PCS 3115 – Sistemas Digitais I

Projeto 4 - Little Sort:
Ordenação De Dados de Uma Memória

EAD – Ensino A Distância

Parte I:
Fluxo de Dados.

Aula: 28 – Data: 08/07 (Q)

Prof. Dr. Marco Túlio Carvalho de Andrade

versão: 1.5 (Julho/2020)

Tópicos da Aula

- Visão geral do Projeto 4
- O que é fornecido aos alunos
- O que submeter ao Juiz
- Detalhamento do projeto
  - Algoritmo e Diagrama ASM
  - Fluxo de Dados
  - Unidade de Controle
  - Exemplo de *Testbench* (reduzido)

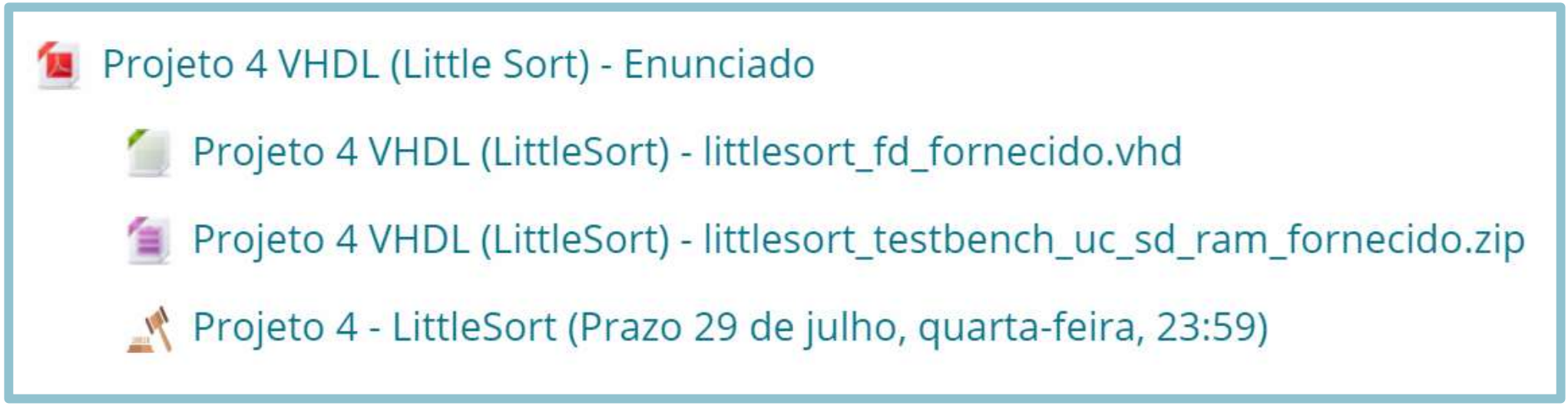
Visão geral do Projeto

- Projeto 4: ***Little Sort*** - Ordenação de Dados de uma Memória

  – Enunciado: Projetar um circuito digital que realiza a ordenação dos dados armazenados em uma memória externa.

Visão geral do Projeto

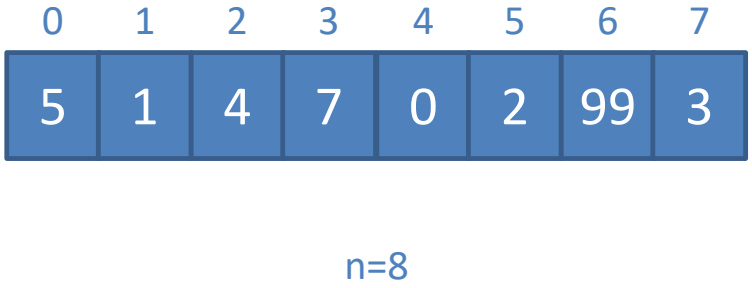
- Projeto 4: ***Little Sort*** - Ordenação de Dados de uma Memória



Visão geral do Projeto

- Projeto 4: ***Little Sort*** - Ordenação de Dados de uma Memória
  - Algoritmo em C:

```
// reordena vetor a com n elementos
void LittleSort (int a[], int n)
{
   int j, temp;
   for (j = 0; j < n - 1; j++)
      if (a[j] > a[j + 1])  {
         temp = a[j];
         a[j] = a[j + 1];
         a[j + 1] = temp;
      }
}
```



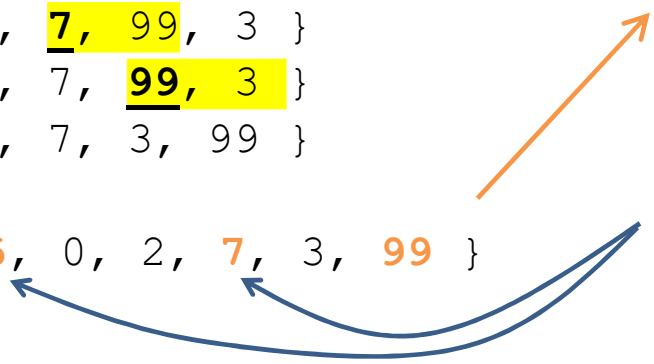
Visão geral do Projeto

- Projeto 4: ***Little Sort*** - Ordenação de Dados de uma Memória
  - Exemplo:

```
Memoria inicial: { 5, 1, 4, 7, 0, 2, 99, 3 }

Passo 0:  { 5, 1, 4, 7, 0, 2, 99, 3 }
Passo 1:  { 1, 5, 4, 7, 0, 2, 99, 3 }
Passo 2:  { 1, 4, 5, 7, 0, 2, 99, 3 }
Passo 3:  { 1, 4, 5, 7, 0, 2, 99, 3 }
Passo 4:  { 1, 4, 5, 0, 7, 2, 99, 3 }
Passo 5:  { 1, 4, 5, 0, 2, 7, 99, 3 }
Passo 6:  { 1, 4, 5, 0, 2, 7, 99, 3 }
Fim:      { 1, 4, 5, 0, 2, 7, 3, 99 }

memoria final:  { 1, 4, 5, 0, 2, 7, 3, 99 }
```

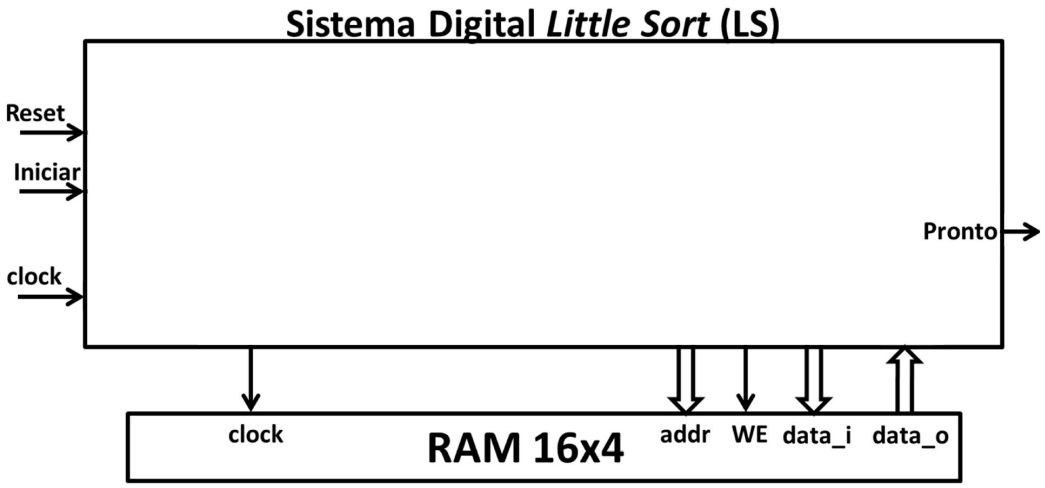


Ao final, o **maior elemento** é armazenado na última posição da memória.

Outros elementos também são reordenados durante o processamento.

Visão geral do Projeto

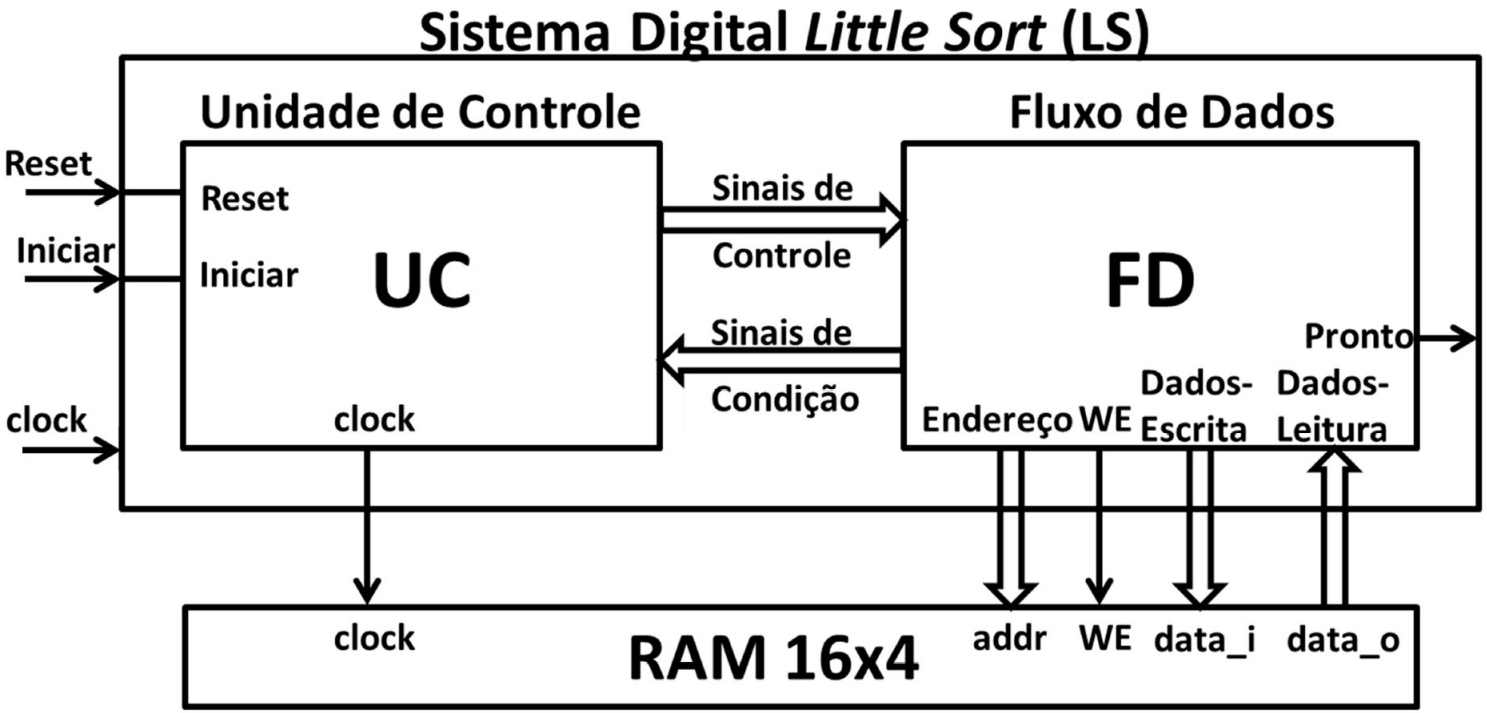
- Diagrama de blocos do Sistema Digital




```
entity littleSort is
  port
  (
    clock:            in  bit;
    reset:            in  bit;
    Iniciar:          in  bit;
    mem_we:           out bit;  -- interface com memoria externa
    mem_endereco:     out bit_vector(3 downto 0);
    mem_dado_write:   out bit_vector(3 downto 0);
    mem_dado_read:    in  bit_vector(3 downto 0);
    Pronto:           out bit
  );
end entity;
```

