



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.

Projetos VHDL Judge

Projeto 1 VHDL (Resto da Divisão) - Enunciado

Projeto 1 - Resto da Divisão (Prazo 8 de julho, quarta-feira, 23:59)

Projeto 2 VHDL (Log2) - Enunciado

Projeto 2 - log2 (Prazo 15 de julho, quarta-feira, 23:59)

Projeto 3 VHDL (Controle Turbo) - Enunciado

Projeto 3 - Turbo (Prazo 22 de julho, quarta-feira, 23:59)

Projeto 4 VHDL (Little Sort) - Enunciado





Projeto 4 VHDL (LittleSort) - littlesort_fd_fornecido.vhd

Projeto 4 VHDL (LittleSort) - littlesort_testbench_uc_sd_ram_fornecido.zip

Projeto 4 - LittleSort (Prazo 29 de julho, quarta-feira, 23:59)

Visão geral do Projeto

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

-  Projeto 4 VHDL (Little Sort) - Enunciado
-  Projeto 4 VHDL (LittleSort) - littlesort_fd_fornecido.vhd
-  Projeto 4 VHDL (LittleSort) - littlesort_testbench_uc_sd_ram_fornecido.zip
-  Projeto 4 - LittleSort (Prazo 29 de julho, quarta-feira, 23:59)

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;
        }
}
```

0	1	2	3	4	5	6	7
5	1	4	7	0	2	99	3

n=8

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 }

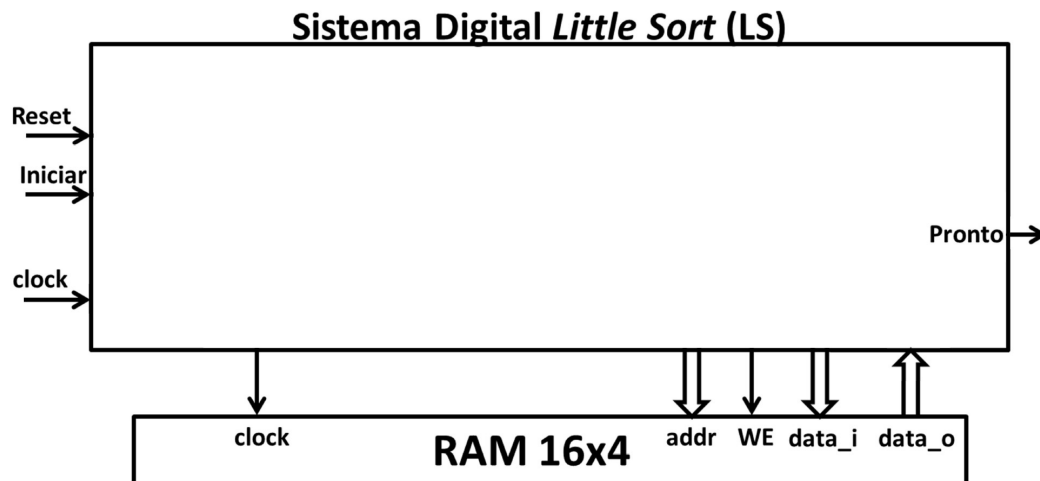
memoria final: { 1, 4, 5, 0, 2, 7, 3, 99 }

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

Outros elementos também são reordenados durante o processamento.

Visão geral do Projeto

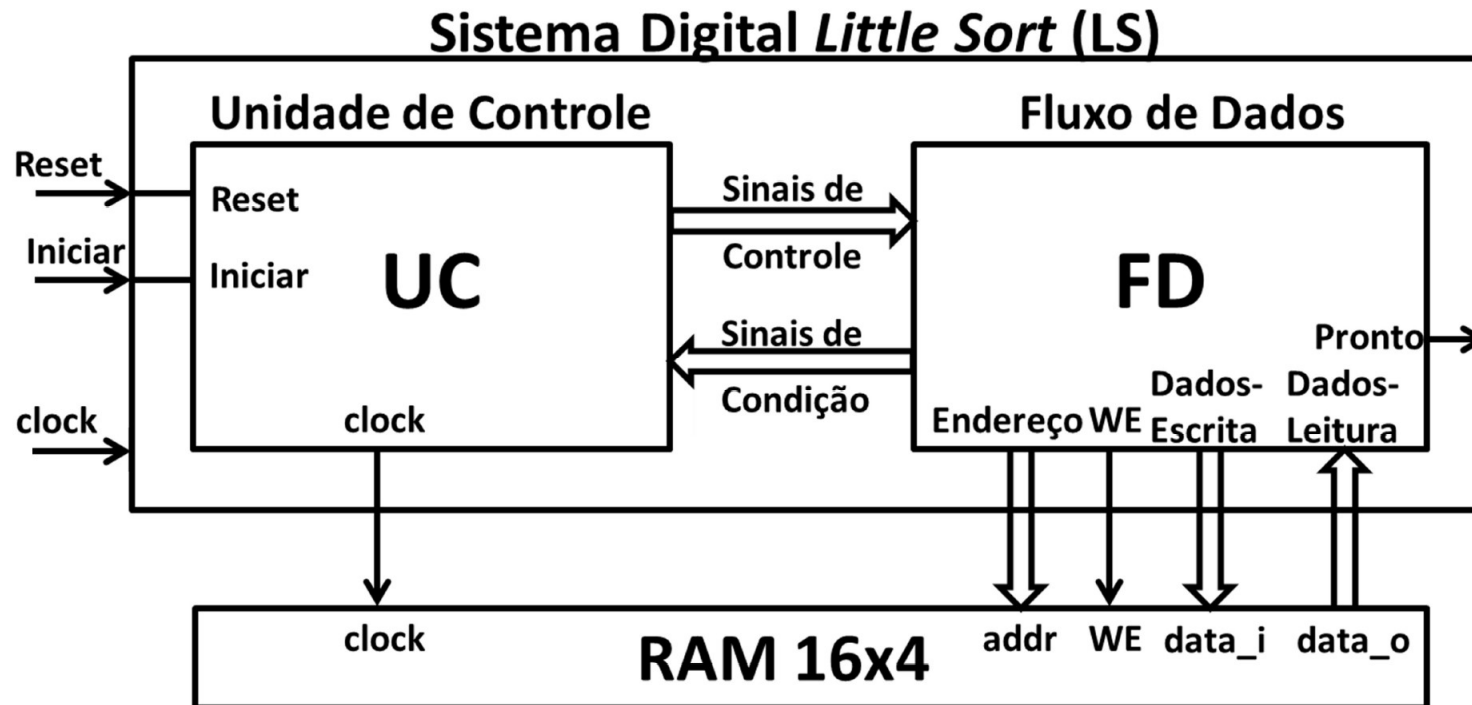
- Diagrama de blocos do Sistema Digital



```
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)



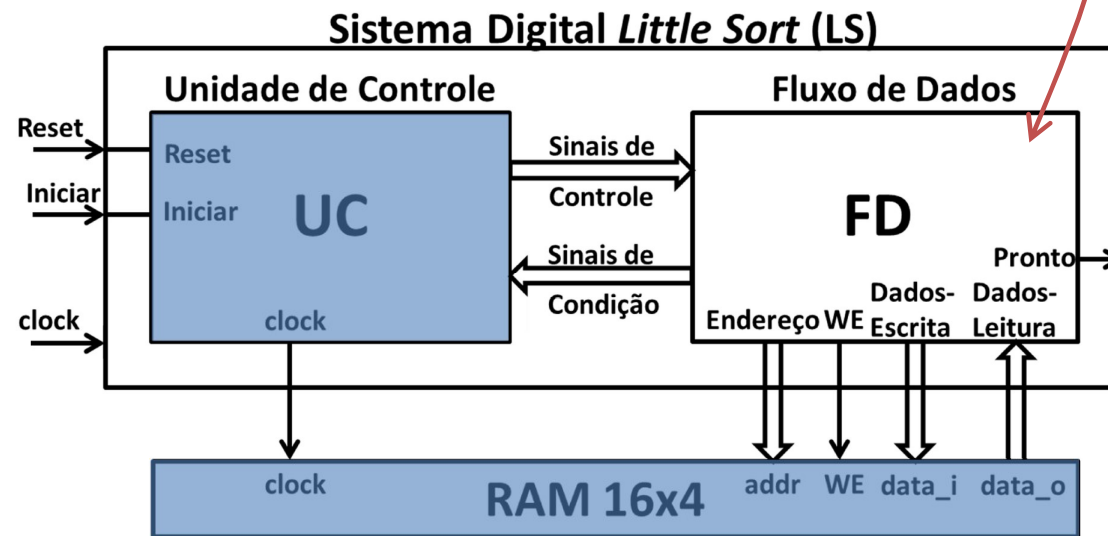
O que é fornecido aos alunos

- O que é fornecido aos alunos

- 2 arquivos VHDL:

