Assembly RISC-V

Como Programar Nisso?

Sumário

- ✔ Stack Pointer;
- ✔ Ferramentas; e
- ✔ Defines.

Stack Pointer

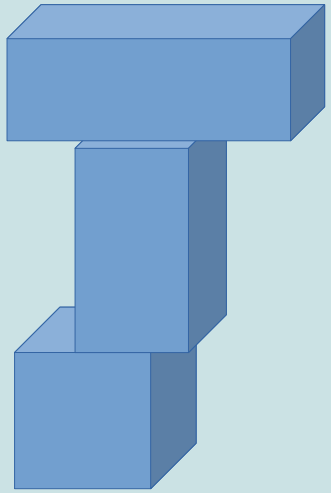
Estruturas de Dados 101

Pilha (*Stack*)

3. Além disso, só podemos acessar o topo.

2. Segue uma ordem de inserção LIFO:
   Last In First Out;

1. Uma estrutura linear similar ao vetor;



Stack Pointer

Armazenamento temporário expansível

→ Apesar do nome, é mais flexível :

- ✔ *Stack* → "Avisamos" que precisamos de espaço; mas
- ✔ *Array* → Permite o acesso por índice sem restrições*

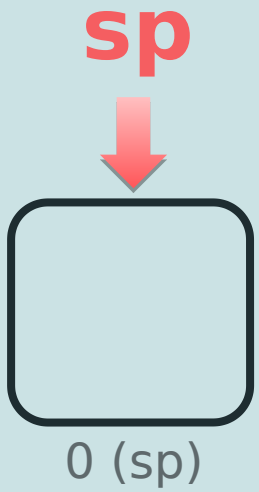
sp

Stack Pointer

Armazenamento temporário expansível

→ Apesar do nome, é mais flexível :

- ✔ *Stack* → "Avisamos" que precisamos de espaço; mas
- ✔ *Array* → Permite o acesso por índice sem restrições*



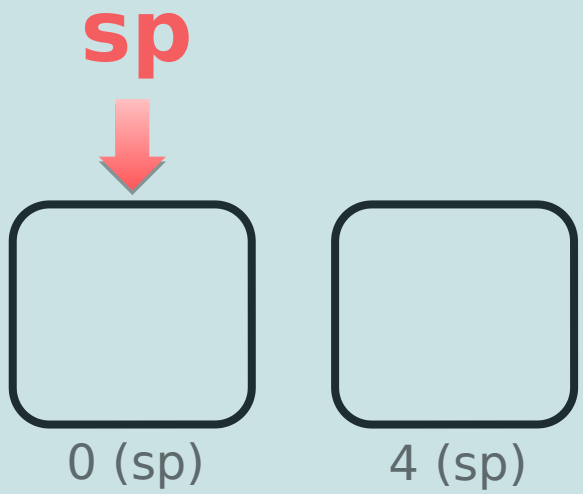
addi sp, sp, -4

Stack Pointer

Armazenamento temporário expansível

→ Apesar do nome, é mais flexível :

- ✔ *Stack* → "Avisamos" que precisamos de espaço; mas
- ✔ *Array* → Permite o acesso por índice sem restrições*



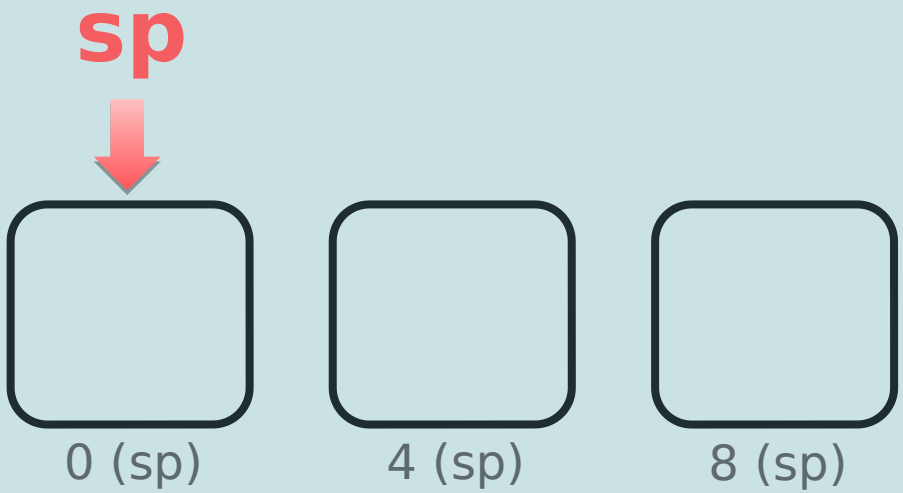
addi sp, sp, -8

Stack Pointer

Armazenamento temporário expansível

→ Apesar do nome, é mais flexível :

