Introdução à arquitetura IA-32

João Canas Ferreira

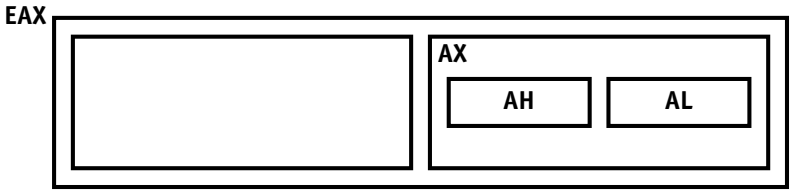
Fevereiro 2013

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Registos do CPU (modo de 32 bits)

32 bits	16 bits	8 bits	
EAX	AX	AL, AH	acumulador
EBX	BX	BL, BH	
ECX	CX	CL, CH	contador
EDX	DX	DL, DH	
EDI	DI	—	
ESI	SI	—	
EBP	BP	—	utilização especial
ESP	SP	—	utilização especial
EFLAGS	—	—	"bandeiras" (em modo 32 bits)
EIP	—	—	*instruction pointer* (em modo 32 bits)

Os registos indicados em cada linha NÃO são independentes.



Formato de instruções

Número de operandos

- Zero operandos: `nop`
- Um operando: `jmp destino`
- Dois operandos: `add eax,ebx`

Categorias de operandos

- Registos (eax, bx, dl, ...)
- Constantes (valores *imediatos*): 21, 17c3h, 0a3h
- Referências a posições de memória (diversos modos de endereçamento)

Regras para instruções de 2 operandos

- Formato geral: `instr op1, op2`
  Significado: op1 ← op1 <operação> op2
- Operandos têm tamanho igual.
- Apenas um operando pode ser uma referência a memória.

Operações de transferência de dados

Instrução `mov`: **Copiar** valor de um local para outro.

O local de origem NÃO é alterado.

Exemplos

- `mov eax,ebx` registos ficam com valor igual (o valor de ebx)
- `mov eax,eax` sem efeito prático
- `mov bl,dl`
- `mov dx, bx`
- `mov dx, 0ABCDh` copia constante para registo dx

Instruções não existentes

- `mov eax,bx` operandos de tamanhos diferentes
- `mov 0ABCDh,dx` não se pode alterar uma constante

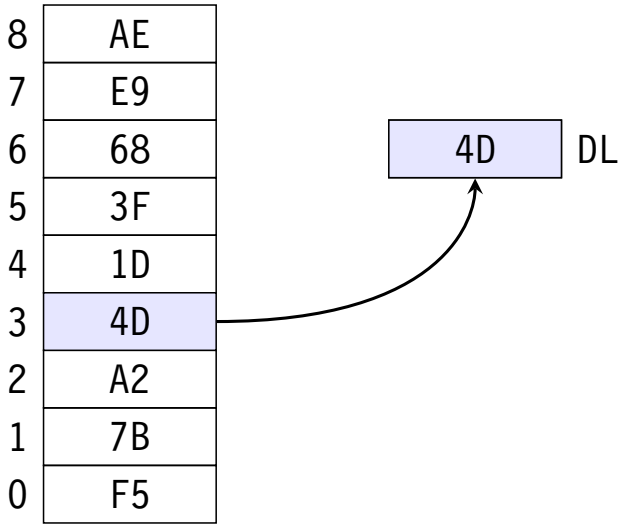
Operações aritméticas e lógicas (subconjunto)

IA-32	MIPS	Operação
`add eax,ebx`	`add $s0,$s0,$s1`	adição
`add eax,20`	`addi $s0,$s0,20`	adição
`sub eax,ebx`	`sub $s0,$s0,$s1`	subtração
`inc eax`	`addi $s0,$s0,1`	incrementar
`dec eax`	`addi $s0,$s0,-1`	decrementar
`neg eax`	`sub $s0,$zero,$s0`	simétrico
`and eax,ebx`	`and $s0,$s0,$s1`	E (bit a bit))
`or eax,ebx`	`or $s0,$s0,$s1`	OU (bit a bit)
`xor eax,ebx`	`xor $s0,$s0,$s1`	OU-exclusivo (bit a bit)
`not eax`	`xori $s0,$s0,0xFFFF`	complemento