Visão geral do Projeto

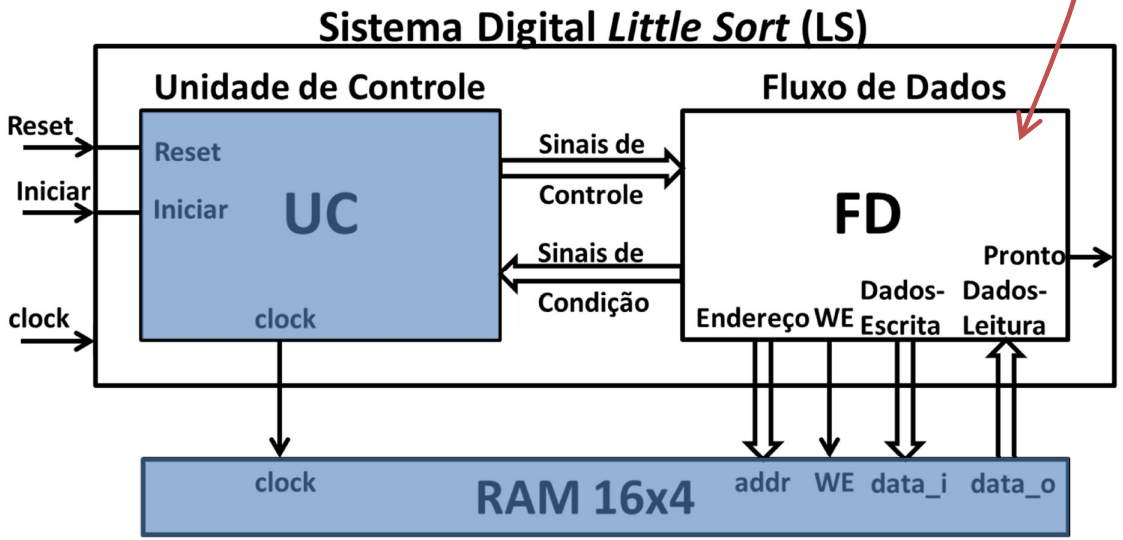
- Diagrama de blocos do **Sistema Digital** (detalhamento)



O que é fornecido aos alunos

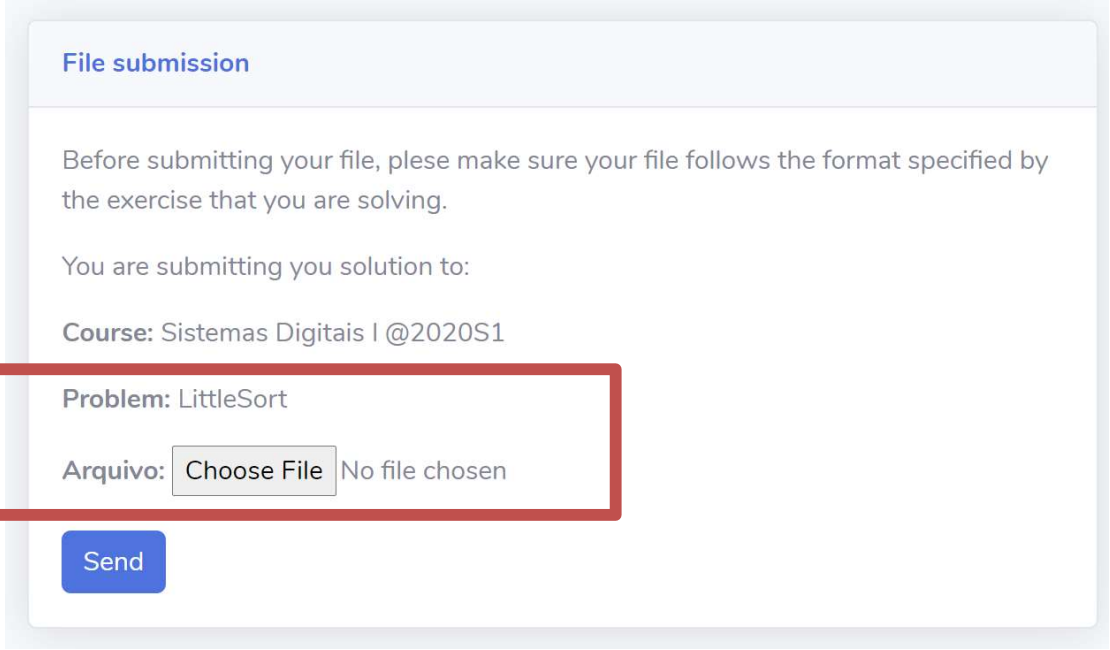
- O que é fornecido aos alunos
  - 2 arquivos VHDL:
    - littlesort_fd_fornecido.vhd
    - littlesort_testbench_uc_sd_ram_fornecido.vhd
    - memoria1.dat (conteúdo da RAM16x4))

Parcial (completar)



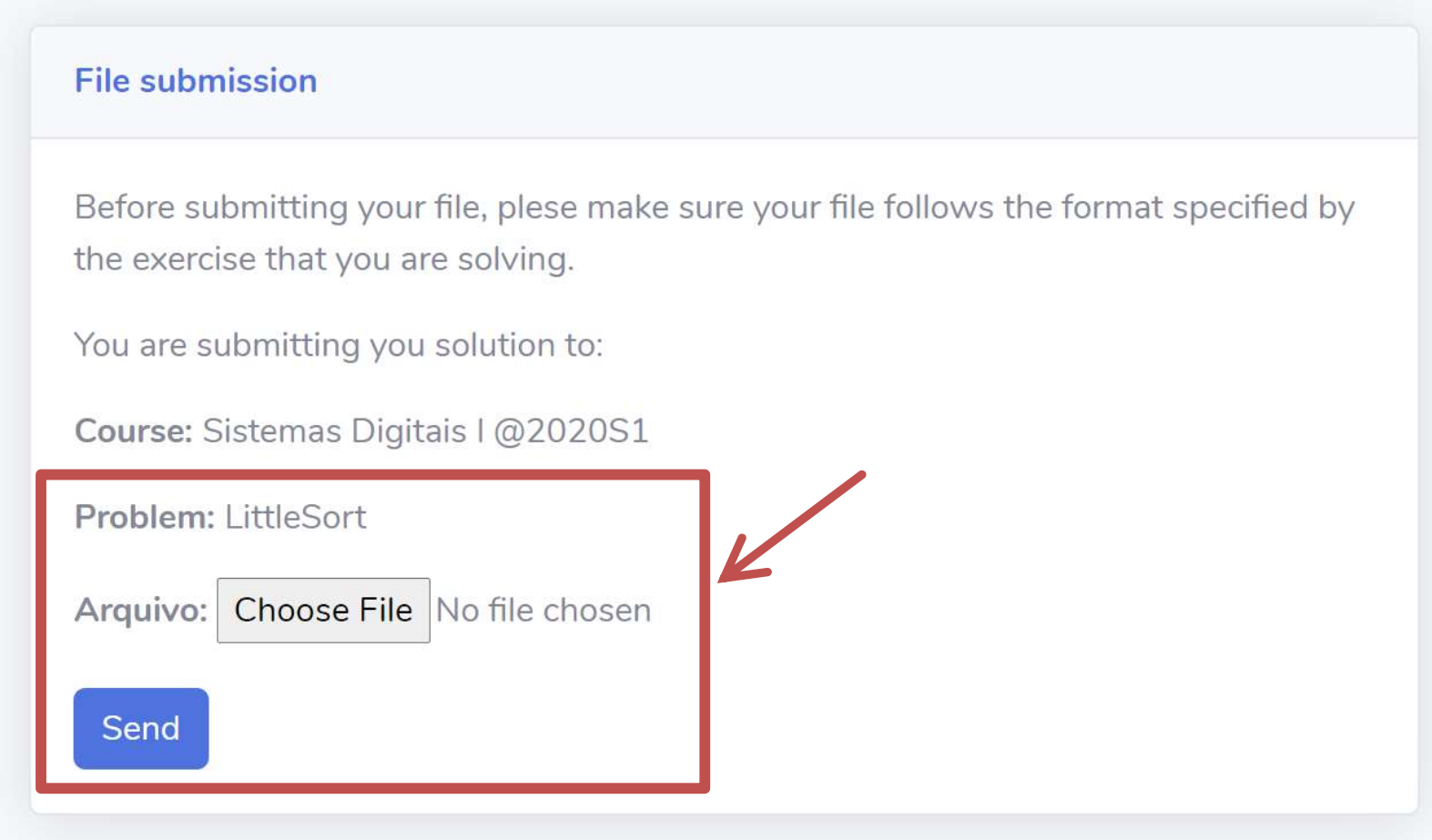
O que submeter ao Juiz

- Submissão ao Juiz
  - Submeter <u>um arquivo VHDL</u> contendo somente o **Fluxo de Dados** do circuito (entidade littlesort_fd e componentes internos).
  - Usar arquivo VHDL littlesort_fd_fornecido.vhd