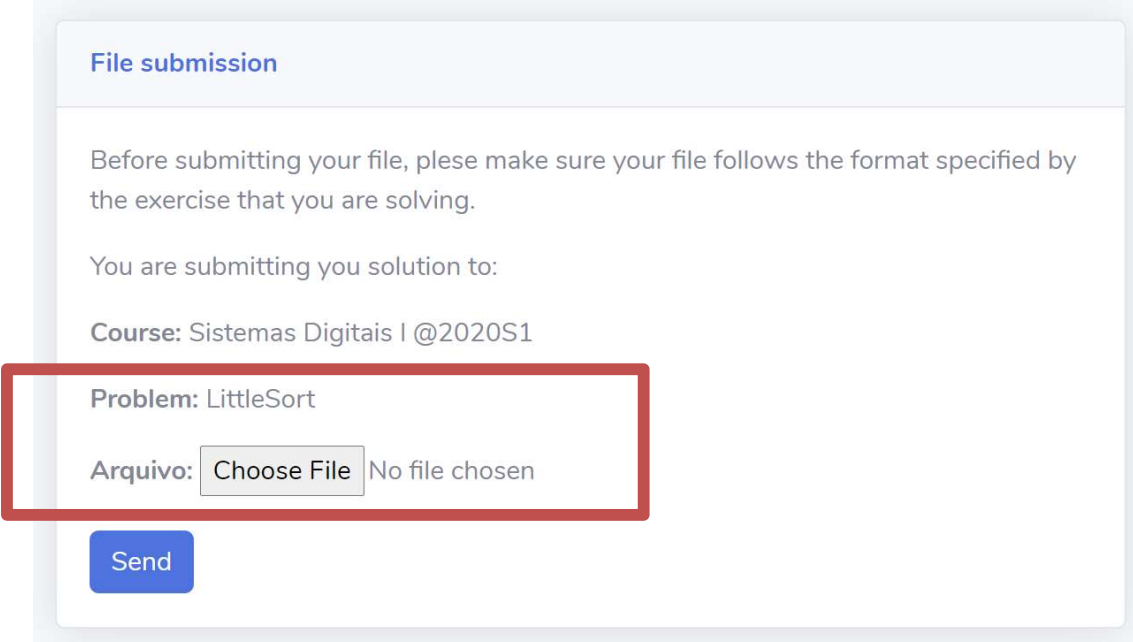
- littlesort_fd_fornecido.vhd
- littlesort_testbench_uc_sd_ram_fornecido.vhd
- memoria1.dat (conteúdo da RAM16x4))

Parcial
(completar)



O que submeter ao Juiz

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



The screenshot shows a web interface for file submission. It has a title 'File submission' in blue. Below the title, there is a paragraph: 'Before submitting your file, please make sure your file follows the format specified by the exercise that you are solving.' followed by 'You are submitting your solution to:'. Then, it says 'Course: Sistemas Digitais I @2020S1'. Below that, 'Problem: LittleSort' is displayed. Under 'Arquivo:', there is a 'Choose File' button and the text 'No file chosen'. At the bottom, there is a blue 'Send' button. A red rectangular box highlights the 'Problem: LittleSort' and the 'Arquivo:' section.

File submission

Before submitting your file, please make sure your file follows the format specified by the exercise that you are solving.

You are submitting your solution to:

Course: Sistemas Digitais I @2020S1

Problem: LittleSort

Arquivo: No file chosen

O que submeter ao Juiz

- Submissão ao Juiz

File submission

Before submitting your file, please make sure your file follows the format specified by the exercise that you are solving.

You are submitting your solution to:

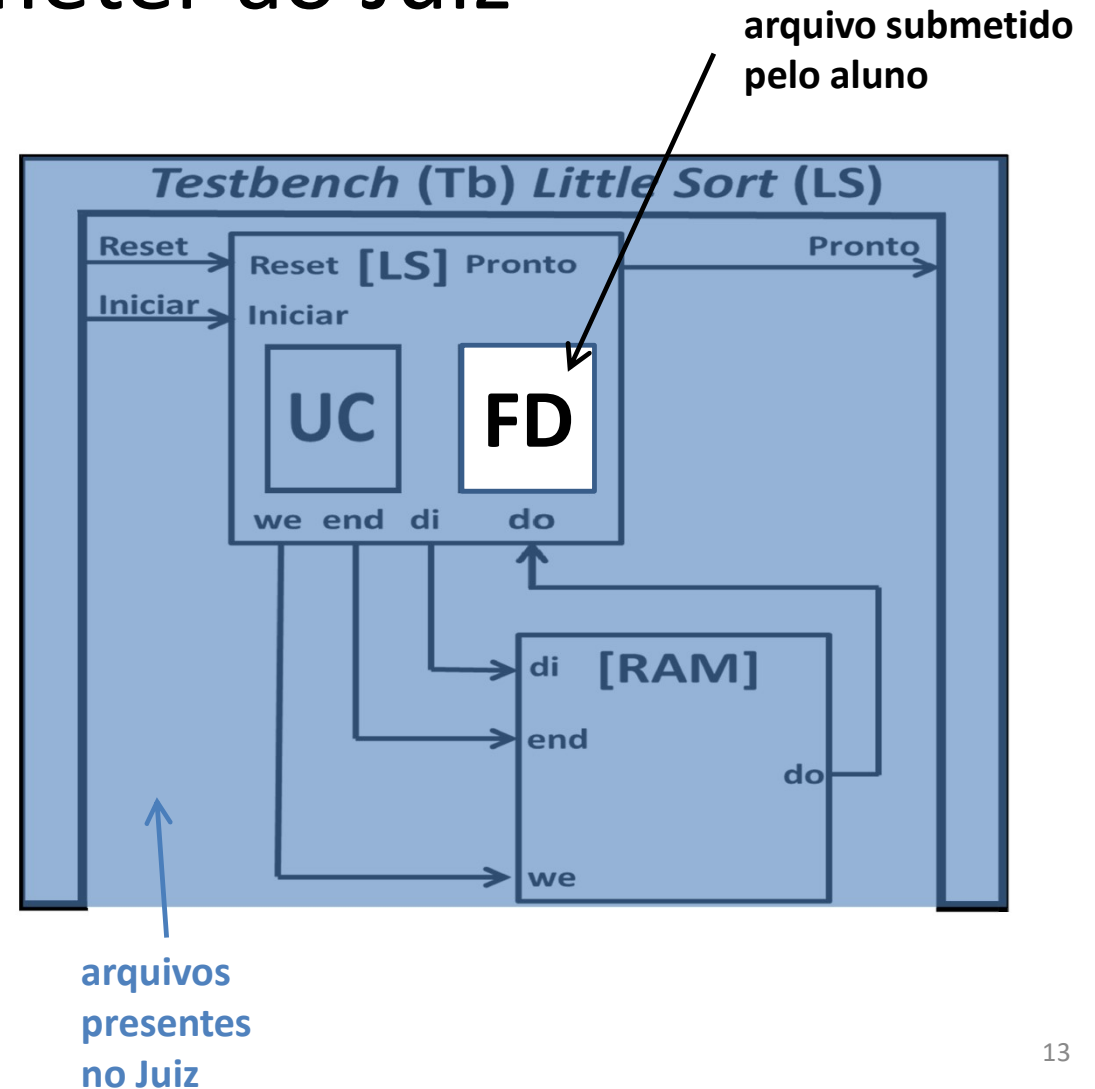
Course: Sistemas Digitais I @2020S1

Problem: LittleSort

Arquivo: No file chosen

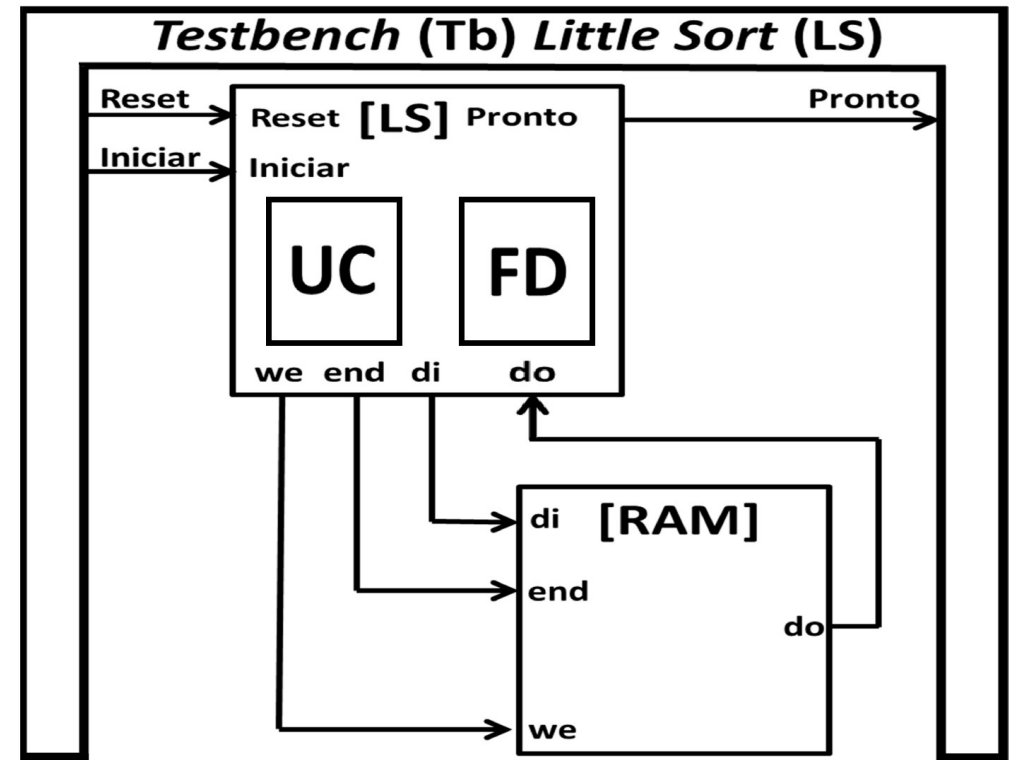
O que submeter ao Juiz

- 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:

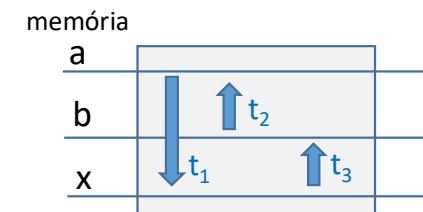
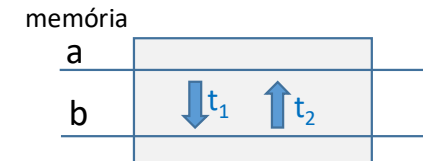
```
1: x = a;
2: a = b;
3: b = x
```

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:

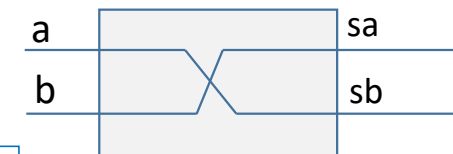
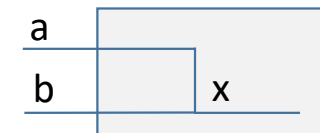
```
1: x <= a;
2: a <= b;
3: b <= x;
```

Em VHDL, isso funciona:

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

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

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



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:

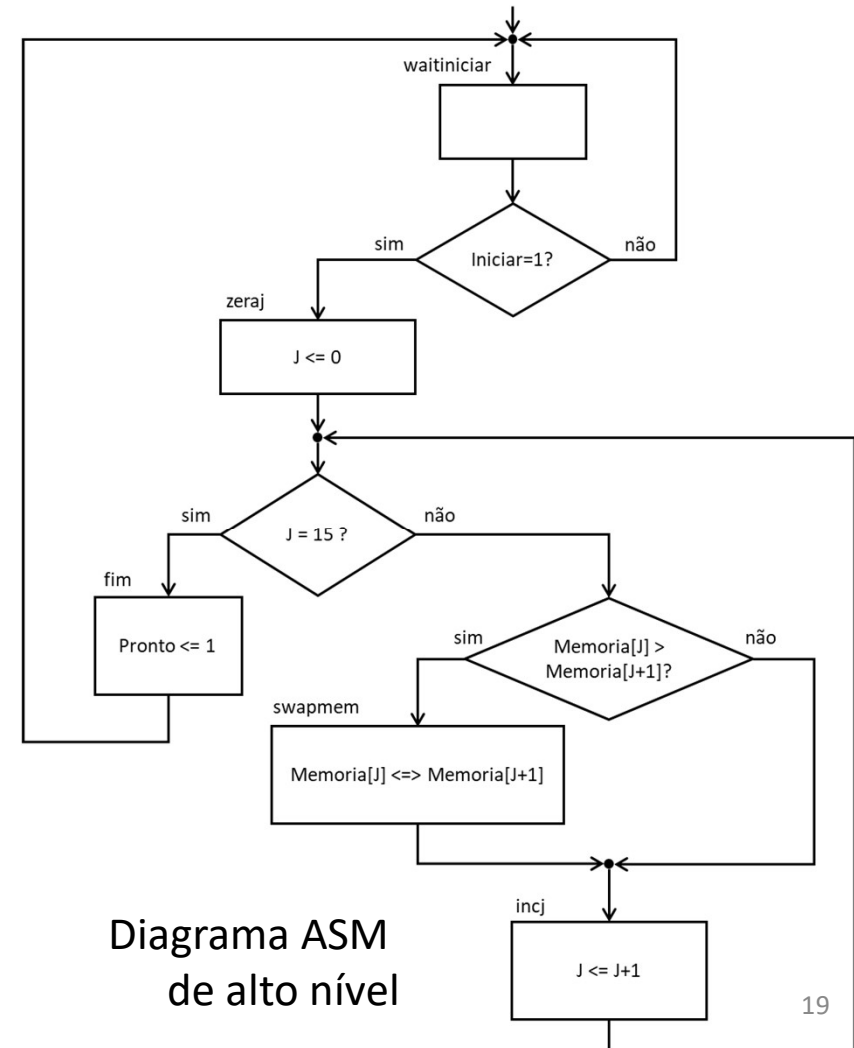
```
// reordena vetor a com n elementos
algoritmo LittleSort (vetor a[], inteiro n)
{
    inteiro j;
    for (j = 0; j < n-1; j++) {
        if (a[j] > a[j + 1]) {
            troca valores entre a[j] e a[j + 1];
        }
    }
}
```

Detalhamento do projeto

- Algoritmo e Diagrama ASM

Pseudocódigo:

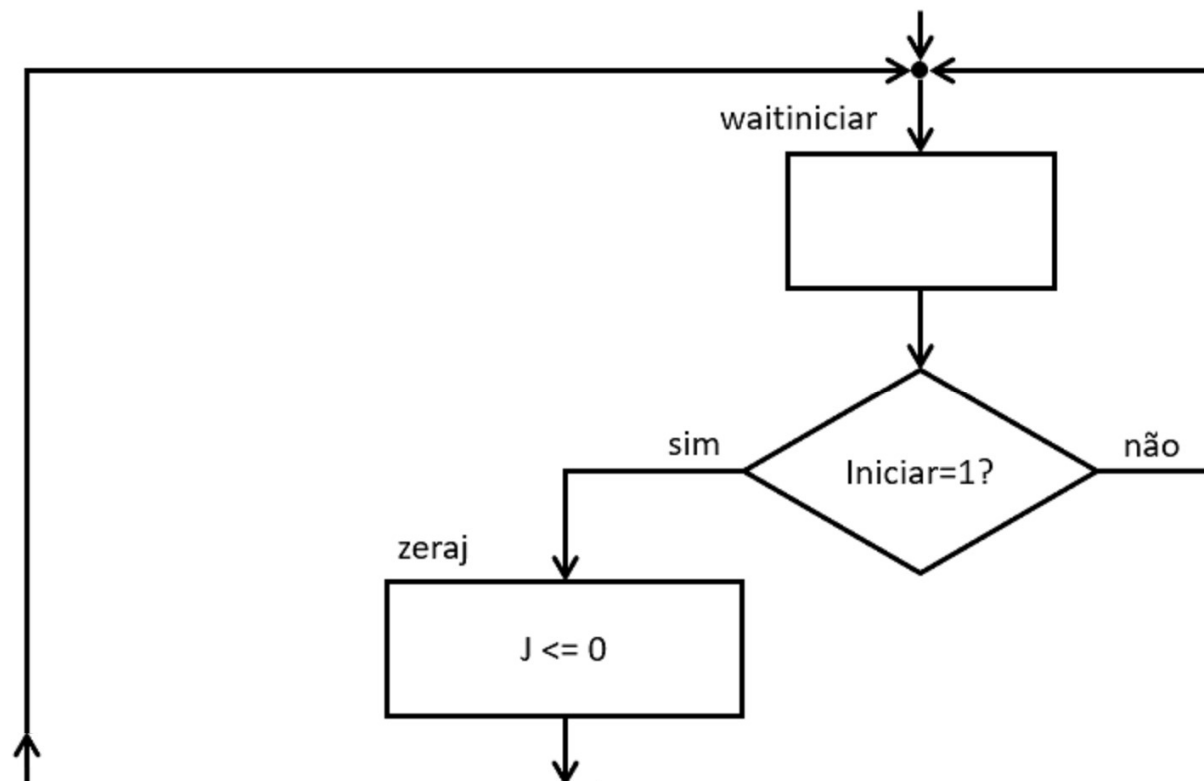
```
// reordena vetor a com n elementos
algoritmo LittleSort (vetor a[], inteiro n)
{
    inteiro j;
    for (j = 0; j < n-1; j++) {
        if (a[j] > a[j + 1]) {
            troca valores entre a[j] e a[j + 1];
        }
    }
}
```



