Mais instruções aritméticas

João Canas Ferreira

Março 2015

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Multiplicação sem sinal

- Multiplicação de 2 números de `N` bits dá um resultado em `2N` bits.
- A instrução MUL multiplica um operando por AL, AX ou EAX
- Formato:
  - `MUL reg/mem8`
  - `MUL reg/mem16`
  - `MUL reg/mem32`

Multiplicando	Multiplicador	Produto
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

A notação REG2:REG1 indica que o resultado tem os bits menos significativos em REG1 e os bits mais significativos em REG2.

MUL: exemplos de utilização

100H * 2000H, operandos de 16 bits

```
            .data
val1        WORD    2000H
val2        WORD    100H
            .code
            mov     ax, val1
            mul     val2        ; DX:AX=00200000H, OF=CF=1
```

12345H * 1000H, operandos de 32 bits

```
mov         eax, 12345H
mov         ebx, 1000H
mul         ebx     ; EDX:EAX=0000000012345000H, OF=CF=0
```

- CF indica se metade superior tem dígitos diferentes de 0

Multiplicação por adições e deslocamentos

- SHL pode ser usado para multiplicar por potências de 2
- Qualquer número pode ser decomposto numa soma de potências de 2
- Exemplo: para multiplicar EAX por $36 = 32 + 4$ basta
  - p1 = EAX * 4 (deslocamento de 2 bits)
  - p2 = EAX * 32 (deslocamento de 5 bits)
  - Calcular p1 + p2

```
mov   eax,123
mov   ebx,eax
shl   eax,5                 ; × 2^5
shl   ebx,2                 ; × 2^2
add   eax,ebx
```

Questão: como multiplicar por 10?

Multiplicação com sinal

- A instrução IMUL multiplica um operando por AL, AX ou EAX
- Os operandos estão em complemento para dois

48*4, operandos de 8 bits

```
mov     al, 48
mov     bl, 4
imul    bl          ; AX=00C0H, CF=OF=1
```

4823424*(-423), operandos de 32 bits

```
mov         eax, 4823424
mov         ebx, -423
imul        ebx     ; EDX:EAX=FFFFFFFF86635D80H, CF=OF=0
```

- CF=OF=0: resultado cabe na metade inferior do destino (EAX em 32 bits)
- Existem outros formatos da instrução IMUL

Divisão de números sem sinal

- A instrução DIV divide AX, DX:AX ou EDX:EAX por um valor inteiro sem sinal e de tamanho apropriado (resultado é truncado)
- Divisão por zero gera uma exceção (interrompe a execução)
- Formato:
  - DIV reg/mem8
  - DIV reg/mem16
  - DIV reg/mem32

Dividendo	Divisor	Quociente	Resto
AX	reg/mem8	AL	AH
DX:AX	reg/mem16	AX	DX
EDX:EAX	reg/mem32	EAX	EDX

Se quociente não couber no registo, ocorre uma exceção.

```
mov   edx, 0
mov   eax, 8003h  ; dividendo
mov   ecx, 100h   ; divisor
div   ecx         ; EAX=00000080H, EDX=3
```

Garantir que EDX (ou DX) contém o valor pretendido

Extensão de sinal

- Para efetuar divisão com sinal, pode ser necessário expandir o dividendo para EDX (ou DX).
- CBW: extensão de sinal de AL para AX
- CWD: extensão de sinal de AX para DX:AX
- CDQ: extensão de sinal de EAX para EDX:EAX

```
        .data
val     SDWORD -101         ; FFFFFF9Bh
        .code
        mov    eax, val
        cdq                  ; EDX:EAX = FFFFFFFFFFFFFF9Bh
```


Divisão com sinal

- Instrução IDIV: divisão com sinal
- Mesmos operandos que a instrução DIV
- Resultado truncado em direção a 0; sinal do resto igual ao do dividendo

Exemplo: -48/5

```
        mov    al, -48
        cbw                  ; extensão de sinal de AL para AX
        mov    bl,5
        idiv   bl            ; AL=-9, AH=-3
```

