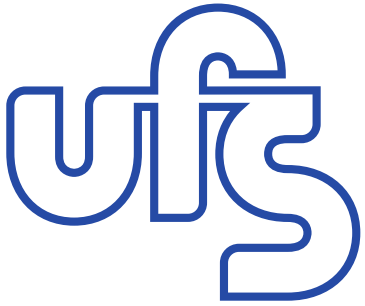
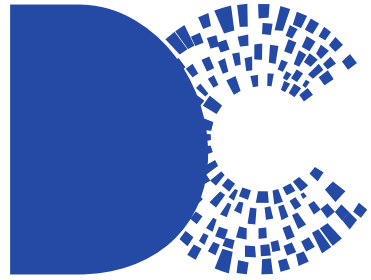




Controle de fluxo

Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

Introdução

- O que é controle de fluxo?
  - Define qual sequência de instruções serão executadas pelo processador através da manipulação do PC

Introdução

- O que é controle de fluxo?
  - Define qual sequência de instruções serão executadas pelo processador através da manipulação do PC
    - Condicional

Introdução

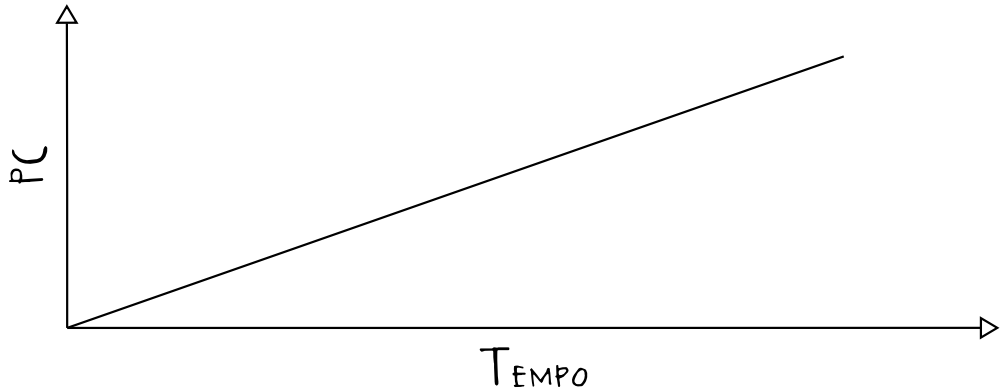
- O que é controle de fluxo?
  - Define qual sequência de instruções serão executadas pelo processador através da manipulação do PC
    - Condicional
    - Iterativo

Introdução

- O que é controle de fluxo?
  - Define qual sequência de instruções serão executadas pelo processador através da manipulação do PC
    - Condicional
    - Iterativo
    - Funções ou procedimentos

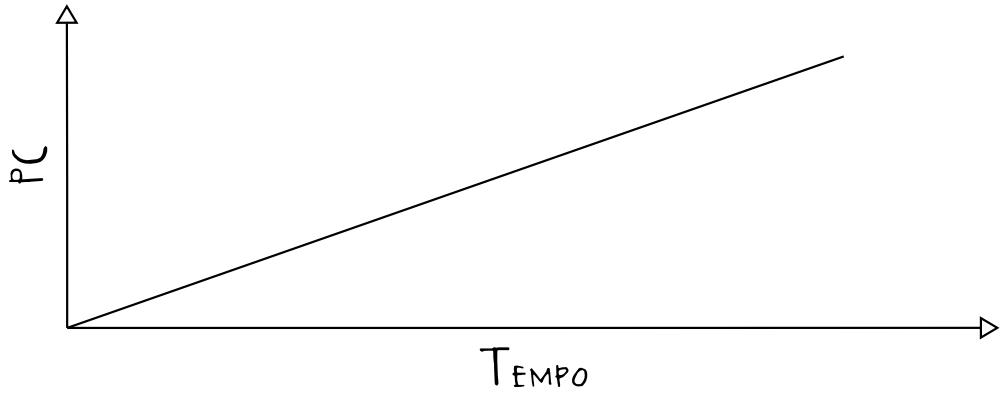
Introdução

- Controle do fluxo de execução (PC)
  - Sequencial (sem desvios)

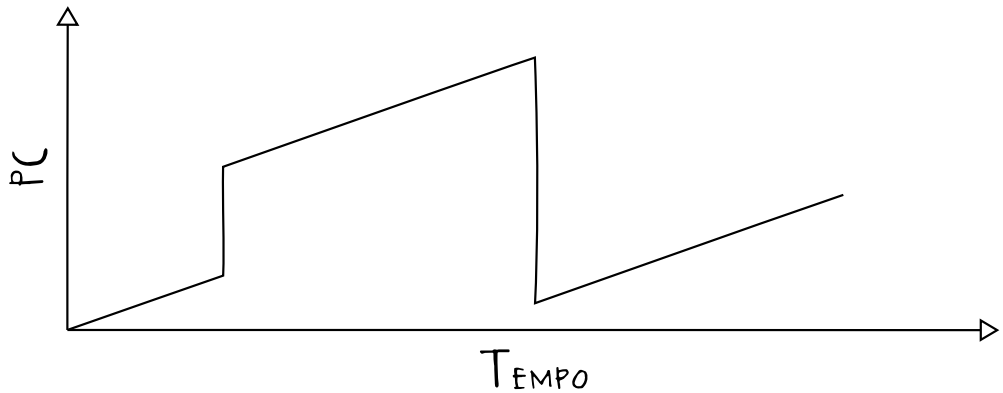


Introdução

- Controle do fluxo de execução (PC)
  - Sequencial (sem desvios)



  - Não sequencial (com desvios)



Introdução

- Controle de fluxo no hardware (unidade de controle)
  - É feita a coordenação dos componentes do sistema, definindo a ordem e a temporização dos eventos, onde cada instrução é convertida em uma sequência de ações em uma máquina de estados finita (FSM)

Introdução

- Controle de fluxo no hardware (unidade de controle)
  - É feita a coordenação dos componentes do sistema, definindo a ordem e a temporização dos eventos, onde cada instrução é convertida em uma sequência de ações em uma máquina de estados finita (FSM)
  - *Hardwired*
    - Utiliza lógica booleana para gerar os sinais de controle
    - Tem alto desempenho, mas não permite atualizações

Introdução

- Controle de fluxo no hardware (unidade de controle)
  - É feita a coordenação dos componentes do sistema, definindo a ordem e a temporização dos eventos, onde cada instrução é convertida em uma sequência de ações em uma máquina de estados finita (FSM)
  - *Hardwired*
    - Utiliza lógica booleana para gerar os sinais de controle
    - Tem alto desempenho, mas não permite atualizações
  - Micro-programada
    - Implementado por micro-instruções reprogramáveis
    - Flexibilidade na modificação do projeto de controle

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*, *if else if else*, *switch*
  - Iterativo: *for*, *while*
  - Funções e procedimentos

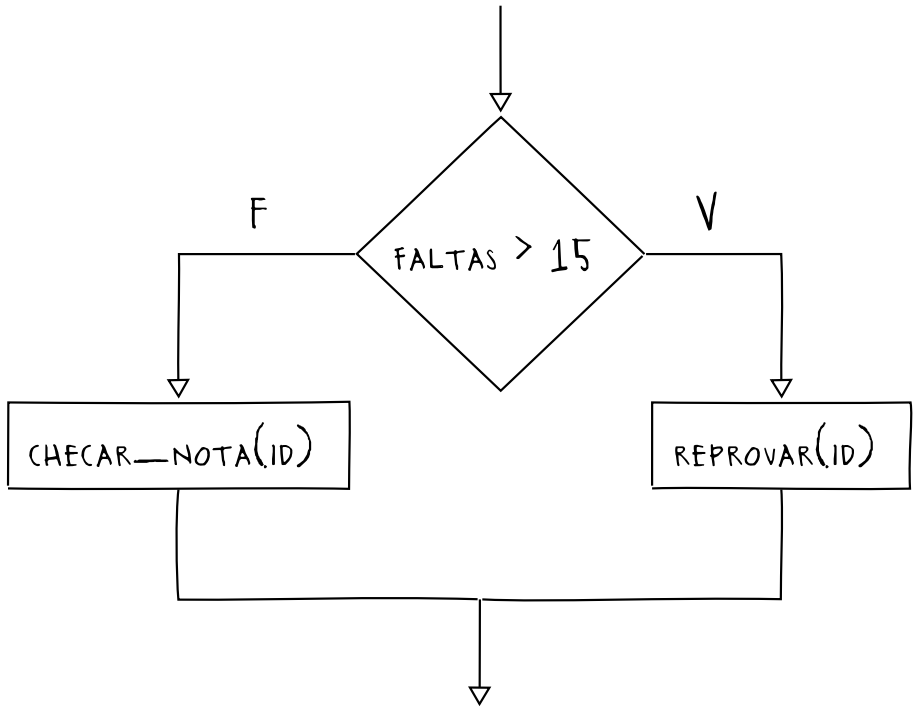
Controle condicional

- *if else*

```
1   // Inteiros com tamanho fixo
2   #include <stdint.h>
3   // Biblioteca padrão
4   #include <stdlib.h>
5   // Procedimento DAA
6   void DAA(uint32_t id, uint8_t faltas) {
7       // (faltas > 15 horas) -> reprovar
8       if(faltas > 15) reprovar(id);
9       // (faltas <= 15) -> checar nota
10      else checar_nota(id);
11  }
…   ...
```

