



Controle de fluxo Arquitetura de Computadores

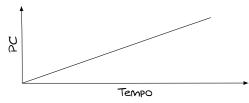
Bruno Prado

Departamento de Computação / UFS

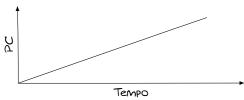
- O que é controle de fluxo?
 - Define a sequência de instruções que são executadas dinamicamente pelo processador
 - Quando não ocorrem desvios, as instruções são executadas sequencialmente incrementando o PC

- 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-roting

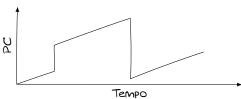
- Comportamento do fluxo de execução
 - Sequencial (sem desvios)



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



Não sequencial (com desvios)

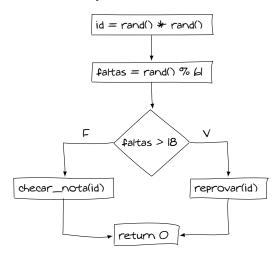


- Nos primórdios da computação, todo o controle de fluxo era de responsabilidade do desenvolvedor
 - Linguagem de máquina e de montagem
 - Controle de fluxo por desvios (goto)
 - Baixa abstração e mais erros humanos

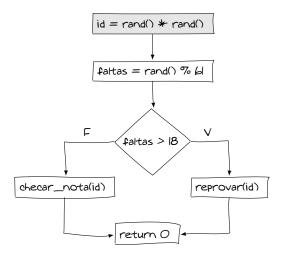
- Nos primórdios da computação, todo o controle de fluxo era de responsabilidade do desenvolvedor
 - Linguagem de máquina e de montagem
 - Controle de fluxo por desvios (goto)
 - Baixa abstração e mais erros humanos
- Com o nascimento da programação estruturada em C, o foco é descrever o comportamento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

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

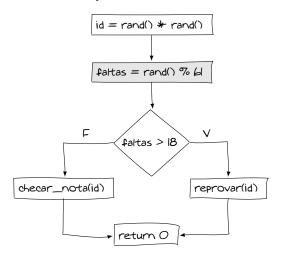
- Sentença if, else
 - ► Fluxo de execução



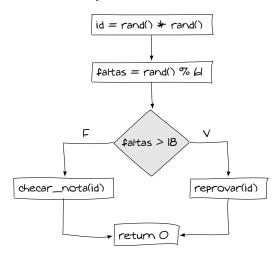
- ▶ Sentença if, else
 - ► Fluxo de execução



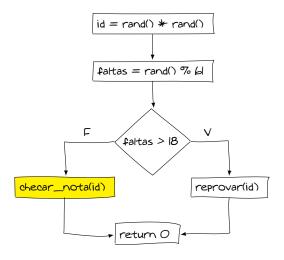
- Sentença if, else
 - ► Fluxo de execução



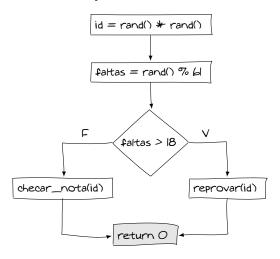
- Sentença if, else
 - ► Fluxo de execução



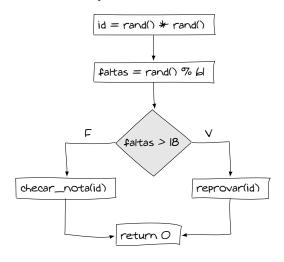
- Sentença if, else
 - ► Fluxo de execução



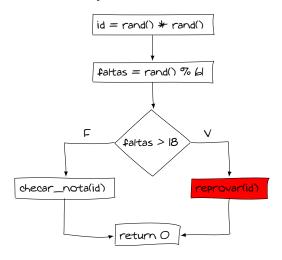
- Sentença if, else
 - ► Fluxo de execução



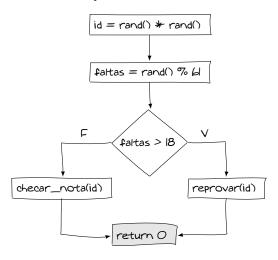
- Sentença if, else
 - ► Fluxo de execução



- Sentença if, else
 - ► Fluxo de execução



- Sentença if, else
 - ► Fluxo de execução



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

► Sentença if, else

0x00000000	Bun 7	1
:	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[0x107]	
0x00000028	cmpi r2,18	
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun	
0x00000038	Bun reprovar	
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	fá
:	:	

id faltas

0x0000000	Bun 7	
:	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[0x107]	
0x00000028	cmpi r2, 18	·
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun	
0x00000038	Bun reprovar	
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	faH
:	:	

► Sentença if, else

0x00000000	Bun 7	
<u>:</u>	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[0x107]	
0x00000028	cmpi r2, 18	
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun	_
0x00000038	Bun reprovar	
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	fal
<u>:</u>	:	

id faltas

0x00000000	Bun 7	
<u>:</u>	: :	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[0x107]	
0x00000028	cmpi r2,18	
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun	
0x00000038	Bun reprovar	
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	faH
:	:	

0x00000000	Bun 7	
:	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[0x107]	
0x00000028	cmpi r2,18	
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun	
0x00000038	Bun reprovar	•
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	faltas
:	<u>:</u>	

► Sentença if, else

0x00000000	Bun 7	
:	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[Ox107]	
0x00000028	cmpi r2,18	
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun	
0x00000038	Bun reprovar	
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	fal
<u>:</u>	<u>:</u>	

id faltas

► Sentença if, else

0x0000000	Bun 7	
:	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[0x107]	
0x00000028	cmpi r2,18	
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun	
0x00000038	Bun reprovar	•
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	fa
:	:	

id faltas

0x00000000	Bun 7	
:	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[0x107]	
0x00000028	cmpi r2,18	
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun	
0x00000038	Bun reprovar	
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	faltas
<u>:</u>	<u>:</u>	

0x00000000	Bun 7	
:	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[0x107]	
0x00000028	cmpi r2,18	L
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun l	
0x00000038	Bun reprovar	•
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	faltas
:	:	

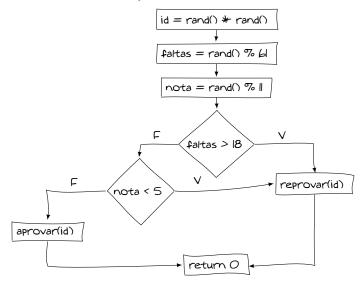
0x00000000	Bun 7	
:	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[0x107]	
0x00000028	cmpi r2, 18	·
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun	
0x00000038	Bun reprovar	
0x0000003C	int O	
:	:	
0x00000100	?	id
0x00000104	?	faltas
:	:	

