Software Básico

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

Linguagem Assembly

- **Rótulo (*Label*):**
  - O rótulo é sempre um conjunto de caracteres e letras terminadas por **dois pontos** e
  - É usado para simplificam endereçamento de uma instrução ou trecho de código;
  - Exemplo:
    - **Label1:**

Linguagem Assembly

- **Instruções de salto incondicional: jump endereço**
  - **jump label**
    - Transfere incondicionalmente o controle para um novo endereço sem salvar o valor atual do **registrador RIP**.

Linguagem Assembly

- **Instruções de salto incondicional: jump endereço**

Mnemonic	Opcode	Description
JMP *rel8off*	EB *cb*	Short jump with the target specified by an 8-bit signed displacement.
JMP *rel16off*	E9 *cw*	Near jump with the target specified by a 16-bit signed displacement.
JMP *rel32off*	E9 *cd*	Near jump with the target specified by a 32-bit signed displacement.
JMP *reg/mem16*	FF /4	Near jump with the target specified *reg/mem16*.
JMP *reg/mem32*	FF /4	Near jump with the target specified *reg/mem32*. (No prefix for encoding in 64-bit mode.)
JMP *reg/mem64*	FF /4	Near jump with the target specified *reg/mem64*.

  - ***rel8off***—*Signed 8-bit offset relative to the instruction pointer*.
  - ***rel16off***—*Signed 16-bit offset relative to the instruction pointer*.
  - ***rel32off***—*Signed 32-bit offset relative to the instruction pointer*.

Linguagem Assembly

- **Instruções de comparação: cmp reg1, reg2**
- **cmp** [b | w | l | q] **%rbx, %rax**
  - Esta instrução compara o SEGUNDO argumento com o primeiro (no caso, %rax com %rbx) colocando o resultado em um *bit* de um registrador especial (RFLAGS).
  - Esta instrução subtrai o operador origem do destino (destino – origem), mas não armazena o resultado da operação, apenas afeta o estado das *flags* de estado.

Linguagem Assembly

- **Instruções de comparação: cmp reg1, reg2**
- **cmp** [b | w | l | q] **%rbx, %rax**
  - Este registrador (RFLAGS) é afetado por vários tipos de instrução, e contém informações sobre a última instrução executada;
  - Os *bits* deste registrador podem ser testados individualmente, e no caso da operação ***jump if greater***, verifica se o bit ZF (zero flag) é igual a zero e se SF=OF (SF=Sign Flag e OF=Overflow Flag).

Linguagem Assembly

- Registrador de Flags (RFLAGS): Consiste em um grupo individual de *bits* de controle (*flag*) [**O D I T S Z A P C**]:
  - **OF** (***Overflow Flag***): Setada quando ocorre overflow aritmético.
  - **DF** (***Direction Flag***): Setada para auto-incremento em instruções de *string*.
  - **IF** (***Interruption Flag***): Permite que ocorram interrupções quando setada. Pode ser setada pelo sistema ou pelo usuário.

Linguagem Assembly

- Registrador de Flags (RFLAGS): Consiste em um grupo individual de *bits* de controle (*flag*) [**O D I T S Z A P C**]:
  - **TF (*Trap Flag*)** (*debug*): Usada por debugadores para executar programas passo a passo.
  - **SF (*Signal Flag*)**: Resetada (SF=0) quando um resultado for um número positivo ou zero e setada (SF=1) quando um resultado for negativo.
  - **ZF (*Zero Flag*)**: Setada quando um resultado for igual a zero.

Linguagem Assembly

- Registrador de Flags (RFLAGS): Consiste em um grupo individual de *bits* de controle (*flag*) [**O D I T S Z A P C**]:
  - **AF** (***Auxiliar Flag***): Setada quando há “vai um” na metade inferior de um byte.
  - **PF** (***Parity Flag***): Setada quando o número de bits 1 de um resultado for par.
  - **CF** (***Carry Flag***): Setada se houver “vai um” no bit de maior ordem do resultado. Também usada por instruções para tomadas de decisões.

