



UNIVERSIDADE  
FEDERAL DE  
SERGIPE



DEPARTAMENTO  
DE COMPUTAÇÃO

# Controle de fluxo

## Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

# Introdução

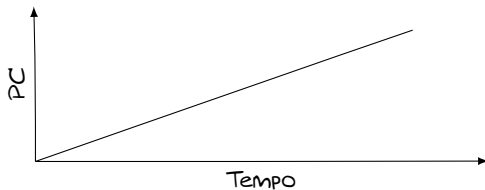
- ▶ O que é controle de fluxo?
  - ▶ Define a sequência de instruções que são executadas dinamicamente pelo processador
  - ▶ Quando não ocorrem desvios, as instruções são executadas sequencialmente incrementando o PC

# Introdução

- ▶ O que é controle de fluxo?
  - ▶ Define a sequência de instruções que são executadas dinamicamente pelo processador
  - ▶ Quando não ocorrem desvios, as instruções são executadas sequencialmente incrementando o PC
- ▶ Quais são os seus tipos?
  - ▶ Condicional
  - ▶ Iterativo
  - ▶ Sub-rotina

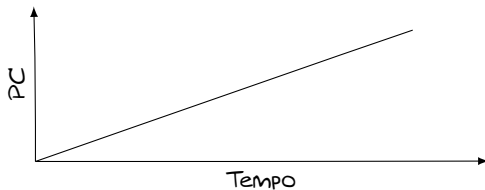
# Introdução

- ▶ Comportamento do fluxo de execução
  - ▶ Sequencial (sem desvios)

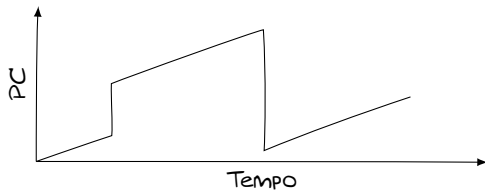


# Introdução

- ▶ Comportamento do fluxo de execução
  - ▶ Sequencial (sem desvios)



- ▶ Não sequencial (com desvios)



# Introdução

- ▶ Nos primórdios da computação, todo o controle de fluxo era de responsabilidade do desenvolvedor
  - ▶ Linguagem de máquina e de montagem
  - ▶ Controle de fluxo por desvios (**goto**)
  - ▶ Baixa abstração e mais erros humanos

# Introdução

- ▶ Nos primórdios da computação, todo o controle de fluxo era de responsabilidade do desenvolvedor
  - ▶ Linguagem de máquina e de montagem
  - ▶ Controle de fluxo por desvios (**goto**)
  - ▶ Baixa abstração e mais erros humanos
- ▶ Com o nascimento da programação estruturada em C, o foco é descrever o comportamento do sistema, abstraindo detalhes de funcionamento do *hardware*
  - ▶ Condicional: **if, else, else if, switch**
  - ▶ Iterativo: **do while, for, while**
  - ▶ Sub-rotina: procedimentos e funções

# Controle condicional

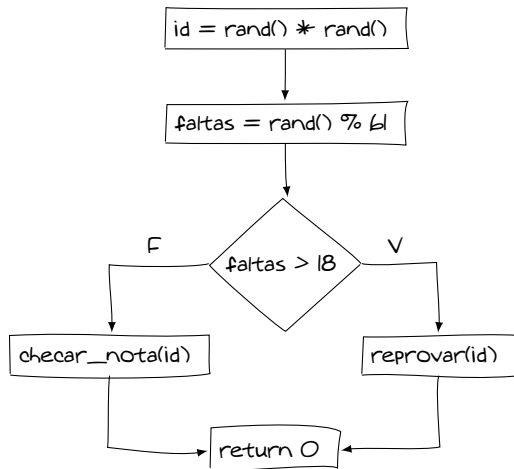
## ► Sentença **if**, **else**

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Biblioteca padrão
4 #include <stdlib.h>
...
10 // Função principal
11 int main() {
12     // ID do aluno
13     uint32_t id = rand() * rand();
14     // Quantidade de faltas acumuladas
15     uint8_t faltas = rand() % 61;
16     // Se faltas > 18 horas, então reprovar
17     if(faltas > 18) reprovar(id);
18     // Se não, checar nota
19     else checar_nota(id);
20     // Retorno sem erros
21     return 0;
22 }
```



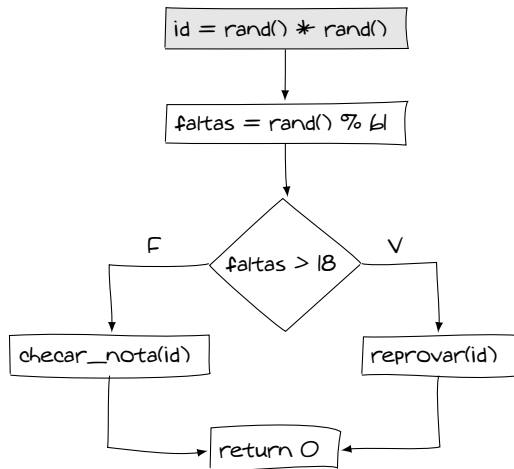
# Controle condicional

- ▶ Sentença **if, else**
  - ▶ Fluxo de execução



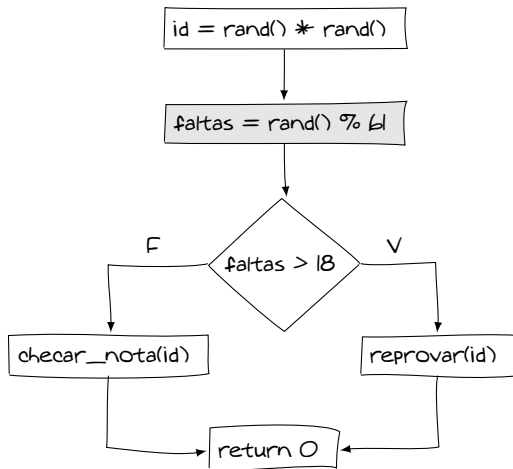
# Controle condicional

- ▶ Sentença **if, else**
  - ▶ Fluxo de execução



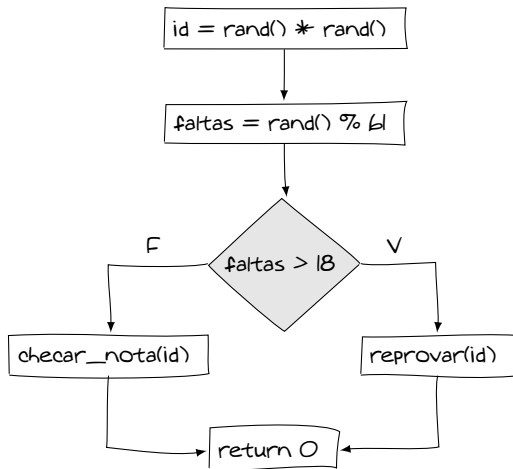
# Controle condicional

- ▶ Sentença **if, else**
  - ▶ Fluxo de execução



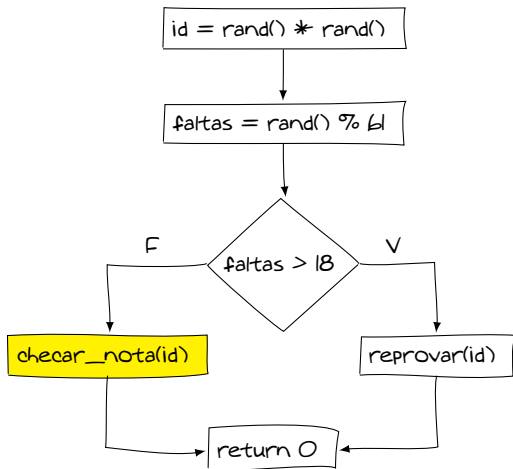
# Controle condicional

- ▶ Sentença **if, else**
  - ▶ Fluxo de execução



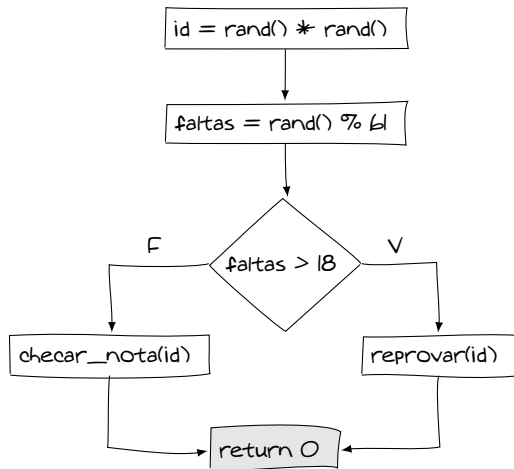
# Controle condicional

- ▶ Sentença **if, else**
  - ▶ Fluxo de execução



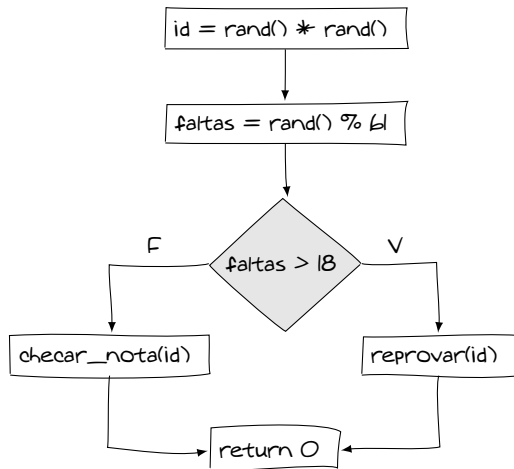
# Controle condicional

- ▶ Sentença **if, else**
  - ▶ Fluxo de execução



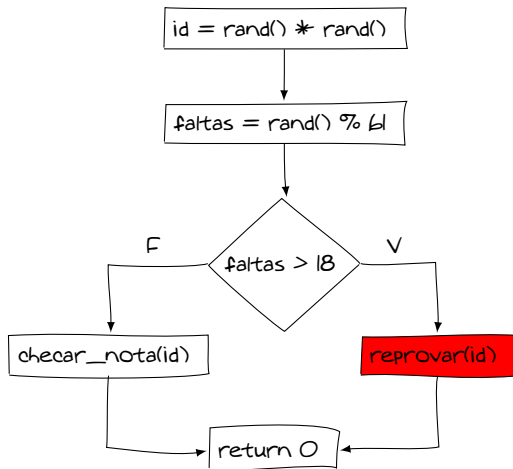
# Controle condicional

- ▶ Sentença **if, else**
  - ▶ Fluxo de execução



# Controle condicional

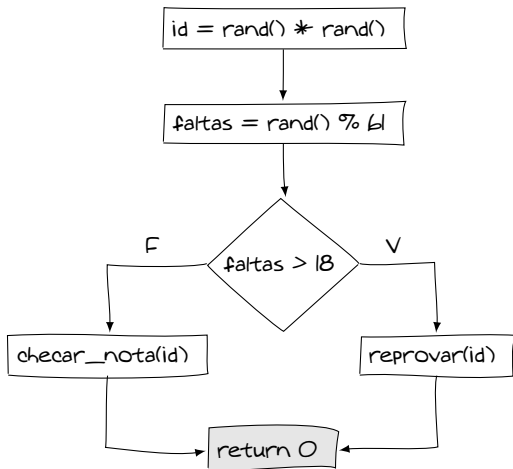
- ▶ Sentença **if, else**
  - ▶ Fluxo de execução





# Controle condicional

- ▶ Sentença **if, else**
  - ▶ Fluxo de execução



# Controle condicional

## ► Sentença **if, else**

```
1 // Função principal
2 main:
3     // r1 = id, r2 = faltas
4     132 r1, [0x40]
5     18 r2, [0x107]
6     // faltas ? 18
7     cmpi r2, 18
8     bgt V
9     // faltas <= 18
10    F:  bun checar_nota
11        bun 1
12        // faltas > 18
13    V:  bun reprovar
14    // Fim
15    int 0
```

# Controle condicional

## ► Sentença **if, else**

0x00000000

⋮

0x00000020

0x00000024

0x00000028

0x0000002C

0x00000030

0x00000034

0x00000038

0x0000003C

⋮

0x000000100

0x000000104

⋮

Bun 1
⋮
132 r1, [0x40]
18 r2, [0x107]
cmpi r2, 18
Bgt 2
Bun checar_nota
Bun 1
Bun reprovar
int 0
⋮
?
?
⋮

id  
faltas

# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
⋮	⋮	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
⋮	⋮	
0x000000100	?	id faltas
0x000000104	?	
⋮	⋮	

# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
⋮	⋮	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
⋮	⋮	
0x000000100	?	id
0x000000104	?	faltas
⋮	⋮	

# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
:	:	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
:	:	
0x000000100	?	id faltas
0x000000104	?	
:	:	

# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
:	:	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
:	:	
0x000000100	?	id
0x000000104	?	faltas
:	:	

# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
:	:	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
:	:	
0x000000100	?	id
0x000000104	?	faltas
:	:	



# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
:	:	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
:	:	
0x000000100	?	id
0x000000104	?	faltas
:	:	

# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
:	:	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
:	:	
0x000000100	?	id
0x000000104	?	faltas
:	:	

# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
⋮	⋮	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
⋮	⋮	
0x000000100	?	id faltas
0x000000104	?	
⋮	⋮	

# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
:	:	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
:	:	
0x000000100	?	id
0x000000104	?	faltas
:	:	

# Controle condicional

## ► Sentença **if, else**

0x00000000	Bun 7	
⋮	⋮	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	Bgt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun 1	
0x00000038	Bun reprovar	
0x0000003C	int 0	
⋮	⋮	
0x000000100	?	id
0x000000104	?	faltas
⋮	⋮	

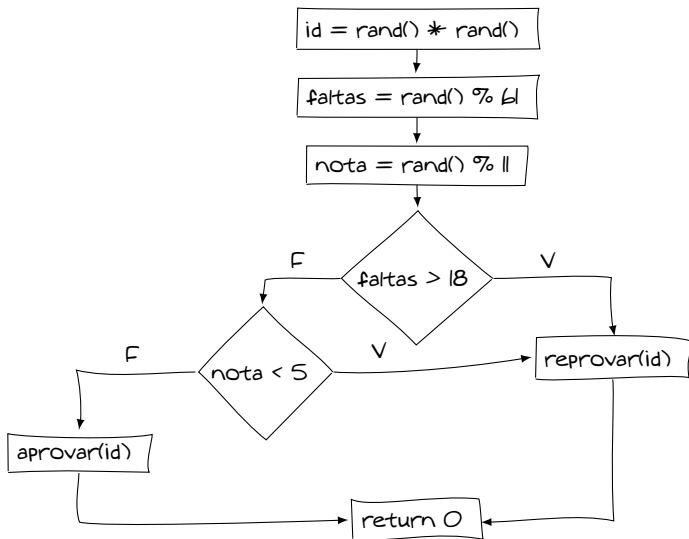
# Controle condicional

## ► Sentença **if**, **else if**, **else**

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
...
10 // Função principal
11 int main() {
12     // ID do aluno
13     uint32_t id = rand() * rand();
14     // Quantidade de faltas acumuladas e nota final
15     uint8_t faltas = rand() % 61, nota = rand() % 11;
16     // Se faltas > 18 horas, então reprovar
17     if(faltas > 18) reprovar(id);
18     // Se não, se nota < 5, então reprovar
19     else if(nota < 5) reprovar(id);
20     // Se não, aprovar
21     else aprovar(id);
22     // Retorno sem erros
23     return 0;
24 }
```