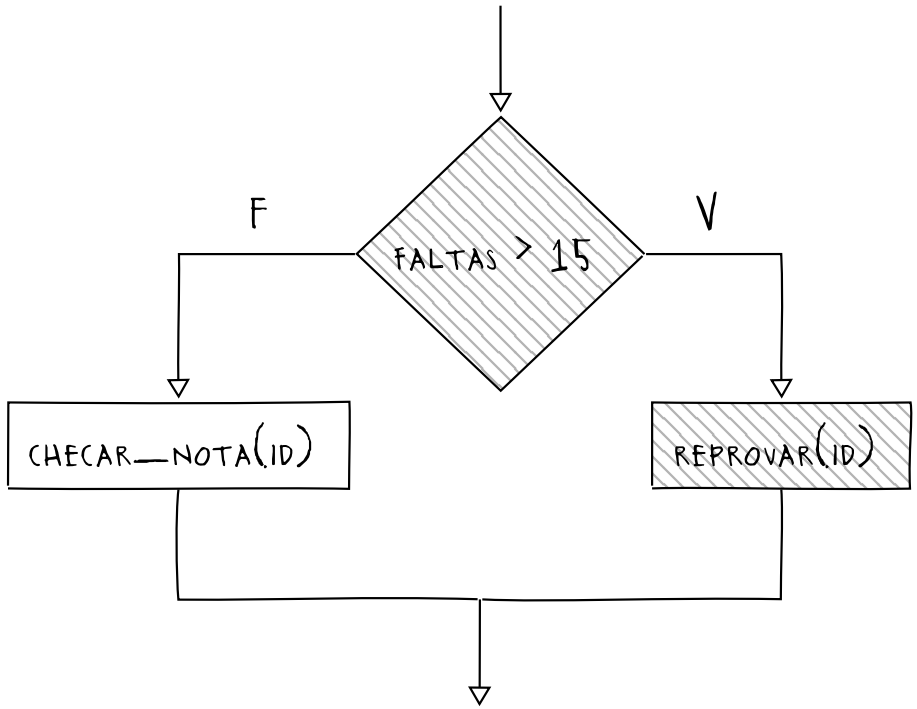
Controle condicional

- *if else*



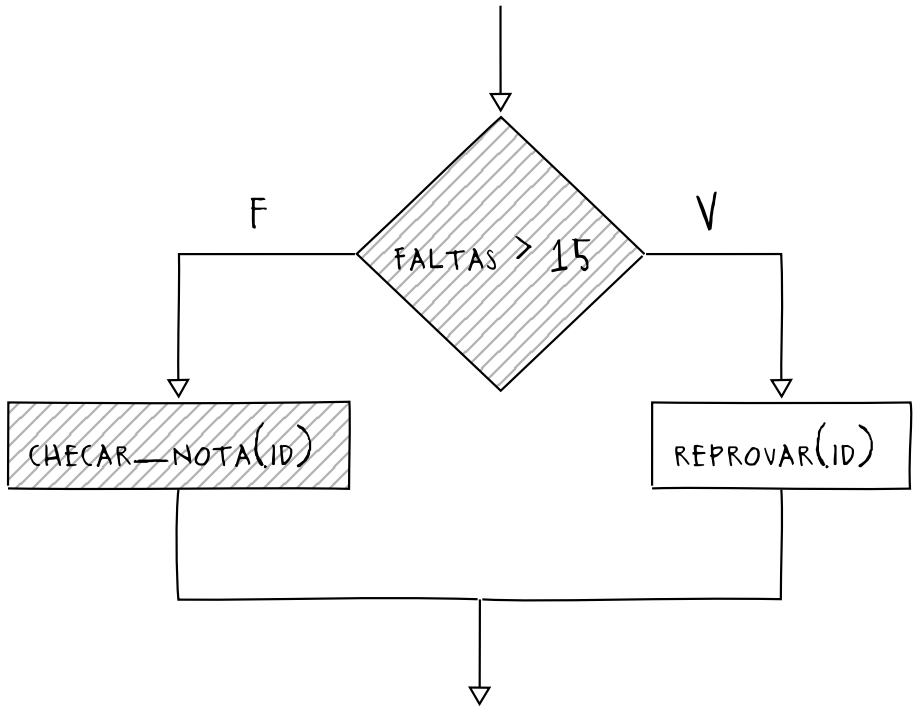
Controle condicional

- *if else*



Controle condicional

- *if else*



Controle condicional

- *if else*

```
25  # DAA(a0 = id, a1 = faltas)
26  DAA:
27      # t0 = 15
28      li t0, 15
29      # (faltas > 15) -> reprovar
30      bgt a1, t0, reprovar
31      # (faltas <= 15) -> checar_nota
32      checar_nota:
...         ...
45      reprovar:
...         ...
55      # end
56      ret
```

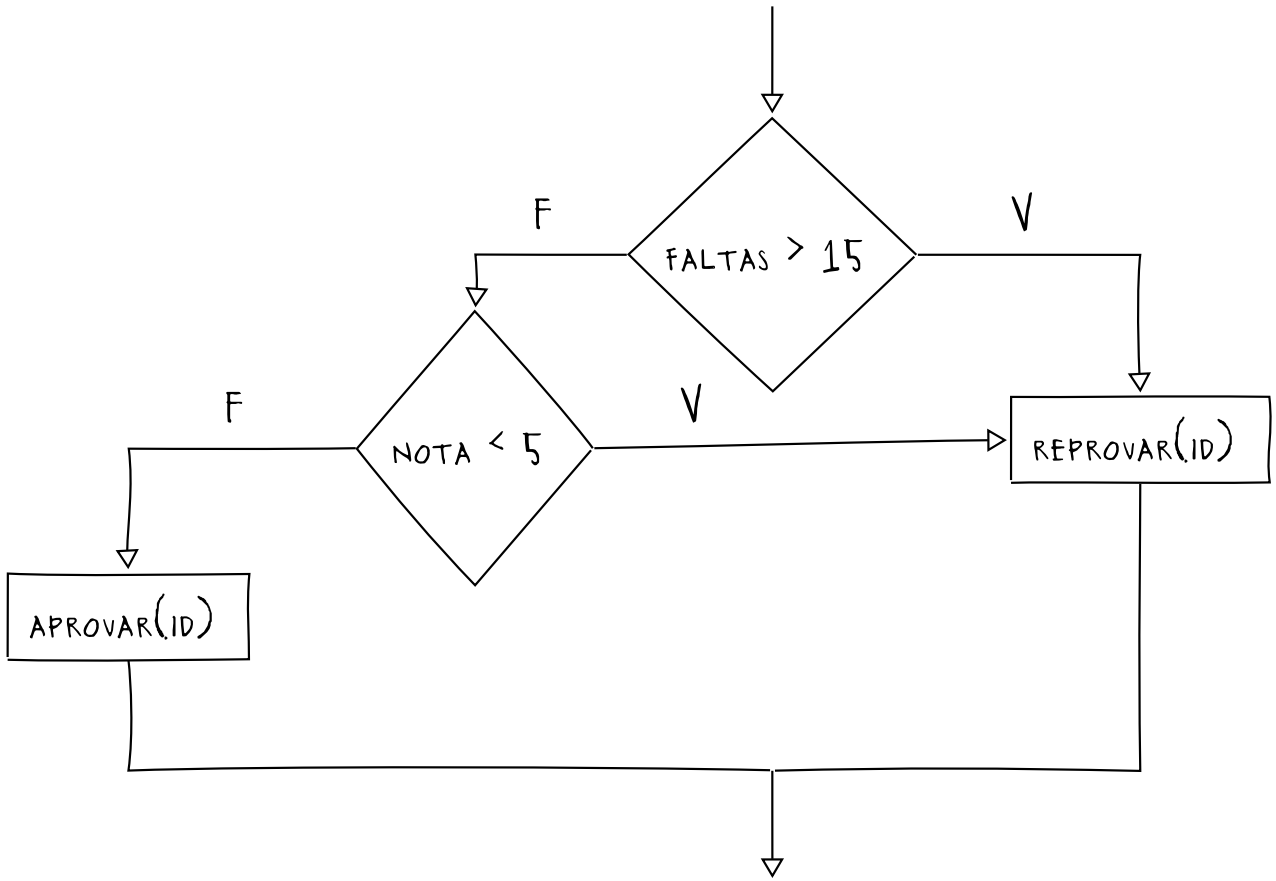
Controle condicional

- *if else if else*

```
1   // Inteiros com tamanho fixo
2   #include <stdint.h>
3   // Biblioteca padrão
4   #include <stdlib.h>
5   // Procedimento DAA
6   void DAA(uint32_t id, uint8_t faltas, uint8_t nota) {
7       // (faltas > 15) -> reprovar
8       if(faltas > 15) reprovar(id);
9       // (faltas <= 15 && nota < 5) -> reprovar
10      else if(nota < 5) reprovar(id);
11      // (faltas <= 15 && nota >= 5) -> aprovar
12      else aprovar(id);
13  }
…   ...
```