Linguagem Assembly

- **Instruções de comparação: cmp op1, op2**
  - **cmp** [b | w | l | q] **[imm, reg/mem];**
  - **cmp** [b | w | l | q] **[reg, reg/mem];**
  - **cmp** [b | w | l | q] **[reg/mem, reg];**

Linguagem Assembly

- **Instruções de salto - Jump on Condition – J*cc:***
  - Verifica as *flags* de *status* do registrador *rFLAGS* e, se as flags atendem à condição especificada pelo código de condição no mnemônico (*cc*), salta para a instrução alvo localizada no ***offset*** especificado. Caso contrário, a execução continua com a instrução seguindo a instrução ***Jcc***.

Linguagem Assembly

- **Instruções de salto - Jump on Condition*:***
  - jg (***jump if greater***)
  - jge (***jump if greater or equal***)
  - jl (***jump if less***)
  - jle (***jump if less or equal***)
  - je (***jump if equal***)
  - jne (***jump if not equal***)

Linguagem Assembly

- **Instruções de salto - Jump on Condition – j*cc:***

`je` *Label*	`jz`	`ZF`	Equal / zero
`jne` *Label*	`jnz`	`~ZF`	Not equal / not zero
`jg` *Label*	`jnle`	`~(SF ^ OF) & ~ZF`	Greater (signed >)
`jge` *Label*	`jnl`	`~(SF ^ OF)`	Greater or equal (signed >=)
`jl` *Label*	`jnge`	`SF ^ OF`	Less (signed <)
`jle` *Label*	`jng`	`(SF ^ OF) | ZF`	Less or equal (signed <=)

Comando de desvio

- **Exemplo:**

```c
#include <stdio.h>

int a = 7;
int b = 10;
int r;
int main()
{
    if(a > 3 && b == 5)
    {
        r = 9;
    }

    return 0;
}
```

```
00000000000005fa <main>:
 5fa:   55                      push   %rbp
 5fb:   48 89 e5                mov    %rsp,%rbp
 5fe:   8b 05 0c 0a 20 00       mov    0x200a0c(%rip),%eax # 201010 <a>
 604:   83 f8 03                cmp    $0x3,%eax
 607:   7e 15                   jle    61e <main+0x24>
 609:   8b 05 05 0a 20 00       mov    0x200a05(%rip),%eax # 201014 <b>
 60f:   83 f8 05                cmp    $0x5,%eax
 612:   75 0a                   jne    61e <main+0x24>
 614:   c7 05 fe 09 20 00 09    movl   $0x9,0x2009fe(%rip) # 20101c <r>
 61b:   00 00 00
 61e:   b8 00 00 00 00          mov    $0x0,%eax
 623:   5d                      pop    %rbp
 624:   c3                      retq
```

Comando de desvio

- **Exemplo:**

```c
#include <stdio.h>

int a = 7;
int b = 10;
int r;
int main()
{
    if(a > 3 || b == 5)
    {
        r = 9;
    }

    return 0;
}
```

```
00000000000005fa <main>:
 5fa:   55                      push   %rbp
 5fb:   48 89 e5                mov    %rsp,%rbp
 5fe:   8b 05 0c 0a 20 00       mov    0x200a0c(%rip),%eax   # 201010 <a>
 604:   83 f8 03                cmp    $0x3,%eax
 607:   7f 0b                   jg     614 <main+0x1a>
 609:   8b 05 05 0a 20 00       mov    0x200a05(%rip),%eax   # 201014 <b>
 60f:   83 f8 05                cmp    $0x5,%eax
 612:   75 0a                   jne    61e <main+0x24>
 614:   c7 05 fe 09 20 00 09    movl   $0x9,0x2009fe(%rip)   # 20101c <r>
 61b:   00 00 00
 61e:   b8 00 00 00 00          mov    $0x0,%eax
 623:   5d                      pop    %rbp
 624:   c3                      retq
```

