Software Básico

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
NORTE DE MINAS GERAIS

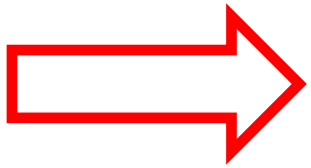
Esqueleto de programa em Assembly

- Programas em ***assembly*** são **divididos em seções**, e o programa apresentado contém **duas seções**.
  - A ***.section .data*** contém as variáveis globais do programa (como este programa não usa variáveis globais, esta seção está vazia)
  - A ***.section .text*** contém os comandos a serem executados quando o programa for colocado em execução.

Esqueleto de programa em Assembly

```c
int main()
{

    return 0;
}
```



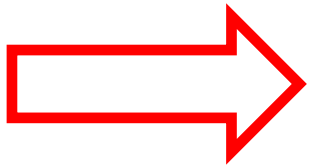
```
1 .section .data
2 .section .text
3 .globl _start
4 _start:
5 movq $60, %rax
6 movq $13, %rdi
7 syscall
```

- O rótulo **_start** é especial e deve sempre estar presente em um programa **assembly**.
- Ele corresponde ao **endereço da primeira instrução** do programa que será executada, e deve ser declarada como **globl** (linha 3).

Esqueleto de programa em Assembly

```c
int main()
{

    return 0;
}
```



```
1 .section .data
2 .section .text
3 .globl _start
4 _start:
5 movq $60, %rax
6 movq $13, %rdi
7 syscall
```

- O rótulo **_start** indica o local onde a execução do programa deve iniciar.

Execução do programa

- **as** exemplo1.s -o exemplo1.o
  - ***as*****:** *The portable GNU assembler*
  - converte um programa escrito em *assembly* (**fonte.s**) num arquivo objeto (**fonte.o**). Um **arquivo objeto** é uma versão incompleta de um arquivo executável.
- **ld** exemplo1.o -o exe
  - ***ld****: The GNU linker*
  - ***ld*** combina arquivos **.objeto** e gera um arquivo executável.
- O comando **./**exe invoca o sistema operacional (***loader***) que o coloca em execução.

GNU Binutils

- The GNU ***Binutils*** are a collection of binary tools. The main ones are:
  - **ld** - the GNU linker.
  - **as** - the GNU assembler.

  - **Instalação**:
    - sudo apt-get update –y
    - sudo apt-get install -y binutils-common

Esqueleto de programa em Assembly

```
1 long int a, b;
2 int main ( );
3 {
4    a=7;
5    b=7;
6    a = a+b;
7    return a;
8 }
```

```
1  .section .data
2  a: .quad 0
3  b: .quad 0
4  .section .text
5  .globl _start
6  _start:
7  movq $7, a
8  movq $7, b
9  movq a, %rax
10 movq b, %rbx
11 addq %rbx,%rax
12 movq $60, %rax
13 movq %rbx, %rdi
14 syscall
```

Chamadas ao sistema Linux

```
...
12 movq $60, %rax
13 movq %rbx, %rdi
14 syscall
```

- A chamada ao sistema é executada pelo comando ***syscall***. Os dois comandos anteriores são os parâmetros desta chamada.

- O valor em %rax indica qual serviço é pedido (terminar a execução do programa) enquanto que o valor em %rdi indica qual o valor a ser colocado na variável de ambiente $?.

Linguagem Assembly

- A linguagem de montagem (*Assembly*) x86-64 possui dois padrões de programação:
  - **Sintaxe *Intel***
    - Dominante no sistema operacionais *Microsoft Windows;*
  - **Sintaxe *AT&T***
    - Dominante nos sistemas operacionais Unix e Linux;

Linguagem Assembly

- # para comentário

- **Valor imediato**:
  - ***$valor***
  - Valor decimal: $15
  - Valor hexadecimal: $0x0A
- **Constante**:
  - decimal 15
  - hexadecimal 0x0A

Linguagem Assembly

- **Declaração de Variáveis globais**
  - as variáveis não estão associados a nenhum **tipo.**
  - **Sintaxe**:

<Identificador> : .<quantidade de bytes> <valor>

.section .data

a: .*byte* #(8 *bits*)
b: .*word* #(16 *bits*)
c: .*long* #(32 *bits*)
d: .*quad* #(64 *bits*)

Observação: <valor> é opcional

Linguagem Assembly

- **Declaração de vetor global**
  - A forma de criar um vetor com valores fixos. Basta listá-los lado a lado na seção data, ao lado do rótulo associado àquele vetor
  - ***<Identificador>*** **:** **.*<quantidade de bytes>*** ***<valores separados por vírgulas>***

