Sistemas Digitais I	Projeto 4	Aula 28	Responsáveis:
PCS3115	2020S1	08/07/2020	MT/ETM

Nome: **#USP:** **Turma:**

PCS3115 – Sistemas Digitais I – Projeto 4 – *Little Sort* – 2020S1

Ordenação de Dados em Uma Memória

Versão 03/07/2.020

Concepção: Marco Túlio Carvalho de Andrade

Revisão Técnica, Testbench, Simulações: Edson Toshimi Midorikawa

Para aqueles que simplesmente estão com pressa, há a possibilidade de se dirigir diretamente aos capítulos de **ENTREGAS – TAREFAS A SEREM REALIZADAS** e **INSTRUÇÕES PARA SUBMISSÃO DO PROJETO NO e-DISCIPLINAS**.

Para aqueles que acreditam na sabedoria popular:

"***Paso lento, pero sostenido***"

Sugerimos a leitura desde o começo do documento.

SUMÁRIO

I-INTRODUÇÃO

Uma das maneiras conhecidas de se ordenar um conjunto de palavras, de acordo com a magnitude que cada elemento do conjunto representa, é a técnica de Ordenação por Troca (*Sort by Exchange*). Imagine um conjunto de N Registros, R_1,..., R_N, onde todos possuem uma chave (*Key*) associada (armazenada), K_1,..., K_N. As chaves tem uma representação que as diferencia de acordo com sua magnitude. A técnica consiste em comparar K_1 e K_2, intercambiando R_1 e R_2, se as chaves estão fora de ordem (ou seja, $K_1 > K_2$); então fazer o mesmo com o par de registros R_2 e R_3, R_3 e R_4, ... , R_{N-1} e R_N. Durante esta primeira sequência de varredura com a operação de comparação de magnitude de pares de chaves relacionadas a registros vizinhos, os registros com as maiores chaves tendem a deslocar-se para a direita (supondo que a varredura está sendo feita da esquerda para a direita e que se convencionou mover as chaves de maior magnitude para os registros mais à direita). No final desta primeira varredura o registro que tiver a chave de maior magnitude será o que estará ocupando a posição R_N. Repetições do processo levarão os registros apropriados para as posições, R_{N-1}, R_{N-2}, e assim por diante.

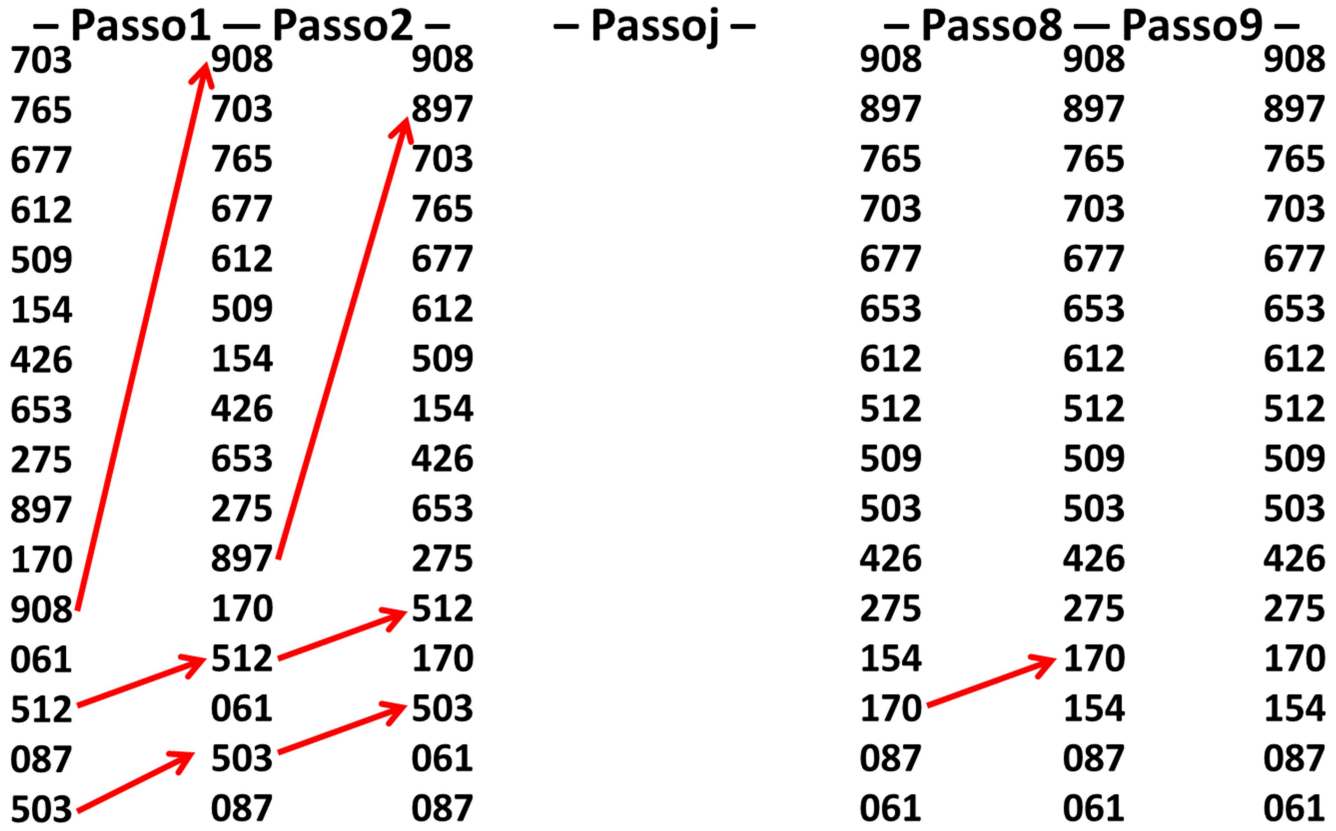


Figura I.1 – Exemplo do uso da técnica de Ordenação por Troca (*Sort by Exchange*) [*].
[*] Adaptação da Figura 14, página 106 de: Donald E. Knuth; *The Art of Computer Programming*, *Volume* 3, *Sorting and Searching*, *Second Edition*

Na Figura I.1 aparece um exemplo do uso da técnica de Ordenação por Troca, aplicada a um conjunto de 16 registros, com suas respectivas 16 chaves. Nesta Figura I.1 a representação escolhida foi a vertical, com as chaves maiores sendo deslocadas para cima (em vez de para a direita como na apresentação da técnica em parágrafo anterior). Há que se notar que, quando foi realizado o Passo 9 não houve nenhuma troca de registro, o que indica que para aquele exemplo o problema já foi solucionado, não há mais o que fazer.

Com este método de representação e deslocamento vertical, a técnica passou a ser conhecida como *Bubble Sort*, isto porque as chaves de maior magnitude tendem a um movimento de *Bubble Up*, ou seja, saltam como bolhas ascendentes, a partir de suas posições iniciais.
Um algoritmo *Bubble Sort* para resolução desta classe de problemas pode ser encontrado na referência *The Art of Computer Programming*, *Volume* 3, *Sorting and Searching*, *Second Edition*, em sua página 107. Este livro é um texto clássico do autor Donald E. Knuth. Reproduz-se neste documento o algoritmo *Bubble Sort*, tal como é apresentado no livro:

Algorithm B *Bubble Sort* – Records R_1,..., R_N, are rearranged in place; after sort is complete their Keys will be in order; $K_1 \leq \ldots \leq K_N$.