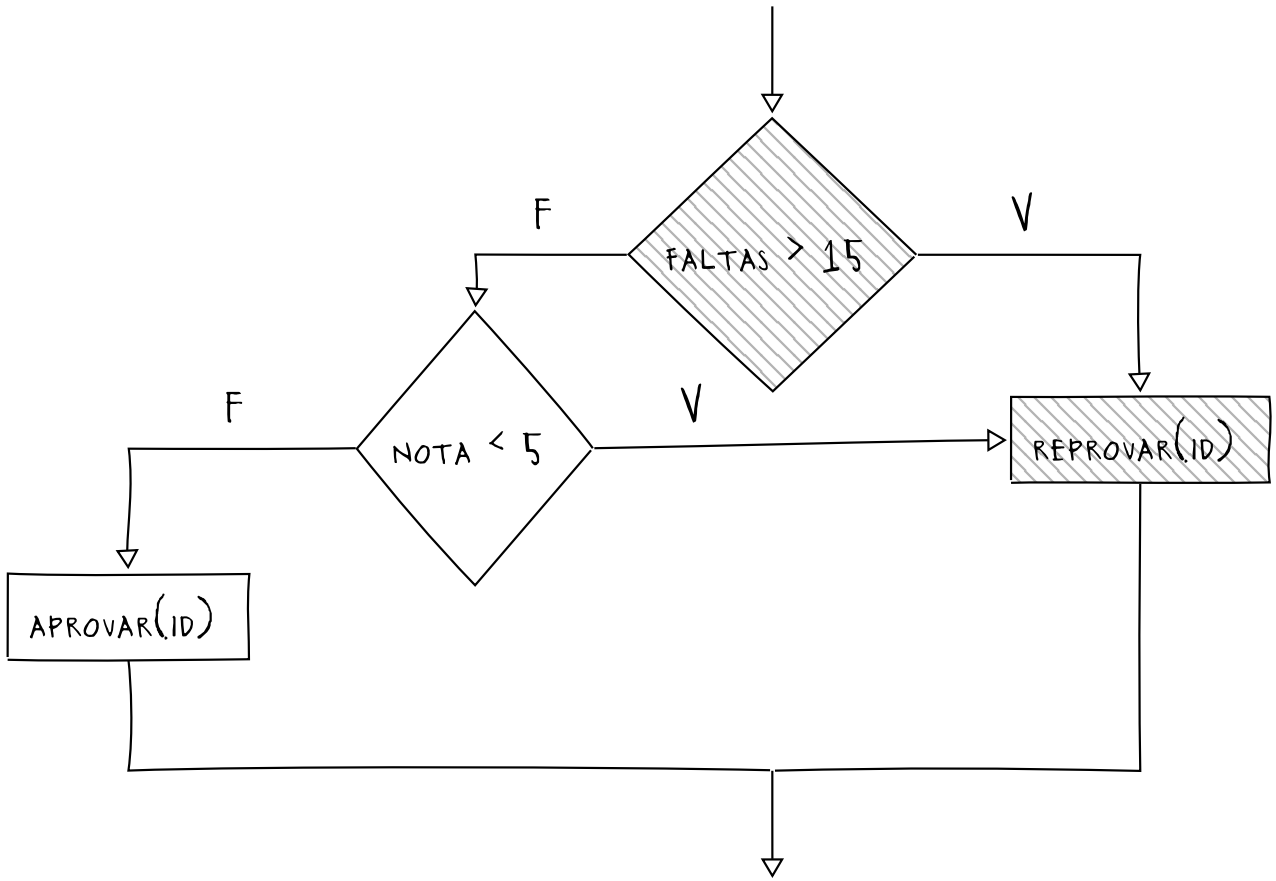
Controle condicional

- *if else if else*



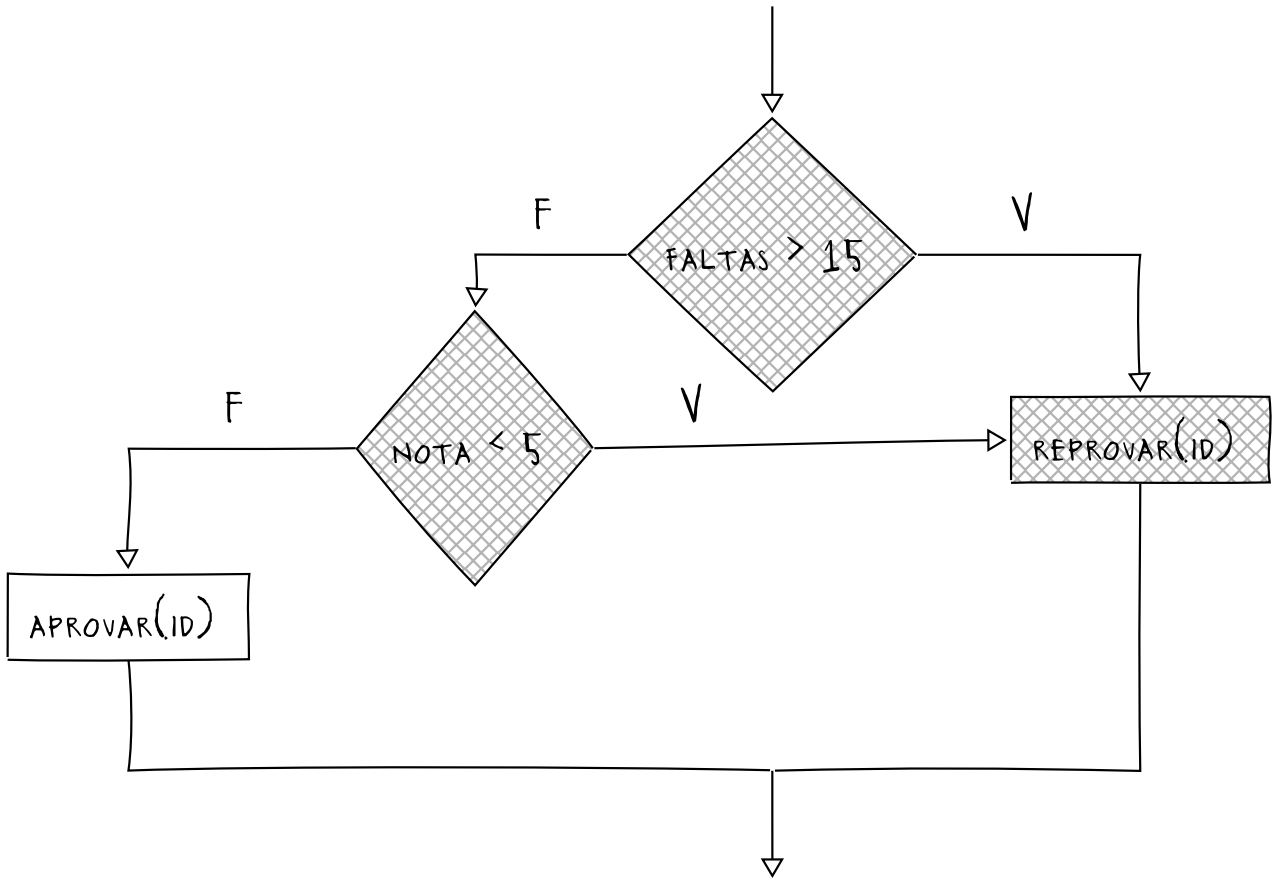
Controle condicional

- *if else if else*



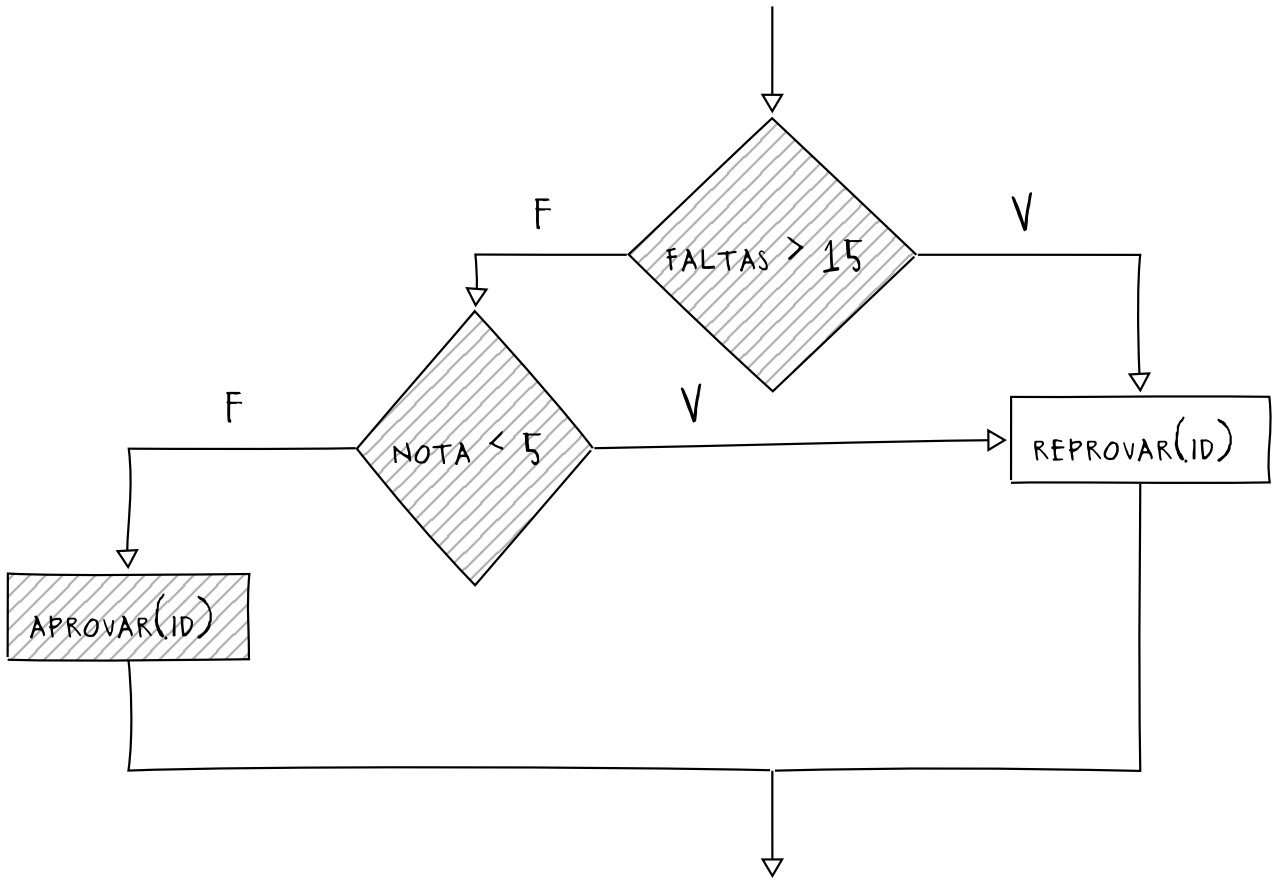
Controle condicional

- *if else if else*



Controle condicional

- *if else if else*



Controle condicional

- *if else if else*

```
25  # DAA(a0 = id, a1 = faltas, a2 = nota)
26  DAA:
27      # t0 = 15, t1 = 5
28      li t0, 15
29      li t1, 5
30      # (faltas > 15) -> reprovar
31      bgt a1, t0, reprovar
32      # (faltas <= 15 && nota < 5) -> reprovar
33      blt a2, t1, reprovar
34      # (faltas <= 15 && nota >= 5) -> aprovar
