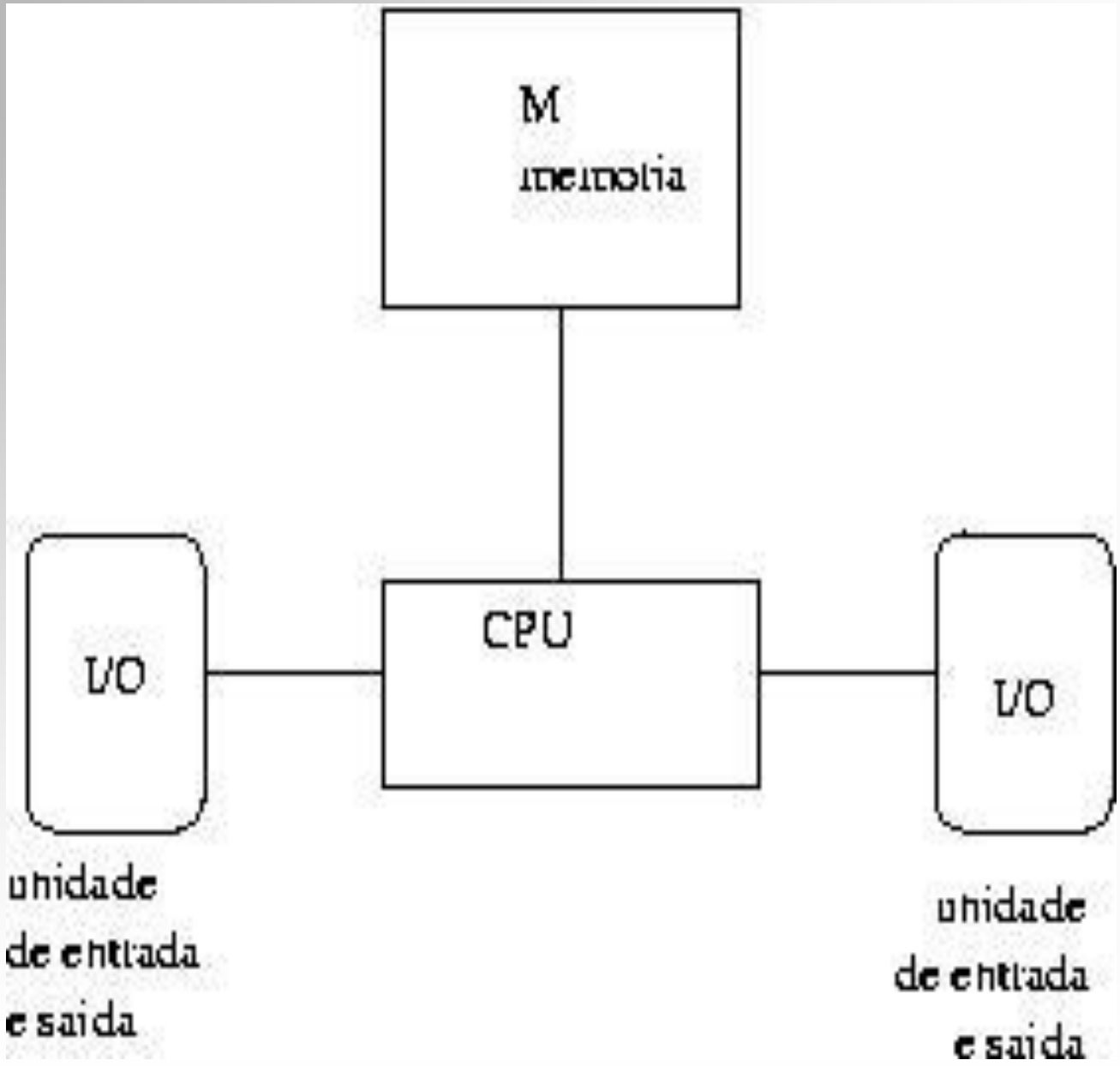
Tópicos sobre Arquitectura de Computadores -- 3

José A. Cardoso e Cunha
DI-FCT/UNL

1. Arquitectura de Von Neumann
2. Programação em linguagem ´´máquina´´
3. Sistema de entradas e saídas
4. Hierarquia das unidades de memória
5. Organização interna do processador

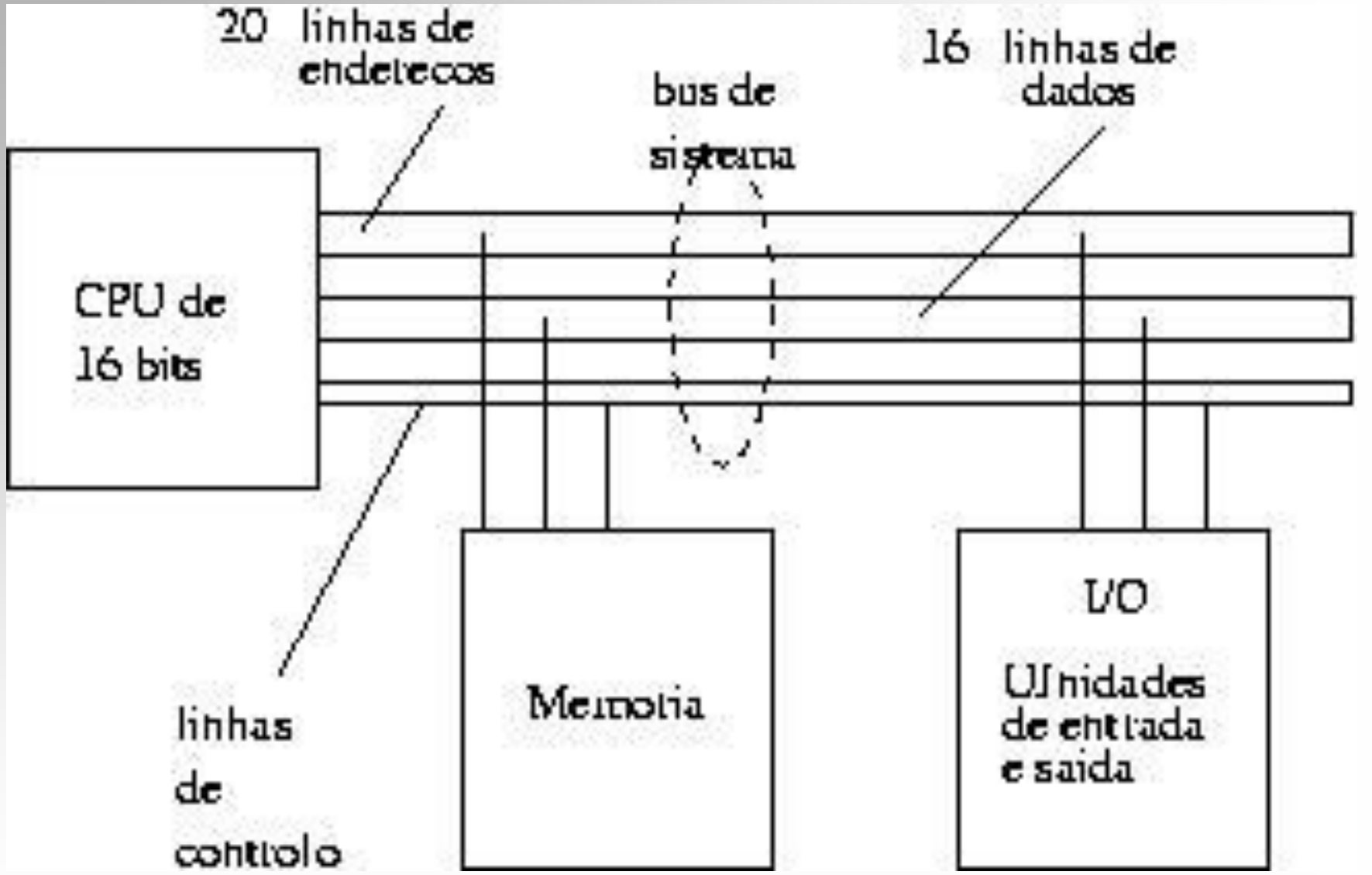

M
I/O
CPU
I/O
unidade
de entrada
e saida
unidade
de entrada
e saida

Processadores Intel: uma evolução ao longo de 30 anos

- **1968: Intel → unidades (*chips*) de memória**
- **Dois principais projectos:**
  - **1 *chip* CPU para uma calculadora**
  - **1 *chip* controlador para um *terminal* (teclado/écran)**
  - → **Microprocessadores (CPU integrado em *chip*) genérico (modelo de Von Neumann)**
    - → **1971: 4004 (4 bits) Bus: 4 linhas dados; 4KBytes de memória, *45 instruções, ciclo da ordem de 20 micro-segundos (50 KIPS…)***
    - → **8008 (8 bits), Bus: 8 linhas dados; 16 KBytes de memória, *48 instruções***
- → **1974/75: 8080/8085 (8 bits), *246 instruções, ciclo da ordem de 2 micro-segundos (500 KIPS…)***
  - → **Bus: 8 linhas dados;**
  - → **Bus: 16 de endereços: 64 KBytes de memória**