- **B1**. [Initialize **BOUND**.] Set **BOUND ← N** (**BOUND** is the highest index which the record is not known to be in its final position, that we are indicating that nothing is known at this point.);

- **B2**. [Loop on **j**.] Set **t ← 0**. Perform step **B3** for j = 1, 2, ..., **BOUND - 1** and then go to the step **B4;**

- **B3**. [Compare/exchange $\mathbf{R_j : R_{j+1}}$.] If $\mathbf{K_j > K_{j+1}}$, interchange $\mathbf{R_j \leftrightarrow R_{j+1}}$ and set **t ← j;**

- **B4**. [Any Exchanges?] If **t = 0**, terminate the algorithm. Otherwise **Set BOUND ← t** and return to Step **B2**.

Neste mesmo livro/página, em seguida do texto do algoritmo, o autor apresenta em sua Figura 15, um diagrama de fluxo (*Flow Chart*) do *Bubble Sort*, muito útil no entendimento do funcionamento do algoritmo. Uma adaptação deste diagrama aparece na Figura II.2 deste documento.

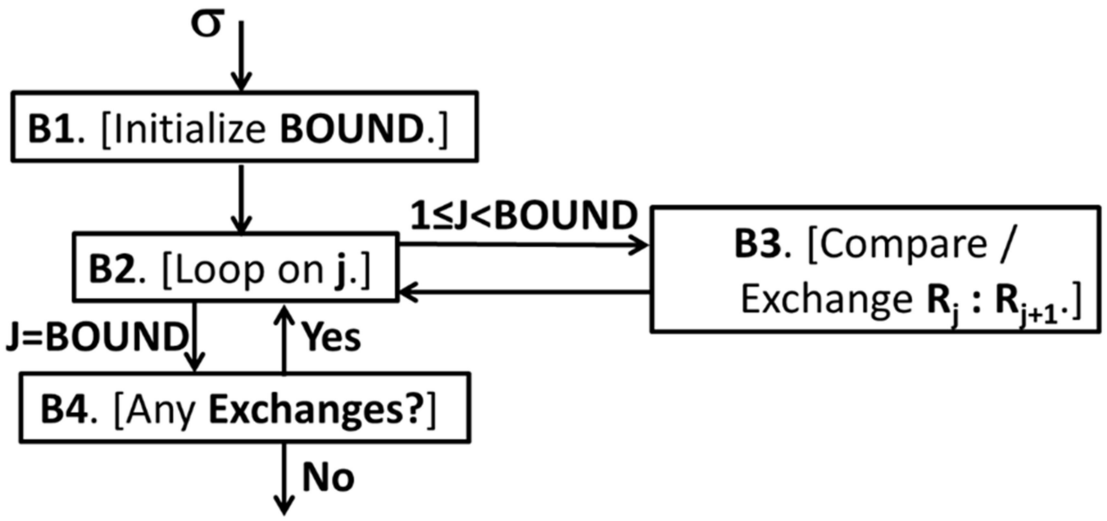


Figura II.2 – Adaptação do *Flow Chart* do *Bubble Sort* [**].
[**] Adaptação da Figura 15, página 107 de: Donald E. Knuth; *The Art of Computer Programming*, *Volume* 3, *Sorting and Searching*, *Second Edition*

A apreciação do diagrama de fluxo nos permite várias considerações.

Imagine que em sua vida profissional alguém lhe encomenda a síntese de um sistema digital (*hardware*) que corresponda à implementação física do algoritmo *Bubble Sort* (ou parte dele, que

será o nosso caso, como poderão constatar adiante). Seguramente lhes ocorrerá fazer uso de um método de projeto que lhes permita recorrer a uma série de ações sistemáticas para sua materialização. Um método fortemente recomendável para abordar e resolver a encomenda não é outro senão o da divisão do problema maior (por exemplo, o *Bubble Sort*) em dois outros problemas menores, quais sejam, sintetizar uma **Unidade de Controle** (**UC**) e um **Fluxo de Dados** (**FD**). Os dois estão baseados na definição do **QUE** deve ser feito.

- **Problema** a ser resolvido – **QUE**:
    - **Unidade de Controle (UC)** – **QUANDO**, em qual **SEQUENCIA**, qual o mecanismo para enviar **ORDENS DE EXECUÇÃO** e receber os **RESULTADOS DA EXECUÇÃO**;
    - **Fluxo de Dados (FD)** – **COMO** realizar a operação propriamente dita, qual o mecanismo para receber **ORDENS DE EXECUÇÃO** e enviar os **RESULTADOS DA EXECUÇÃO**.

O exposto anteriormente transforma cada um dos dois problemas a serem resolvidos – sintetizar uma UC e um FD – em uma série de sub-problemas específicos para cada uma delas (UC e FD).

- **Unidade de Controle (UC):**
    - **QUANDO o QUE** – **Estado** de uma FSM (UC);
    - **SEQUENCIA** – **Próximo Estado** de uma FSM (UC);
    - **ORDENS DE EXECUÇÃO** – **SINAIS DE CONTROLE** gerados por uma FSM (UC) – Saídas da UC, entradas do FD;
    - **RESULTADOS DA EXECUÇÃO** – **STATUS FLAGS, RESULTS, BOOLEAN,** gerados pelo FD, para informar a UC sobre os resultados (*Results*, *Status Flags*) das operações ordenadas. Além do mais, verificar se chegou numa resposta final do sub-problema ou ainda não (*Boolean*, *Status Flags*) – Entradas da UC, saídas do FD;

- **Fluxo de Dados (FD):**
    - **COMO o QUE** – Implica em definir quais são os circuitos combinatórios elementares (portas lógicas, etc.), blocos combinatórios elementares (mux, demux, decoder, encoder, comparador, somador, ULA, etc.), circuitos sequenciais elementares (biestáveis, latches, Flip-flops), blocos sequenciais elementares (registradores de propósito geral, contadores, LFSRs, memórias, bancos de registradores, etc.), que serão necessários para a realização das operações necessárias (o COMO sobre o QUE);
    - **ORDENS DE EXECUÇÃO** – **SINAIS DE CONTROLE** gerados por uma FSM (UC) – , Entradas do FD, saídas da UC;
    - **RESULTADOS DA EXECUÇÃO** – **STATUS FLAGS, RESULTS, BOOLEAN** gerados pelo FD, para informar a UC sobre os resultados (*Results*, *Status Flags*) das operações ordenadas, e também se chegou numa resposta final do sub-problema ou ainda não (*Boolean*, *Status Flags*) – Saídas do FD, entradas da UC,;

A análise do *Flow Chart* da Figura II.2 nos permite extrair considerações da ordem do **QUANDO** o **QUE** deve ocorrer, e da sua **SEQUENCIA** de ocorrência.

A análise do algoritmo *Bubble Sort* nos permite extrair considerações da ordem do **COMO** o **QUE** pode ser sintetizado. Continuando esta linha de raciocínio, existem outras ferramentas de representação (Diagramas ASM, por exemplo) que nos proporcionam outras vistas dos vários matizes dos processos de síntese.