Comando de desvio

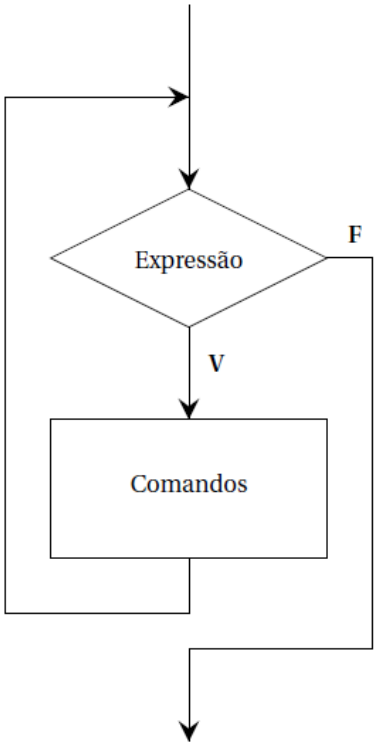
- **Exemplo:**

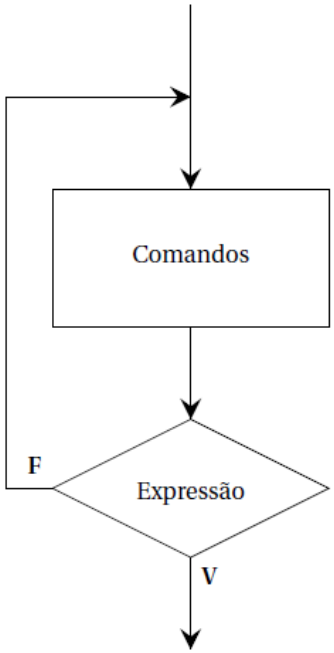
```c
#include <stdio.h>
int a = 7;
int r;
int main()
{
    switch( a )
    {
        case 3:
                r = 9;
                break;
        case 5:
                r = 11;
                break;
        case 7:
                r = 12;
    }
    return 0;
}
```

```
00000000000005fa <main>:
 5fa:   55                      push   %rbp
 5fb:   48 89 e5                mov    %rsp,%rbp
 5fe:   8b 05 0c 0a 20 00       mov    0x200a0c(%rip),%eax   # 201010 <a>
 604:   83 f8 05                cmp    $0x5,%eax
 607:   74 16                   je     61f <main+0x25>
 609:   83 f8 07                cmp    $0x7,%eax
 60c:   74 1d                   je     62b <main+0x31>
 60e:   83 f8 03                cmp    $0x3,%eax
 611:   75 22                   jne    635 <main+0x3b>
 613:   c7 05 ff 09 20 00 09    movl   $0x9,0x2009ff(%rip)   # 20101c <r>
 61a:   00 00 00
 61d:   eb 16                   jmp    635 <main+0x3b>
 61f:   c7 05 f3 09 20 00 0b    movl   $0xb,0x2009f3(%rip)   # 20101c <r>
 626:   00 00 00
 629:   eb 0a                   jmp    635 <main+0x3b>
 62b:   c7 05 e7 09 20 00 0c    movl   $0xc,0x2009e7(%rip)   # 20101c <r>
 632:   00 00 00
 635:   b8 00 00 00 00          mov    $0x0,%eax
 63a:   5d                      pop    %rbp
 63b:   c3                      retq
```

Comandos de repetição

Comandos repetitivos são aqueles que permitem que um conjunto de instruções seja repetido até que uma determinada **condição ocorra**.





Comandos de repetição

```c
#include <stdio.h>
int a;
int r;
int main()
{
    r = 1;
    for(a = 1; a<= 100; a++)
    {
        r++;
    }

    return 0;
}
```

```
00000000000005fa <main>:
 5fa:   55                      push   %rbp
 5fb:   48 89 e5                mov    %rsp,%rbp
 5fe:   c7 05 0c 0a 20 00 01    movl   $0x1,0x200a0c(%rip)   # 201014 <r>
 605:   00 00 00
 608:   c7 05 06 0a 20 00 01    movl   $0x1,0x200a06(%rip)   # 201018 <a>
 60f:   00 00 00
 612:   eb 1e                   jmp    632 <main+0x38>
 614:   8b 05 fa 09 20 00       mov    0x2009fa(%rip),%eax   # 201014 <r>
 61a:   83 c0 01                add    $0x1,%eax
 61d:   89 05 f1 09 20 00       mov    %eax,0x2009f1(%rip)   # 201014 <r>
 623:   8b 05 ef 09 20 00       mov    0x2009ef(%rip),%eax   # 201018 <a>
 629:   83 c0 01                add    $0x1,%eax
 62c:   89 05 e6 09 20 00       mov    %eax,0x2009e6(%rip)   # 201018 <a>
 632:   8b 05 e0 09 20 00       mov    0x2009e0(%rip),%eax   # 201018 <a>
 638:   83 f8 64                cmp    $0x64,%eax
 63b:   7e d7                   jle    614 <main+0x1a>
 63d:   b8 00 00 00 00          mov    $0x0,%eax
 642:   5d                      pop    %rbp
 643:   c3                      retq
```