Detalhamento do projeto

- Algoritmo e Diagrama ASM

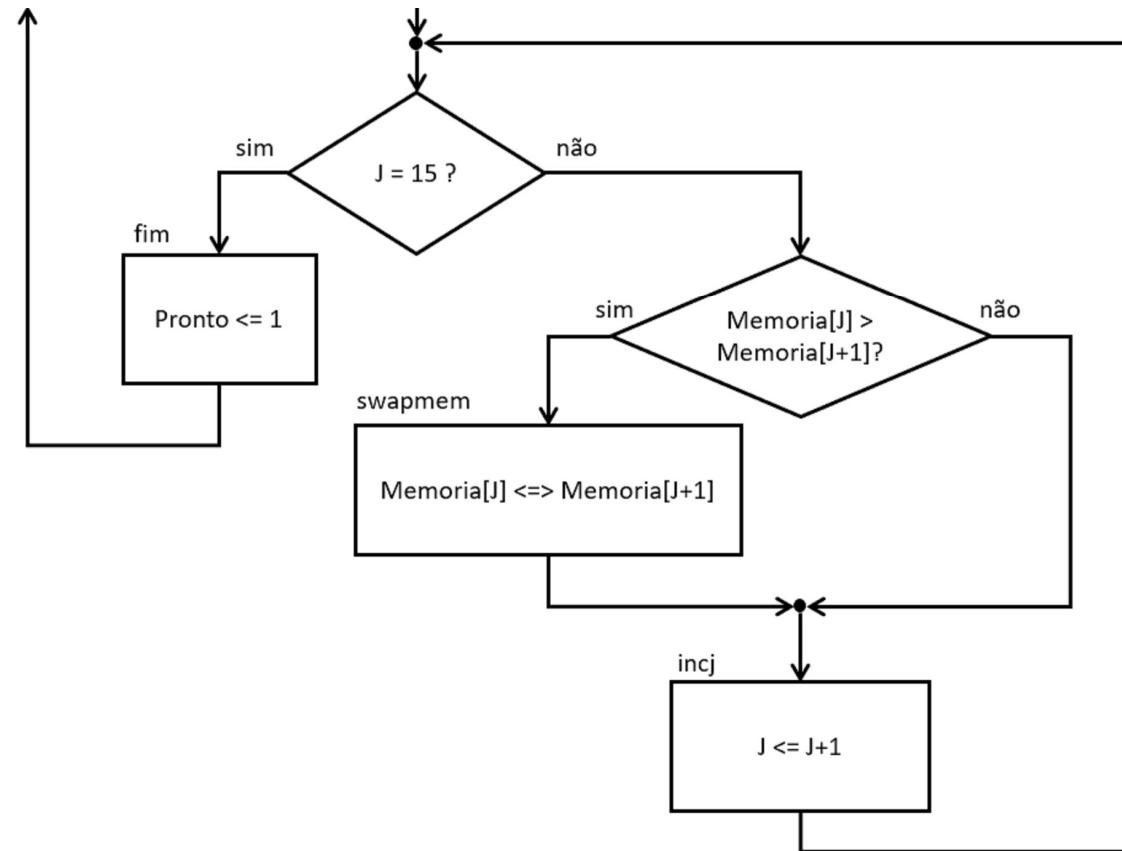
Diagrama ASM
de alto nível
(parte 1)



Detalhamento do projeto

- Algoritmo e Diagrama ASM

Diagrama ASM
de alto nível
(parte 2)

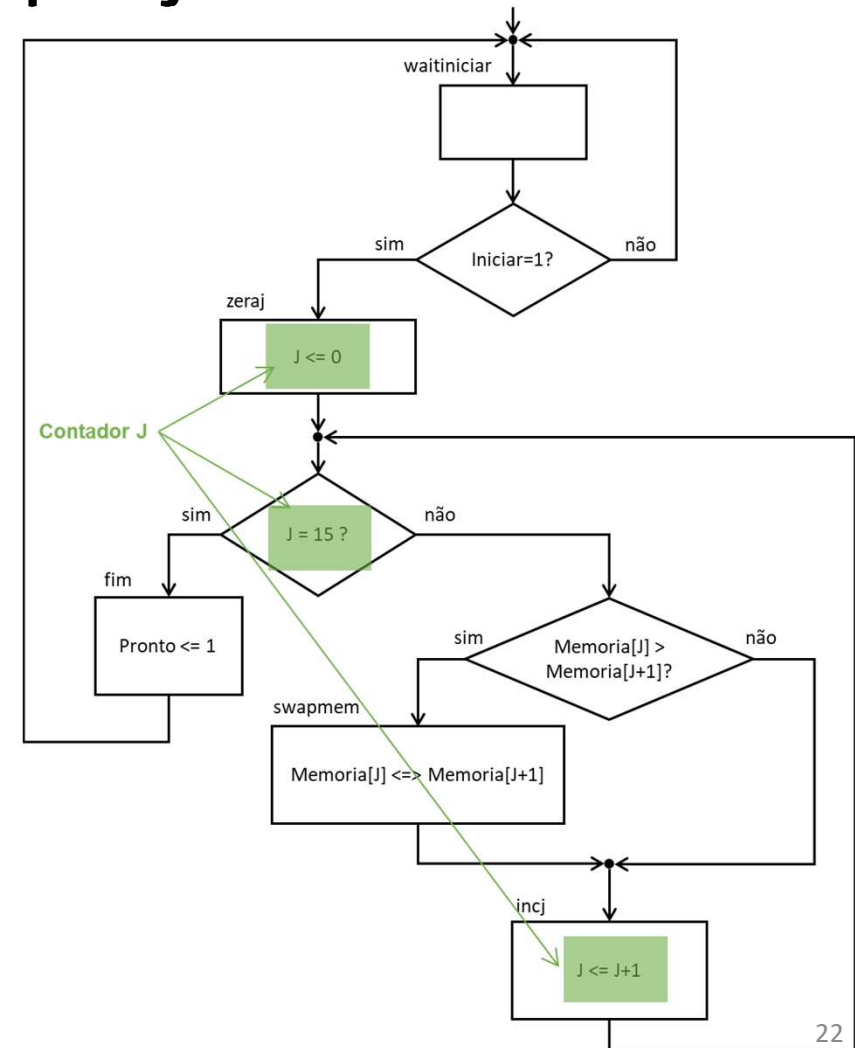


Detalhamento do projeto

- Elementos do Fluxo de Dados

– Identificação a partir do Diagrama ASM

1. Memória
2. Contador J

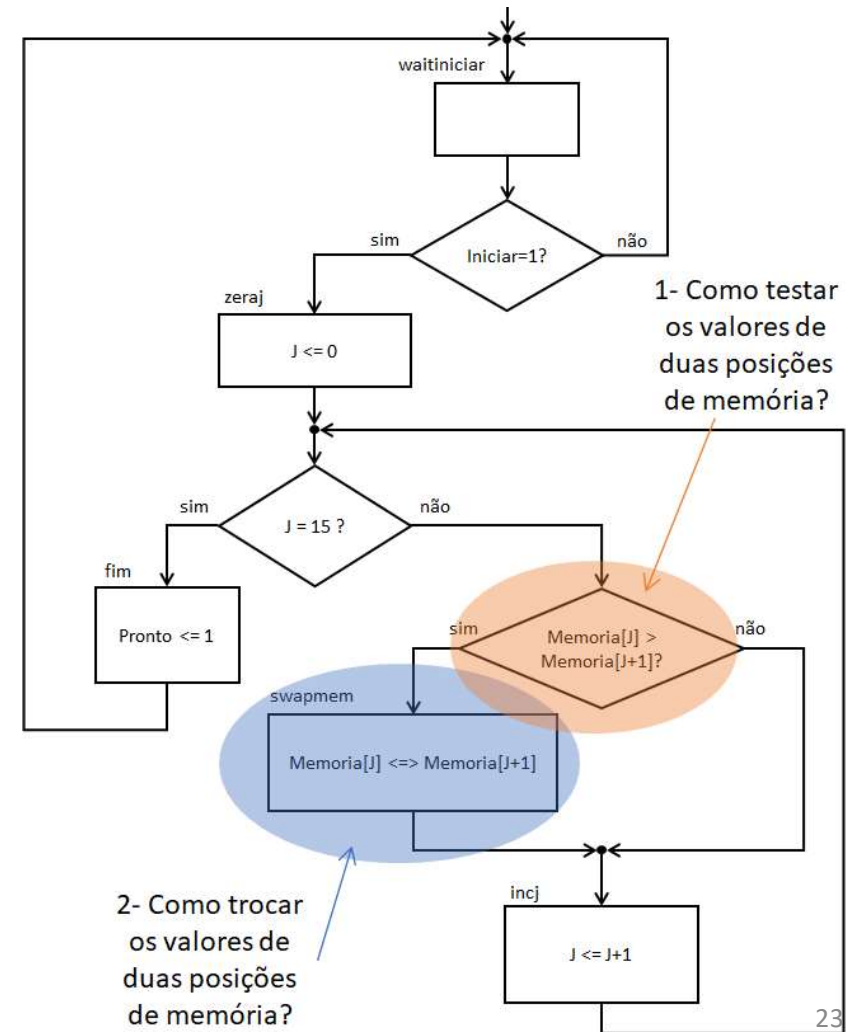


Detalhamento do projeto

- 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**

endereço $\leq p$

we ≤ 0

valor lido em saída

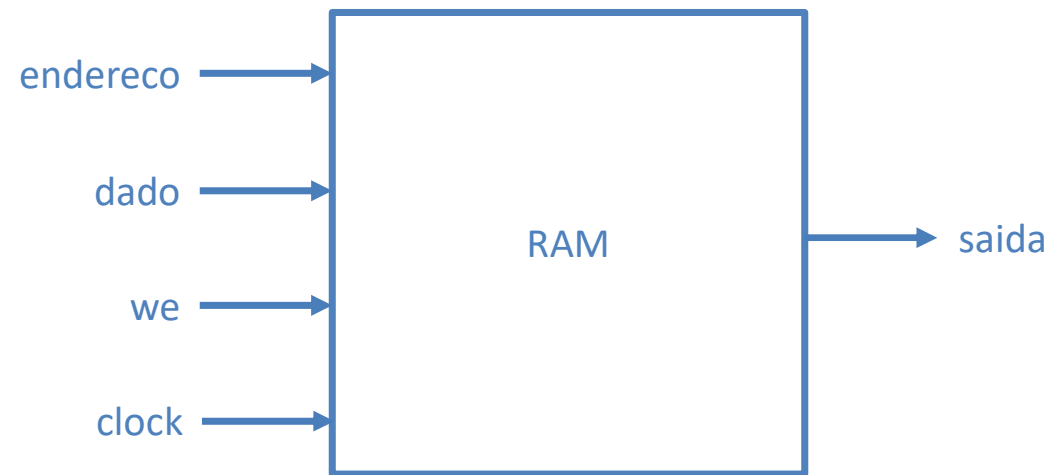
2. Escrita do valor **d** na posição **p**

