https://schweigi.github.io/assembler-simulator/

1- Considerar o código “Writes Hello World to the output”, comentar e identificar o que cada linha realiza, em resumo, descrever o código.

Obs: Considerar as Instruction Set

```
        JMP start
hello: DB "Hello World!" ; Variable
       DB 0      ; String terminator

start:
        MOV C, hello     ; Point to var
        MOV D, 232       ; Point to output
        CALL print
        HLT              ; Stop execution

print:                   ; print(C:*from, D:*to)
        PUSH A
        PUSH B
        MOV B, 0
.loop:
        MOV A, [C]       ; Get char from var
        MOV [D], A       ; Write to output
        INC C
        INC D
        CMP B, [C]       ; Check if end
        JNZ .loop        ; jump if not

        POP B
        POP A
        RET
```

DB - Define byte
JMP - Salto incondicional
CALL - Chamada
HLT - Interrompe a operação do processador
PUSH - Empurra valor para uma pilha
INC - Incrementa o registrador em 1
CMP - Compara dois valores
JNZ - Salta se não houver zero
POP - Remove valor da pilha
RET - Sai da subrotina

```
        JMP start
hello: DB "Hello World!" ; Variable
       DB 0       ; String terminator
```

RAM

31	15	72	101	108	108	111	32	87	111	114	108	100	33	0	6
2	2	6	3	232	56	24	0	50	0	50	1	6	1	0	3

TABELA ASCII

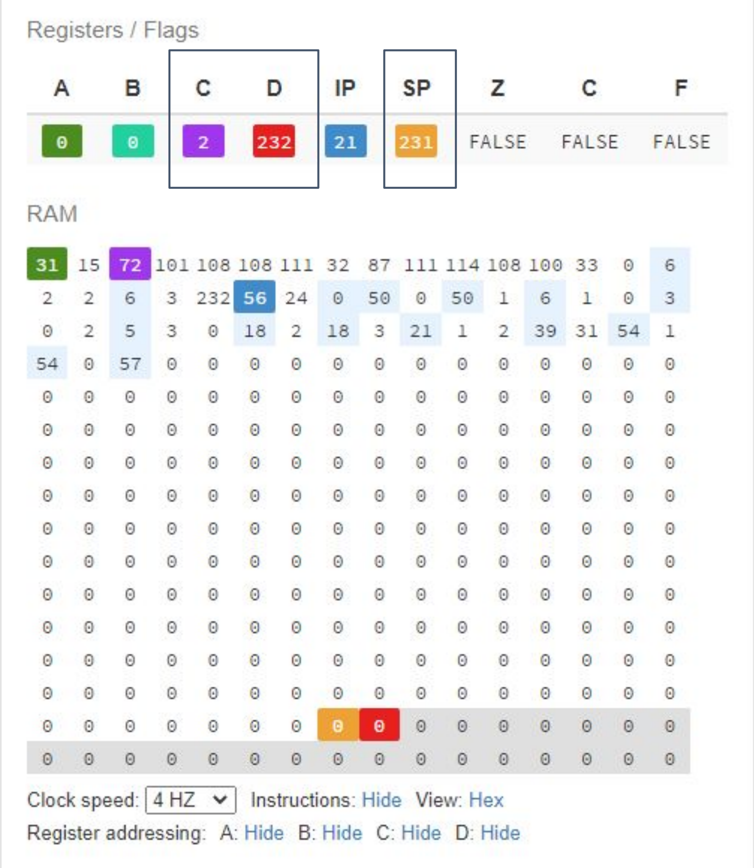
15 - Shift In
72 - H
101-e
108- l
108- l
111- o
32 - espaço
87 - W
111 - o
114 - r
108 - l
100 - d
33 - !

```
        JMP start
hello:  DB "Hello World!" ; Variable
        DB 0      ; String terminator

start:
        MOV C, hello    ; Point to var
        MOV D, 232      ; Point to output
        CALL print
        HLT             ; Stop execution

print:                  ; print(C:*from, D:*to)
        PUSH A
        PUSH B
        MOV B, 0
.loop:
        MOV A, [C]      ; Get char from var
        MOV [D], A      ; Write to output
        INC C
        INC D
        CMP B, [C]      ; Check if end
        JNZ .loop       ; jump if not

        POP B
        POP A
        RET
```