- **1978: 8086**
  - **CPU 16 bits,** ***imensas variantes de instruções, ciclo da ordem de 400 nano-segundos (2.5 MIPS…); Registadores dedicados***
  - **Bus 20 linhas de endereços; 16 linhas de dados**
  - **Segmentos de memória (4)**
- **1980: 8087 co-processador vírgula flutuante, (***+ 64 instruções, operando sobre uma pilha)*
- **1980: 8088**
  - **CPU 16 bits, Bus 20 linhas de endereços; 8 linhas de dados (mais lento, mais barato do que 8086, compatível com periféricos com interfaces de 8 bits; base original do IBM PC)**
- **8086/8088: Espaços de endereços reais: ambos limitados a 1 MegaByte**

  (grande impacto: sistema DOS - ***Disk Operating System*****) (+ 80188/186)**


20 linhas de endereços
bus de sistema
16 linhas de dados
CPU de 16 bits
linhas de controlo
Memória
I/O
Unidades de entrada e saída

- **1982: 80286**
  - Compatível com 8086, ***ciclo da ordem de 250 nano-segundos (4 MIPS…)***
  - Bus com 24 linhas de endereços: → 16 MegaBytes
  - Bus com 16 linhas de dados
    (surgiu no IBM PC AT e PS/2)
  - Novos modelos de memória (**2^{14} segmentos**)+ protecção
- **1985: 80386 → IA-32**
  - **CPU de 32 bits, Bus com 32 linhas de endereços: → 4 GigaBytes, Bus com 32 linhas de dados**
  - Modo de operação compatível com modelos anteriores, novos modos ender. / instruções, melhorias na arquitectura
- **1989: 80486**
  - Compatível com modelos anteriores, Melhorias de desempenho, maior grau de integração, novas funcionalidades na arquitectura (vírgula flutuante, cache, multiprocessadores), **originalmente ciclo ~ 25 nanoseg. (40 MIPS…)**
- **… 80586 … Pentium …**

- **1993: Pentium**
  - Bus: 32 linhas de endereços, **64 linhas de dados**
  - Melhorias ao modelo de Von Neumann: execução de 2 instruções em paralelo, pelo CPU; + suporte para multiprocessadores
- **1995: Pentium PRO (P6)**
  - Até 3 instruções em paralelo, pelo CPU
  - Bus: 36 linhas de endereços reais (até 64 GigaBytes)
  - Melhorias ao nível da arquitectura
- **1989-1995: 80486/Pentium/Pentium Pro**: **têm só mais 4 novas instruções do que modelos anteriores…**
- **1997: +MMX** (MultiMedia Extensions) + 57 novas instruções vírgula flutuante s/ pilha, modelo SIMD
- **Pentium II**: sem mais novas instruções…

- **1999: Pentium III**: + 70 novas instruções: SSE (Streaming SIMD Extensions); **mais registadores, de 128bits; múltiplas operações sobre números de 32 bits em vírgula flutuante (*precisão simples*) em paralelo; + melhorias no acesso a memória**
- **2001: Pentium IV**:+ 144 novas variantes de instruções SSE2; **duas operações de aritmética de precisão dupla (64 bits) em paralelo**
- **2003: AMD64**:
  - expande a arquitectura IA-32 para 64 bits de endereços e 64 bits de dados (novo modo de operação ***long mode***)
  - ***Legacy mode***: compatível com IA-32
  - **Compatibility mode**:
    - programas utilizadores IA-32
    - SO: long mode AMD64

- **2004: <u>Intel EMT64</u>** (*Extended Memory 64 Technology*); + novas extensões SSE3, +13 novas instruções (**aritméticas, gráficos sobre matrizes, codificação de imagem, conversão de números floating point,etc…**)
- Mais ainda … a ver mais adiante …

→ **O resultado desta evolução…**

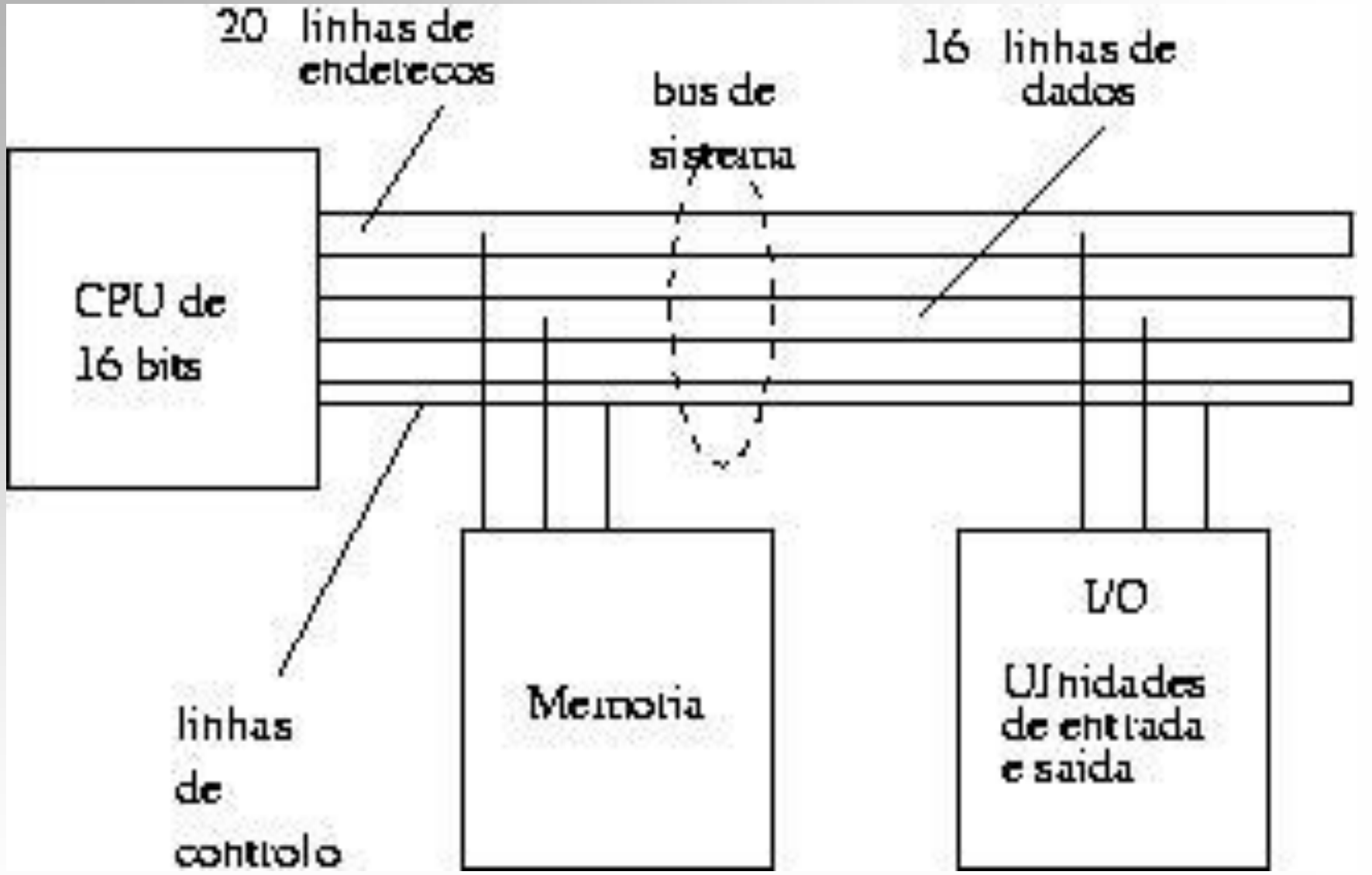
- → **<u>Uma arquitectura muito complexa e difícil de compreender/programar</u>**
  - → **´´refúgio´´ do compilador e do programador num subconjunto pequeno de funcionalidades… resultado do 80386 (IA-32)**
- → **Mas: a mais utilizada em todo o mundo….**

a) Arquitectura 8086 (16 bits)
b) Arquitectura 80386 / IA-32 (32 bits)

