

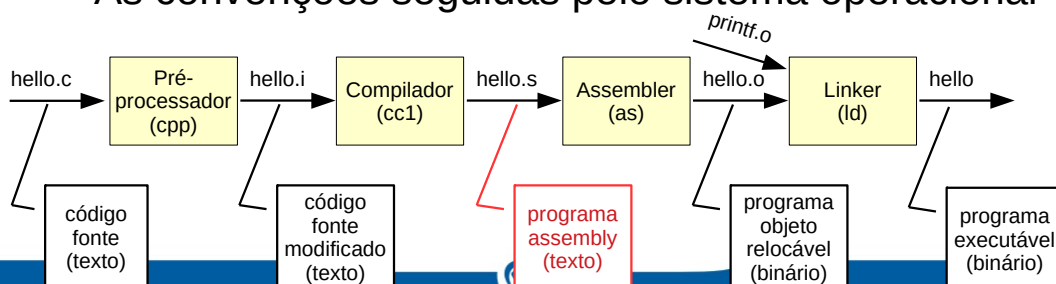
Software Básico

Introdução ao Assembly

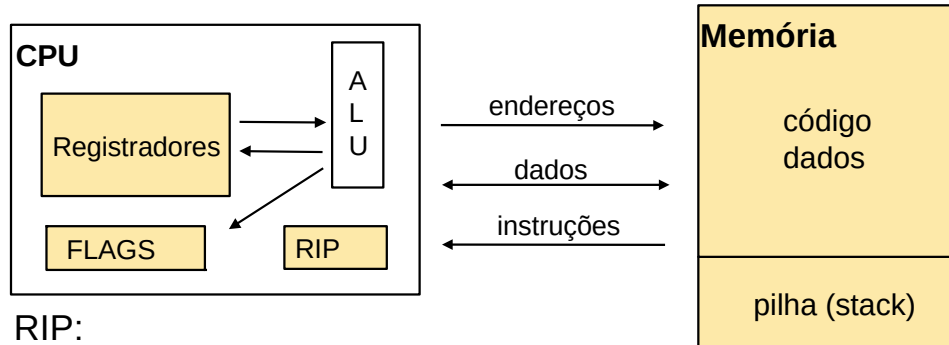


Representação de Programas

- O compilador gera o código de um programa conforme
 - O conjunto de instruções da máquina alvo
 - As regras estabelecidas pela linguagem de programação (por exemplo, linguagem C)
 - As convenções seguidas pelo sistema operacional



Visão do Programador Assembly



RIP:

- endereço da próxima instrução

Registradores:

- valores inteiros, endereços

FLAGS:

- status da última operação
- *overflow?* zero? resultado < 0?



Linguagem de Montagem

- Instruções executam operações simples
 - Transferência de dados
 - Operações aritméticas e lógicas
 - Controle do fluxo de execução
 - Desvios e chamadas de função
- Tipos de dados básicos
 - Valores inteiros (1, 2, 4, 8 bytes)
 - Endereços de memória
 - Valores em ponto flutuante



Programa em C

```
int nums[] = {10, -21, -30, 45};

int main() {
    int i, *p;
    for (i = 0, p = nums; i != 4; i++, p++)
        printf("%d\n", *p);
    return 0;
}
```



Programa em Assembly

Seção
de
dados

```
.data
nums: .int    10, -21, -30, 45
```

Seção
de
código

```
.text
.globl main
main:
    movl    $0, %ebx        /* i */
    movq    $nums, %r12     /* p */
L1:
    cmpl    $4, %ebx        /* if (i == 4) */
    je      L2              /* goto L2 */
    movl    (%r12), %eax     /* eax = *p */
    addl    $1, %ebx        /* i++ */
    addq    $4, %r12        /* p++ */
    jmp     L1
L2:
    ret
```