.section .data

vetorA: .quad 3, 67, 34, 222, 45, 75, 54, 34, 44, 33, 22, 66, 0

Linguagem Assembly

- **Modelo de endereçamento**:
  - Os operandos de uma instrução em *Assembly* podem variar de acordo com o local em que o dado se encontra. Como exemplo, observe a diferença entre as instruções
  - movq %rax, %rbx (***endereçamento registrador***)
  - movq $0, %rbx (***endereçamento imediato***);
  - movq A, %rbx (***endereçamento direto***);
  - movq (%rbx) , %rax (***endereçamento indireto***)
  - movq A(,%rdi,4), %rbx. (***endereçamento indexado***)

Linguagem Assembly

- No modo de **endereçamento indireto**, um dos parâmetros indica uma referência indireta a memória (normalmente por um ***registrador entre parênteses)***.
  - Por exemplo, a instrução **movq (%rbx), %rax** copia o conteúdo do endereço contido no registrador %rbx para o registrador %rax, algo como:
    - *%rax = M[%rbx]*
    - Onde M[%rbx] indica o conteúdo de memória indexado por %rbx.

Linguagem Assembly

- **Endereçamento indireto:**
  - **Dupla indireção:** ((%reg)) não é válido.
  - Somente um operando pode ser indireto.
    - Por isso, movq (%rbx), (%rax) não é válido.
  - É possível usar endereçamento indireto com rótulos, ou seja, são válidas as instruções do tipo movq (A), %rax.

Linguagem Assembly

- **Endereçamento indexado**:
  - No endereçamento indexado, a instrução usa um "**endereço base**" e um deslocamento.
  - Um exemplo é a instrução
    - **movq *vetorA*(,%rdi, 4), %rbx**,

      que usa "***vetorA***" como base e %rdi x 4 como deslocamento.
  - A instrução pode ser melhor entendida pela fórmula:
    - *%rbx = Memória*[&***vetorA*** + %rdi*4], onde &***vetorA*** indica o endereço de "***vetorA***".

Acessando os dados - operandos

$*Imm* = indica um valor imediato ***Imm;***

E_a = indica o registrador **a;**

R[E_a] = indica o valor contido no registrador **a;**

Type	Form	Operand value	Name
Immediate	$\$Imm$	Imm	Immediate
Register	E_a	$R[E_a]$	Register
Memory	Imm	$M[Imm]$	Absolute
Memory	(E_a)	$M[R[E_a]]$	Indirect
Memory	$Imm(E_b)$	$M[Imm + R[E_b]]$	Base + displacement
Memory	(E_b, E_i)	$M[R[E_b] + R[E_i]]$	Indexed
Memory	$Imm(E_b, E_i)$	$M[Imm + R[E_b] + R[E_i]]$	Indexed
Memory	$(, E_i, s)$	$M[R[E_i] \cdot s]$	Scaled indexed
Memory	$Imm(, E_i, s)$	$M[Imm + R[E_i] \cdot s]$	Scaled indexed
Memory	(E_b, E_i, s)	$M[R[E_b] + R[E_i] \cdot s]$	Scaled indexed
Memory	$Imm(E_b, E_i, s)$	$M[Imm + R[E_b] + R[E_i] \cdot s]$	Scaled indexed

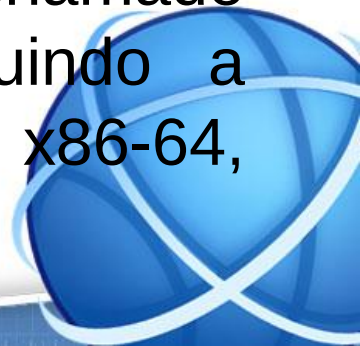
Registradores x86-64

64bits	32bits	16bits	8bits
r0 (rax)	r0d (eax)	r0w (ax)	r0l (al)
r1 (rcx)	r1d (ecx)	r1w (cx)	r1l (cl)
r2 (rdx)	r2d (edx)	r2w (dx)	r2l (dl)
r3 (rbx)	r3d (ebx)	r3w (bx)	r3l (bl)
r4 (rsp)	r4d (esp)	r4w (sp)	
r5 (rbp)	r5d (ebp)	r5w (bp)	
r6 (rsi)	r6d (esi)	r6w (si)	
r7 (rdi)	r7d (edi)	r7w (di)	
r8 ~ r15			

Registradores x86-64

- **Registradores**
  - Um registrador importante, mas que não está listado acima é o **%rip** (***instruction pointer***). Ele armazena sempre o endereço da próxima instrução a ser executada.

