Lic. Engenharia de Sistemas e Informática

1º ano

2004/05

Luís Paulo Santos

Módulo

Arquitectura de Computadores

Arquitectura de um Computador

Arquitectura e Organização de Computadores

Organização: Componentes que realizam a arquitectura:
- organização do CPU (*pipeline*, ...)
- unidades específicas (FPU, MM, ...)
- barramentos (largura, velocidade)
- frequência do relógio

Arquitectura: atributos visíveis ao programador:
- conjunto de instruções
- tamanho da palavra (*bits*)
- registos

Tópicos a analisar

- tamanho da palavra
- registos visíveis aos programador
- modos de acesso aos operandos
- tipos de instruções presentes num CPU
- formatos de instruções em linguagem máquina
- ordenação de *bytes*

Este é um parâmetro fundamental do sistema que determina, em *bits*:

- o tamanho, por defeito, dos números inteiros
- o tamanho dos endereços
- o tamanho dos registos de uso genérico
- o tamanho, por defeito, do bloco de dados que é lido da memória

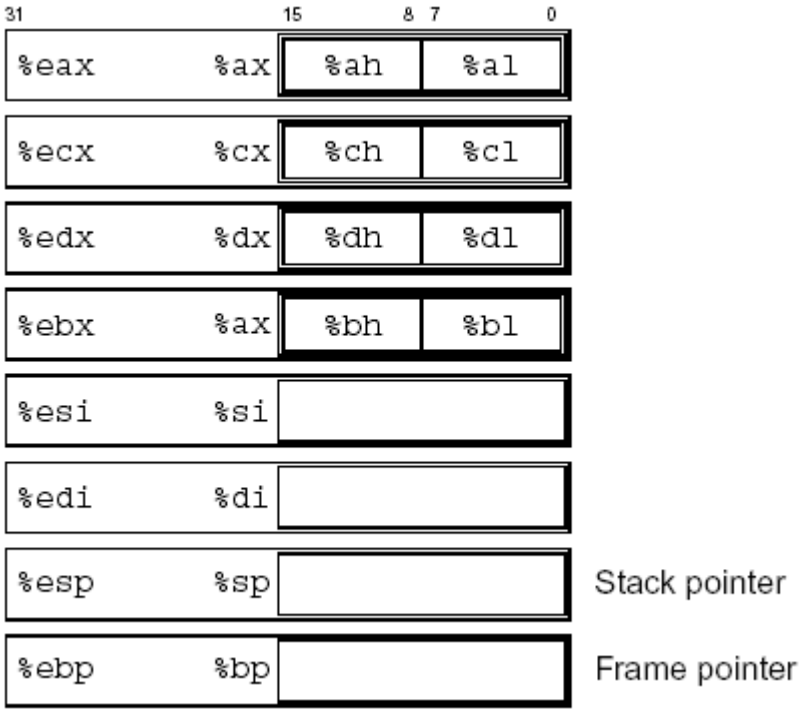
O IA32 tem uma palavra de 32 *bits*, mas, por razões históricas, são as quantidades de 16 *bits* que são referenciadas como palavras (w), sendo as quantidades de 32 *bits* referenciadas como palavras longas (l).

32 *bits* permitem endereçar 2^{32} *bytes* = 4 Gbytes

Registos visíveis aos programador

– em arquitecturas RISC (32 registos genéricos...)

– no IA32



- Imediato:
  - O operando é uma constante especificada na própria instrução:

```
addl 10, %eax          # 10 - operando imediato
```

- Em Registo:
  - O operando é um registo:

```
addl 10, %eax          # %eax - registo
```

- Em Memória:
  - O operando encontra-se em memória.
    Na instrução especifica-se o endereço da posição de memória
  - O endereço pode ser uma constante, estar num registo ou resultar de uma operação:

```
addl 10, [0x7FFF0000]     # addr = 0x7FFF0000
addl 10, [%eax]           # addr está no reg. %eax
addl 10, [%eax+%ebx+10]   # addr = %eax+%ebx+10
```

Tipos de instruções presentes num CPU

- operações aritméticas e lógicas
  - soma, subtração, multipl, div, ...
  - AND, OR, NOT, XOR, comparação, ...
  - deslocamento de bits, ...
- transferência de informação
  - de/para registos/memória, ...
- controlo do fluxo de execução
  - para apoio a estruturas de controlo
  - para apoio à invocação de procedimentos