endereço $\leq p$

we ≤ 1

dado $\leq d$

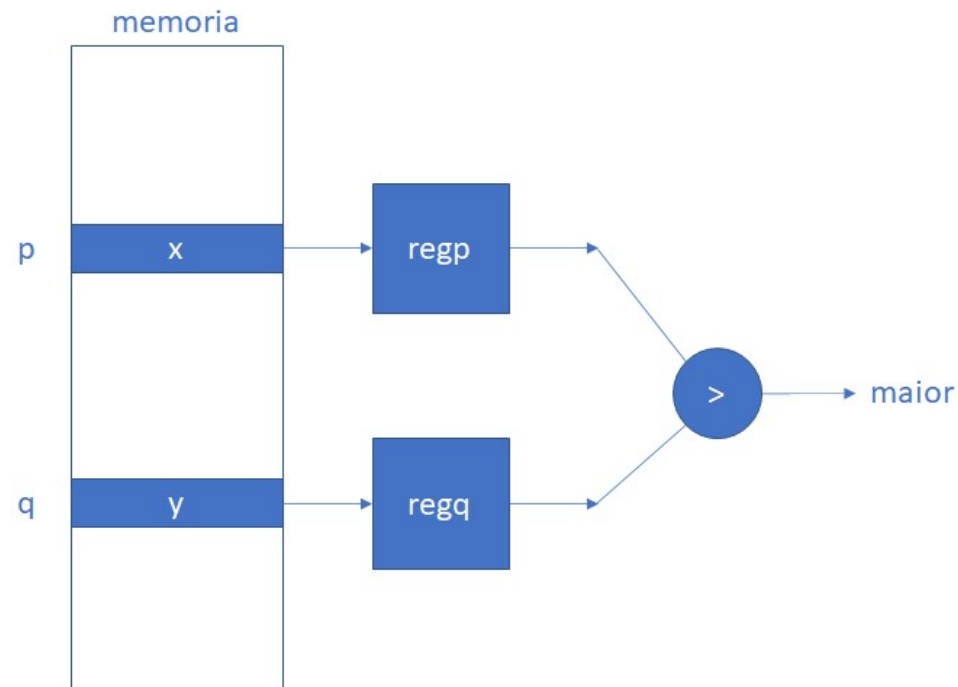
operação realizada na borda do *clock*



Acesso de apenas uma posição por vez.

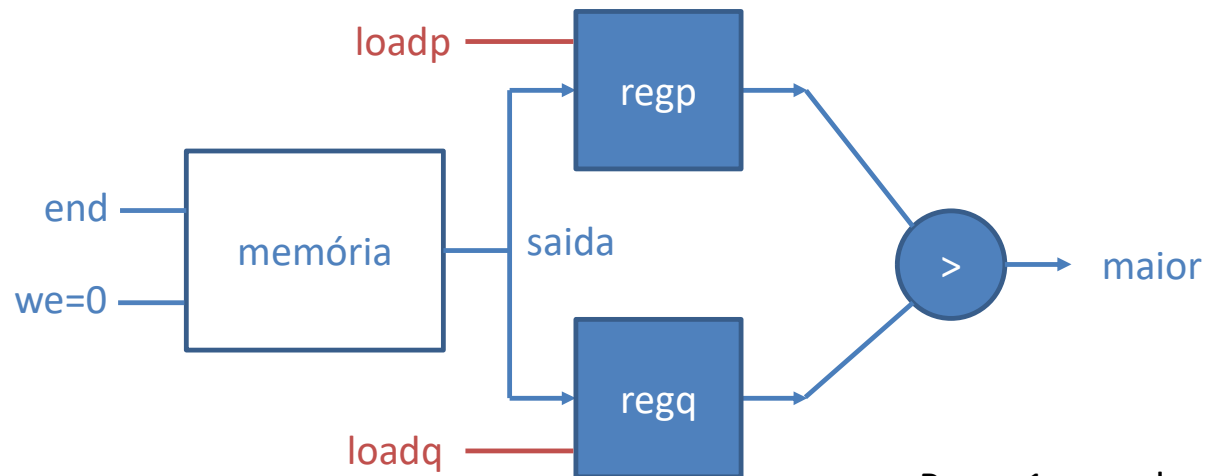
Detalhamento do projeto

- 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 memória[p] em regp (**loadp=1**)

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

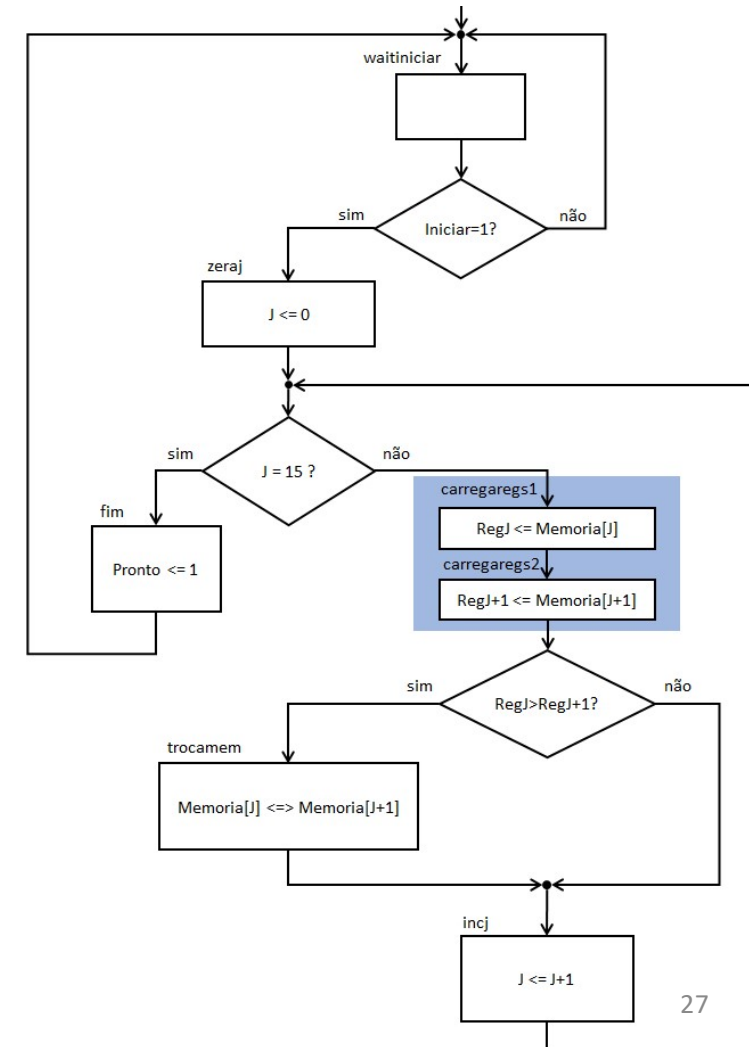
Passo 3: comparar valores de regp e regq