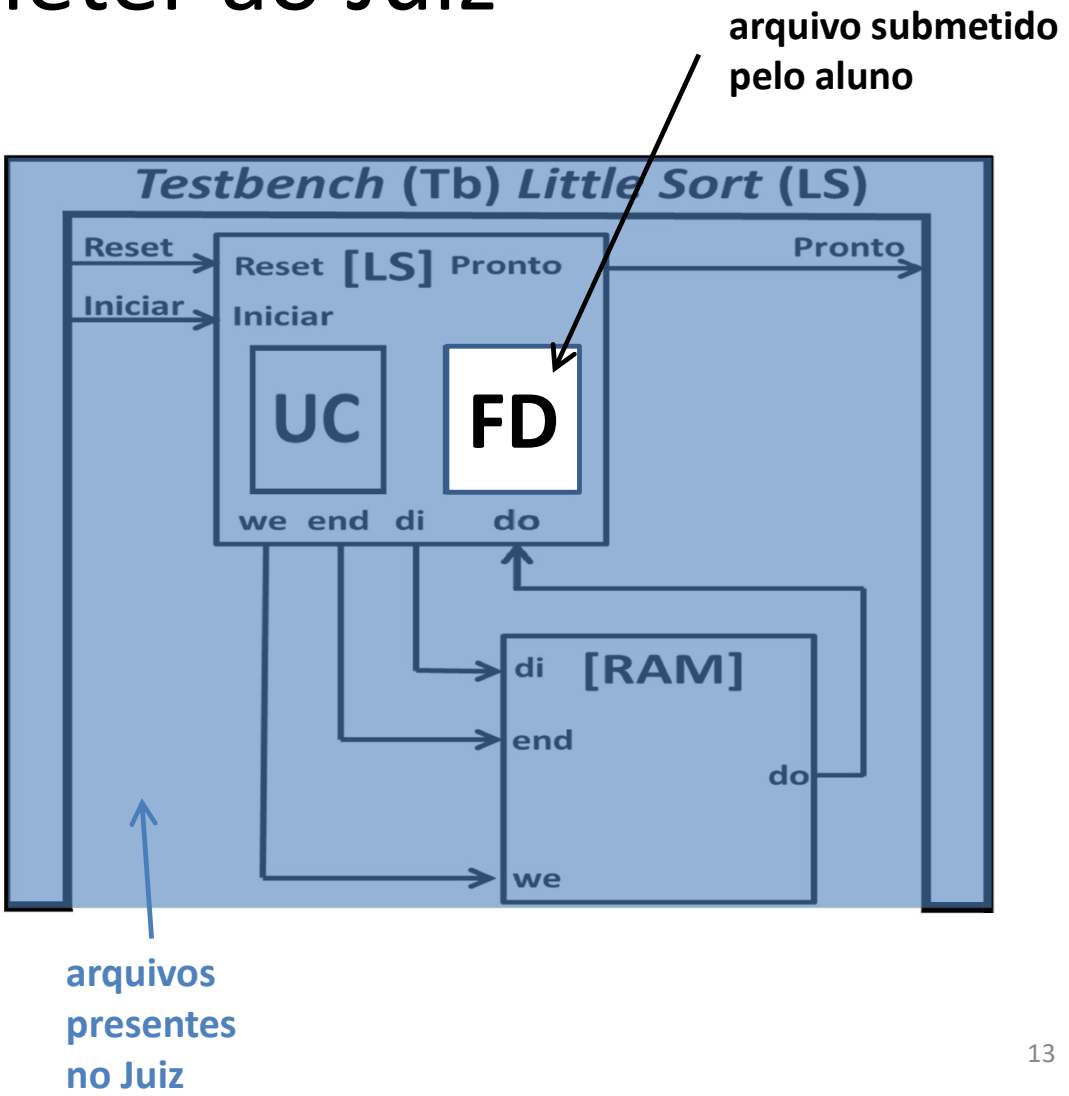
O que submeter ao Juiz

- Submissão ao Juiz



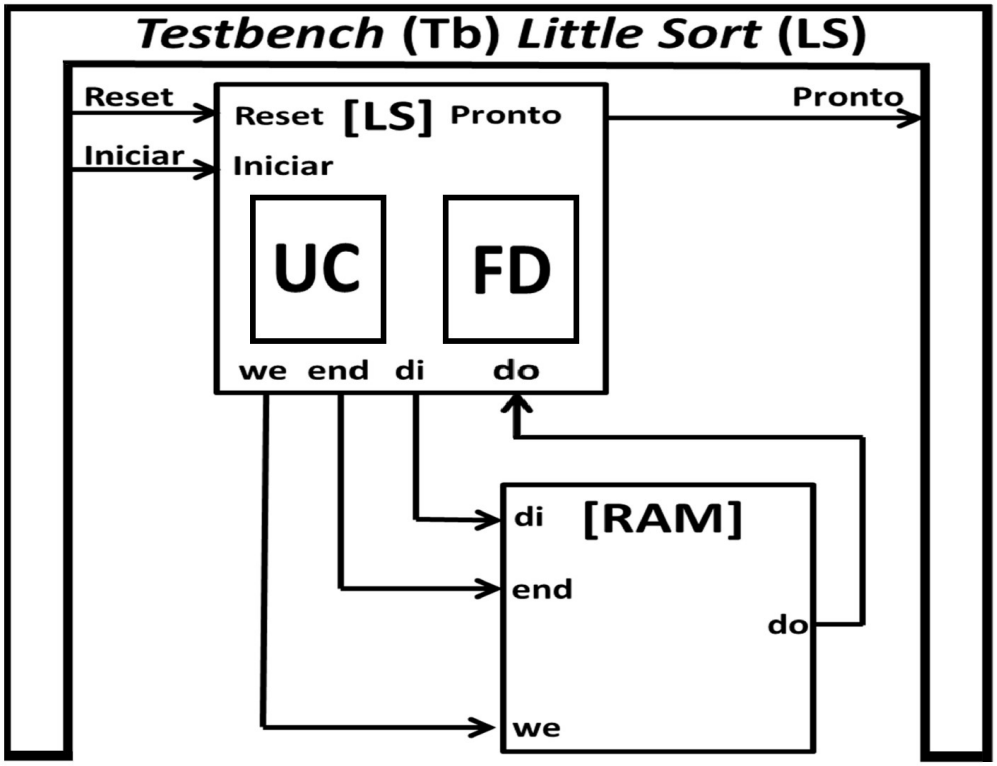
O que submeter ao Juiz

- Submissão ao Juiz
  - Submeter um arquivo VHDL contendo somente o **Fluxo de Dados** do circuito (entidade littlesort_fd e componentes internos).



O que submeter ao Juiz

- Submissão ao Juiz
  - O que o Juiz faz?
    1. Agrupa arquivo submetido pelo aluno com os arquivos do *testbench* de avaliação do projeto;
    2. Compilação do projeto com GHDL;
    3. Simulação e avaliação do projeto.



O projeto pode ser testado da mesma forma, usando GHDL, EDA Playground.

Detalhamento do projeto

- Algoritmo em C

```
// reordena vetor a com n elementos
void LittleSort (int a[], int n)
{
   int j, temp;
   for (j = 0; j < n - 1; j++)
      if (a[j] > a[j + 1])  {
         temp = a[j];
         a[j] = a[j + 1];
         a[j + 1] = temp;
      }
}
```

Modelo temporal: exemplo ilustrativo (Aula VHDL – Slide 12)

- As declarações em VHDL são executadas **concorrentemente**
  - Estamos descrevendo um hardware, **não uma sequência de instruções** de software…
- Exemplo: operação de **swap(a,b)**: "troca a por b e vice-versa"

Em C, isso **não** funciona:

```
1:   a = b;
2:   b = a;
```

Em C, isso funciona:

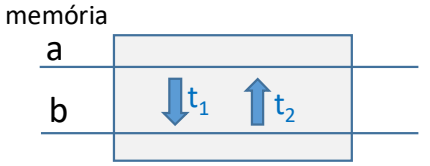
```
1:   x = a;
2:   a = b;
3:   b = x
```

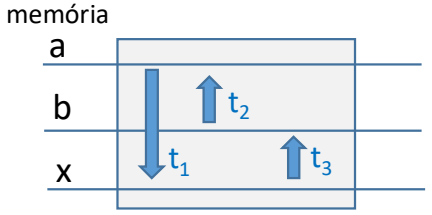
Em C, também funciona:
(mas desperdiça memória...)

```
1:   sa = b;
2:   sb = a;
```

tempo	memória	
	a	b
0	5	7
1	7	7
2	7	7

tempo	memória		
	a	b	x
0	5	7	*
1	5	7	5
2	7	7	5
3	7	5	5





Modelo temporal: exemplo ilustrativo (Aula VHDL – Slide 13)

- As declarações em VHDL são executadas **concorrentemente**
  - Estamos descrevendo um hardware, **não uma sequência de instruções** de software…
- Exemplo: operação de **swap(a,b)**: "troca a por b e vice-versa"

Em C, isso **não** funciona:

```
1:  a = b;
2:  b = a;
```

Em C, isso funciona:

```
1:  x = a;
2:  a = b;
3:  b = x
```

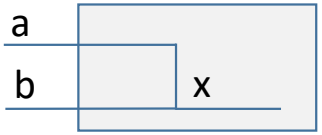
Em VHDL, isso **não** funciona:

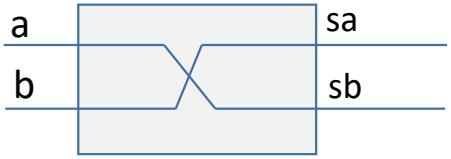
```
1:  x <= a;
2:  a <= b;
3:  b <= x;
```

Em VHDL, isso funciona:

```
1:  sa <= b;
2:  sb <= a;
```

tempo	memória		
	a	b	x
0	5	7	*
1,2,3	??	??	??






```
entity swap is
  port (a, b: in STD_LOGIC; sa,sb: out STD_LOGIC);
end swap
```

Detalhamento do projeto

- Algoritmo e Pseudocódigo

Algoritmo em C:

```
// reordena vetor a com n elementos
void LittleSort (int a[], int n)
{
   int j, temp;
   for (j = 0; j < n - 1; j++)
      if (a[j] > a[j + 1])  {
         temp = a[j];
         a[j] = a[j + 1];
         a[j + 1] = temp;
      }
}
```

Pseudocódigo:

```
// reordena vetor a com n elementos
algoritmo LittleSort (vetor a[], inteiro n)
{
      inteiro j;
      for (j = 0; j < n-1; j++)  {
            if (a[j] > a[j + 1])  {
                  troca valores entre a[j] e a[j + 1];
            }
      }
}
```

Detalhamento do projeto

- Algoritmo e Diagrama ASM

Pseudocódigo:

```
// reordena vetor a com n elementos
algoritmo LittleSort (vetor a[], inteiro n)
{
    inteiro j;
    for (j = 0; j < n-1; j++)  {
        if (a[j] > a[j + 1])  {
            troca valores entre a[j] e a[j + 1];
        }
    }
}
```

