

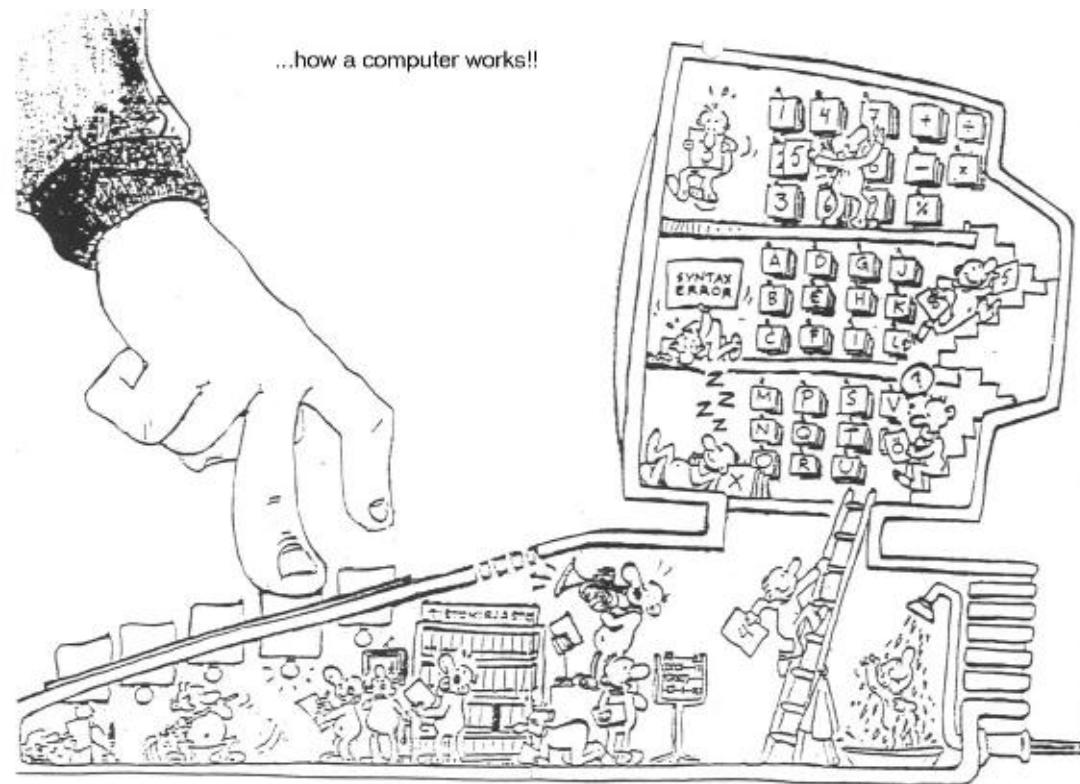


Instituto Superior de  
Engenharia de Coimbra

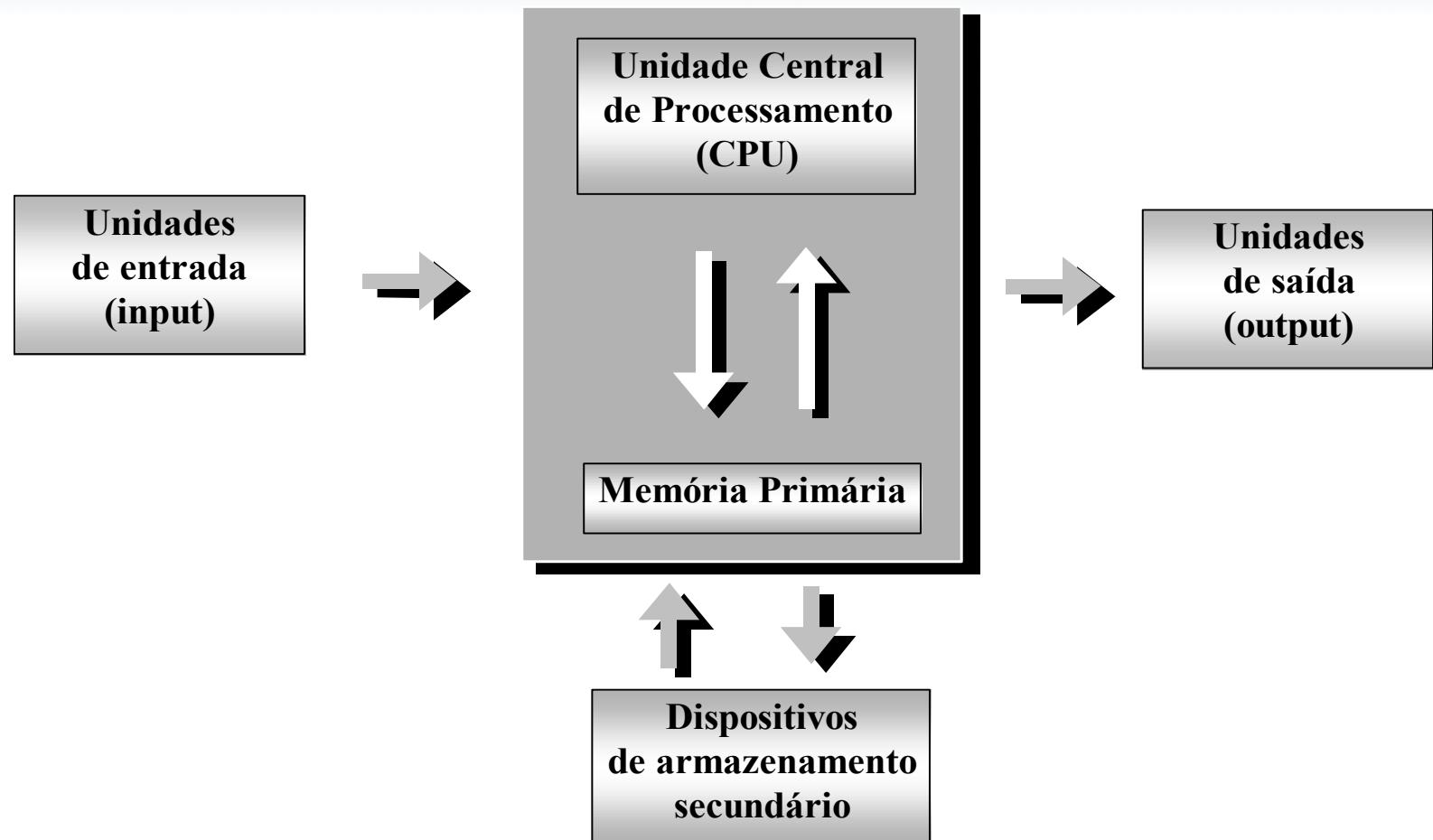
# TECNOLOGIAS E ARQUITECTURAS DE COMPUTADORES - LINGUAGEM ASSEMBLY



# COMO FUNCIONA UM COMPUTADOR?



# MODELO DE VON NEUMANN



- ◎ Circuito integrado que contém milhões de componentes electrónicos elementares, organizados de modo a poderem executar instruções.
- ◎ Realiza as funções de processamento e cálculo, constituindo a peça mais importante de um computador.