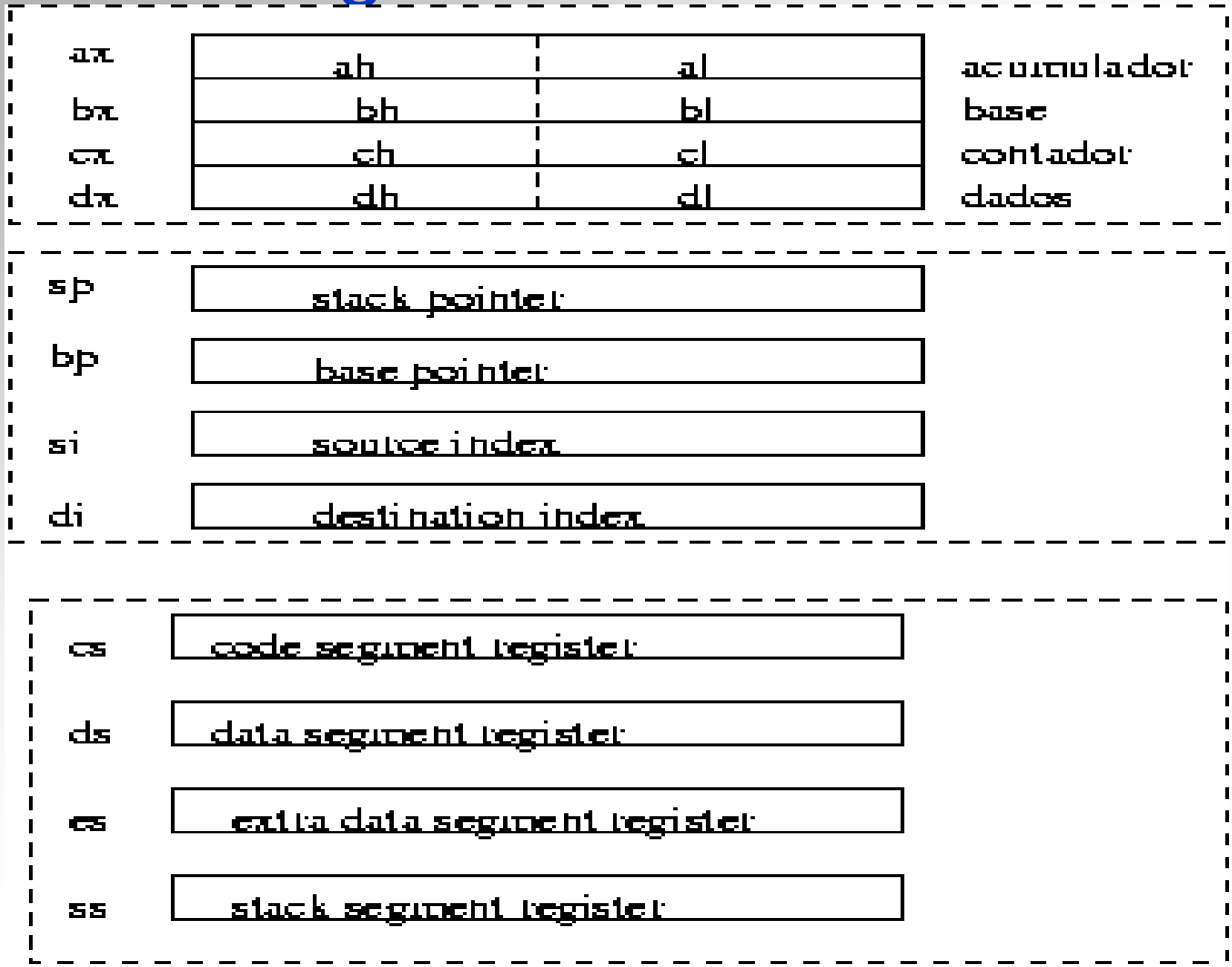
Arquitectura do 8086

- Um número de instruções da ordem das centenas (CISC…)
- Espaço de endereços reais: 1 MegaByte
- CPU com ALU que opera s/ números de 8 e de 16 bits

  (→ compatibilidade com modelos anteriores)
- Mecanismo de segmentação: suporte de programas recolocáveis, independentes da posição de memória onde são carregados
- Registadores de dados e de endereços, não completamente genéricos…


20 linhas de endereços
bus de sistema
16 linhas de dados
CPU de 16 bits
linhas de controlo
Memória
I/O
Unidades de entrada e saída

Registadores do CPU



Registadores ´´genéricos´´ AX, BX, CX, DX

- Utilização na maioria das instruções
- Alguns têm utilização dedicada em operações específicas, por exemplo:
  - BX utilizado como ´´Base´´ para formar um endereço efectivo em modos de endereçamento baseado
  - CX utilizado como contador (em iterações de ciclos)
  - CL utilizado em rotações e desvios de bits
  - AX e DX em instruções de multiplicação/ divisão e I/O
- Podem ser operados como 16 ou 8 bits:

  metades *High (byte mais significativo)* e *Low* (*byte menos significativo*)

  AX = <AH><AL>

  ou AH

  ou AL

Exemplos

Ao usar um registador de 8 bits como operando, a instrução opera sobre 8 bits:

mov al, 9Ch --- endereçamento imediato

Se usar um registador de 16 bits como operando, actua sobre 2 bytes:

mov ax, [20h] --- endereçamento directo

se Mem[20h] = 9Ch -- endereço inferior

Mem[21h] = 10h -- endereço superior

Então AX recebe <10><9C> -- ´´little endian´´

ou

mov [20h], ax

Incrementar bytes de 200h a 202h

Usar:

mov al, [bx] --- bx, como registador indirecto

mov [bx], al

mov ax,[bx] -- 2 bytes,indicado por (bx) e (bx)+1

```
    mov bx, 200h
    mov cx, 3
L:  mov al, [bx]
    inc al
    mov [bx], al
    inc bx
    dec cx
    jnz L
```

Registadores de Estado do CPU

FR - *Flags Register*: 16 bits, que indicam o estado corrente do CPU, após a execução de cada instrução:

- Flags (1 bit cada) associadas ao resultado da última operação efectuada pela ALU: (**importantes para as comparações e saltos condicionais**)
  - **CF – *Carry Flag***:
    - **Posta a 1 pelo CPU, se houve transporte do bit + significativo**
    - **Posta a 0, caso contrário**
  - **OF – *Overflow Flag***:
    - **posta a 1 se houve transbordo na última operação efectuada; posta a 0, se não houve.**
  - **SF – *Sign Flag***:
    - **posta a 1 se o resultado foi negativo;a 0 se foi positivo**
  - **ZF – *Zero Flag***:
    - **posta a 1 se o resultado foi nulo; a 0 se não foi nulo**

Há outras flags, de controlo das operações do CPU.

Registadores de endereço

- **SP, BP, SI, DI**
  - Utilizados como registadores de base ou de índice nos modos de endereçamento:
    - SP – *Stack Pointer* (topo da pilha)
    - BP – *Base Pointer* (base da ´´*activation frame*´´ *corrente*)
    - SI / DI – *Source* / *Destination Index* (registadores de índice, com função de auto-decremento/incremento, nas instruções de transferência e sobre ´´strings´´)
- Não indicam o endereço real: têm de ser somados aos valores contidos nos registadores de base dos segmentos:
  - SP e BP são somados ao valor em SS (Stack Segment Register, aponta a base do seg. Pilha)
  - SI e DI são somados aos valores em DS e ES (Data Segment e Extra Segment Registers, bases dos segmentos de dados)