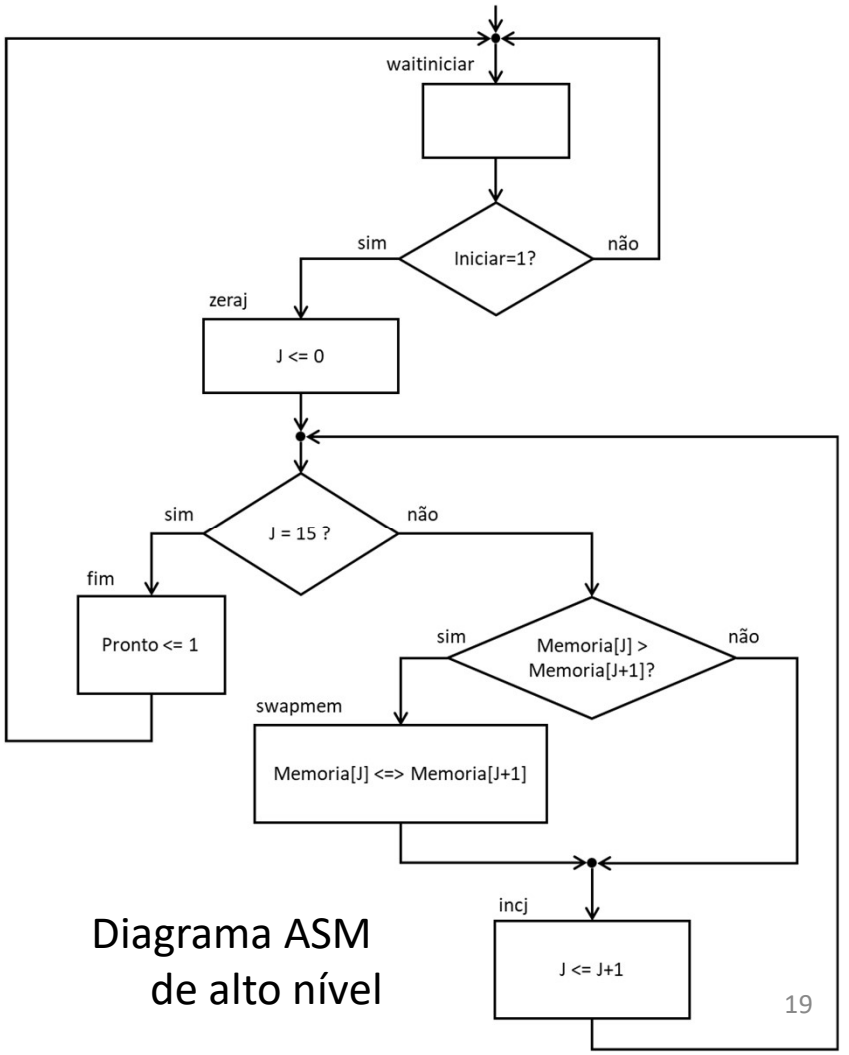
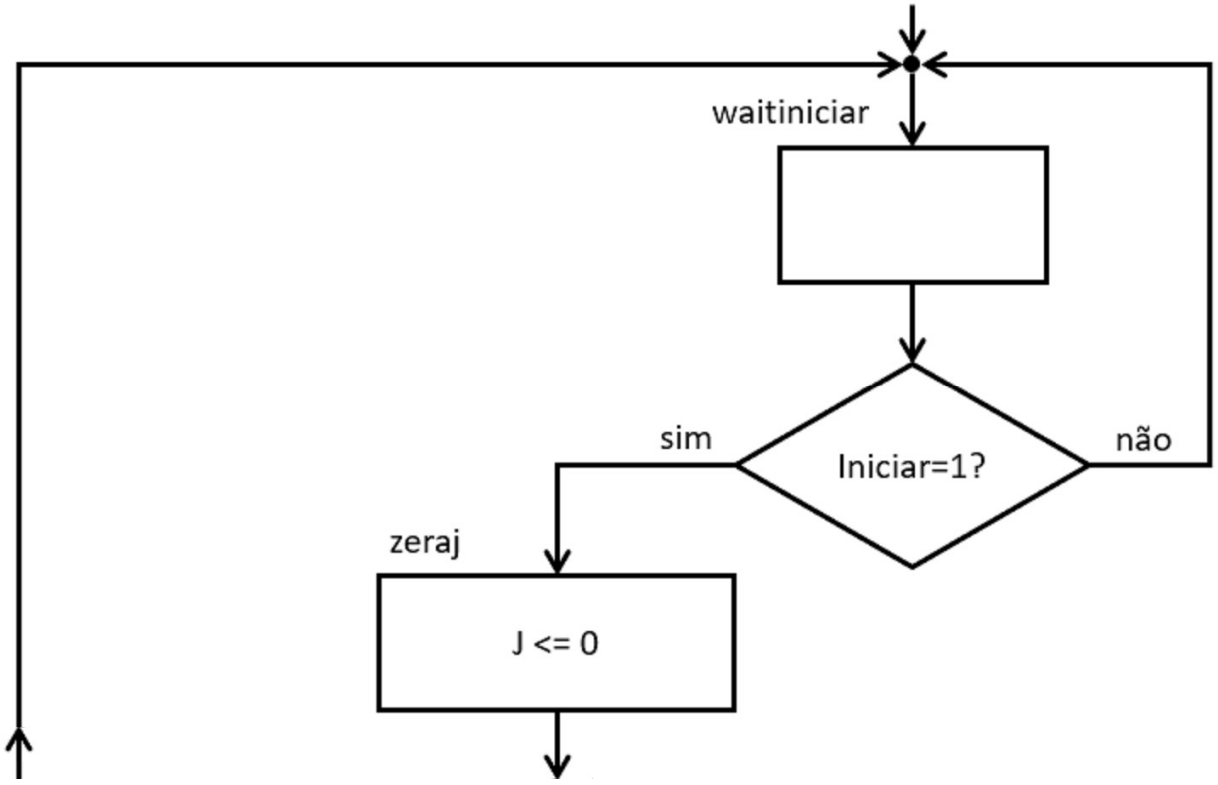


Diagrama ASM de alto nível

Detalhamento do projeto

- Algoritmo e Diagrama ASM

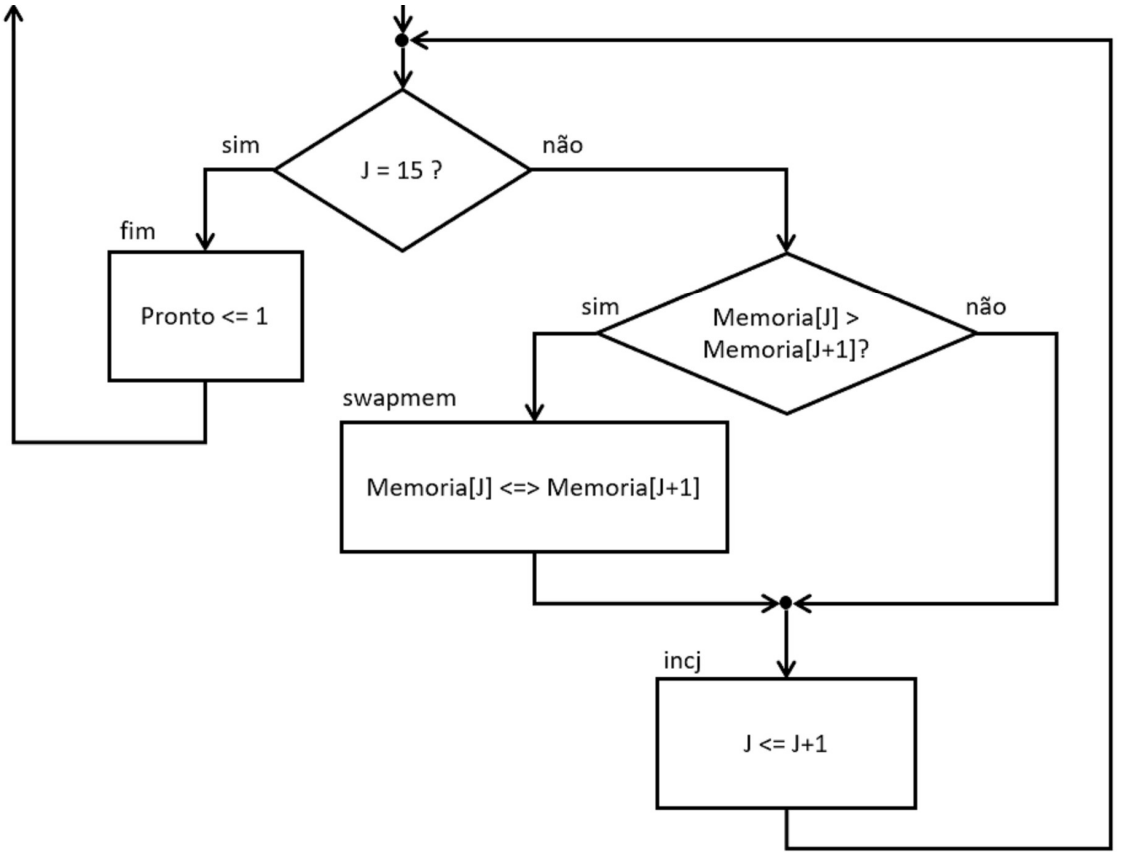
Diagrama ASM de alto nível (parte 1)



Detalhamento do projeto

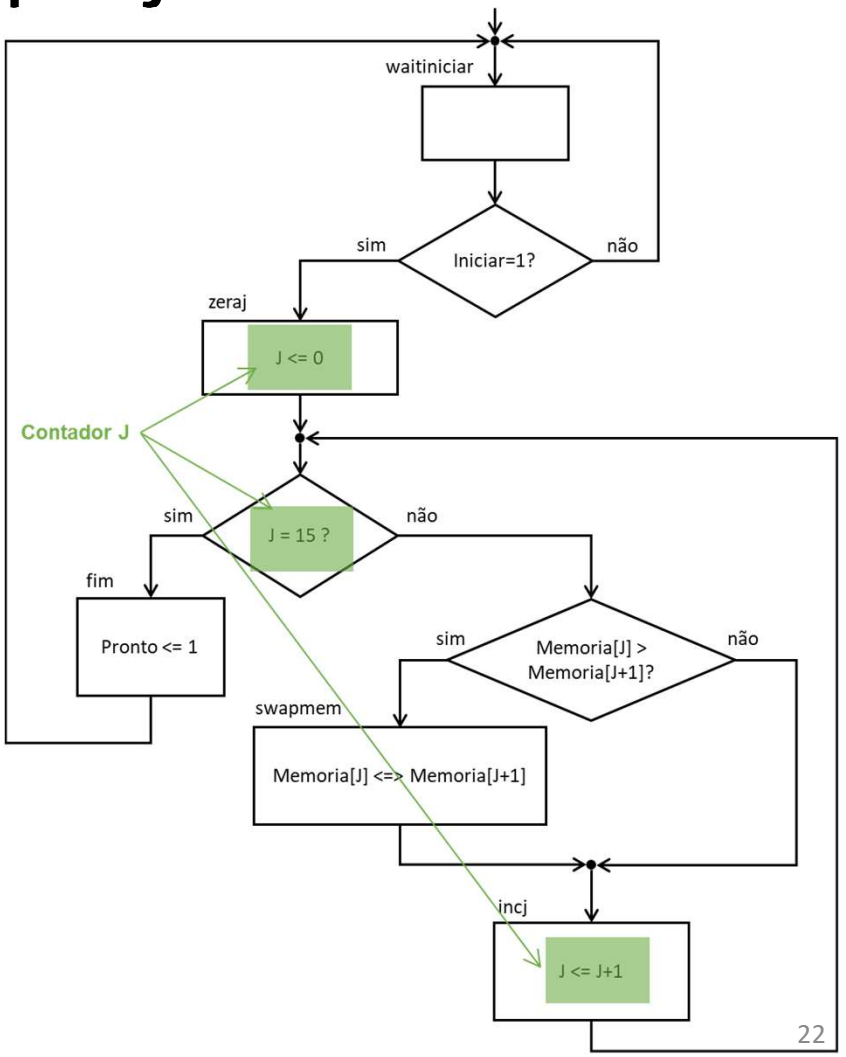
- Algoritmo e Diagrama ASM

Diagrama ASM
de alto nível
(parte 2)