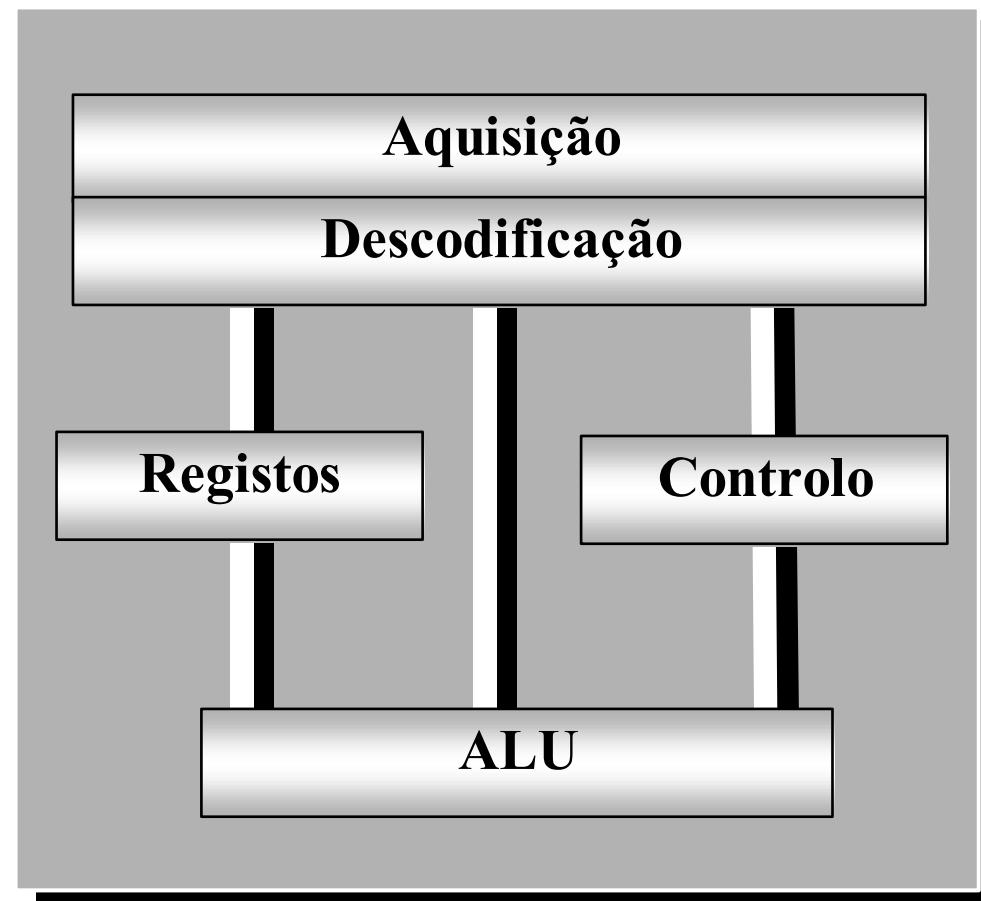
- ◎ Cada comando da CPU é codificado com a presença ou ausência de um conjunto de sinais eléctricos nos seus pinos.
- ◎ Esses sinais - cada um representando um bit de informação é codificado como um 0 ou um 1 - juntos formam uma sequência de bits.

- ◎ Determinadas sequências de bits recebem significados especiais dos projectistas do microprocessador, passando a constituir as instruções desse microprocessador.
- ◎ O conjunto de comandos que um determinado modelo de microprocessador entende, e ao qual pode reagir, é denominado conjunto de instruções ou conjunto de comandos.

## ◎ Função

- Executar programas armazenados na memória principal, acedendo às suas instruções, examinando-as e executando-as.
- Os circuitos do microprocessador têm de passar, frequentemente, por várias etapas para executar uma instrução.
  - A instrução informa o microprocessador para executar um grupo de etapas que compõem uma operação. O conjunto completo desses grupos que compõem cada instrução é o microcódigo.

- ◎ Composta por várias partes distintas
  - Secção de aquisição e descodificação de instruções
    - Onde são recebidas as instruções provindas de memórias para, em seguida, serem descodificadas de modo a que a CPU possa determinar quais as operações a realizar.
  - Secção de execução
    - Onde são processadas as instruções e dados recebidos. É constituída por:
      - Unidade de controlo
      - Unidade Aritmética e Lógica (ALU)
      - Registos



## ◎ Unidade de controlo

- Controla ou determina as operações a efectuar em cada instante, enviando sinais apropriados aos outros componentes.
- Desempenha as seguintes funções:
  - Extrai da memória, uma a uma as sucessivas instruções do programa, à custa de:
    - Instruction Pointer – envia para memória o endereço de cada uma das sucessivas instruções.
    - Instruction Register – recebe cada uma dessas instruções, retendo-a o tempo necessário para que possa ser analisada.
  - Analisa essas instruções.

- Gera sinais de comando que são enviados aos diversos componentes e permitem executar cada instrução analisada.
- Unidade Aritmética e Lógica (ALU)
  - Responsável pela execução de operações aritméticas e lógicas.
- Registros
  - Usados para armazenar resultados temporários (com que a ALU vai efectuar as operações que lhe são indicadas) e certas informações de controlo (Instruction Pointer, Stack Pointer, ...).

# GENERALIDADES

- ◎ Para executar um programa, o computador tem de ter na memória a sequência de instruções que o constituem, na forma binária.
- ◎ Existem múltiplas linguagens de programação alternativas, mas basicamente dividem-se em três tipos:
  - Linguagens máquina
  - Linguagens Assembly
  - Linguagens de alto nível

# GENERALIDADES

## ◎ Linguagem Máquina

- Esta é a linguagem entendida pelo computador. Um programa em linguagem máquina consiste numa sequência de códigos binários que correspondem às instruções que se pretende que o computador execute.
- Esta linguagem é muito pouco usada pelos programadores, uma vez que se torna muito difícil memorizar os milhares de códigos binários de instruções que um processador tem.