Detalhamento do projeto

- Detalhamento do Diagrama ASM

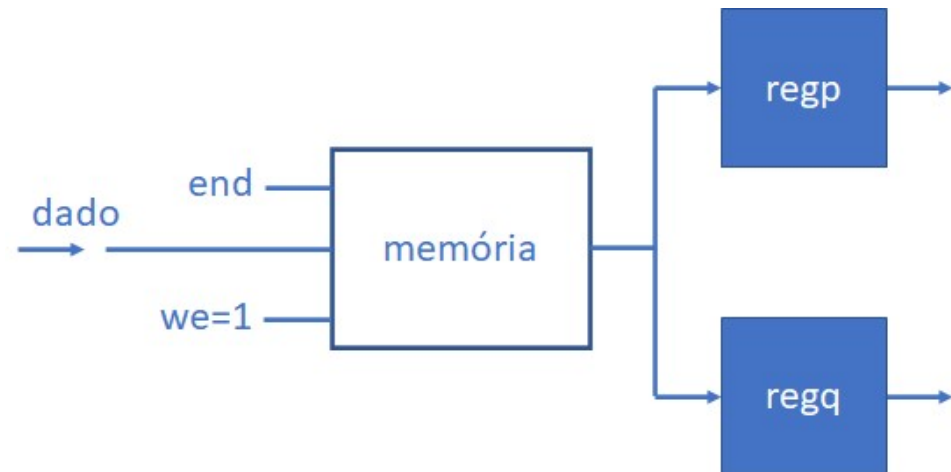
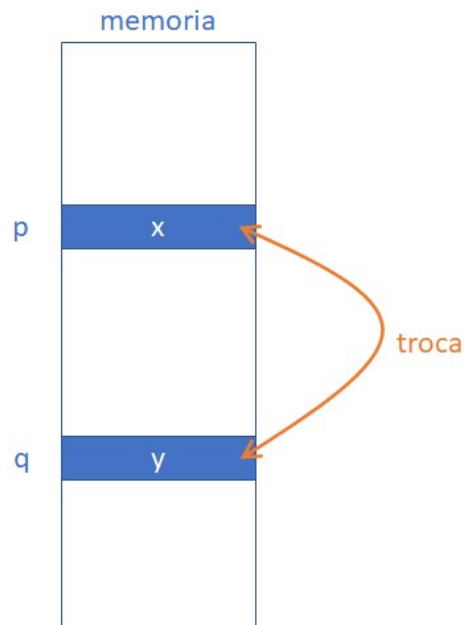
1. Comparação do conteúdo de duas posições

- 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



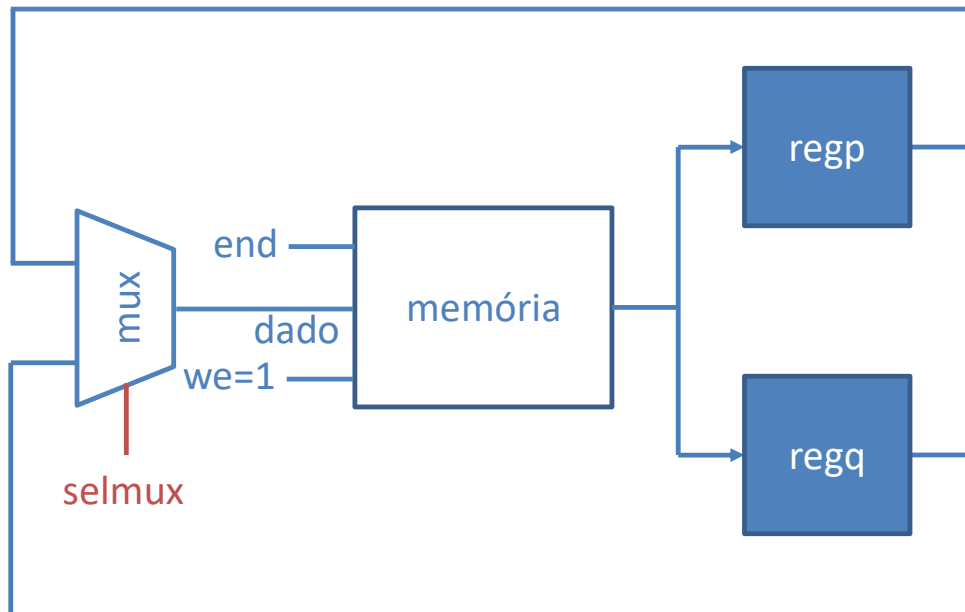
Detalhamento do projeto

- 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**)

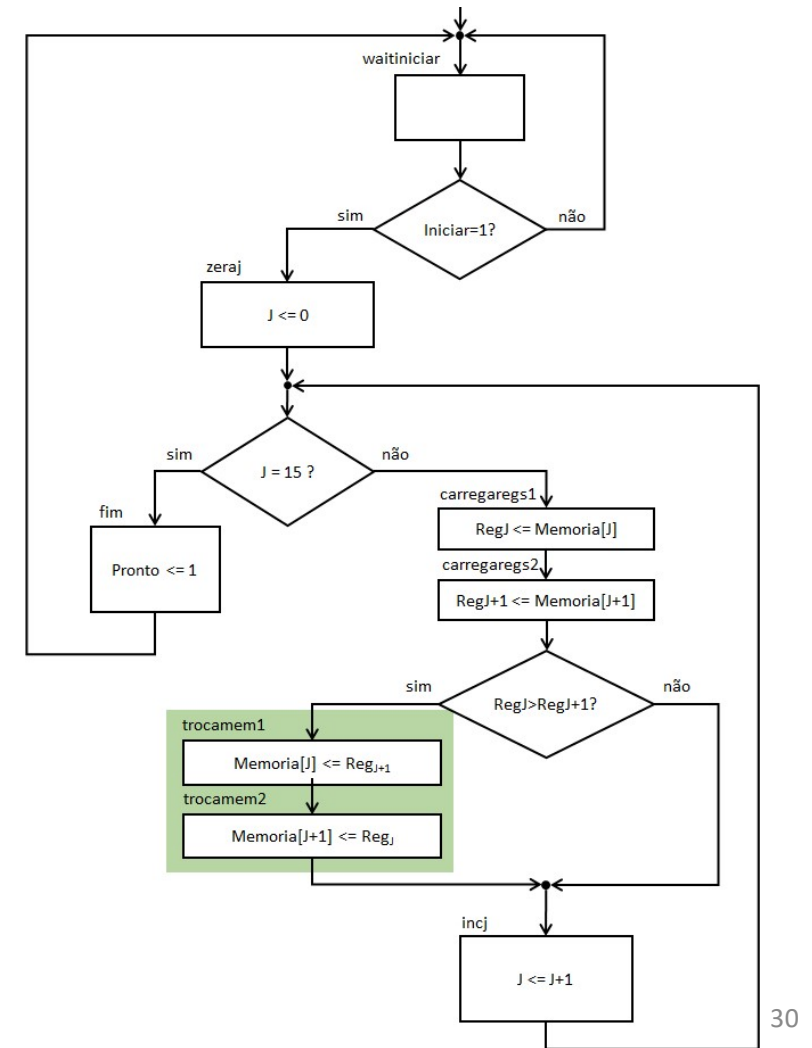
Passo 2: guardar valor de regq na posição p (**selmux=1**)

Detalhamento do projeto

- Detalhamento do Diagrama ASM

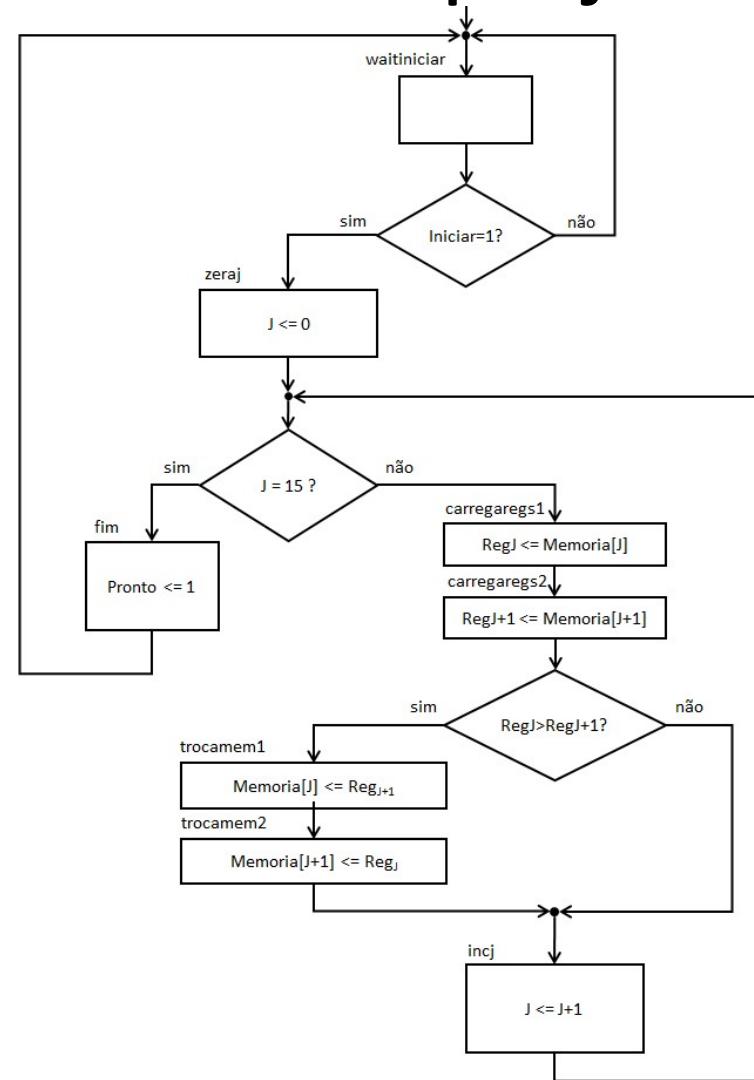
2. Troca de valores de duas posições

- 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.