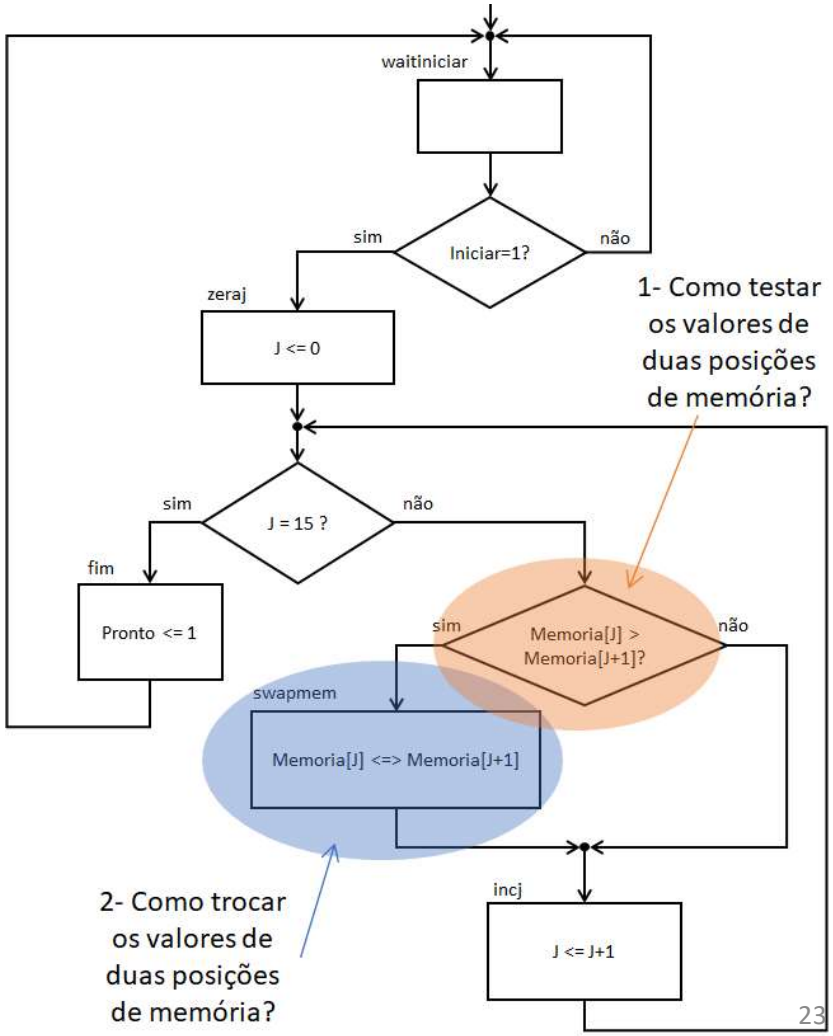
Detalhamento do projeto

- Elementos do Fluxo de Dados
  - Identificação a partir do Diagrama ASM
    1. Memória
    2. Contador J



Detalhamento do projeto

- Detalhamento do Diagrama ASM
  - Operações sobre o conteúdo da memória:
    1. Comparação do conteúdo de duas posições
    2. Troca de valores de duas posições



Detalhamento do projeto

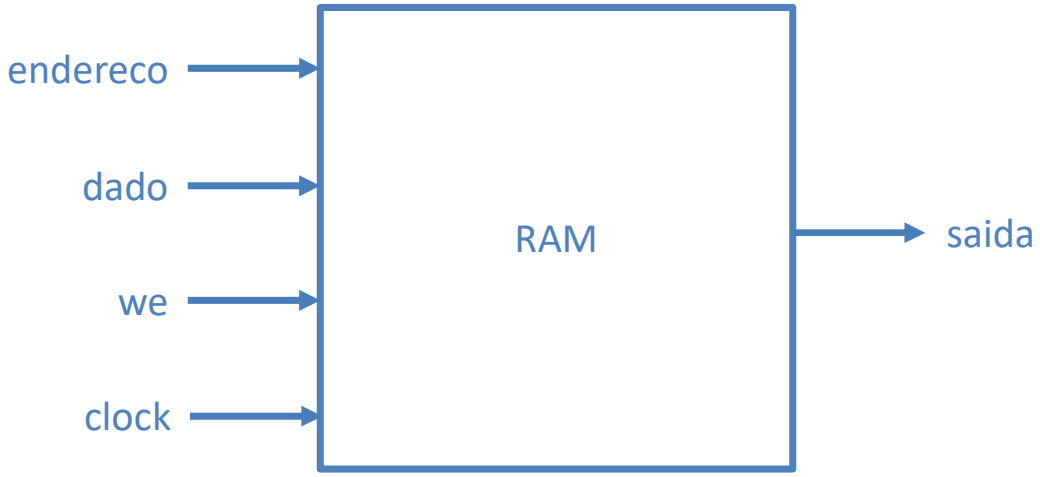
- Detalhamento do Diagrama ASM
  - Operações sobre a memória RAM:

1. Leitura da posição **p**

   endereco <= p
   we <= 0
   valor lido em saida

2. Escrita do valor **d** na posição **p**

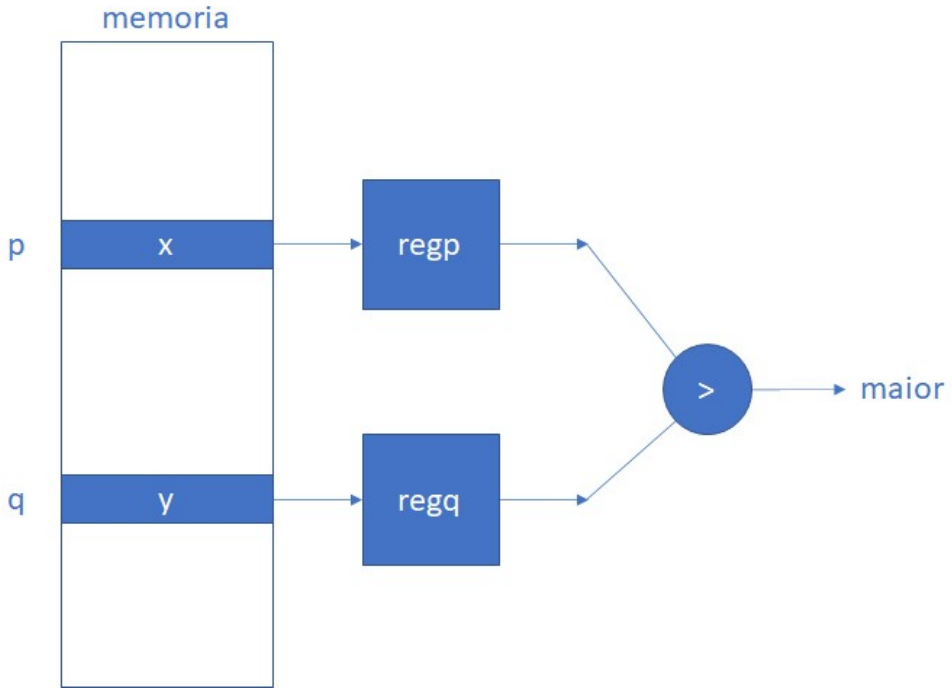
   endereco <= p
   we <= 1
   dado <= d
   operação realizada na borda do *clock*



Acesso de apenas uma posição por vez.

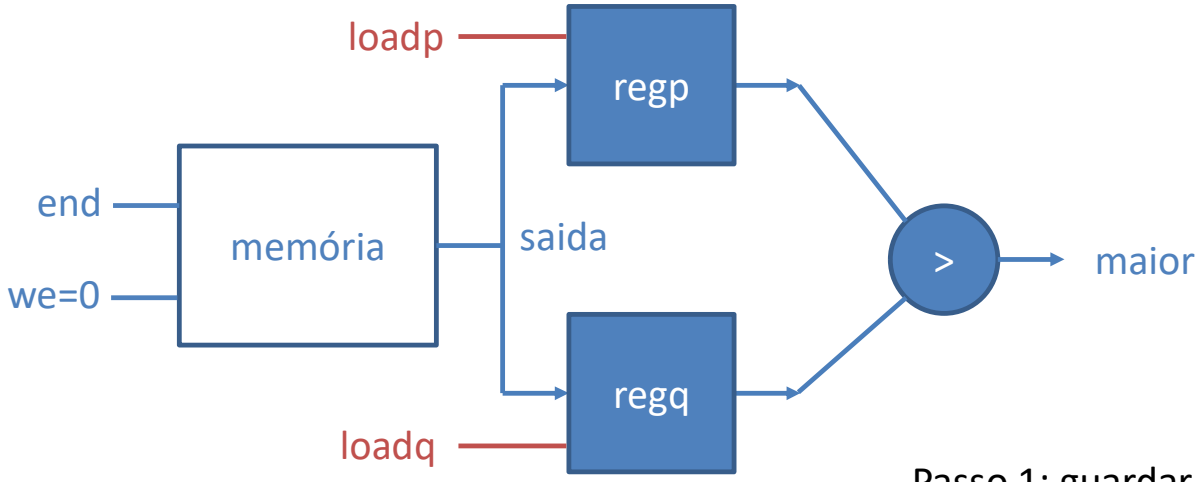
Detalhamento do projeto

- Detalhamento do Diagrama ASM
  1. Comparação do conteúdo de duas posições



Detalhamento do projeto

- Detalhamento do Diagrama ASM
  1. Comparação do conteúdo de duas posições



Passo 1: guardar valor de memoria[p] em regp (loadp=1)