# GENERALIDADES

## ◎ Linguagem Assembly

- Com o objectivo de tornar mais fácil a programação sem perder o controlo do hardware, muitos programadores utilizam linguagens simbólicas - Assembly.
- A linguagem Assembly é constituída por um conjunto de instruções simbólicas que são mapeadas directamente para linguagem máquina usando assemblers (Tradutores).
- Os assemblers traduzem, uma a uma, as instruções da linguagem Assembly em instruções de linguagem máquina.

# GENERALIDADES

## ◎ Linguagens de alto nível

- Estas linguagens estão mais próximas da linguagem natural do que da linguagem máquina.
- Uma instrução numa linguagem de alto nível corresponde normalmente a várias instruções em linguagem máquina.
- O trabalho de traduzir uma linguagem na outra cabe a programas bastante complexos designados:
  - Compiladores – geram código objecto.
  - Linkers – transformam o código objecto em código máquina.

# GENERALIDADES

- ◎ O processador executa operações com operandos que na maior parte dos casos estão na memória principal.
- ◎ O processador tem zonas de memória especiais onde se podem colocar e ler dados, designadas registos.

# REGISTOS DO 8086

- ◎ No total o 8086 é composto por 14 registos de 16 bits cada, podendo ser categorizados em:
  - Uso genérico
    - Dados (AX, BX, CX e DX)
    - Índice (SI e DI)
    - Ponteiros (SP e BP)
  - Segmento (CS, DS, ES e SS)
  - Uso especial (IP e Flags)

# REGISTOS DE USO GENÉRICO - DADOS

- ◎ Aparecem normalmente como operandos de instruções aritméticas e lógicas.
  - AX – Acumulador
    - Concentra, normalmente, os resultados de algumas instruções aritméticas e lógicas.
  - BX – Base
    - É o único registo de dados que pode ser usado para gerar um endereço de memória, para aceder a dados dentro do segmento de dados. Pode funcionar como registo acumulador, mas é orientado essencialmente para a referência de determinadas células de memória.

# REGISTOS DE USO GENÉRICO - DADOS

## ● CX – Contador

- Contém, normalmente, o número de vezes que se quer repetir uma instrução. Pode funcionar como acumulador, mas é essencialmente utilizado como contador.

## ● DX – Dados

- Tem 2 propósitos especiais: manter o overflow de certas operações aritméticas (funcionando como acumulador) e manter os endereços de I/O quando se acede a dados no bus de I/O.

# REGISTOS DE USO GENÉRICO

- ◎ Apesar de todos os registos deste processador serem de 16 bits, os registos de dados podem ser utilizados como registos de oito bits. Utilizando para isso o byte mais significativo e o byte menos significativo de cada registo.

# REGISTOS DE USO GENÉRICO

AX



BX



CX



DX



SI



DI



BP



SP



# REGISTOS DE USO GENÉRICO-ÍNDICE

- ◎ São usados para armazenar endereços de offset (deslocamento) dentro de segmentos, onde os dados devem ser acedidos.
- ◎ Podem ser usados como ponteiros para aceder à memória indirectamente.
  - SI (Source Index)
  - DI (Destination Index)
- ◎ Nota: são muito usados para manipulação de strings e zonas de memória.

# REGISTOS DE USO GENÉRICO-PONTEIROS

- ◎ Associada aos programas existe uma zona de memória especial, designada por Pilha, onde se armazena informação importante relacionada com o estado da máquina, valores temporários, endereços de retorno das funções executadas, parâmetros de funções e variáveis locais.
- ◎ Para acesso a essa memória são disponibilizados os registos SP (Stack Pointer) e BP (Base Pointer).

# REGISTOS DE USO GENÉRICO-PONTEIROS

- ◎ SP (Stack Pointer)
  - É um indicador que aponta sempre para o topo da pilha.
  - A correcta operação de um programa depende, muitas vezes, da correcta utilização deste registo.
- ◎ BP (Base Pointer)
  - Utilizado para a constituição do endereço base da pilha.
  - Utiliza-se normalmente para aceder a parâmetros e variáveis locais, de um procedimento.

# REGISTOS DE SEGMENTO

- ◎ O 8086 possui 4 registos de segmento:
  - CS (Code Segment)
  - DS (Data Segment)
  - ES (Extra Segment)
  - SS (Stack Segment)
- ◎ Têm como função a selecção de blocos (segmentos) da memória principal.
- ◎ Um registo de segmento aponta para o início de um segmento de memória.

# REGISTOS DE SEGMENTO

## ◎ CS (Code Segment)

- É utilizado para obter o endereço físico das instruções. Faz-se acompanhar pelo IP (Instruction Pointer) que se auto incrementa quando as instruções são executadas. Aponta para o segmento que contém as instruções máquina correntemente em execução.

# REGISTOS DE SEGMENTO

- ◎ DS (Data Segment)
  - É utilizado para obter o endereço físico da zona de dados. Geralmente aponta para a zona onde estão armazenadas as variáveis globais do programa. É sempre possível mudar o valor do registo DS para aceder a dados adicionais noutros segmentos.

# REGISTOS DE SEGMENTO

- ◎ SS (Stack Segment)
  - É utilizado em operações envolvendo a pilha.  
Aponta para o segmento que contém a pilha.
- ◎ ES (Extra Segment)
  - É utilizado em operações com strings. Além disso, serve como apontador para um segmento de reserva para programas que necessitem de zonas de dados muito grandes.

# REGISTOS DE USO ESPECIAL

- ◎ Os registos IP e de flags não são acedidos da mesma forma que os outros registos, em vez disso, é a CPU que geralmente os manipula directamente.
- IP (Instruction Pointer)
  - Indica o Offset (dentro do segmento de código corrente) da próxima instrução a executar.