Modos de endereçamento de memória

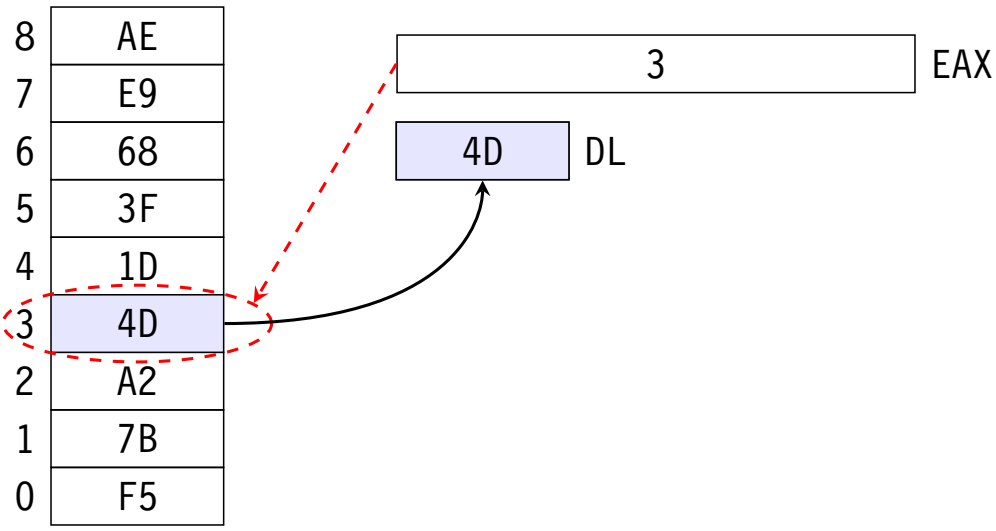
- Endereçamento direto de memória
  - `mov eax, [1234ABCDh]`
    transfere valor da posição de memória 1234ABCDh para o registo eax
  - `add BYTE PTR [1234ABCDh], 21`
    soma 21 ao valor guardado na posição de memória 1234ABCDh
- Endereçamento indireto (via um registo)
  - `mov eax, [edi]`
    transfere valor da posição cujo endereço está em `edi` para o registo eax
  - `add BYTE PTR [edx], 21`
    soma 21 ao valor guardado na posição cujo endereço está em edx
- Endereçamento indexado (via registo + constante)
  - `mov eax, [30+edi]`
    transfere valor da posição de endereço edi+30 para o registo eax
  - `add BYTE PTR 100[edx], 21` *sintaxe alternativa*
    soma 21 ao valor guardado na posição de endereço edx+100