Exemplo: -48/5 em 32 bits

```
        mov    eax, -48
        cdq                  ; extensão de EAX para EDX
        mov    ebx,5
        idiv   ebx           ; EAX=-9, EDX=-3
```

Exemplos de expressões aritméticas (1/2)

Exemplo: v4 = (v1 + v2)* v3 (sem sinal)

```
        mov     eax, var1
        add     eax, var2    ; EAX = v1 + v2
        mul     var3         ; EAX = EAX * var3
        jc      excesso      ; resultado tem mais de 4 bytes
        mov     var4, eax
```

Exemplo: eax = (-v1 * v2) + v3 (com sinal)

```
        mov     eax, var1
        neg     eax
        imul    var2
        jo      excesso1     ; resultado tem mais de 4 bytes
        add     eax, var3
        jo      excesso2
```


Exemplos de expressões aritméticas (2/2)

Exemplo: v4 = (v1 + v2) / v3 (sem sinal)

```
        mov     eax, var1
        add     eax, var2    ; EAX = v1 + v2
        xor     edx, edx     ; garantir que EDX=0
        div     var3         ; EAX = 0:EAX / var3
        mov     var4, eax
```

Qual o resultado para v1 = 13, v2 = 3, v3 = 3? 5

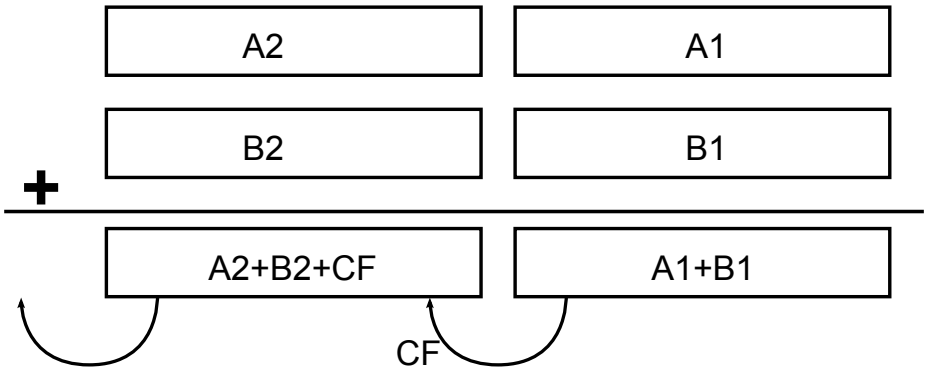
Exemplo: eax = (-v1 / v2) + v3 (com sinal)

```
        mov     eax, var1
        neg     eax
        cdq                  ; extensão de sinal
        idiv    var2
        add     eax, var3
        jo      excesso
```

→ -1

Extensão da adição: método

- Como implementar aritmética com operandos de mais de 4 bytes?
  - Operações são feitas por blocos (do menos significativo para o mais significativo)
  - Valor de CF deve ser passado de bloco para bloco
  - IA-32 suporta extensão de adição e subtração



- Valores guardados em memória com bloco menos significativo primeiro (i.e., no endereço mais baixo).

Extensão da adição: instrução ADC

- ADC: *add with carry*
- Soma dois operandos e valor de CF
- `ADC eax,ebx` → eax := eax + ebx + CF

Somar val64 ao valor de EDX:EAX

```
        .data
val64   DWORD  1Fh, 1       ; 2^32 + 31
        .code
        add    eax, val64[0]; menos significativos
        adc    edx, val64[4]; mais significativos
        jc     excesso      ; assume representação sem sinal
```

Declaração de dados alternativa: `val64 QWORD 0000000010000001Fh`

Extensão da subtração

- SBB: *subtract with borrow*
- Subtrai do destino o outro operando e e valor de CF
- `SBB eax,ebx` → eax := eax - ebx - CF

Subtrair val64 ao valor de EDX:EAX

```
        .data
val64   DWORD  1Fh, 1       ; Valor é 2^32 + 31 = 4294967327
        .code
        sub    eax, val64[0]; menos significativos
        sbb    edx, val64[4]; mais significativos
        jc     fora_de_gama; assume representação sem sinal
```

- Esta abordagem é extensível a números de 12, 16, ... bytes