Na parte V deste texto encontra-se um Apêndice com uma questão de prova (Questão 3, Prova 3, 2.018S2) que contempla um algoritmo mais próximo do problema que se quer resolver. Para facilitar o entendimento este algoritmo será repetido aqui.

Algorithm P3-3-2.018S2 – Declarações:

Register (A[3:0] % Vector Register Temporary Data A %)
Memory (DW[3:0]; DR[3:0] % Vector DataWrite, DataRead %
Write; Read % Signal bit %
AW[3:0]; AR[3:0] % Vector AddressWrite, AddressRead %)
Counter (RAC[3:0]; WAC[3:0] % ReadAddressCounter; WriteAddressCounter %)
Boolean (Order % Ordering: TRUE OR FALSE %

Condições Iniciais: Order := TRUE; Memória pré-carregada.

Algorithm:

```
While (Order is TRUE) {
    Order := FALSE
    RAC[3:0] ← 0                          % ReadAddressCounter Clear %
    WAC[3:0] ← 0                          % WriteAddressCounter Clear %
    While (RAC[3:0] /= '1111') {          % ReadAddressCounter reachs 1111 (F) %
        A[3:0] ← DR[3:0]                  % Load position "i" value in Temp. Regist. A %
        RAC[3:0] ← RAC[3:0] + 1           % ReadAddressCounter CountUp, to read B %
        IF A > B (                        % B is the value of position "i+1" %
            DW[3:0] ← DR[3:0]             % DW[3:0] ← B ; Write B %
            WAC[3:0] ← WAC[3:0] + 1       % WriteAddressCounter CountUp, to write A %
            DW[3:0] ← A[3:0]              % DW[3:0] ← A ; Write A %
            Order := TRUE
        Else
            WAC[3:0] ← WAC[3:0] + 1       % WriteAddressCounter CountUp %
        )
    }
}
```

Neste algoritmo, diferentemente do *Bubble Sort*, as sucessivas varreduras para comparação dos registros são feitas sobre todas as posições de memória. Em outras palavras, este algoritmo tinha outras funções naquela prova e não se aproveitava do recurso de economia de tempo de execução que o *Bubble Sort* dispõe. Ele não se aproveita do fato de que após a primeira varredura o maior dos elementos em magnitude já deverá estar registrado na última posição de memória. Portanto a última posição de memória não precisará ser lida e comparada na próxima varredura, Do mesmo modo, a cada varredura diminui a necessidade de varredura em mais uma posição.

O algoritmo que vai ser utilizado neste enunciado, na solução do "**problema a ser resolvido**", é semelhante a este último com a ressalva de que será exigido apenas o primeiro passo de varredura completa e ordenação dos dados. Em outras palavras, será feita apenas uma varredura completa e a solução que o aluno submeter ao e-disciplinas deve garantir que, nesta única varredura o maior, em magnitude, dos dados lidos, estará armazenado na última posição.

II-PROBLEMA A SER RESOLVIDO

Considere que se vai realizar a síntese de um **Sistema Digital** (**SD**) composto por uma **Unidade de Control**e (**UC**) e um **Fluxo de Dados** (**FD**). Considere dada uma **memória RAM** externa a este Sistema Digital. Esta memória possui 4 bits de endereço (AD_3, AD_2, AD_1, AD_0) e tamanho de palavra de 4 bits (D_3, D_2, D_1, D_0). A memória RAM tem conteúdo não nulo pré-gravado.

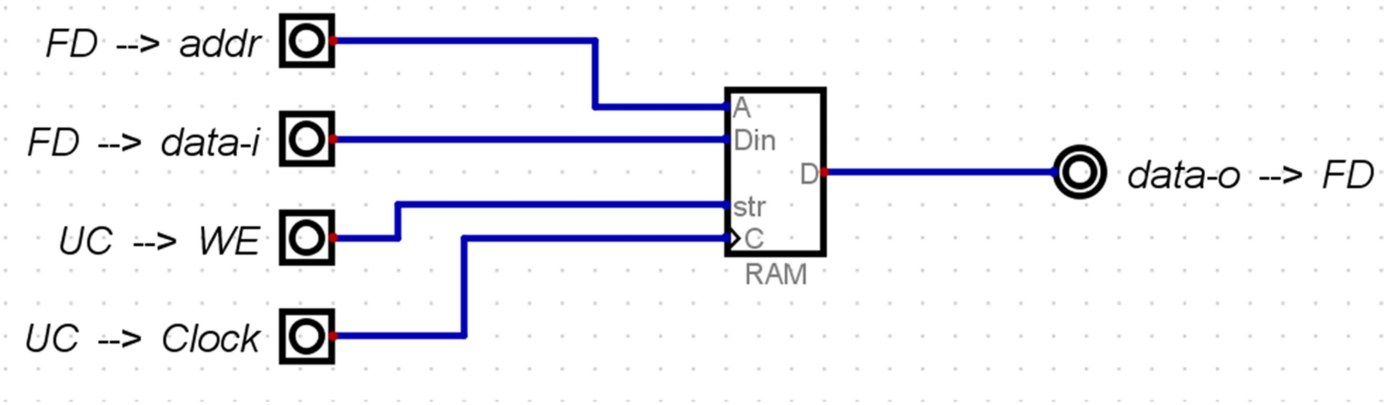


Figura II.1 – Diagrama de Blocos da Memória RAM externa.

A memória RAM está conectada (integrada) ao Sistema Digital por meio de três vias (*Buses*): Via de Dados (*Data Bus*), Via de Endereços (*Address Bus*) e Via de Controle (*Control Bus*). A Via de Controle conecta a Unidade de Controle do Sistema Digital com a memória. As Vias de Endereços e de Dados conectam a memória RAM com o Fluxo de Dados do Sistema Digital.