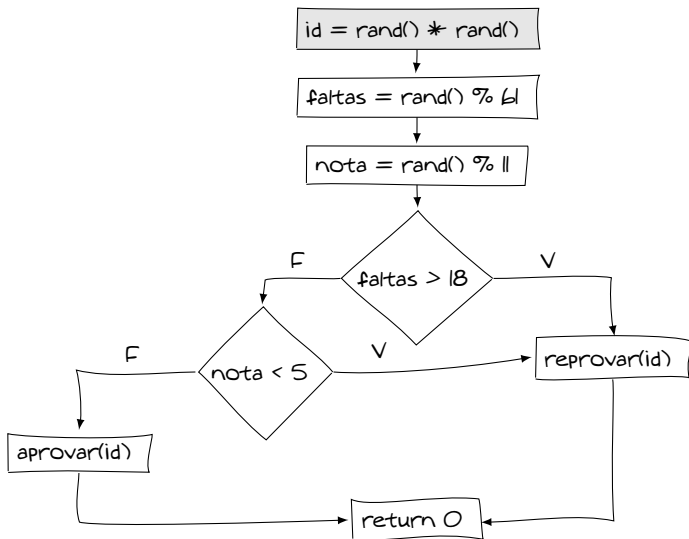
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



# Controle condicional

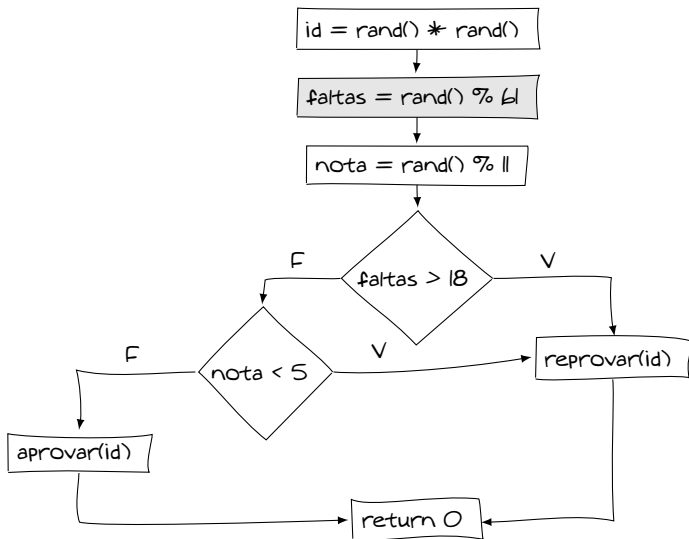
- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução





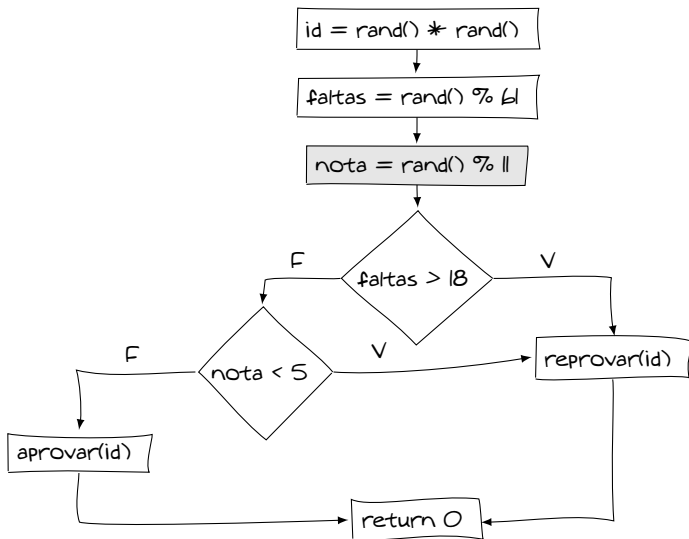
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



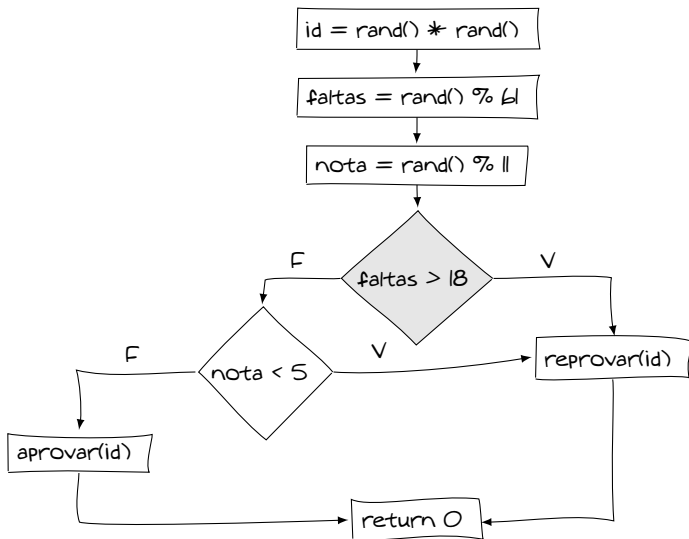
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



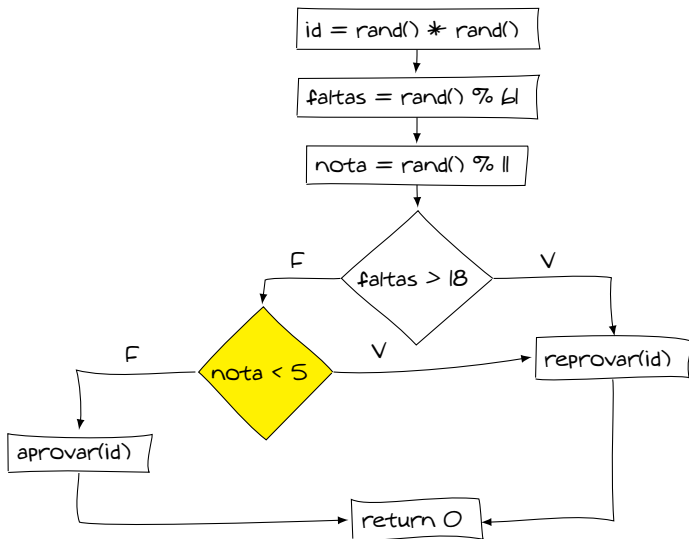
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



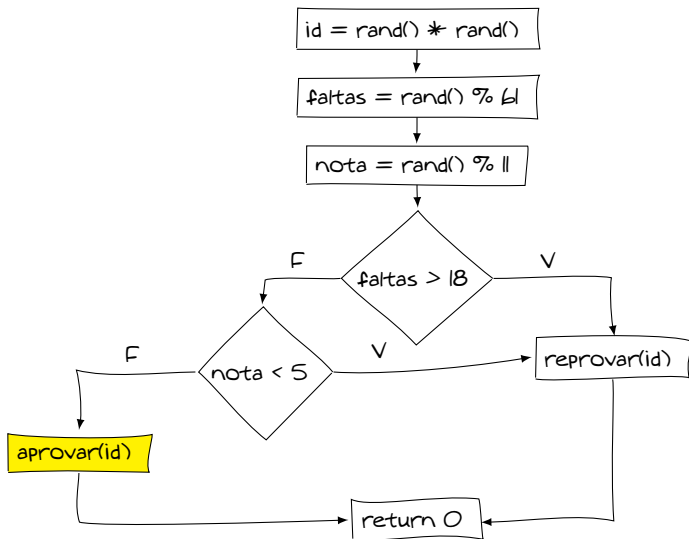
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



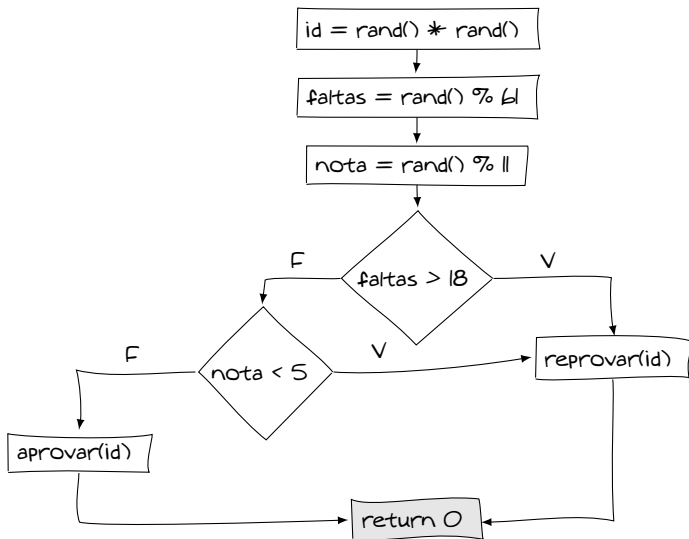
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



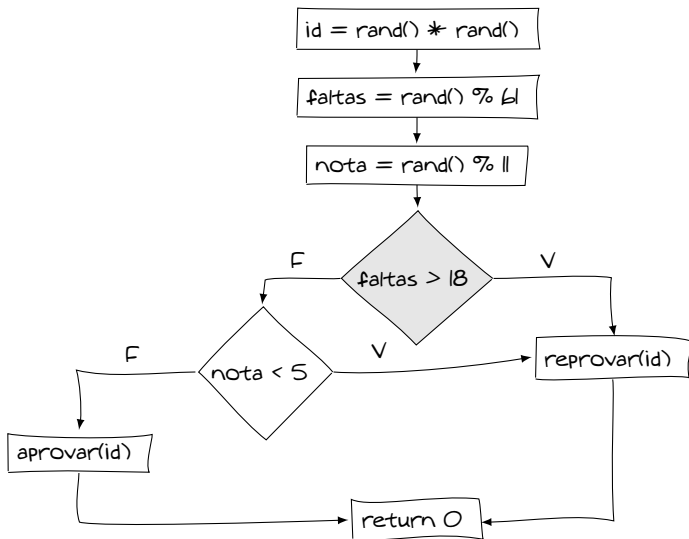
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



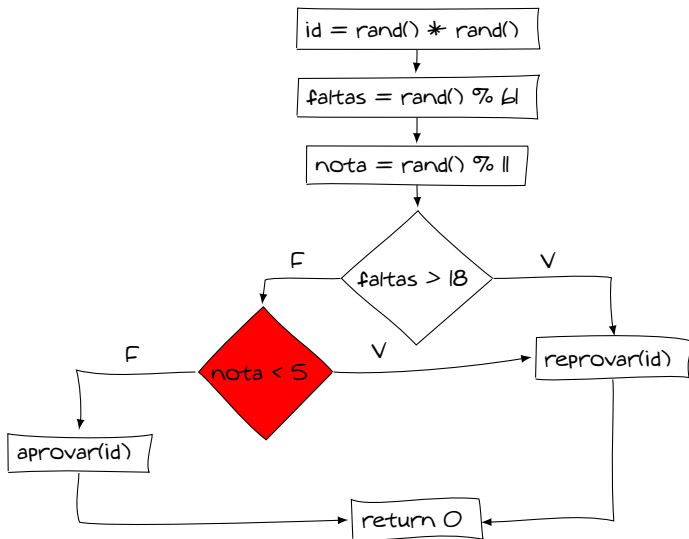
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



# Controle condicional

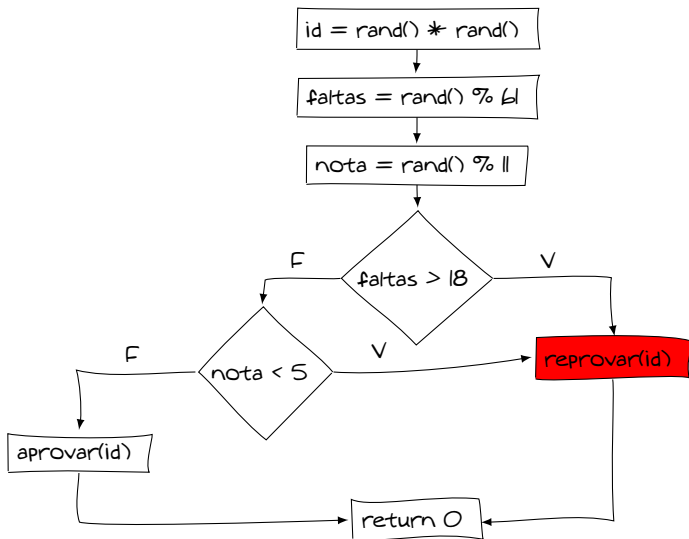
- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução





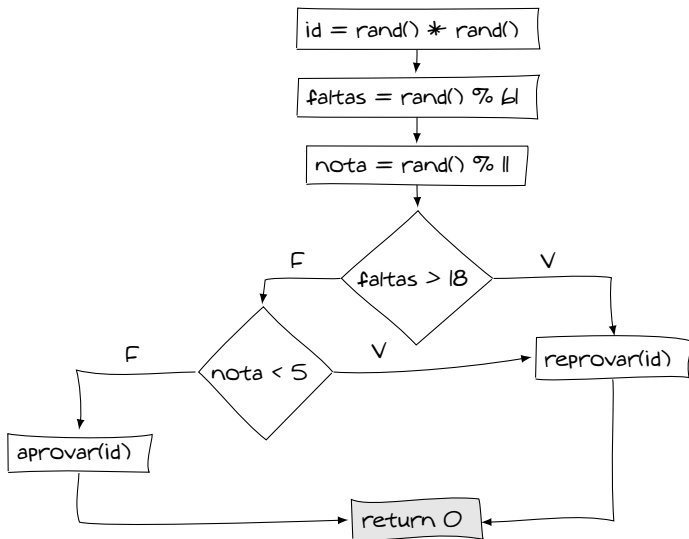
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



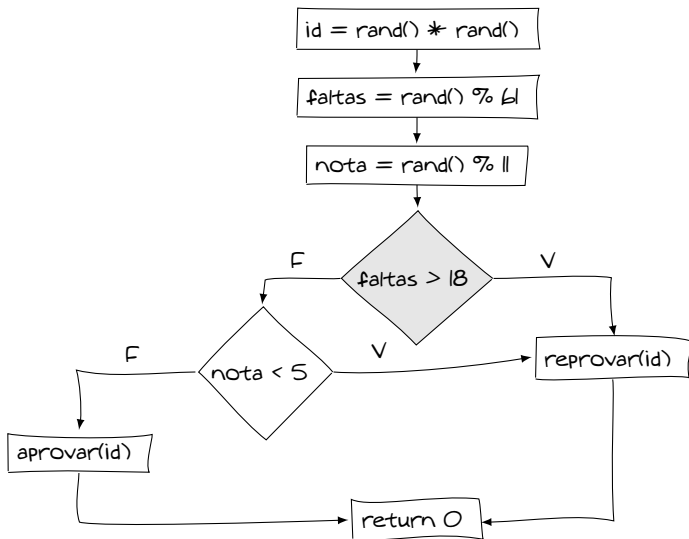
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



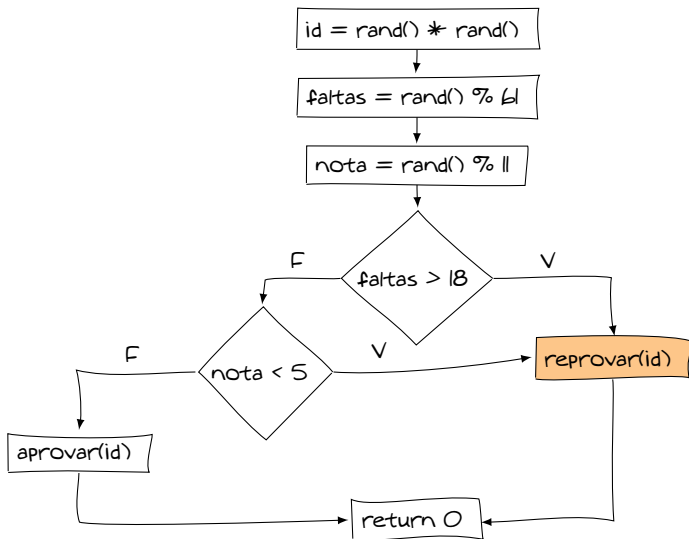
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