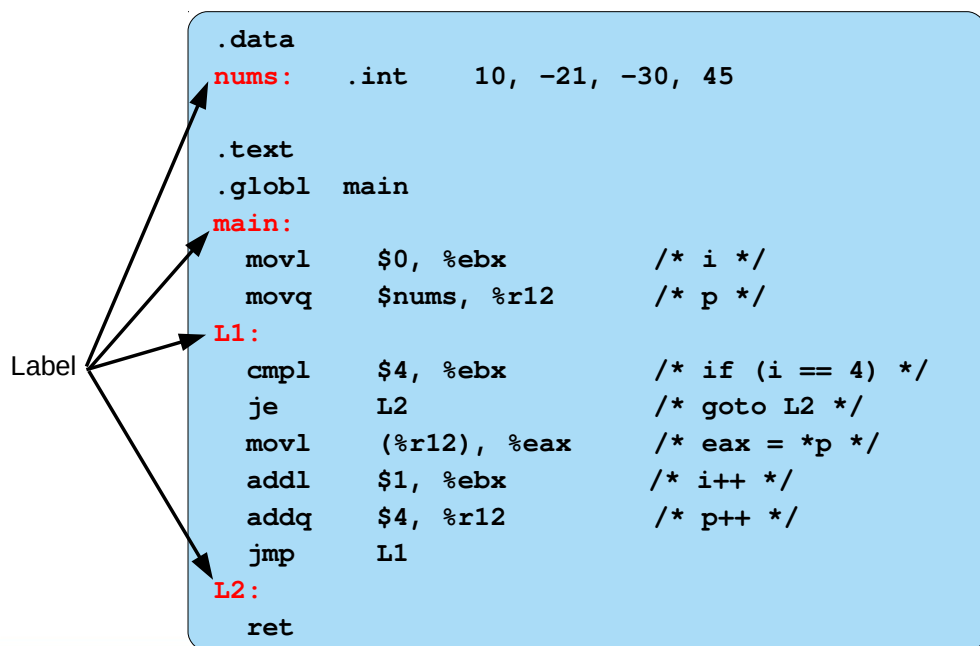


Programa em Assembly

- Seção de dados
 - .data
 - .section .data
- Seção de código
 - .text
 - .section .text



Programa em Assembly



Programa em Assembly

- Labels são usados para nomear
 - Variáveis globais
 - Funções
 - Posições para *jumps*
- Labels iniciados por “.L” não são visíveis fora do arquivo (tabela de símbolos)
 - Comumente usados para os *jumps*



Armazenamento de Dados



Armazenamento de Dados

- Podemos ter:
 - Registradores da CPU
 - Variáveis globais
 - Variáveis locais (veremos na parte de funções)



Registradores

63	31	15	8	7	0
%rax	%eax	%ax	%ah	%al	
%rbx	%ebx	%bx	%bh	%bl	
%rcx	%ecx	%cx	%ch	%cl	
%rdx	%edx	%dx	%dh	%dl	
%rsi	%esi	%si		%sil	
%rdi	%edi	%di		%dil	
%rbp	%ebp	%bp		%bpl	
%rsp	%esp	%sp		%spl	
%r8	%r8d	%r8w		%r8b	
%r9	%r9d	%r9w		%r9b	
%r10	%r10d	%r10w		%r10b	
%r11	%r11d	%r11w		%r11b	
%r12	%r12d	%r12w		%r12b	
%r13	%r13d	%r13w		%r13b	
%r14	%r14d	%r14w		%r14b	
%r15	%r15d	%r15w		%r15b	

Armazenam valores inteiros e endereços (ponteiros)

- Podem ser usados como valores de 8, 16, 32 ou 64 bits

Algumas instruções demandam registradores específicos

%rbp e **%rsp** são usados como ponteiros para a pilha



Variáveis Globais

- Definidas na área de dados
- Sintaxe
`<nome> <tipo> <expressão>`
- Nome
 - Nome é um label
- Tipo
 - Pode ser numérico ou string
- Expressão
 - Usaremos valores escalares ou array
 - Valor **não** pode ser omitido



13

Variáveis Globais

- Tipos Inteiros

TIPO	SYNÔNIMO	TAMANHO
.byte	-	1 byte (8 bits)
.word	.short	2 bytes (16 bits)
.long	.int	4 bytes (32 bits)
.quad	-	8 bytes (64 bits)



14

Variáveis Globais

- String

TIPO	OBSERVAÇÃO
.ascii	Não adiciona o '\0' no final da string
.asciz	Adiciona o '\0' no final da string
.string	Adiciona o '\0' no final da string



Variáveis Globais

- Exemplos

```
.data

    num:    .int      1024
           i:    .short  0x33AF
    str01:   .ascii  "Hello\0"
    str02:   .asciz  "Hello"
    str03:   .string "Hello"

    vet:    .int  1,2,3,4,5,6
```



Instruções Assembly

Movimentação de Dados



17

Movimentação de Dados

- Instrução “mov”
`mov[b w l q] fonte, destino`
- Sufixo indica o tamanho do tipo
 - movb: 1 byte
 - movw: 2 bytes (word)
 - movl: 4 bytes (long-word)
 - movq: 8 bytes (quad-word)