0x00000000	Bun 7	
:	:	
0x00000020	132 rl,[0x40]	
0x00000024	18 r2,[Ox107]	
0x00000028	cmpi r2, 18	
0x0000002C	BGt 2	
0x00000030	Bun checar_nota	
0x00000034	Bun l	
0x00000038	Bun reprovar	
0x0000003C	int O	
<u>:</u>	:	
0x00000100	?	id
0x00000104	?	faltas
:	:	

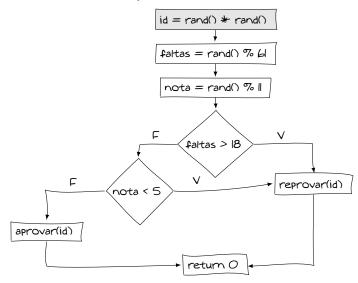
Sentença if, else if, else

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

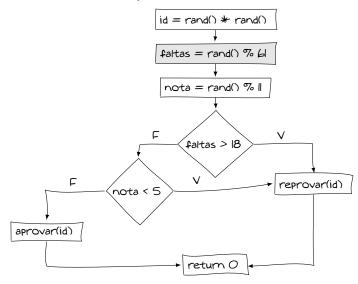
- Sentença if, else if, else
 - ► Fluxo de execução



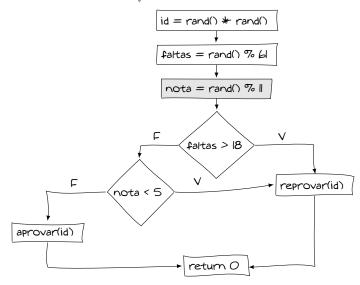
- Sentença if, else if, else
 - ► Fluxo de execução



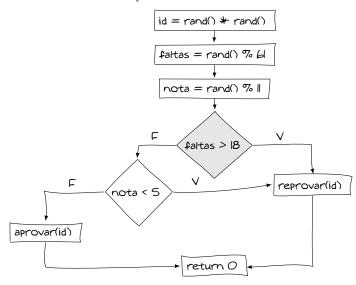
- Sentença if, else if, else
 - ► Fluxo de execução



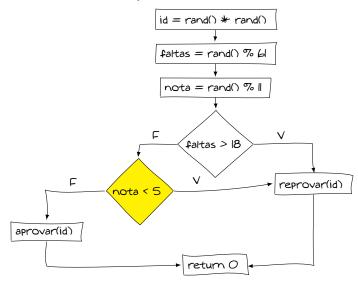
- Sentença if, else if, else
 - ► Fluxo de execução



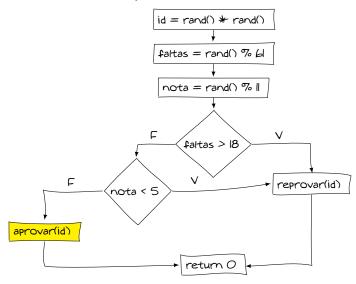
- Sentença if, else if, else
 - ► Fluxo de execução



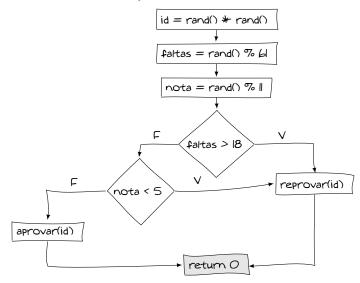
- Sentença if, else if, else
 - ► Fluxo de execução



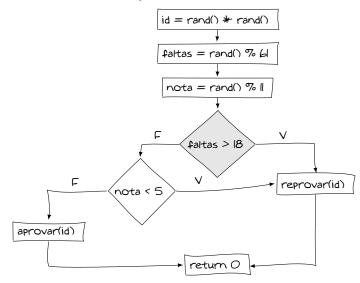
- Sentença if, else if, else
 - ► Fluxo de execução



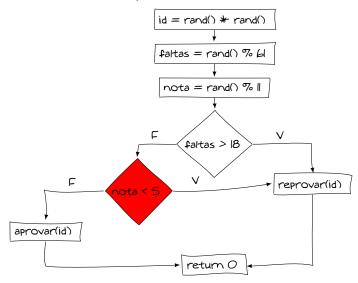
- Sentença if, else if, else
 - ► Fluxo de execução



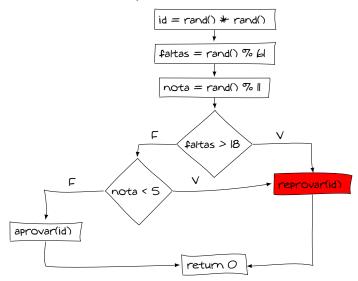
- Sentença if, else if, else
 - ► Fluxo de execução



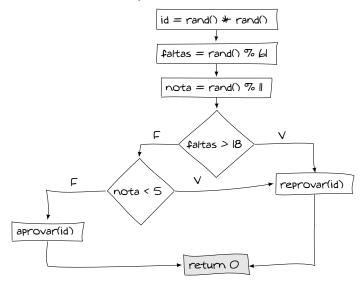
- Sentença if, else if, else
 - ► Fluxo de execução



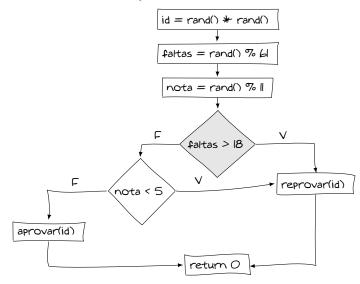
- Sentença if, else if, else
 - ► Fluxo de execução



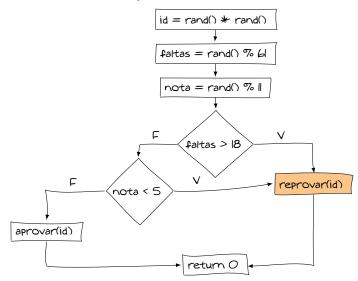
- Sentença if, else if, else
 - ► Fluxo de execução



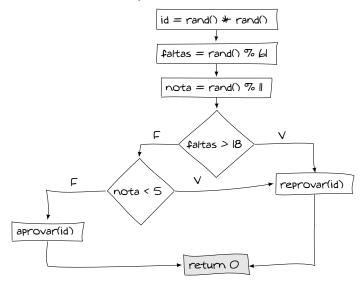
- Sentença if, else if, else
 - ► Fluxo de execução



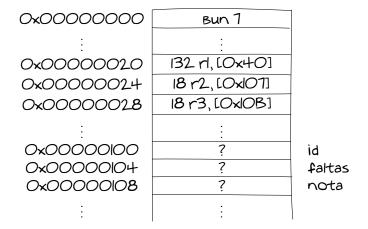
- Sentença if, else if, else
 - ► Fluxo de execução



- Sentença if, else if, else
 - ► Fluxo de execução



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



i i	<u>:</u>
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3, [OxIOB]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	cmpi r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun 1
0x00000044	Bun reprovar
0x00000048	int O
i i	÷