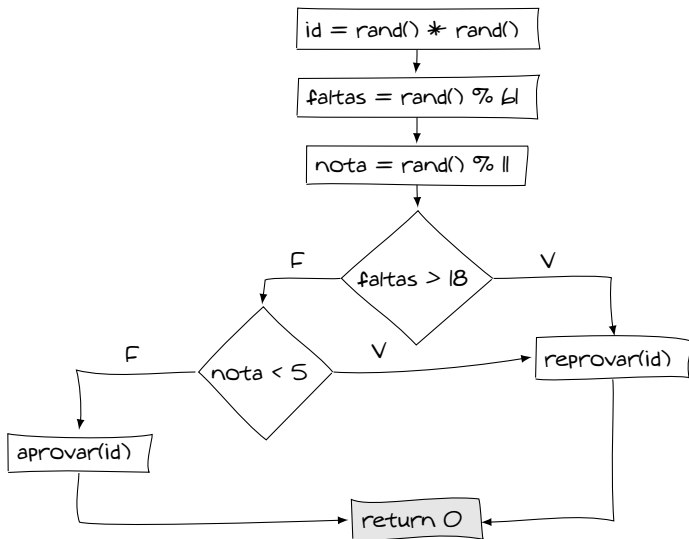
# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



# Controle condicional

- ▶ Sentença **if**, **else if**, **else**
- ▶ Fluxo de execução



# Controle condicional

## ► Sentença **if, else if, else**

```
1 // Função principal
2 main:
3     // r1 = id, r2 = faltas, r3 = nota
4     132 r1, [0x40]
5     18 r2, [0x107]
6     18 r3, [0x10B]
7     // faltas ? 18
8     cmpi r2, 18
9     bgt V
10    // nota ? 5
11    F: cmpi r3, 5
12        blt V
13        // faltas <= 18 and nota >= 5
14        bun aprovar
15        bun 1
16        // faltas > 18 or nota < 5
17    V: bun reprovar
18    // Fim
19    int 0
```

# Controle condicional

## ► Sentença **if**, **else if**, **else**

0x00000000	Bun 7	
⋮	⋮	
0x00000020	132 r1, [0x40]	
0x00000024	18 r2, [0x107]	
0x00000028	18 r3, [0x10B]	
⋮	⋮	
0x00000100	?	id
0x00000104	?	faltas
0x00000108	?	nota
⋮	⋮	

# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	132 r1, [0x40]
0x000000024	18 r2, [0x107]
0x000000028	18 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	BlT 2
0x00000003C	Bun aprovar
0x000000040	Bun 1
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮



# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	132 r1, [0x40]
0x000000024	18 r2, [0x107]
0x000000028	18 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	BlT 2
0x00000003C	Bun aprovar
0x000000040	Bun I
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮

# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	132 r1, [0x40]
0x000000024	18 r2, [0x107]
0x000000028	18 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	Blh 2
0x00000003C	Bun aprovar
0x000000040	Bun 1
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮

# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	132 r1, [0x40]
0x000000024	18 r2, [0x107]
0x000000028	18 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	BlT 2
0x00000003C	Bun aprovar
0x000000040	Bun 1
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮

# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	132 r1, [0x40]
0x000000024	18 r2, [0x107]
0x000000028	18 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	BlT 2
0x00000003C	Bun aprovar
0x000000040	Bun 1
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮

# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	132 r1, [0x40]
0x000000024	18 r2, [0x107]
0x000000028	18 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	Blh 2
0x00000003C	Bun aprovar
0x000000040	Bun 1
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮

# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	132 r1, [0x40]
0x000000024	18 r2, [0x107]
0x000000028	18 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	Blh 2
0x00000003C	Bun aprovar
0x000000040	Bun 1
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮

# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	l32 r1, [0x40]
0x000000024	l8 r2, [0x107]
0x000000028	l8 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	BlT 2
0x00000003C	Bun aprovar
0x000000040	Bun l
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮

# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	132 r1, [0x40]
0x000000024	18 r2, [0x107]
0x000000028	18 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	Blh 2
0x00000003C	Bun aprovar
0x000000040	Bun 1
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮



# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	132 r1, [0x40]
0x000000024	18 r2, [0x107]
0x000000028	18 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	Blh 2
0x00000003C	Bun aprovar
0x000000040	Bun 1
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮

# Controle condicional

## ► Sentença **if**, **else if**, **else**

⋮	⋮
0x000000020	l32 r1, [0x40]
0x000000024	l8 r2, [0x107]
0x000000028	l8 r3, [0x10B]
0x00000002C	cmpi r2, 18
0x000000030	Bgt 4
0x000000034	cmpi r3, 5
0x000000038	Blh 2
0x00000003C	Bun aprovar
0x000000040	Bun l
0x000000044	Bun reprovar
0x000000048	int 0
⋮	⋮

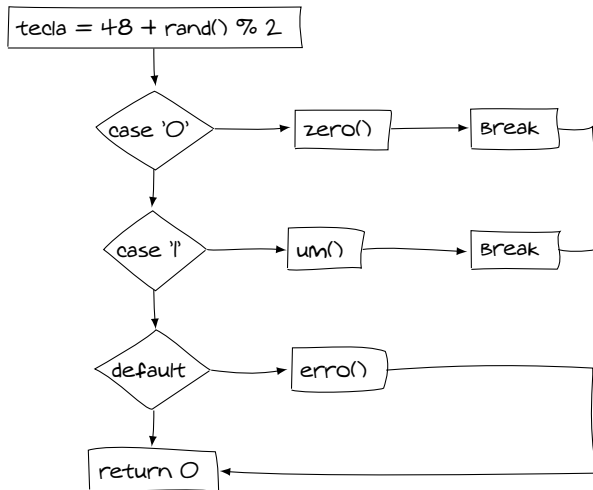
# Controle condicional

## ► Sentença **switch**

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
...
10 // Função principal
11 int main() {
12     // Tecla binária (0 ou 1)
13     uint8_t tecla = 48 + rand() % 2;
14     // Selecionando tecla
15     switch(tecla) {
16         // '0'
17         case '0': zero(); break;
18         // '1'
19         case '1': um(); break;
20         // Diferente de '0' ou '1'
21         default: erro();
22     }
23     // Retorno sem erros
24     return 0;
25 }
```

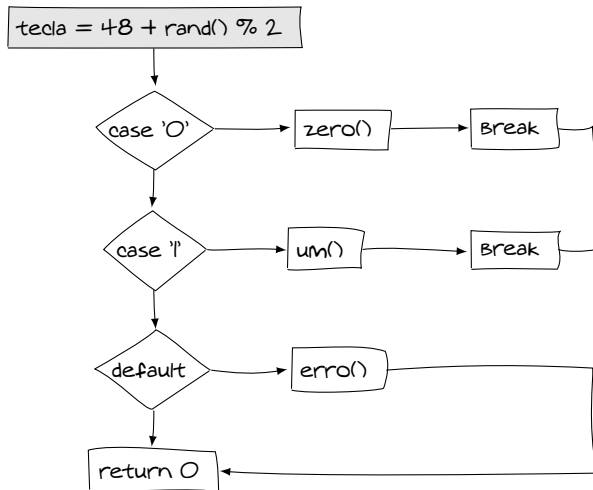
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



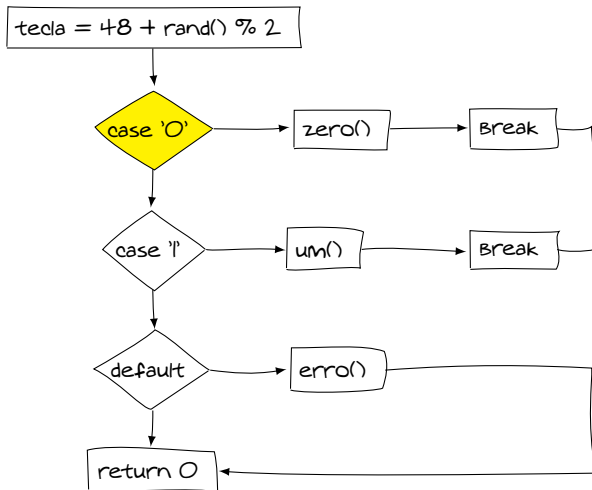
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



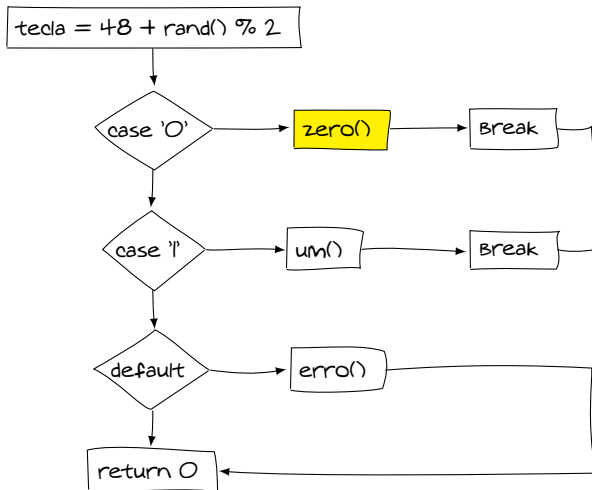
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



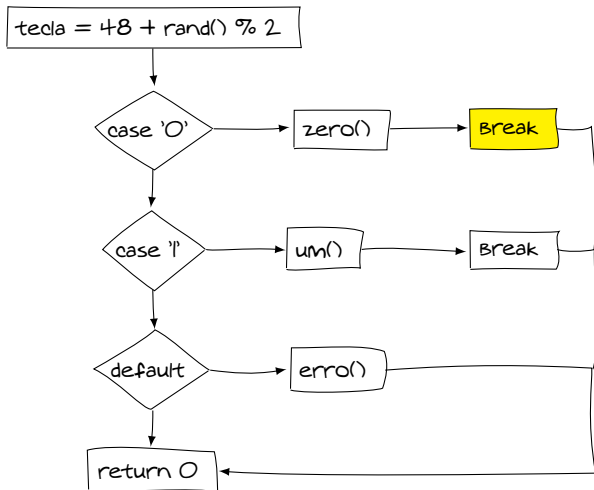
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



# Controle condicional

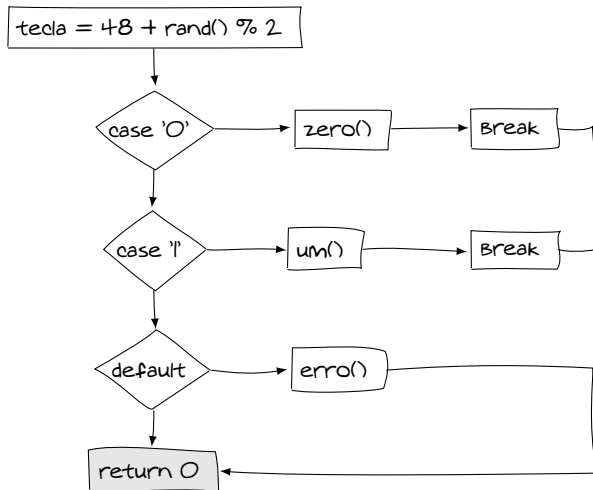
- ▶ Sentença **switch**
  - ▶ Fluxo de execução





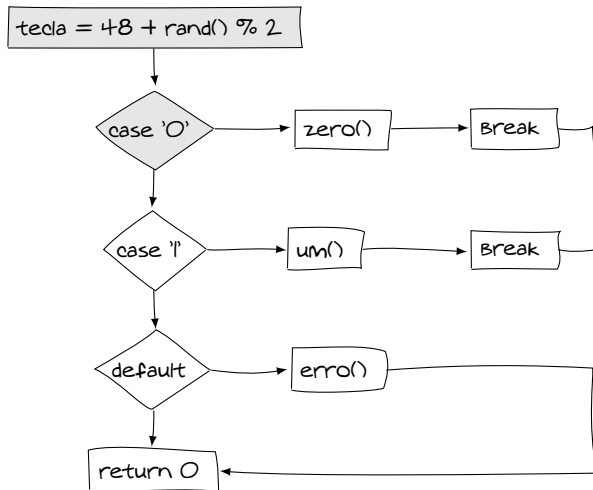
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



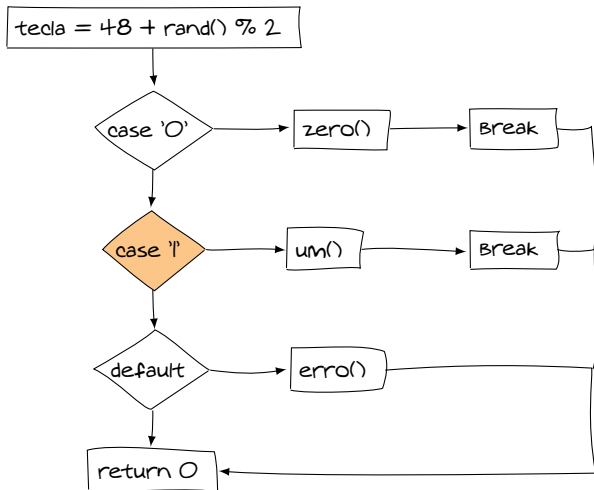
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



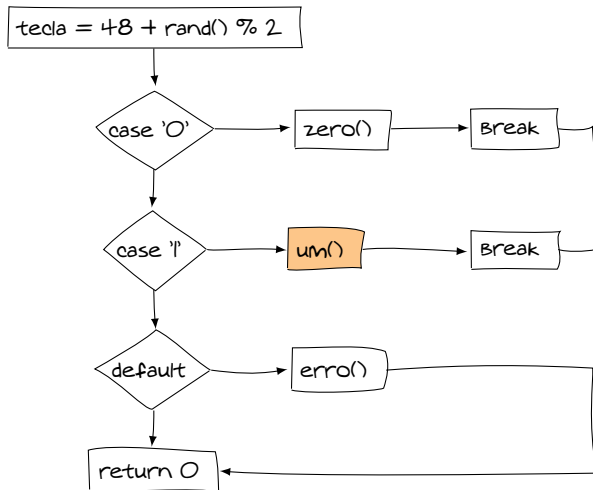
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



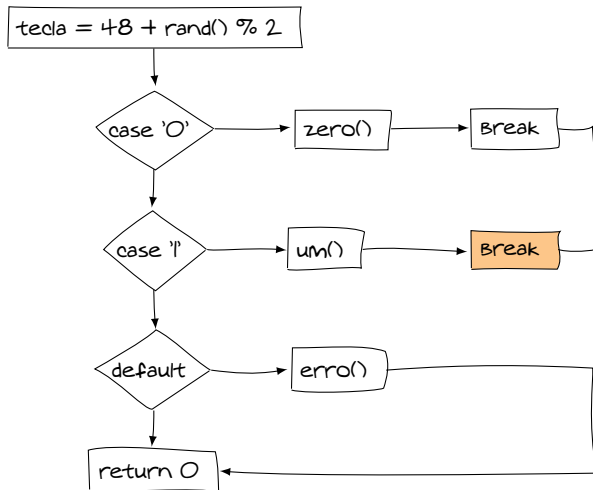
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



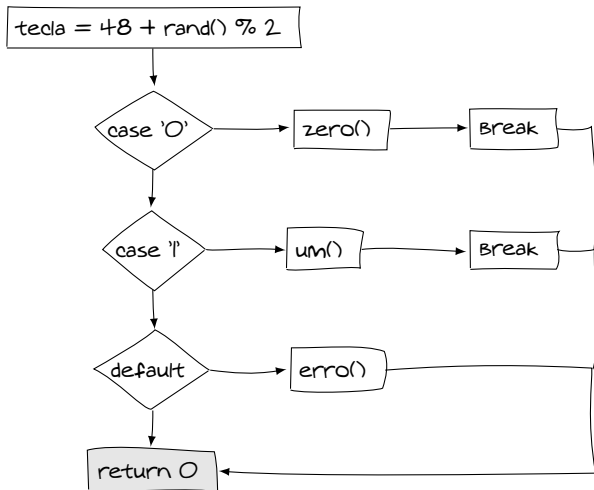
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



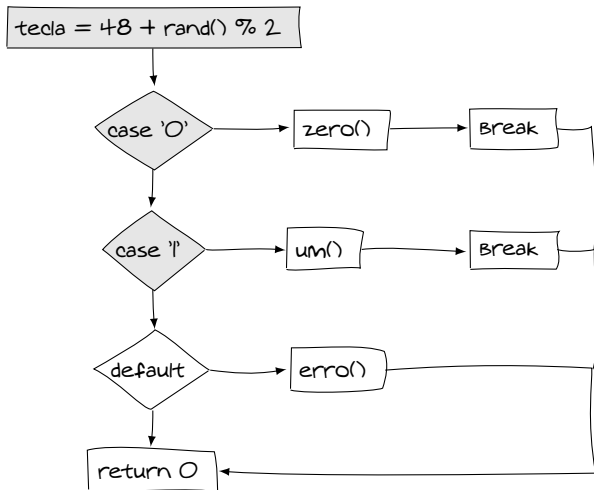
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



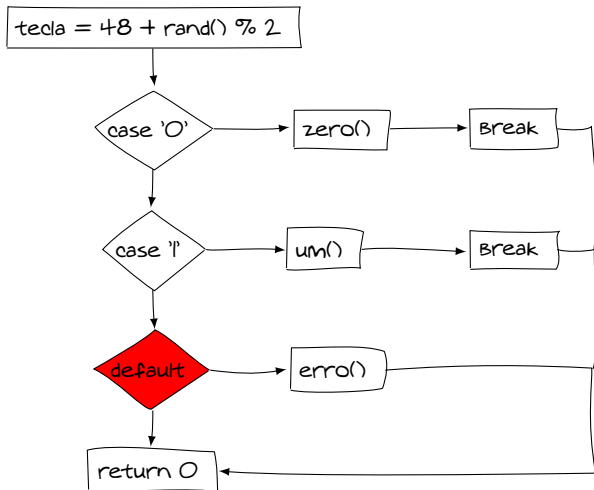
# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



# Controle condicional

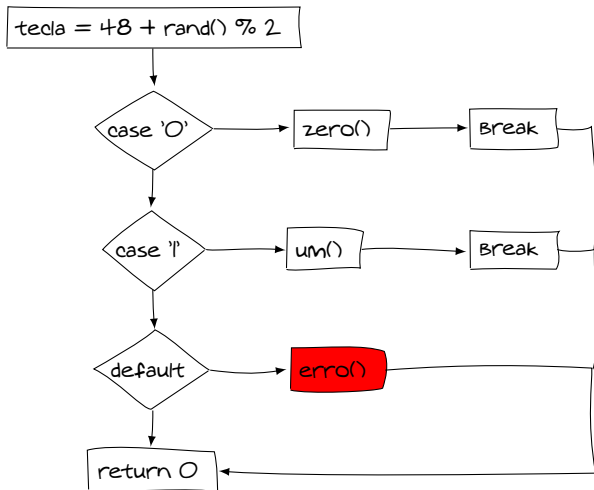
- ▶ Sentença **switch**
  - ▶ Fluxo de execução





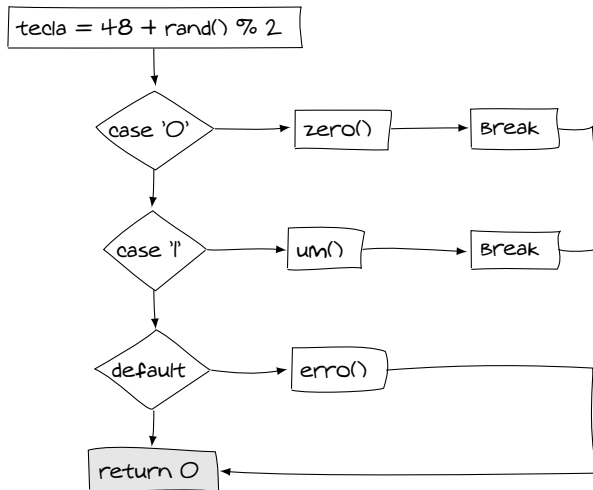
# Controle condicional

- ▶ Sentença **switch**
- ▶ Fluxo de execução



# Controle condicional

- ▶ Sentença **switch**
  - ▶ Fluxo de execução



# Controle condicional

## ► Sentença **switch**