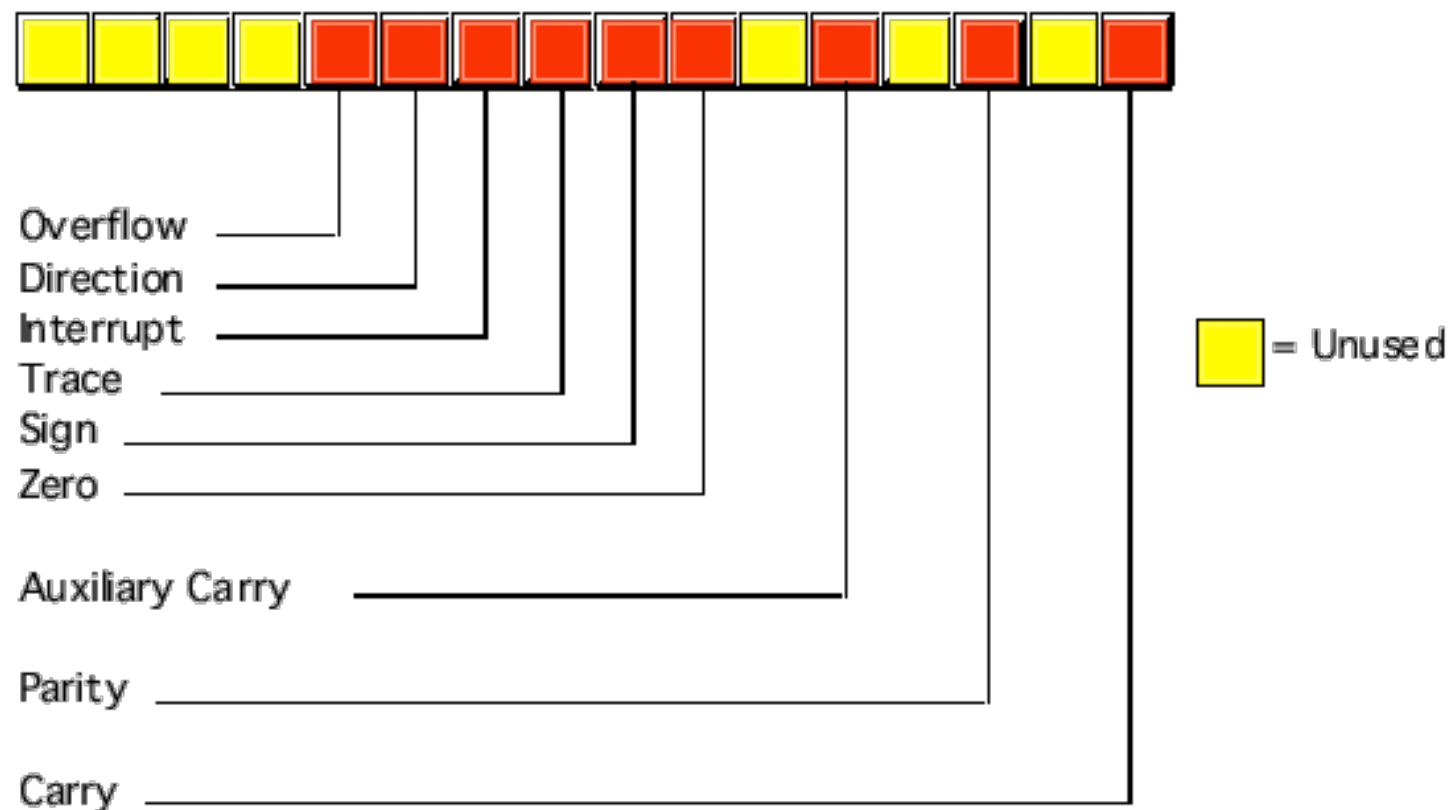
# REGISTOS DE USO ESPECIAL-IP

- ◎ O registo IP associa-se sempre ao CS para formar o endereço de memória onde se encontra a instrução.
- ◎ O registo IP não é directamente acessível ao programador, sendo alterado indirectamente pelas instruções JMP, CALL, RET, INT e IRET.

# REGISTOS DE USO ESPECIAL-FLAGS

- ◎ É um registo especial que reflecte em cada um dos seus bits situações que ocorrem no interior do processador durante as operações aritméticas, lógicas entre outras.
- ◎ É constituído por 9 flags que estão agrupadas, para facilidade de acesso, num registo de 16 bits, chamado Registo de Flags.

# REGISTROS DE USO ESPECIAL-FLAGS



# REGISTOS DE USO ESPECIAL-FLAGS

- ◎ O (Overflow) – Indica um “transbordo” do penúltimo para o último bit do resultado.
- ◎ D (Direction) – Fixa a direcção do progresso em operações sobre strings.
- ◎ I (Interrupt) – Permite a passagem de sinais de interrupção.
- ◎ T (Trace) – Força o processador a executar uma rotina de Trace após cada instrução.
- ◎ S (Sign) – Indica o sinal do resultado.

# REGISTOS DE USO ESPECIAL-FLAGS

- ◎ Z (Zero) – Indica um resultado de zero.
- ◎ A (Aux. Carry) – Transbordo entre os grupos de 4 bits, para aritmética BCD.
- ◎ P (Parity) – Indica (é igual a 1) se existe um número par de bits no resultado.
- ◎ C (Carry) - Transbordo das operações numéricas para fora do limite do resultado.

# SEGMENTOS DE UM PROGRAMA

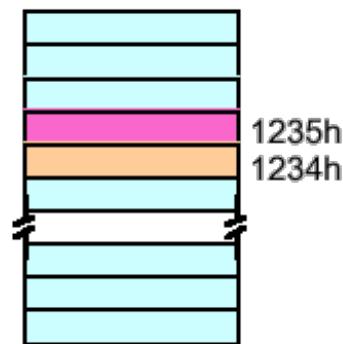
- ◎ Um programa em Assembly possui 3 zonas principais de memória, às quais correspondem 3 segmentos:
  - Segmento de dados (Data Segment)
    - Zona onde são colocados os dados usados no programa.
  - Segmento de código (Code Segment)
    - Zona destinada ao código/instruções do programa.
  - Segmento de pilha (Stack Segment)
    - Zona destinada ao:
      - armazenamento dos endereços de retorno aquando da chamada de procedimentos;
      - armazenamento temporário de dados.

# COMO ACEDER AOS DADOS

- ◎ As diferentes formas que o microprocessador utiliza para aceder aos dados chamam-se modos de endereçamento.
- ◎ Na linguagem Assembly o modo de endereçamento usado é indicado na própria instrução.
- ◎ Antes porém de abordar os diversos modos de endereçamento do 8086 convém estudar o seu processo de segmentação.

# SEGMENTAÇÃO

- ◎ A memória assemelha-se a um array linear de bytes.
- ◎ Um único índice (endereço) selecciona determinado byte desse array. Este tipo de endereçamento é designado de linear ou em flat.



# SEGMENTAÇÃO

- ◎ No 8086 o bus de endereços é de 20 bits, permitindo endereçar  $2^{20}$  bytes, ou seja 1 Mbyte.
- ◎ No 8086 os registos são de apenas 16 bits, o que torna impossível referenciar todos os bytes em memória com apenas um registo.
- ◎ O 8086 utiliza um esquema denominado segmentação para aceder a 1 Mbyte completo de memória, com registos de apenas 16 bits.

