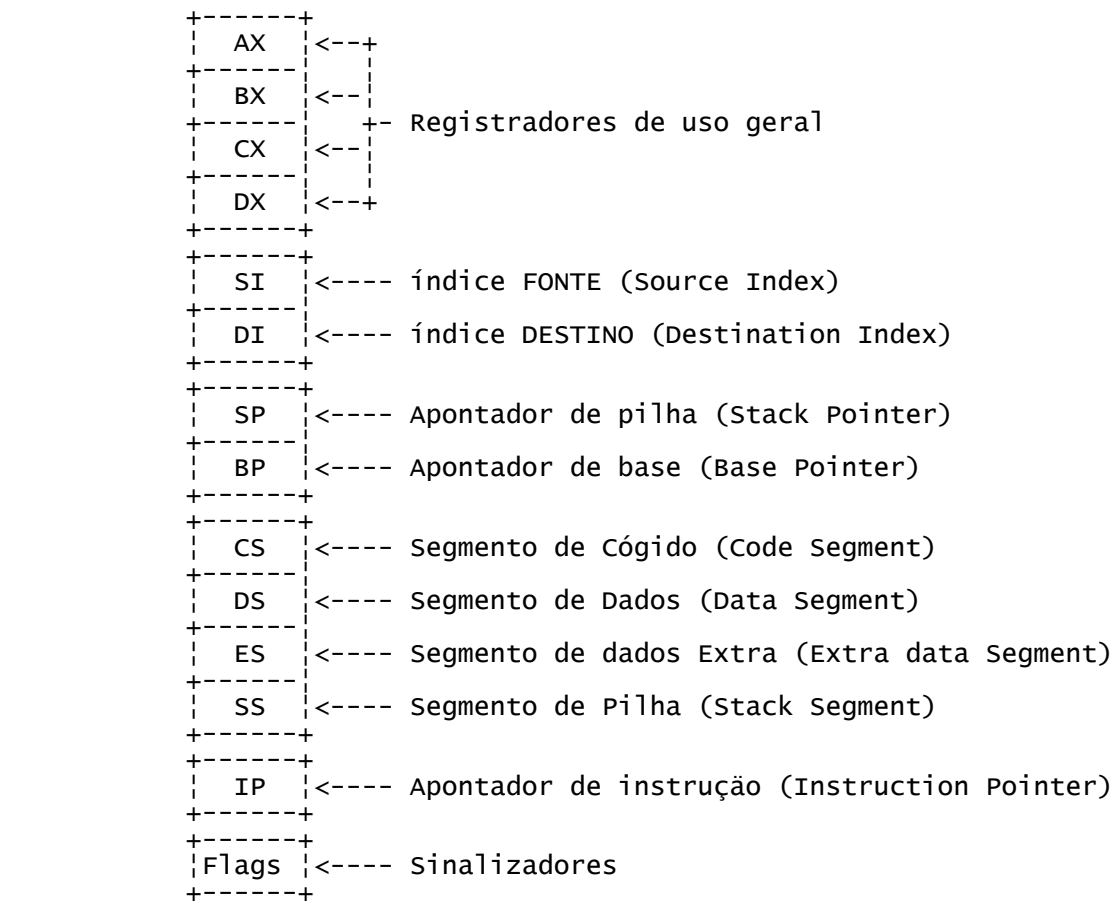
```
i----------------------------------------------©
¦ RBT   ¦   Curso de Assembly   ¦   Aula Nº 03 ¦
È----------------------------------------------¥
```

Por: Frederico Pissarra

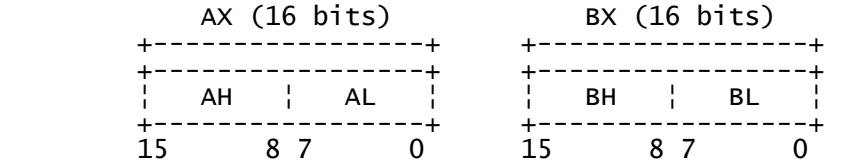
```
i--------------©
¦ ASSEMBLY III ¦
È--------------¥
```

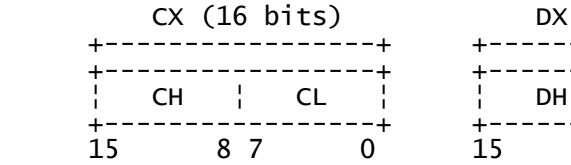
Comecemos a dar uma olhadela na arquitetura dos microprocessadores da família INTEL 80x86... Vamos aos registradores!

Entenda os registradores como se fossem variáveis que o microprocessador disponibiliza ao sistema. TODOS os registradores têm 16 bits de tamanho e aqui vai a descrição deles:



Por enquanto vamos nos deter na descrição dos registradores uso geral... Eles podem ser subdivididos em dois registradore de oito bits cada:





AH é o byte mais significativo do registrador AX, enquanto que AL é o menos significativo. Se alterarmos o conteúdo de AL, estaremos alterando o byte menos significativo de AX ao mesmo tempo... Não existem registradores de oito bits em separado... tudo é uma coisa só. Portanto, ao manipularmos AH, estaremos manipulando AX ao mesmo tempo!