Registers / Flags

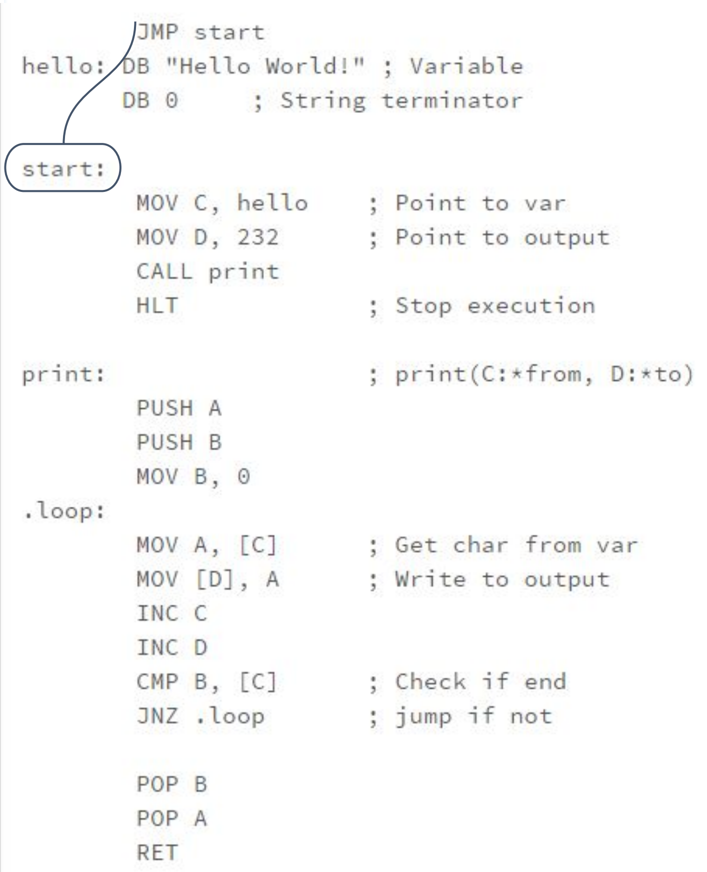
A	B	C	D	IP	SP	Z	C	F
0	0	2	232	21	231	FALSE	FALSE	FALSE

RAM

31	15	72	101	108	108	111	32	87	111	114	108	100	33	0	6
2	2	6	3	232	56	24	0	50	0	50	1	6	1	0	3
0	2	5	3	0	18	2	18	3	21	1	2	39	31	54	1
54	0	57	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Clock speed: 4 HZ Instructions: Hide View: Hex

Register addressing: A: Hide B: Hide C: Hide D: Hide



```
        JMP start
hello:  DB "Hello World!" ; Variable
        DB 0     ; String terminator

start:
        MOV C, hello    ; Point to var
        MOV D, 232      ; Point to output
        CALL print
        HLT             ; Stop execution

print:                  ; print(C:*from, D:*to)
        PUSH A
        PUSH B
        MOV B, 0
.loop:
        MOV A, [C]      ; Get char from var
        MOV [D], A      ; Write to output
        INC C
        INC D
        CMP B, [C]      ; Check if end
        JNZ .loop       ; jump if not

        POP B
        POP A
        RET
```

Registers / Flags

A	B	C	D	IP	SP	Z	C	F
0	0	2	232	21	231	FALSE	FALSE	FALSE

RAM

31	15	72	101	108	108	111	32								6
2	2	6	3	232	56	24									
0	2	5	3	0	18	2									
54	0	57	0	0	0	0	0								0
0	0	0	0	0	0	0	0								
0	0	0	0	0	0	0	0								
0	0	0	0	0	0	0	0								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Clock speed: 4 HZ Instructions: Hide View: Hex