į.	<u>:</u>
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3, [OxIOB]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	cmpi r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun l
0x00000044	Bun reprovar
0x00000048	int O
i i	i i

i i	÷
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3, [OxIOB]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	empi r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun l
0x00000044	Bun reprovar
0x00000048	int O
i i	÷

<u>:</u>	: :
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3, [OxIOB]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	cmpi r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun 1
0x00000044	Bun reprovar
0x00000048	int O
i i	:

÷	<u>:</u>
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3,[Ox10B]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	cmpi r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun l
0x00000044	Bun reprovar
0x00000048	int O
į.	: :

<u>:</u>	<u>:</u>
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3, [OxIOB]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	cmpi r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun 1
0x00000044	Bun reprovar
0x00000048	int O
i i	<u>:</u>

÷ ·	<u>:</u>
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3, [OxIOB]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	cmpi r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun l
0x00000044	Bun reprovar
0x00000048	int O
i i	÷

i i	i i
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3,[Ox10B]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	cmpi r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun l
0x00000044	Bun reprovar
0x00000048	int O
<u> </u>	÷

÷ ·	<u>:</u>
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3, [OxIOB]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	cmpi r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun l
0x00000044	Bun reprovar
0x00000048	int O
i i	i :

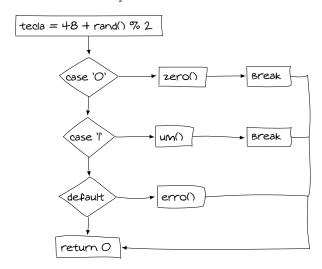
<u>:</u>	<u>:</u>
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3, [OxIOB]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	стрі r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun l
0x00000044	Bun reprovar
0x00000048	int O
i i	÷

÷ ·	<u>:</u>
0x00000020	132 rl,[0x40]
0x00000024	18 r2,[Ox107]
0x00000028	18 r3, [OxIOB]
0x0000002C	cmpi r2,18
0x00000030	BGt 4
0x00000034	стрі r3,5
0x00000038	Blt 2
0x0000003C	Bun aprovar
0x00000040	Bun
0x00000044	Bun reprovar
0x00000048	int O
Ė	i:

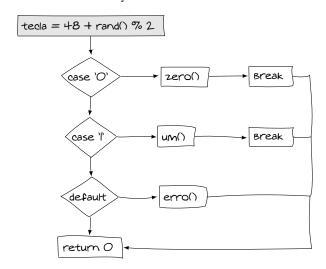
Sentença switch

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

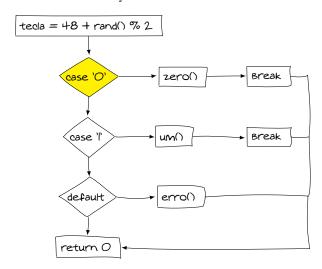
- Sentença switch
 - ► Fluxo de execução



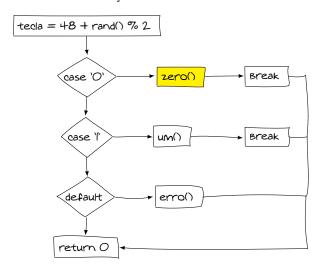
- Sentença switch
 - ▶ Fluxo de execução



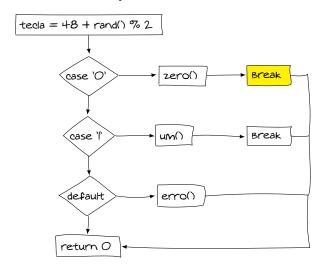
- Sentença switch
 - ► Fluxo de execução



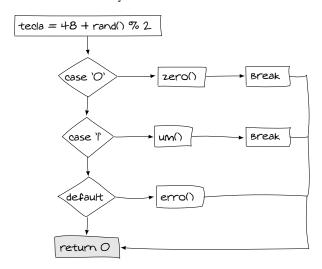
- Sentença switch
 - ► Fluxo de execução



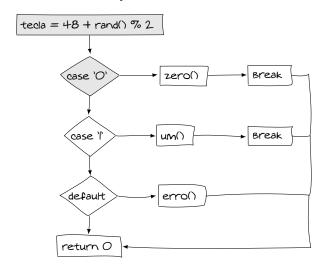
- Sentença switch
 - ► Fluxo de execução



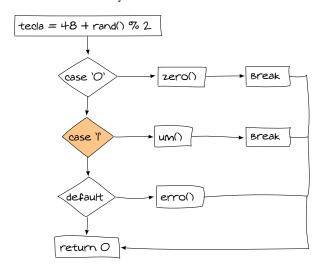
- Sentença switch
 - ► Fluxo de execução



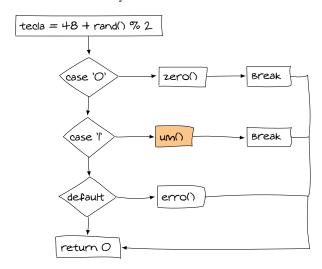
- Sentença switch
 - ▶ Fluxo de execução



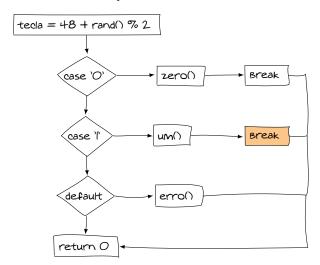
- Sentença switch
 - ► Fluxo de execução



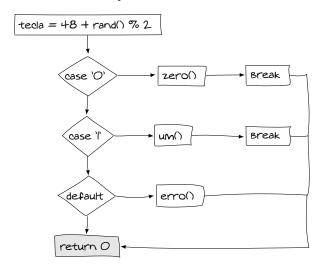
- Sentença switch
 - ► Fluxo de execução



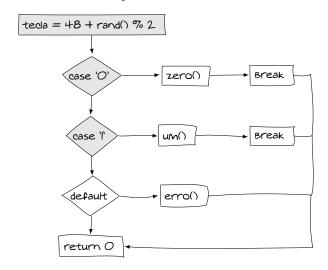
- Sentença switch
 - ► Fluxo de execução



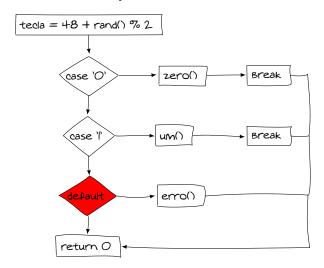
- Sentença switch
 - ► Fluxo de execução



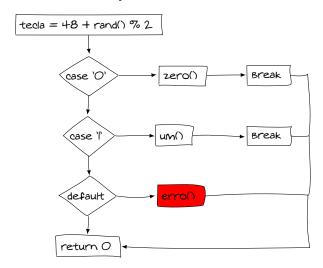
- Sentença switch
 - ► Fluxo de execução