35      aprovar:
...         ...
47      reprovar:
...         ...
57      # end
58      ret
```

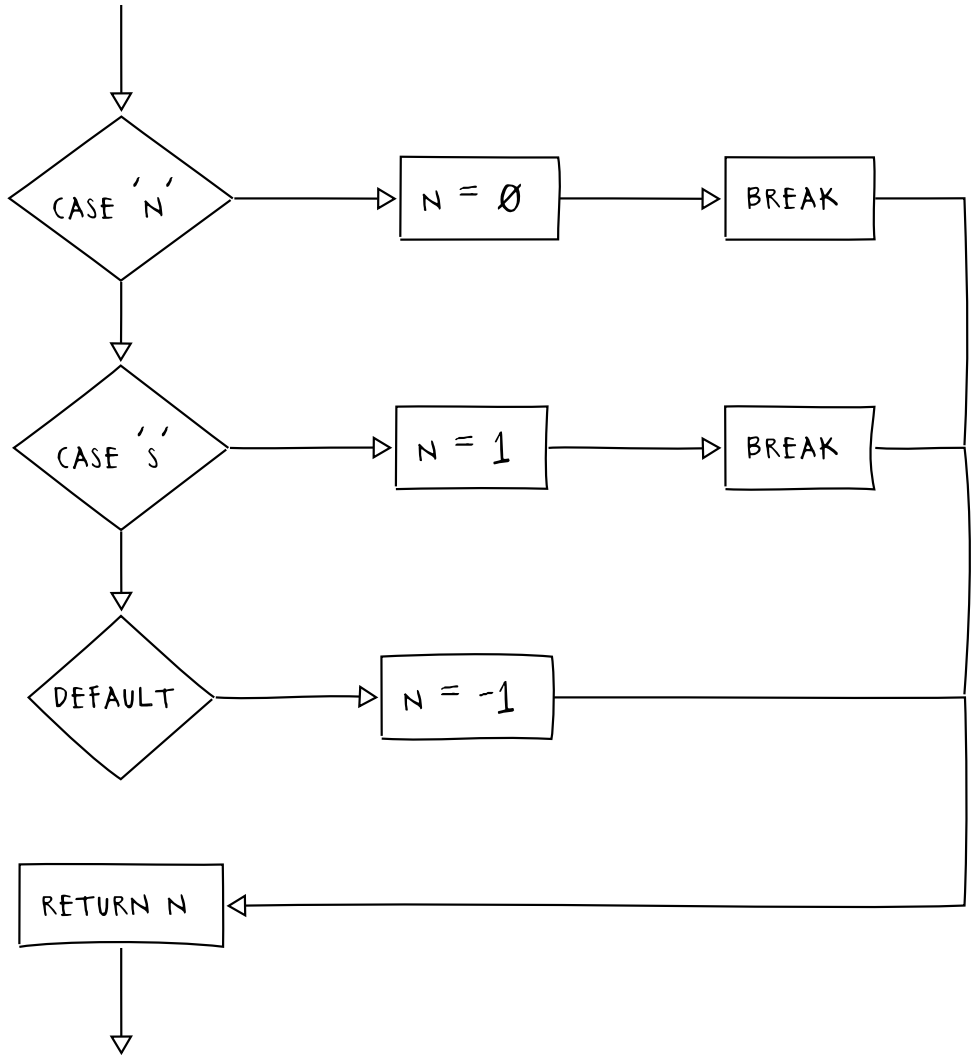
Controle condicional

- *switch*

```
1   // Inteiros com tamanho fixo
2   #include <stdint.h>
3   // Biblioteca padrão
4   #include <stdlib.h>
5   // Função escolha
6   uint8_t escolha(char tecla) {
7       // Valor numérico
8       uint8_t n;
9       // Checando tecla
10      switch(tecla) {
11          // Valores esperados (n, s)
12          case 'n': n = 0; break;
13          case 's': n = 1; break;
14          default: n = -1;
15      }
16      // Retornando valor
17      return n;
18  }
…   ...
```