- ✔ *Stack* → "Avisamos" que precisamos de espaço; mas
- ✔ *Array* → Permite o acesso por índice sem restrições*



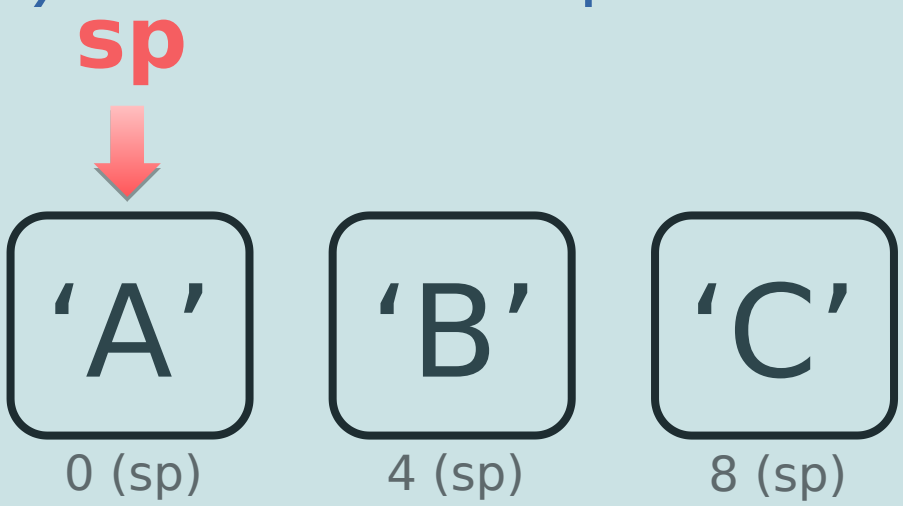
addi sp, sp, -12

Stack Pointer

Armazenamento temporário expansível

→ Apesar do nome, é mais flexível :

- ✔ *Stack* → "Avisamos" que precisamos de espaço; mas
- ✔ *Array* → Permite o acesso por índice sem restrições*

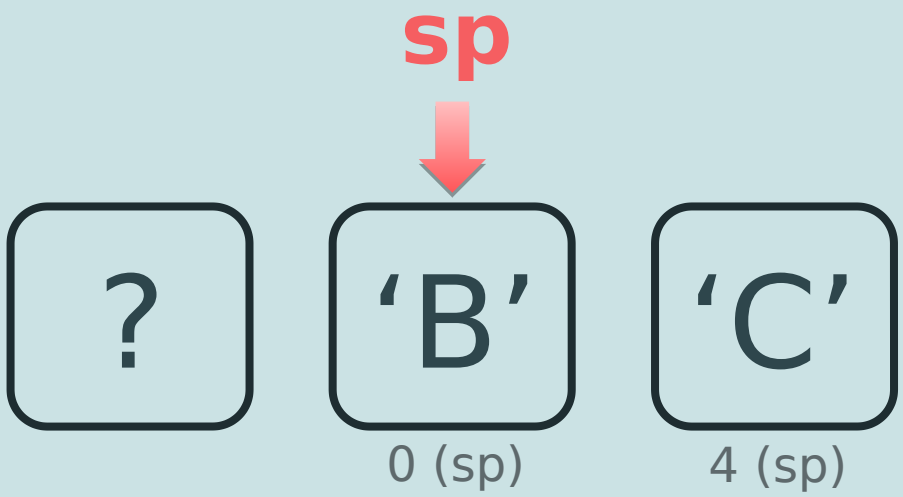


Stack Pointer

Armazenamento temporário expansível

→ Apesar do nome, é mais flexível :

- ✔ *Stack* → "Avisamos" que precisamos de espaço; mas
- ✔ *Array* → Permite o acesso por índice sem restrições*



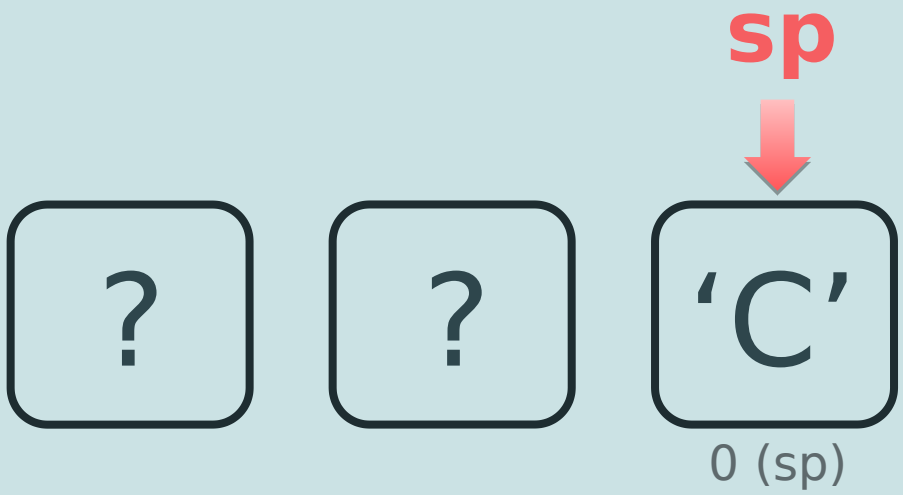
addi sp, sp, 4

Stack Pointer

Armazenamento temporário expansível

→ Apesar do nome, é mais flexível :