# SEGMENTAÇÃO

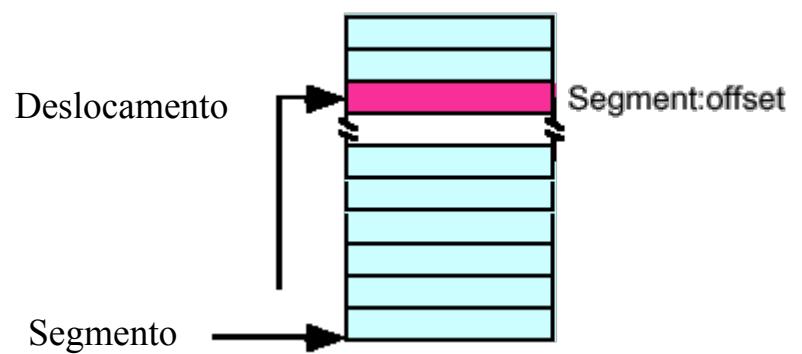
- ◎ Assim, em vez de um registo, são utilizados dois registos. Um desses registos aponta para um valor de memória chamado base do segmento, enquanto o outro determina o deslocamento em relação a essa base num valor máximo de 64 Kbytes.
- Os registos que representam bases de segmento são os registos de segmento (CS, DS, SS e ES).
- Os registos que representam o deslocamento são os registos de base (BX, BP e SP), de índice (SI e DI) e o IP.

# SEGMENTAÇÃO

- ◎ Os registos de deslocamento trabalham associados aos registos de segmento.
  - O IP (Instruction Pointer) está associado ao registo CS.
  - O SP (Stack Pointer) e o BP (Base Pointer) estão associados ao registo SS.
  - Os registos BX, SI e DI estão sempre associados aos registos DS e ES.

# SEGMENTAÇÃO

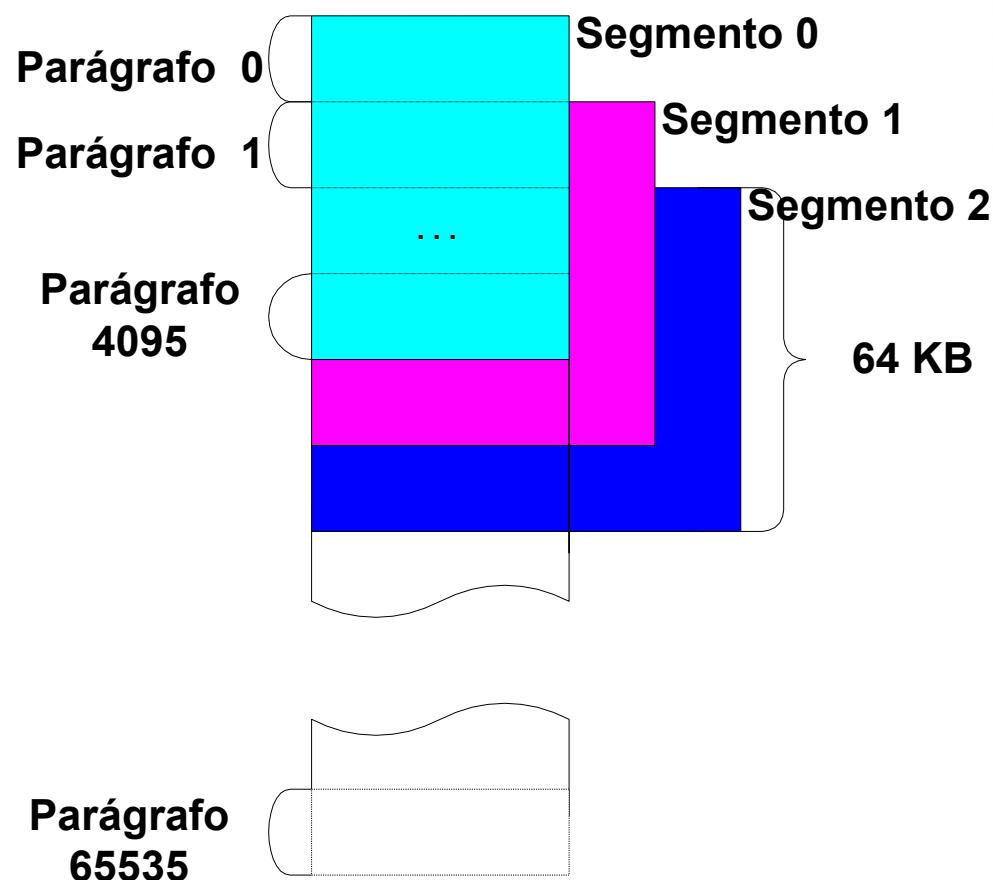
- ◎ Em resumo, o endereçamento segmentado usa 2 componentes para especificar uma localização de memória: um valor de segmento e um deslocamento dentro deste segmento.
- Este par é normalmente representado por segment:offset.



# SEGMENTAÇÃO

- ◎ O tamanho do offset limita o tamanho máximo de um segmento. No 8086, com offsets de 16 bits, um segmento não pode ser maior do que 64 Kbytes.
- ◎ A parte do segmento é de 16 bits, o que permite a um programa ter até 65536 segmentos diferentes.
  - Esses segmentos, no entanto, não são disjuntos.
  - Pensando em memória linear (a realidade da memória física) a cada novo conjunto de 16 bytes (Parágrafo) corresponde o início de um novo segmento.

# SEGMENTAÇÃO



# SEGMENTAÇÃO

- ◎ Um endereço representado na forma segment:offset tem o nome de endereço lógico.
- ◎ O actual endereço linear que aparece no bus de endereços é o endereço físico ou real.
- ◎ Existe uma função que mapeia o endereço lógico ou virtual num endereço físico ou real.
  - A CPU multiplica o valor do segmento por 16 (10h) e adiciona o offset a este resultado.

- ⦿ Qual o endereço real do endereço lógico 1000:1F00?

$$\begin{array}{r}
 1000:1F00 \\
 \downarrow \\
 10000 \\
 + 1F00 \\
 \hline
 11F00
 \end{array}$$

# SEGMENTAÇÃO

- ◎ Problema: pode surgir uma multiplicidade de representações para referir dados em memória.
  - Ex: 11F0:0, 1100:F00, 1080:1700, são diferentes endereços de memória que se referem ao mesmo endereço físico 11F00h.
- ◎ Solução: Normalização de endereços.

# NORMALIZAÇÃO DE ENDEREÇOS

- ◎ A parte do segmento pode conter qualquer valor de 16 bits.
- ◎ A parte do deslocamento tem de ser um valor na gama 0...0Fh.
- ◎ Normalizando o endereço físico 11F00 do endereço lógico 1000:1F00, fica 11F0:0H.