Ex: instruções aritméticas/lógicas no IA32

inc D	D← D +1	Increment
dec D	D← D –1	Decrement
neg D	D← -D	Negate
not D	D← ˜D	Complement
add S, D	D← D + S	Add
sub S, D	D← D – S	Subtract
imul S, D	D← D * S	32 bit Multiply
and S, D	D← D & S	And
or S, D	D← D | S	Or
xor S, D	D← D ˆ S	Exclusive-Or
shl k, D	D← D << k	Left Shift
shr k, D	D← D >> k	Logical Right Shift

Ex: instruções de transferência de informação no IA32

mov	S, D	D←S	Move (byte,word,long_word)
push	S		Escreve valor no topo da *stack*
pop	D		Lê valor do topo da *stack*

D – destino [Reg | Mem] **S** – fonte [Imm | Reg | Mem]
D e **S** não podem ser ambos operandos em memória

Ex: instruções de controlo de fluxo no IA32

jmp Label	Salto incondicional
jz Label	Salto se última operação deu zero
js Label	Salto se última operação deu negativo
jg Label	Salto se última comparação deu >
call Label	Salta para procedimento / função
ret	Regressa de procedimento / função

As instruções em linguagem máquina são sequências de *bits.*

Ex.: 11100011010010010001000111101010101

A unidade de controlo descodifica estas sequências pois conhece o formato das instruções.

O formato das instruções varia com a família de CPUs:

Ex.: O conjunto de instruções e respectivo formato é completamente diferente para a arquitectura Intel32 e PowerPC.

O formato de instruções define o conjunto de campos que constituem a instrução. Exemplo:

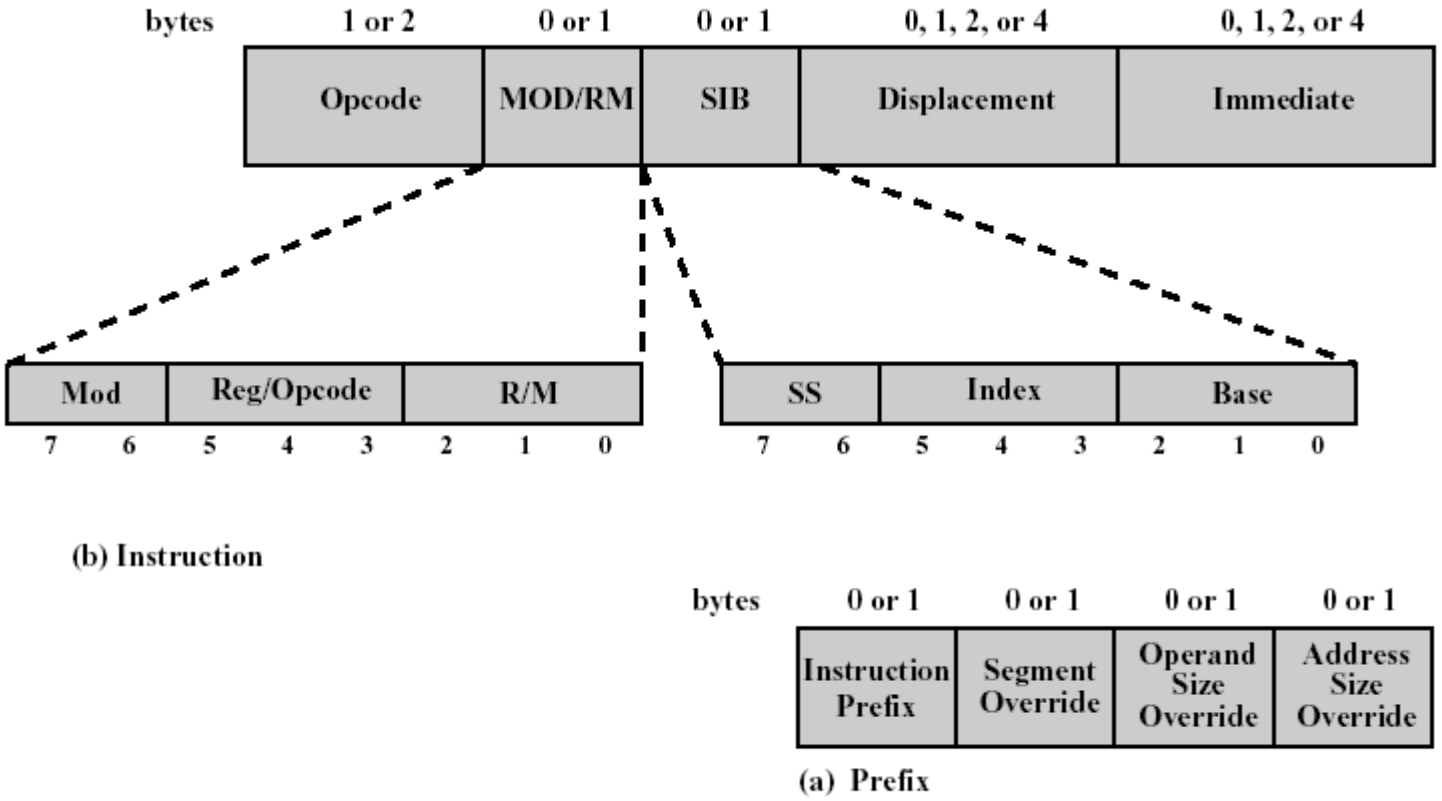
opcode	modo endereçamento	operando	operando