- ✔ *Stack* → "Avisamos" que precisamos de espaço; mas
- ✔ *Array* → Permite o acesso por índice sem restrições*




```
addi sp, sp, 4
addi sp, sp, 4
```

Stack Pointer

Armazenamento temporário expansível

→ Apesar do nome, é mais flexível :

- ✔ *Stack* → "Avisamos" que precisamos de espaço; mas
- ✔ *Array* → Permite o acesso por índice sem restrições*


```
addi sp, sp, 4
addi sp, sp, 4
addi sp, sp, 4
```

Stack Pointer

Armazenamento temporário expansível

Exemplo:
Se precisarmos usar um registrador salvo dentro de uma função

Stack Pointer

Armazenamento temporário expansível

```
main:
        li s0, 1                # s0 = 1
        call move               # chamada de função

        li a7, 10               # Encerra Programa
        ecall

move: addi sp, sp, -4   #     aloca 1 word(s) na pilha
        sw s0, 0(sp)            #     salva s0
        li s0, 10               #     podemos usar o s0
        [...]                   #           sem risco de perder/sobrescrever
        lw s0,  0(sp)           #     recupera s0
        addi sp, sp, 4          #     libera 1 word(s) da pilha
return: ret                     # return da função
```

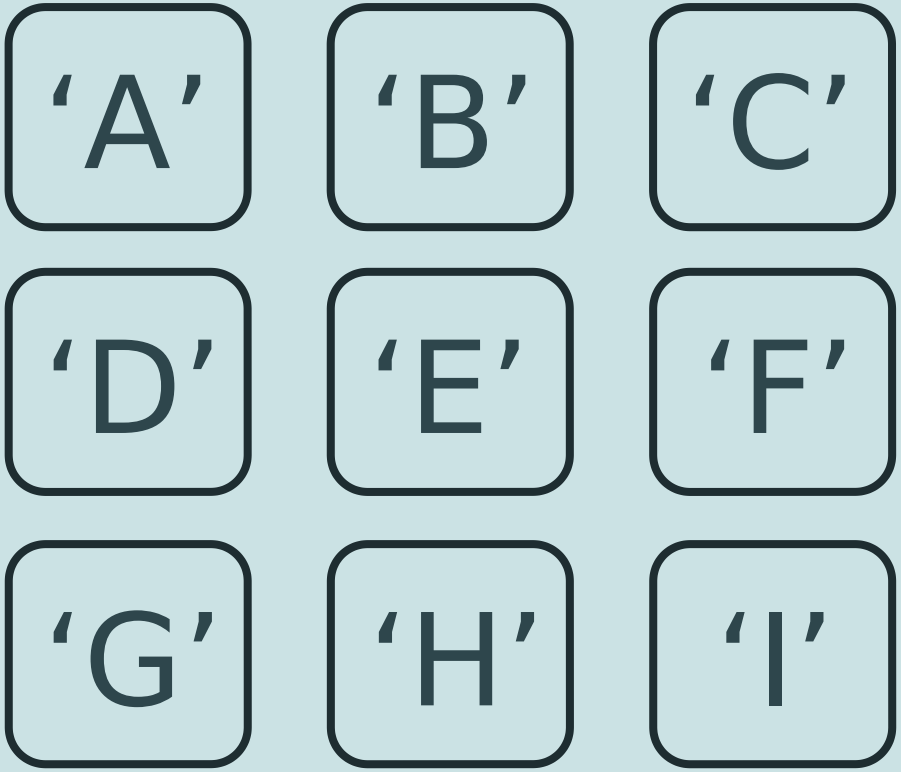
Exemplo:
Se precisarmos usar um registrador salvo dentro de uma função