Register addressing: A: Hide B: Hide C: Hide D: Hide

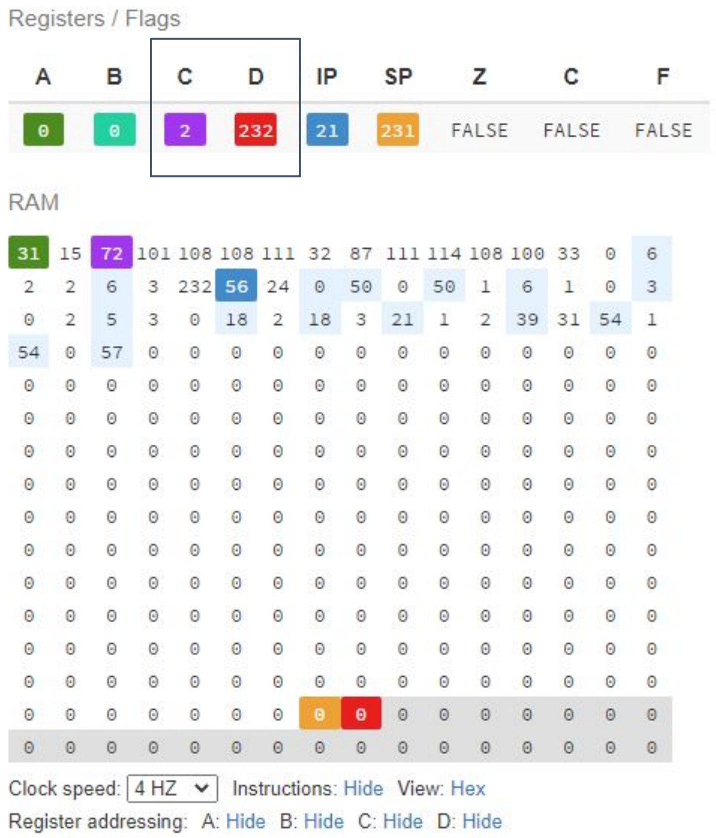
SP (stack pointer), aponta para o topo da pilha. Ele deve ser inicializado antes de utilizarmos a pilha.

```
        JMP start
hello: DB "Hello World!" ; Variable
       DB 0     ; String terminator

start:
        MOV C, hello    ; Point to var
        MOV D, 232      ; Point to output
        CALL print
        HLT             ; Stop execution

print:                  ; print(C:*from, D:*to)
        PUSH A
        PUSH B
        MOV B, 0
.loop:
        MOV A, [C]      ; Get char from var
        MOV [D], A      ; Write to output
        INC C
        INC D
        CMP B, [C]      ; Check if end
        JNZ .loop       ; jump if not

        POP B
        POP A
        RET
```