18

Movimentação de Dados

- Instrução “mov”

mov[b w l q] fonte, destino

- Sufixo indica o tamanho do tipo

- Os argumentos “fonte” e “destino” devem ser compatíveis com o sufixo
 - Não podemos recuperar 1 byte da memória e colocar em um registrador que indique 8 bytes



Movimentação de Dados

- Instrução “mov”

mov[b w l q] fonte, destino

- Argumentos podem ser:

FONTE	DESTINO
Constante	Registrador
Constante	Memória
Registrador	Registrador
Registrador	Memória
Memória	Registrador

Não há memória → memória!!!



Argumentos

- Os argumentos das instruções podem ser
 - Constantes (também conhecidos como imediatos)
 - Registradores
 - Memória
 - Modo direto
 - Modo indireto
 - Modo base-deslocamento
 - Modo indexado
 - Modo indexado e escalado



Constantes e Registradores

- Constantes (imediatos) são precedidas por “\$”, seguido por valor inteiro em notação C
- Registradores são especificados pelo nome

```
movl    $1024, %eax
movl    $0xFF, %ebx
movb    $0,    %al
movl    %ebx,   %ecx
movq    %r12,   %r13
movabsq $0x11fff22aa33bb00, %rax /* const de 64 bits */
```



Constantes e Registradores

- “mov” aceita no máximo inteiro 32-bits (signed) como imediato (constante)
- “movabs” move um imediato 64-bits (signed) para um registrador

```
movabs <im64>, <reg64>
```

- Obs: Não há como mover um imediato 64-bits diretamente para a memória

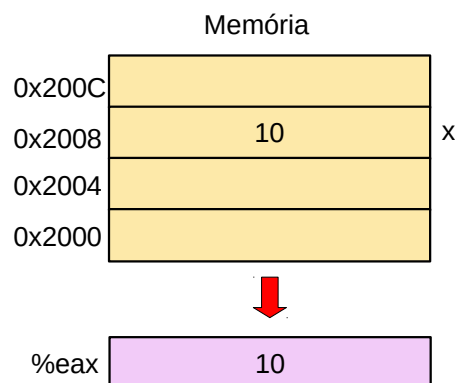


Memória: Modo Direto

- O endereço de memória é especificado por uma constante ou rótulo (*label*)

```
.data
    x:    .int    10

.text
main:
    movl  x, %eax
```

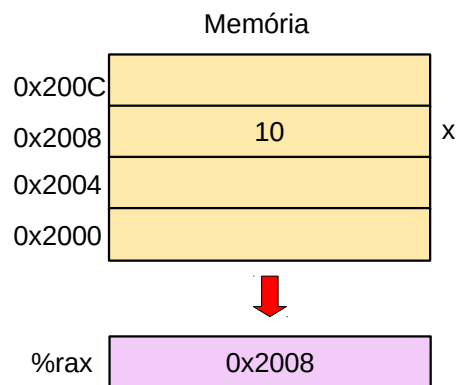


Memória: Modo Direto

- **Atenção:** se usarmos \$x, colocamos o endereço da variável no registrador
- Endereços são 64-bits

```
.data
    x:    .int    10

.text
main:
    movq  $x, %rax
```



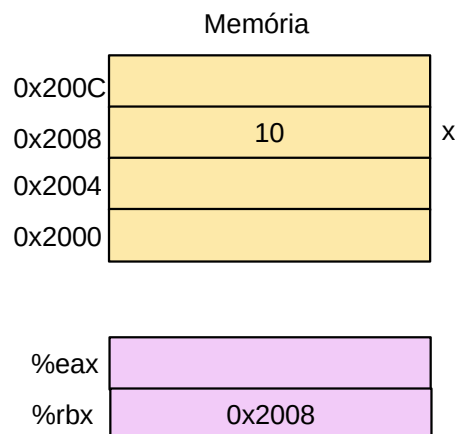
25

Memória: Modo Indireto

- O endereço de memória está em um registrador
- Formato: (R_{Base})

```
.data
    x:    .int    10

.text
main:
    movl  $1, (%rbx)
    movl  (%rbx), %eax
```