```
1 // Função principal
2 main:
3     // r1 = tecla
4     l8 r1, [0x103]
5     // case '0'
6     cmpi r1, 48
7     bne 2
8     bun zero
9     bun 5
10    // case '1'
11    cmpi r1, 49
12    bne 2
13    bun um
14    bun 1
15    // default
16    bun erro
17    // Fim
18    int 0
```

# Controle condicional

## ► Sentença **switch**

⋮	⋮
0x000000020	18 r1, [0x103]
0x000000024	cmpi r1, 48
0x000000028	bne 2
0x00000002C	bun zero
0x000000030	bun 5
0x000000034	cmpi r1, 49
0x000000038	bne 2
0x00000003C	bun um
0x000000040	bun 1
0x000000044	bun erro
0x000000048	int 0
⋮	⋮
0x000000100	?

teda

# Controle condicional

## ► Sentença **switch**

⋮	⋮	
0x000000020	18 r1, [0x103]	
0x000000024	cmpi r1, 48	
0x000000028	bne 2	
0x00000002C	bun zero	
0x000000030	bun 5	
0x000000034	cmpi r1, 49	
0x000000038	bne 2	
0x00000003C	bun um	
0x000000040	bun 1	
0x000000044	bun erro	
0x000000048	int 0	
⋮	⋮	
0x000000100	?	teda

# Controle condicional

## ► Sentença **switch**

⋮	⋮	
0x000000020	18 r1, [0x103]	
0x000000024	cmpi r1, 48	
0x000000028	bne 2	
0x00000002C	bun zero	
0x000000030	bun 5	
0x000000034	cmpi r1, 49	
0x000000038	bne 2	
0x00000003C	bun um	
0x000000040	bun 1	
0x000000044	bun erro	
0x000000048	int 0	
⋮	⋮	
0x000000100	?	teda

# Controle condicional

## ► Sentença **switch**

⋮	⋮
0x000000020	18 r1, [0x103]
0x000000024	cmpi r1, 48
0x000000028	bne 2
0x00000002C	bun zero
0x000000030	bun 5
0x000000034	cmpi r1, 49
0x000000038	bne 2
0x00000003C	bun um
0x000000040	bun 1
0x000000044	bun erro
0x000000048	int 0
⋮	⋮
0x000000100	?

teda

# Controle condicional

## ► Sentença **switch**

⋮	⋮	
0x000000020	18 r1, [0x103]	
0x000000024	cmpi r1, 48	
0x000000028	bne 2	
0x00000002C	bun zero	
0x000000030	bun 5	
0x000000034	cmpi r1, 49	
0x000000038	bne 2	
0x00000003C	bun um	
0x000000040	bun 1	
0x000000044	bun erro	
0x000000048	int 0	
⋮	⋮	
0x000000100	?	teda



# Controle condicional

## ► Sentença **switch**

⋮	⋮
0x000000020	18 r1, [0x103]
0x000000024	cmpi r1, 48
0x000000028	Bne 2
0x00000002C	Bun zero
0x000000030	Bun 5
0x000000034	cmpi r1, 49
0x000000038	Bne 2
0x00000003C	Bun um
0x000000040	Bun 1
0x000000044	Bun erro
0x000000048	int 0
⋮	⋮
0x000000100	?

teda

# Controle condicional

## ► Sentença **switch**

⋮	⋮
0x000000020	18 r1, [0x103]
0x000000024	cmpi r1, 48
0x000000028	bne 2
0x00000002C	bun zero
0x000000030	bun 5
0x000000034	cmpi r1, 49
0x000000038	bne 2
0x00000003C	bun um
0x000000040	bun 1
0x000000044	bun erro
0x000000048	int 0
⋮	⋮
0x000000100	?

teda

# Controle condicional

## ► Sentença **switch**

⋮	⋮
0x000000020	18 r1, [0x103]
0x000000024	cmpi r1, 48
0x000000028	Bne 2
0x00000002C	Bun zero
0x000000030	Bun 5
0x000000034	cmpi r1, 49
0x000000038	Bne 2
0x00000003C	Bun um
0x000000040	Bun 1
0x000000044	Bun erro
0x000000048	int 0
⋮	⋮
0x000000100	?

teda

# Controle condicional

## ► Sentença **switch**

⋮	⋮
0x000000020	18 r1, [0x103]
0x000000024	cmpi r1, 48
0x000000028	Bne 2
0x00000002C	Bun zero
0x000000030	Bun 5
0x000000034	cmpi r1, 49
0x000000038	Bne 2
0x00000003C	Bun um
0x000000040	Bun 1
0x000000044	Bun erro
0x000000048	int 0
⋮	⋮
0x000000100	?

teda

# Controle condicional

## ► Sentença **switch**

⋮	⋮
0x000000020	18 r1, [0x103]
0x000000024	cmpi r1, 48
0x000000028	bne 2
0x00000002C	bun zero
0x000000030	bun 5
0x000000034	cmpi r1, 49
0x000000038	bne 2
0x00000003C	bun um
0x000000040	bun 1
0x000000044	bun erro
0x000000048	int 0
⋮	⋮
0x000000100	?

teda

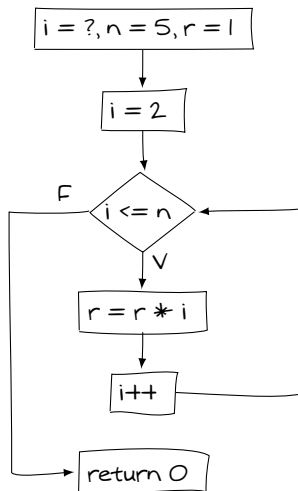
# Controle iterativo

## ► Sentença **for**

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Declaração de variáveis
6     uint32_t i, n = 5, r = 1;
7     // Controle iterativo for
8     for(i = 2; i <= n; i++) {
9         // r = r * i
10        r = r * i;
11    }
12    // Retorno sem erros
13    return 0;
14 }
```

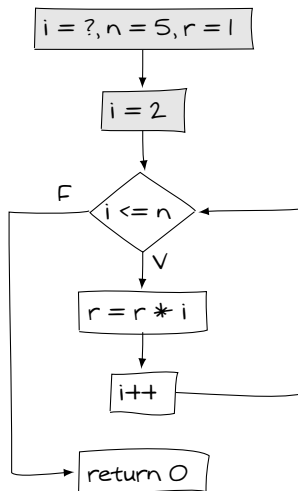
# Controle iterativo

- ▶ Sentença **for**
- ▶ Fluxo de execução



# Controle iterativo

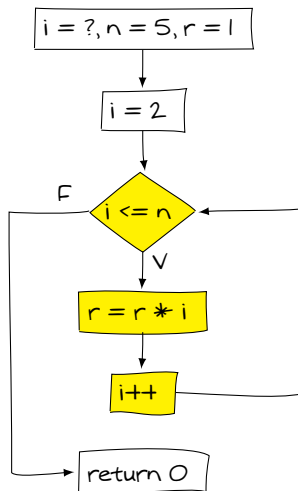
- ▶ Sentença **for**
- ▶ Fluxo de execução





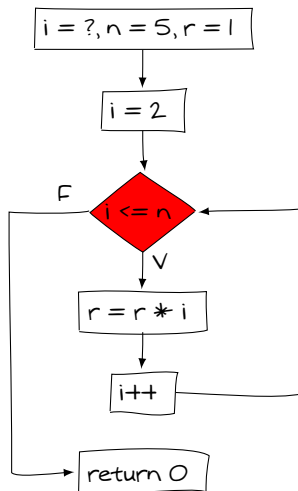
# Controle iterativo

- ▶ Sentença **for**
- ▶ Fluxo de execução



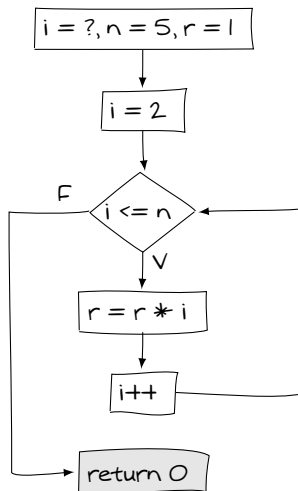
# Controle iterativo

- ▶ Sentença **for**
- ▶ Fluxo de execução



# Controle iterativo

- ▶ Sentença **for**
- ▶ Fluxo de execução



# Controle iterativo

## ► Sentença **for**

```
1 // Função principal
2 main:
3     // r1 = i = 2, r2 = n = 5, r3 = r = 1
4     mov r1, 2
5     mov r2, 5
6     mov r3, 1
7     // i ? n
8     cmp r1, r2
9     // r1 > r2
10    bgt 3
11    // r = r * i
12    mul r3, r3, r1
13    // i++
14    addi r1, r1, 1
15    bun -5
16    // Fim
17    int 0
```

# Controle iterativo

## ► Sentença **for**

⋮	⋮
0x000000020	mov r1, 2
0x000000024	mov r2, 5
0x000000028	mov r3, 1
0x00000002C	cmp r1, r2
0x000000030	Bgt 3
0x000000034	mul r3, r3, r1
0x000000038	addi r1, r1, 1
0x00000003C	Bun -5
0x000000040	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **for**

⋮	⋮
0x000000020	mov r1, 2
0x000000024	mov r2, 5
0x000000028	mov r3, 1
0x00000002C	cmp r1, r2
0x000000030	Bgt 3
0x000000034	mul r3, r3, r1
0x000000038	addi r1, r1, 1
0x00000003C	Bun -5
0x000000040	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **for**

⋮	⋮
0x000000020	mov r1, 2
0x000000024	mov r2, 5
0x000000028	mov r3, 1
0x00000002C	cmp r1, r2
0x000000030	Bgt 3
0x000000034	mul r3, r3, r1
0x000000038	addi r1, r1, 1
0x00000003C	Bun -5
0x000000040	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **for**

⋮	⋮
0x000000020	mov r1, 2
0x000000024	mov r2, 5
0x000000028	mov r3, 1
0x00000002C	cmp r1, r2
0x000000030	Bgt 3
0x000000034	mul r3, r3, r1
0x000000038	addi r1, r1, 1
0x00000003C	Bun -5
0x000000040	int 0
⋮	⋮



# Controle iterativo

## ► Sentença **for**

⋮	⋮
0x000000020	mov r1, 2
0x000000024	mov r2, 5
0x000000028	mov r3, 1
0x00000002C	cmp r1, r2
0x000000030	Bgt 3
0x000000034	mul r3, r3, r1
0x000000038	addi r1, r1, 1
0x00000003C	Bun -5
0x000000040	int 0
⋮	⋮