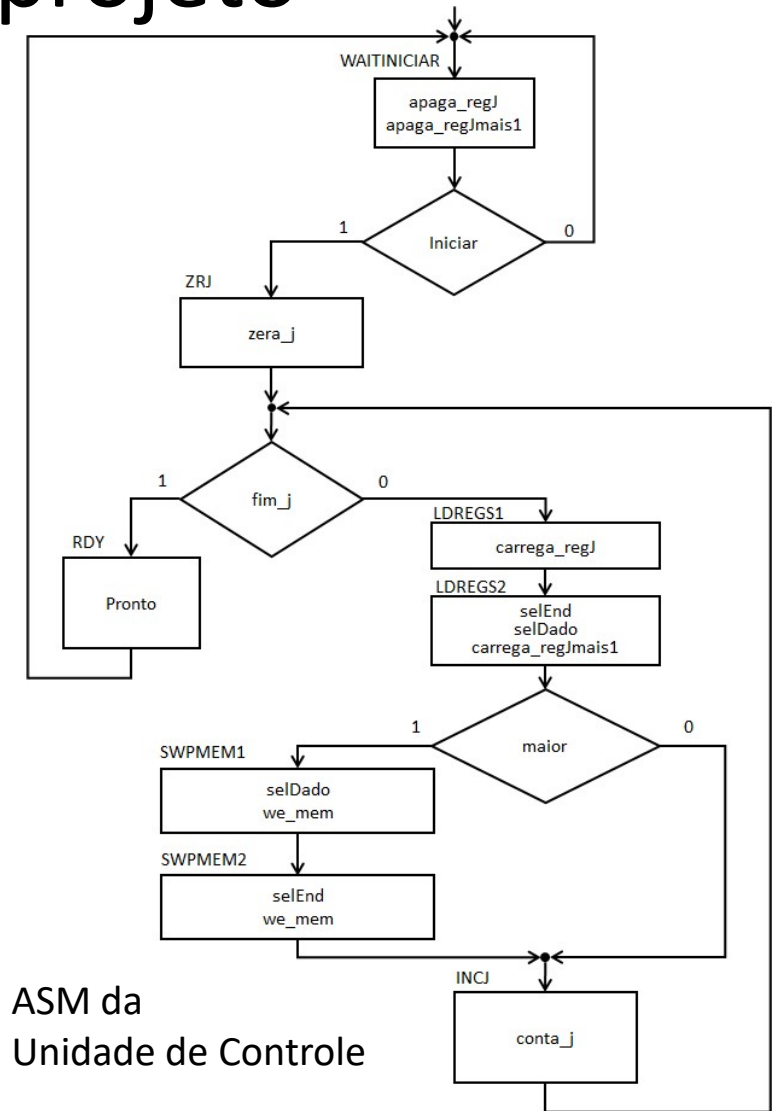
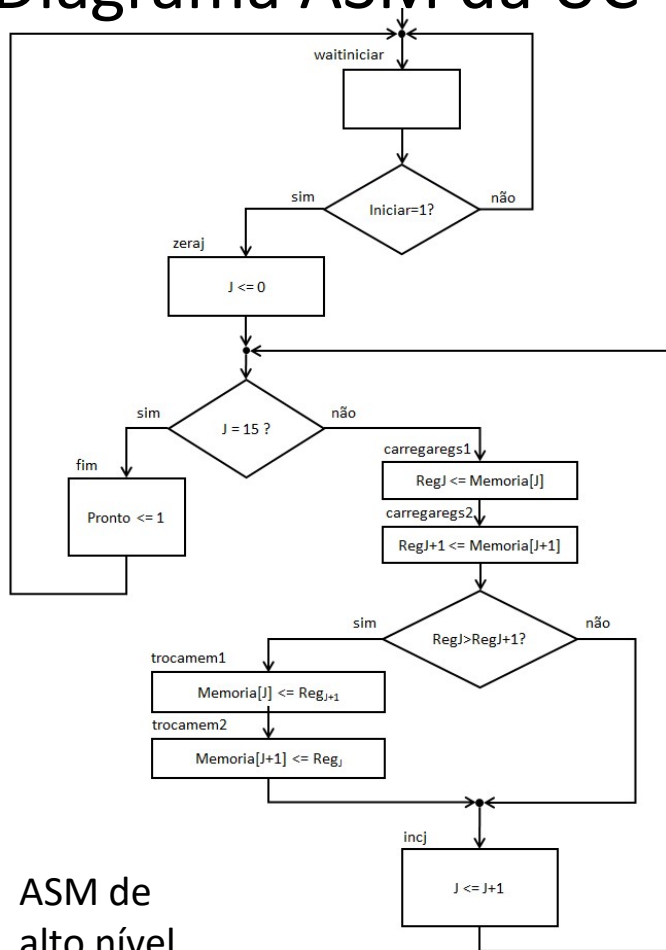
Projeto do FD
a ser realizado
e submetido

