipos de microcomputadores.

Nos PC cada posição de memória é determinada por um conjunto de 2 registos, o primeiro dos quais é um dos registos (e existem 4 deles no 8088/8086 e 80286) designados por **registo de segmento** e o segundo é um dos outros que vimos atrás como tendo capacidade para endereçar memória.

Os registos de segmento percorrem a memória a «passadas» de 16 bytes (também se diz **1 parágrafo**) de cada vez, e os outros a passadas de apenas 1 byte.

E para cada localização de memória existe sempre um registo de segmento que nos dá um valor dito **valor de segmento**, e um registo «dos outros» que nos dá um valor dito valor do **deslocamento** (ou **offset**) dentro desse segmento.

Contudo, vê-se facilmente que o mesmo local de memória pode ficar representado por mais de um par de valores segmento/deslocamento, na realidade pode ser representado por até 4096 pares diferentes.

Por exemplo, imagine-se que o registo CS (um registo de segmento designado por Code Segment) tem o valor 16000 (em decimal), e que o registo IP tem o valor 3000 (em decimal).

Isto significa nem mais nem menos que estamos a apontar (neste caso é para a instrução em execução) para o **endereço absoluto de memória** 259000 (16000x16+3000).

Imagine-se agora que CS continha 16001 e que IP

continha 2984. Se fizer as contas obtém também o endereço absoluto de memória 259000.

Uma confusão muito comum é imaginar-se que os segmentos se encontram espaçados de 64 KB, de tal modo que um computador com 640 KB de RAM possui 10 segmentos. Isso não é correcto, pois os segmentos podem estar parcialmente sobrepostos como se viu no exemplo acima. E qualquer segmento pode iniciar-se em qualquer parágrafo (múltiplo de 16 bytes) da memória.

Se neste momento está perfeitamente confuso então considere-se uma pessoa normal, pois os conceitos de segmentação da memória, registos de segmento e deslocamento dentro do segmento não são de imediato intuitivos. Encha-se de coragem e continue em frente, pois se há quem demore apenas alguns minutos a ganhar a percepção correcta destes conceitos, a maioria demora horas ou dias (quando não é mais).

Fique neste momento ciente que nem tudo é mau a respeito da segmentação; na verdade, os processadores 80286 e superiores, quando a trabalhar em modo protegido, segmentam a memória para poderem isolar programas (genéricamente designados em modo protegido por **processos**) e as suas partes umas das outras e assim implementarem a virtualização da memória e a independência dos processos.

Os PC 8088/8086 e o 80286 possuem 4 registos de segmento e são eles o CS (*Code Segment* ou Segmento de Código), o DS (*Data Segment* ou Segmento de Dados), o ES (*Extra Segment* ou Segmento Extra) e o SS (*Stack Segment* ou Segmento da Pilha).

O valor contido em cada um destes registos define uma área de 64 KB.

O CS trabalha conjuntamente com o IP para apontar para a instrução que vai ser executada pelo programa.

Do valor contido no registo CS diz-se que é o valor corrente do segmento de código.

Se o conjunto de instruções de um programa for maior que 64 KB é necessário, pelo menos uma vez, e algures dentro desse programa, alterar o valor de CS. Ao contrário dos outros registos de segmento, o valor de CS é apenas alterável indirectamente (de modo semelhante ao IP), por exemplo por instruções que provoquem um tipo de salto designado FAR (para longe).

O DS contém o valor corrente do segmento de dados.

O ES pode ser considerado também como um segmento de dados mas de carácter auxiliar para possibilitar a execução de certas instruções.

O SS é o segmento do Stack.

Nas instruções *Assembly* que endereçam memória existe sempre um segmento considerado o *default* para essa instrução.

Por exemplo, instruções que endereçam o stack fazem uso dos registos SP ou BP e assumem implicitamente que o segmento *default* é o SS. Isso significa que não é necessário mencionar o nome do segmento na instrução.

A título ilustrativo, a instrução MOV AX,[BP] significa que o registo AX ficará com o conteúdo da posição de memória endereçada pelo valor do registo BP dentro do segmento SS.