Registers / Flags

A	B	C	D	IP	SP	Z	C	F
0	0	2	232	21	231	FALSE	FALSE	FALSE

RAM

```
31  15  72 101 108 108 111  32  87 111 114 108 100  33   0   6
 2   2   6   3 232  56  24   0  50   0  50   1   6   1   0   3
 0   2   5   3   0  18   2  18   3  21   1   2  39  31  54   1
54   0  57   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Clock speed: 4 HZ Instructions: Hide View: Hex

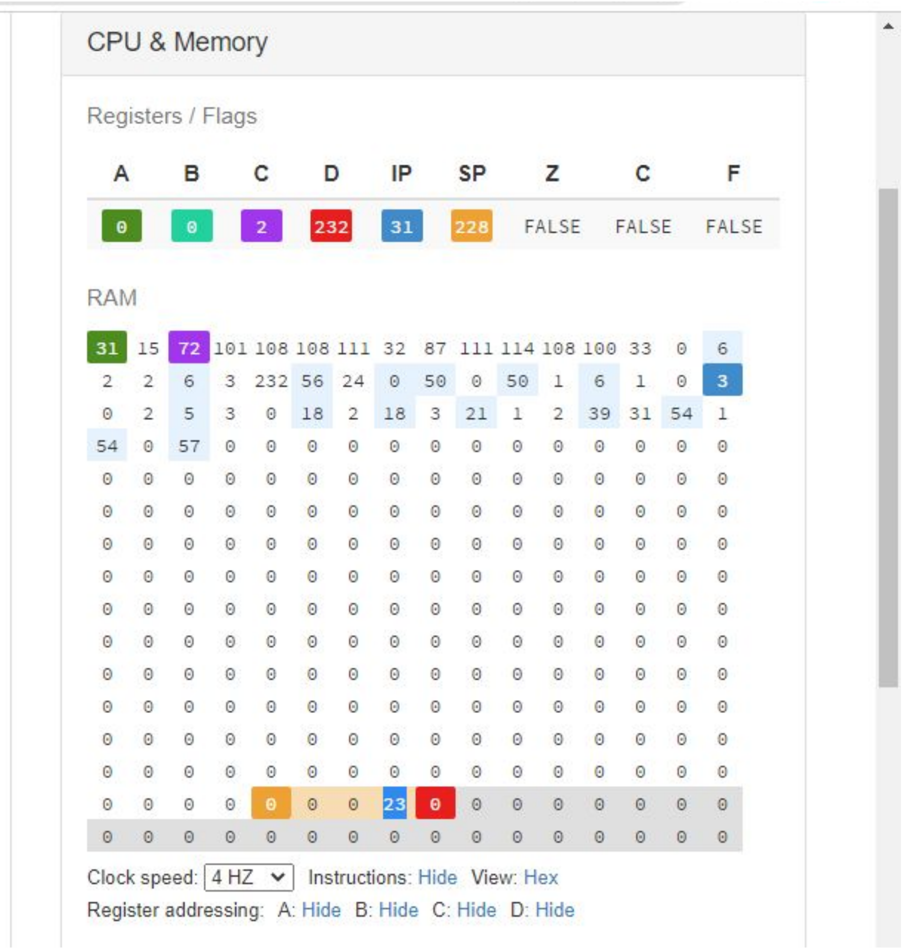
Register addressing: A: Hide B: Hide C: Hide D: Hide

```
        JMP start
hello: DB "Hello World!" ; Variable
       DB 0      ; String terminator

start:
        MOV C, hello    ; Point to var
        MOV D, 232      ; Point to output
        CALL print
        HLT             ; Stop execution

print:                  ; print(C:*from, D:*to)
        PUSH A
        PUSH B
        MOV B, 0
.loop:
        MOV A, [C]      ; Get char from var
        MOV [D], A      ; Write to output
        INC C
        INC D
        CMP B, [C]      ; Check if end
        JNZ .loop       ; jump if not

        POP B
        POP A
        RET
```



CPU & Memory

Registers / Flags

A	B	C	D	IP	SP	Z	C	F
0	0	2	232	31	228	FALSE	FALSE	FALSE

RAM

31	15	72	101	108	108	111	32	87	111	114	108	100	33	0	6
2	2	6	3	232	56	24	0	50	0	50	1	6	1	0	3
0	2	5	3	0	18	2	18	3	21	1	2	39	31	54	1
54	0	57	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Clock speed: 4 HZ Instructions: Hide View: Hex

Register addressing: A: Hide B: Hide C: Hide D: Hide

```
        JMP start
hello: DB "Hello World!" ; Variable
       DB 0      ; String terminator

start:
        MOV C, hello     ; Point to var
        MOV D, 232       ; Point to output
        CALL print
        HLT              ; Stop execution

print:                   ; print(C:*from, D:*to)
        PUSH A
        PUSH B
        MOV B, 0
.loop:
        MOV A, [C]       ; Get char from var
        MOV [D], A       ; Write to output
        INC C
        INC D
        CMP B, [C]       ; Check if end
        JNZ .loop        ; jump if not