  - Todas as arquiteturas baseadas na ***arquitetura von Neumann*** tem um registrador equivalente. Na maioria delas, este registrador é chamado "***program counter***", ou PC, mas seguindo a nomenclatura dos processadores da linha x86-64, eles são chamados ***instruction pointer***.



Registradores x86-64

- **Registradores com funcionalidades específicas**

Registrador	Uso
%rax	Temporário, 1^o retorno procedimento
%rbx	Temporário
%rcx	4^o parâmetro inteiro
%rdx	3^o parâmetro inteiro
%rsp	*stack-pointer*
%rbp	*base-pointer*
%rsi	2^o parâmetro inteiro
%rdi	1^o parâmetro inteiro
%r8	5^o parâmetro inteiro
%r9	6^o parâmetro inteiro
%r10	Temporário
%r11	Temporário
%r12-r15	Temporário

Linguagem Assembly

Todas as instruções contendo mais de um operando, guardam seu resultado em op2.

Para especificar o tamanho do(s) operando(s), acrescentar sufixo na instrução: B,W,L ou Q.

Sufixo	Nome	Tamanho
B	**BYTE**	**1 byte** (8 bits)
W	**WORD**	**2 bytes** (16 bits)
L	**LONG (DOUBLE)**	**4 bytes** (32 bits)
Q	**QUAD WORD**	**8 bytes** (64 bits)
S	**SINGLE FLOAT**	**4 bytes** (32 bits)
D	**DOUBLE FLOAT**	**8 bytes** (64 bits)

Linguagem Assembly

Três tipos básicos de operandos:

- **Imediato** – uma constante inteira (8, 16, 32 , ou 64 bits):
  - Valor é codificado dentro da instrução($valor);
- **Registrador** – nome de um registrador:
  - O nome do registrador é codificado dentro da instrução - %rax;
- **Memória** – referência a uma posição na memória:
  - O endereço da memória é codificado dentro da instrução, ou um registrador contém o endereço de uma posição de memória;

Linguagem Assembly

- **Instrução mov s (origem) para d (destino) de acordo com seu tamanho:**
- **mov[b | w | l | q] s , d move s to d**
  - **movb** $10,%ah **; para *byte 8bits***
  - **movw** %bx,%ax **; para word 16*bits***
  - **movl** $ebx,%eax **; para *long(double) 32bits***
  - **movq** $10,%rax **;para *qword 64bits***

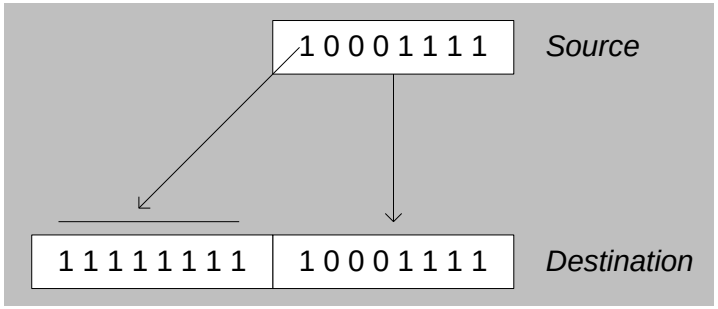
Linguagem Assembly

- **Instrução mov op1, op2**
  - **copia em op1 conteúdo de op2**

Mnemonic	Opcode	Description
MOV *reg/mem8*, *reg8*	88 */r*	Move the contents of an 8-bit register to an 8-bit destination register or memory operand.
MOV *reg/mem16*, *reg16*	89 */r*	Move the contents of a 16-bit register to a 16-bit destination register or memory operand.
MOV *reg/mem32*, *reg32*	89 */r*	Move the contents of a 32-bit register to a 32-bit destination register or memory operand.
MOV *reg/mem64*, *reg64*	89 */r*	Move the contents of a 64-bit register to a 64-bit destination register or memory operand.
MOV *reg8*, *reg/mem8*	8A */r*	Move the contents of an 8-bit register or memory operand to an 8-bit destination register.
MOV *reg16*, *reg/mem16*	8B */r*	Move the contents of a 16-bit register or memory operand to a 16-bit destination register.
MOV *reg32*, *reg/mem32*	8B */r*	Move the contents of a 32-bit register or memory operand to a 32-bit destination register.
MOV *reg64*, *reg/mem64*	8B */r*	Move the contents of a 64-bit register or memory operand to a 64-bit destination register.
MOV *reg16/32/64/mem16*, *segReg*	8C */r*	Move the contents of a segment register to a 16-bit, 32-bit, or 64-bit destination register or to a 16-bit memory operand.
MOV *segReg*, *reg/mem16*	8E */r*	Move the contents of a 16-bit register or memory operand to a segment register.