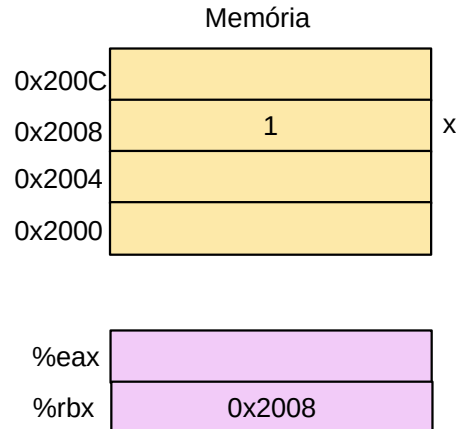
26

Memória: Modo Indireto

- O endereço de memória está em um registrador
- Formato: (R_{Base})

```
.data
    x:    .int    10

.text
main:
    movl    $1, (%rbx)
    movl    (%rbx), %eax
```



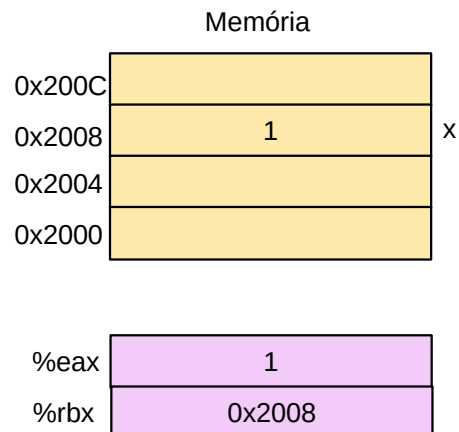
27

Memória: Modo Indireto

- O endereço de memória está em um registrador
- Formato: (R_{Base})

```
.data
    x:    .int    10

.text
main:
    movl    $1, (%rbx)
    movl    (%rbx), %eax
```



28

Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: $\text{Im}(\text{R}_{\text{Base}}) \rightarrow \text{Addr} = \text{Im} + \text{R}_{\text{Base}}$

```
.data
vet: .int 4, 5, 6, 7

.text
main:
    movl    $1, 4(%rbx)
    movl    8(%rbx), %eax
```

Memória

0x200C	7
0x2008	6
0x2004	5
0x2000	4

vet

%eax	
%rbx	0x2000

29

Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: $\text{Im}(\text{R}_{\text{Base}}) \rightarrow \text{Addr} = \text{Im} + \text{R}_{\text{Base}}$

```
.data
vet: .int 4, 5, 6, 7

.text
main:
    movl    $1, 4(%rbx)
    movl    8(%rbx), %eax
```

Memória

0x200C	7
0x2008	6
0x2004	5
0x2000	4

vet

%eax	
%rbx	0x2000

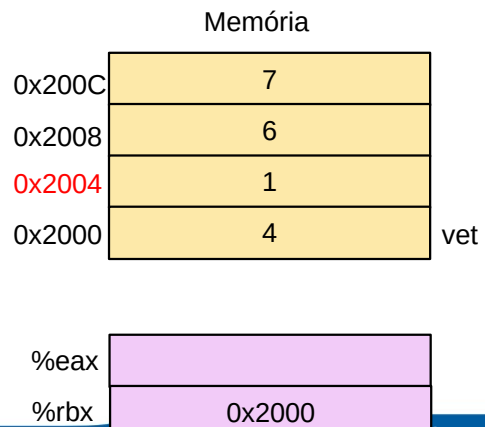
30

Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: $\text{Im}(\text{R}_{\text{Base}}) \rightarrow \text{Addr} = \text{Im} + \text{R}_{\text{Base}}$

```
.data
vet: .int 4, 5, 6, 7

.text
main:
    movl    $1, 4(%rbx)
    movl    8(%rbx), %eax
```

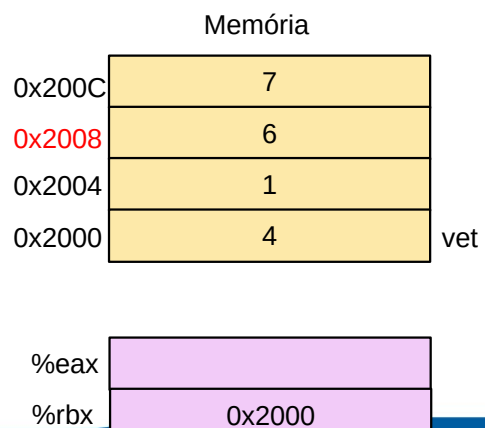


Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: $\text{Im}(\text{R}_{\text{Base}}) \rightarrow \text{Addr} = \text{Im} + \text{R}_{\text{Base}}$