        POP B
        POP A
        RET
```

Output

CPU & Memory

Registers / Flags

A	B	C	D	IP	SP	Z	C	F
72	0	2	232	34	228	FALSE	FALSE	FALSE

RAM

31 15 72 101 108 108 111 32 87 111 114 108 100 33 0 6

Labels

Name	Address	Value
.loop	31	3
hello	2	72 ('H')
print	24	50 ('2')
start	15	6

```
        JMP start
hello: DB "Hello World!" ; Variable
       DB 0      ; String terminator

start:
        MOV C, hello     ; Point to var
        MOV D, 232       ; Point to output
        CALL print
        HLT              ; Stop execution

print:                   ; print(C:*from, D:*to)
        PUSH A
        PUSH B
        MOV B, 0
.loop:
        MOV A, [C]       ; Get char from var
        MOV [D], A       ; Write to output
        INC C
        INC D
        CMP B, [C]       ; Check if end
        JNZ .loop        ; jump if not

        POP B
        POP A
        RET
```

Output

H

CPU & Memory

Registers / Flags

A	B	C	D	IP	SP	Z	C	F
72	0	2	232	37	228	FALSE	FALSE	FALSE

RAM

31	15	72	101	108	108	111	32	87	111	114	108	100	33	0	6
2	2	6	3	232	56	24	0	50	0	50	1	6	1	0	3
0	2	5	3	0	18	2	18	3	21	1	2	39	31	54	1
54	0	57	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

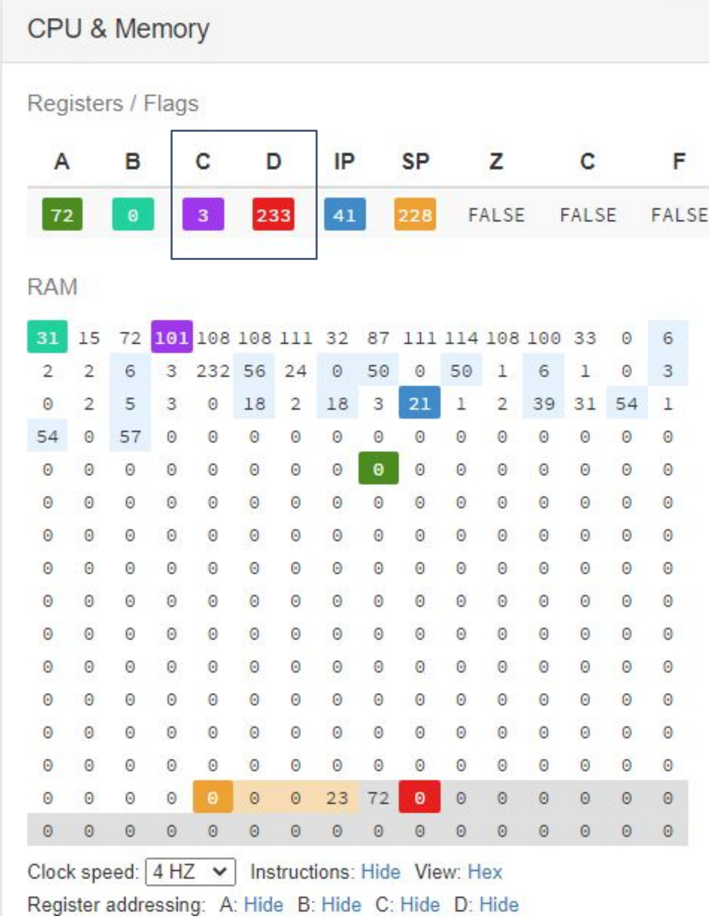
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	23	72	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
        JMP start
hello: DB "Hello World!" ; Variable
       DB 0      ; String terminator

start:
        MOV C, hello     ; Point to var
        MOV D, 232       ; Point to output
        CALL print
        HLT              ; Stop execution

print:                   ; print(C:*from, D:*to)
        PUSH A
        PUSH B
        MOV B, 0
.loop:
        MOV A, [C]       ; Get char from var
        MOV [D], A       ; Write to output
        INC C
        INC D
        CMP B, [C]       ; Check if end
        JNZ .loop        ; jump if not

        POP B
        POP A
        RET
```