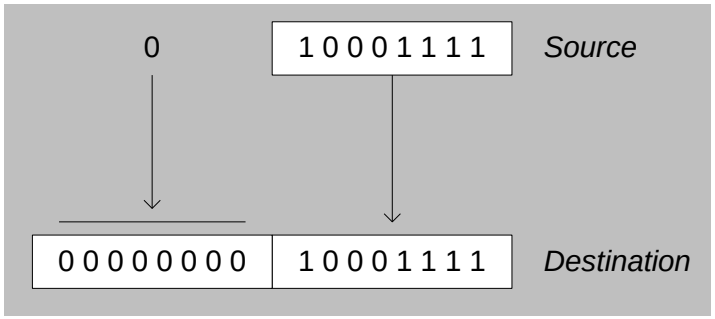
Linguagem Assembly

- **Instrução mov s (origem) para d (destino) de acordo com seu tamanho:**
- **movs[b | w | l | q] s , d move s to d**
- **movs com sinal**
  - movs[bw|bl|bq|wl|wq|lq] s, d
  - **bw *byte*** para ***word***
  - **bl *byte*** para ***long*** *(dword - double)*
  - **bq *byte*** para ***qword***
  - **wl *word*** para ***long***
  - **wq *word*** para ***qword***
  - **lq *long*** para ***qword***



Linguagem Assembly

- **Instrução mov s (origem) para d (destino) de acordo com seu tamanho:**
- **movz[b | w | l | q] s , d move s to d**
- **movz com extensão zero**
  - movz[bw|bl|bq|wl|wq|lq] s, d
  - **bw *byte*** para ***word***
  - **bl *byte*** para ***long*** *(dword - double)*
  - **bq *byte*** para ***qword***
  - **wl *word*** para ***long***
  - **wq *word*** para ***qword***
  - **lq *long*** para ***qword***



Linguagem Assembly

- **Instruções aritméticas**: **add [b | w | l | q] :**
  - **Combinações de parâmetros:**

```
add <reg>, <reg>
add <mem>, <reg>
add <reg>, <mem>
add <con>, <reg>
add <con>, <mem>
```

  - **Exemplo:**

addq	reg_1, reg_2	$reg_2 \leftarrow reg_2 + reg_1$
addq	reg, mem	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem] + reg$
addq	$imm32, reg$	$reg \leftarrow reg + imm32$
addq	$imm32, mem$	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem] + imm32$
addq	mem, reg	$reg \leftarrow reg + \mathrm{M}[mem]$

Linguagem Assembly

- **Instruções aritméticas**: **sub [b | w | l | q] :**
  - **Combinações de parâmetros:**

```
add <reg>, <reg>
add <mem>, <reg>
add <reg>, <mem>
add <con>, <reg>
add <con>, <mem>
```

  - **Exemplo:**

subq	reg_1, reg_2	$reg_2 \leftarrow reg_2 - reg_1$
subq	reg, mem	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem] - reg$
subq	$imm32, reg$	$reg \leftarrow reg - imm32$
subq	$imm32, mem$	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem] - imm32$
subq	mem, reg	$reg \leftarrow reg - \mathrm{M}[mem]$

Linguagem Assembly

- **Instruções aritméticas**: **mul [b | w | l | q]**
  - **Realiza operação de multiplicação sem sinal;**

MUL *reg/mem8*	F6 /4	Multiplies an 8-bit register or memory operand by the contents of the AL register and stores the result in the AX register.
MUL *reg/mem16*	F7 /4	Multiplies a 16-bit register or memory operand by the contents of the AX register and stores the result in the DX:AX register.
MUL *reg/mem32*	F7 /4	Multiplies a 32-bit register or memory operand by the contents of the EAX register and stores the result in the EDX:EAX register.
MUL *reg/mem64*	F7 /4	Multiplies a 64-bit register or memory operand by the contents of the RAX register and stores the result in the RDX:RAX register.