Exemplo de endereçamento direto



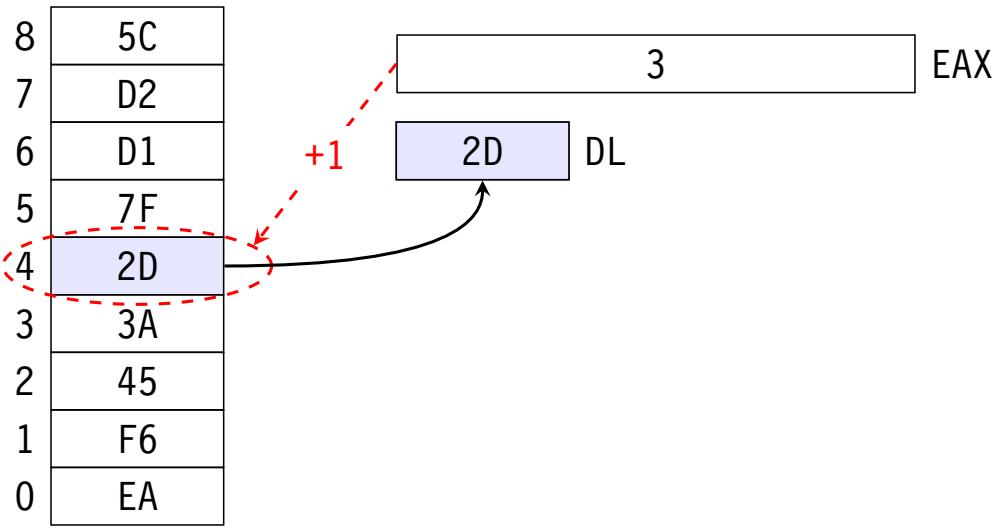

```
mov dl, [3]
```


Exemplo de endereçamento indireto




```
mov dl, [eax]
```

Exemplo de endereçamento indexado

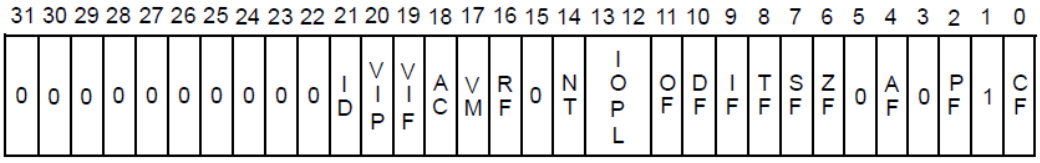



```
mov dl, [eax+1]
```


Registo de "flags"

Flag: campo de 1 bit

- O registo EFLAGS é actualizado após a execução de cada instrução.
- Cada instrução afeta um subconjunto específico de bits.



CF *Carry flag:* =1 se houve transporte (aritmética sem sinal)

ZF *Zero flag:* =1 se resultado da operação foi zero

SF *Sign flag:* =1 se resultado da operação é negativo

OF *Overflow flag:* =1 se houve *overflow* (aritmética com sinal)

Exemplo: as instruções add e sub afetam os quatro indicadores *(flags)* mencionados (e outros).

Saltos condicionais (1)

- Os saltos condicionais são do tipo: `J<cond> destino`
- O salto é tomado se a condição for verdadeira.

Utilização de instrução de salto condicional

```
add     eax,ebx     ; afeta flags
jo      erro        ; salta se existir overflow
...                 ; processamento "regular"
erro: ...           ; tratar de overflow
```

- Comparações explícitas são efetuadas com a instrução

  `cmp op1,op2`

- A instrução `cmp` afeta *flags* exactamente como

  `sub op1,op2`

  mas *sem alterar op1*.

Saltos condicionais (sem sinal)

Instruction Mnemonic	Condition (Flag States)	Description
Unsigned Conditional Jumps		
JA/JNBE	(CF or ZF) = 0	Above/not below or equal
JAE/JNB	CF = 0	Above or equal/not below
JB/JNAE	CF = 1	Below/not above or equal
JBE/JNA	(CF or ZF) = 1	Below or equal/not above
JC	CF = 1	Carry
JE/JZ	ZF = 1	Equal/zero
JNC	CF = 0	Not carry
JNE/JNZ	ZF = 0	Not equal/not zero
JNP/JPO	PF = 0	Not parity/parity odd
JP/JPE	PF = 1	Parity/parity even
JCXZ	CX = 0	Register CX is zero
JECXZ	ECX = 0	Register ECX is zero

Fonte: Intel 64 and IA-32 Architectures Software Developer's Manual Vol. 1

Saltos condicionais (com sinal)