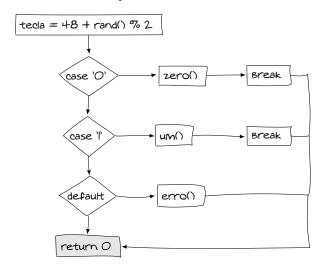
- Sentença switch
 - ► Fluxo de execução



- Sentença switch
 - ► Fluxo de execução



- Sentença switch
 - ► Fluxo de execução



Sentença switch

```
// Função principal
   main:
3
        // r1 = tecla
        18 r1, [0x103]
4
        // case '0'
5
        cmpi r1, 48
        bne 2
        bun zero
8
        bun 5
        // case '1'
10
        cmpi r1, 49
11
        bne 2
12
        bun um
1.3
14
        bun 1
15
        // default
16
        bun erro
        // Fim
17
        int 0
18
```

Sentença switch

į.	i i
0x00000020	18 rl, [Ox103]
0x00000024	cmpi rl, 48
0x00000028	Bne 2
0x0000002C	Bun zero
0x00000030	BUN 5
0x00000034	cmpi rl, 49
0x00000038	Bne 2
0x0000003C	Bun um
0x00000040	Bun 1
0x00000044	bun erro
0x00000048	int O
÷	÷
0x00000100	?

tecla

► Sentença **switch**

÷ .	÷	
0x00000020	18 rl,[0x103]	
0x00000024	cmpi rl, 48	
0x00000028	Bne 2	
0x0000002C	Bun zero	
0x00000030	Bun 5	
0x00000034	cmpi rl, 49	
0x00000038	Bne 2	
0x0000003C	Bun um	
0x00000040	Bun 1	
0x00000044	bun erro	
0x00000048	int O	
÷	÷	
0x00000100	?	tecla

Sentença switch

÷ :	i i	
0x00000020	18 rl,[Ox103]	
0x00000024	cmpi rl, 48	
0x00000028	Bne 2	
0x0000002C	Bun zero	
0x00000030	Bun 5	
0x00000034	cmpi rl, 49	
0x00000038	Bne 2	
0x0000003C	Bun um	
0x00000040	Bun l	
0x00000044	Bun erro	
0x00000048	int O	
÷	÷	
0x00000100	?	tecla

Sentença switch

<u>:</u>	i :	
0x00000020	18 rl,[Ox103]	
0x00000024	cmpi rl, 48	
0x00000028	Bne 2	
0x0000002C	Bun zero	
0x00000030	Bun 5	
0x00000034	cmpi rl, 49	
0x00000038	Bne 2	
0x0000003 C	Bun um	
0x00000040	Bun	
0x00000044	Bun erro	
0x00000048	int O	
÷	÷	
0x00000100	?	tecla

► Sentença **switch**

÷	÷	
0x00000020	18 rl, [0x103]	
0x00000024	cmpi rl, 48	
0x00000028	Bne 2	
0x0000002C	Bun zero	
0x00000030	Bun 5	
0x00000034	cmpi rl, 49	
0x00000038	Bne 2	
0x0000003C	Bun um	
0x00000040	Bun	
0x00000044	Bun erro	
0x00000048	int O	
÷	÷	
0x00000100	?	tecla

► Sentença **switch**

i i	<u>:</u>	
0x00000020	18 rl, [Ox103]	
0x00000024	cmpi rl, 48	
0x00000028	Bne 2	
0x0000002C	Bun zero	
0x00000030	Bun 5	
0x00000034	cmpi rl, 49	
0x00000038	Bne 2	
0x0000003C	Bun um	
0x00000040	Bun 1	
0x00000044	bun erro	
0x00000048	int O	
÷	÷	
0x00000100	?	te

tecla

Sentença switch

÷ :	i i	
0x00000020	18 rl,[Ox103]	
0x00000024	cmpi rl, 48	
0x00000028	Bne 2	
0x0000002C	Bun zero	
0x00000030	Bun 5	
0x00000034	cmpi rl, 49	
0x00000038	Bne 2	
0x0000003C	Bun um	
0x00000040	Bun	•
0x00000044	Bun erro	
0x00000048	int O	
<u>:</u>	÷	
0x00000100	?	tecla

Sentença switch

<u>:</u>	÷	
0x00000020	18 rl,[Ox103]	
0x00000024	cmpi rl, 48	
0x00000028	Bne 2	
0x0000002C	Bun zero	_
0x00000030	Bun 5	
0x00000034	cmpi rl, 49	
0x00000038	Bne 2	
0x0000003C	Bun um	
0x00000040	Bun	
0x00000044	Bun erro	
0x00000048	int O	
÷	÷	
0x00000100	?	t

tecla

► Sentença **switch**

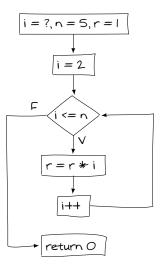
÷ :	i :	
0x00000020	18 rl,[0x103]	
0x00000024	cmpi rl, 48	
0x00000028	Bne 2	
0x0000002C	Bun zero	
0x00000030	Bun 5	
0x00000034	cmpi 11,49	
0x00000038	Bne 2	
0x0000003C	Bun um	
0x00000040	Bun	
0x00000044	Bun erro	
0x00000048	int O	
÷	:	
0x00000100	?	tecla

► Sentença **switch**

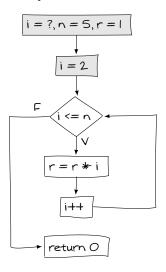
i i	÷	
0x00000020	18 rl, [Ox103]	
0x00000024	cmpi rl, 48	
0x00000028	Bne 2	
0x0000002C	Bun zero	
0x00000030	Bun 5	
0x00000034	cmpi rl, 49	
0x00000038	Bne 2	
0x0000003C	Bun um	
0x00000040	Bun 1	
0x00000044	bun erro	
0x00000048	int O	
<u>:</u>	÷	
0x00000100	?	tecla

```
// Inteiros com tamanho fixo
   #include <stdint.h>
   // Função principal
   int main() {
5
       // Declaração de variáveis
       uint32_t i, n = 5, r = 1;
6
       // Controle iterative for
       for(i = 2; i <= n; i++) {
8
           // r = r * i
10
           r = r * i;
       }
11
       // Retorno sem erros
12
       return 0;
1.3
14
```