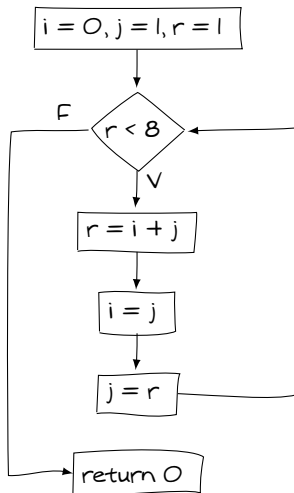
# Controle iterativo

## ► Sentença **while**

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Declaração de variáveis
6     uint32_t i = 0, j = 1, r = 1;
7     // Controle iterativo while
8     while(r < 8) {
9         // r = i + j
10        r = i + j
11        // i = j, j = r
12        i = j;
13        j = r
14    }
15    // Retorno sem erros
16    return 0;
17 }
```

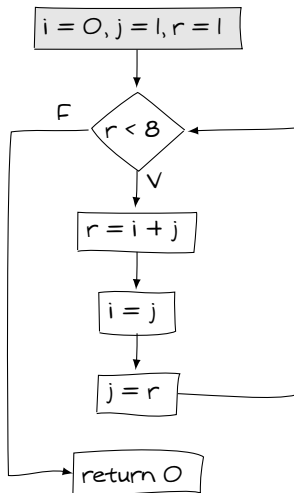
# Controle iterativo

- ▶ Sentença **while**
- ▶ Fluxo de execução



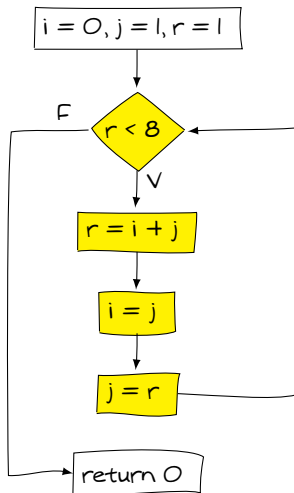
# Controle iterativo

- ▶ Sentença **while**
- ▶ Fluxo de execução



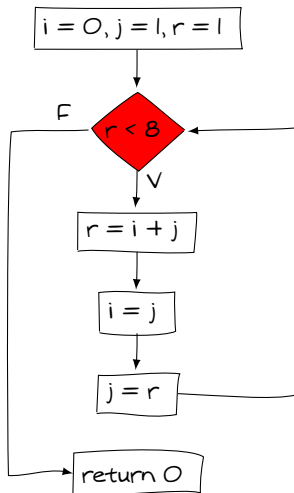
# Controle iterativo

- ▶ Sentença **while**
- ▶ Fluxo de execução



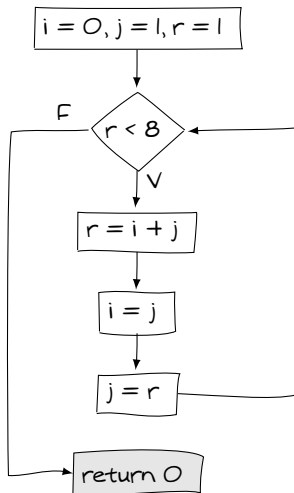
# Controle iterativo

- ▶ Sentença **while**
- ▶ Fluxo de execução



# Controle iterativo

- ▶ Sentença **while**
- ▶ Fluxo de execução



# Controle iterativo

## ► Sentença **while**

```
1 // Função principal
2 main:
3     // r1 = i = 0, r2 = j = 1, r3 = r = 1
4     mov r1, 0
5     mov r2, 1
6     mov r3, 1
7     // r ? 8
8     cmpi r3, 8
9     // r >= 8
10    bge 4
11    // r = i + j
12    add r3, r1, r2
13    // i = j, j = r
14    mov r1, r2
15    mov r2, r3
16    bun -6
17    // Fim
18    int 0
```



# Controle iterativo

## ► Sentença **while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	cmpi r3, 8
0x000000030	Bge 4
0x000000034	add r3, r1, r2
0x000000038	mov r1, r2
0x00000003C	mov r2, r3
0x000000040	Bun -6
0x000000044	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	cmpi r3, 8
0x000000030	Bge 4
0x000000034	add r3, r1, r2
0x000000038	mov r1, r2
0x00000003C	mov r2, r3
0x000000040	Bun -6
0x000000044	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	cmpi r3, 8
0x000000030	Bge 4
0x000000034	add r3, r1, r2
0x000000038	mov r1, r2
0x00000003C	mov r2, r3
0x000000040	Bun -6
0x000000044	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	cmpi r3, 8
0x000000030	Bge 4
0x000000034	add r3, r1, r2
0x000000038	mov r1, r2
0x00000003C	mov r2, r3
0x000000040	Bun -6
0x000000044	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	cmpi r3, 8
0x000000030	Bge 4
0x000000034	add r3, r1, r2
0x000000038	mov r1, r2
0x00000003C	mov r2, r3
0x000000040	Bun -6
0x000000044	int 0
⋮	⋮

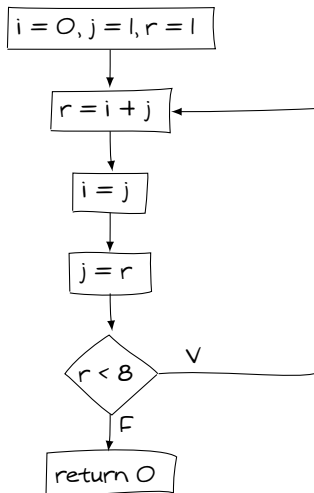
# Controle iterativo

## ► Sentença **do while**

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Declaração de variáveis
6     uint32_t i = 0, j = 1, r = 1;
7     // Controle iterativo do while
8     do {
9         // r = i + j
10        r = i + j
11        // i = j, j = r
12        i = j;
13        j = r
14    }
15    while(r < 8);
16    // Retorno sem erros
17    return 0;
18 }
```

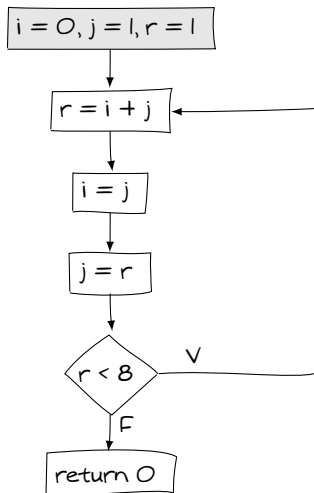
# Controle iterativo

- ▶ Sentença **do while**
- ▶ Fluxo de execução



# Controle iterativo

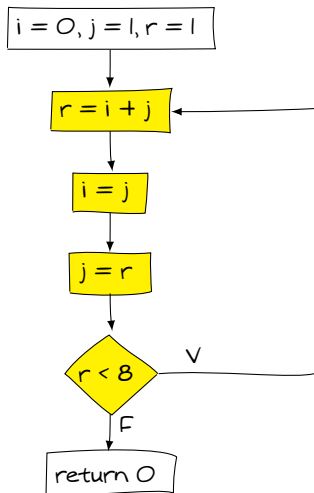
- ▶ Sentença **do while**
- ▶ Fluxo de execução





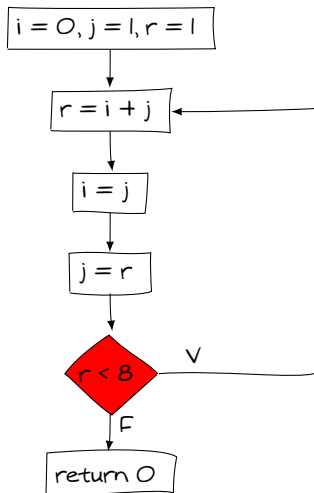
# Controle iterativo

- ▶ Sentença **do while**
- ▶ Fluxo de execução



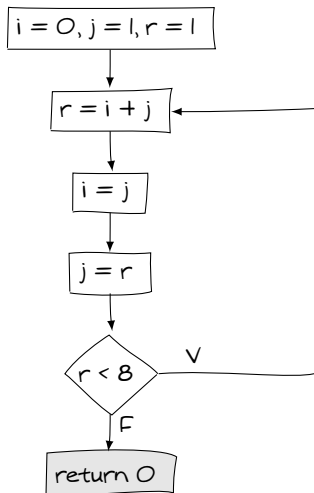
# Controle iterativo

- ▶ Sentença **do while**
- ▶ Fluxo de execução



# Controle iterativo

- ▶ Sentença **do while**
- ▶ Fluxo de execução



# Controle iterativo

## ► Sentença **do while**

```
1 // Função principal
2 main:
3     // r1 = i = 0, r2 = j = 1, r3 = r = 1
4     mov r1, 0
5     mov r2, 1
6     mov r3, 1
7     // r = i + j
8     add r3, r1, r2
9     // i = j, j = r
10    mov r1, r2
11    mov r2, r3
12    // r ? 8
13    cmpi r3, 8
14    // r < 8
15    blt -5
16    // Fim
17    int 0
```

# Controle iterativo

## ► Sentença **do while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	add r3, r1, r2
0x000000030	mov r1, r2
0x000000034	mov r2, r3
0x000000038	cmpi r3, 8
0x00000003C	blt -5
0x000000040	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **do while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	add r3, r1, r2
0x000000030	mov r1, r2
0x000000034	mov r2, r3
0x000000038	cmpi r3, 8
0x00000003C	blt -5
0x000000040	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **do while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	add r3, r1, r2
0x000000030	mov r1, r2
0x000000034	mov r2, r3
0x000000038	cmpi r3, 8
0x00000003C	blt -5
0x000000040	int 0
⋮	⋮

# Controle iterativo

## ► Sentença **do while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	add r3, r1, r2
0x000000030	mov r1, r2
0x000000034	mov r2, r3
0x000000038	cmpi r3, 8
0x00000003C	blt -5
0x000000040	int 0
⋮	⋮



# Controle iterativo

## ► Sentença **do while**

⋮	⋮
0x000000020	mov r1, 0
0x000000024	mov r2, 1
0x000000028	mov r3, 1
0x00000002C	add r3, r1, r2
0x000000030	mov r1, r2
0x000000034	mov r2, r3
0x000000038	cmpi r3, 8
0x00000003C	blt -5
0x000000040	int 0
⋮	⋮

# Sub-rotina

- ▶ Qual a motivação para utilização de sub-rotina?
  - ▶ Depuração de comportamento
  - ▶ Modularização e reuso de software
  - ▶ Realização de tarefas específicas

# Sub-rotina

- ▶ Qual a motivação para utilização de sub-rotina?
  - ▶ Depuração de comportamento
  - ▶ Modularização e reuso de software
  - ▶ Realização de tarefas específicas
- ▶ Etapas na execução de uma sub-rotina
  1. Preparação dos argumentos
  2. Chamada ou invocação
  3. Execução da sub-rotina
  4. Retorno ao fluxo anterior

# Sub-rotina

- ▶ Qual a motivação para utilização de sub-rotina?
  - ▶ Depuração de comportamento
  - ▶ Modularização e reuso de software
  - ▶ Realização de tarefas específicas
- ▶ Etapas na execução de uma sub-rotina
  1. Preparação dos argumentos
    - ▶ Os parâmetros de entrada e de saída da sub-rotina podem ser passados via pilha ou registradores
  2. Chamada ou invocação
  3. Execução da sub-rotina
  4. Retorno ao fluxo anterior

# Sub-rotina

- ▶ Qual a motivação para utilização de sub-rotina?
  - ▶ Depuração de comportamento
  - ▶ Modularização e reuso de software
  - ▶ Realização de tarefas específicas
- ▶ Etapas na execução de uma sub-rotina
  1. Preparação dos argumentos
  2. Chamada ou invocação
    - ▶ Antes do PC ser atualizado com o endereço da sub-rotina, o valor PC + 4 referente próxima instrução é salvo na pilha (endereço de retorno)
  3. Execução da sub-rotina
  4. Retorno ao fluxo anterior

# Sub-rotina

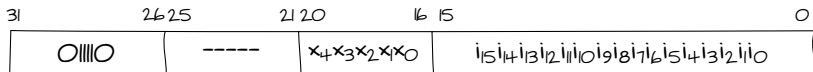
- ▶ Qual a motivação para utilização de sub-rotina?
  - ▶ Depuração de comportamento
  - ▶ Modularização e reuso de software
  - ▶ Realização de tarefas específicas
- ▶ Etapas na execução de uma sub-rotina
  1. Preparação dos argumentos
  2. Chamada ou invocação
  3. Execução da sub-rotina
    - ▶ O comportamento descrito na função ou procedimento é executado, sendo de responsabilidade do programador o salvamento e a restauração do contexto
  4. Retorno ao fluxo anterior

# Sub-rotina

- ▶ Qual a motivação para utilização de sub-rotina?
  - ▶ Depuração de comportamento
  - ▶ Modularização e reuso de software
  - ▶ Realização de tarefas específicas
- ▶ Etapas na execução de uma sub-rotina
  1. Preparação dos argumentos
  2. Chamada ou invocação
  3. Execução da sub-rotina
  4. Retorno ao fluxo anterior
    - ▶ No final da execução da sub-rotina, o endereço salvo na pilha é restaurado para o PC, retomando o fluxo antes da chamada da sub-rotina

# Sub-rotina

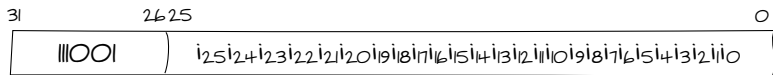
- ▶ Operação de chamada de sub-rotina (call)
  - ▶ Tipo F
  - ▶  $MEM[SP] = PC + 4, SP = SP - 4$
  - ▶  $PC = \left( R[x] + i_{15:0}^{16} \right) \ll 2$





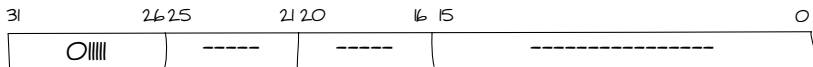
# Sub-rotina

- ▶ Operação de chamada de sub-rotina (call)
  - ▶ Tipo S
  - ▶  $MEM[SP] = PC + 4, SP = SP - 4$
  - ▶  $PC = PC + 4 + \left[ \left( i_{25}^6 : i \right) \ll 2 \right]$



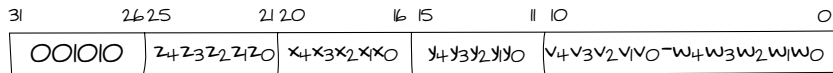
# Sub-rotina

- ▶ Operação de retorno de sub-rotina (ret)
  - ▶ Tipo F
  - ▶  $SP = SP + 4, PC = MEM[SP]$



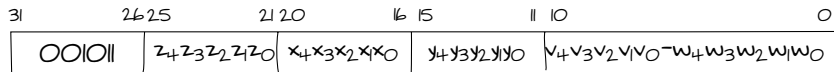
# Sub-rotina

- ▶ Operação de empilhamento (push)
  - ▶ Tipo U
  - ▶  $i = v, w, x, y, z$
  - ▶  $i \neq 0 \rightarrow MEM[SP] = R[i], SP = SP - 4$



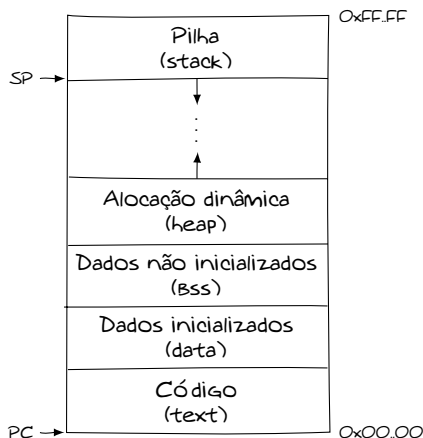
# Sub-rotina

- ▶ Operação de desempilhamento (pop)
  - ▶ Tipo U
  - ▶  $i = v, w, x, y, z$
  - ▶  $i \neq 0 \rightarrow SP = SP + 4, R[i] = MEM[SP]$



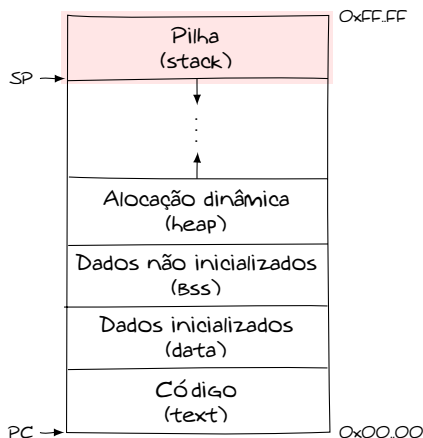
# Sub-rotina

- Estrutura de pilha na memória
  - A inserção decrementa e a remoção incrementa o valor do ponteiro do topo da pilha (SP)



# Sub-rotina

- Estrutura de pilha na memória
  - A inserção decrementa e a remoção incrementa o valor do ponteiro do topo da pilha (SP)



Alocação estática, passagem de parâmetros e suporte para chamadas aninhadas/recursivas