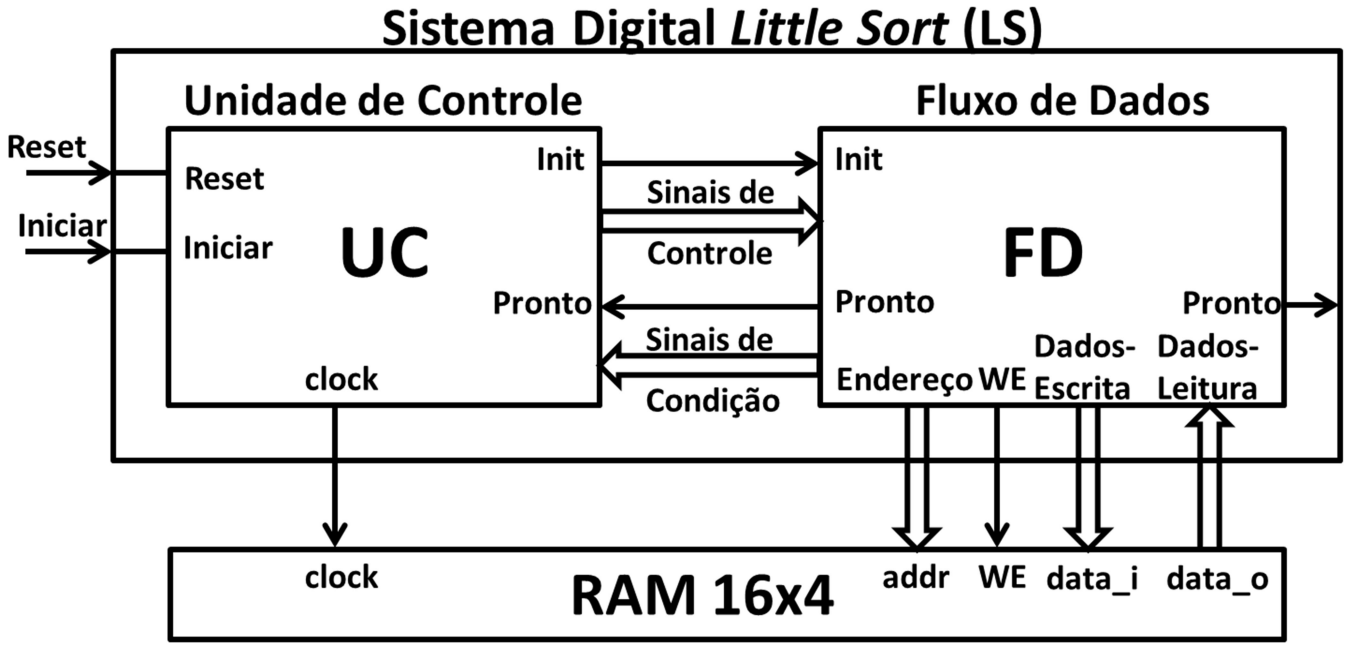


Figura II.2 – Diagrama de Blocos do Sistema Digital.

Deseja-se que o Sistema Digital execute uma (e apenas uma) varredura completa nesta memória, realizando a ordenação do conjunto de palavras ali gravadas. Esta ordenação será de acordo com a magnitude que cada elemento do conjunto de palavras armazenadas representa. A técnica a ser aplicada deve ser obrigatoriamente a de Ordenação por Troca (*Sort by Exchange*). O resultado da nova organização de dados da memória RAM será gravado nesta mesma RAM que, reiterando, é

externa ao Sistema Digital. É requisito deste projeto o uso do algoritmo, escrito em linguagem C, que é apresentado a seguir.

Algorithm LS *Little Sort* :

```
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;
                }
}
```

A síntese que se pede deve ser conduzida de tal maneira que, um conjunto de sinais de controle gerados por um programa de teste (*testbench*) permita certificar-se, por verificação externa na memória RAM, que o conteúdo foi efetivamente reorganizado, de maneira correta e se encontra gravado nesta.

O problema a ser resolvido (e a ser submetido ao e-disciplinas) pelo aluno é a síntese do Fluxo de Dados do Sistema Digital "*Little Sort*"!

III-PROPOSTA DE OBTENÇÃO DA SOLUÇÃO

Propõe-se que o Fluxo de Dados seja dividido em dois Blocos de maneira que estes permitam testes separadamente um do outro.

O que importa, neste momento, é que os blocos, individualmente, sejam auto suficientes em termos de facilidade de teste, cada um sem depender do outro.

O projeto deve ser feito usando uma **descrição estrutural**, integrando diversos componentes em um **projeto hierárquico**.

Bloco 1

- **Função**:
  - Responsável pela geração e atualização dos endereços de leitura e de escrita na RAM externa;
  - Responsável pela escolha de um endereço, dentre dois endereços possíveis, para efetuar operação de Leitura na RAM. Um endereço para ler o registro R_i e outro (uma posição adiante de R_i) para ler o registro R_{i+1};
  - Responsável pela escolha de uma palavra de dados, dentre duas palavras de dados possíveis, para efetuar a operação de Escrita na RAM. Uma palavra de dados se o registro que se quer escrever na memória RAM for R_i e outro, no caso contrário, se for para escrever a palavra de dados R_{i+1};
- **Elementos constituintes**:
  - Circuitos combinatórios elementares (portas lógicas, etc.), blocos combinatórios elementares (mux, demux, decoder, encoder, comparador, somador, ULA, etc.), circuitos sequenciais elementares (biestáveis, latches, Flip-flops), blocos sequenciais elementares (registradores de propósito geral, contadores, LFSRs, memórias, bancos de registradores, etc.), que serão usados para a realização das operações necessárias (dependerá de **sua solução** para executar a **função** a que se **destina**).

Para este Bloco execute os seguintes passos:

1-) Desenhe uma caixa retangular que represente o bloco com todas suas entradas e saídas. Pede-se que as entradas que sejam provenientes da Unidade de Controle sejam desenhadas com nomes e flechas na parte superior da caixa, e saídas do Bloco (sinais de condição) para a Unidade de Controle e para a memória RAM externa o sejam na parte inferior. Do mesmo modo, entradas que sejam provenientes de outros Blocos do Fluxo de Dados sejam desenhadas com nomes e flechas na parte esquerda da caixa, e saídas do Bloco para outros Blocos do Fluxo de Dados o sejam na parte direita da caixa. Este diagrama será de utilidade quando a entidade (*Entity*) do circuito tiver que ser definida.

2-) Faça o projeto da lógica combinatória e/ou sequencial necessária para que o bloco execute as operações que são de sua responsabilidade. Descreva o resultado do projeto em VHDL e por meio de um diagrama lógico do circuito.

3-) Faça um *testbench* para o bloco.

Bloco 2

- **Função**:
  - Responsável pelo registro temporário dos pares de palavras de dados que serão comparados em magnitude;
  - Responsável pela comparação em magnitude das palavras de dados correspondentes a R_i e R_{i+1} .
- **Elementos constituintes**:
  - Circuitos combinatórios e/ou sequenciais, blocos combinatórios e/ou sequenciais, que serão usados para a realização das operações necessárias (dependerá de **sua solução** para executar a **função** a que se **destina**).

Para este Bloco execute os seguintes passos:

1-) Desenhe uma caixa retangular que represente o bloco com todas suas entradas e saídas. Pede-se que as entradas que sejam provenientes da Unidade de Controle sejam desenhadas com nomes e flechas na parte superior da caixa, e saídas do Bloco para a Unidade de Controle e para a memória RAM externa o sejam na parte inferior. Do mesmo modo, entradas que sejam provenientes de outros Blocos do Fluxo de Dados sejam desenhadas com nomes e flechas na parte esquerda da caixa, e saídas do Bloco para outros Blocos do Fluxo de Dados o sejam na parte direita da caixa. Este diagrama será de utilidade quando a entidade (*Entity*) do circuito tiver que ser definida.

2-) Faça o projeto da lógica combinatória e/ou sequencial necessária para que o bloco execute as operações que são de sua responsabilidade. Descreva o resultado do projeto em VHDL e por meio de um diagrama lógico do circuito.

3-) Faça um *testbench* para o bloco.

Bloco 3 – Não obrigatório – Desafio

- **Função**:
  - Seria desejável que este Fluxo de Dados pudesse ser utilizado para implementar o algoritmo *Bubble Sort* de maneira completa; Para tal seria necessário fazer constar um mecanismo de controle dos *loops* para atualizar os limites de varredura; Lembrar que naquele algoritmo, o último endereço da varredura anterior não precisa mais ser varrido; Também seria importante contar com uma variável *Booleana* para *exit* da varredura se algum passo de varredura não apresentou nenhuma reordenação de dados.
- **Elementos constituintes**:
  - Circuitos combinatórios e/ou sequenciais, blocos combinatórios e/ou sequenciais, que serão usados para a realização das operações necessárias.