# MODOS DE ENDEREÇAMENTO DO 8086

- ◎ Endereçamento por Registo - é passado como parâmetro um registo.
  - Ex: MOV AX, BX
- ◎ Endereçamento Imediato
- ◎ Endereçamento por Memória
  - Endereçamento Directo
  - Endereçamento Indirecto por Registo
    - Indexado
    - Baseado
    - Indexado Baseado

# ENDEREÇAMENTO IMEDIATO

- ◎ Endereçamento Imediato: é passado como parâmetro o dado pretendido.
- Ex1: MOV AX, 8B67h

AX  
8B67h

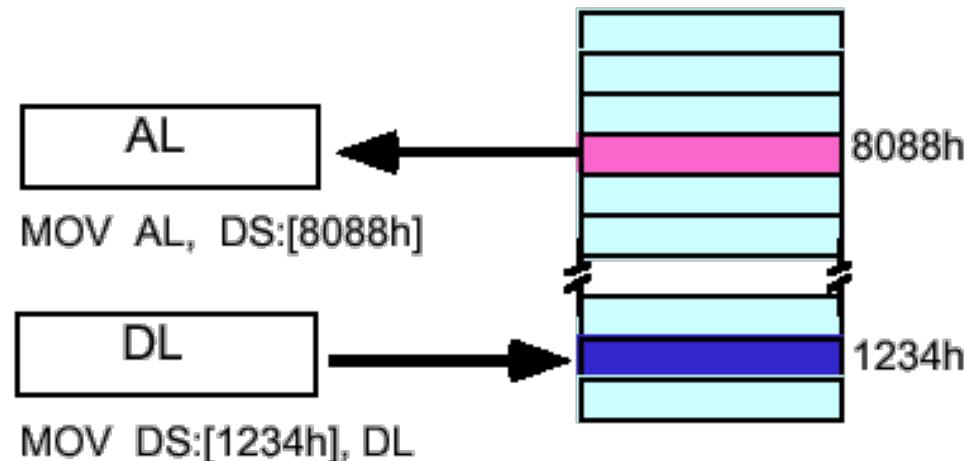
# ENDEREÇAMENTO DIRECTO

- ◎ Endereçamento Directo: é passado como parâmetro o endereço do dado em questão. Consiste na passagem de uma constante de 16 bits que especifica o endereço da localização destino.
  - Ex1: MOV AX, [8B67h]
  - Ex2: MOV AL, DS:[8088h]  
MOV AL, [8088h]

# ENDEREÇAMENTO DIRECTO

Ex3: MOV DS:[1234h],DL

MOV [1234h],DL



# ENDEREÇAMENTO DIRECTO

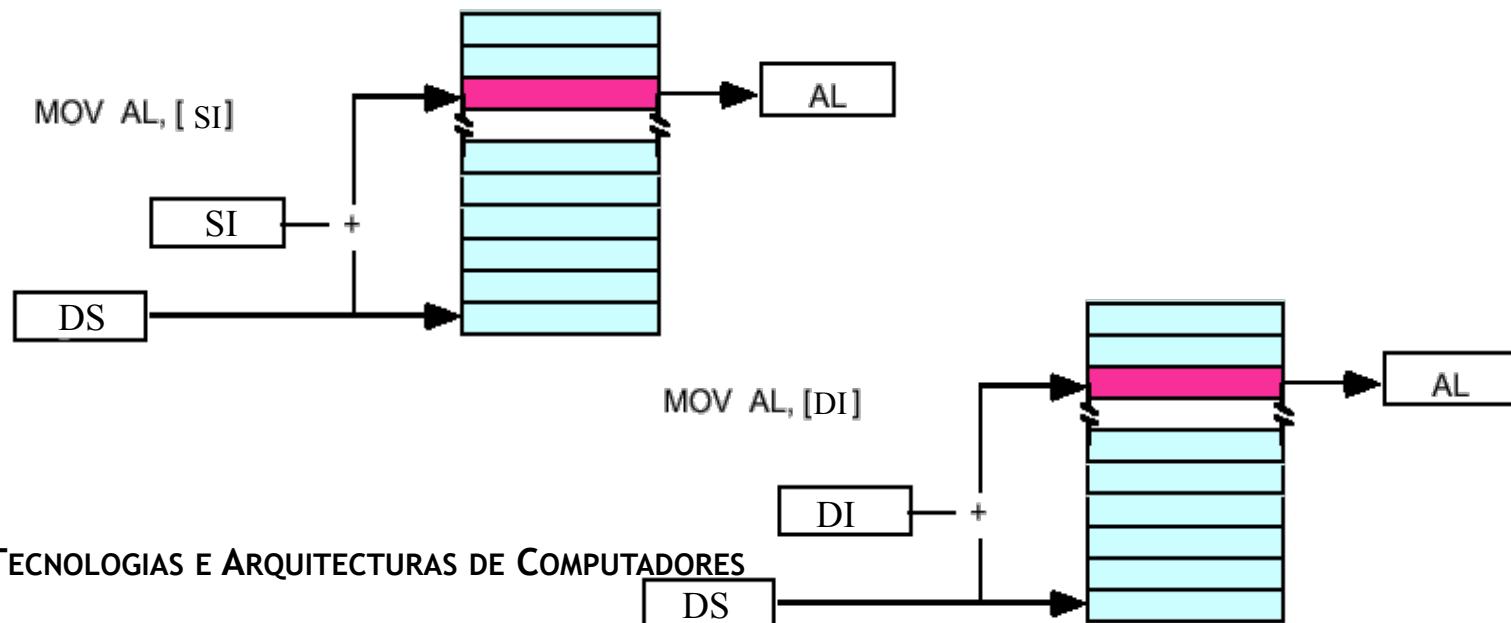
- ◎ Por default todos os valores de deslocamento se referem ao segmento de dados. Caso se pretenda fornecer deslocamentos num segmento diferente há que usar o prefixo de segmento antes do endereço.
  - MOV AX, CS:[1234h]

# ENDEREÇAMENTO INDIRECTO POR REGISTO

- ◎ Endereçamento Indirecto por Registo: é passado como parâmetro um registo que contém o endereço de dados.
- ◎ Existem 4 formas deste modo de endereçamento, no 8086:
  - `MOV AL, [BX]`
  - `MOV AL, [BP]*`
  - `MOV AL, [SI]`
  - `MOV AL, [DI]`