CPU & Memory

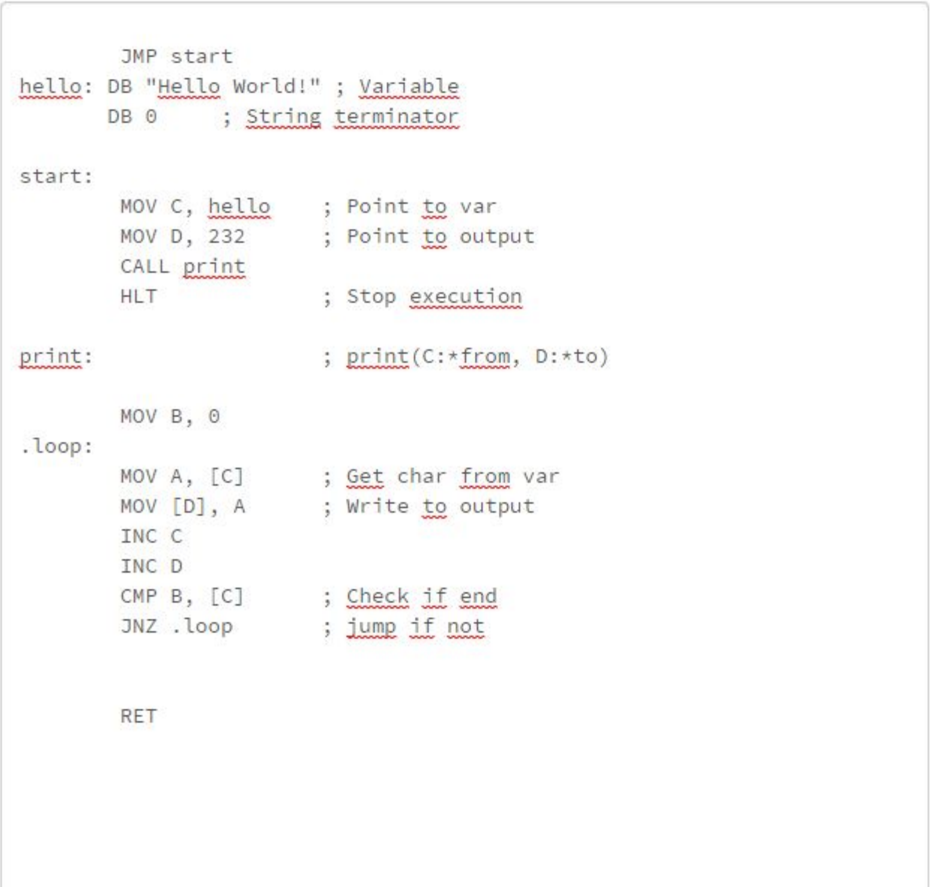
Registers / Flags

A	B	C	D	IP	SP	Z	C	F
72	0	3	233	41	228	FALSE	FALSE	FALSE

RAM

31	15	72	101	108	108	111	32	87	111	114	108	100	33	0	6
2	2	6	3	232	56	24	0	50	0	50	1	6	1	0	3
0	2	5	3	0	18	2	18	3	21	1	2	39	31	54	1
54	0	57	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	23	72	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Clock speed: 4 HZ



