EXERCÍCIOS DE ORGANIZAÇÃO DE COMPUTADORES

Autor: Fernando Gehm Moraes
Data: 01/03/2001

Esta é uma primeira versão de uma lista de exercícios para a disciplina de Organização de Computadores, contendo todos os exercícios que estavam distribuídos em uma série de listas separadas.

Solicito que ao encontrarem erros, ou tiverem sugestões de novos exercícios, enviar as contribuições para **moraes@inf.pucrs.br**.

1. Usando o Diagrama Y como modelo para representar o processo de projeto de sistemas digitais, classifica-se descrições de sistemas digitais segundo dois critérios: o nível de abstração e o domínio de descrição. Baseado neste modelo, classifique as descrições da tabela mais à direita usando números e letras de acordo com o nível e o domínio mais adequado para cada uma. Complete as duas colunas em branco usando os números e letras das duas primeiras tabelas.

Nível de Abstração
1. Elétrico
2. Lógico
3. Arquitetural
4. Sistêmico

Domínio de Descrição
A. Comportamental
B. Estrutural
C. Físico/Geométrico

Descrições		
		Diagrama de portas (esquemático)
		Tabela verdade
		Layout de um circuito integrado
		Desenho de placa-mãe de um PC
		Grafo de transição de estados
		Tabela de transição de estados
		Algoritmo descrevendo o comportamento de um PC
		Diagrama de tempos
		Diagrama de transistores

EXERCÍCIOS DE PROGRAMAÇÃO ASSEMBLY

Escreva os programas abaixo utilizando a linguagem de montagem (*assembly*) do processador hipotético Cleópatra.

2. Somar uma constante a um vetor armazenado à partir do endereço de memória *end1*. O número de elementos do vetor está armazenado na posição de memória *end2*.
3. Escreva um programa para mover um vetor armazenado entre as posições de memória *início1* e *fim1* para as posições de memória *início2* e *fim2*.
4. Contar o número de posições de memória com conteúdo igual a *AAH* no vetor armazenado entre os endereços *80H* e *F5H*.
5. Dados dois vetores, iniciando nos endereços de memória *E1* e *E2* respectivamente, gere um terceiro vetor, iniciando no endereço *E3*, de tal forma que $E3_i = E1_i$ xor $E2_i$, $0 \leq i < n$, onde *n* indica número de elementos dos vetores.
6. Dado dois inteiros, *A* e *B*, armazenados nos endereços de memória *n1* e *n2*, armazenar no endereço de memória *max* o valor máximo entre A e B - *max(A,B)*, e no endereço *min* o *min(A,B)*.
7. Multiplicar por somas sucessivas 2 inteiros positivos (8 bits), armazenando o resultado em 16 bits (multiplique por exemplo FFH * FFH, o resultado é FE01H). O multiplicando e o multiplicador devem estar armazenados nos endereços *n1* e *n2*, respectivamente. O resultado deverá ser armazenado nos endereços *mh* (resultado da parte alta, no exemplo FEH) e *ml* (resultado da parte baixa, no exemplo 01H).
8. Modifique o programa anterior de multiplicação para que seja realizado o número *mínimo* de multiplicações, ou seja: Se n1>n2 realizar n1*n2, caso contrário realizar n2*n1. Por exemplo, se fizermos 4 * 94 teremos o valor 4 somado sucessivamente 94 vezes. Se for feito o contrário, 94 * 4, teremos apenas 4 somas sucessivas de 94, o que é muito mais rápido em termos de tempo de CPU consumido.
9. Faça a soma de dois números de 16 bits, retornando no flag de overflow (V) o transbordo ou não da operação.

```
      c
     n1  n11
+    n2  n21
    --------
   c r1  r11
```

Atenção: cuidar com o carry da parte menos significativa. Para o cálculo de transbordo podem haver 2 situações: soma do carry com n1 e do resultado do (carry+n1) com n2.