1-) Desenhe uma caixa retangular que represente o bloco com todas suas entradas e saídas. Pede-se que as entradas que sejam provenientes da Unidade de Controle sejam desenhadas com nomes e flechas na parte superior da caixa, e saídas do Bloco para a Unidade de Controle e para a memória RAM externa o sejam na parte inferior. Do mesmo modo, entradas que sejam provenientes de outros Blocos do Fluxo de Dados sejam desenhadas com nomes e flechas na parte esquerda da caixa, e saídas do Bloco para outros Blocos do Fluxo de Dados o sejam na parte direita da caixa. Este diagrama será de utilidade quando a entidade (*Entity*) do circuito tiver que ser definida.

2-) Faça o projeto da lógica combinatória e/ou sequencial necessária para que o bloco execute as operações que são de sua responsabilidade. Descreva o resultado do projeto em VHDL e por meio de um diagrama lógico do circuito.

3-) Faça um *testbench* para o bloco.

IV- ENTREGAS – TAREFAS A SEREM REALIZADAS PARA ENTREGAR

O que se deve entregar é o projeto de um **Fluxo de Dados** composto pela integração dos **Blocos 1 e 2**, cuja síntese foi a proposta na Parte III deste enunciado.

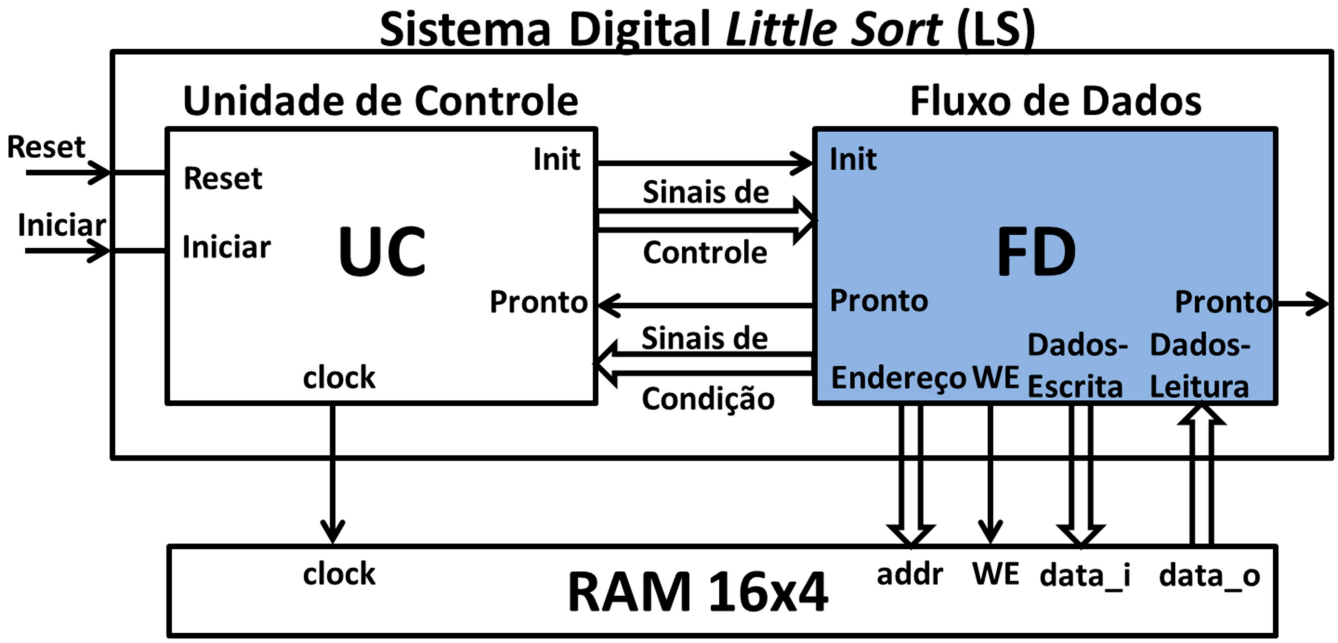


Figura IV.1 – Diagrama de Blocos do Sistema Digital *Little Sort* (LS) – Com Destaque Para o Trabalho que o Aluno Deve Realizar (Entregar).

Para que fique mais claro em que consiste o trabalho que deve ser realizado pelo Aluno, vai-se elencar o que será fornecido de material de apoio para que se possa realizar o trabalho específico:

- Arquivo littlesort_fd_fornecido.vhd – Este arquivo contém:
  - cont4.vhd;
  - fa_1bit.vhd;
  - fa_4bit.vhd;
  - mux4_2x1;
  - reg4.vhd;
  - comp4,vhd
  - entity littlesort_fd.vhd – Trata-se da especificação da ***entity*** que deve ser usada pelo aluno para submeter seu projeto4, do **Fluxo de Dados**, ao e-disciplinas.
- Arquivo littlesort_testbench_uc_fd_ram_fornecido.vhd – Este arquivo contém:
  - tb_mux4_2to1.vhd;
  - ram16x4.vhd;
  - tb_littlesort_uc.vhd;
  - tb_littlesort.vhd – Trata-se de uma proposta de como fazer um ***testbench*** que, de maneira opcional (não é obrigatório) pode ser usado pelo aluno para testar sua proposta antes de submeter seu projeto4 ao e-disciplinas.
- Arquivo memoria1_fornecida.dat – Este arquivo contém os dados de uma memória RAM 16x4 totalmente preenchida com conteúdo não nulo (isto servirá para o aluno testar a síntese de seu **Fluxo de Dados**).

O arquivo “littlesort_fd_fornecido.vhd” será disponibilizado no e-disciplinas juntamente com a publicação do enunciado do problema. Com relação ao conteúdo da parte “tb_littlesort.vhd”, do arquivo “littlesort_testbench_uc_fd_ram_fornecido.vhd”, esse conteúdo em particular só poderá ser definido após a Aula28 on-line que será sobre este Projeto4. Este segundo arquivo a ser fornecido só será disponibilizado após a Aula28 on-line. Proceder-se-á desta forma para que se se possa gerar uma versão que atenda as dúvidas dos alunos, que porventura surjam na aula.

A **entidade da síntese** (implementação) do **Fluxo de Dados** do ***Little Sort*** que o aluno submeterá precisa estar, **obrigatoriamente**, de acordo com a que está definida no conteúdo “entity littlesort_fd.vhd” (Figura IV.2). Esta deve ser também idêntica na solução submetida. Se não ocorrer estritamente desta forma o **Juiz** do e-disciplinas **não irá processar o arquivo submetido de maneira correta**.

Figura IV.2 – Definição de ***entity*** (uso **obrigatório**) do Fluxo de Dados que se Deve Entregar.

O uso de uma memória RAM externa incorporada ao Sistema Digital impõe uma série de dificuldades adicionais no momento de elaborar um *testbench* para verificação do funcionamento correto do circuito. Para explicar estas dificuldades far-se-á uso de diagramas de blocos simplificados, para o Sistema Digital (Figura IV.3) a ser projetado e seu respectivo *testbench* (Figura IV.4).