Modos de endereçamento do 8086

[bx+si]
[bx+di]
[bp+si]
[bp+di]
[si]
[di]
[end]
[bx]
imediato

[bx+si+n]
[bx+di+n]
[bp+si+n]
[bp+di+n]
[si+n]
[di+n]
[bp+n]
[bx+n]

ax (ou ah ou l)
bx
cx
dx
sp
bp
si
di

Modos de endereçamento do 8086

Imediato

Por registador **reg. de 16 ou de 8 bits**

Indirecto por registador **[$r_i$reg] → Mem[$r_i$reg] -- (bx,si,di)**

Baseado **[$b_i$reg+desloc] -- (bx,bp,si,di)**

Baseado ou indexado **com desloc**

Baseado e indexado **[breg+reg_i]**

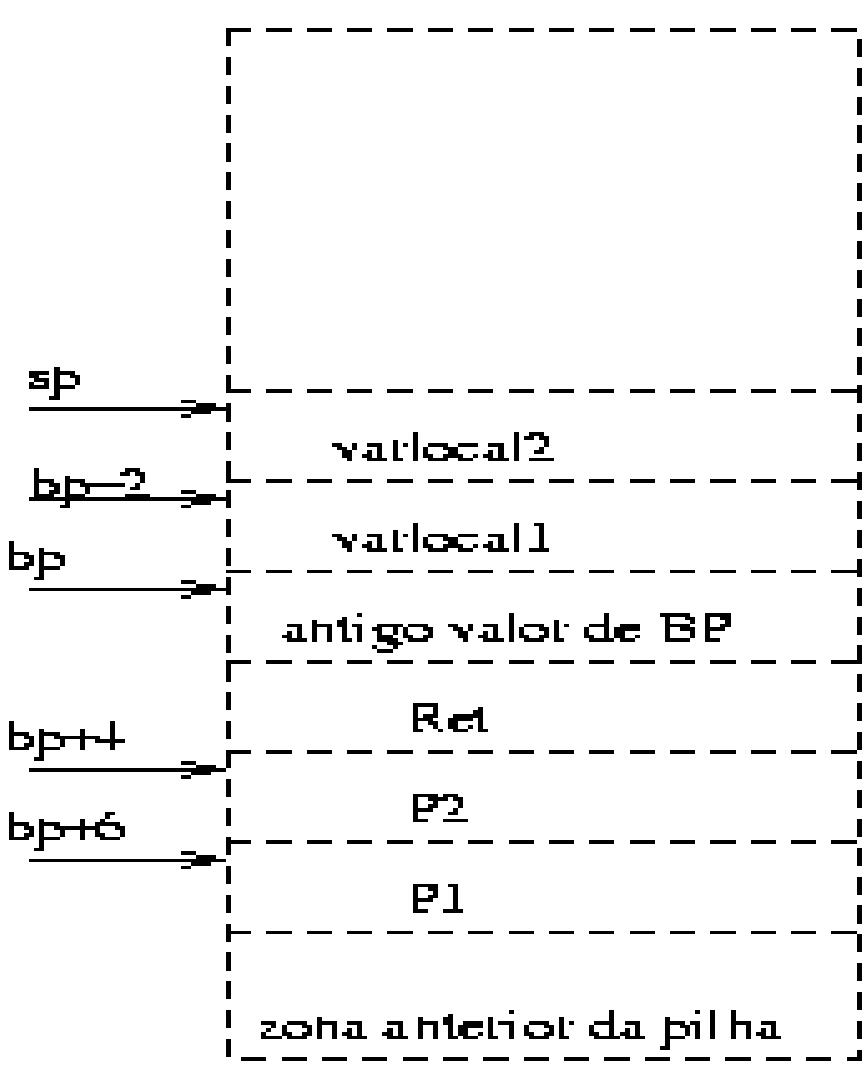
(bx,bp) (di, si)

Relativo ao PC em **jmp, call, branch condic.**

com deslocamento

Aplicação dos modos de ender.

- Acesso simples: end. directo
- *Arrays*: ender. indirecto via SI (índice no array)
- Variáveis localizadas em zona de memória apontada por outra variável:
  - se o apontador em BX → ender. indirecto via BX
  - se é um *array* de base apontada por BX e o índice no *array* está em SI → [bx+si]
- Marcas no stack:
  - Usa BP como marca, apontando o início da *activation frame* e [bp+desloc] para aceder a variáveis locais e a parâmetros
  - Usa [bp+si+desloc] para aceder elementos de *array*



Subrot:

```
push BP
mov bp sp
sub sp, 4
.....
mov bx, [bp+4]
mov si, [bp+6]
dec si
mov al , [bx+si]
.....
mov [bp-2], dx
....
mov sp,bp
pop bp
ret
```

B

C

Programa:

```
push P1
push P2
```

A

```
call Subrot
add  sp,4
```

D

.....

Por registador:

mov ax, dx

mov al, bh

inc ax

Imediato:

mov ax, 1AC0h

add al, AAh

Se os operandos estão em memória, os modos de endereçamento indicam um deslocamento relativo à base de um segmento

- Directo:

  mov al, [00AAh] -- copia Mem[DS:00AA]

  es: mov al, [00AAh] -- copia Mem[ES:00AA]

- Registador indirecto:

  mov [bx], al

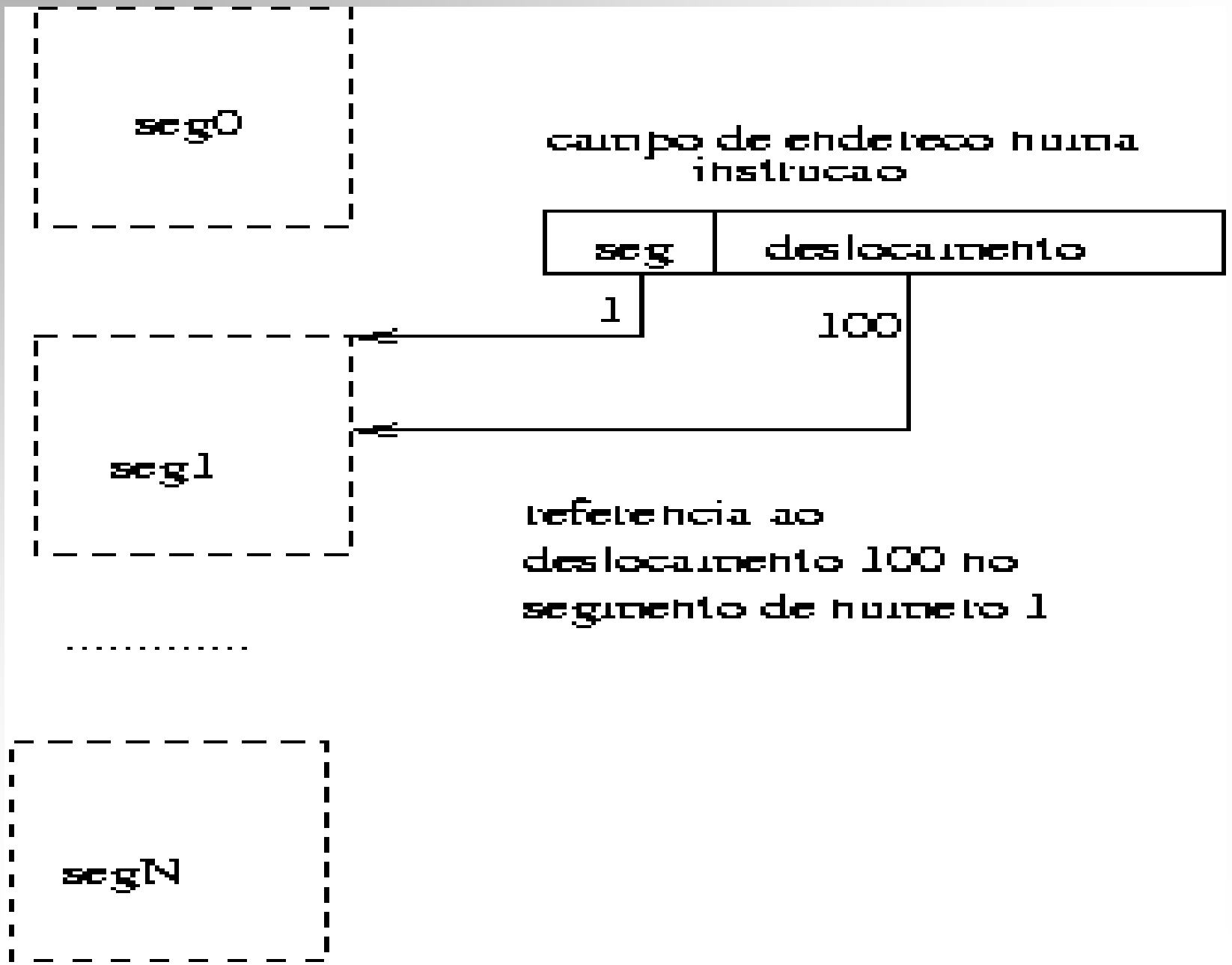
- Baseado ou indexado:

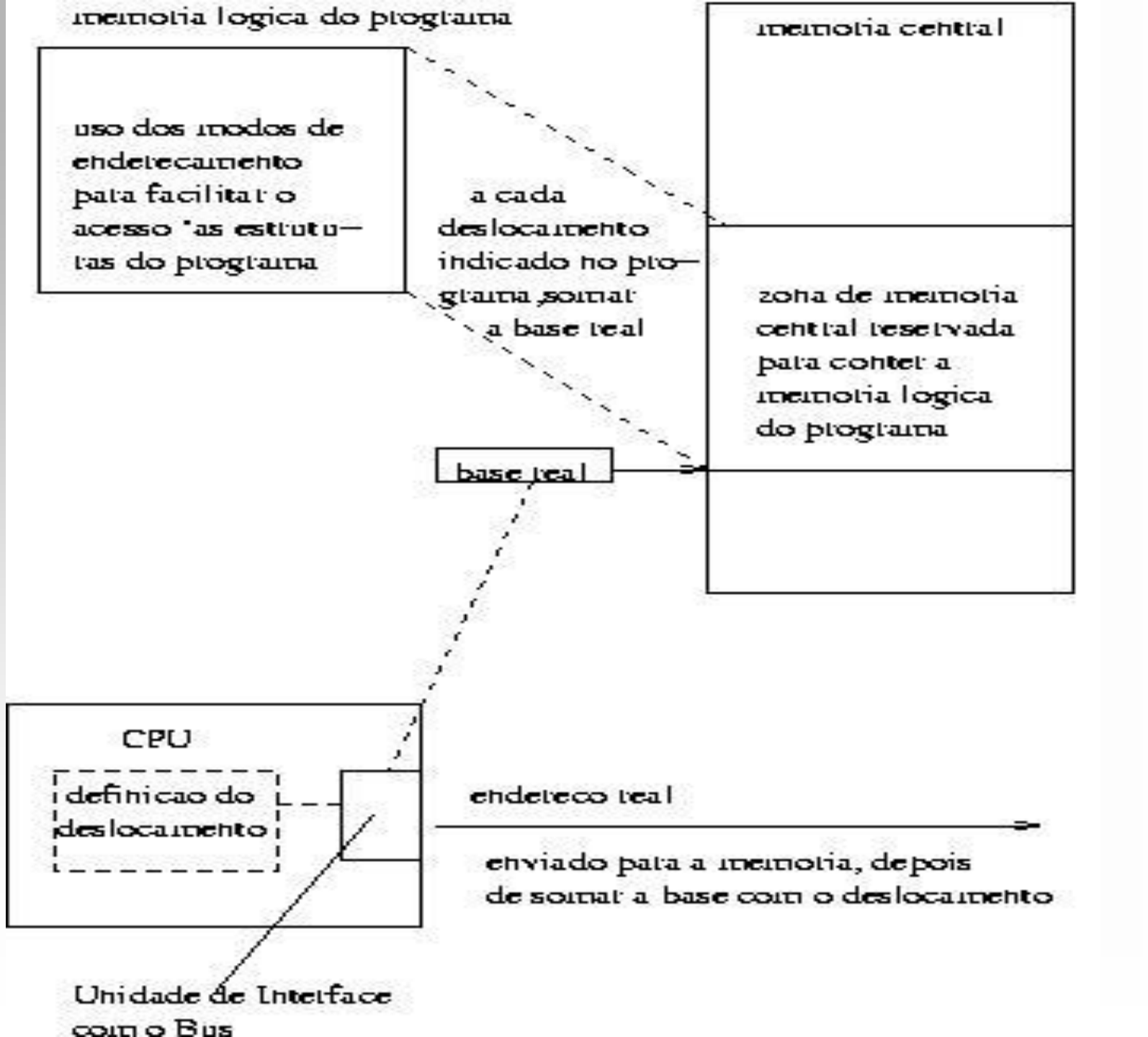
  mov [bx+4], al

IP - *Instruction Pointer*

- Indica a posição da instrução corrente no segmento de código do programa (equivale ao conceito de *Program Counter*)
- É afectado pelas instruções de salto e de chamada de subrotina
- IP não contém um endereço real de memória, mas sim um deslocamento (de 16 bits) relativo à base do segmento de código
- Para se obter o endereço real da instrução: tem de se somar IP ao valor contido no registador de base do segmento de código (CS)

Segmentação de um programa




memoria logica do programa
memoria central
uso dos modos de endereçamento para facilitar o acesso as estruturas do programa
a cada deslocamento indicado no programa somar a base real
zona de memoria central reservada para conter a memoria logica do programa
base real
CPU
definicao do deslocamento
endereco real
enviado para a memoria, depois de somar a base com o deslocamento
Unidade de Interface com o Bus

Segmentação, em geral

- Estruturar os programas em regiões ´´lógicas´´
- Facilita a recolocação dos programas
- Facilita a protecção de regiões de memória
- As bases dos segmentos podem estar contidas:
  - (1) Numa Tabela guardada em Memória Central (Tabela de Segmentos)
  - (2) Numa Tabela em registadores dedicados do CPU
  - (3) Em (1), com uma cópia de parte em (2)

Em geral são inicializadas pelo Sistema de Operação.

- **Por limitações de concepção do 8086… os arquitectos decidiram admitir só 4 Segmentos lógicos, correntes, a cada momento da execução do programa…**
  - **Não havia mais espaço na *chip* do CPU (em 1978…)**

Registadores base dos segmentos

Contém os valores dos endereços de base das regiões de memória onde são carregados os segmentos que compõem um programa em linguagem máquina do 8086:

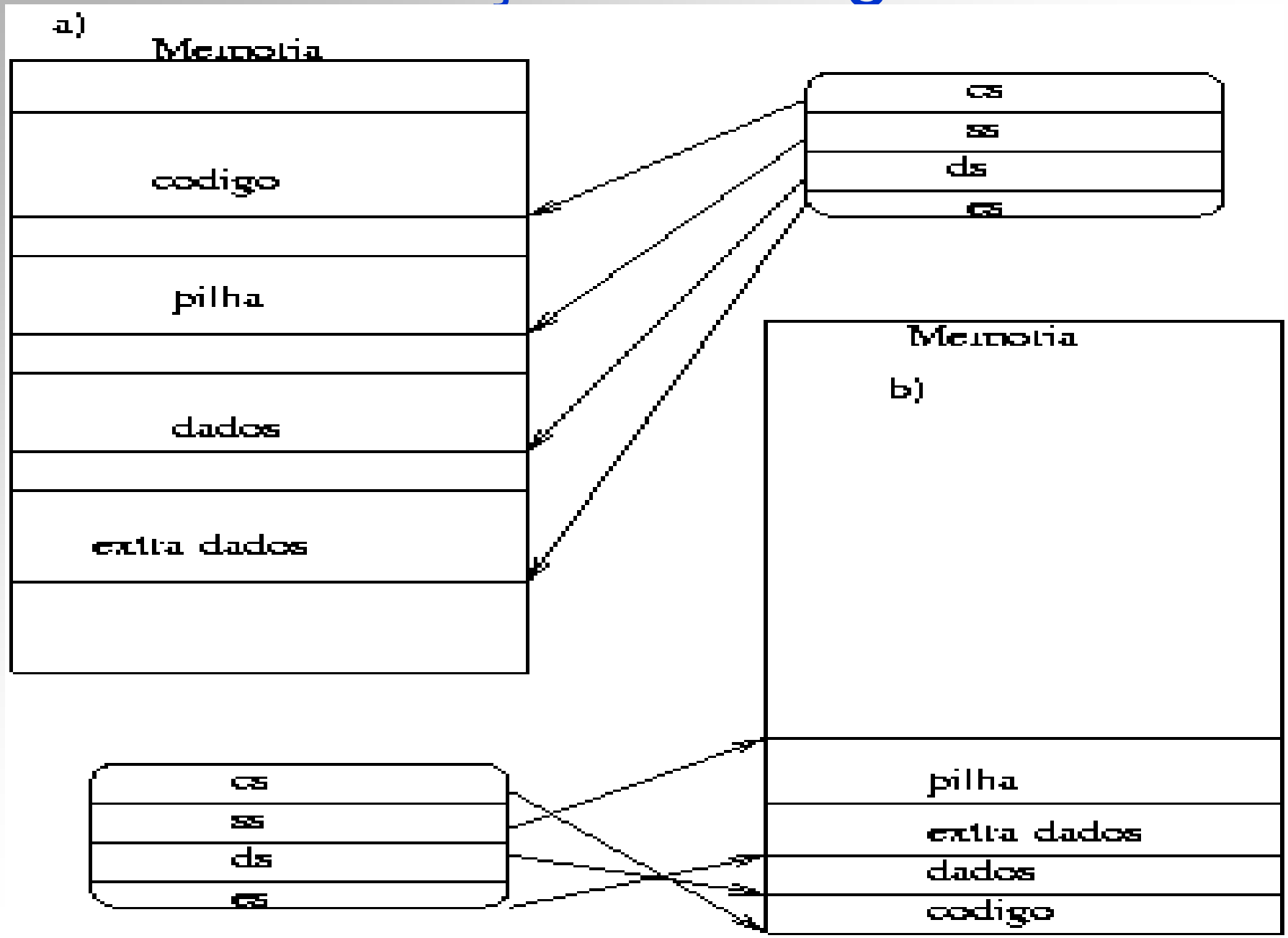
CS - Segmento de código (Code Segmenter Register)