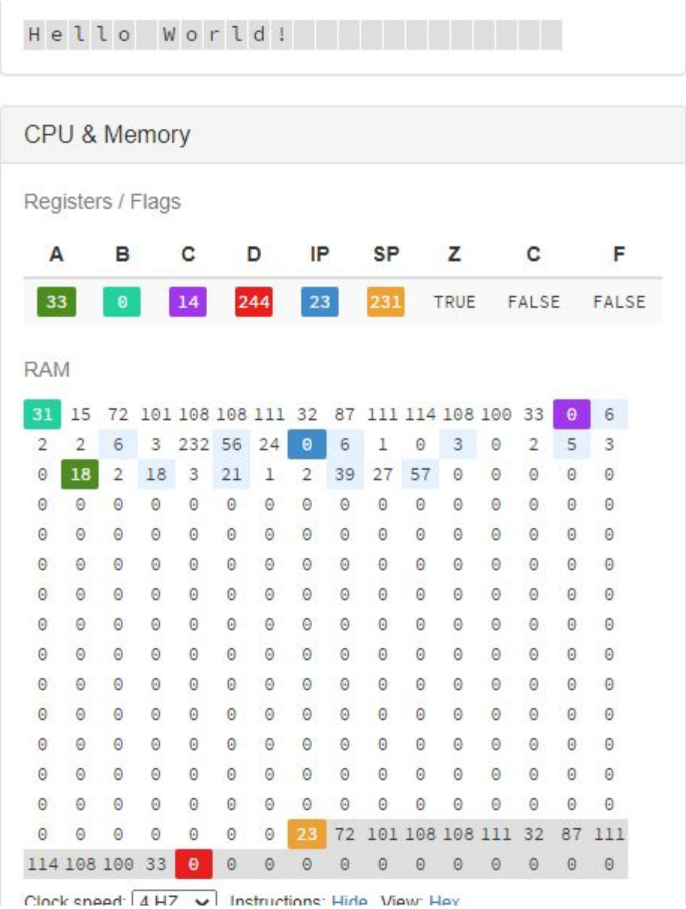
```
        JMP start
hello: DB "Hello World!" ; Variable
       DB 0      ; String terminator

start:
        MOV C, hello    ; Point to var
        MOV D, 232      ; Point to output
        CALL print
        HLT             ; Stop execution

print:                  ; print(C:*from, D:*to)

        MOV B, 0
.loop:
        MOV A, [C]      ; Get char from var
        MOV [D], A      ; Write to output
        INC C
        INC D
        CMP B, [C]      ; Check if end
        JNZ .loop       ; jump if not


        RET
```



```
Hello World!
```

CPU & Memory

Registers / Flags

A	B	C	D	IP	SP	Z	C	F
33	0	14	244	23	231	TRUE	FALSE	FALSE

RAM

```
 31  15  72 101 108 108 111  32  87 111 114 108 100  33   0   6
  2   2   6   3 232  56  24   0   6   1   0   3   0   2   5   3
  0  18   2  18   3  21   1   2  39  27  57   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0  23  72 101 108 108 111  32  87 111
114 108 100  33   0   0   0   0   0   0   0   0   0   0   0   0
```

Simulador

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

myCompiler | Português | Recentes | Login | Inscreva-se

teste

Assembly

Código de execução | Guardar código

```
section .data
    msg db "Hello world!", 0ah

section .text
    global _start

_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, msg
    mov rdx, 13
    syscall
    mov rax, 60
    mov rdi, 0
    syscall
```

Entrada do programa

Saída do programa

(Execute o programa para exibir sua saída)

Destrinchando o “Hello, World!”