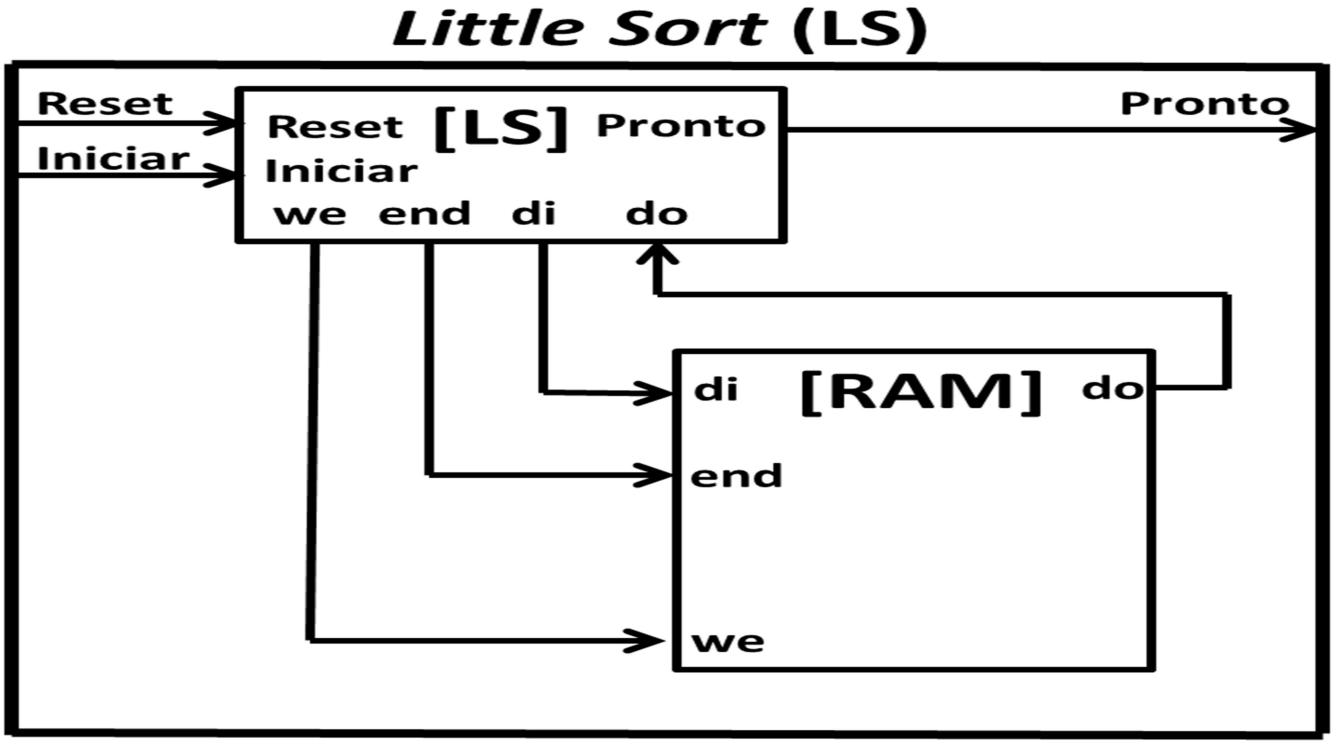


Figura IV.3 – Diagrama de Blocos Simplificado do *Little Sort*.

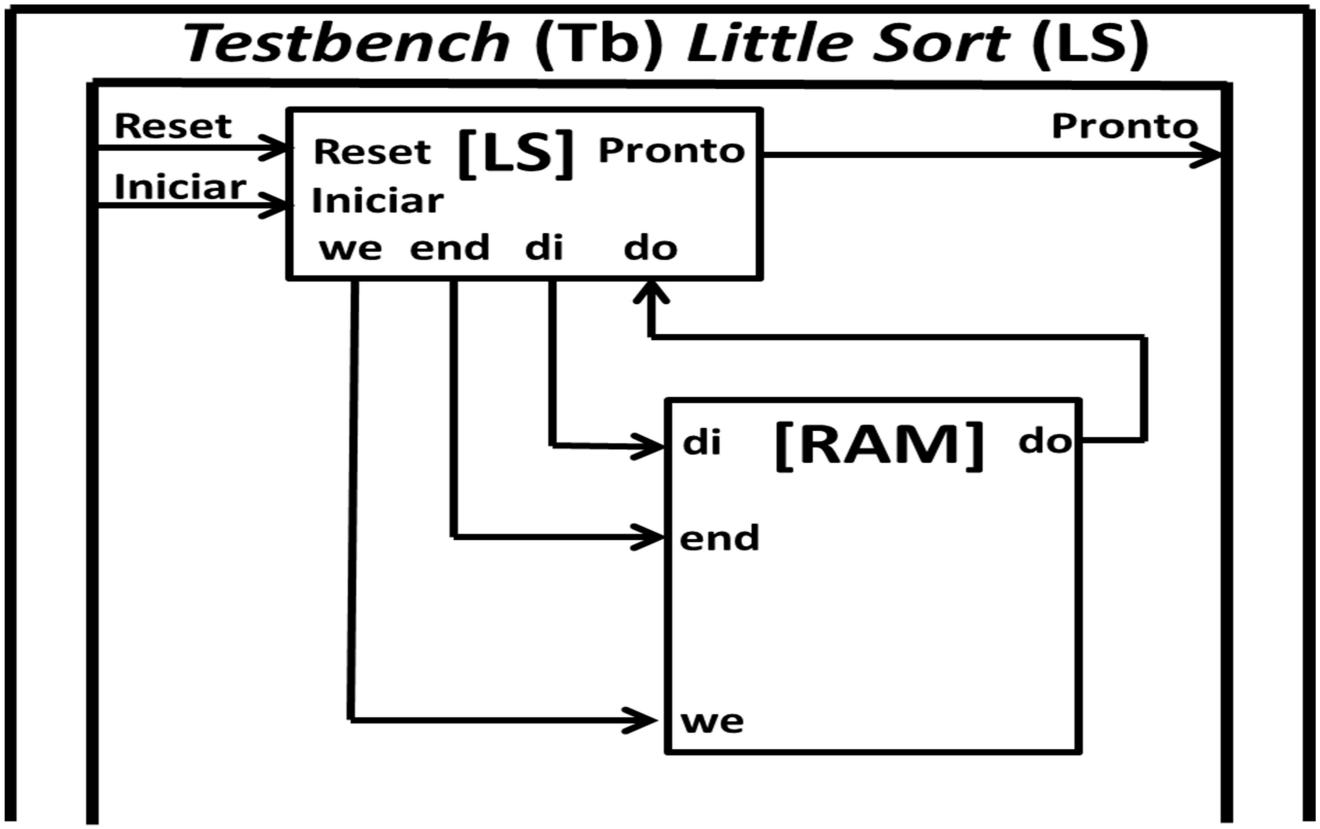


Figura IV.4 – Diagrama de Blocos Simplificado do *Testbench* do *Little Sort*.

Esse *testbench* da Figura IV.4 segue a maneira que estamos acostumados a concebê-lo. Porém uma análise cuidadosa nos permite ver que não temos recursos suficientes para que o *testbench* possa, após provocar uma varredura e reordenamento na memória, verificar se através da leitura da memória se o reordenamento foi feito de maneira correta. Para permitir tal verificação propõe-se que o testbench tenha a estrutura da Figura IV.5.

