

# Arquitetura Intel iX86

- CISC - Complex Instruction Set Computer
- Duas décadas de evolução mantendo compatibilidade
  - 1978: i8086, 16-bits, 29Ktransistores
  - 1985: i386, 32-bits, 275Ktransistores
  - 1989: i486, FPU on-chip, 1,2Mtransistores
  - 1993: Pentium, 3,1Mtransistores
  - 1999: Pentium III, 9,5Mtransistores ( 6000X)

# iX86 - a CISC Architecture (1)

- Muitas instruções, incluindo instruções complexas
  - Codificação eficiente => códigos de instrução de comprimento variável (1 a 5 bytes no 8086)
- Poucos registos “multi-purpose” (8)
- Arquitectura Memória-Memória
  - Instruções aritméticas e lógicas admitem operandos em memória
- Múltiplos “Modos de Endereçamento” de dados

# iX86 - a CISC Architecture (2)

- “Flags” - bits que refletem o resultado de operações aritméticas e lógicas:
  - Zero, Carry, Sign, Overflow, ...
- Espaço de endereçamento não linear - memória segmentada
  - Endereço efectivo = endereço-base do segmento + deslocamento (offset)
  - Segmentos: Code (CS), Data (DS), Stack (SS), ...
  - CS, DS, SS, ES - registos de segmentação: contêm endereço-base do segmento

# iX86 vs. MIPS

- 32 registos gerais
- Arquitectura *Load/store*

*add r3, r2, r1*

- Memória linear

gama de endereços:  $0 \dots 2^{32}-1$

- Inexistência de “*flags*”  
(códigos de condição)

- 8 registos especializados
- Arquitectura Mem.-Mem.

*add [BX], AX*

Mem[DS: BX] := Mem[DS:BX] + (AX)

- Memória segmentada:

End.Efectivo =

End.-Base Segm. + deslocamento

- Instruções específicas para operações sobre o stack

*push, pop*

- “*flags*”

# iX86 - Espaço de endereçamento

- Endereços de 20-bits - **1 Mbyte** endereçável
- Arquitectura de 16-bits - registos de 16-bits

## **Como utilizá-los para endereçar 1 MB?**

Memória segmentada: segmentos de **64KB (16 bits)**

Registos dedicados contêm o endereço base de cada segmento: CS (code seg.), DS (data seg.), SS (stack seg.)

$$\begin{aligned}\mathbf{Endereço\ efectivo} &= \mathbf{(Registo\ Seg.) * 10_{16} + (R)} \\ &= \mathbf{(Registo\ Seg.) \ll 4 + (R)}\end{aligned}$$

(tanto para dados como para instruções)

# Registos “multi-purpose”

- **AX** - Acumulador - 1\*16-bits ou 2\*8-bits  
Mult., Div., I/O
- **BX** - *base index* - endereçar memória
- **CX** - Contador - *LOOP/L OOPD*
- **DX** - g.p.r. - parte do resultado em Mult. E do dividendo em Div.
- **BP** - *base pointer*
- **DI** - g.p.r.; aponta para o destino de strings
- **SI** - g.p.r.; aponta para a origem de strings

# Registos dedicados

- Registos de segmentação: **CS, DS, SS, ES**

- **IP** - *Instruction Pointer*

Endereço Instrução =  $CS * 10H + IP$

- **SP** - *Stack Pointer*

Topo do stack =  $SS * 10H + SP$

- Flags: **Zero, Sign, Overflow, Carry, Aux. Carry**

**Parity, Direction**

**Interrupt, Trap**

# iX86 - Reportório de instruções

## Instruções de transferência de dados

***MOV Dest, Source***

***Dest***

***Source***

Registo

Registo

Registo

Memória - **load**

Memória Registo - **store**

- Que modos de endereçar dados em memória são suportados pela arquitectura?

# iX86 - Modos de endereçamento de dados

- **Directo a Registo:**

*MOV AX, BX # AX := BX*

- **Imediato:**

*MOV BX, 1000H # BX := 1000H*

- H - hexadecimal    1000H = 0001 0000 0000 0000

- **Directo à Memória (Absoluto):**

*MOV AX, [1000H] # (BX) := Mem[1000H]*

*MOV AX, DATA6*

# iX86 - Modos de endereçamento de dados

- **Indirecto a Registo:**

*MOV AX,[BX] # AX := Mem[DS:BX]*

- Registos que podem ser usados para endereçar memória: ***BX, DI, SI, BP***

***BX, DI, SI*** - Data Segment usado

Mem[***DS:BX***]    Mem[***DS:DI***]    Mem[***DS:SI***]

***BP*** - Stack Segment usado

Mem[***SS:BP***]

# iX86 - Modos de endereçamento de dados

- **Modo Deslocamento** (ou *Register-Relative*):

*MOV AX,[BX+4]*

- **Indexado** (*Base plus Index*):

*MOV AX,[BX+DI]*

- **Base Relative plus Index** :

*MOV AX,[BX+4]*

# LEA

- **LEA** - **L**oad **E**ffective **A**ddress

**LEA** AX, NUM # (AX) := &NUM

- **LDS** - carrega de duas posições contíguas de memória **DS** com o endereço-base de um segmento e o outro registo especificado na instrução com um *offset* no segmento

**LDS** BX, [DI]

# iX86 - Instruções de transferência de dados de/para o **stack**

- ***PUSH*** - coloca no stack o conteúdo de:
  - Registo
  - Imediato
  - Conteúdo de posição de memória (data segment) de 16-bits

***PUSHA*** - salvaguarda no stack todos os registos

***PUSHF*** - salvaguarda as *flags* no stack

- ***POP*** - inverso de ***PUSH***

***POPA, POPF***

# Directivas Assembly

- Data: ***DB***- Define Byte  
***DW*** - Define Word
- ***EQU*** - Equate                   TEN EQU 10
- ***PROC, ENDP*** - início e fim de subrotina  
***PROC NEAR*** - S.R. no segmento do programa  
***PROC FAR*** - S.R. noutro segmento