# Sub-rotina

## ► Implementação da função fatorial recursiva

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função fatorial
4 uint32_t fatorial(uint32_t n) {
5     // Caso base
6     if(n == 0) return 1;
7     // Caso recursivo
8     else return n * fatorial(n - 1);
9 }
10 // Função principal
11 int main() {
12     // fatorial(3)
13     uint32_t r = fatorial(3);
14     // Retorno sem erros
15     return 0;
16 }
```

# Sub-rotina

## ► Implementação da função fatorial recursiva

```
1 // Função fatorial
2 fatorial:
3     // Caso base
4     cmpi r1, 0
5     bne 2
6     mov r2, 1
7     bun 5
8     // Caso recursivo
9     push r1
10    subi r1, r1, 1
11    call fatorial
12    pop r1
13    mul r2, r2, r1
14    // Retorno da função
15    ret
...    . . .
```



# Sub-rotina

## ► Implementação da função fatorial recursiva

```
1  // Função fatorial
2  fatorial:
...  ...
16  // Função principal
17  main:
18      // SP = 32 KiB
19      mov sp, 0x7FFC
20      // fatorial(3)
21      mov r1, 3
22      call fatorial
23      // Fim
24      int 0
```

# Sub-rotina

- Execução da função fatorial recursiva

0x0000	Bun II
⋮	⋮
0x0048	mov sp, 0x7FFC
0x004C	mov r1, 3
0x0050	call -13
0x0054	int 0
⋮	⋮
0x7FF8	
0x7FFC	

$R1 = 0, R2 = 0, SP = 0$

# Sub-rotina

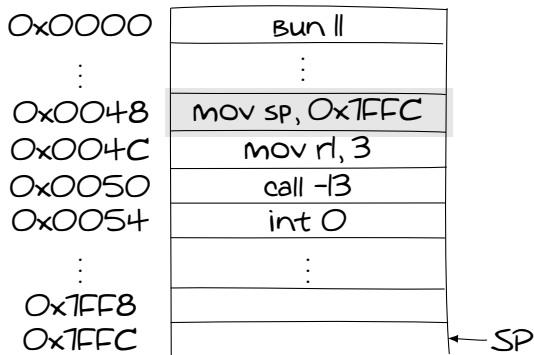
- ▶ Execução da função fatorial recursiva

0x0000	Bun II
⋮	⋮
0x0048	mov sp, 0x7FFC
0x004C	mov r1, 3
0x0050	call -13
0x0054	int 0
⋮	⋮
0x7FF8	
0x7FFC	

$R1 = 0, R2 = 0, SP = 0$

# Sub-rotina

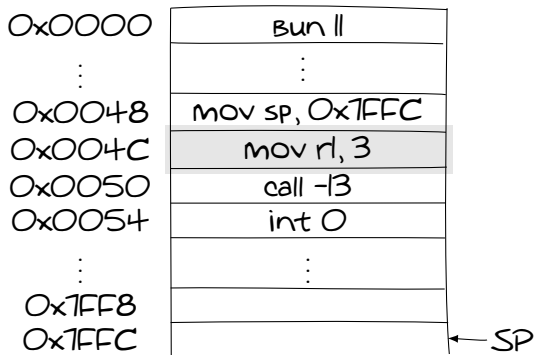
- ▶ Execução da função fatorial recursiva



$R1 = 0, R2 = 0, SP = 0x7FFC$

# Sub-rotina

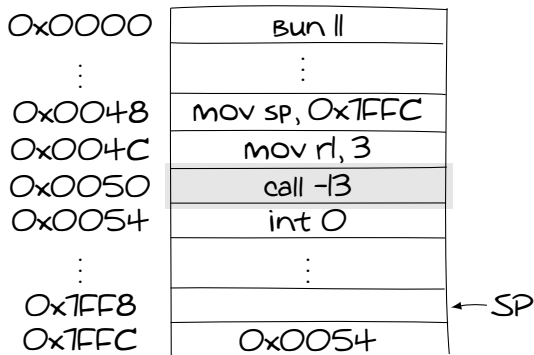
- ▶ Execução da função fatorial recursiva



$R1 = 3, R2 = 0, SP = 0x7FFC$

# Sub-rotina

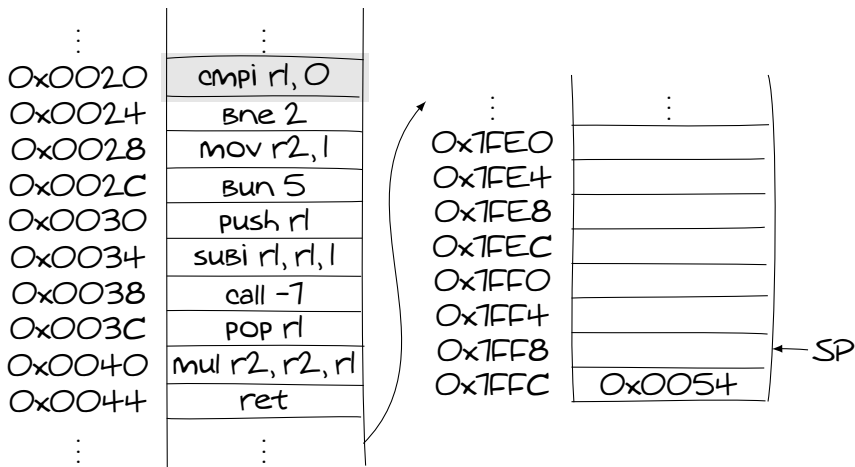
- Execução da função fatorial recursiva



$R1 = 3, R2 = 0, SP = 0x7FF8$

## Sub-rotina

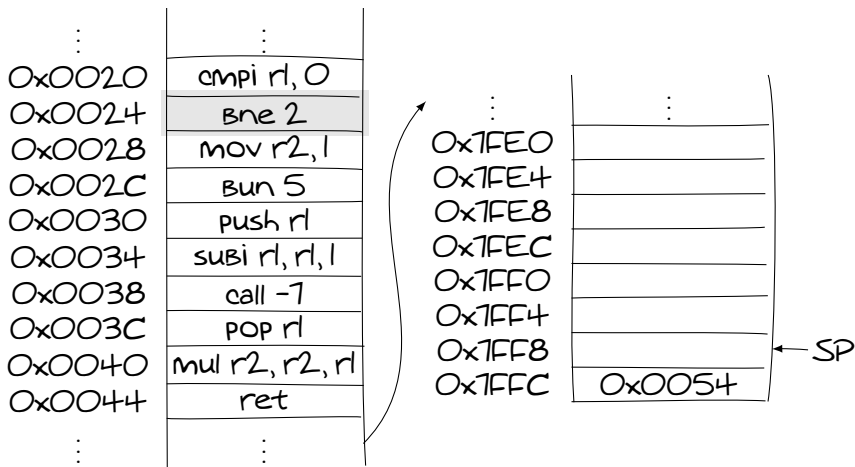
- ▶ Execução da função fatorial recursiva



$R1 = 3, R2 = 0, SP = 0x7FF8$

## Sub-rotina

- ▶ Execução da função fatorial recursiva

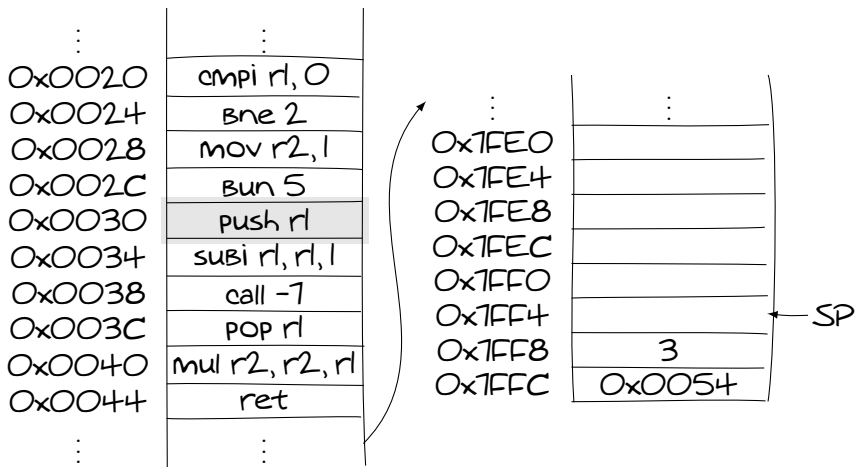


$R1 = 3, R2 = 0, SP = 0x7FF8$



## Sub-rotina

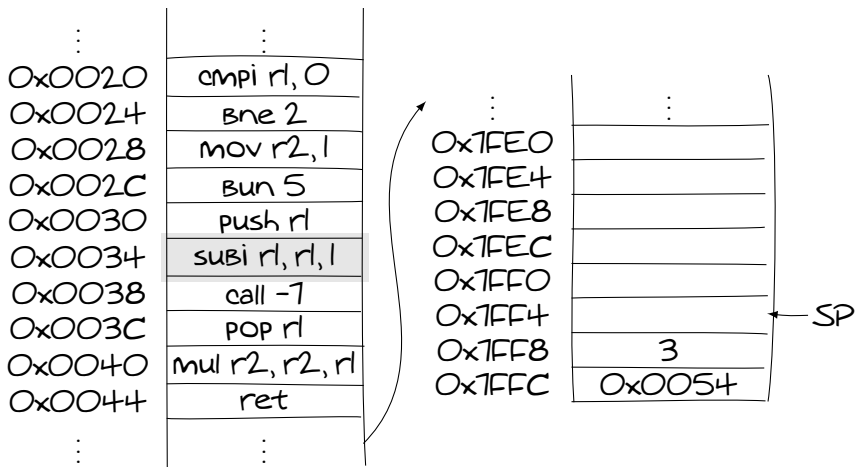
- ▶ Execução da função fatorial recursiva



$R1 = 3, R2 = 0, SP = 0x7FF4$

