


Simulador

https://www.mycompiler.io/pt/new/asm-x86_64?fork=Kn6PYf3VBmr

 myCompiler

Português ▼ | Recentes | Login | Inscreva-se

teste

Assembly ▼ ⓘ

```
1 section .data
2     msg db "Hello world!", 0ah
3
4 section .text
5     global _start
6
7 _start:
8     mov rax, 1
9     mov rdi, 1
10    mov rsi, msg
11    mov rdx, 13
12    syscall
13    mov rax, 60
14    mov rdi, 0
15    syscall
```

▶ Código de execução

📄 Guardar código

Entrada do programa

Saída do programa

(Execute o programa para exibir sua saída)

https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

Integer value	Name	<unistd.h> symbolic constant ^[1]	<stdio.h> file stream ^[2]
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

Registadores

RAX - Um registrador "acumulador", usado em instruções aritméticas.

RBX - Registrador base, usado para endereçamento de base.

RCX - Usado em instruções de repetição - loop.

RDY - Armazena dados durante operações entrada/saída.

RDI - Índice de destino em comandos de manipulação de string.

RSI - Índice de origem em comandos de manipulação de string.

} **Movimentação de dados**

RBP - Base do stack frame

RSP - Armazena endereço do topo da pilha

} **só devem ser usados preferencialmente para trabalhar com pilha.**

R8-R15 - Utilizados para armazenar variáveis temporárias

**Aloque o valor 30 para o registrador RAX
e o valor 45 para o registrador RBX, some.**

Por fim, utilize a syscall para imprimir.

Qual letra foi impressa?

```
section .bss
```

```
letra: resb 1
```

```
section .text
```

```
global _start
```

```
_start:
```

```
mov rax, 30
```

```
mov rbx, 45
```

```
add rax, rbx
```

```
mov [letra], rax
```

```
mov rax, 1
```

```
mov rdi, 1
```

```
mov rsi, letra
```

```
mov rdx, 1
```

```
syscall
```

```
mov rax, 60
```

```
mov rdi, 0
```

```
syscall
```



Faça um programa que receba 1 letra minúscula e converta para maiúscula.

```
section .bss
letra: resb 1
letra_mai: resb 1
```

```
section .text
global _start
```


```
_start:
```

```
    mov rax, 0
    mov rdi, 0
    mov rsi, letra
    mov rdx, 1
    syscall
```

```
    mov rbx, [letra]
    sub rbx, 32
    mov [letra_mai], rbx
```

```
    mov rax, 1
    mov rdi, 1
    mov rsi, letra_mai
    mov rdx, 1
    syscall
```

```
    mov rax, 60
    mov rdi, 0
    syscall
```



Faça um programa que calcule o tamanho de uma string.

```
section .data

msg: db "hello",0

section .text

strlen:
    mov rax, 0

    .loop:
        cmp byte [rdi+rax],0
        je .end
        inc rax
        jmp .loop
    .end:
        ret

global _start
_start:
    mov rdi, msg
    call strlen
    mov rdi, rax

    mov rax, 60
    syscall
```

```

1  section .data
2
3  msg: db "hello",0
4
5  section .text
6
7- strlen:
8      mov  rax, 0
9
10-   .loop:
11       cmp byte [rdi+rax],0
12       je  .end
13       inc rax
14       jmp .loop
15-   .end:
16       ret
17
18- global _start
19- _start:
20     mov rdi, msg
21     call strlen
22     mov rdi, rax
23
24     mov rax, 60
25     syscall
26

```