10. Faça um programa que gere o *n* primeiros números da seqüência de Fibonnaci, à partir do endereço *idx*.
11. Considere a série de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ... Suponha que a posição de memória END contém um número qualquer desta série, e que a posição de memória END+1 possui o número imediatamente anterior. Escreva um programa em linguagem assembly (de montagem) para a arquitetura Cleópatra que preencha a memória a partir do endereço END+2 com os elementos da série anteriores a estes, até chegar ao primeiro número, ou seja, 0. O segundo elemento armazenado (END+1) nunca será zero.

- Você pode utilizar três variáveis auxiliares como índices contendo inicialmente END, END+1 e END+2, utilizando estes índices para endereçamento indireto.

Exemplo numérico (não significa que o programa deva iniciar com estes valores):

Condição inicial da memória de dados:

		END	END+1									
		34	21									

Condição da memória após a execução do programa:

		END	END+1	END+2	END+3	END+4	END+5	END+6	END+7	END+8	END+9	
		34	21	13	8	5	3	2	1	1	0	

12. Escreva um programa para criar um vetor *vet*, à partir do vetor *a*, ambos com dimensão *n*, segundo a relação estabelecida na equação abaixo. A interpretação da equação é: cada elemento de *vet*, na posição *k*, receberá o somatório do vetor *a*, entre 0 e k.

$$vet_k = \sum_{i=0}^{k} a_i , \quad 0 \le k < n$$

Exemplo, Condição inicial da memória de dados:

n	a(0)	a(1)	a(2)	a(3)	vet(0)	vet (1)	vet (2)	vet (3)				
4	5	8	12	6								

Condição da memória após a execução do programa:

n	a(0)	a(1)	a(2)	a(3)	vet(0)	vet (1)	vet (2)	vet (3)				
4	5	8	12	6	**5**	5+8= **13**	13+12= **25**	25+6= **31**				

13. Escreva um programa para criar um vetor *novo*, à partir de dois vetores *v1* e *v2* de mesma dimensão *n*, segundo a relação estabelecida na equação abaixo. A interpretação da equação é: cada elemento de *novo*, na posição *k*, receberá o somatório dos máximos dos elementos dos vetores *v1* e *v2*, entre *0* e *k*.

$$novo_k = \sum_{i=0}^{k} max(v1_i, v2_i), \quad 0 \le k < n$$

Exemplo, condição inicial da memória de dados:

n	V1(0)	V1(1)	V1(2)	V1(3)	V2(0)	V2(1)	V2(2)	V2(3)	novo(0)	novo(1)	novo(2)	novo(3)
4	5	8	12	6	7	2	10	8				

Condição da memória após a execução do programa:

n	V1(0)	V1(1)	V1(2)	V1(3)	V2(0)	V2(1)	V2(2)	V2(3)	novo(0)	novo(1)	novo(2)	novo(3)
4	5	**8**	**12**	6	**7**	2	10	**8**	**7**	7+8 = **15**	15+12 = **27**	27+8= **35**

14. Escreva um programa para criar um vetor *novo*, à partir de dois vetores *v1* e *v2* de mesma dimensão *n*, segundo a relação estabelecida na equação abaixo. A interpretação da equação é: cada elemento de novo, na posição k, receberá o somatório dos mínimos dos elementos dos vetores v1 e v2, entre 0 e k.

$$novo_k = \sum_{i=0}^{k} min(v1_i, v2_i), \quad 0 \le k < n$$

15. Implemente um algoritmo simples de ordenação crescente. O vetor de origem deve estar armazenado entre as posições de memória *ini1* e *fim1*, e o vetor ordenado entre as posições de memória *ini2* e *fim2*.

Sugestão de algoritmo *buble sort*:

```
do{  troca=0;
     i=0;
     do{ if ( vet[i] > vet[i+1] )
             { int x=vet[i];  vet[i]=vet[i+1]; vet[i+1]=x;  troca=1; }
         i=i+1;
       } while( i <n-1 );
 } while(troca);
```

16. Implemente um algoritmo simples de ordenação decrescente. O vetor de origem deve estar armazenado entre *ini1* e *fim1*, e o vetor ordenado entre *ini2* e *fim2*.