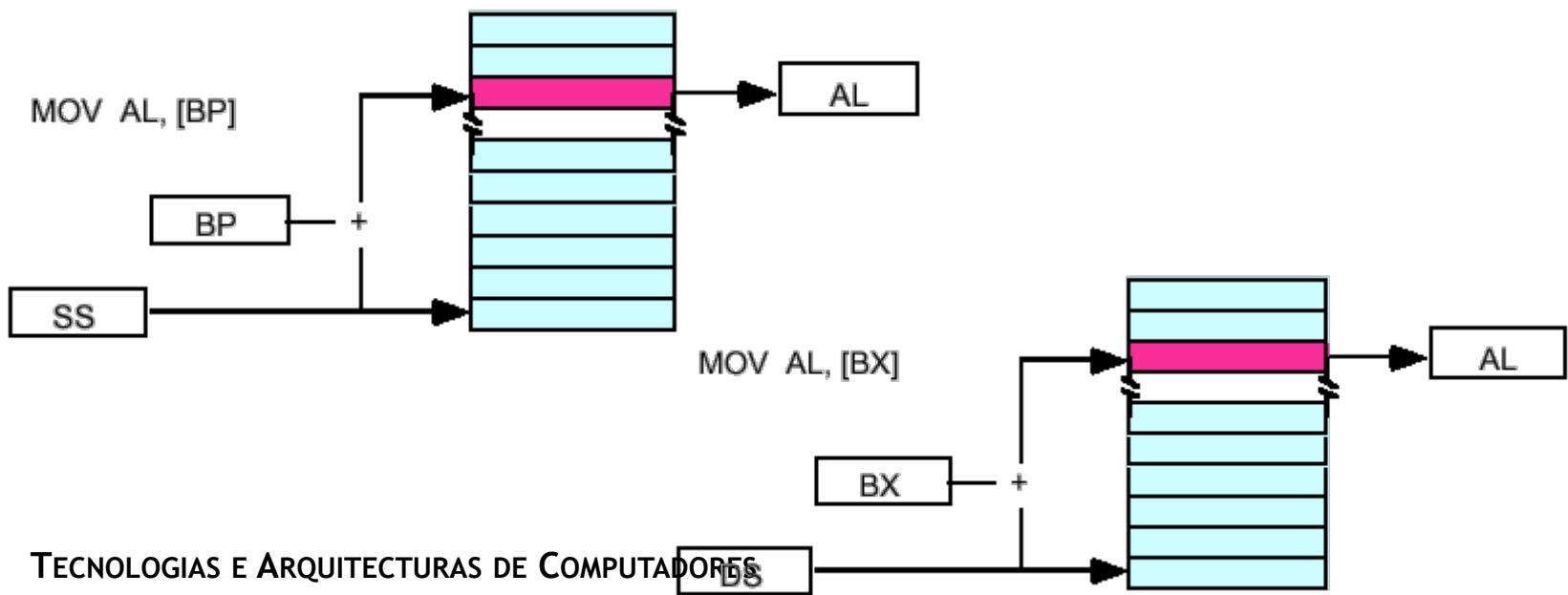
# ENDEREÇAMENTO INDEXADO

- Endereçamento Indexado: é passado como parâmetro um registo de índice (SI ou DI). O valor a usar está no endereço cujo valor está contido no registo.



# ENDEREÇAMENTO BASEADO

- Endereço Baseado: é passado como parâmetro um registo de base (BX ou BP). O valor a usar está no endereço cujo valor está contido no registo.

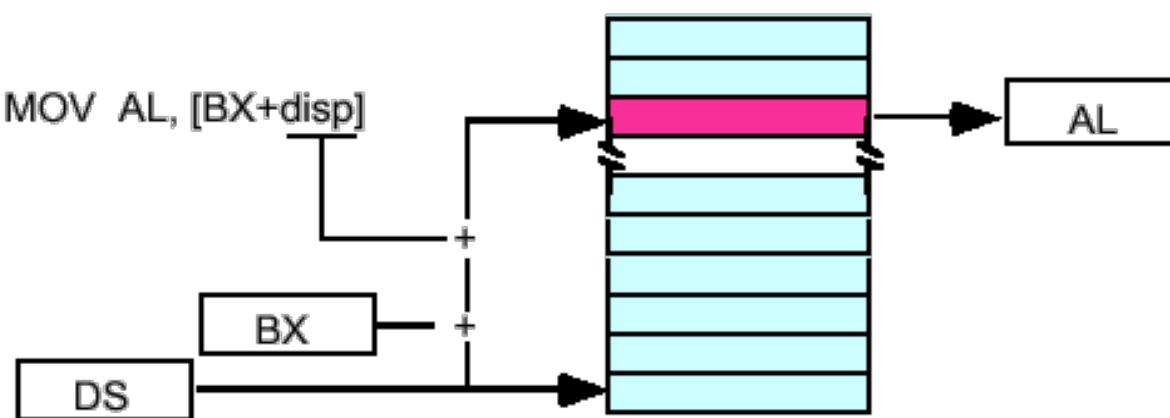


# ENDEREÇAMENTO INDEXADO/ BASEADO

- ◎ Existe uma grande semelhança entre os modos de endereçamentos Indexado e Baseado. O cálculo do offset final é idêntico, variando apenas os registos utilizados.
- ◎ Existe ainda uma variação destes modos de endereçamento que consiste na adição de uma constante/deslocamento (displacement) ao registo especificado.

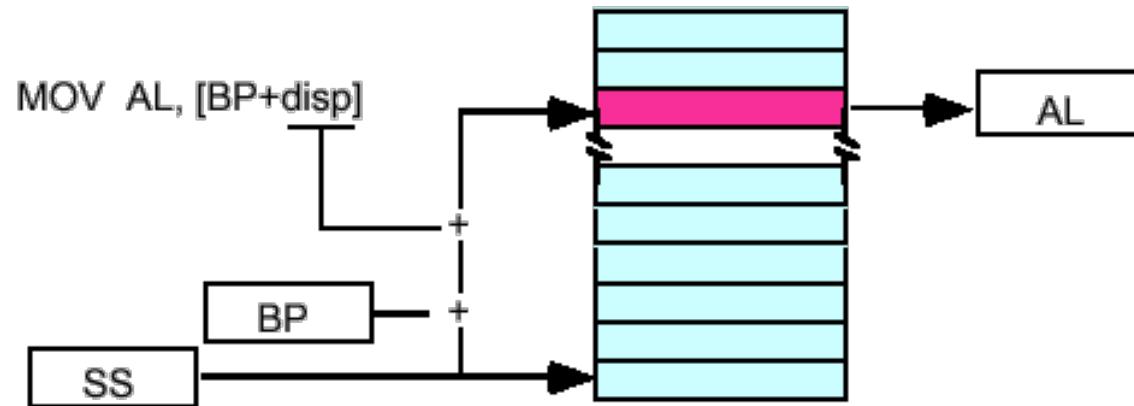
# ENDEREÇAMENTO INDEXADO/BASEADO COM DESLOCAMENTO