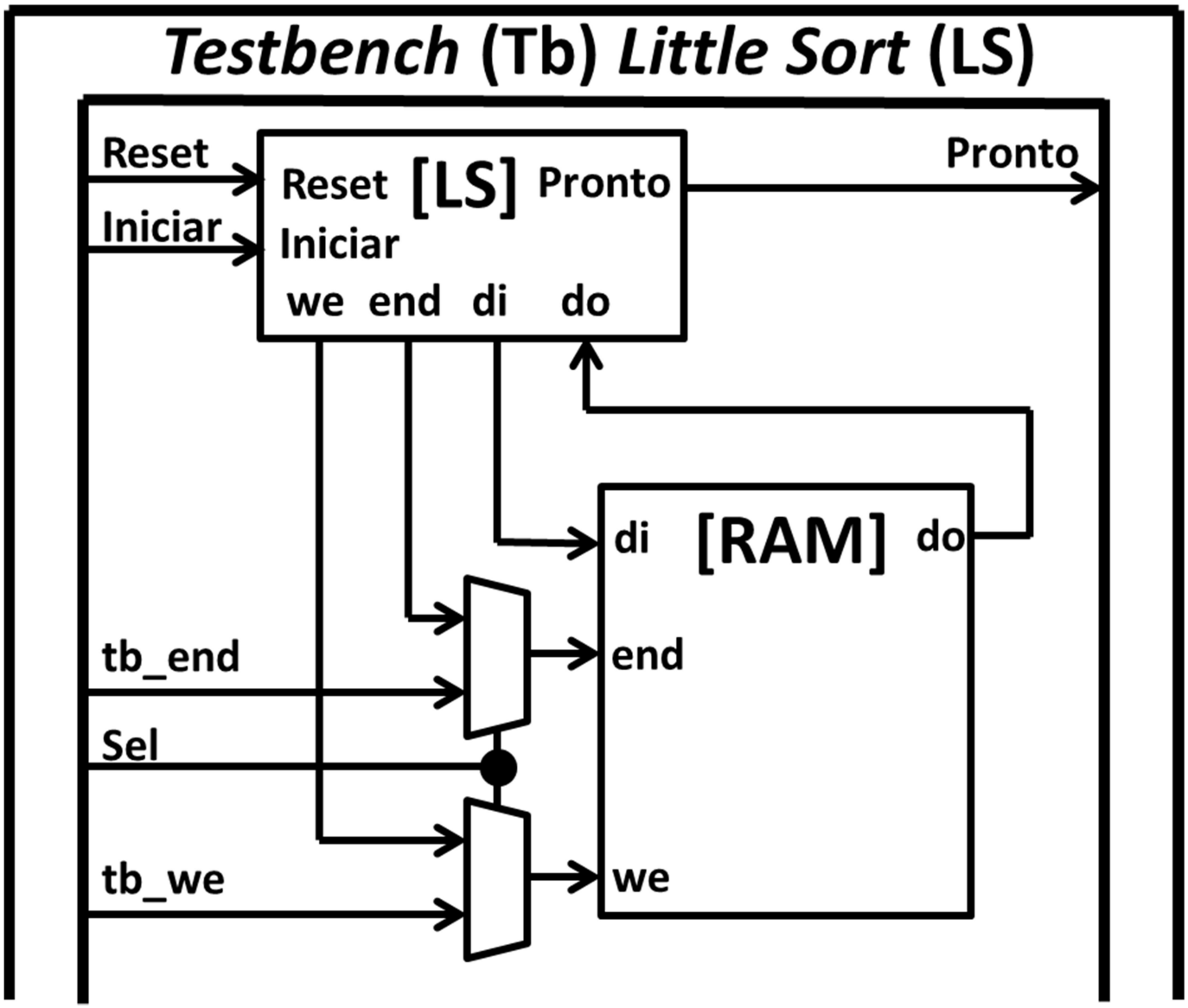


Figura IV.5 – Diagrama de Blocos Simplificado de Estrutura do *Testbench* sugerido para o projeto *Little Sort*.

Para a entrega do **Fluxo de Dados** sugere-se a execução dos seguintes passos:

1-) Desenhe duas caixas retangulares maiores, uma à direita, que represente o Fluxo de Dados, e outra a sua esquerda, que represente a Unidade Controle. Sinais trocados entre a UC e o FD devem ser desenhados no interstício entre as duas. Uma outra caixa menor representará a memória RAM externa. Sinais trocados entre o FD e a RAM serão desenhados na parte de baixo da caixa do FD.

2-) Faça o projeto da lógica combinatória e/ou sequencial necessária para que os blocos 1 e 2 possam atuar de maneira integrada e correta. Integre esta lógica a algum dos blocos refazendo o projeto do bloco e repetindo os passos necessários para síntese daquele bloco. Esta etapa de ter que acrescentar

lógica para integrar os blocos deve ser evitada. Um bom projeto dos blocos 1 e 2 não necessitaria mais do que a simples interligação dos sinais comuns entre eles.

3-) Faça um *testbench* para o Sistema assim constituído (Fluxo de Dados mais memória RAM).

4-) Submeta o resultado ao e-Disciplinas de acordo com as instruções do Capítulo VI:

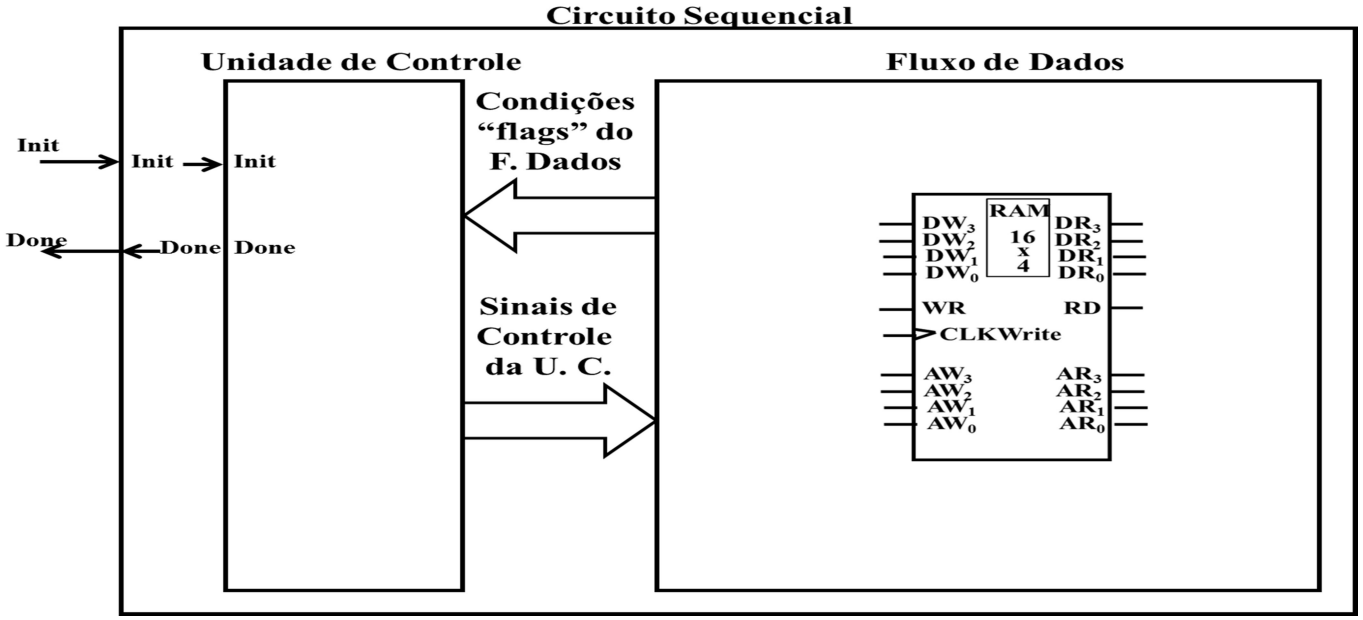
VI- INSTRUÇÕES PARA SUBMISSÃO DO PROJETO NO e-DISCIPLINAS

5-) Bom trabalho!