17. Implemente uma rotina denominada TOLOWER que, quando chamada, lê um valor da posição de memória denominada LETRA e, se este código de 8 bits representar uma letra em ASCII, retorna esta mesma letra, na mesma posição de memória, só que como uma **letra minúscula**. Em ASCII, as letras maiúsculas são representadas pela seqüência de códigos **041H** a **05AH** (**A** a **Z**, respectivamente) e as letras minúsculas são representadas pela seqüências de códigos **061H** a **07AH** (**a** a **z**, respectivamente). Se o código não representar uma letra, ele não deve ser alterado e a rotina TOLOWER retorna sem nada fazer.

18. Um utilitário muto utilizado no UNIX é o comando *grep*, que indica se uma determinada *string* encontra-se em um arquivo. Tanto as strings como os arquivos terminam com '0'. Implemente este algoritmo utilizando a linguagem de montagem do processador Cleópatra. Exemplo:

string	m	a	m	a	e	0

arquivo	x	a	m	a	m	a	m	e	s	m	a	m	a	e	x	z	0

Um possível algoritmo para este problema é:

```
cont = 0;     // diz quantas vez ocorreram a string em arquivo
ix1  = arquivo;
last = arquivo;
ix2  = string;
do {
   if( string[ix2] = arquivo[ix1] )   // caracter igual
     { if( ix2=string)
          last = ix1;     // marca a posição no arquivo onde começou a comparação
         ix1 ++;
       ix2 ++;
       if( string[ix2] = 0 )   // terminou string = sequencia encontrada
          { cont ++;
            last = ix1;
            ix2 = string;
          }
      }
   else
      { last ++;   // em caso de diferença incrementa a última posição de igualdade
        ix2 = string ;
        ix1 = last;
      }
}  while (arquivo[ix1] != 0)
```

BLOCO DE DADOS/CONTROLE

19. Completar a tabela abaixo com o código objeto referente às instruções do programa.

Endereço de Memória	Mnemônico/Dado	Código objeto (binário e hexa)
00H	LDA (Instrução 1, Byte 1)	
01H	25H (Instrução 1, Byte 2)	
02H	OR (Instrução 2, Byte 1)	
03H	#1AH (Instrução 2, Byte 2)	
04H	STA (Instrução 3, Byte 1)	
05H	17H (Instrução 3, Byte 2)	
	Memória de Dados	
17H	00H	
…	…	…
25H	D6H	1101 0110 D6H

A tabela abaixo já contém as microinstruções que atuam no BD para executar as três instruções da tabela acima. O exercício envolve completar a tabela de simulação, detalhando o conteúdo dos comandos da micro-instrução e o conteúdo dos registradores. A primeira linha, correspondente ao primeiro ciclo de relógio está preenchida, e é a seqüência MAR ←PC. *Dica: ZZ indica barramento de dados em alta impedância, ou seja, estão identificados os ciclos onde não há nem leitura nem escrita na memória*