Nasm

Intel x64

```
section .data
```

Section .data
Manipulação dos dados

```
section .text
    global _start

_start:
```

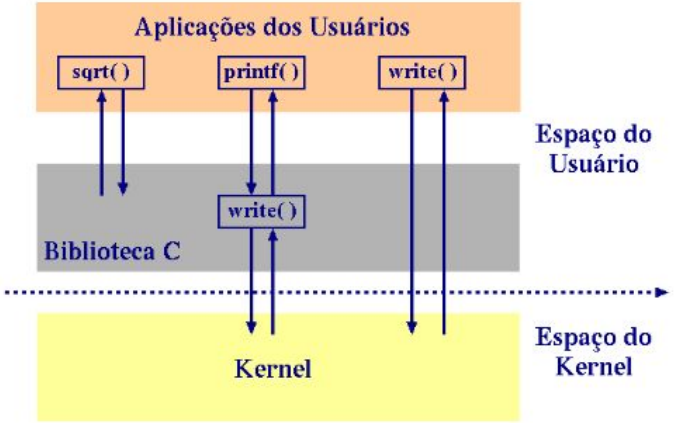
Section .text
área dos códigos

arch	syscall NR	return	arg0	arg1	arg2	arg3	arg4	arg5
arm	r7	r0	r0	r1	r2	r3	r4	r5
arm64	x8	x0	x0	x1	x2	x3	x4	x5
x86	eax	eax	ebx	ecx	edx	esi	edi	ebp
x86_64	rax	rax	rdi	rsi	rdx	r10	r8	r9

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

System Call

As chamadas de sistemas são funções (interfaces) usadas pelos aplicativos para solicitar a execução de algum serviço ao **kernel** do sistema operacional.



"Na linguagem de alto nível, o programador normalmente não utiliza as chamadas de sistema no seu código. "

Função de biblioteca → **Uma ou mais chamadas de sistema**

"Por exemplo, a função *printf()* da Linguagem C é mapeada na chamada *write()* para escrever em um arquivo."

Fonte: https://guialinux.uniriotec.br/chamadas-de-sistema/

```
section .data

section .text
    global _start

_start:
```

Destrinchando o “Hello, World!”

```
section .data
    mensagem db 'Hello, World!'

section .text
    global _start

_start:
```

```
section .data
    mensagem db 'Hello, World!'
section .text
    global _start
_start:
    mov rax, 1
```

%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
0	sys_read	unsigned int fd	char *buf	size_t count			
1	sys_write	unsigned int fd	const char *buf	size_t count			
2	sys_open	const char *filename	int flags	int mode			

```
section .data
    mensagem db 'Hello, World!'
section .text
    global _start
_start:
    mov rax, 1
```

%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
0	sys_read	unsigned int fd	char *buf	size_t count			
1	sys_write	unsigned int fd	const char *buf	size_t count			
2	sys_open	const char *filename	int flags	int mode			

```
section .data
    mensagem db 'Hello, World!'
section .text
    global _start
_start:
    mov rax, 1
```

File descriptor

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

%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
0	sys_read	unsigned int fd	char *buf	size_t count			
1	sys_write	unsigned int fd	const char *buf	size_t count			
2	sys_open	const char *filename	int flags	int mode			

```
section .data
    mensagem db 'Hello, World!'
section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
```

```
section .data
    mensagem db 'Hello, World!'
section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
```

Ponteiro para operando

Tamanho em bytes

%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
0	sys_read	unsigned int fd	char *buf	size_t count			
1	sys_write	unsigned int fd	const char *buf	size_t count			
2	sys_open	const char *filename	int flags	int mode			

```
section .data
    mensagem db 'Hello, World!'
section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, mensagem
    mov rdx, 13
```

```
section .data
    mensagem db 'Hello, World!'
    lenMensagem equ $-mensagem
section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, mensagem
    mov rdx, lenMensagem
```

```
section .data
    mensagem db 'Hello, World!'
section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, mensagem
    mov rdx, 13
    syscall
```

Terminou aqui???

```
section .data
    mensagem db 'Hello, World!'
section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, mensagem
    mov rdx, 13
    syscall
```

0 deu certo
1 erro

%rax	System call	%rdi	%rsi	%rdx
60	sys_exit	int error_code		

Precisamos encerrar o programa

```
section .data
    mensagem db 'Hello, World!'
section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, mensagem
    mov rdx, 13
    syscall

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