## Sub-rotina

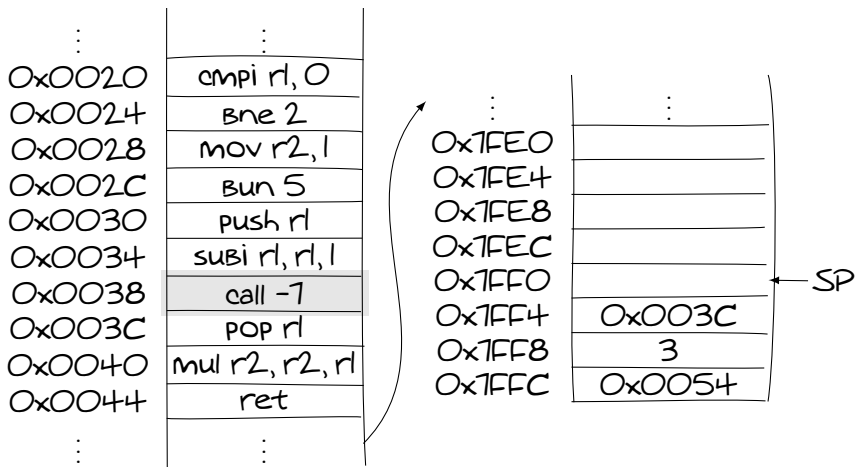
- ▶ Execução da função fatorial recursiva



$R1 = 2, R2 = 0, SP = 0x7FF4$

## Sub-rotina

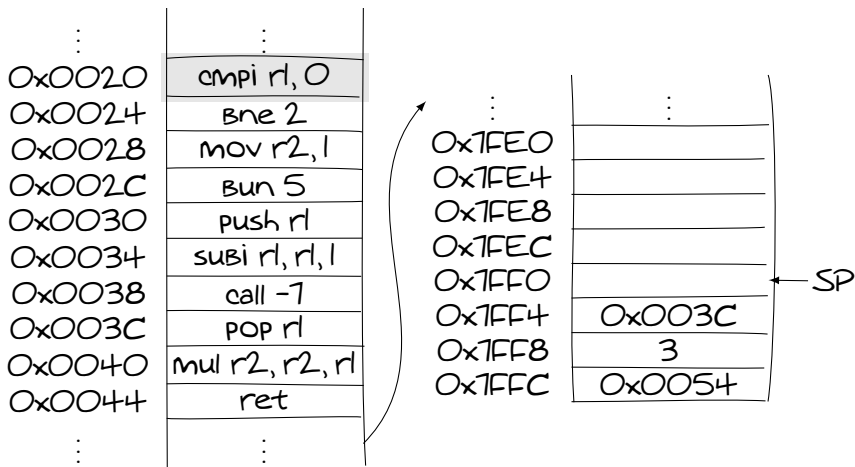
- ▶ Execução da função fatorial recursiva



$R1 = 2, R2 = 0, SP = 0x7FF0$

## Sub-rotina

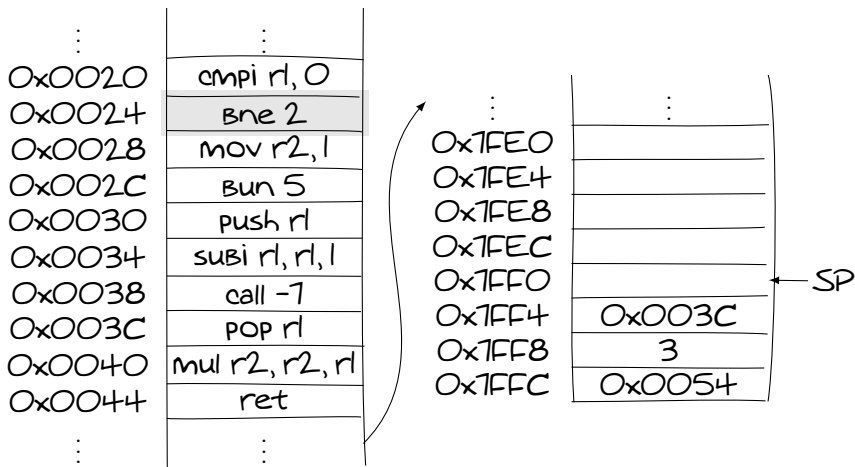
- ▶ Execução da função fatorial recursiva



$R1 = 2, R2 = 0, SP = 0x7FF0$

## Sub-rotina

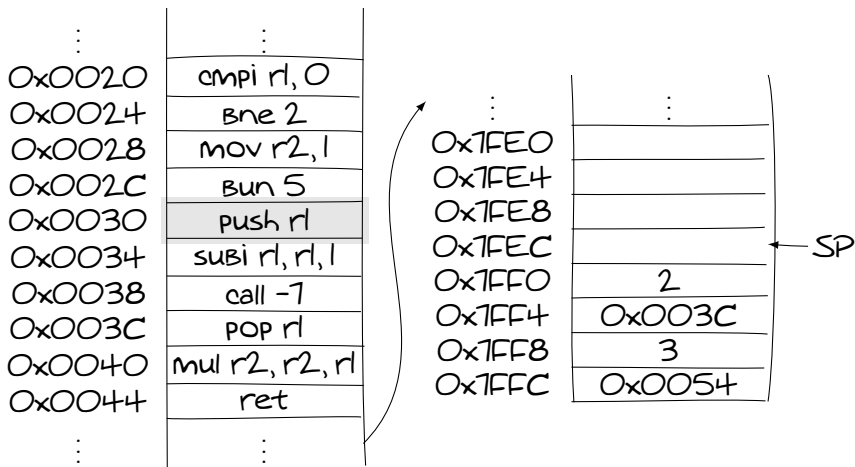
- ▶ Execução da função fatorial recursiva



$R1 = 2, R2 = 0, SP = 0x7FF0$

## Sub-rotina

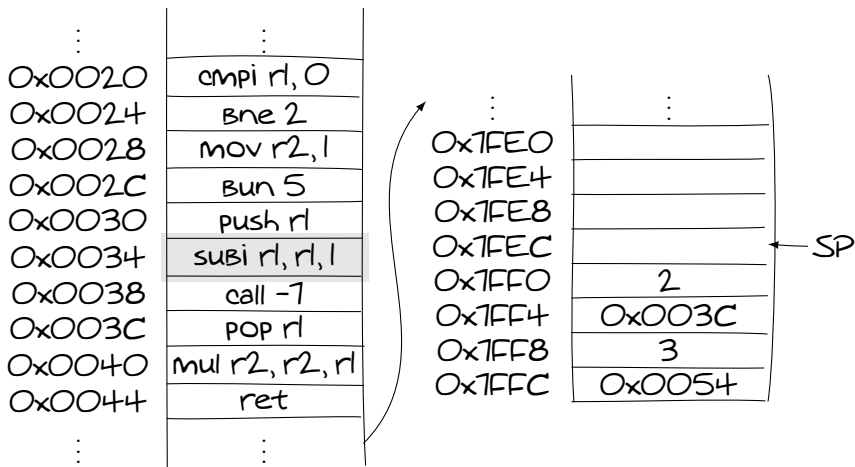
- ▶ Execução da função fatorial recursiva



$R1 = 2, R2 = 0, SP = 0x7FEC$

## Sub-rotina

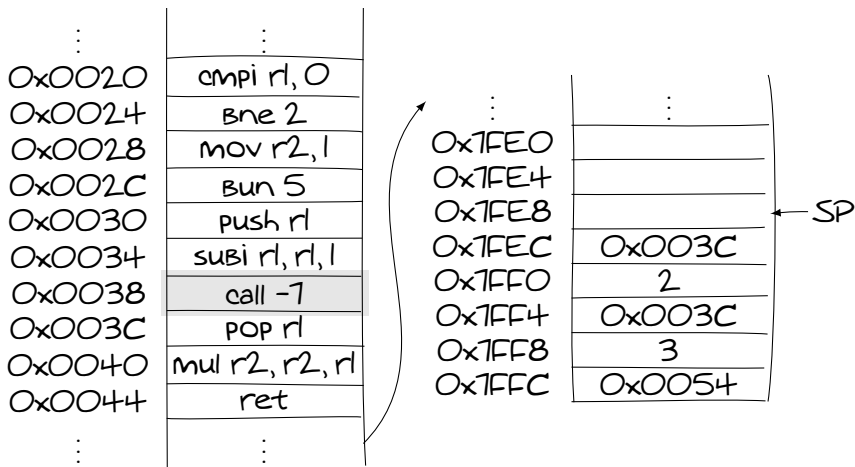
- ▶ Execução da função fatorial recursiva



$R1 = 1, R2 = 0, SP = 0x7FEC$

## Sub-rotina

- ▶ Execução da função fatorial recursiva

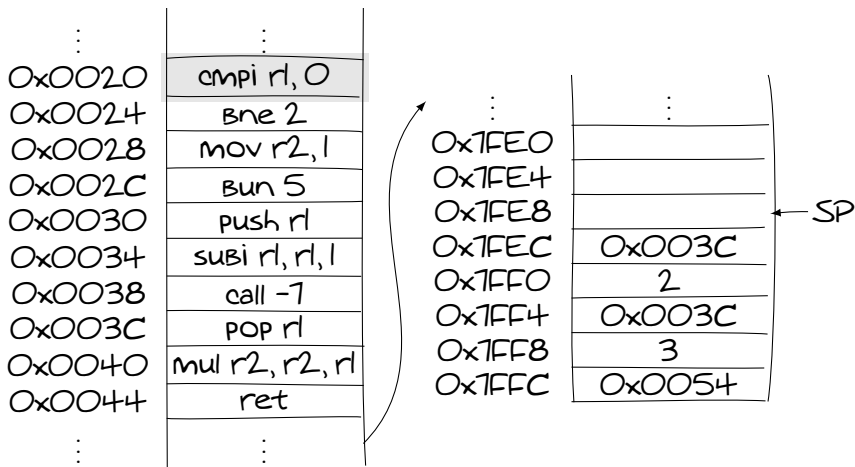


$R1 = 1, R2 = 0, SP = 0x7FE8$



## Sub-rotina

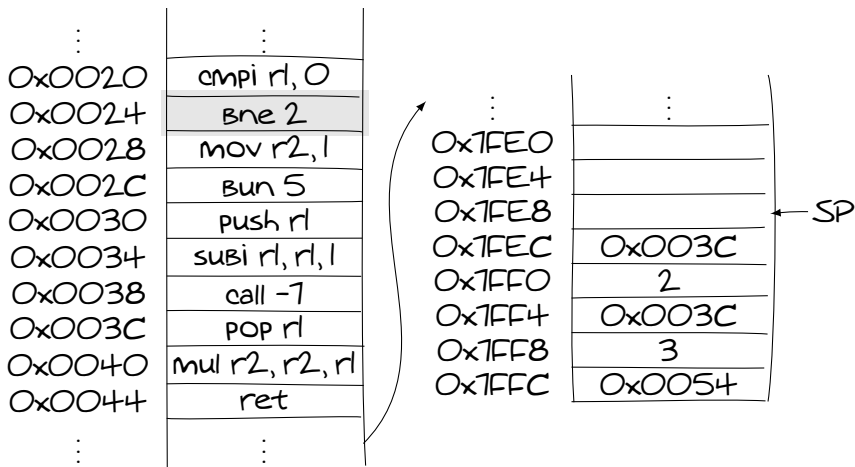
- ▶ Execução da função fatorial recursiva



$R1 = 1, R2 = 0, SP = 0x7FE8$

## Sub-rotina

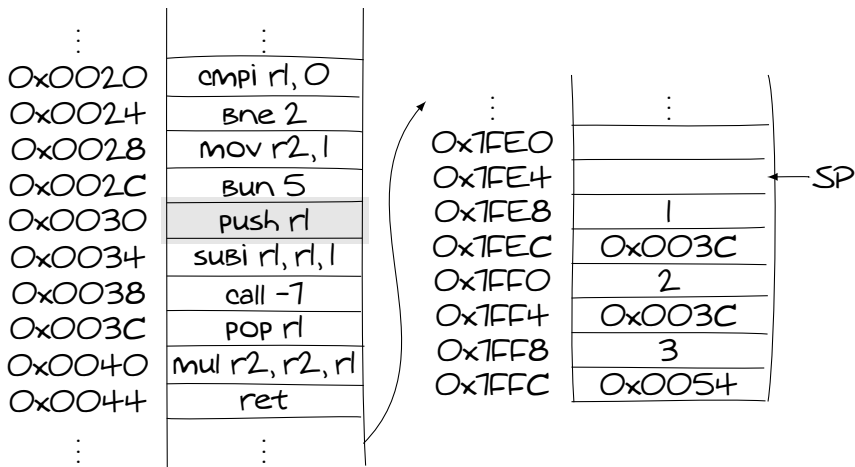
- ▶ Execução da função fatorial recursiva



$R1 = 1, R2 = 0, SP = 0x7FE8$

## Sub-rotina

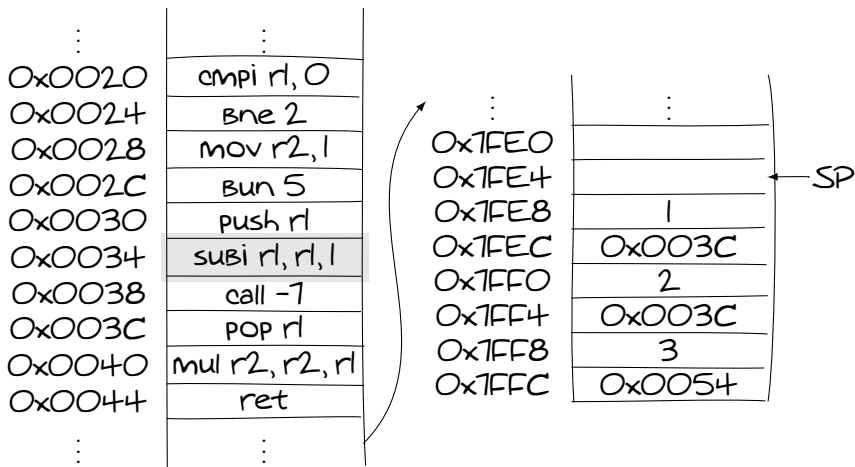
- ▶ Execução da função fatorial recursiva



$R1 = 1, R2 = 0, SP = 0x7FE4$

## Sub-rotina

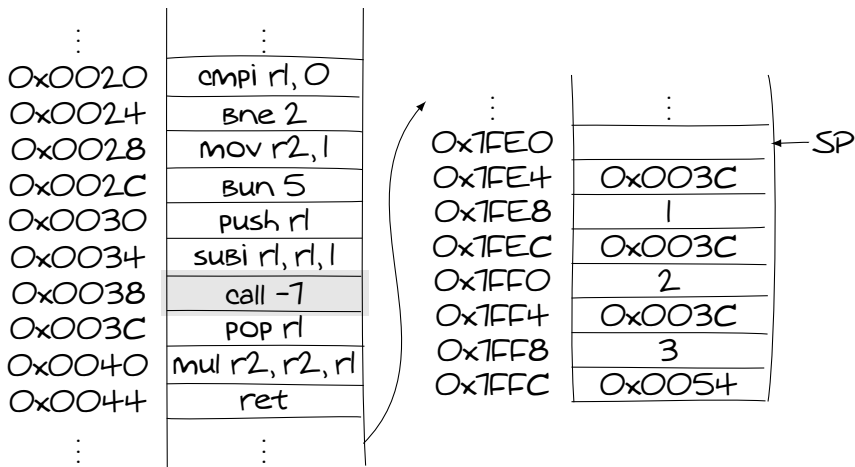
- ▶ Execução da função fatorial recursiva



$R1 = 0, R2 = 0, SP = 0x7FE4$

## Sub-rotina

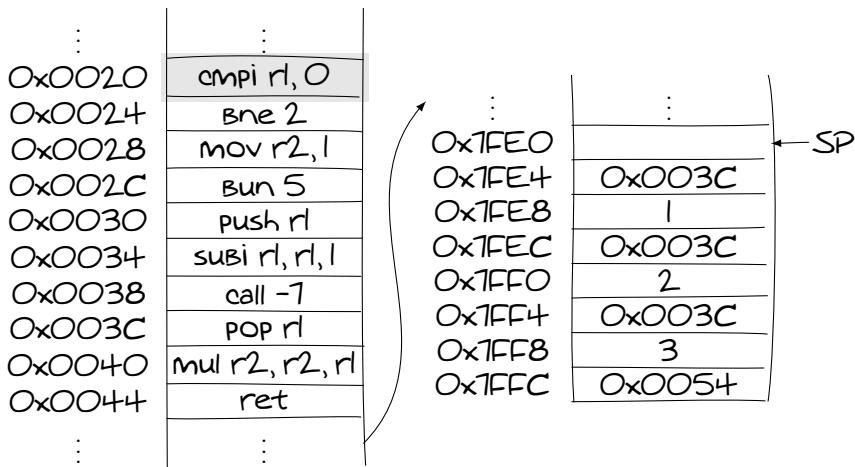
- ▶ Execução da função fatorial recursiva



$R1 = 0, R2 = 0, SP = 0x7FE0$

## Sub-rotina

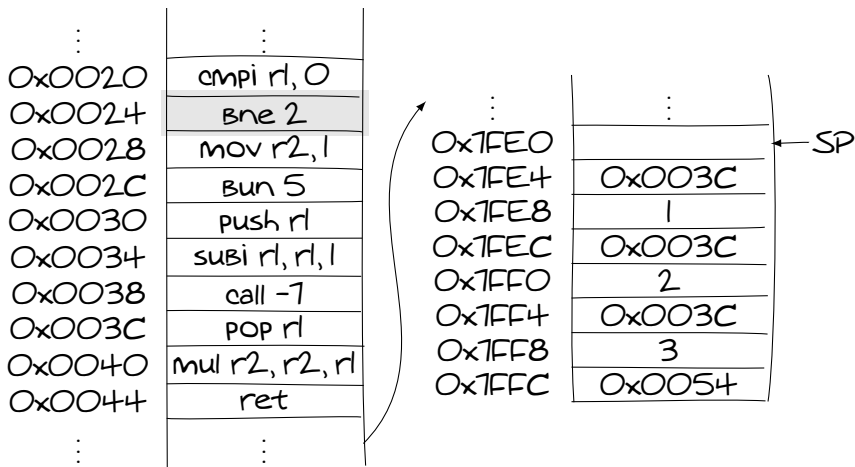
- ▶ Execução da função fatorial recursiva



$R1 = 0, R2 = 0, SP = 0x7FE0$

## Sub-rotina

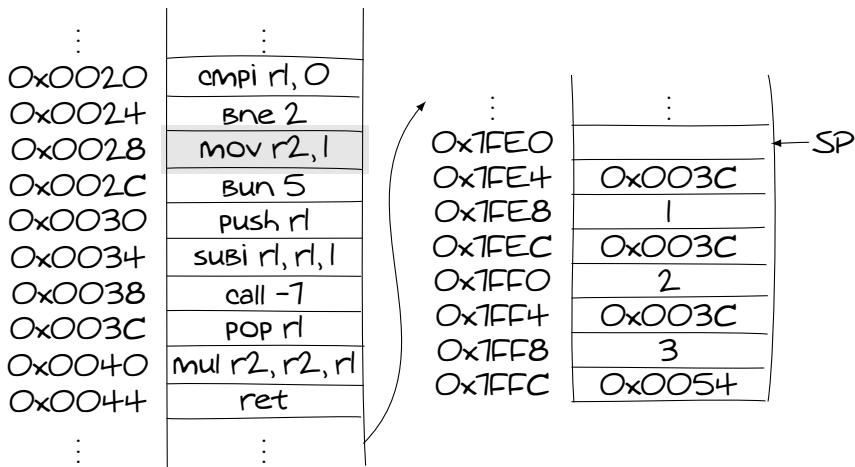
- ▶ Execução da função fatorial recursiva



$R1 = 0, R2 = 0, SP = 0x7FE0$

## Sub-rotina

- ▶ Execução da função fatorial recursiva

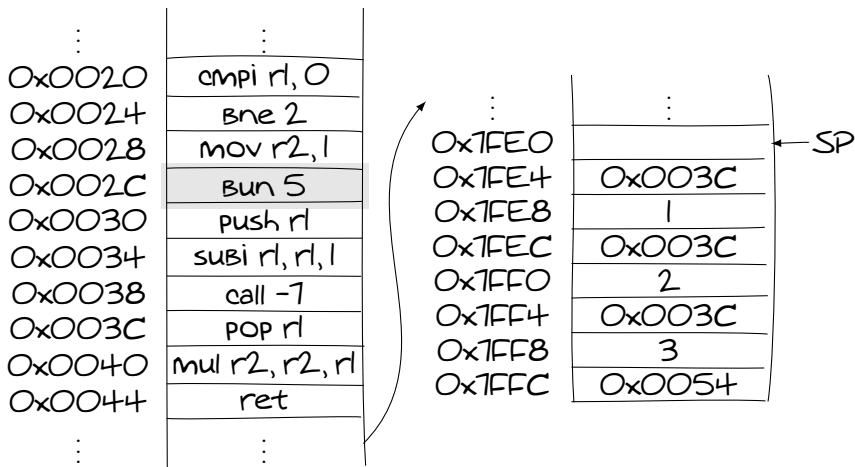


$R1 = 0, R2 = 1, SP = 0x7FE0$



## Sub-rotina

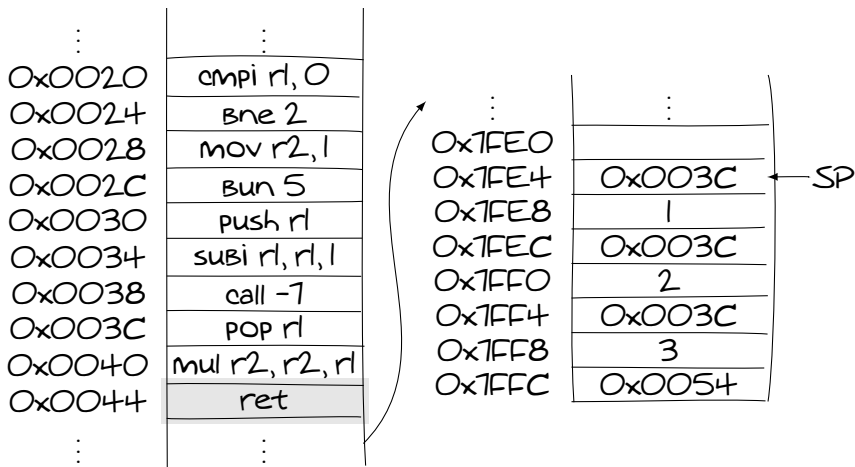
- ▶ Execução da função fatorial recursiva



$R1 = 0, R2 = 1, SP = 0x7FE0$

## Sub-rotina

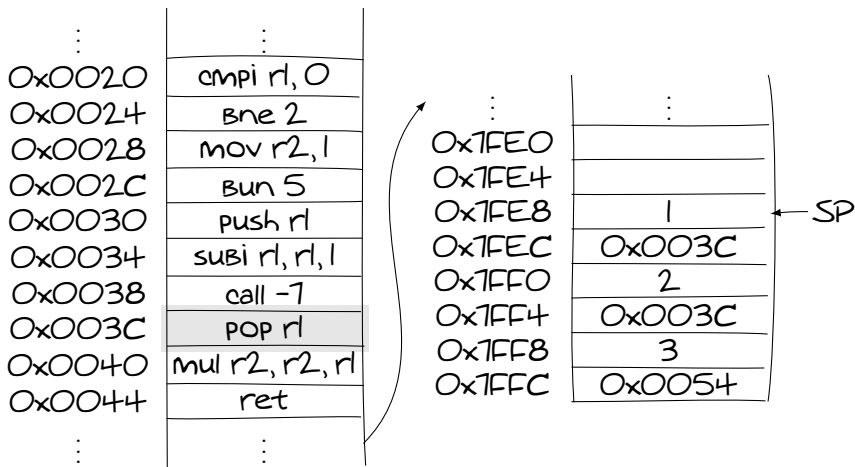
- ▶ Execução da função fatorial recursiva



$R1 = 0, R2 = 1, SP = 0x7FE4$