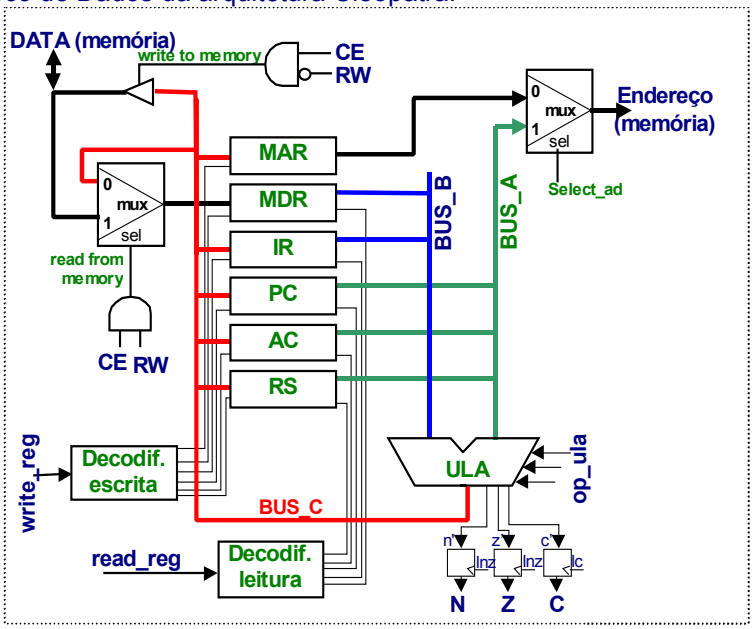
CK	DATAMEM DATABUD	write	read	ULA	ce	rw	Inz	Icv	MAR Address Bus	MDR	IR	PC	AC	RS		
0	ZZ														MAR ← PC	
1															MDR ← PMEM(MAR); PC++	F
2	ZZ														IR ←MDR	
3	ZZ														MAR ← PC	
4															MDR ← PMEM (MAR) ; PC++	
5	ZZ														MAR ← MDR	E
6															MDR ← PMEM (MAR)	
7	ZZ														AC ← MDR ; LNZ	
8	ZZ														MAR ← PC	
9															MDR ← PMEM(MAR); PC++	F
10	ZZ														IR ←MDR	
11	ZZ														MAR ← PC	
12															MDR ← PMEM (MAR) ; PC++	E
13	ZZ														AC ← AC OR MDR; LNZ	
14	ZZ														MAR ← PC	
15															MDR ← PMEM(MAR); PC++	F
16	ZZ														IR ←MDR	
17	ZZ														MAR ← PC	
18															MDR ← PMEM (MAR) ; PC++	
19	ZZ														MAR ← MDR	E
20															PMEM(MAR) ← AC	

20. A arquitetura Cleópatra necessita muitos ciclos de relógio para executar uma determinada instrução. São necessários 3 ciclos para a busca e de 1 a 7 ciclos para a execução de uma instrução. A figura abaixo mostra uma organização alternativa para o Bloco de Dados da arquitetura Cleópatra.

A seguinte modificação foi adotada: o endereço para a memória passa a ser selecionável ou pelo registrador MAR, como anteriormente, ou pelo barramento A, através da inserção de um multiplexador. A motivação para tal modificação for reduzir a seqüência abaixo para apenas um ciclo de clock:

t_i: MAR ← PC

t_{i+1}: MDR ← PMEM (MAR) ; PC ++



Pede-se:

a) Com esta modificação a seqüência acima pode ser realmente reduzida a um ciclo? Caso seja possível, mostre detalhadamente, através de um desenho, o caminho que os dados irão percorrer.

b) Qual a implicação desta modificação no bloco de controle ?

c) Mostrar para esta modificação a seqüência de microinstruções para:
- ciclo de busca;
- ciclo de execução em modo imediato, direto e indireto para a instrução de soma;
- ciclo de execução para a instrução STA em modo direto.

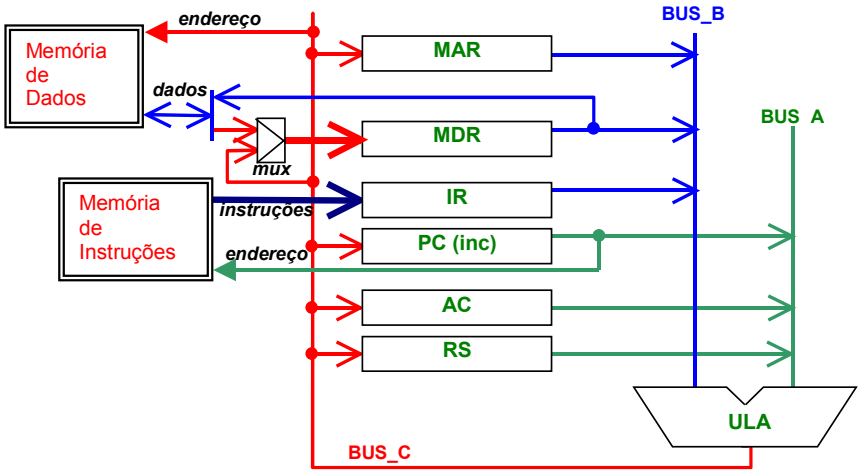
d) Faça depois uma **tabela** comparando o número de ciclos necessários para as seqüências indicadas, com a implementação original da Cleópatra.

e) Faça a microssimulação neste bloco operativo modificado do programa abaixo. Indique apenas as microinstruções e o conteúdo dos registradores e qualificadores.