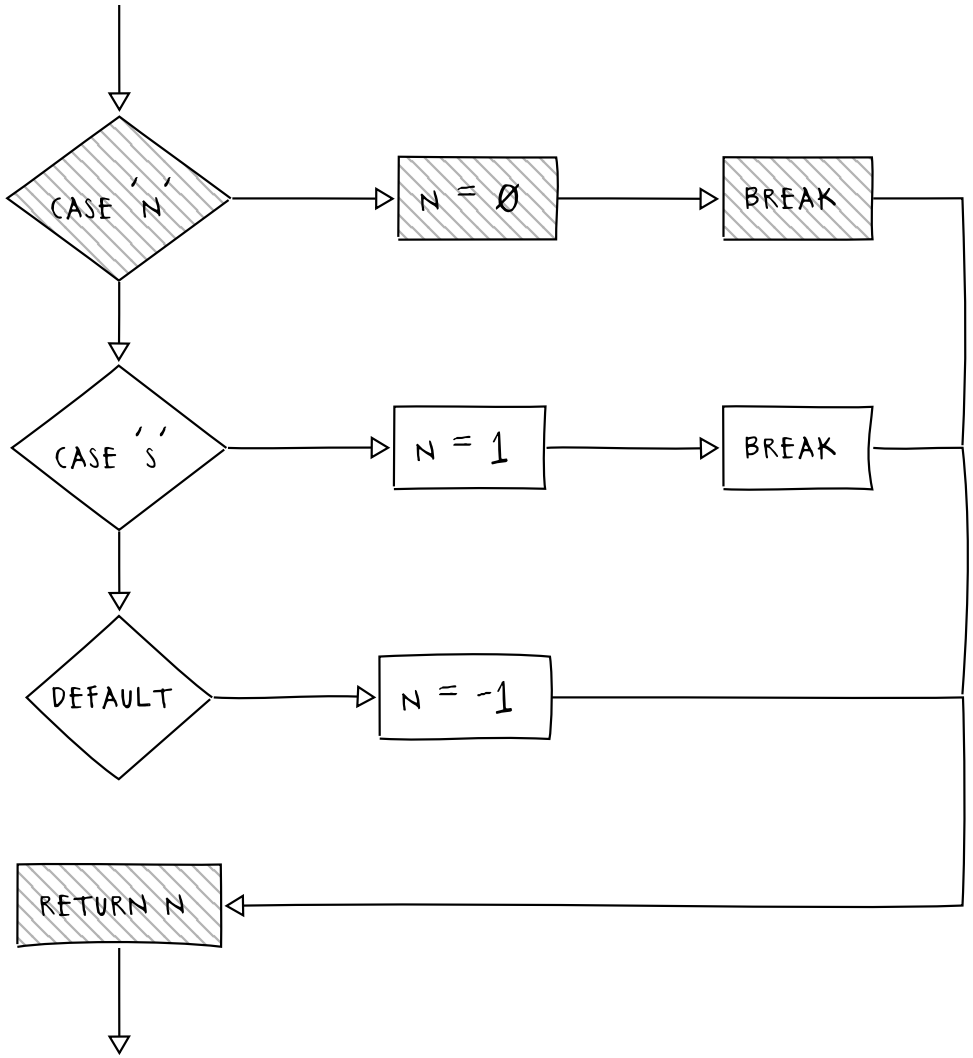
Controle condicional

- *switch*



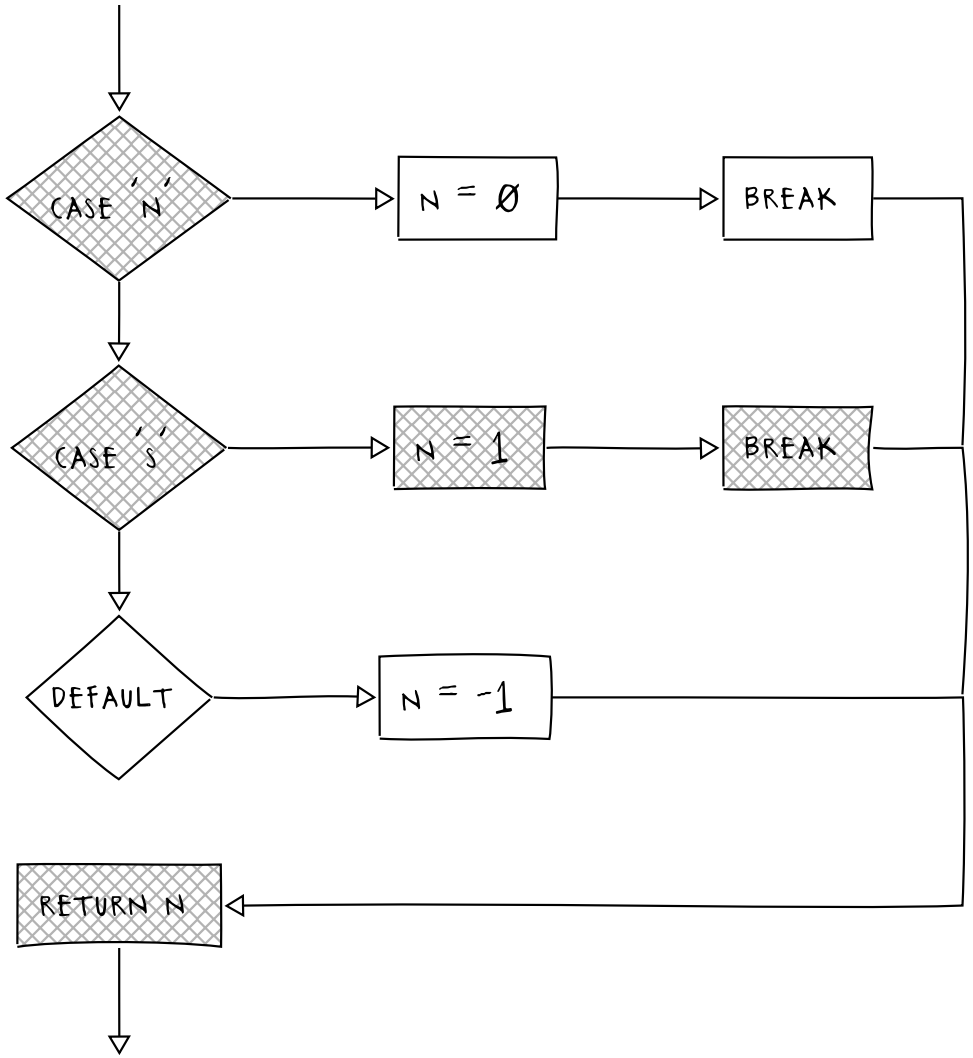
Controle condicional

- *switch*



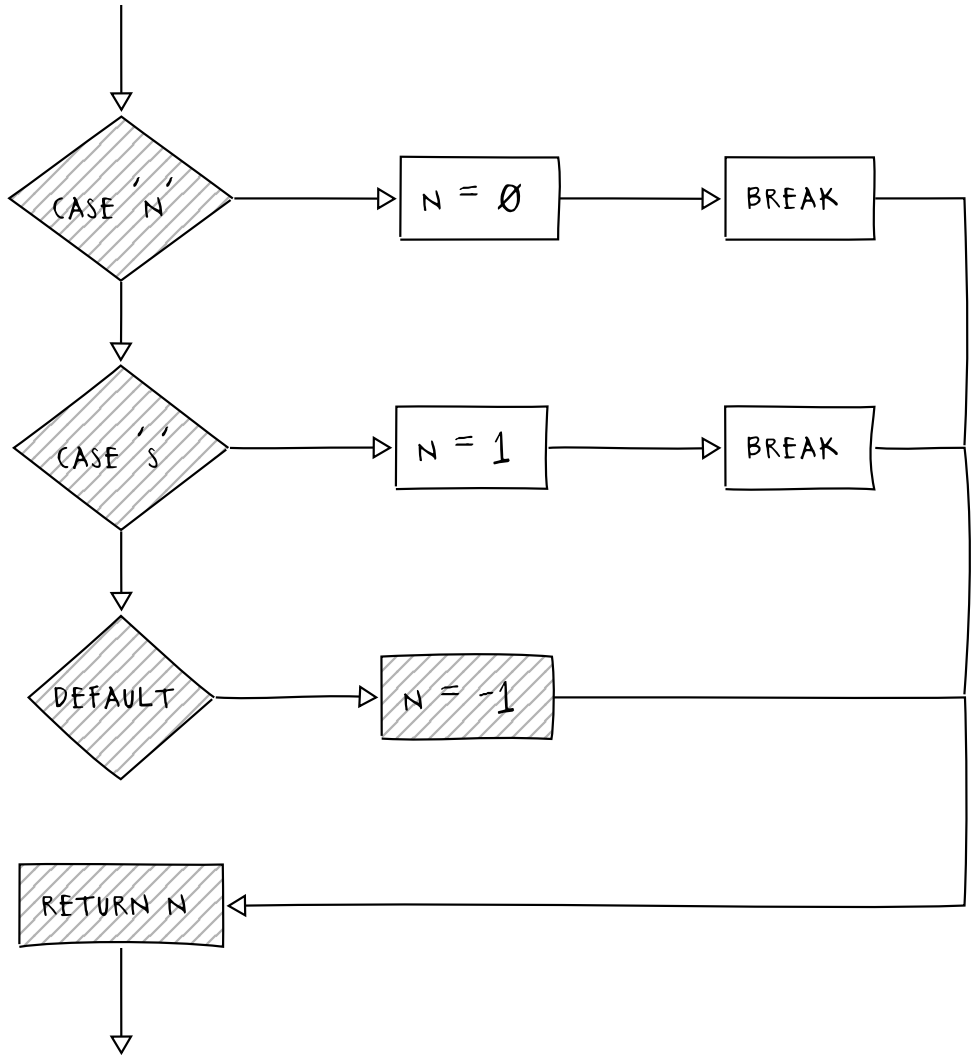
Controle condicional

- *switch*



Controle condicional

- *switch*



Controle condicional

- *switch*

```
25  # escolha(a0 = tecla)
26  escolha:
27      init: # t0 = 'n', t1 = 's'
28          li t0, 110
29          li t1, 115
30      case_0: # tecla == 'n'
31          bne a0, t0, case_1
32          mv a0, zero
33          j end
34      case_1: # tecla == 's'
35          bne a0, t1, case_d
36          li a0, 1
37          j end
38      case_d: # default
39          li a0, -1
40      end:
41          ret
```

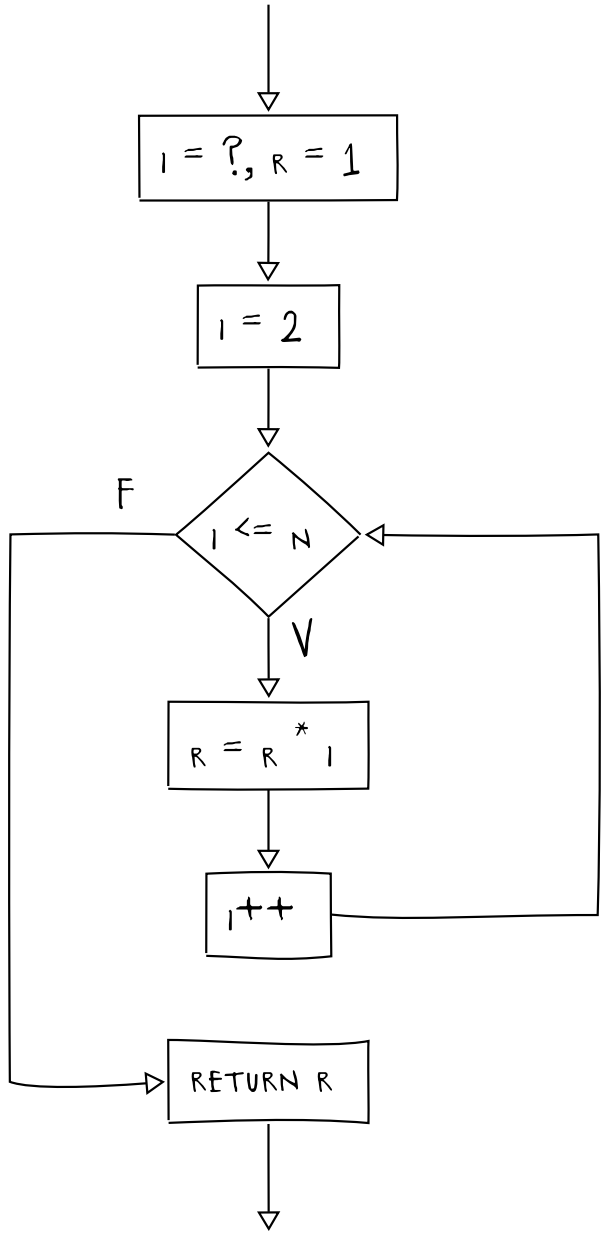
Controle iterativo

- *for*

```
1   // Inteiros com tamanho fixo
2   #include <stdint.h>
3   // Biblioteca padrão
4   #include <stdlib.h>
5   // Função fatorial
6   uint32_t fatorial(uint32_t n) {
7       // Declaração de variáveis
8       uint32_t i, r = 1;
9       // Repete enquanto i <= n
10      for(i = 2; i <= n; i++) {
11          // r = r * i
12          r = r * i;
13      }
14      // Retornando valor
15      return r;
16  }
…   ...
```