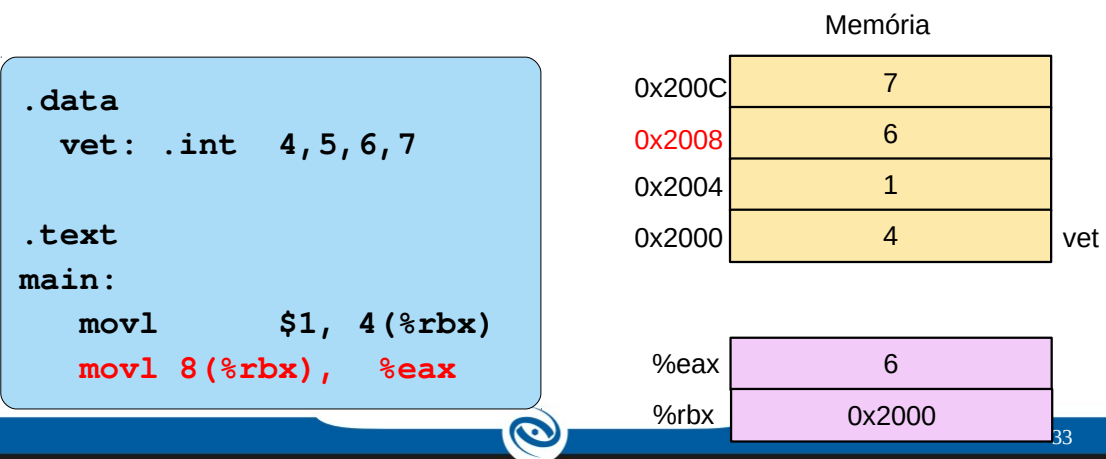
```
.data
vet: .int 4, 5, 6, 7

.text
main:
    movl    $1, 4(%rbx)
    movl    8(%rbx), %eax
```



Memória: Base-Deslocamento

- Base é um registrador tem um endereço de memória
- Deslocamento é um número inteiro (positivo ou negativo)
- Formato: $\text{Im}(\text{R}_{\text{Base}}) \rightarrow \text{Addr} = \text{Im} + \text{R}_{\text{Base}}$



Movimentação com Extensão

Movimentação com Extensão

- As instruções movs e movz são usadas para converter tipos **menores** em tipos **maiores**
- Um paralelo com a linguagem C

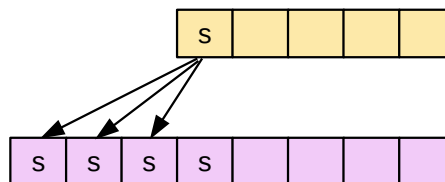
```
short s = 20;
int    i = 0;
i = (int)s;
```

```
unsigned char c = 20;
unsigned long l = 0;
l = (unsigned long)c;
```

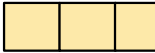
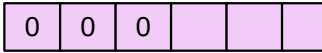


Movimentação com Extensão

- movs: extensão usa o bit de sinal
- Sufixo diferencia as extensões
 - movsbw
 - movsbl
 - movsbq
 - movswl
 - movswq
 - movslq

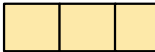
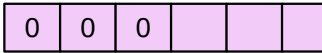


Movimentação com Extensão

- movz: extensão usa zero
- Sufixo diferencia as extensões
 - movzbw
 - movzbl 
 - movzbq
 - movzwl 
 - movzwq
 - ~~movzld~~: Não possui essa instrução!!!



Movimentação com Extensão

- movz: extensão usa zero
- Sufixo diferencia as extensões
 - movzbw
 - movzbl 
 - movzbq
 - movzwl 
 - movzwq
 - ~~movzld~~: Não possui essa instrução!!!

Instruções que escrevem um valor no registrador de 32 bits zeram a parte alta do registrador de 64 bits correspondente ...