Comandos de repetição

```c
#include <stdio.h>
int a;
int r;
int main()
{
    r = 1;
    a = 1;
    while(a <= 100)
    {
        r++;
        a++;
    }
    return 0;
}
```

```
0000000000000005fa <main>:
 5fa:   55                      push   %rbp
 5fb:   48 89 e5                mov    %rsp,%rbp
 5fe:   c7 05 0c 0a 20 00 01    movl   $0x1,0x200a0c(%rip)   # 201014 <r>
 605:   00 00 00
 608:   c7 05 06 0a 20 00 01    movl   $0x1,0x200a06(%rip)   # 201018 <a>
 60f:   00 00 00
 612:   eb 1e                   jmp    632 <main+0x38>
 614:   8b 05 fa 09 20 00       mov    0x2009fa(%rip),%eax   # 201014 <r>
 61a:   83 c0 01                add    $0x1,%eax
 61d:   89 05 f1 09 20 00       mov    %eax,0x2009f1(%rip)   # 201014 <r>
 623:   8b 05 ef 09 20 00       mov    0x2009ef(%rip),%eax   # 201018 <a>
 629:   83 c0 01                add    $0x1,%eax
 62c:   89 05 e6 09 20 00       mov    %eax,0x2009e6(%rip)   # 201018 <a>
 632:   8b 05 e0 09 20 00       mov    0x2009e0(%rip),%eax   # 201018 <a>
 638:   83 f8 64                cmp    $0x64,%eax
 63b:   7e d7                   jle    614 <main+0x1a>
 63d:   b8 00 00 00 00          mov    $0x0,%eax
 642:   5d                      pop    %rbp
 643:   c3                      retq
```

Comandos de repetição

```c
#include <stdio.h>
int a;
int r;
int main()
{
    r = 1;
    a = 1;
    do
    {
        r++;
        a++;
    }while(a <= 100);
    return 0;
}
```

```
00000000000005fa <main>:
 5fa:   55                      push   %rbp
 5fb:   48 89 e5                mov    %rsp,%rbp
 5fe:   c7 05 0c 0a 20 00 01    movl   $0x1,0x200a0c(%rip)   # 201014 <r>
 605:   00 00 00
 608:   c7 05 06 0a 20 00 01    movl   $0x1,0x200a06(%rip)   # 201018 <a>
 60f:   00 00 00
 612:   8b 05 fc 09 20 00       mov    0x2009fc(%rip),%eax   # 201014 <r>
 618:   83 c0 01                add    $0x1,%eax
 61b:   89 05 f3 09 20 00       mov    %eax,0x2009f3(%rip)   # 201014 <r>
 621:   8b 05 f1 09 20 00       mov    0x2009f1(%rip),%eax   # 201018 <a>
 627:   83 c0 01                add    $0x1,%eax
 62a:   89 05 e8 09 20 00       mov    %eax,0x2009e8(%rip)   # 201018 <a>
 630:   8b 05 e2 09 20 00       mov    0x2009e2(%rip),%eax   # 201018 <a>
 636:   83 f8 64                cmp    $0x64,%eax
 639:   7e d7                   jle    612 <main+0x18>
 63b:   b8 00 00 00 00          mov    $0x0,%eax
 640:   5d                      pop    %rbp
 641:   c3                      retq
```

Comandos de repetição

```
1  .section .text
2  .globl _start
3  _start:
4  movq $0, %rax
5  movq $10, %rbx
6  loop:
7  cmpq %rbx, %rax
8  jg fim_loop
9  add $1, %rax
10  jmp loop
11  fim_loop:
12  movq $60, %rax
13  movq %rbx, %rdi
14  syscall
```

Rótulo	Endereço	instrução (hexa)	instrução
_start	0x400078	48 c7 c0 00 00 00 00	mov $0x0,%rax
	0x40007f	48 c7 c3 0a 00 00 00	mov $0xa,%rbx
loop	0x400086	48 39 d8	cmp %rbx,%rax
	0x400089	7f 06	jg 400091 <fim_loop>
	0x40008b	48 83 c0 01	add $0x1,%rax
	0x40008f	eb f5	jmp 400086 <loop>
fim_loop	0x400091	48 c7 c0 3c 00 00 00	mov $0x3c,%rax
	0x400098	48 89 df	mov %rbx,%rdi
	0x40009b	0f 05	syscall