Endereço na memória	Mnemônicos e/ou Dados	Código objeto (**preencher**)
0	LDA	
1	F5H	
2	ADD,I	
3	40H	
…	…	
40	AB	
…	…	
AB	BF	
…	…	
F5	41	

f) Quantos ciclos foram necessários? E para a Cleópatra original, quantos ciclos seriam? Qual o ganho final, em porcentagem ?

21. A arquitetura Cleópatra necessita muitos ciclos de relógio para executar uma determinada instrução. São necessários 3 ciclos para a busca e de 1 a 7 ciclos para a execução de uma instrução. A figura abaixo mostra uma organização alternativa para o bloco operativo da arquitetura Cleópatra. As seguintes modificações foram adotadas, visando reduzir o número de ciclos de clock por instrução:
    - 2 memórias separadas, uma para dados e outra para instruções.
    - Todas instruções obrigatoriamente com um operando. As operações sem operandos (not, rts e halt) passam a ter um operando "nop" (vazio). Isto permite ao PC apontar tanto para a instrução quanto para o operando.
    - Acesso à memória de dados pela saída da ULA. Esta característica permite acessar a memória de dados pelo MDR, PC, MAR ou qualquer outro registrador.
    - O registrador PC é auto-incrementável. Isto significa que ele é incrementado sem passar pela ULA. Há um micro-comando que o incrementa (PC++), com possível operação em paralelo na ULA.



Pergunta-se:

a) O ciclo de busca (carga do IR) e a carga do operando (carga do MDR) podem ser feitos em um ciclo de relógio, ao contrário da organização anterior, onde 3 ciclos eram necessários. Detalhe o ciclo de busca, através de micro-instruções, explicando o caminho percorrido pelos dados.

b) Para realizar uma execução em modo de endereçamento direto, 2 ciclos são necessários:

MDR ← PMEM(MDR) ; *mdr recebe o conteúdo apontado por ele próprio*
AC ← AC op MDR ; *realiza alguma operação sobre o acumulador*

Mostre as microinstruções correspondentes aos modos imediato e indireto.

c) Faça a microssimulação deste Bloco de Dados para o mesmo programa visto em aula. Note que o conjunto código de instrução e dado ocupam o mesmo endereço, porém em memórias separadas.

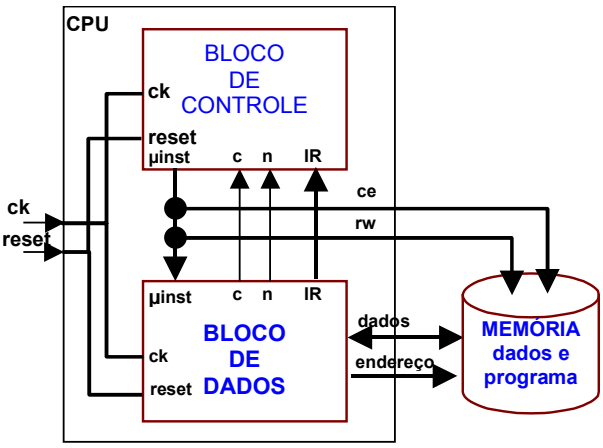
Instruções e Operandos nas Memória de Instruções e de Dados			
Endereço de Memória	**Instruções**	**Operandos**	**Código objeto (binário e hexa)**
00H	LDA (Instrução 1, Byte 1)	25H (Instrução 1, Byte 2)	
01H	OR (Instrução 2, Byte 1)	#1AH (Instrução 2, Byte 2)	
02H	STA (Instrução 3, Byte 1)	17H (Instrução 3, Byte 2)	
Dados na Memória de Dados			
17H	00H	0000 0000	00H
…	…	…	
25H	D6H	1101 0110	D6H

d) Quantos ciclos foram necessários? Comparando com a organização anterior, onde 21 ciclos foram necessários, qual foi o ganho?

e) Existe algum problema para executar o conjunto de instruções Cleópatra nesta nova arquitetura? Dica: Pense nos modos de endereçamento! Existe algum desperdício de memória nesta arquitetura?