V- APÊNDICE – Questão 3 – Prova 3 – Sistemas Digitais II – 2.018S2

Questão 3. (2,5 pontos) –

Considere dada uma memória de escrita/leitura de 16 posições e tamanho de palavra igual a 4 bits. Ela é acionada por sinal de clock para escrita, quando o sinal Write (WR) estiver ativo, e sensível ao nível para leitura, quando o sinal Read (RD) estiver ativo. Possui duas entradas de dados distintas (Dual Port), uma para a escrita (DW_3, DW_2, DW_1, DW_0) e outra para leitura (DR_3, DR_2, DR_1, DR_0). Além de duas entradas distintas de bits de endereçamento para gerar o endereço de escrita (AW_3, AW_2, AW_1, AW_0) e outra para o endereço de leitura (AR_3, AR_2, AR_1, AR_0). A memória é pré-carregada com 16 palavras distintas de dados. Não existe posição da memória que tenha o mesmo valor binário que outra posição e todos os valores binários (hexadecimais) de "0000" até "1111" (de "0" até "F") estão contemplados. Estes dados não estão registrados em ordem de magnitude crescente (do menor para o maior). O problema que se quer resolver é ordenar os dados da memória da magnitude menor para a maior. Uma idéia é percorrer os endereços da memória e se for verificado que o dado na posição "i" é maior que o dado na posição "i+1", fazer a troca. A troca se daria por meio de um registrador temporário para armazenar o dado da posição "i", seguido da escrita do dado da posição "i+1" no endereço "i". Finalmente sucedida pela escrita do dado do registrador temporário no endereço "i+1". A execução da solução é iniciada por meio de uma entrada externa Init, e sua consecução é indicada por meio de uma saída externa Done. Segue diagrama de blocos do circuito sequencial que se quer implementar.



Segue pseudo-código do algoritmo que se quer utilizar para a obtenção do circuito sequencial desejado. **Declarações:**

Register (A[3:0] % Vector Register Temporary Data A %)
Memory (DW[3:0]; DR[3:0] % Vector DataWrite, DataRead %
Write; Read % Signal bit %
AW[3:0]; AR[3:0] % Vector AddressWrite, AddressRead %)
Counter (RAC[3:0]; WAC[3:0] % ReadAddressCounter; WriteAddressCounter %)
Boolean (Order % Ordering: TRUE OR FALSE %

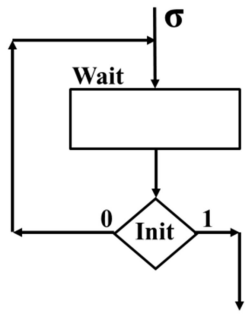
Condições Iniciais: Order := TRUE; Memória pré-carregada

Algorítmo:

```
While (Order is TRUE) {
    Order := FALSE
    RAC[3:0] ← 0                          % ReadAddressCounter Clear %
    WAC[3:0] ← 0                          % WriteAddressCounter Clear %
    While (RAC[3:0] /= '1111') {          % ReadAddressCounter reachs 1111 (F) %
        A[3:0] ← DR[3:0]                  % Load position "i" value in Temp. Regist. A %
        RAC[3:0] ← RAC[3:0] + 1           % ReadAddressCounter CountUp, to read B %
        IF A > B (                        % B is the value of position "i+1" %
            DW[3:0] ← DR[3:0]             % DW[3:0] ← B ; Write B %
            WAC[3:0] ← WAC[3:0] + 1       % WriteAddressCounter CountUp, to write A %
            DW[3:0] ← A[3:0]              % DW[3:0] ← A ; Write A %
            Order := TRUE
        Else
            WAC[3:0] ← WAC[3:0] + 1       % WriteAddressCounter CountUp %
        )
    }
}
```

3.a. (1,5 pontos) – Desenvolver o Fluxo de Dados para o circuito sequencial desejado.

3.b. (1,0 ponto) – Desenvolver um Diagrama ASM para o circuito sequencial desejado. Este Diagrama ASM pode ser de alto nível (onde apareçam operações semelhantes às do pseudo-código do algorítmo dado), ou já preparado para ser usado na implementação da Unidade de Controle (onde aparecem os sinais de controle necessários para a realização das operações no Fluxo de Dados).



VI- INSTRUÇÕES PARA SUBMISSÃO DO PROJETO NO e-DISCIPLINAS

Há um link específico no e-Disciplinas para submissão deste projeto. Acesse-o somente quando estiver confortável para enviar sua solução. Em cada atividade, você pode enviar apenas um único arquivo codificado em UTF-8 (*). O nome do arquivo não importa, mas sim a descrição VHDL que está dentro. A entidade da síntese (implementação) do **Fluxo de Dados** do ***Little Sort*** que o aluno submeterá precisa estar, obrigatoriamente, de acordo com a que foi definida na Parte IV do enunciado e deve ser idêntica na sua solução ou o juiz não irá processar seu arquivo.

Quando acessar o link no e-Disciplinas, o navegador abrirá uma janela para envio do arquivo. Selecione-o e envie para o juiz. Jamais recarregue a página de submissão pois seu navegador pode enviar o arquivo novamente, o que vai ser considerado pelo juiz como um novo envio e pode prejudicar sua nota final. Caso desista do envio, simplesmente feche a janela.

Depois do envio, a página carregará automaticamente o resultado do juiz (**), quando você poderá fechar a janela. A nota dada pelo juiz é somente para a submissão que acabou de fazer. Esta atividade permite até 10 submissões e a nota será a maior nota dentre todas as suas submissões. Sua nota na atividade poderá ser vista no e-Disciplinas e pode diferir da nota dada pelo juiz.

Sugere-se **fortemente não usar** a biblioteca std_logic_1164, a textio e qualquer outra biblioteca não padronizada para minimizar possível fonte de problemas. Também certifique-se que não imprime nada na saída da simulação.

Atenção: não atualize a página de envio e não envie a partir de conexões instáveis (e.g. móveis) para evitar que seu arquivo chegue corrompido no juiz.

(*) Qualquer editor de código moderno suporta UTF-8 (e.g. Atom, Sublime, Notepad++, etc).

(**) Pode demorar alguns segundos até o juiz processar seu arquivo.