Comando de desvio

Com acesso a memória

```
1 int i, a;
2 main ( )
3 {
4    i=0; a=0;
5    while ( i<10 )
6    {
7         a+=i;
8         i++;
9    }
10   return a;
11 }
```

```
1  .section .data
2  i: .quad 0
3  a: .quad 0
4  .section .text
5  .globl _start
6  _start:
7  movq $0, i
8  movq $0, a
9  movq i, %rax
10 while:
11 cmpq $10, %rax
12 jge fim_while
13 movq a, %rdi
14 addq %rax, %rdi
15 movq %rdi, a
16 addq $1, %rax
17 movq %rax, i
18 jmp while
19 fim_while:
20 movq $60, %rax
21 syscall
```

Comandos de repetição

Sem acesso a memória

```
1 int i, a;
2 main ( )
3 {
4    i=0; a=0;
5    while ( i<10 )
6    {
7         a+=i;
8         i++;
9    }
10   return a;
11 }
```

```
1  .section .text
2  .globl _start
3  _start:
4  movq $0, %rax
5  while:
6  cmpq $10, %rax
7  jge fim_while
8  movq a, %rdi
9  addq %rax, %rdi
10 addq $1, %rax
11 jmp while
12 fim_while:
13 movq $60, %rax
14 syscall
```

Comandos de repetição

```
1 int i, a;
2 main ( )
3 {
4    i=0; a=0;
5    while ( i<10 )
6    {
7         a+=i;
8         i++;
9    }
10   return a;
11 }
```

```
1  .section .text
2  .globl _start
3  _start:
4  movq $0, %rax
5  while:
6  cmpq $10, %rax
7  jge fim_while
8  movq a, %rdi
9  addq %rax, %rdi
10 addq $1, %rax
11 jmp while
12 fim_while:
13 movq $60, %rax
14 syscall
```

```
1  .section .data
2  i: .quad 0
3  a: .quad 0
4  .section .text
5  .globl _start
6  _start:
7  movq $0, i
8  movq $0, a
9  movq i, %rax
10 while:
11 cmpq $10, %rax
12 jge fim_while
13 movq a, %rdi
14 addq %rax, %rdi
15 movq %rdi, a
16 addq $1, %rax
17 movq %rax, i
18 jmp while
19 fim_while:
20 movq $60, %rax
21 syscall
```

Comando de repetição

```
1 long int data_items[]={3, 67, 34,
                         222, 45, 75,
                         54, 34, 44,
                         33, 22, 11, 66, 0};
2 long int i, intmaior;
3 main ( long int argc, char **argv)
4 {
5 maior = data_items[0];
6 i=1;
7 while (data_items[i] != 0)
8 {
9 if (data_items[i] > maior)
10 maior = data_items[i];
11 i++;
12 }
13 return (maior);
14 }
```

```
1 .section .data
2 i: .quad 0
3 maior: .quad 0
4 data_items: .quad 3, 67, 34, 222,
                    45, 75, 54, 34,
                    44, 33, 22, 11, 66, 0
5 .section .text
6 .globl _start
7 _start:
8 movq $0, %rdi
9 movq data_items(, %rdi, 8), %rbx
10 movq $1, %rdi
11 loop:
12 movq data_items(, %rdi, 8), %rax
13 cmpq $0, %rax
14 je fim_loop
15 cmpq %rbx, %rax
16 jle fim_if
17 movq %rax, %rbx
18 fim_if:
19 addq $1, %rdi
20 jmp loop
21 fim_loop:
22 movq %rbx, %rdi
23 movq $60, %rax
24 syscall
```

Tipos de Dados

Nome	Descrição
.ascii	Text string
.asciz	Null-terminated text string
.string	Null-terminated text string
.byte	Byte value
.short	16-bit integer number
.int	32-bit integer number
.long	32-bit integer number (same as .int)
.quad	8-byte integer number

Tipos de Dados

- **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

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***)

Modelo de endereçamento

- **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

Exemplos

- **Problema 1**: percorrer vetor de *long long int* e calcular a soma;
- **Problema 2**: percorrer *string* e calcular o seu tamanho;
- **Problema 3**: percorrer *string* e contar a ocorrência de determinado caractere;
- **Desafio 1**: converter uma *string* em convertê-la em número inteiro;

Referência