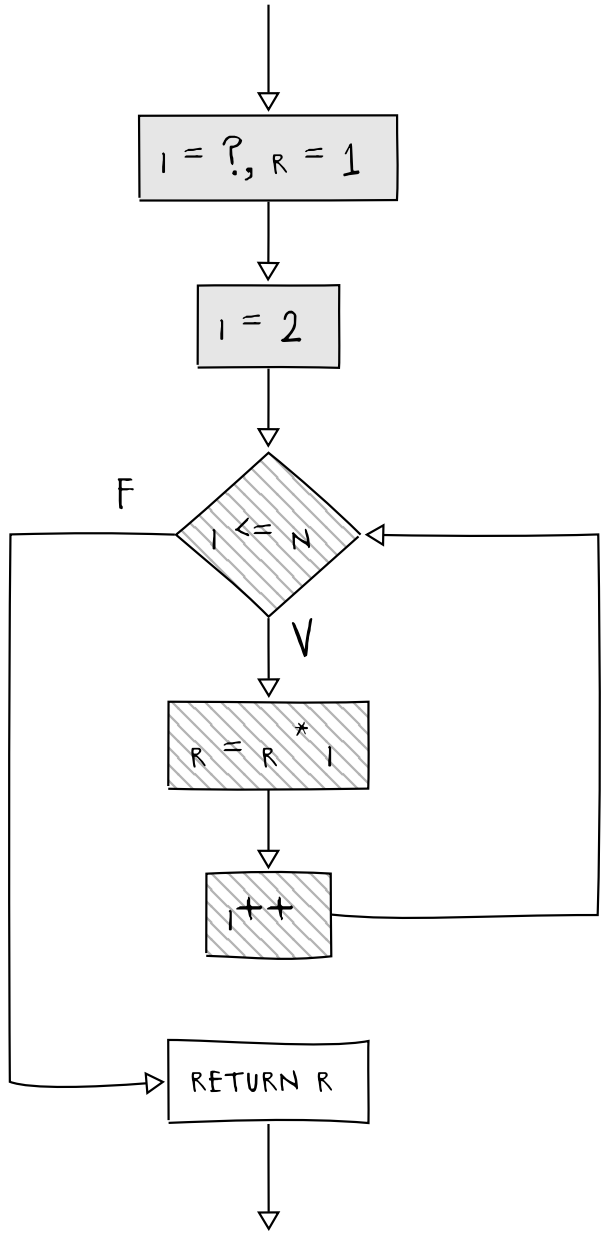
Controle iterativo

- *for*



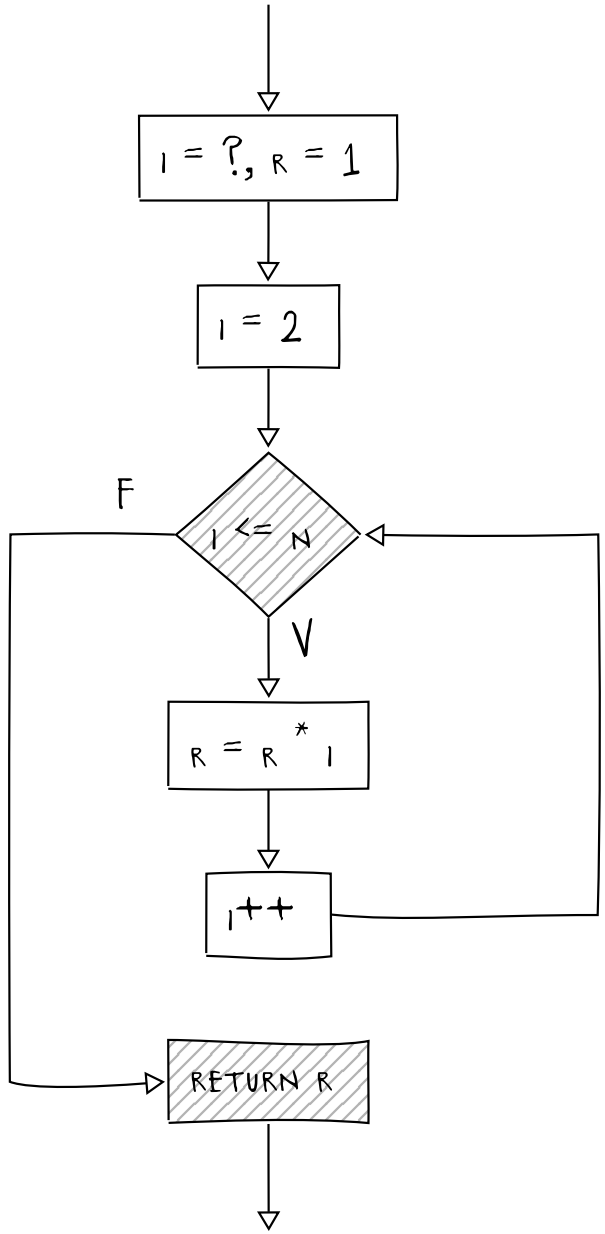
Controle iterativo

- *for*



Controle iterativo

- *for*



Controle iterativo

- *for*

```
25  # fatorial(a0 = n)
26  fatorial:
27      # t0 = i = 2, t1 = r = 1
28      init:
29          li t0, 2
30          li t1, 1
31      # i <= n
32      iteration:
33          bgt t0, a0, end
34          # r = r * i, i++
35          mul t1, t1, t0
36          addi t0, t0, 1
37          j iteration
38      # end
39      end:
40          mv a0, t1
41          ret
```

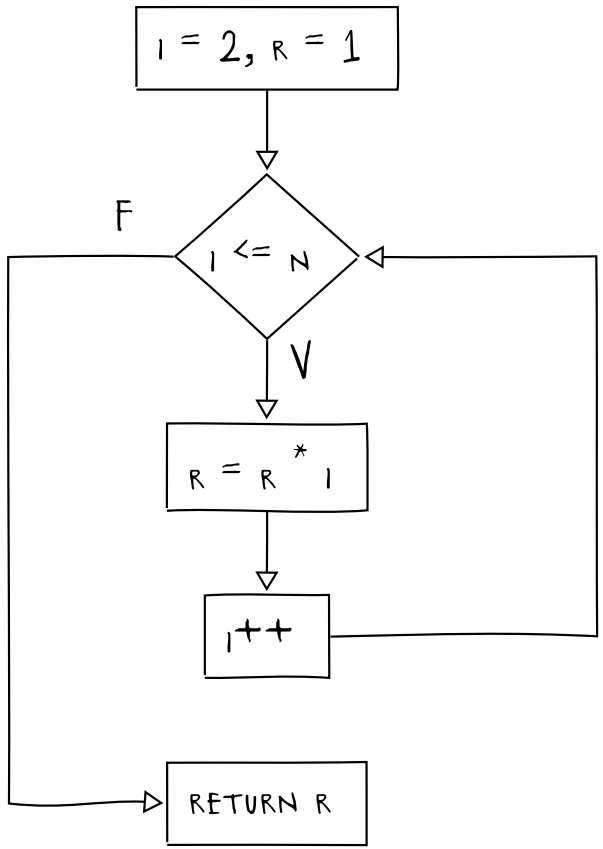
Controle iterativo

- *while*

```
1   // Inteiros com tamanho fixo
2   #include <stdint.h>
3   // Biblioteca padrão
4   #include <stdlib.h>
5   // Função fatorial
6   uint32_t fatorial(uint32_t n) {
7       // Declaração de variáveis
8       uint32_t i = 2, r = 1;
9       // Repete enquanto i <= n
10      while(i <= n) {
11          // r = r * i
12          r = r * i;
13          // i = i + 1
14          i++;
15      }
16      // Retornando valor
17      return r;
18  }
…   ...
```