Contudo, a instrução pode ser compelida a utilizar outro segmento que não o *default*, através de uma indicação explícita do nome do segmento.

No caso acima, se escrevessemos MOV AX,DS:[BP] o registo AX ficaria carregado com o conteúdo da posição de memória endereçada pelo valor do registo BP dentro do segmento de dados DS.

Diz-se num caso destes que houve uma **derrogação** (em inglês *override*) do segmento *default*.

d) O registo Flags

Todos os processadores desta família reservam um registo cujo objectivo se assemelha ao de um painel de controle.

É constituído por **flags** (sinalizadores) que podem assumir apenas um de dois resultados, 0 ou 1. Cada flag ocupa um bit no registo Flags. As flags cumprem três finalidades genéricas:

“Se o conjunto de instruções de um programa for maior que 64 KB é necessário, pelo menos uma vez, e algures dentro desse programa, alterar o valor de CS”

Umas servem para prestar informação complementar sobre o que se passou na execução da última instrução (*Status Flags*), outras para controlar o comportamento do microprocessador na execução de certas instruções (*Control Flags*), e ainda outras para auxiliar internamente a operação do processador (*System Flags*).

Nos processadores 8088/8086 e 80286 o registo Flags é de 16 bits.

Como pode verificar pela figura, nem todos os bits são utilizados, pois alguns são considerados reservados para eventual utilização por futuros membros da família. Vamos conhecê-los então um pouco melhor:

CF – É uma *status flag*. Após uma operação aritmética reflecte se houve transporte do último bit, isto é, se *vai um*. Se CF for 1 representa-se essa situação por CY e se for 0 representa-se por NC.

PF – É uma *status flag*. Após uma operação lógica ou aritmética se o resultado tiver número impar de bits 1 este flag será 0 (e representa-se PO, *parity odd*), se for par será 1 (e representa-se por PE, *parity even*).

AF – É outra *status flag*. Indica se houve transporte do bit 3 (isto é o quarto bit, pois eles contam-se a partir do 0). Em caso positivo representa-se por AC e em caso negativo por NA. Este flag é pouco utilizado na prática pelos programas.

ZF – Mais uma *status* flag. Indica se o resultado de uma operação é 0.

Se for 0 o flag apresenta-se com o valor 1 e representa-se por ZR em caso contrário por NZ.

SF – Também uma *status flag* e indica se o resultado de uma operação é positivo ou negativo. Se negativo representa-se por NG e se for positivo por PL.

TF – É uma *system flag*. É utilizada pelo processador quando executa instruções «passo a passo» como por exemplo de dentro do programa DEBUG.

IF – Outra *system flag*. Sinaliza se os *interrupts* podem ser atendidos. O conceito de interrupt (interrupção) será explanado no próximo artigo.

No caso de poderem ser atendidos este flag apresenta o valor 1 e representa-se por EI (*enable interrupts*), e em caso contrário por DI (*disable interrupts*).

DF – Trata-se de uma *control flag*. Se o valor do bit for 0 (e representa-se por UP) então algumas instruções que envolvem os registo de indexação (designadas instruções *string*) incrementam automáticamente esse registos posicionando-os no byte seguinte da *string*. Em caso contrário, isto é se o bit for 1 (e representa-se por DN), os registos de indexação são decrementados.

OF – É um *status flag*. Indica se o resultado da operação aritmética ultrapassou a capacidade do byte ou word que deveria receber esse resultado. Se assim for escreve-se OV (*overflow*) e em caso contrário NV (*not overflow*).

E terminamos aqui a pequena apresentação que queríamos fazer deste componente fundamental do PC que é o CPU.

A etapa de hoje foi crucial para criar as bases necessárias para «chegar á fala» com o PC através do *Assembly* e,embora saibamos que ela se revestiu de grande dificuldade para muitos dos leitores, temos a certeza que irá facilitar em muito a compreensão do que se irá seguir nos próximos artigos.

José Páscoa