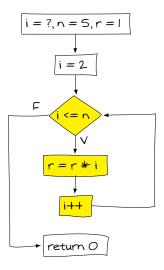
- Sentença for
 - ► Fluxo de execução



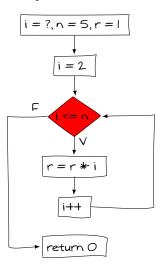
- Sentença for
 - ► Fluxo de execução



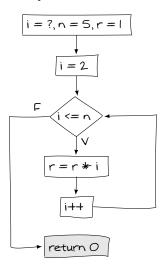
- Sentença for
 - ► Fluxo de execução



- Sentença for
 - ► Fluxo de execução



- Sentença for
 - ► Fluxo de execução



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

i i	į.
0x00000020	mov rl, 2
0x00000024	mov r2,5
0x00000028	mov r3,1
0x0000002C	cmp rl, r2
0x00000030	BGt 3
0x00000034	mul r3, r3, r1
0x00000038	addi rl, rl, l
0x0000003C	виn -5
0x00000040	int O
i i	÷

<u>:</u>	i:
0x00000020	mov rl, 2
0x00000024	mov r2,5
0x00000028	mov r3,1
0x0000002C	cmp rl, r2
0x00000030	BGt 3
0x00000034	mul r3, r3, r1
0x00000038	addi rl, rl, l
0x0000003C	виn -5
0x00000040	int O
i i	i:

i i	i i
0x00000020	mov rl, 2
0x00000024	mov r2,5
0x00000028	mov r3,1
0x0000002C	cmp rl, r2
0x00000030	BGt 3
0x00000034	mul r3, r3, r1
0x00000038	addi rl, rl, l
0x0000003C	Bun -5
0x00000040	int O
i i	÷

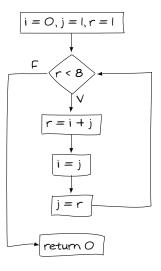
i i	į.
0x00000020	mov rl, 2
0x00000024	mov r2,5
0x00000028	mov r3,1
0x0000002C	cmp rl, r2
0x00000030	BGt 3
0x00000034	mul r3, r3, r1
0x00000038	addi rl, rl, l
0x0000003C	виn -5
0x00000040	int O
i i	÷

÷ :	:
0x00000020	mov rl, 2
0x00000024	mov r2,5
0x00000028	mov r3,1
0x0000002C	cmp rl, r2
0x00000030	BGt 3
0x00000034	mul r3, r3, r1
0x00000038	addi rl, rl, l
0x0000003C	виn -5
0x00000040	int O
i i	<u>:</u>

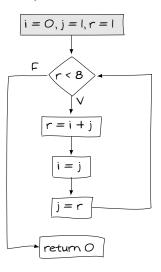
Sentença while

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

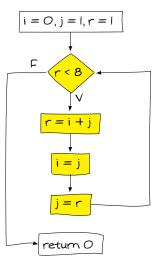
- Sentença while
 - ► Fluxo de execução



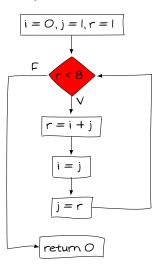
- Sentença while
 - ► Fluxo de execução



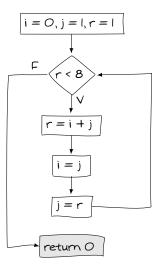
- Sentença while
 - ► Fluxo de execução



- Sentença while
 - ► Fluxo de execução



- Sentença while
 - ► Fluxo de execução



Sentença while

```
// Função principal
   main:
       // r1 = i = 0, r2 = j = 1, r3 = r = 1
3
       mov r1, 0
       mov r2, 1
5
       mov r3, 1
       // r ? 8
       cmpi r3, 8
8
       // r >= 8
       bge 4
10
       // r = i + j
11
       add r3, r1, r2
12
       // i = j, j = r
1.3
14
       mov r1, r2
15
       mov r2, r3
16
       bun - 6
       // Fim
17
       int 0
18
```

► Sentença **while**

i i	i:
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	cmpi r3,8
0x00000030	BGE 4
0x00000034	add r3, r1, r2
0x00000038	mov r1, r2
0x0000003C	mov r2, r3
0x00000040	Bun-6
0x00000044	int O
÷ ·	<u>:</u>

► Sentença **while**

i i	į.
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	cmpi r3,8
0x00000030	BGE 4
0x00000034	add r3, r1, r2
0x00000038	mov rl, r2
0x0000003C	mov r2, r3
0x00000040	Bun-6
0x00000044	int O
<u>:</u>	<u>:</u>

Sentença while

<u>:</u>	i i
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	cmpi r3,8
0x00000030	BGE 4
0x00000034	add r3, r1, r2
0x00000038	mov rl, r2
0x0000003C	mov r2, r3
0x00000040	Bun-6
0x00000044	int O
i i	÷

Sentença while

i i	i i
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	cmpi r3,8
0x00000030	BGE 4
0x00000034	add r3, r1, r2
0x00000038	mov rl, r2
0x0000003C	mov r2, r3
0x00000040	Bun-6
0x00000044	int O
i i	<u>:</u>

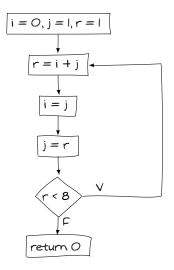
Sentença while

<u>:</u>	i i
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	cmpi r3,8
0x00000030	BGE 4
0x00000034	add r3, r1, r2
0x00000038	mov r1, r2
0x0000003C	mov r2, r3
0x00000040	Bun-6
0x00000044	int O
:	÷

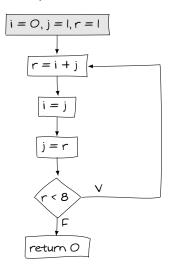
Sentença do while

```
// Inteiros com tamanho fixo
   #include <stdint.h>
   // Função principal
   int main() {
       // Declaração de variáveis
5
       uint32_t i = 0, j = 1, r = 1;
       // Controle iterativo do while
       do {
8
           // r = i + j
           r = i + j
10
            // i = j, j = r
11
12
            i = j;
            i = r
1.3
14
15
       while (r < 8);
16
       // Retorno sem erros
17
       return 0;
18
```

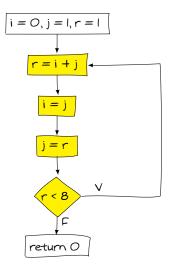
- Sentença do while
 - ► Fluxo de execução



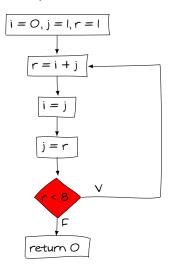
- Sentença do while
 - ► Fluxo de execução



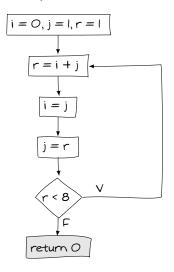
- Sentença do while
 - ► Fluxo de execução



- Sentença do while
 - ► Fluxo de execução



- Sentença do while
 - ► Fluxo de execução



Sentença do while

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

► Sentença **do while**

:	:
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	add r3, r1, r2
0x00000030	mov r1, r2
0x00000034	mov r2, r3
0x00000038	cmpi r3,8
0x0000003C	Blt -5
0x00000040	int O
i i	i:

► Sentença **do while**

:	:
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	add r3, r1, r2
0x00000030	mov rl, r2
0x00000034	mov r2, r3
0x00000038	cmpi r3,8
0x0000003C	Blt -5
0x00000040	int O
i i	<u>:</u>

Sentença do while

i i	i i
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	add r3, r1, r2
0x00000030	mov r1, r2
0x00000034	mov r2, r3
0x00000038	cmpi r3,8
0x0000003C	Blt -5
0x00000040	int O
i i	÷

► Sentença **do while**

:	:
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	add r3, r1, r2
0x00000030	mov r1, r2
0x00000034	mov r2, r3
0x00000038	cmpi r3,8
0x0000003C	Blt -5
0x00000040	int O
i i	<u>:</u>

► Sentença **do while**

i i	i i
0x00000020	mov rl, O
0x00000024	mov r2,1
0x00000028	mov r3,1
0x0000002C	add r3, r1, r2
0x00000030	mov r1, r2
0x00000034	mov r2, r3
0x00000038	cmpi r3,8
0x0000003C	Blt -5
0x00000040	int O
:	:

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

- 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

- 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

- 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
 - Retorno ao fluxo anterior

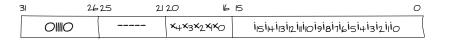
- Qual a motivação para utilização de sub-rotina?
 - Depuração de comportamento
 - Modularização e reuso de software
 - Realização de tarefas específicas
- Etapas na execução de uma sub-rotina
 - 1. Preparação dos argumentos
 - 2. Chamada ou invocação
 - 3. Execução da sub-rotina
 - O comportamento descrito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

- 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
 - 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

- Operação de chamada de sub-rotina (call)
 - ► Tipo F

$$ightharpoonup MEM[SP] = PC + 4, SP = SP - 4$$

$$PC = \left(R[x] + i_{15}^{16} : i \right) \ll 2$$

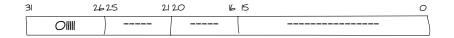


- Operação de chamada de sub-rotina (call)
 - ► Tipo S

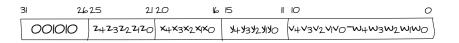
$$\blacktriangleright$$
 MEM[SP] = PC + 4, SP = SP - 4

$$PC = PC + 4 + \left[\begin{pmatrix} 6 \\ i_{25} : i \end{pmatrix} \ll 2 \right]$$

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



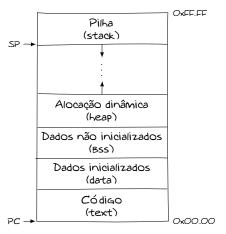
- Operação de empilhamento (push)
 - ► Tipo U
 - i = V, w, x, y, z
 - $i \neq 0 \rightarrow MEM[SP] = R[i], SP = SP 4$



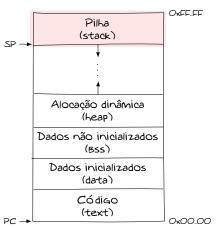
- Operação de desempilhamento (pop)
 - Tipo U
 - i = V, w, x, y, z
 - $i \neq 0 \rightarrow SP = SP + 4, R[i] = MEM[SP]$