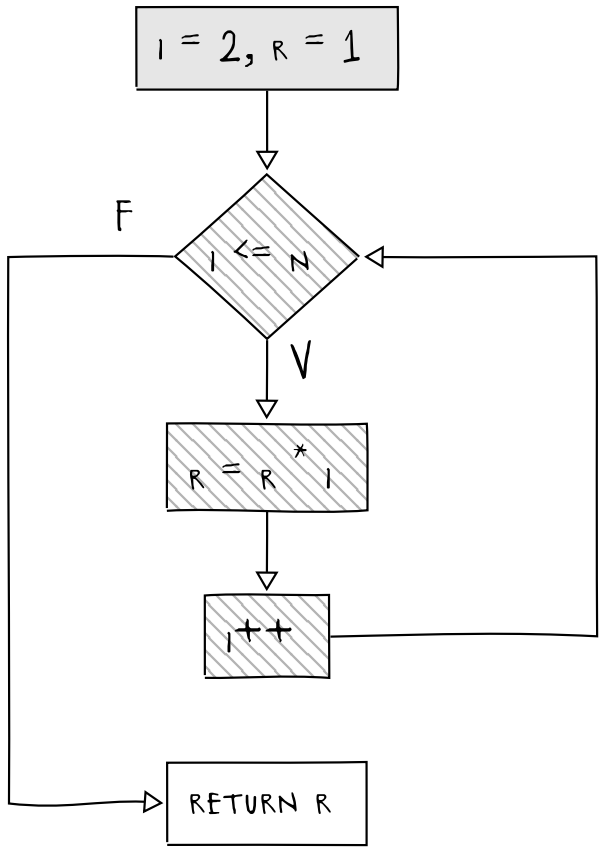
Controle iterativo

- *while*



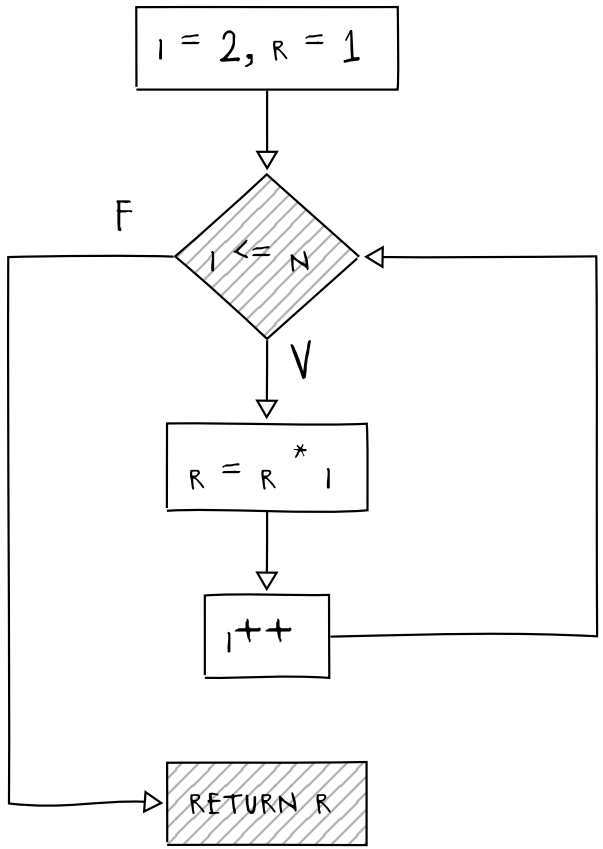
Controle iterativo

- *while*



Controle iterativo

- *while*



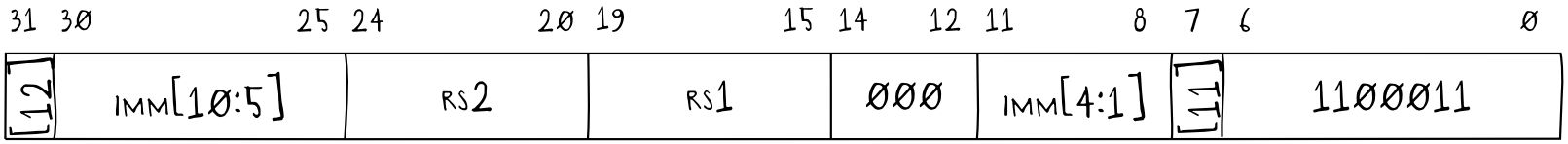
Controle iterativo

- *while*

```
25  # fatorial(a0 = n)
26  fatorial:
27      # t0 = i = 2, t1 = r = 1
28      init:
29          li t0, 2
30          li t1, 1
31      # i <= n
32      iteration:
33          bgt t0, a0, end
34          # r = r * i, i++
35          mul t1, t1, t0
36          addi t0, t0, 1
37          j iteration
38      # end
39      end:
40          mv a0, t1
41          ret
```

Instruções de desvio

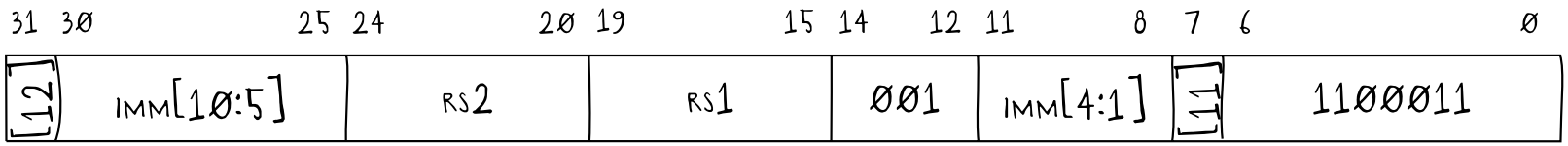
- Desvie se for igual (*beq*)




```
beq rs1, rs2, imm:
    if (rs1 == rs2):
        pc += sign_extension({ imm, 0 })

beqz rs1, imm:
    beq rs1, x0, imm
```

Instruções de desvio

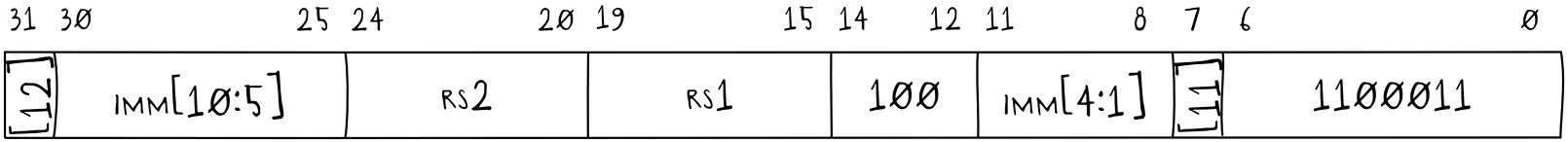
- Desvie se não for igual (*bne*)




```
bne rs1, rs2, imm:
    if (rs1 != rs2):
        pc += sign_extension({ imm, 0 })

bnez rs1, imm:
    bne rs1, x0, imm
```

Instruções de desvio

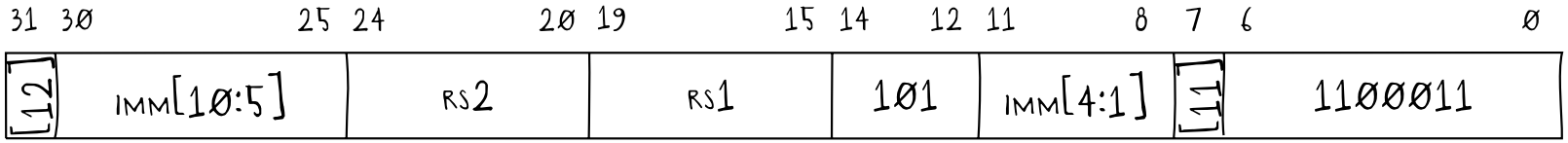
- Desvie se for menor que com sinal (*blt*)




```
blt rs1, rs2, imm:
    if (rs1 < rs2):
        pc += sign_extension({ imm, 0 })

bltz rs1, imm:
    blt rs1, x0, imm

bgtz rs2, imm:
    blt x0, rs2, imm

bgt rs2, rs1, imm:
    blt rs1, rs2, imm
```

Instruções de desvio

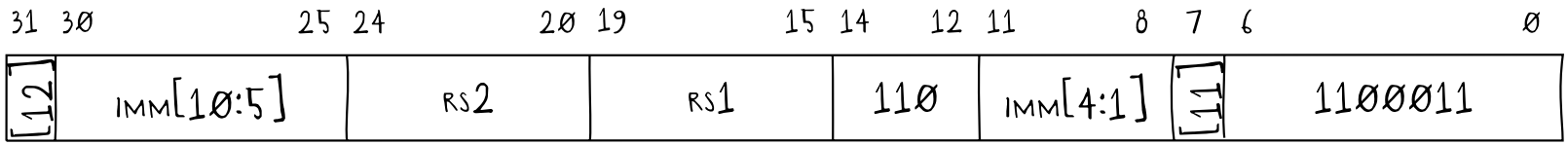
- Desvie se for maior ou igual com sinal (*bge*)




```
bge rs1, rs2, imm:
    if (rs1 >= rs2):
        pc += sign_extension({ imm, 0 })

blez rs2, imm:
    bge x0, rs2, imm

bgez rs1, imm:
    bge rs1, x0, imm

ble rs2, rs1, imm:
    bge rs1, rs2, imm
```

Instruções de desvio

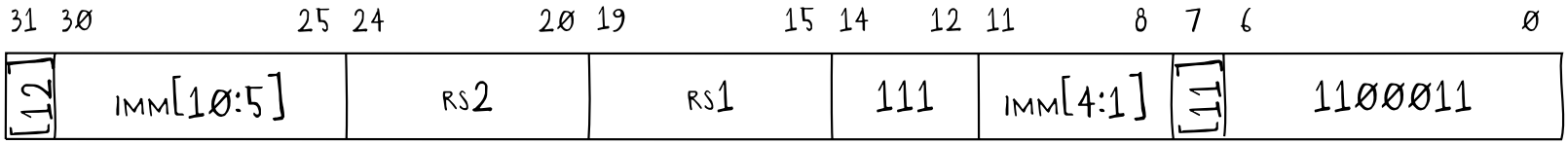
- Desvie se for menor que sem sinal (*bltu*)




```
bltu rs1, rs2, imm:
    if (rs1 < rs2):
        pc += sign_extension({ imm, 0 })

bgtu rs2, rs1, imm:
    bltu rs1, rs2, imm
```

Instruções de desvio

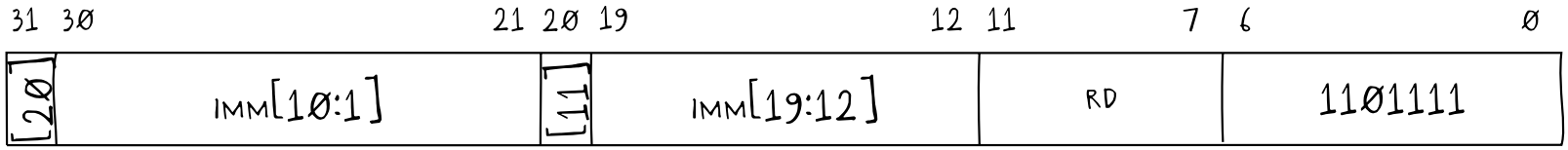
- Desvie se for maior ou igual sem sinal (*bgeu*)




```
bgeu rs1, rs2, imm:
    if (rs1 >= rs2):
        pc += sign_extension({ imm, 0 })

bleu rs2, rs1, imm:
    bgeu rs1, rs2, imm
```