## Sub-rotina

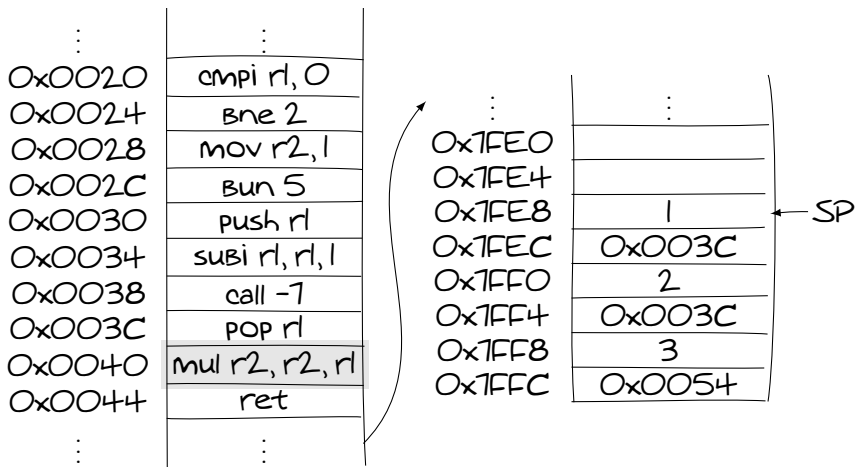
- ▶ Execução da função fatorial recursiva



$R1 = 1, R2 = 1, SP = 0x7FE8$

## Sub-rotina

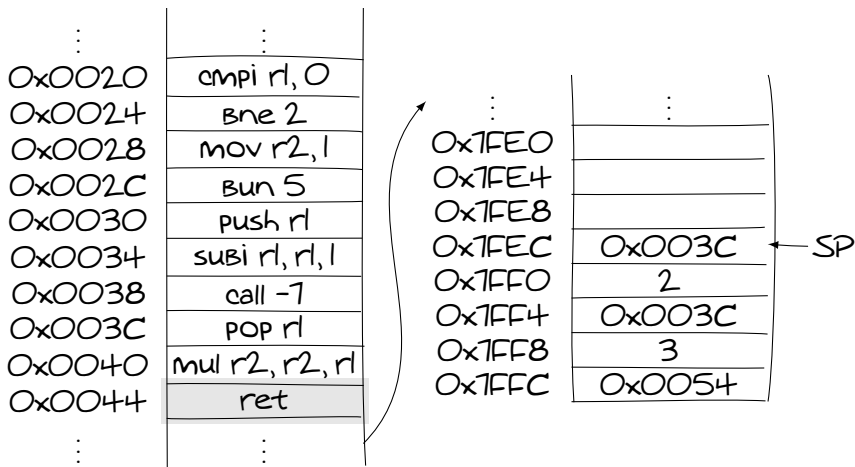
- ▶ Execução da função fatorial recursiva



$R1 = 1, R2 = 1, SP = 0x7FE8$

## Sub-rotina

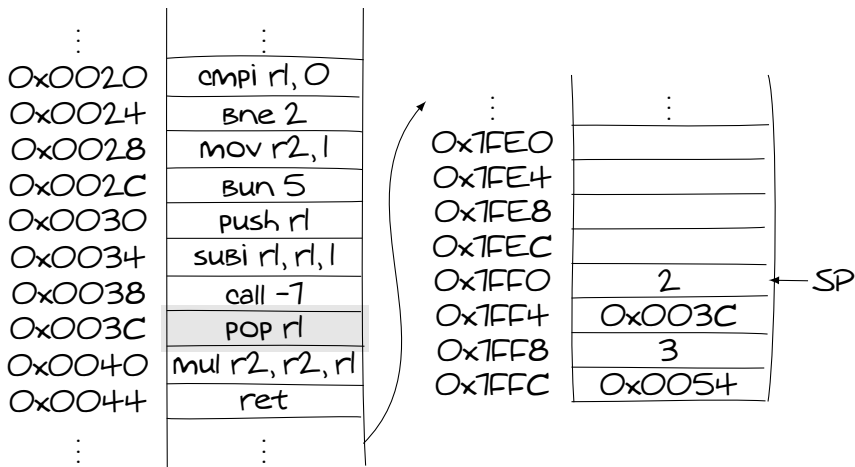
- ▶ Execução da função fatorial recursiva



$R1 = 1, R2 = 1, SP = 0x7FEC$

## Sub-rotina

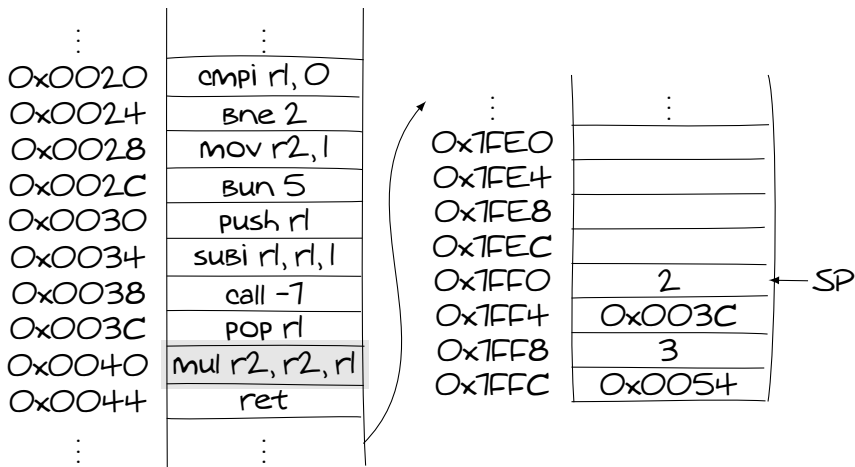
- ▶ Execução da função fatorial recursiva



$R1 = 2, R2 = 1, SP = 0x7FF0$

## Sub-rotina

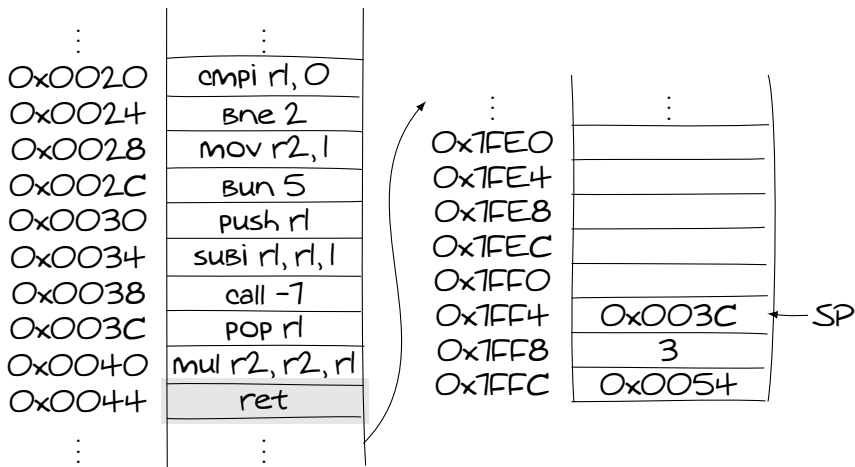
- ▶ Execução da função fatorial recursiva



$R1 = 2, R2 = 2, SP = 0x7FF0$

## Sub-rotina

- ▶ Execução da função fatorial recursiva

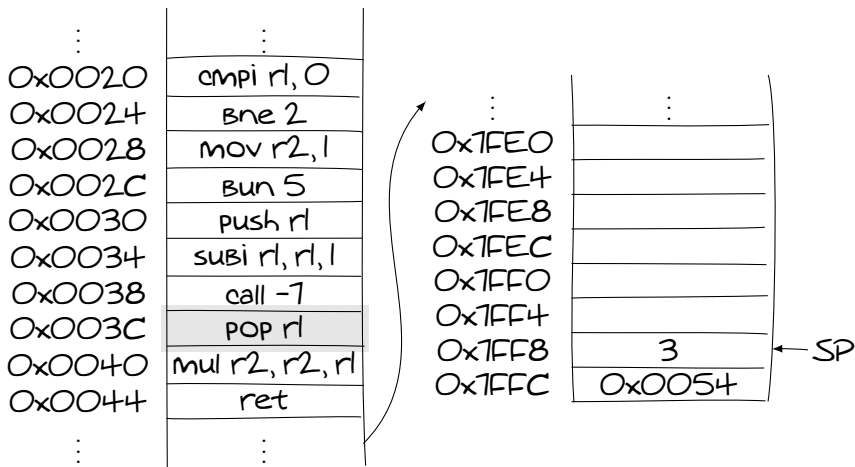


$R1 = 2, R2 = 2, SP = 0x7FF4$



## Sub-rotina

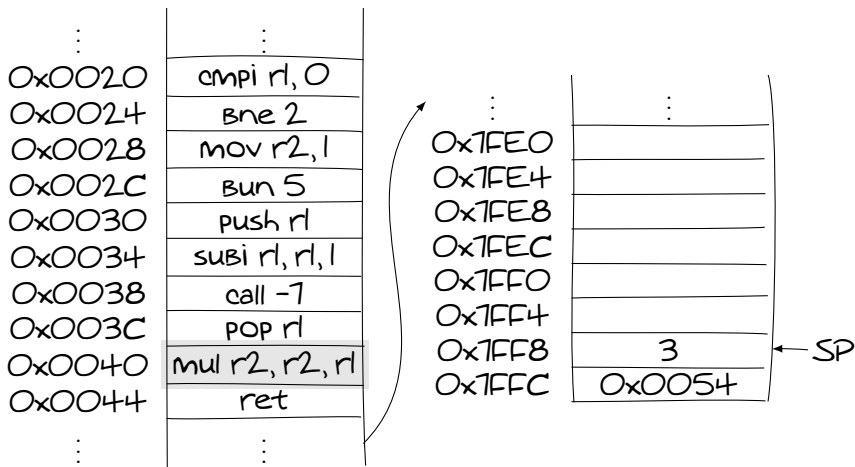
- ▶ Execução da função fatorial recursiva



$R1 = 3, R2 = 2, SP = 0x7FF8$

## Sub-rotina

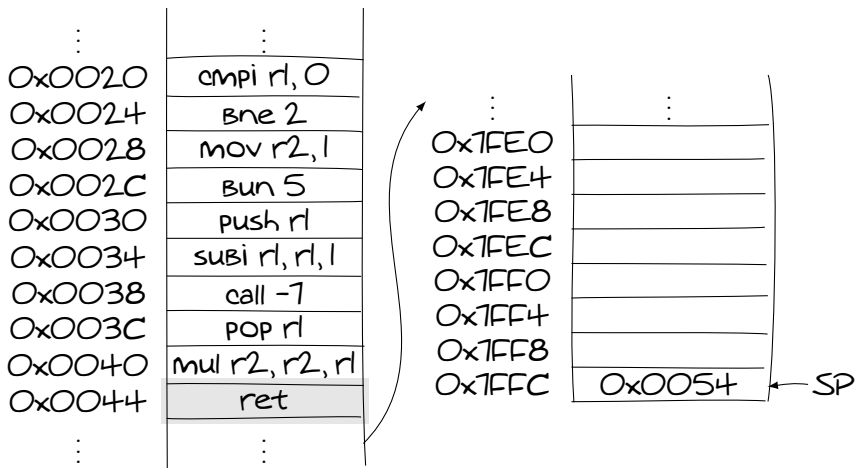
- ▶ Execução da função fatorial recursiva



$R1 = 3, R2 = 6, SP = 0x7FF8$

## Sub-rotina

- ▶ Execução da função fatorial recursiva



$R1 = 3, R2 = 6, SP = 0x7FFC$

# Sub-rotina

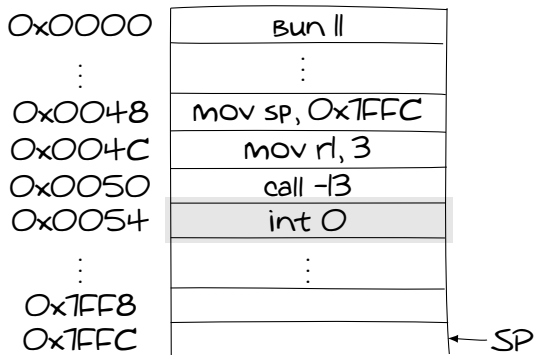
- Execução da função fatorial recursiva

0x0000	Bun II
⋮	⋮
0x0048	mov sp, 0x7FFC
0x004C	mov r1, 3
0x0050	call -13
0x0054	int 0
⋮	⋮
0x7FF8	
0x7FFC	0x0054 ← SP

$R1 = 3, R2 = 6, SP = 0x7FFC$

# Sub-rotina

- ▶ Execução da função fatorial recursiva



$R1 = 3, R2 = 6, SP = 0x7FFC$

# Sub-rotina

- Execução da função fatorial recursiva

0x0000	Bun II
⋮	⋮
0x0048	mov sp, 0x7FFC
0x004C	mov r1, 3
0x0050	call -13
0x0054	int 0
⋮	⋮
0x7FF8	
0x7FFC	← SP

$R1 = 3, R2 = 6, SP = 0x7FFC$

## Exemplo

- ▶ Considere a implementação recursiva em C da função *fibonacci* descrita abaixo
  - ▶ Realize a sua tradução para código de montagem
  - ▶ Execute passo a passo seu funcionamento para  $n = 3$

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Vetor auxiliar
4 uint32_t V[48] = { 0 };
5 // Função fibonacci
6 uint32_t fibonacci(uint32_t n) {
7     // Caso base
8     if(n <= 1) return n;
9     // Caso recursivo
10    else if(V[n] == 0) V[n] = fibonacci(n - 2) +
        fibonacci(n - 1);
11    // Retornando valor
12    return V[n];
13 }
... ..
```