Ferramentas

Bitmap Display

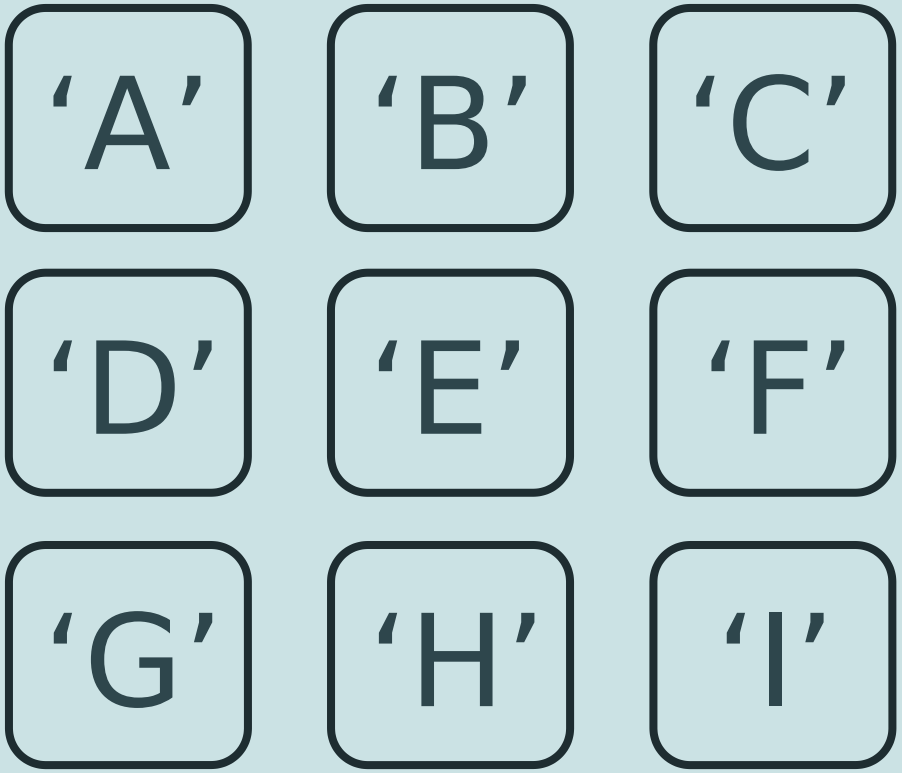
ABC
DEF
GHI

Bitmap Display



Bitmap Display

3x3



Endereço	Conteúdo
0x10010000	‘A’
0x10010004	‘B’
0x10010008	‘C’
0x1001000C	‘D’
0x10010010	‘E’
0x10010014	‘F’
...	...

Bitmap Display

- ✔ Conversor online multiformato para bmp 24 cores/pixel; e
- ✔ Conversor multiuso (possui vários módulos encadeáveis).

Magenta invisível (c7)
#FD01FE

Cores de 8 bits
11000111
redGreBl

Keyboard MMIO

Endereço	Conteúdo
0xFFFF0000	Clicou?
0xFFFF0004	‘A’

Dica : **Clicou? é desativado com lw**

Defines

Defines

Aqueles que não pertence aos outros grupos

```
.include "cool.asm"             # Caminho até o arquivo

.macro SUM(%a, %b)              # SUM recebe 2 registradores
      add %a, %a, %b            # retorna o resultado da soma
.end_macro                      # no primeiro

.eqv GRAVIDADE 10               # GRAVIDADE = 10

.text
      addi t0, zero, GRAVIDADE # t0 = 0 + GRAVIDADE = 0 + 10
      SUM(t1,t0)                        # t1 = t1 + t0
```

Defines

Aqueles que não pertence aos outros grupos

→ .include “cool.asm”

Incluir arquivos: em linguagem C, seria `#include "cool.asm"`

Representam a inclusão de um código contido em outro arquivo.

Na prática, podemos imaginar como um grande *Copy and Paste* de código: no lugar desse comando, será colocado o conteúdo do arquivo.

Mais configurações: .global .extern

Defines

Aqueles que não pertence aos outros grupos

→ .macro && .end_macro

Definem macros: em linguagem C, seria `#define SUM(a,b) (a + b)`

Representa um "molde" de código, ou seja, um trecho de código que pode ser modelado de acordo com o uso.

Na prática, podemos imaginar como *Copy and Paste* de código: no lugar desse comando, será colocado o "molde" com os argumentos.

Macros vs Funções

Macros

Prós	Contras
- Executadas mais rápidas (*inline*) - Idealmente, são simples e objetivas	- Códigos mais extensos

Funções

Prós	Contras
- Interrupções - Permitem recursão - “Reusáveis”	- Mais lentas, se comparadas à macros

Defines

Aqueles que não pertence aos outros grupos

→ .eqv GRAVIDADE 10

Equivalência: em linguagem C, seria `#define GRAVIDADE 10`

Representam um valor literal que será substituído durante a compilação.

Pode ser visto como uma macro sem argumentos.

Memória Principal

Endereço	Conteúdo	Label
0x00400000	add a0,t0,t1	.text
0x00400004	ecall	
...	...	
0x0FFFFFFC	ecall	END .text
...	...	
0x10010000	“Hey!”	.data
...	...	
0x1003FFFC	“Bye!”	END .data

The End

Thales Menezes