Instruções de desvio

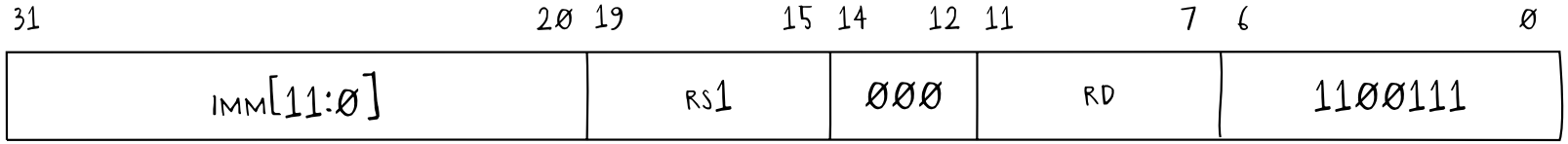
- Desvio com vinculação (*jump and link*)




```
jal rd, imm:
    rd = pc + 4
    pc += sign_extension({ imm, 0 })

jal imm:
    jal ra, imm

j imm:
    jal x0, imm

call imm:
    jal ra, imm
```

Instruções de desvio

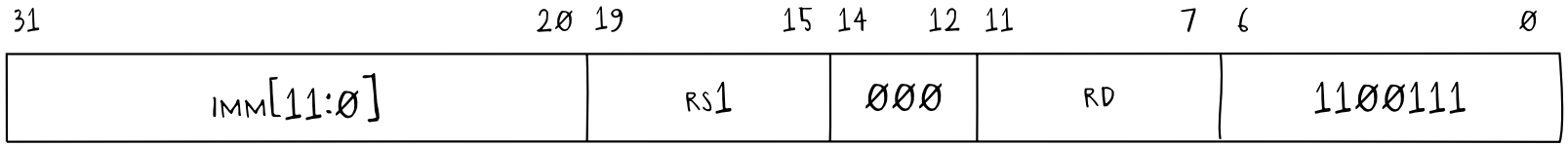
- Desvio com vinculação em registrador (*jump and link register*)




```
jalr rd, rs1, imm:
    rd = pc + 4
    pc = rs1 + sign_extension(imm)

jalr rs1:
    jalr ra, rs1, 0

jr rs1:
    jalr x0, rs1, 0

...
```

Instruções de desvio

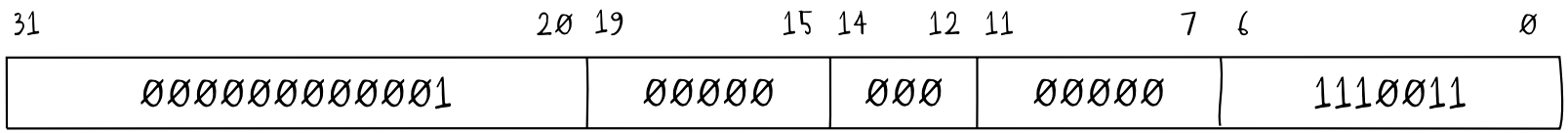
- Desvio com vinculação em registrador (*jump and link register*)




```
jalr rd, rs1, imm:
    rd = pc + 4
    pc = rs1 + sign_extension(imm)

...

ret:
    jalr x0, ra, 0

call imm32:
    auipc ra, get_31_12(imm32)
    jalr ra, ra, get_11_0(imm32)
```

Instruções de desvio

- Transferência de controle para ambiente (*ebreak*)



É UTILIZADA PARA FINALIZAÇÃO DA SIMULAÇÃO

Funções e procedimentos

- Convenção de chamada (ILP32)
  - Define as regras para passagem de parâmetros
  - Como os registradores devem ser armazenados e utilizados
  - Alinhamento de 16 bytes para pilha

Registrador	Rótulo	Descrição	Preservado
x0	zero	Valor constante zero	-
x1	ra	Endereço de retorno	Sim
x2	sp	Ponteiro da pilha	Sim
x3	gp	Ponteiro global	-
x4	tp	Ponteiro de thread	-
x5	t0	Temporário/Link alternativo	Não
x6-x7	t1-t2	Temporários	Não
x8	s0/fp	Valor salvo/Ponteiro do quadro	Sim
x9	s1	Valor salvo	Sim
x10-x11	a0-a1	Argumentos de função/retorno	Não
x12-x17	a2-a7	Argumentos da função	Não
x18-x27	s2-s11	Valores salvos	Sim
x28-x31	t3-t6	Temporários	Não

Funções e procedimentos

- Função fatorial recursiva

```
1  // Inteiros com tamanho fixo
2  #include <stdint.h>
3  // Biblioteca padrão
4  #include <stdlib.h>
5  // Função fatorial recursiva
6  uint32_t fatorial(uint32_t n) {
7      // Caso base
8      if(n == 0) return 1;
9      // Recorrência
10     else return n * fatorial(n - 1);
11 }
12 // Função principal
13 int main() {
14     // fatorial(5)
15     fatorial(5);
16     // Retorno sem erro
17     return 0;
18 }
```

Funções e procedimentos

- Função principal

```
57 # Setando a função main como global
58 .global main
59 # main()
60 main:
61     # Prólogo
62     addi sp, sp, -16
63     sw ra, 0(sp)
64     # fatorial(5)
65     li a0, 5
66     call fatorial
67     # Epílogo
68     lw ra, 0(sp)
69     addi sp, sp, 16
70     # Setando valor de retorno para zero (sucesso)
71     addi a0, zero, 0
72     # Retornando da chamada
73     ret
```

Funções e procedimentos

- Função principal

```
57 # Setando a função main como global
58 .global main
59 # main()
60 main:
61     # Prólogo
62     addi sp, sp, -16
63     sw ra, 0(sp)
64     # fatorial(5)
65     li a0, 5
66     call fatorial
67     # Epílogo
68     lw ra, 0(sp)
69     addi sp, sp, 16
70     # Setando valor de retorno para zero (sucesso)
71     addi a0, zero, 0
72     # Retornando da chamada
73     ret
```

Funções e procedimentos

- Função principal

```
57  # Setando a função main como global
58  .global main
59  # main()
60  main:
61      # Prólogo
62      addi sp, sp, -16
63      sw ra, 0(sp)
64      # fatorial(5)
65      li a0, 5
66      call fatorial
67      # Epílogo
68      lw ra, 0(sp)
69      addi sp, sp, 16
70      # Setando valor de retorno para zero (sucesso)
71      addi a0, zero, 0
72      # Retornando da chamada
73      ret
```

Funções e procedimentos

- Função principal

```
57 # Setando a função main como global
58 .global main
59 # main()
60 main:
61     # Prólogo
62     addi sp, sp, -16
63     sw ra, 0(sp)
64     # fatorial(5)
65     li a0, 5
66     call fatorial
67     # Epílogo
68     lw ra, 0(sp)
69     addi sp, sp, 16
70     # Setando valor de retorno para zero (sucesso)
71     addi a0, zero, 0
72     # Retornando da chamada
73     ret
```

Funções e procedimentos

- Função fatorial recursiva

```
25  # fatorial(a0 = n)
26  fatorial:
27      # begin
28      begin:
29          # Prólogo
30          addi sp, sp, -16
31          sw ra, 0(sp)
…       ...
50      # end
51      end:
52          # Epílogo
53          lw ra, 0(sp)
54          addi sp, sp, 16
55          # Retornando valor
56          mv a0, t0
57          ret
```

Funções e procedimentos

- Função fatorial recursiva

```
25  # fatorial(a0 = n)
26  factorial:
…       ...
32      # Caso base
33      base:
34          # Checar se n != 0
35          bnez a0, recursive
36          # Retornando 1
37          li t0, 1
38          j end
…       ...
```

Funções e procedimentos

- Função fatorial recursiva

```
25  # fatorial(a0 = n)
26  factorial:
…       ...
39      # Recorrência
40      recursive:
41          # Salvando n na pilha
42          sw a0, 4(sp)
43          # fatorial(n - 1)
44          addi a0, a0, -1
45          call fatorial
46          # Restaurando n da pilha
47          lw a0, 4(sp)
48          # Calculando n * fatorial(n - 1)
49          mul t0, t0, a0
…       ...
```

Exercício

- Considerando a implementação da função *fibonacci*
  - Realize a sua tradução para código de montagem e realize a sua execução para preencher um vetor ($n = 48$)

```
1   // Inteiros com tamanho fixo
2   #include <stdint.h>
3   // Biblioteca padrão
4   #include <stdlib.h>
5   // Sequência de Fibonacci
6   uint32_t V[48] = { 0 };
7   // Função fibonacci
8   uint32_t fibonacci(uint32_t n) {
9       // (n <= 1) -> r = n
10      uint32_t r = n, tn2 = 0, tn1 = 1;
11      // (n > 1)  -> r = t(n - 2) + t(n - 1)
12      for(uint32_t i = 2; i < n; i++, tn2 = tn1, tn1 = r)
13          r = tn2 + tn1;
14      // fibonacci(n)
15      return r;
16  }
…   ...
```