Linguagem Assembly

- **Instruções aritméticas**: **imul [b | w | l | q]:**
  - **Realiza operação de multiplicação com sinal;**
  - **Combinações de parâmetros:**

```
imul <reg32>, <reg32>
imul <mem>, <reg32>
imul <con>, <reg32>, <reg32>
imul <con>, <mem>, <reg32>
```

  - **Exemplos:**

imulq	reg_1, reg_2	$reg_2 \leftarrow reg_2 \times reg_1$
imulq	mem, reg	$reg \leftarrow reg \times \mathrm{M}[mem]$
imulq	$imm32, reg$	$reg \leftarrow reg \times imm32$

Linguagem Assembly

- **Instruções aritméticas**: **div [b | w | l | q]**

Division Size	Dividend	Divisor	Quotient	Remainder	Maximum Quotient
Word/byte	AX	reg/mem8	AL	AH	255
Doubleword/word	DX:AX	reg/mem16	AX	DX	65,535
Quadword/doubleword	EDX:EAX	reg/mem32	EAX	EDX	$2^{32} - 1$
Double quadword/ quadword	RDX:RAX	reg/mem64	RAX	RDX	$2^{64} - 1$

Mnemonic	Opcode	Description
DIV *reg/mem8*	F6 /6	Perform unsigned division of AX by the contents of an 8-bit register or memory location and store the quotient in AL and the remainder in AH.
DIV *reg/mem16*	F7 /6	Perform unsigned division of DX:AX by the contents of a 16-bit register or memory operand store the quotient in AX and the remainder in DX.
DIV *reg/mem32*	F7 /6	Perform unsigned division of EDX:EAX by the contents of a 32-bit register or memory location and store the quotient in EAX and the remainder in EDX.
DIV *reg/mem64*	F7 /6	Perform unsigned division of RDX:RAX by the contents of a 64-bit register or memory location and store the quotient in RAX and the remainder in RDX.

Linguagem Assembly

- **Instruções aritméticas**: **idiv [b | w | l | q]**

Division Size	Dividend	Divisor	Quotient	Remainder	Quotient Range
Word/byte	AX	reg/mem8	AL	AH	–128 to +127
Doubleword/word	DX:AX	reg/mem16	AX	DX	–32,768 to +32,767
Quadword/doubleword	EDX:EAX	reg/mem32	EAX	EDX	-2^{31} to $2^{31}-1$
Double quadword/ quadword	RDX:RAX	reg/mem64	RAX	RDX	-2^{63} to $2^{63}-1$

Mnemonic	Opcode	Description
IDIV *reg/mem8*	F6 /7	Perform signed division of AX by the contents of an 8-bit register or memory location and store the quotient in AL and the remainder in AH.
IDIV *reg/mem16*	F7 /7	Perform signed division of DX:AX by the contents of a 16-bit register or memory location and store the quotient in AX and the remainder in DX.
IDIV *reg/mem32*	F7 /7	Perform signed division of EDX:EAX by the contents of a 32-bit register or memory location and store the quotient in EAX and the remainder in EDX.
IDIV *reg/mem64*	F7 /7	Perform signed division of RDX:RAX by the contents of a 64-bit register or memory location and store the quotient in RAX and the remainder in RDX.

Linguagem Assembly

- **Instruções aritméticas**: **idiv [b | w | l | q]**

Division Size	Dividend	Divisor	Quotient	Remainder	Quotient Range
Word/byte	AX	reg/mem8	AL	AH	–128 to +127
Doubleword/word	DX:AX	reg/mem16	AX	DX	–32,768 to +32,767
Quadword/doubleword	EDX:EAX	reg/mem32	EAX	EDX	-2^{31} to $2^{31}-1$
Double quadword/ quadword	RDX:RAX	reg/mem64	RAX	RDX	-2^{63} to $2^{63}-1$

`idivq`	*reg*	`%rax` ← `%rdx` : `%rax` **DIV** *reg* `%rdx` ← `%rdx` : `%rax` **MOD** *reg*
`idivq`	*mem*	`%rax` ← `%rdx` : `%rax` **DIV** M[*mem*] `%rdx` ← `%rdx` : `%rax` **MOD** M[*mem*]

Linguagem Assembly

- **Instruções aritméticas**: **inc [b | w | l | q] op1**

Mnemonic	Opcode	Description
INC *reg/mem8*	FE /0	Increment the contents of an 8-bit register or memory location by 1.
INC *reg/mem16*	FF /0	Increment the contents of a 16-bit register or memory location by 1.
INC *reg/mem32*	FF /0	Increment the contents of a 32-bit register or memory location by 1.
INC *reg/mem64*	FF /0	Increment the contents of a 64-bit register or memory location by 1.