SS - Segmento de pilha (Stack Segment Register)

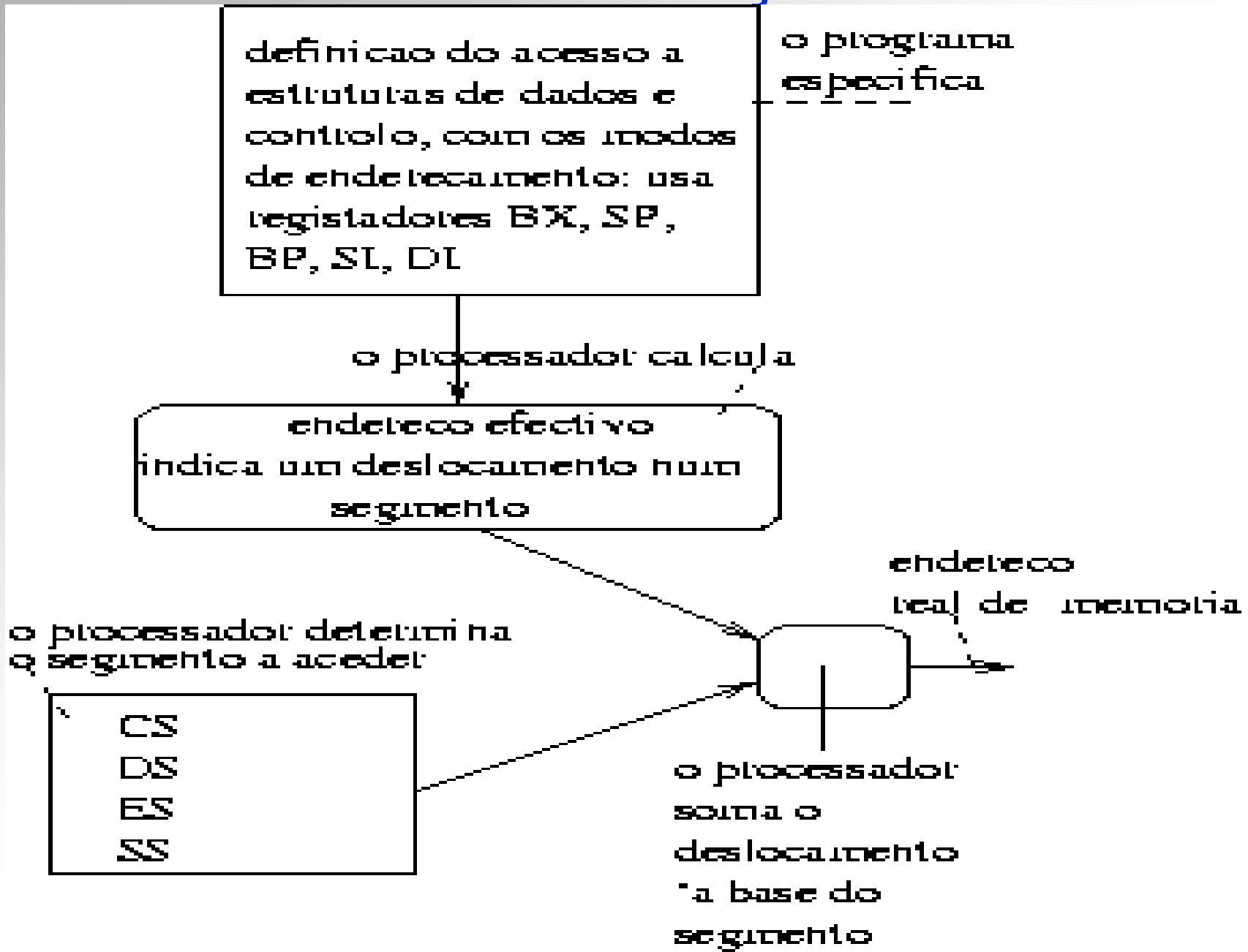
DS - Segmento de dados (Data Segment Register)

ES - Segmento extra de dados (Extra Segment Register)