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



- 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

Implementação da função fatorial recursiva

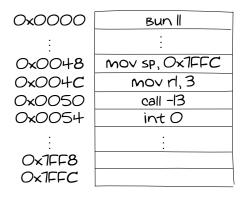
```
// Inteiros com tamanho fixo
  #include <stdint.h>
   // Função fatorial
   uint32_t fatorial(uint32_t n) {
       // Caso base
      if(n == 0) return 1;
       // Caso recursivo
      else return n * fatorial(n - 1);
10
   // Função principal
   int main() {
11
      // fatorial(3)
12
13
     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

Implementação da função fatorial recursiva

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

Implementação da função fatorial recursiva

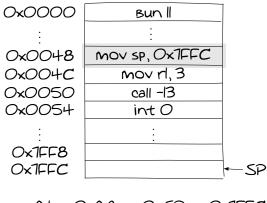
```
// Função fatorial
   fatorial:
   // Função principal
16
   main:
17
       // SP = 32 KiB
18
       mov sp, 0x7FFC
19
       // fatorial(3)
20
21
       mov r1, 3
22
       call fatorial
       // Fim
23
       int 0
24
```



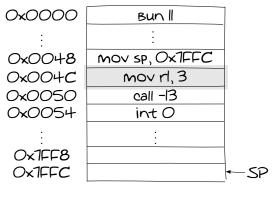
$$RI = 0, R2 = 0, SP = 0$$

0x0000	Bun II
:	:
0x0048	MOV SP, OXTFFC
0x004C	mov rl, 3
0x0050	call -l3
0x0054	int O
:	i i
0x1FF8	
OXTFFC	

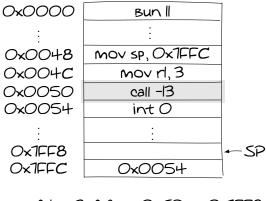
$$RI = O, R2 = O, SP = O$$



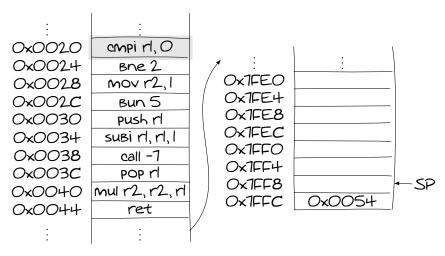
$$RI = 0$$
, $R2 = 0$, $SP = 0$ x 1 FFC



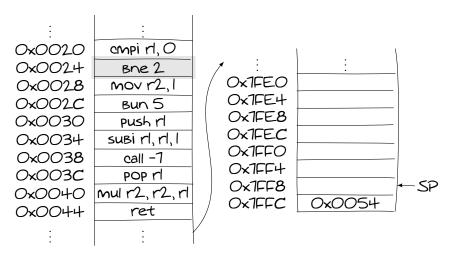
$$RI = 3$$
, $R2 = 0$, $SP = 0$ ×1FFC



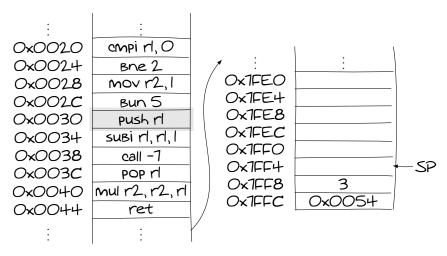
$$RI = 3$$
, $R2 = 0$, $SP = 0$ x1FF8



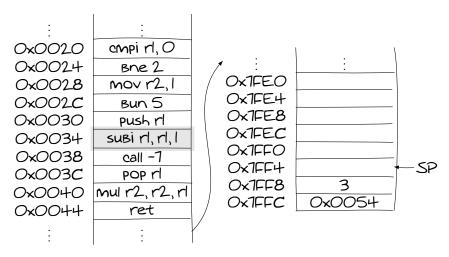
$$RI = 3$$
, $R2 = 0$, $SP = 0$ x1FF8



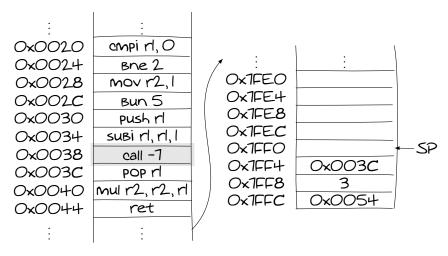
$$RI = 3$$
, $R2 = 0$, $SP = 0$ x1FF8



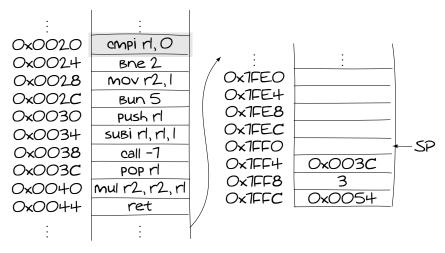
$$RI = 3$$
, $R2 = 0$, $SP = 0$ x 1 FF $+$



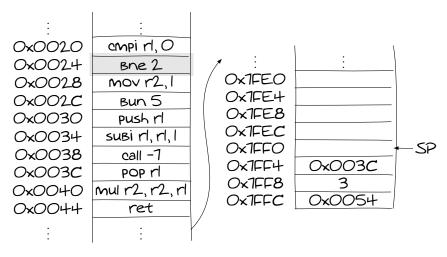
$$RI = 2$$
, $R2 = 0$, $SP = 0$ x $TFF+$



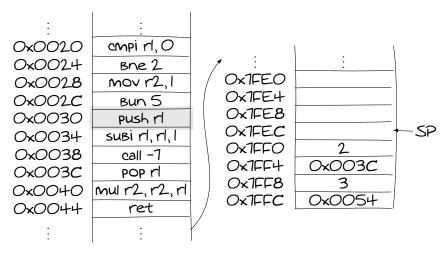
$$RI = 2$$
, $R2 = 0$, $SP = 0$ ×1FF0



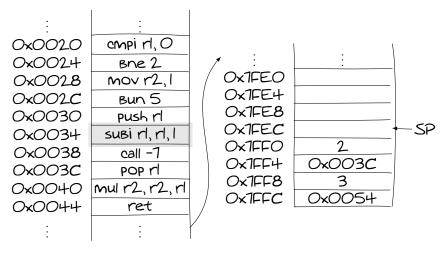
$$RI = 2$$
, $R2 = 0$, $SP = 0$ ×1FF0



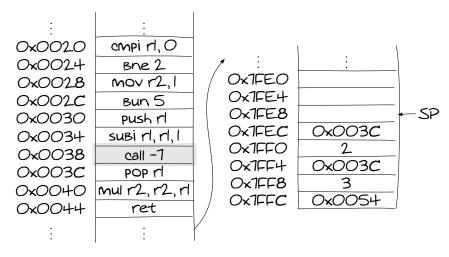
$$RI = 2$$