- MOV AL, disp [BX]
- MOV AL, disp [SI]
- MOV AL, disp [DI]



# ENDEREÇAMENTO INDEXADO/BASEADO COM DESLOCAMENTO

- MOV AL, disp [BP]



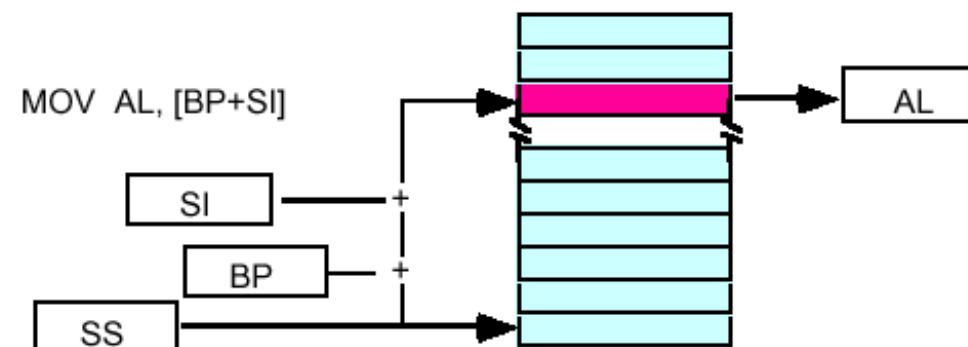
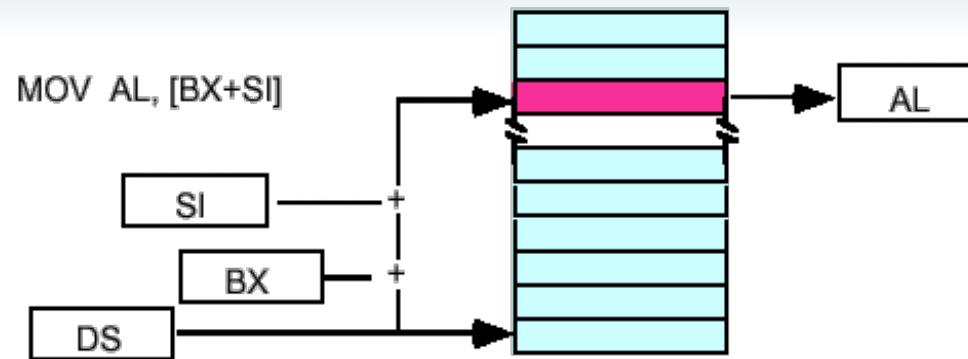
# ENDERECAMENTO INDEXADO BASEADO

- ◎ Endereçamento Indexado Baseado: é passado como parâmetro um par de registos, um de base e um de índice. O valor do dado está no endereço obtido pela soma destes dois elementos. Este modo de endereçamento consiste apenas numa combinação dos dois modos anteriores.

# ENDERECAMENTO INDEXADO BASEADO

- ◎ Este modo de endereçamento calcula o offset adicionando um registo de base (BX ou BP) e um registo de índice (SI ou DI). As combinações possíveis são:
  - MOV AL, [BX] [SI]
  - MOV AL, [BX] [DI]
  - MOV AL, [BP] [SI]
  - MOV AL, [BP] [DI]

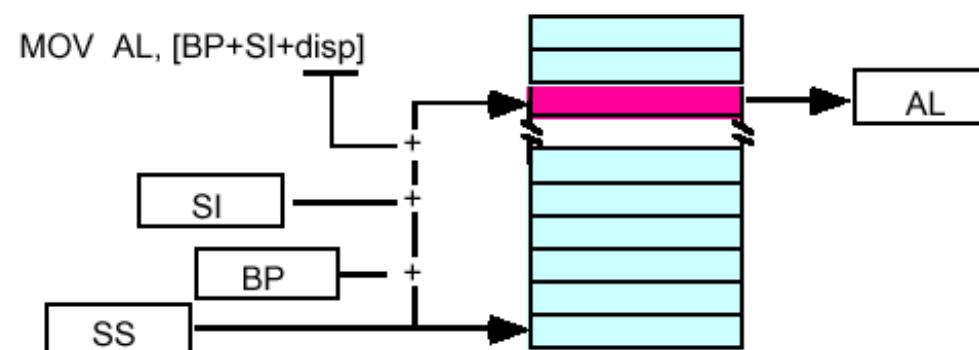
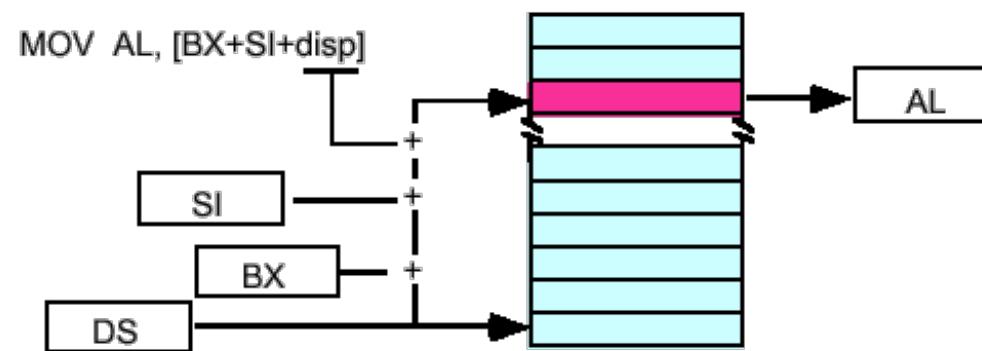
# ENDERECAMENTO INDEXADO BASEADO



# ENDEREÇAMENTO INDEXADO BASEADO COM DESLOCAMENTO

- ◎ Endereçamento Indexado Baseado com Deslocamento: este modo de endereçamento consiste numa ligeira modificação do modo de endereçamento indexado baseado com a adição de um valor constante de 8 ou de 16 bits. Ex:
  - MOV AL, disp [BX] [SI]
  - MOV AL, disp[BX + DI]
  - MOV AL, [BP + SI + disp]
  - MOV AL, [BP] [DI] [disp]

# ENDEREÇAMENTO INDEXADO BASEADO COM DESLOCAMENTO



# MODOS DE ENDEREÇAMENTO VÁLIDOS

- ◎ Os modos de endereçamento válidos resultam das possíveis combinações (não contando as variações sintácticas) entre os registos abaixo:

DISP	[BX]	[SI]
	[BP]	[DI]