22. A figura ao lado apresenta a arquitetura Cleópatra, porém visando reduzir o custo de implementação do hardware, o projetista suprimiu importantes sinais. Pede-se:

- Quais sinais foram suprimidos?
- Quais instruções não serão mais possíveis de serem executadas devido a esta supressão? Por quê ?
- Implemente um algoritmo para calcular a soma de dois vetores (vet3 ← vet1 + vet2), cujos endereços iniciais estão armazenados nos endereços F0, F1, e F2 e o número de elementos na posição F3. Importante: a(s) instrução(ões) que foram suprimidas pelo projetista **não** devem constar obviamente neste código.



23. A tabela abaixo ilustra um exemplo de programa a ser executado pelo Processador Cleópatra. OBSERVAÇÃO: **excelente exercício para visualizar a execução de um JUMP**

Endereço na memória	Mnemônicos e/ou Dados	Código objeto (preencher)
0	**LDA #**	
1	**90H**	
2	**JN ,R**	
3	**02H**	
4	**ADD #**	
5	**F4H**	
6	**AND #**	
7	**87H**	
...	**...**	

Pede-se:

a) Preencha o campo correspondente ao código objeto deste trecho de programa.

b) Complete na tabela abaixo o estado dos registradores a cada ciclo de clock e a micro-instrução correspondente para as 3 primeiras instruções executadas pelo programa. Assuma que todos os registradores inicialmente estão carregados com 0.

CK	DATA	MAR	MDR	IR	PC	AC	n/z/c/v	*Microinstrução*
0		0	0	0	0	0	0/0/0/0	
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								

c) Compute o número de ciclos de relógio para executar este programa.

d) Supondo que o relógio seja de 100 MHz, quanto tempo este programa leva para ser executado?

24. A tabela abaixo ilustra um exemplo de programa a ser executado no Bloco de Dados.

```
L1:   lda IDX,I          |        org #0F9H
      add #03H           | IDX:   DB #VET
      sta IDX,I          | VET:   DB #09H
      lda IDX            |        DB #12H
      add #01H           |        DB #13H
      sta IDX            |        DB #17H
      jz  f1,R           |        DB #1AH
      jmp L1,R           |        DB #89H
f1:   hlt
```

Pede-se:

a) Função do programa (quando ocorre a parada do mesmo?)

b) Preencha o campo correspondente ao código objeto deste de programa. Atenção aos deslocamentos relativos.

Endereço na memória	Mnemônicos e/ou Dados	Código objeto
00		
01		
02		
…		

c) Compute o número de ciclos de relógio para executar **este** programa (COM OS DADOS FORNECIDOS)

d) Supondo que o relógio seja de 20 MHz, em quanto tempo este programa vai ser executado ?

25. Após a execução de um programa de teste sobre o processador Cleópatra obteve-se o seguinte relatório, relacionando o número de execuções por tipo de instrução:

Tipo de Instrução e Modo de Endereçamento	Número de Execuções no Programa
STA direto	**320**
LDA/Lógico/Aritmética (ADD/OR/AND) direto	**450**
LDA/Lógico/Aritmética (ADD/OR/AND) indireto	**370**
JMP/JZ/JC relativo (considere o pior caso de tempo de execução para os saltos condicionais)	**100**

Pede-se (justifique cada resposta):

a) Número de ciclos para a execução do programa.

b) Dado um clock de 100 MHz, quanto tempo, em segundos, o programa leva para ser executado ?

26. Modos de endereçamento. Mostre como o modo de endereçamento indireto pode ser substituído pelo modo registrador indireto. Qual o benefício que tal modo, registrador indireto, pode ter sobre o indireto ?

27. Porque o ciclo de decodificação de uma dada instrução não consome ciclos de relógio na arquitetura Cleópatra. Explique, tendo em mente o funcionamento modelo do bloco de controle.

28. Quais as duas alternativas normalmente utilizadas para conectar registradores a barramento? Explique ambas, não esquecendo de comentar porque dois registradores não podem ser conectados diretamente a um barramento.