, $R2 = 0$, $SP = 0$ ×1FF0



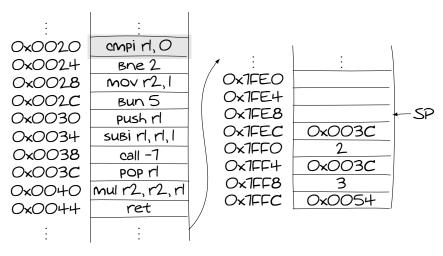
$$RI = 2$$
, $R2 = 0$, $SP = OXTFEC$



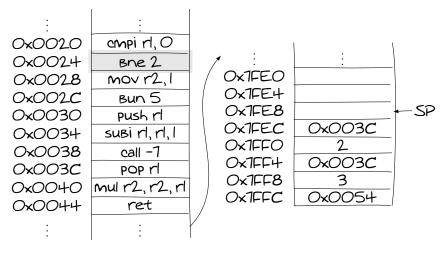
$$RI = I, R2 = O, SP = OxTFEC$$



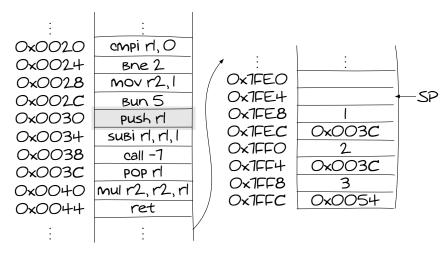
$$RI = I, R2 = O, SP = Ox1FE8$$



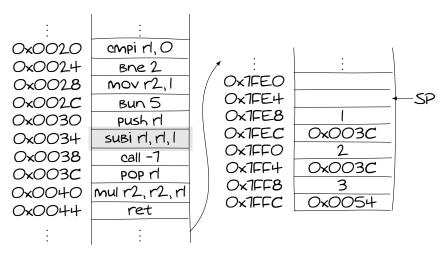
$$RI = I, R2 = O, SP = Ox1FE8$$



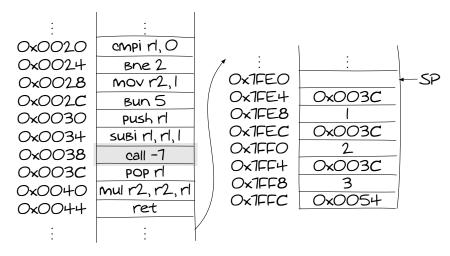
$$RI = I, R2 = O, SP = Ox1FE8$$



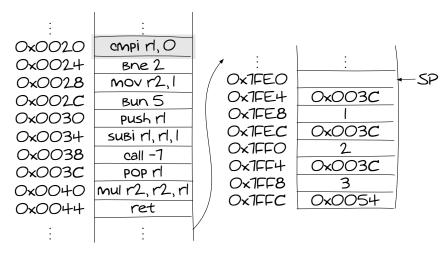
$$RI = 1$$
, $R2 = 0$, $SP = OXTFEH$



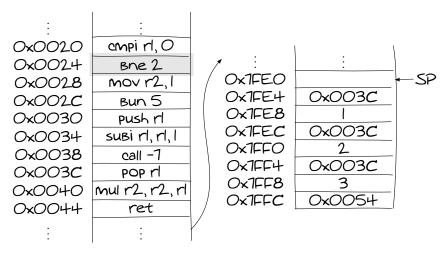
$$RI = O$$
, $R2 = O$, $SP = Ox1FE+$



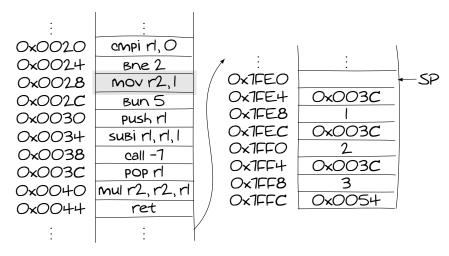
$$RI = 0$$
, $R2 = 0$, $SP = 0$ XTFEO



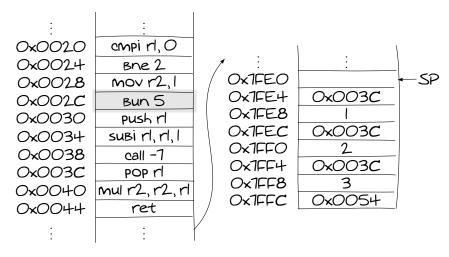
$$RI = 0$$
, $R2 = 0$, $SP = 0$ XTFEO



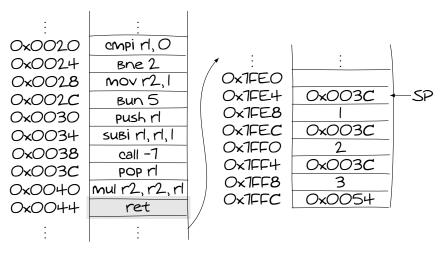
$$RI = 0$$
, $R2 = 0$, $SP = 0$ XTFEO



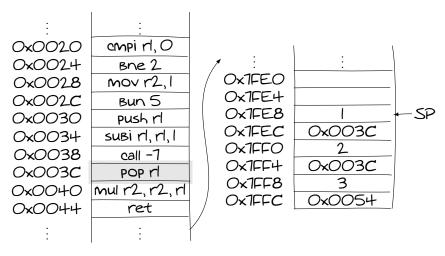
$$RI = 0$$
, $R2 = I$, $SP = Ox1FEO$



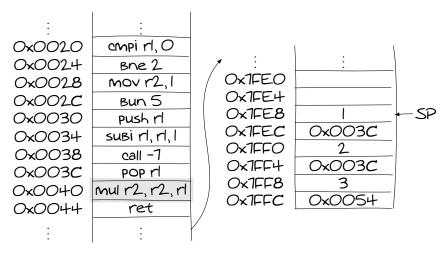
$$RI = 0$$
, $R2 = I$, $SP = Ox1FEO$



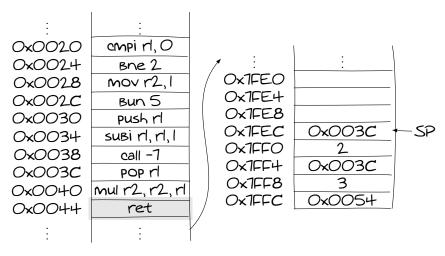
$$RI = O$$
, $R2 = I$, $SP = Ox1FEH$



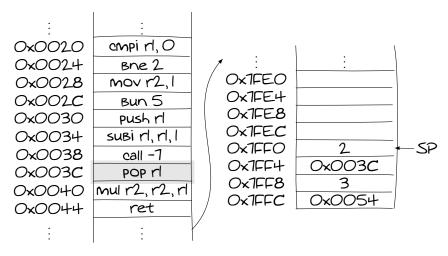
$$RI = I, R2 = I, SP = OXTFE8$$



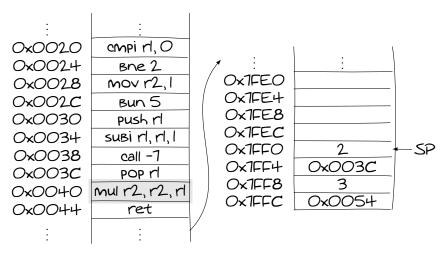
$$RI = I$$
, $R2 = I$, $SP = Ox1FE8$



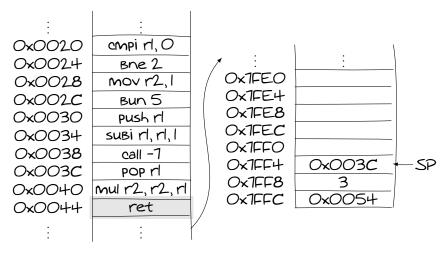
$$RI = I, R2 = I, SP = Ox1FEC$$



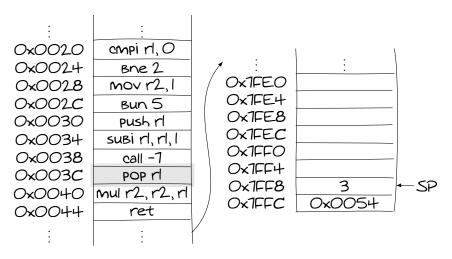
$$RI = 2$$
, $R2 = I$, $SP = Ox1FFO$



$$RI = 2$$
, $R2 = 2$, $SP = Ox1FFO$

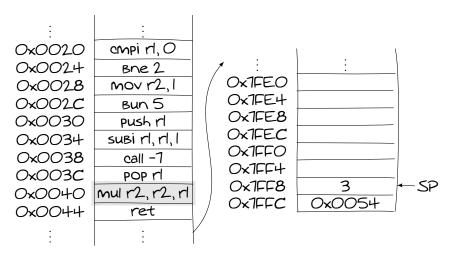


$$RI = 2$$
, $R2 = 2$, $SP = 0$ ×1FF4

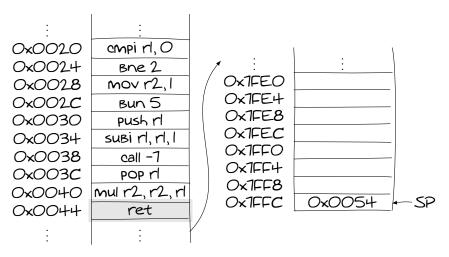


$$RI = 3, R2 = 2, SP = 0x1FF8$$

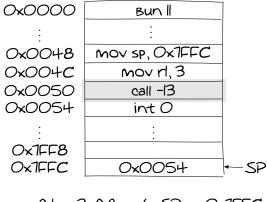
Execução da função fatorial recursiva



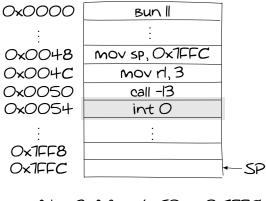
RI = 3, R2 = 6, SP = 0x1FF8



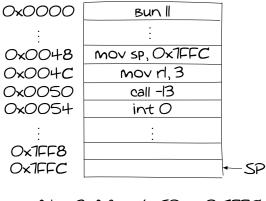
$$RI = 3$$
, $R2 = 6$, $SP = Ox1FFC$



$$RI = 3$$
, $R2 = 6$, $SP = Ox7FFC$



$$RI = 3$$
, $R2 = 6$, $SP = Ox1FFC$



$$RI = 3$$
, $R2 = 6$, $SP = Ox1FFC$

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

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