Passo 2: guardar valor de memoria[q] em regq (loadq=1)

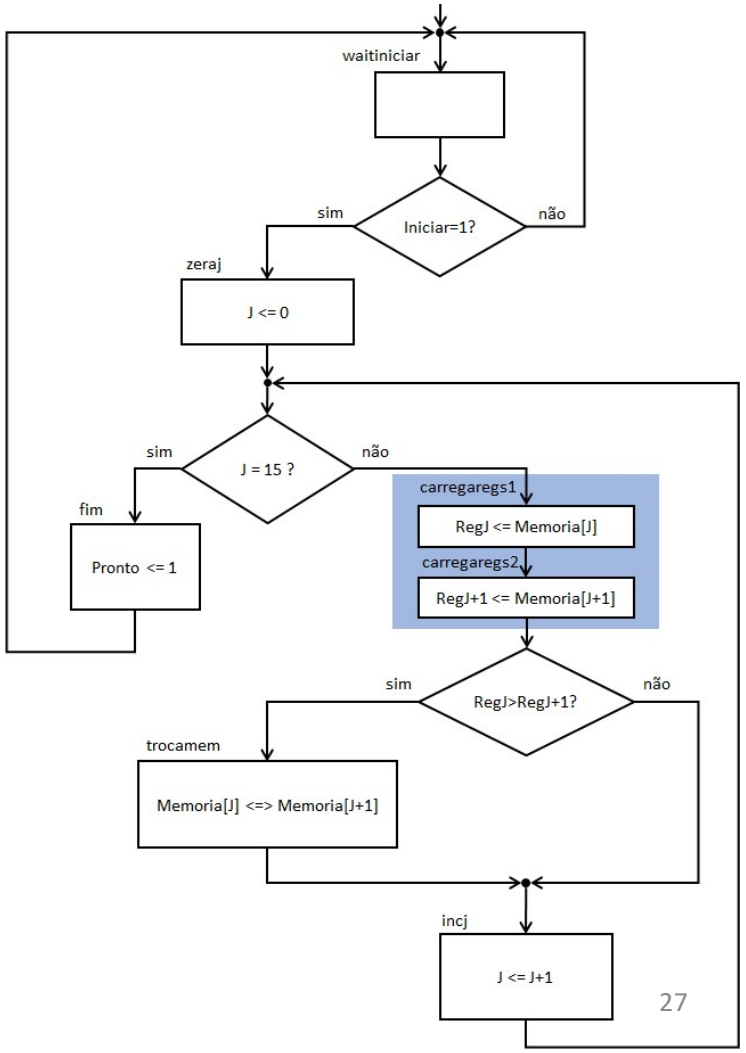
Passo 3: comparar valores de regp e regq

Detalhamento do projeto

- Detalhamento do Diagrama ASM

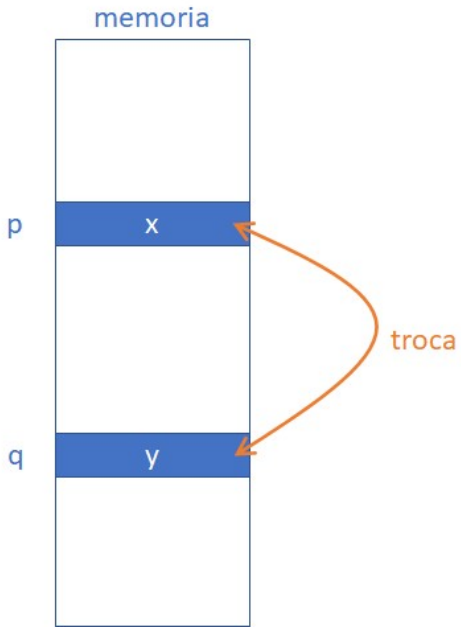
1. Comparação do conteúdo de duas posições

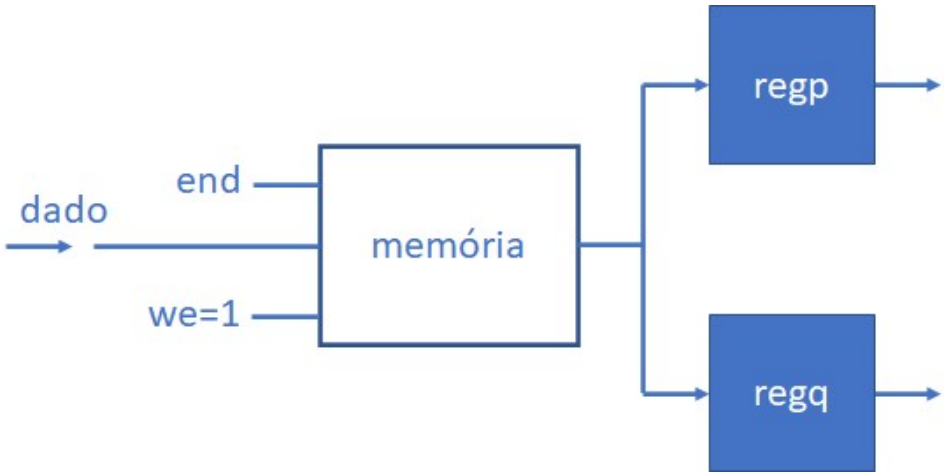
   - Acrescentar estados para armazenamento dos valores das posições j e j+1 em registradores (regJ e regJmais1)
   - Bloco de decisão compara valores usando os registradores



Detalhamento do projeto

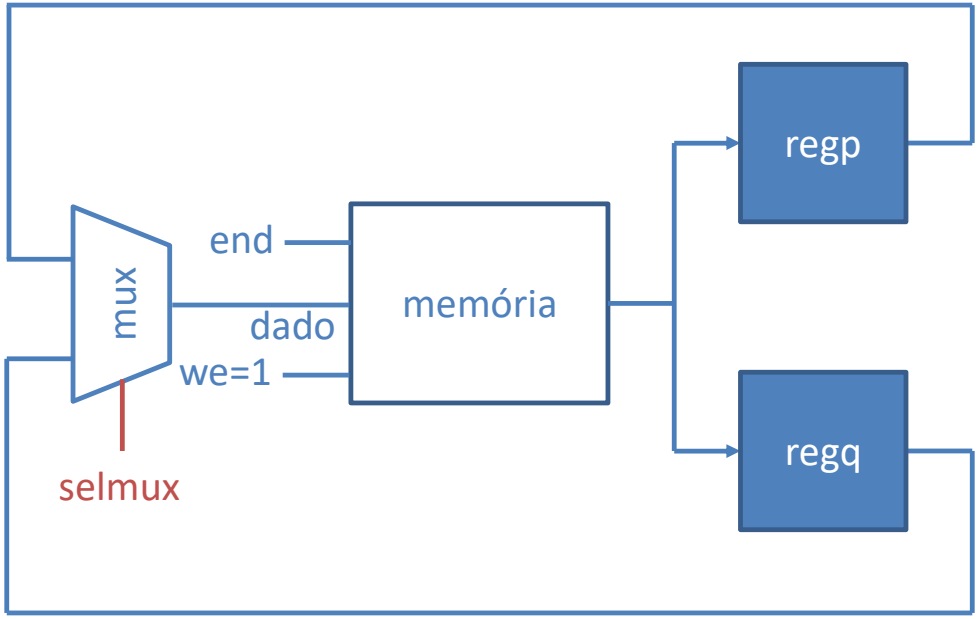
- Detalhamento do Diagrama ASM
  2. Troca de valores de duas posições





Detalhamento do projeto

- Detalhamento do Diagrama ASM
  2. Troca de valores de duas posições



Como ligar as saídas de regp e regq na entrada de dados da memória?
Resposta: MUX

Passo 1: guardar valor de regp na posição q (selmux=0)

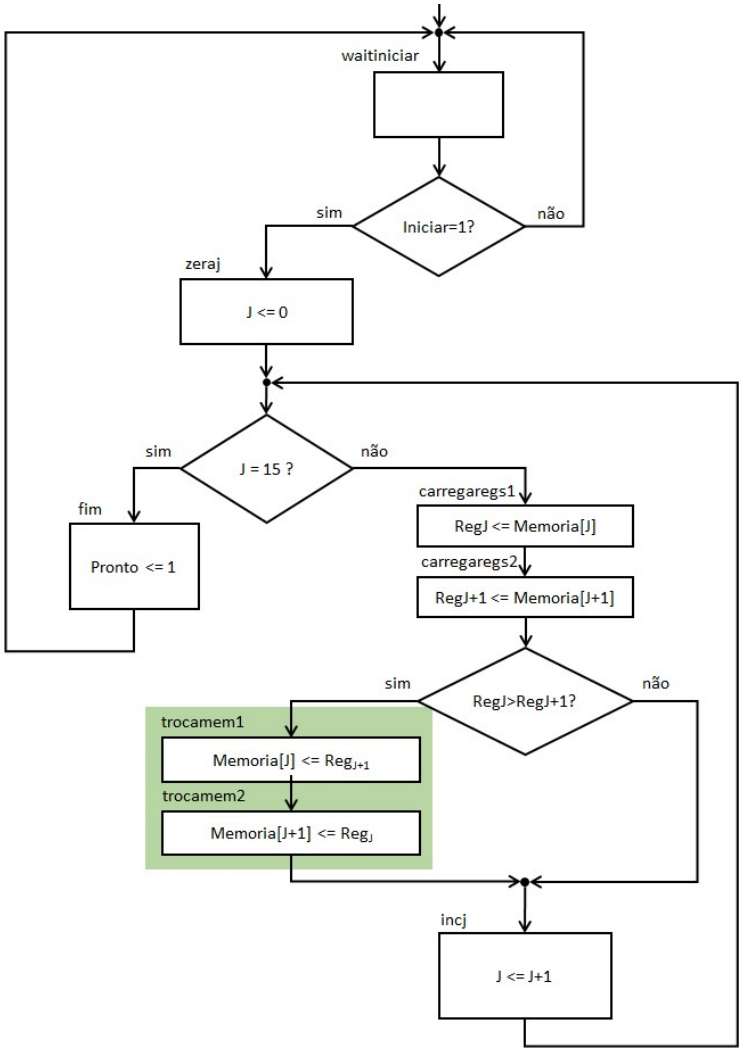
Passo 2: guardar valor de regq na posição p (selmux=1)

