

Aula 3 e 4

- Multiplicação e Divisão
- Instruções lógicas

Multiplicação de inteiros

- MUL (inteiro não sinalizado)

MUL REG

MUL MEM

– operador unário

AL, AH, EAX utilizado como segundo operando

- $AL \times \text{fonte} = AX$

(multiplicação de bytes, resultado em uma palavra)

- $AX \times \text{fonte} = DX:AX$

(multiplicação de palavras, resultado em palavra dupla)

- $EAX \times \text{fonte} = EDX:EAX$

(multiplicação de palavras duplas, resultado em 64 bits)

- CF e OF são ligados se a metade mais significativa do resultado for diferente de zero; demais flags indefinidos

Multiplicação

- **Inteiros sinalizados**

imul REG

imul MEM

imul IMEDIATO, REG, REG

imul IMEDIATO, MEM, REG

imul REG, REG

imul IMEDIATO, REG

imul MEM, REG

* CF e OF são ligados se a metade mais significativa do resultado não for apenas extensão do sinal; demais flags indefinidos

Divisão de inteiros

- **Inteiros não sinalizados**

- div reg

- div mem

- **Inteiros sinalizados**

- idiv reg

- idiv mem

Divisão de inteiros

- $AX / \text{fonte} = AL$ e resto em AH
(divisão por byte)
- $DX:AX / \text{fonte} = AX$ e resto em DX
(divisão por palavra)
- $EDX:EAX / \text{fonte} = EAX$ e resto em EDX
(divisão por palavra dupla)
- Todos os flags são indefinidos
- Se o resultado tiver mais bits do que pode ser armazenado no quociente é gerada uma interrupção do tipo 0 (erro de divisão)

Divisão de inteiros

- `mov $0x8000, %ax`
- `mov $0x2, %cl`
- `divb cl` ?
- O resultado 0x4000 não pode ser armazenado em AL
- Utilizar um registrador maior:
 - `and $0xFFFF0000, %eax`
 - `and $0xFF00, %cx`
 - `divw cx`
 - Agora o resultado pode ser armazenado no registrador AX

Divisão de inteiros (sinalizados)

- `mov $0x8000, %ax`
- `mov $0x2, %cl`
- **Cuidado !!!! `0x8000` (16 bits) \neq `0x00008000` (32 bits)**
- **Com números inteiros é necessário realizar a extensão de sinal!**

Extensão de sinal

- `cbw (al -> ax)`
- `cwd (ax -> dx:ax)`
- `cwde (ax-> eax) *386`
- `cdq (eax-> edx:eax) *386`
- Instrução generalizada
 - `movsx %al, %ax`
 - `movsx %cl, %cx`
 - `movsx %cx, %ecx`

Usando extensão de sinal

- `mov $0x8000, %ax`
- `cwd`
- `mov $0x2, %cx`
- `idiv %cx`

Multiplicação de inteiros

Custo de uma instrução de multiplicação

	Clocks			
operands	286	386	486	Size Bytes
reg8	13	9-14	13-18	2
reg16	21	9-22	13-26	2
reg32		9-38	13-42	2-4
mem8	16	12-17	13-18	2-4
mem16	24	13-26	12-25	2-4
mem32		12-21	13-42	2-4

Multiplicação de inteiros

- Deslocamentos
 - 0010 -> shift left 1 -> 0100
 - Multiplicação por 2

Custo de uma instrução shift

operands	Clocks			Size Bytes
	286	386	486	
reg,1	2	3	3	2
mem,1	7	7	4	2-4
reg,CL	5+n	3	3	2
mem,CL	8+n	7	4	2-4
reg,immed8	5+n	3	2	3
mem,immed8	8+n	7	4	3-5

Divisão de inteiros

- Deslocamentos
 - 0100 -> shift right ->0010 (divisão por 2)

Deslocamento

- SHL (shift left)=SAL (shift arithmetic left)
- shl count, dest
- sal count, dest
 - shl IMM, REG
 - shl IMM, MEM
 - shl %CL, REG
 - shl %CL, MEM

CF armazena o último bit deslocado

OF é 1 se durante um shift left de 1 os dois últimos bits mais significativos forem diferentes do bit deslocado

SF, PF, ZF

Deslocamento

- SHR (shift right)
 - shr IMM, REG
 - shr IMM, MEM
 - shr %CL, REG
 - shr %CL, MEM



– Flags afetados: CF, ZF, PF, SF

Deslocamento

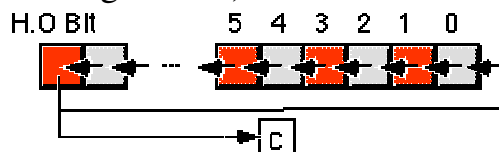
SAR (shift arithmetic right)



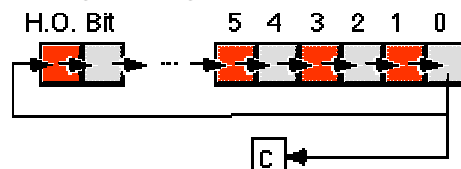
Flags afetados: CF, ZF, PF, SF

Rotate

- ROLL (rotate logical left)



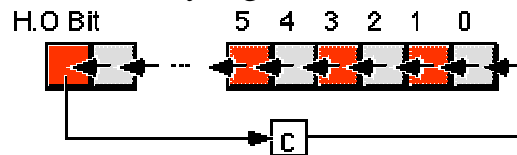
- ROLR (rotate logical right)



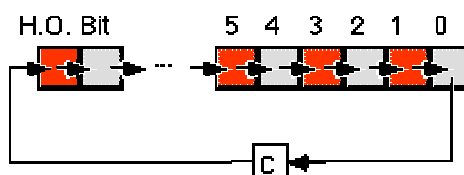
- CF= último bit rotacionado
- OF= sinal diferente do resultado (somente quando cl=1)

Rotate

- RCLL (rotate carry logical left)



- RCLR (rotate carry logical right)



- CF= último bit rotacionado
- OF= resultado com sinal diferente (somente quando $cl=1$)

Instruções de lógicas

- AND
 - OF=0, CF=0, SF, ZF, PF
- NOT
- OR
 - OF=0, CF=0, SF, ZF, PF