1. `opcode` – indica qual a operação a realizar

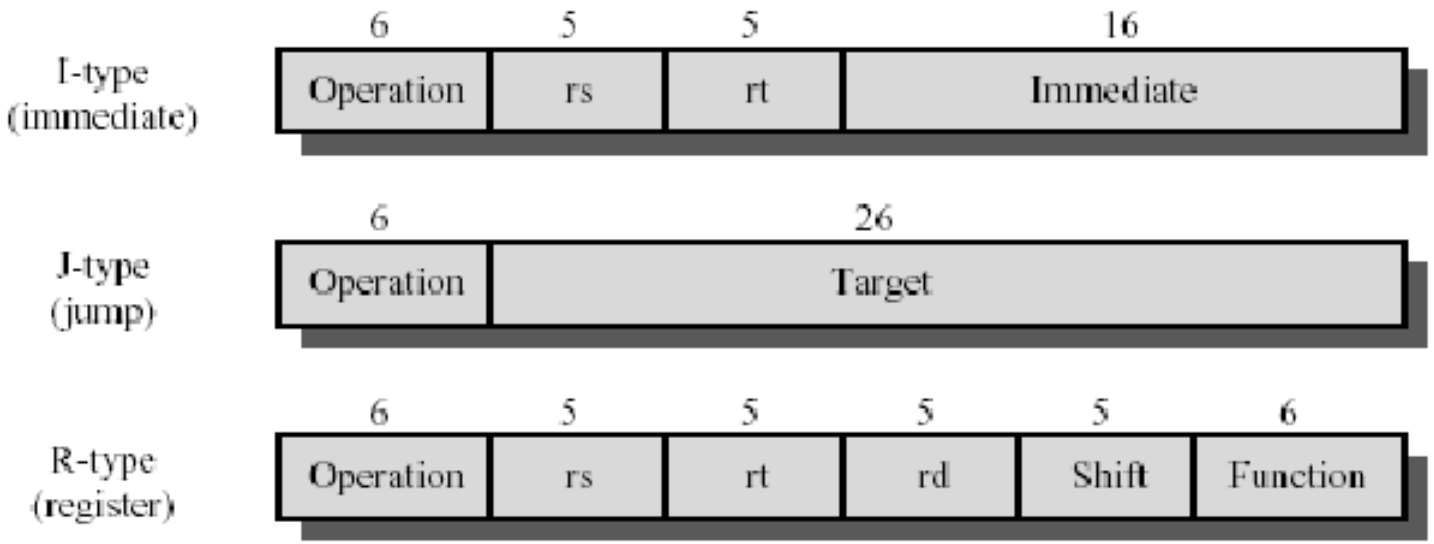
2. `modo de endereçamento` – indica se o(s) operando(s) é imediato, em registo, em memória, etc...

3. `operando` – especifica o operando

Formatos de instruções no Pentium



Formatos de instruções no MIPS (RISC)



- Quando uma palavra é constituída por vários *bytes* é necessário definir a ordem como são guardados em memória:
  - *Little endian:* ***começar pela ponta mais pequena***,ou seja pelo *byte* menos significativo
  - *Big endian:* ***começar pela ponta maior***,ou seja pelo *byte* mais significativo

Exemplo (32 bits) : 0 x 54 AB 00 7C

Little endian

0x0000	0x7C
0x0001	0x00
0x0002	0xAB
0x0003	0x54

Big endian

0x0000	0x54
0x0001	0xAB
0x0002	0x00
0x0003	0x7C