Detalhamento do projeto

- Detalhamento do Diagrama ASM

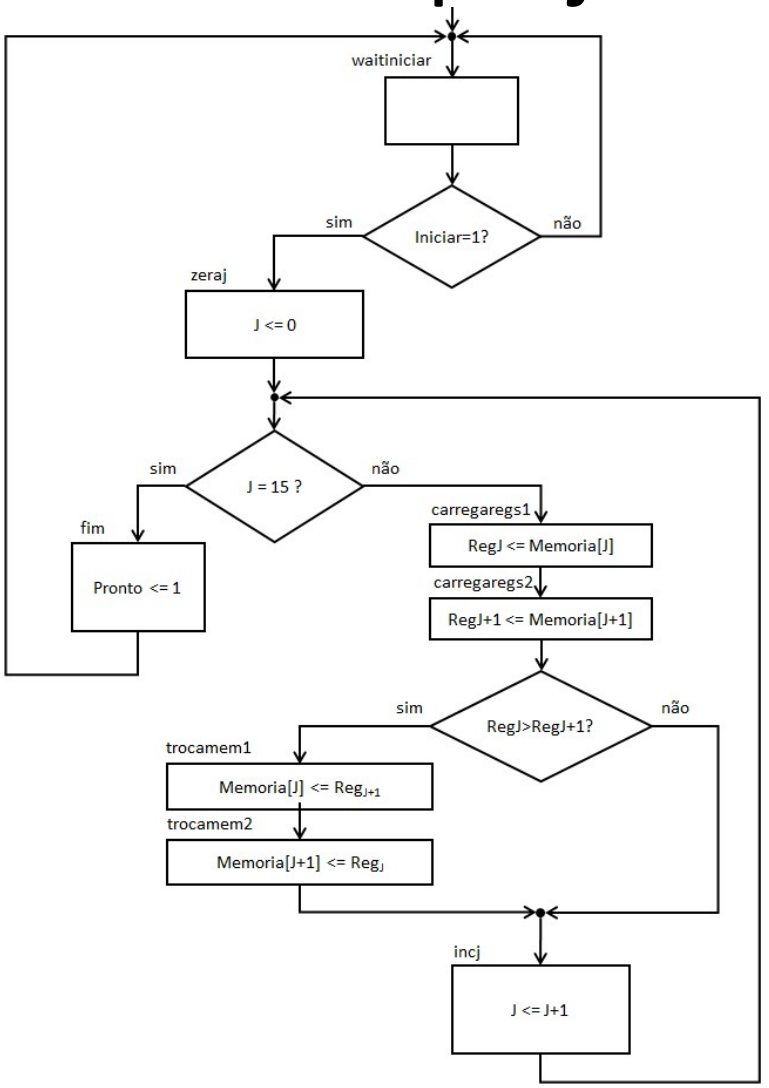
2. Troca de valores de duas posições

  - Estado para troca de valores da memória é substituído por 2 estados para armazenamento dos registradores regJ e regJ+1 na memória



Detalhamento do projeto

- Diagrama ASM de alto nível final



Detalhamento do projeto

- Elementos do Fluxo de Dados
  - [Memória externa]
  - Contador J;
  - Registradores regJ e regJmais1;
  - Mux da entrada de dados da memória;
  - Comparador;
  - Somador para J+1;
  - Multiplexador de endereço da memória.

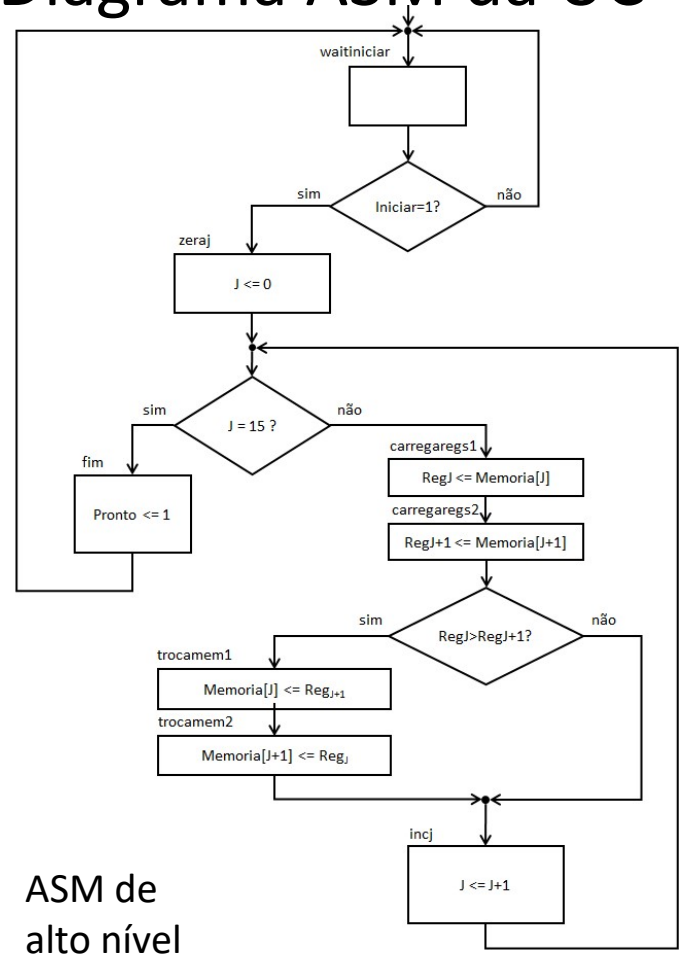
Projeto do FD
a ser realizado
e submetido

Detalhamento do projeto

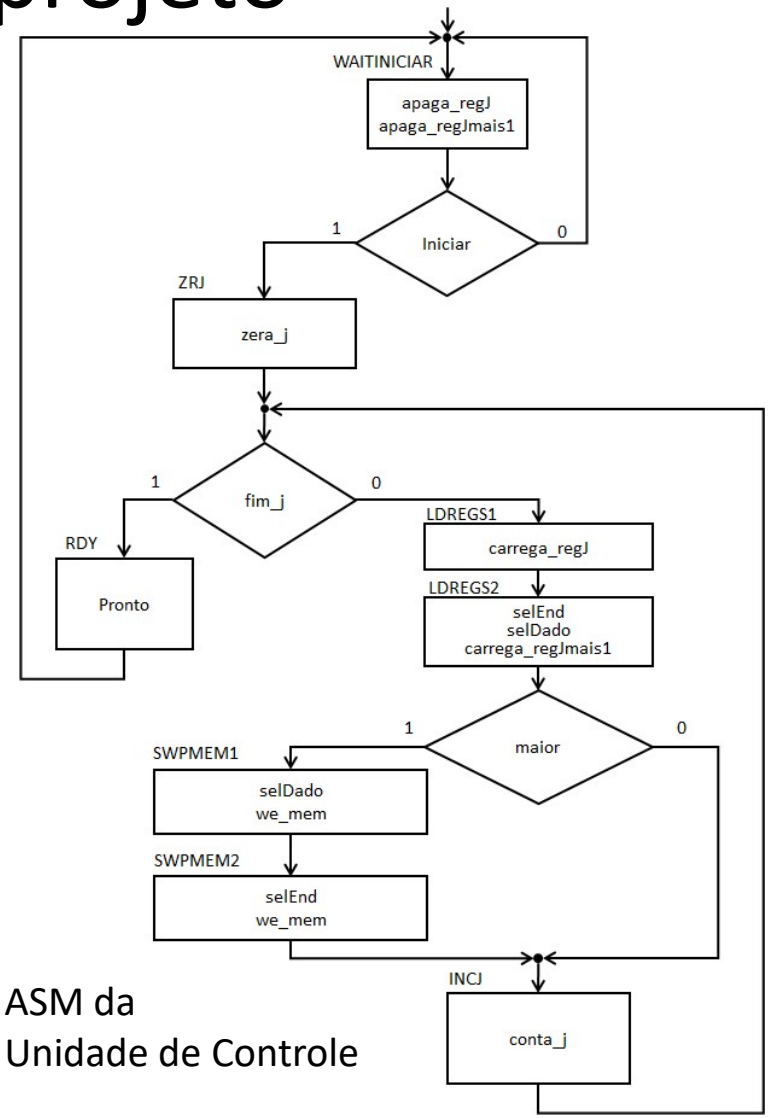
- Fluxo de Dados
  - **Arquivo** littlesort_fd_fornecido.vhd;
  - Usar **Descrição estrutural** em VHDL.