Signed Conditional Jumps		
JG/JNLE	((SF xor OF) or ZF) = 0	Greater/not less or equal
JGE/JNL	(SF xor OF) = 0	Greater or equal/not less
JL/JNGE	(SF xor OF) = 1	Less/not greater or equal
JLE/JNG	((SF xor OF) or ZF) = 1	Less or equal/not greater
JNO	OF = 0	Not overflow
JNS	SF = 0	Not sign (non-negative)
JO	OF = 1	Overflow
JS	SF = 1	Sign (negative)

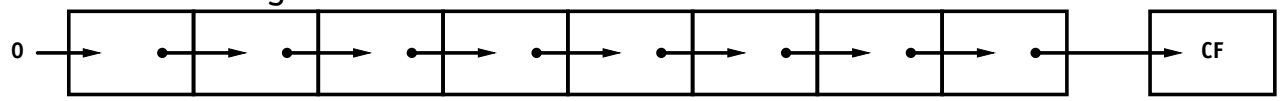
Fonte: Intel 64 and IA-32 Architectures Software Developer's Manual Vol. 1

O programador deve escolher o tipo de teste (com ou sem sinal) com base no tipo de dados que está a processar.

Deslocamento lógico vs. aritmético

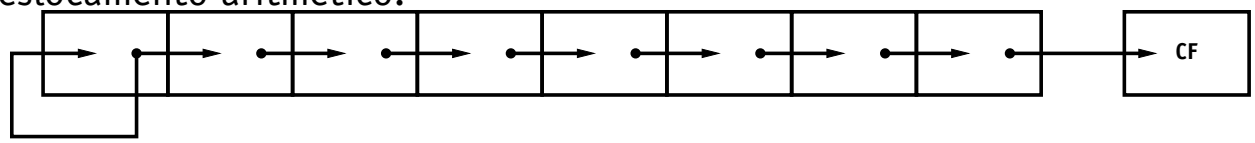
Para a direita

Deslocamento lógico:



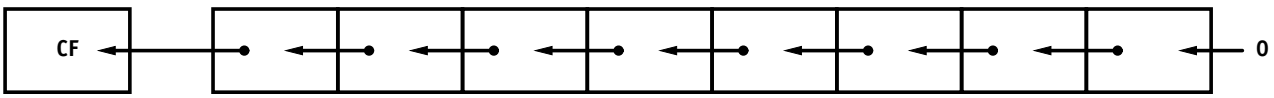
Exemplo: `shr eax, 1` [constante: 0..31]

Deslocamento aritmético:



Exemplo: `sar eax, 1`

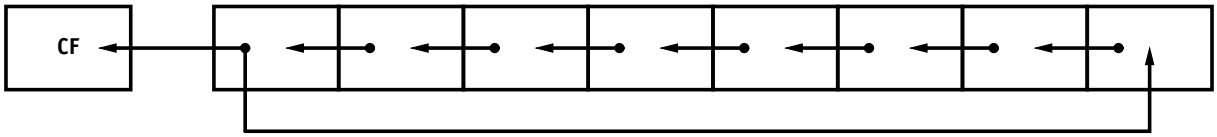
Para a esquerda



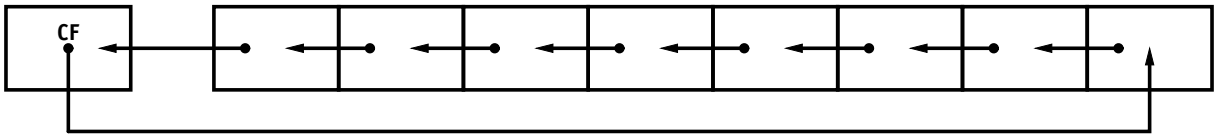
Exemplo: `shl eax, 1` é o mesmo que `sal eax, 1`

Rotações

Para a esquerda



Exemplo: `rol eax, 1` [constante: 0..31]



Exemplo: `rcl eax, 1` [constante: 0..31]

Para a direita

- `ror eax, 1` bit menos significativo copiado para CF
- `rcr eax, 1` rotação inclui CF