Linguagem Assembly

- **Instruções aritméticas**: **dec [b | w | l | q] op1**

Mnemonic	Opcode	Description
DEC *reg/mem8*	FE /1	Decrement the contents of an 8-bit register or memory location by 1.
DEC *reg/mem16*	FF /1	`inc <reg>` `inc <mem>` nent the contents of a 16-bit register or memory n by 1. `dec <reg>`
DEC *reg/mem32*	FF /1	`dec <mem>` nent the contents of a 32-bit register or memory location by 1.
DEC *reg/mem64*	FF /1	Decrement the contents of a 64-bit register or memory location by 1.

Linguagem Assembly

- **Instruções lógicas**: **and [b | w | l | q]**
  - **Realiza operação lógica AND;**
  - **Exemplos e combinações de parâmetros:**

andq	reg_1, reg_2	$reg_2 \leftarrow reg_2$ **AND** reg_1
andq	reg, mem	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem]$ **AND** reg
andq	$imm32, reg$	$reg \leftarrow reg$ **AND** $imm32$
andq	$imm32, mem$	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem]$ **AND** $imm32$
andq	mem, reg	$reg \leftarrow reg$ **AND** $\mathrm{M}[mem]$

Linguagem Assembly

- **Instruções lógicas**: **or [b | w | l | q]**
  - **Realiza operação lógica OR;**
  - **Exemplos e combinações de parâmetros:**

`orq`	reg_1, reg_2	$reg_2 \leftarrow reg_2 \ \mathbf{OR} \ reg_1$
`orq`	reg, mem	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem] \ \mathbf{OR} \ reg$
`orq`	$imm32, reg$	$reg \leftarrow reg \ \mathbf{OR} \ imm32$
`orq`	$imm32, mem$	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem] \ \mathbf{OR} \ imm32$
`orq`	mem, reg	$reg \leftarrow reg \ \mathbf{OR} \ \mathrm{M}[mem]$

Linguagem Assembly

- **Instruções lógicas**: **xor [b | w | l | q]**
  - **Realiza operação lógica XOR;**
  - **Exemplos e combinações de parâmetros:**

`xorq`	reg_1, reg_2	$reg_2 \leftarrow reg_2$ **XOR** reg_1
`xorq`	reg, mem	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem]$ **XOR** reg
`xorq`	$imm32, reg$	$reg \leftarrow reg$ **XOR** $imm32$
`xorq`	$imm32, mem$	$\mathrm{M}[mem] \leftarrow \mathrm{M}[mem]$ **XOR** $imm32$
`xorq`	mem, reg	$reg \leftarrow reg$ **XOR** $\mathrm{M}[mem]$

Linguagem Assembly

- **Instruções lógicas**: **not [b | w | l | q]**

Mnemonic	Opcode	Description
NOT *reg/mem8*	F6 /2	Complements the bits in an 8-bit register or memory operand.
NOT *reg/mem16*	F7 /2	Complements the bits in a 16-bit register or memory operand.
NOT *reg/mem32*	F7 /2	Complements the bits in a 32-bit register or memory operand.
NOT *reg/mem64*	F7 /2	Compliments the bits in a 64-bit register or memory operand.

Linguagem Assembly

- **Instruções lógicas**: **neg [b | w | l | q]**
  - **Realização a negação do valor especificado – inverte o sinal do valor;**
  - **Parâmetros:**

```
neg <reg>
neg <mem>
```

  - **Exemplo:**

```
negl %EBX
#  inverte o sinal de EBX para -EBX
#  inverte o sinal de -EBX para EBX
```

Linguagem Assembly

- **Instruções lógicas: shl [b | w | l | q]**
  - **Realiza operação de movimentação de bits a esquerda;**

`shlq`	$imm32, reg$	$reg \leftarrow reg << imm32$
`shlq`	$imm32, mem$	$mem \leftarrow mem << imm32$

Linguagem Assembly

- **Instruções lógicas: shr [b | w | l | q]**
  - **Realiza operação de movimentação de bits a direita;**

shrq	$imm32, reg$	$reg \leftarrow reg >> imm32$ (logical shift)
shrq	$imm32, mem$	$mem \leftarrow mem >> imm32$

Linguagem Assembly

- **Instruções : lea[b | w | l | q]**
  - **Retorna o endereço efetivo da rótulo especificado;**

`leaq`	*mem*, *reg*	*reg* ← *mem* (load effective address)

Referência