Detalhamento do projeto

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

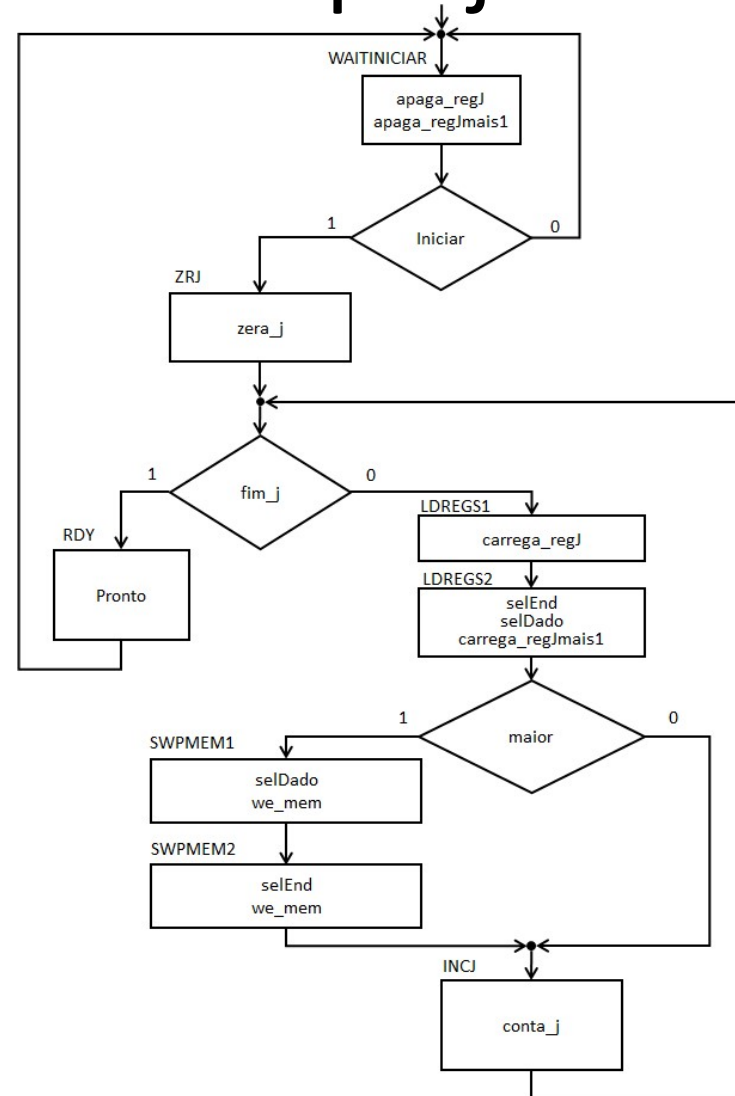
Detalhamento do projeto

- Diagrama ASM da UC



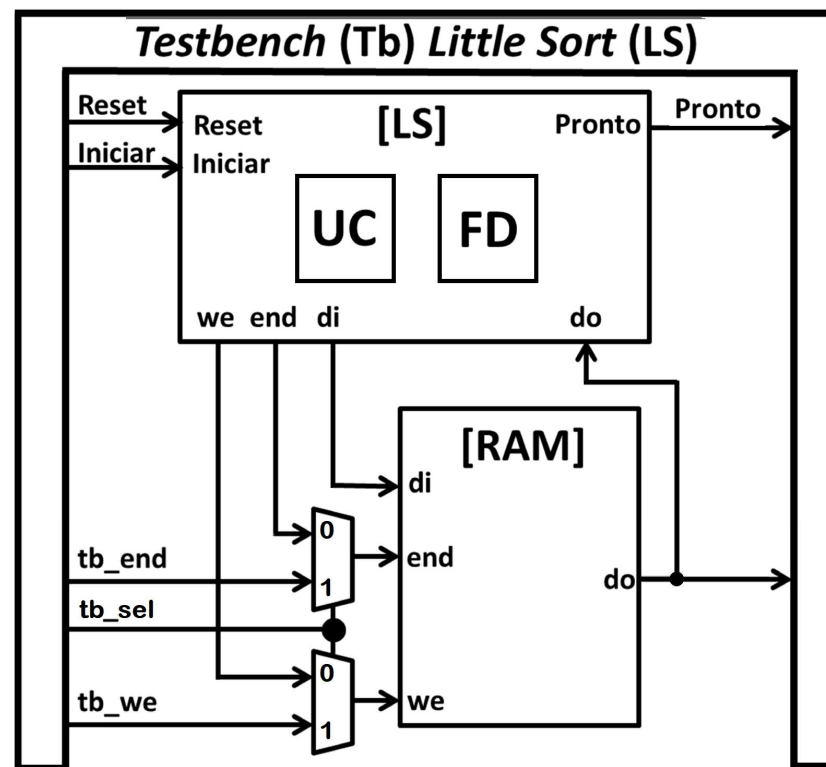
Detalhamento do projeto

- Diagrama ASM da UC
 - Entidade littlesort_uc



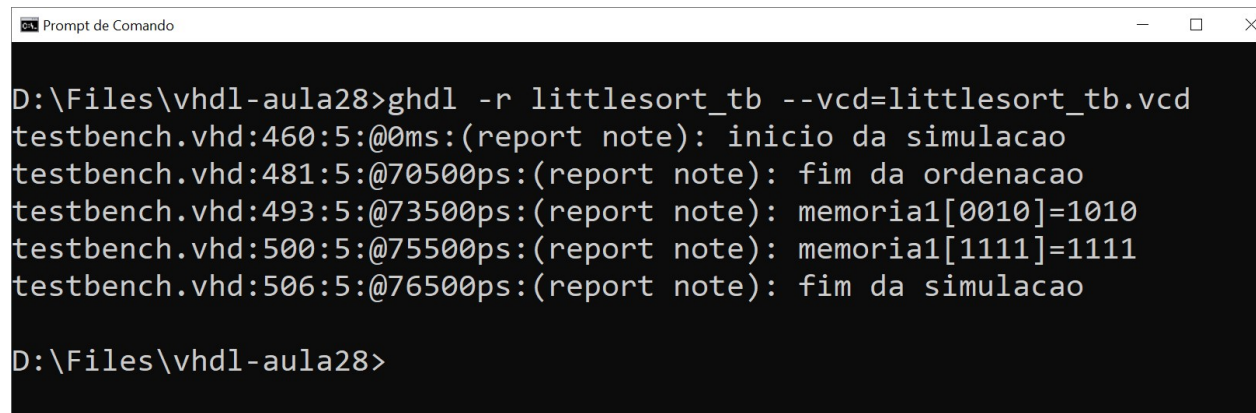
Detalhamento do projeto

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



Detalhamento do projeto

- Simulação com GHDL/GTKwave



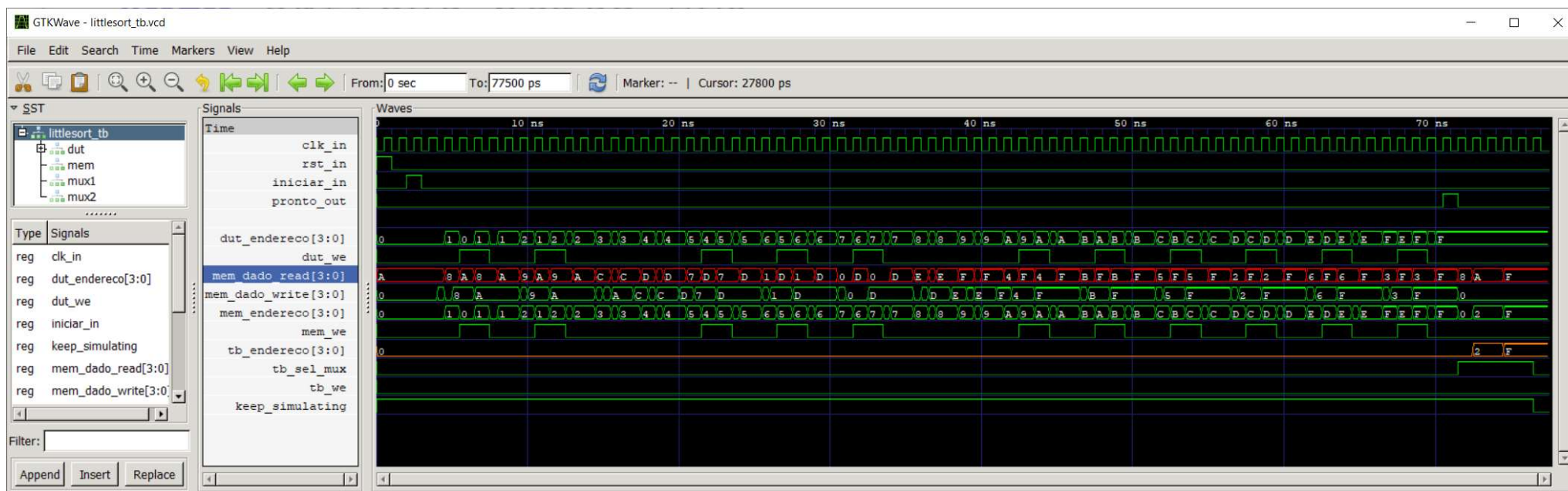
```
Prompt de Comando

D:\Files\vhdl-aula28>ghdl -r littlesort_tb --vcd=littlesort_tb.vcd
testbench.vhd:460:5:@0ms:(report note): inicio da simulacao
testbench.vhd:481:5:@70500ps:(report note): fim da ordenacao
testbench.vhd:493:5:@73500ps:(report note): memoria1[0010]=1010
testbench.vhd:500:5:@75500ps:(report note): memoria1[1111]=1111
testbench.vhd:506:5:@76500ps:(report note): fim da simulacao

D:\Files\vhdl-aula28>
```

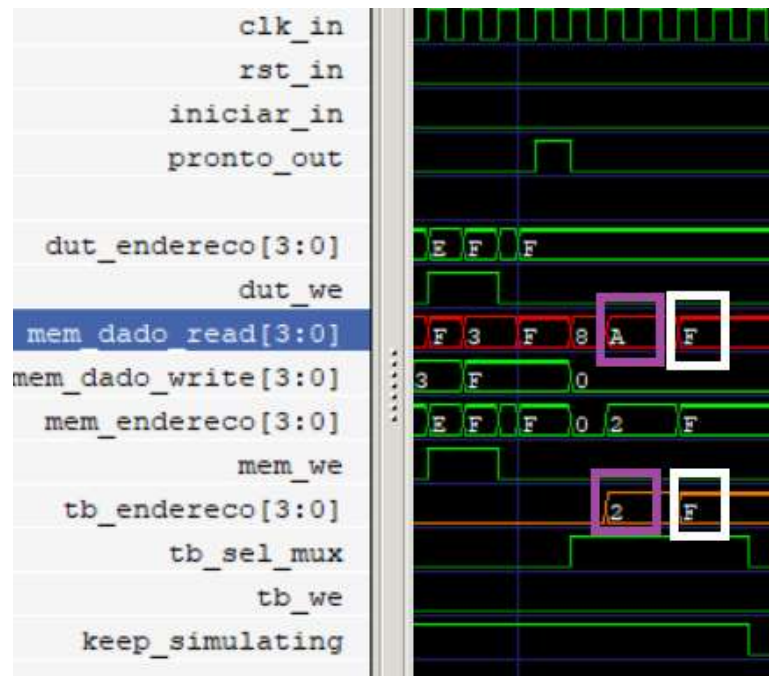
Detalhamento do projeto

- 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

The screenshot displays the EDA Playground web interface. The top navigation bar includes 'Run', 'Save', 'Copy', and a banner for 'Cadence Xcelium 19.0'. The left sidebar shows 'Languages & Libraries' with 'VHDL' selected, and 'Tools & Simulators' with 'GHDL 0.37' selected. The main area is split into two panels: 'testbench.vhd' and 'design.vhd'.

testbench.vhd (VHDL Testbench):

```

1 -----
2 --! @file mux4_2to1.vhd
3 --! @brief 2-to-1 1-bit multiplexer
4 --! @author Edson Midorikawa (emidorik@usp.br)
5 --! @date 2020-06-30
6 -----
7
8 entity mux_2to1 is
9     port
10         (
11             SEL : in bit;
12             A   : in bit;

```

design.vhd (VHDL Design):

```

1 -----
2 --! @file cont4.vhd
3 --! @brief 4-bit synchronous binary counter
4 --! @author Edson Midorikawa (emidorik@usp.br)
5 --! @date 2020-06-16
6 -----
7
8 library ieee;
9 use ieee.numeric_bit
10
11 entity cont4 is
12     port

```

The bottom panel shows the simulation log and waveform. The log includes the command 'elaborate littlesort_tb' and various simulation steps. The waveform displays signals like 'clk_in', 'rst_in', 'iniciar_in', 'pronto_out', 'dut_endereco[3:0]', 'dut_we', 'mem_dado_read[3:0]', 'mem_dado_write[3:0]', 'mem_endereco[3:0]', 'mem_we', 'tb_endereco[3:0]', 'tb_sel_mux', 'tb_we', and 'keep_simulating'.

Perguntas

Dúvidas?

Pode ligar o áudio/microfone ou perguntar via chat.

Obrigado pela participação na aula!