Movimentação com Extensão

- Exemplos

```
movsbl (%r12), %eax
movzbl    %al, %ebx
movslq    %ebx, %rcx

# zera a parte alta de %rdx
# substituto do movzql
movl      $12, %edx
```



Operações Aritméticas



Operações Aritméticas

`inc opr`
`dec opr`
`neg opr`

} Operando pode ser registrador ou memória

```

incl %eax    /* %eax = %eax + 1 */
incl (%rdx)  /* (%rdx) = (%rdx) + 1 */
decq %rax    /* %rax = %rax - 1 */
negl %ebx    /* %ebx = -%ebx */
  
```



Operações Aritméticas

`add fonte, destino` `/* d = d + f */`
`sub fonte, destino` `/* d = d - f */`

```

addl    %ebx, %eax    /* %eax = %eax + %ebx */
addq    $4, %rbx      /* %rbx = %rbx + 4 */
addl    4(%r12), %eax  /* %eax = %eax + 4(%r12) */
subl    %ebx, %eax     /* %eax = %eax - %ebx */
  
```



Operações Aritméticas

Multipliação "signed"

```
imul reg1, reg2      /* r2 = r2 * r1 */
imul mem, reg         /* r = r * m */
imul im, reg1, reg2  /* r2 = im * r1 */
imul im, mem, reg     /* r = im * m */
```

Não há "imulb" !!!

```
imull %eax, %esi      /* %esi = %esi * %eax */
imulq %rax, %rsi      /* %rsi = %rsi * %rax */
imull (%rbx), %esi    /* %esi = %esi * (%rbx) */
imulq val, %rsi       /* %rsi = %rsi * val */
imull $5, %eax, %esi  /* %esi = %eax * 5 */
```

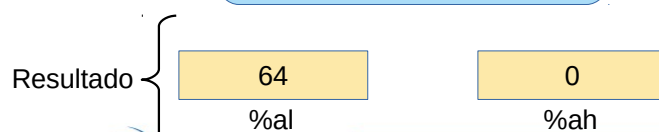
51

Operações Aritméticas

idivb <divisor> } <divisor> deve ser memória ou registrador

- Divide %ax por <divisor> (Divisão "signed")
- Resultado
 - %al → quociente
 - %ah → resto

```
movw $128, %ax
movb $2, %bl
idivb %bl
```



52

Operações Aritméticas

idivw <divisor> } <divisor> deve ser memória ou registrador

- Divide %dx:%ax por <divisor> (Divisão "signed")
- Resultado

- %ax → quociente
- %dx → resto

```
movw $0, %dx
movw $129, %ax
movw $2, %bx
idivw %bx
```

Resultado

64

%ax

1

%dx

53

Operações Aritméticas

idivl <divisor> } <divisor> deve ser memória ou registrador

- Divide %edx:%eax por <divisor> (Divisão "signed")
- Resultado

- %eax → quociente
- %edx → resto

```
movl $0, %edx
movl $125, %eax
movl $10, %ebx
idivl %ebx
```

Resultado

12

%eax

5

%edx

54

Operações Aritméticas

`idivq <divisor>` } <divisor> deve ser memória ou registrador

- Divide %rdx:%rax por <divisor> (Divisão “signed”)
- Resultado

- %rax → quociente
- %rdx → resto

```
movq $0, %rdx
movq $125, %rax
movq $10, %rbx
idivq %rbx
```

Resultado

12

%rax

5

%rdx

Operações Bit a Bit

`and fonte, destino` /* d = d & f */

`or fonte, destino` /* d = d | f */

`xor fonte, destino` /* d = d ^ f */

`not destino` /* d = ~d */

```
andl $0x7FFFFFFF,%eax /* %eax = %eax & 0x7FFFFFFF */
andl %ebx, %eax /* %eax = %eax & %ebx */
orl (%rcx),%eax /* %eax = %eax | (%rcx) */
xorl %eax, %ebx /* %ebx = %ebx ^ %eax */
notq %rax /* %rax = ~%rax */
```


Deslocamento

```
shl im, destino    /* d = d << im          */
shr im, destino    /* d = d >> im (lógico) */
sar im, destino    /* d = d >> im (arit)   */
```

```
shll $2, (%rax) /* (%rax) = (%rax) << 2      */
shrq $16, %rbx  /* %rbx = %rbx >> 16 (lógico) */
sarl $3, %ebx   /* %ebx = %ebx >> 3  (arit)   */
```



Deslocamento

```
shl %cl, destino  /* d = d << %cl          */
shr %cl, destino  /* d = d >> %cl (lógico) */
sar %cl, destino  /* d = d >> %cl (arit)   */
```

```
shll %cl, (%rax) /* (%rax) = (%rax) << %cl      */
shrq %cl, %rbx   /* %rbx = %rbx >> %cl (lógico) */
sarl %cl, %ebx   /* %ebx = %ebx >> %cl (arit)   */
```