Detalhamento do projeto

- Diagrama ASM da UC



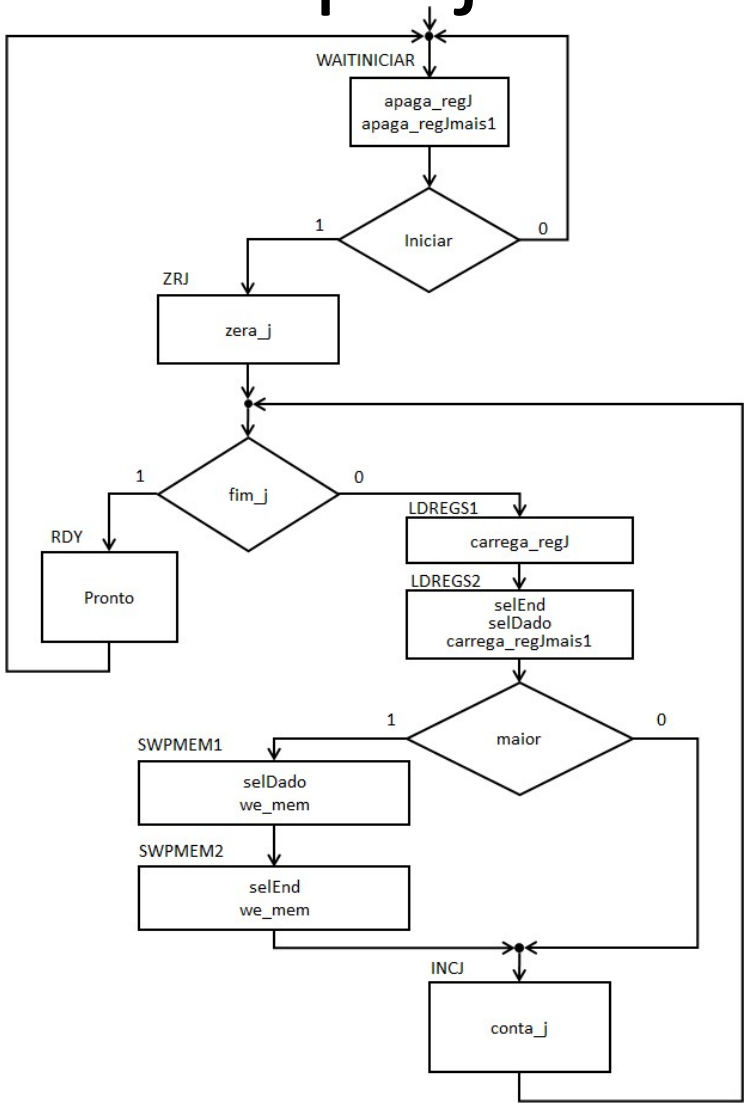
ASM de
alto nível



ASM da
Unidade de Controle

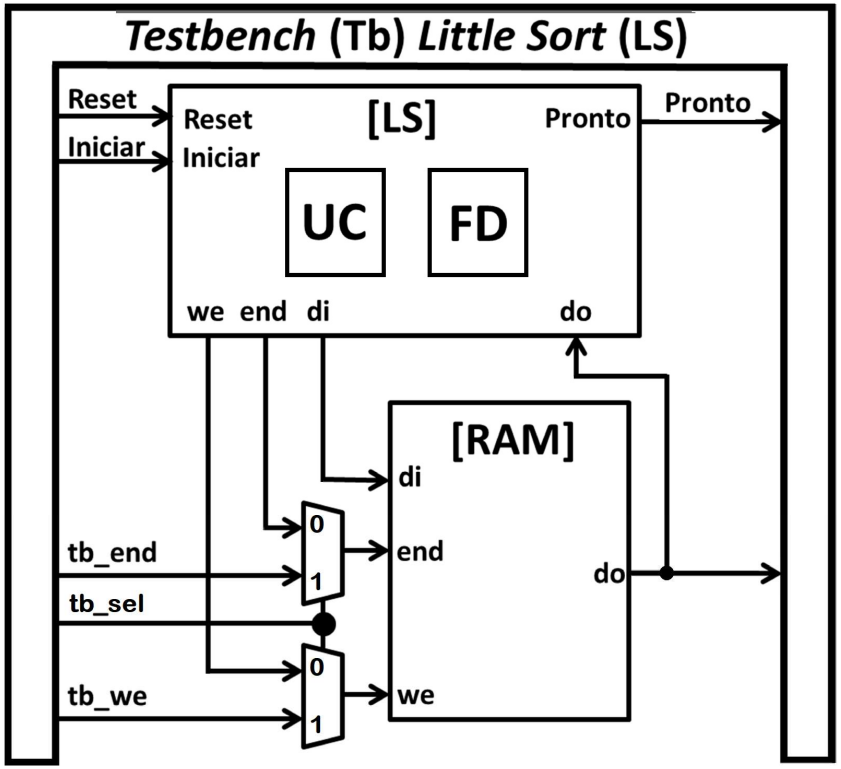
Detalhamento do projeto

- Diagrama ASM da UC
  - **Entidade** littlesort_uc



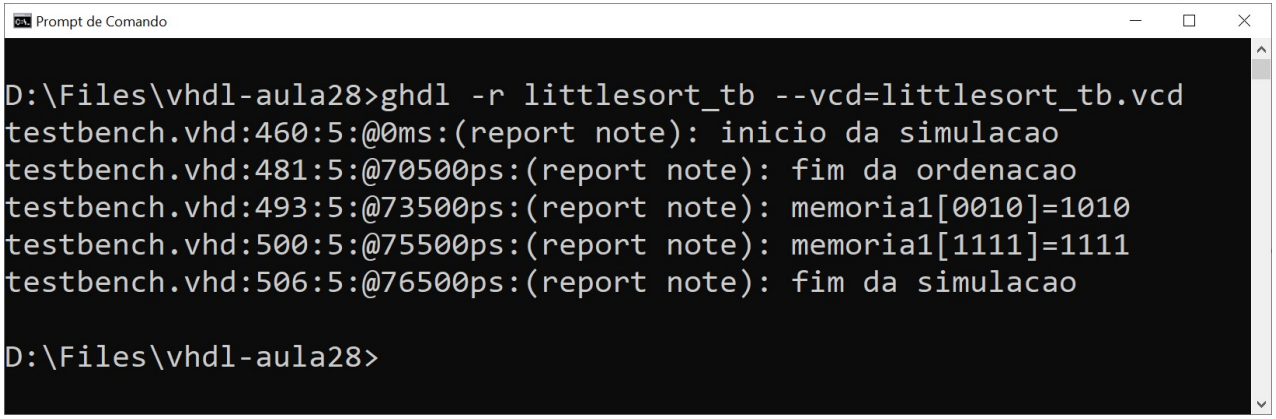
Detalhamento do projeto

- Exemplo de *Testbench* (reduzido)
  - Arquivo littlesort_testbench_uc_sd_ram_fornecido.vhd.



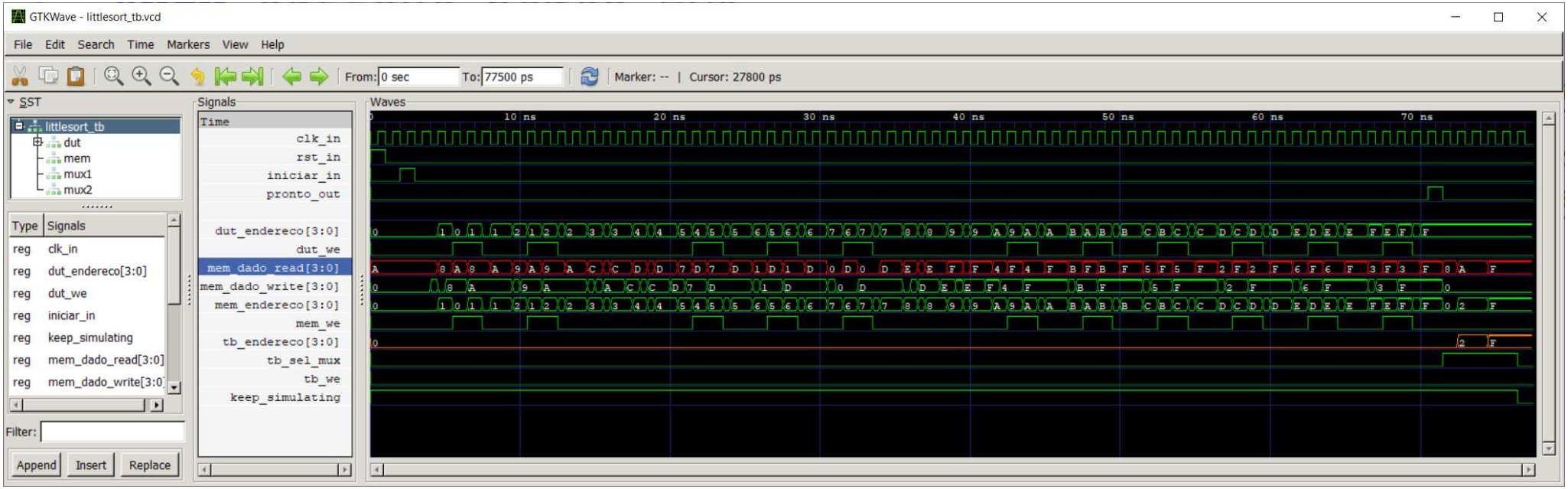
Detalhamento do projeto

- Simulação com GHDL/GTKwave



Detalhamento do projeto

- Simulação com GHDL/GTKwave



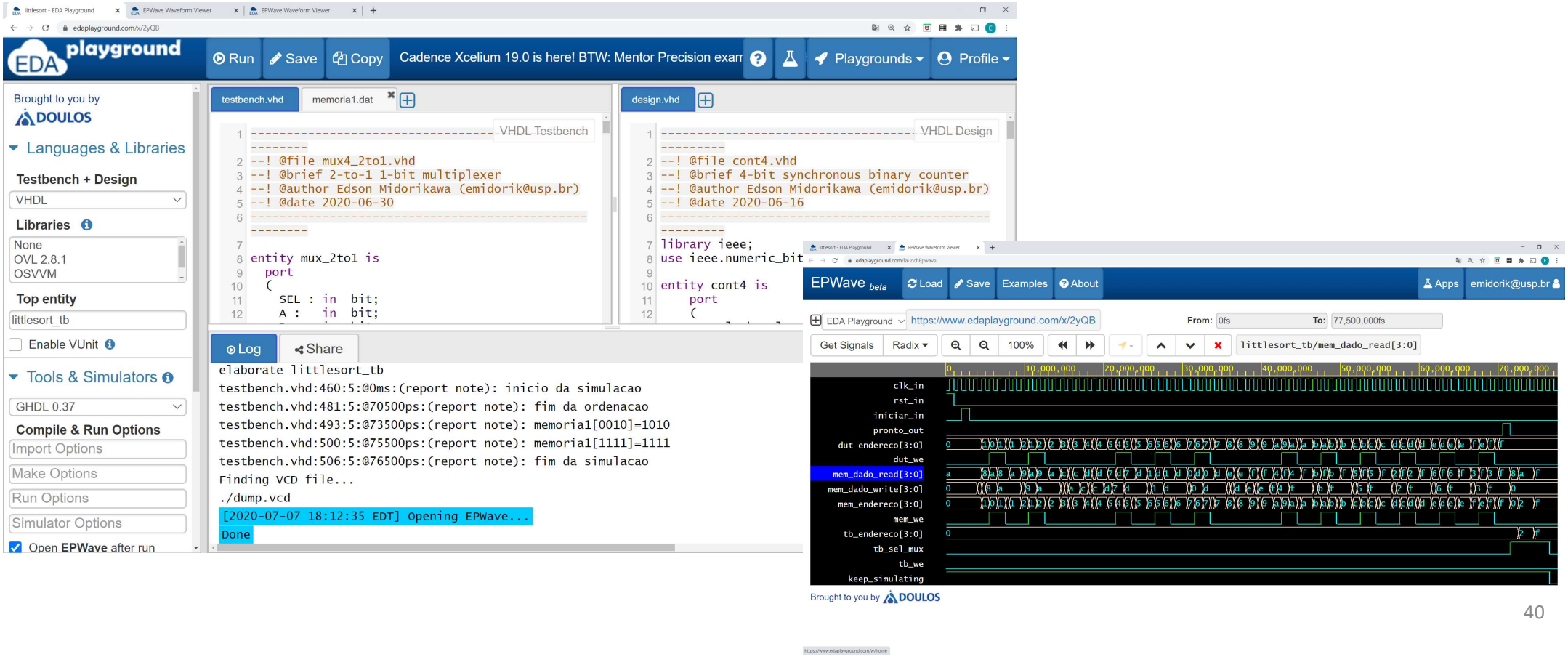
Detalhamento do projeto

- Simulação com GHDL/GTKwave

Valores finais das posições 2 e F da memória após o final da ordenação

Detalhamento do projeto

- Simulação com EDA Playground



Perguntas

Dúvidas?

Pode ligar o áudio/microfone ou perguntar via chat.

Obrigado pela participação na aula!