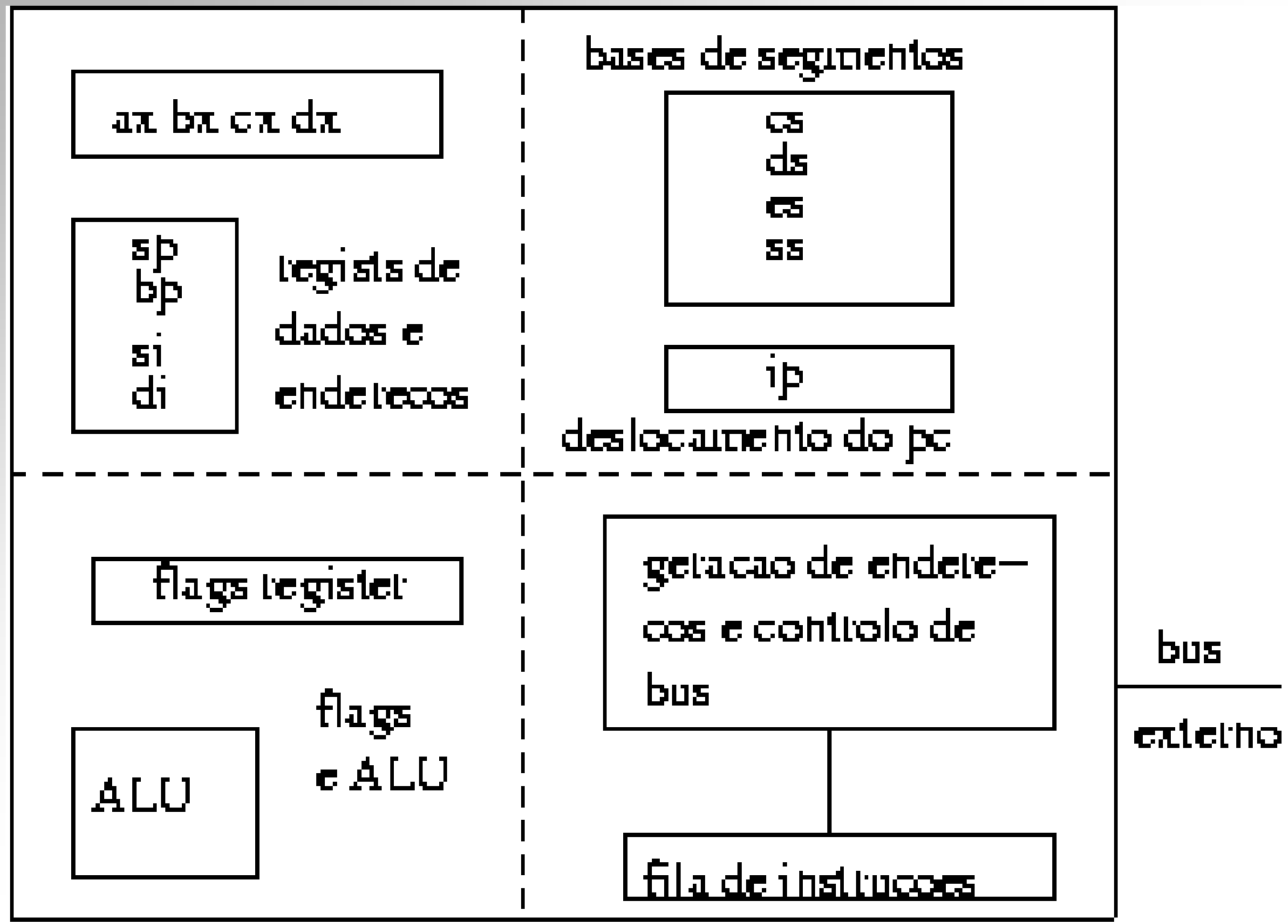
Recolocação de segmentos



Cálculo de endereços no 8086



Registadores do CPU 8086



- Um programa no 8086 só tem acesso, a cada momento, a 4 segmentos lógicos
- As instruções do 8086 não precisam de indicar explicitamente qual o segmento a que se referem: indicam apenas um deslocamento (de 16 bits)
- Os segmentos podem ´´sobrepor-se´´ fisicamente em memória: não há protecção por hardware…
- Podem ter-se diversos casos:
  1. Todos os segmentos sobrepostos em memória (se instruções+dados+pilha < 64 KiloBytes)
  2. Dois segmentos **físicos** em memória:
     - Um para o código (se instruções < 64 KiloBytes)
     - Outro para os dados+pilha (se dados+pilha < 64 KiloBytes)
  3. Mais do que 4 segmentos físicos em memória: exige que o programa modifique os valores dos registadores de base, quando quer mudar de segmento

Endereços reais no 8086

Compostos por um par:

- Base, obtida a partir de um dos 4 registadores de segmento (CS, DS, ES, SS)
- Deslocamento, obtido a partir do cálculo do endereço efectivo, conforme indicado nas instruções

Exemplo:

- Acessos ao código usam CS+IP
- Acessos à pilha usam SS + SP ou SS+BP
- Acessos aos dados usam DS ou ES, mais o endereço efectivo definido pelos modos de endereçamento indicados pelas instruções

Referência	Reg Segmento	Endereço lógico
Fetch instr.	CS	IP
Op. s/ pilha	SS	SP
Uso de BP	SS	end.efectivo
Operandos	DS (ou ES)	end.efectivo

- **Por limitações de concepção do 8086… os arquitectos decidiram que os Registadores de Base dos Segmentos (CS, DS, ES, SS) seriam registadores de 16 bits (pois a ALU e os outros registadores eram todos de 16 bits…)**
- Compreende-se, por questões de uniformidade, mas…
  - O Bus externo tinha 20 linhas de endereço:
    - Isto era necessário, para alcançar, pelo menos, 1 MegaByte

    (16 bits só dariam 64 KiloBytes, muito pouco, já em 1978)

- Como gerar um endereço real de 20 bits, pela soma de dois números de 16 bits?
  - Base (num dos registadores CS, DS, ES, SS)
  - Deslocamento (conforme indicado na instrução)

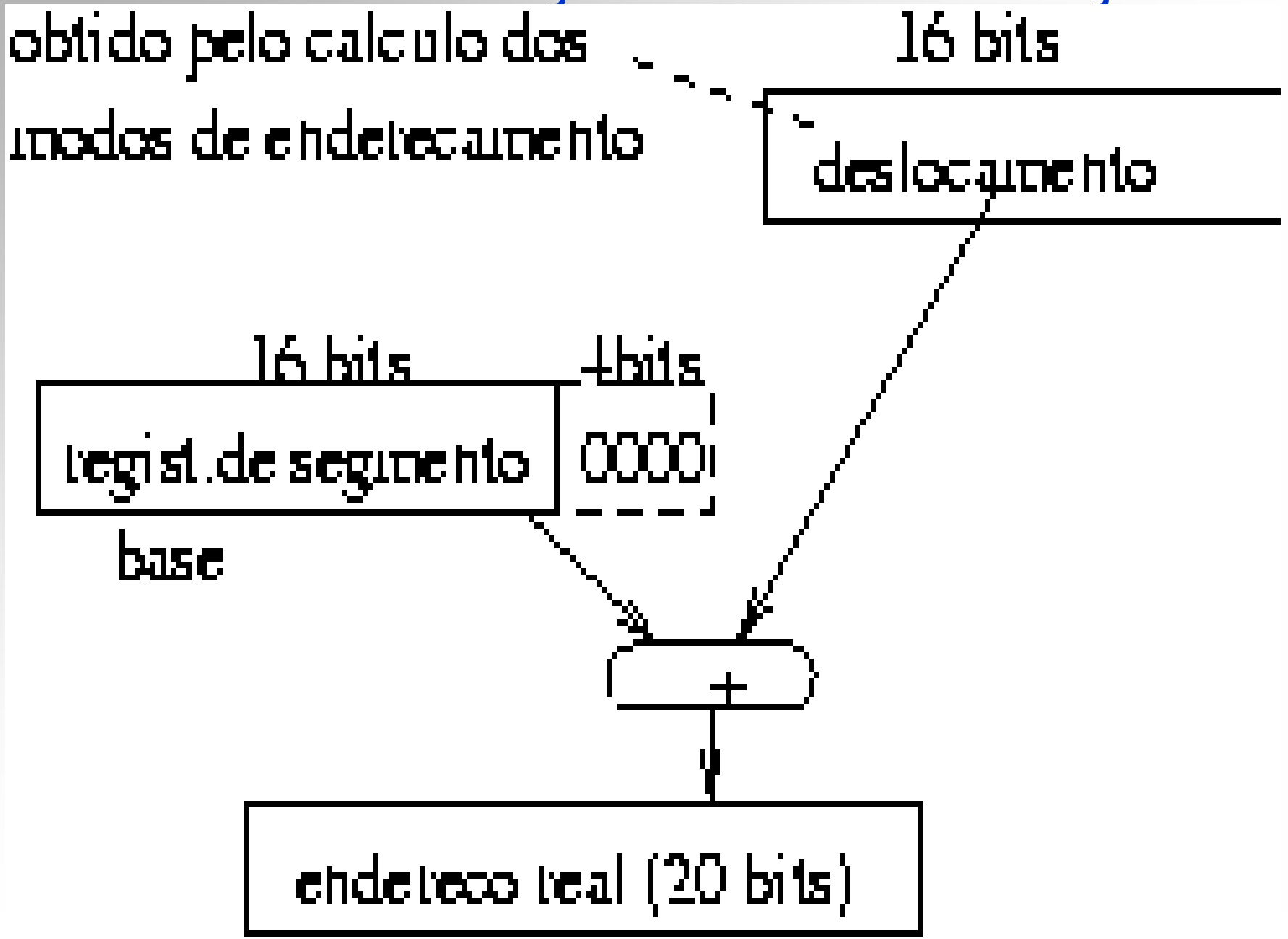
Uma solução ad-hoc

Forçar os endereços de base dos segmentos físicos em memória central, a serem divisíveis por 16:

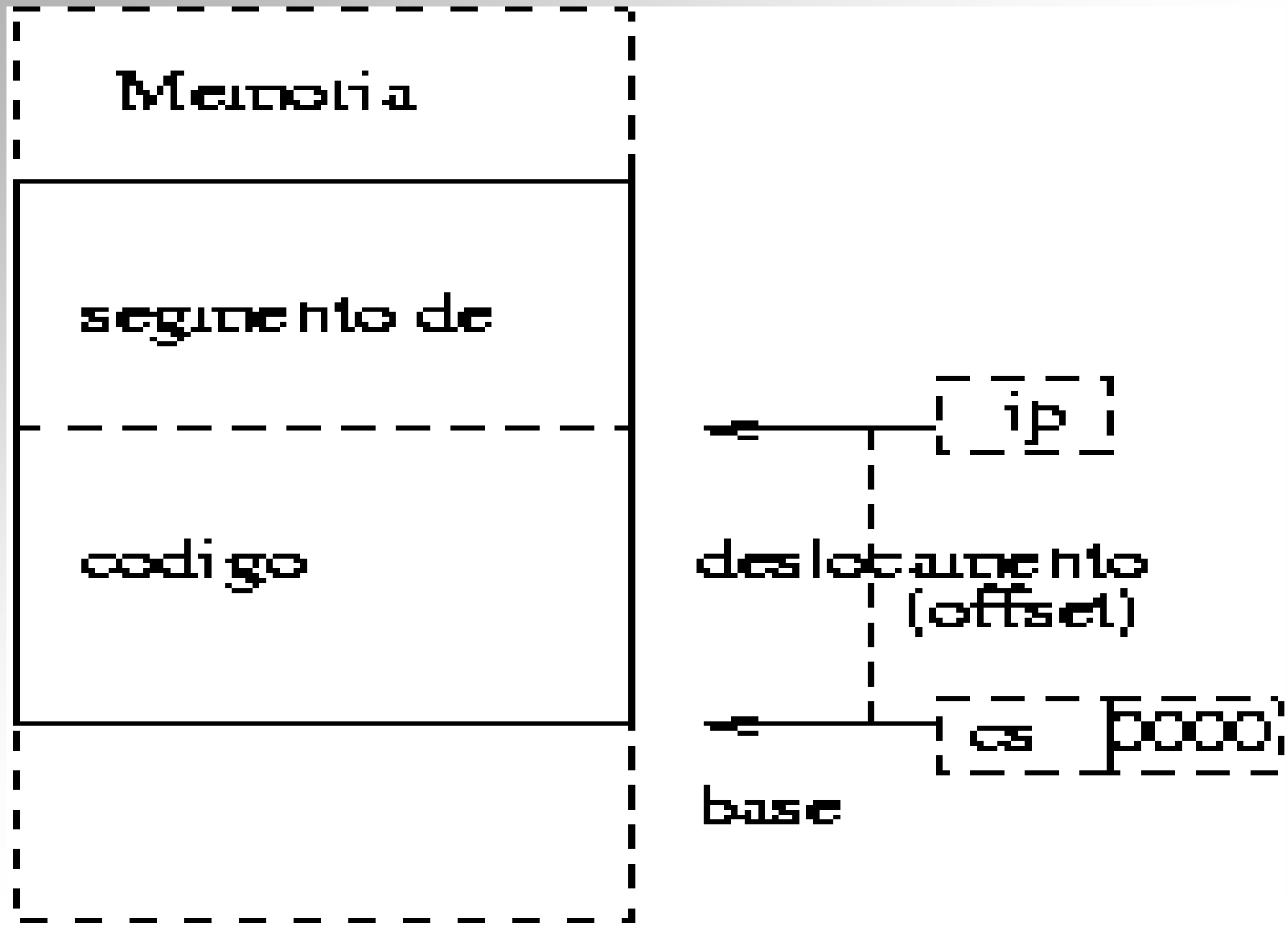
- Os 4 bits menos significativos do endereço de base de um segmento estão sempre a 0

As bases dos segmentos iniciam-se sempre de 16 em 16 bytes em memória.

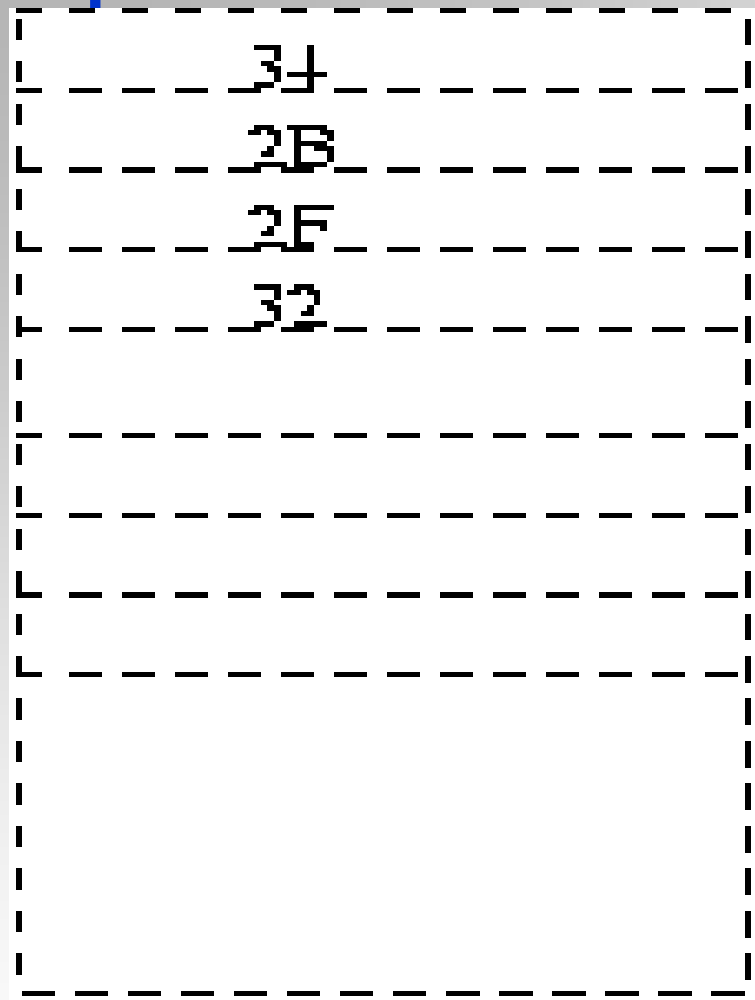
Transformação de endereços



Exemplo de acesso ao seg.código



Apontador: base + deslocamento



base: 32 2F

deslocamento: 2B 34

apontador guardado no endereço n de memória

```
   322F 0
+    2B34
-----------
   -34E24
```

endereço real de 20 bits representado pelo apontador

Conclusões sobre o 8086

- Se o programa não precisa de mais do que 4 segmentos, não há problema:
  - O SO inicializa os registadores de base dos segmentos, após carregar o programa em memória
- Se o programa exige mais do que 4 segmentos (ainda que os Assemblers permitam ao programa designá-los de forma simbólica) torna-se complexo e pouco eficiente ter de mudar explicitamente o valor do registador de segmento, durante a execução…
- Além disso, no 8086, não há protecção de memória…

Solução adoptada no 80286

- Aumentar o número possível de segmentos lógicos até um máximo de **2^{14} = 16 * 1024 segmentos**!
- Manter CS, DS, ES, SS, por compabilidade com as instruções do 8086: **passam a indicar o número do segmento cuja base está guardada numa tabela de segmentos**
- Introduzir, no CPU, mecanismos hardware:
  - **para protecção de memória**
  - **para emular as instruções do 8086 (*modo real 8086*)**
  - **suportar memória virtual (para programas com mais segmentos lógicos do que os que cabem simultaneamente na memória central)**
- Manter os deslocamentos de 16 bits dentro do segmento (**por compatibilidade com o 8086**)
  - → **máximo tamanho de cada segmento = 64 KiloBytes**
- → **O modelo de programação mantêm-se complexo e pouco eficiente** (**um programa com > 4 segmentos é mais lento do que se só tiver 4 segmentos**)

Solução adoptada a partir do 80386

- **Passar o CPU a uma arquitectura de 32 bits:**
  - **ALU de 32 bits (mas continua a manter compatibilidade com modelos anteriores: emulação do 80286 e do 8086 e operações 16 e 8 bits)**
  - **Bus: 32 linhas endereços e 32 linhas de dados**
  - **Manter**:
    - **2^{14} = 16 * 1024 segmentos lógicos**
    - os registadores de base de segmentos CS, DS, ES e SS (**mantêm 16bits**)
  - mas …
    - Passar a deslocamentos de 32 bits dentro do segmento

    → **máximo tamanho de cada segmento:**

    2^{32} = 4 GigaBytes !!!!!

    O programa deixa de precisar de mudar de segmento…

Do 8086 até à arquitectura IA-32

- Os registadores genéricos de 16 bits → 32 bits:

  EAX (**indica Extensão a 32 bits de AX**)

  EBX, ECX, EDX, ESP, EBP, ESI, EDI

- Excepção: os Registadores de Segmento mantêm-se em 16 bits:

  CS, SS, DS, ES, mais dois extra **FS, GS**

- **EIP** estende IP → 32 bits
- **EFLAGS** estende Flags Register → 32 bits

Modos de endereçamento IA-32

- Deslocamentos podem ser 8, 16 ou 32 bits

Modo		
Reg. Indirecto		
Baseado com desloc.		
Baseado mais índice c/ factor de escala	**Base+(2^{Escala} * Indice)** **Escala: 0..3**	**Base: qualquer Reg. Gen.** **Indice: excepto ESP**
Baseado mais índice c/ factor de escala com desloc.	**Base+(2^{Escala} * Indice)+desloc.** **Escala: 0..3**	**Base: qualquer Reg. Gen.** **Indice: excepto ESP**

Escala = 0 → sem factor de escala
Escala = 1 → dados de 16 bits
Escala = 3 → dados de 64 bits