O nome de cada registrador tem o seu sentido de ser... "A" de AX quer dizer que este registrador é um "acumulador" (usado por default em algumas operações matematicas!), por exemplo...

AX -> Acumulador
BX -> Base
CX -> Contador
DX -> Dados

O "X" de AX significa "eXtended". "H" de AH significa "High byte".

Embora estes registradores possam ser usados sem restrições, é interessante atribuir uma função para cada um deles nos nossos programas sempre que possível... Isto facilita a leitura do código e nos educa a seguirmos uma linha de raciocínio mais concisa... Mas, se for de sua preferência não seguir qualquer padrão no uso desses registradores, não se preocupe... não haverá qualquer desvantagem nisso (Well... depende do código, as vezes somos obrigados a usar determinado registrador!).

Alguns pontos importantes quanto a esses nomes serão observados no decorrer do curso... Por exemplo, certas instruções usam AX (ou AL, ou AH) e somente ele, não permitindo o uso de nenhum outro registrador... Outras, usam CX para contar, etc... essas instruções específicas serão vistas em outra oportunidade.

Os registradores SI e DI são usados como índices para tabelas. Em particular, SI é usado para leitura de uma tabela e DI para escrita (fonte e destino... lembra algum procedimento de cópia, nao?). No entanto, esses registradores podem ser usados com outras finalidades... Podemos incluí-los no grupo de "registradores de uso geral", mas assim como alguns registradores de uso geral, eles têm aplicação exclusiva em algumas instruções, SI e DI são usados especificamente como índices em instruções que manipulam blocos (também veremos isso mais tarde!).

Os registradores CS, DS, ES e SS armazenam os segmentos onde estão o código (programa sendo executado), os dados, os dados extras, e a pilha, respectivamente. Lembre-se que a memória é segmentada em blocos de 64kbytes (dê uma olhada na primeira mensagem dessa série).

Quando nos referimos, através de alguma instrução, a um endereço de memória, estaremos nos referindo ao OFFSET dentro de um segmento. O registrador de segmento usado para localizar o dado no offset especificado vai depender da própria instrução... Um exemplo em assembly:

```
+-----------------------------------------------------------------+
|      MOV     AL,[1D4Ah]                                         |
+-----------------------------------------------------------------+
```

O número hexadecimal entre os colchetes é a indicação de um offset em um segmento... Por default, a maioria das instruções usa o segmento de dados (valor em DS). A instrução acima é equivalente a:

```
+-----------------------------------------------------------------+
|      AL = DS:[1D4Ah]                                            |
+-----------------------------------------------------------------+
```

Isto é, em AL será colocado o byte que está armazenado no offset 1D4Ah do segmento de dados (valor em DS). Veremos mais sobre os segmentos e as instruções mais tarde :)

Se quisessemos localizar o byte desejado em outro segmento (mas no mesmo offset) devemos especificar o registrador de segmento na instrução:

```
+-----------------------------------------------------------------+
|      MOV     AL,ES:[1D4Ah]                                      |
+-----------------------------------------------------------------+
```

Aqui o valor de ES será usado.

O registrador IP (Instruction Pointer) é o offset do segmento de código que contém a próxima instrução a ser execuatda. Este registrador näo é acessível por qualquer instrução (pelo menos näo pelas documentadas pela Intel)... é de uso interno do microprocessador. No entanto existem alguns macetes para conseguirmos obter o seu conteúdo (o que na maioria das aplicações näo é necessario... Para que conhecer o endereço da próxima instrução se ela var ser executada de qualquer jeito?).

O registrador SP é o offset do segmento SS (segmento de pilha) onde o próximo dado vai ser empilhado. A pilha serve para armazenar dados que posteriormente podem ser recuperados sem que tenhamos que usar um dos registradores para esse fim. Também é usada para armazenar o endereço de retorno das sub-rotinas. A pilha "cresce" de cima para baixo, isto é, SP é decrementado cada vez que um novo dado é colocado na pilha. Note também que existe um registrador de segmento exclusivo para a pilha... SP sempre está relacionado a esse segmento (SS), como foi dito antes.

Para ilustrar o funcionamento da pilha, no gráfico abaixo simularemos o empilhamento do conteúdo do registrador AX através da instrução:

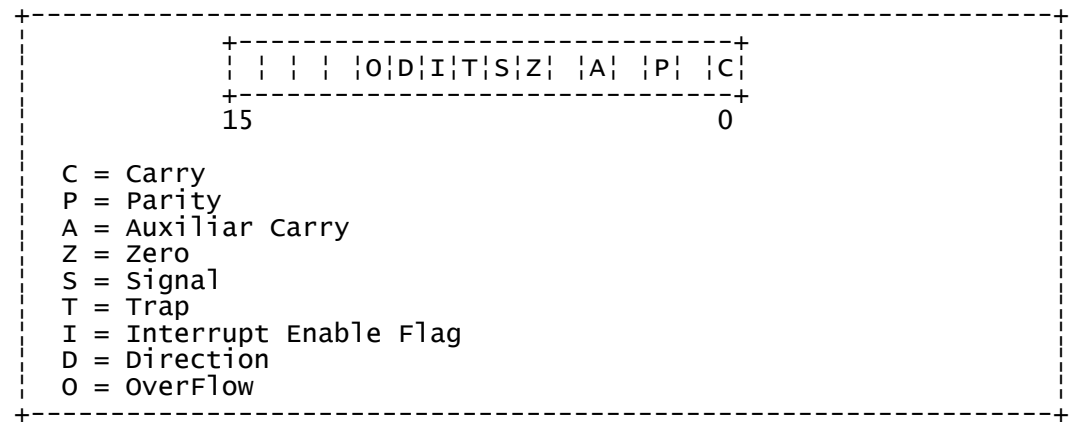
```
+-----------------------------------------------------------------+
|      PUSH    AX                                                 |
+-----------------------------------------------------------------+

+-----------------------------------------------------------------+
|  AX = A527h (Valor em AX)                                       |
|                                                                 |
|   +-------+                        +-------+                   |
|   |  ????h |<----- SP = n          |  A527h |                   |
|   +-------+                        +-------+                   |
|   |       |                        |       |<----- SP = n - 1  |
|   +-------+                        +-------+                   |
|                                                                 |
| (antes de PUSH AX)               (depois de PUSH AX)            |
```

Observe que SP sempre aponta para um espaço vago na pilha.

Na realidade SP é decrementado de duas posiçöes ao invés de apenas uma... mas, esse detalhe deixo para mais tarde.

O registrador BP pode ser usado como apontador para a base da pilha (já que, por default, está relacionado a SS) ou como um registrador de uso geral... depende do seu programa. Veremos isso detalhadamente mais tarde.

Um dos registradores mais importantes de qualquer microprocessador é o de "Flags". Eis uma descrição dos bits deste registrador (a descrição abaixo aplica-se ao 8086. Normalmente näo acessamos diretamente o registrador de flags - embora possamos fazê-lo - por isso näo é conveniente assumirmos que os bits estäo sempre no mesmo lugar para qualquer microprocessador da família 80x86!):



¦ Carry:

Esse flag é setado sempre quando houver "vai um" depois de uma adição ou quando há BORROW depois de uma subtração. Ou quando, numa operação de deslocamento de bits, o bit mais ao extremo for deslocado para fora do dado (suponha um byte... se todos os bits forem deslocados em uma posição para a direita, o que acontece com o bit 0?... Resposta: Vai para o carry!)

¦ Parity:

Depois de uma instrução aritimética ou lógica este bit informa se o resultado tem um número par de "1"s ou näo.

¦ Auxiliar Carry:

Igual ao carry, mas indica o "vai um" no meio de um dado (no caso de um byte, se houve "vai um" do bit 3 para o bit 4!).

¦ Zero:

Depois de uma operação aritimética ou lógica, esse flag indica se o resultado é zero ou näo.

¦ Signal:

Depois de uma instrução aritimética ou lógica, este flag é

uma cópia do bit de mais alta ordem do resultado, isto é, seu sinal (dê uma olhada na "representaçäo de números negativos em binário" no texto anterior!).

¦ Trap:

Quando setado (1) executa instruçöes passo-a-passo... Não nos interessa estudar esse bit por causa das diferenças de implementaçäo deste flag em toda a família 80x86.

¦ Interrupt Enable Flag

Habilita/Desabilita o reconhecimento de interrupçöes mascaráveis pela CPU. Sobre interrupçöes, veremos mais tarde!

¦ Direction:

Quando usamos instruçöes de manipulaçäo de blocos, precisamos especificar a direçäo que usaremos (do inicio para o fim ou do fim para o inicio).
Quando D=0 a direçäo é a do início para o fim... D=1, então a direçäo é contrária!

¦ OverFlow:

Depois de uma instruçäo aritimética ou lógica, este bit indica se houve mudança no bit mais significativo, ou seja, no sinal. Por exemplo, se somarmos FFFFh + 0001h obteremos 00h. O bit mais significativo variou de 1 para 0 (o counteúdo inicial de um registrador era FFFFh e depois da soma foi para 0000h), indicando que o resultado saiu da faixa (overflow) - ora, FFFFh + 0001h = 10000h, porém um registrador tem 16 bits de tamanho e o resultado cabe em 17 bits. Neste exemplo, o bit de carry também será setado pois houve "vai um" do bit 15 para o inexistente bit 16, mas näo confunda o flag de overflow com o carry!

Quando aos demais bits, näo se pode prever seus estados lógicos (1 ou 0).

Na próxima mensagem começaremos a ver algumas instruçöes do microprocessador 8086. Ainda näo escreveremos nenhum programa, a intençäo é familiarizá-lo com a arquitetura do microprocessador antes de começarmos a colocar a mäo na massa... tenha um pouco de paciência! :)