29. Explique o funcionamento de uma instrução de salto condicional, modo relativo de endereçamento, na arquitetura Cleópatra. Utilize as seguintes questões como guia para elaboração da resposta:
    a) Como são tratados os flags na unidade de controle (decodificação, seqüenciador, …) ?
    b) O que ocorre com o **micro-PC** em caso de salto executado? E não executado?
    c) O que ocorre com o **PC** em caso de salto executado? E não executado?

VHDL

30. Marque **V** ou **F** para as asserções abaixo, relacionadas a conceitos da linguagem de descrição de hardware - VHDL.

[ ] Apenas sinais podem ser utilizados para transferir informação entre processos.
[ ] Cada arquitetura é um conjunto de processos executando concorrentemente, compartilhando sinais.
[ ] Processos não podem ser usados conjuntamente com atribuição de sinais na mesma arquitetura.
[ ] Processos são muito bons para especificar sistemas digitais porque estes funcionam seqüencialmente, como nos processos
[ ] É possível especificar eventos futuros em atribuições de sinais.
[ ] Todos os processos em uma arquitetura estão ativos todo tempo.
[ ] Execução de um processo termina quando a cláusula end process é alcançada.
[ ] Um processo com lista de sensitividade não pode conter waits.
[ ] Processos especificam operações sequenciais para a descrição do comportamento de sistemas digitais.
[ ] A variável de controle em um for loop deve ser declarada no início de seu processo.
[ ] Sinais em um processo são alterados quando este processo é suspenso.
[ ] Tanto sinais quanto variáveis podem ser utilizados para armazenar informação temporal.
[ ] Descrições estruturais podem ser hierárquicas.
[ ] A mesma entidade pode ser utilizada em diferentes arquiteturas.
[ ] Para representar um barramento de 8 bits utiliza-se o tipo byte.
[ ] A ordem na qual a especificação dos bits é feita em um vetor não é importante.
[ ] Todas as portas declaradas em entities devem ter associado um modo de acesso.
[ ] Todos os sinais de um sistema são declarados na entidade.
[ ] Descrições estruturais são compostas de componentes e sinais.
[ ] Todos os componentes devem ser especificados utilizando descrições comportamentais.
[ ] Circuito a ser testado em um test_bench deve ser referenciado como um componente.
[ ] Estímulos em um test_bench devem ser especificados de maneira sequencial (em processos).
[ ] Um test_bench não contém portas.
[ ] Combinar estilo de descrição estrutural e comportamental em um mesmo arquivo é ilegal em VHDL.
[ ] Uma declaração de porta de entrada pode ser somente in ou out.
[ ] Portas são sinais, e não variáveis.
[ ] Estados em máquina de estados são normalmente declarados como enumerações.
[ ] Elementos em um array podem ser de tipos diferentes.
[ ] Boolean true é equivalente ao bit 1.
[ ] Atribuição de sinais podem ser feitas tanto com “=>” ou “<=” dependendo da necessidade do projetista.

31. Desenhe um diagrama de esquemáticos que tenha funcionalidade equivalente ao código VHDL abaixo. Não se esqueça de desenhar os limites da entidade, identificando entradas e saídas do circuito. Justifique com palavras seu desenho.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity P1_1_001 is
port( a,b,c,d : in std_logic_vector(5 downto 0);
      reset,clock, ch1, pm : in std_logic;
      s : out std_logic_vector(5 downto 0));
end P1_1_001;

architecture P1_1_001 of P1_1_001 is
signal pm1,pm2,res : std_logic_vector(5 downto 0);

begin

process (clock, reset)
begin
               if reset='1' then s <= (others=>'0');
               elsif clock'event and clock='1' then s <= res;
               end if;
end process;

res <= pm1+pm2 when pm='1' else pm1-pm2;
pm1 <= a when ch1='0' else b;
pm2 <= c when ch1='0' else d;

end P1_1_001;
```

32. Desenhe um diagrama de esquemáticos que tenha funcionalidade equivalente ao código VHDL abaixo.

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_unsigned.all;

entity periferico is
       port( i0, i1, i2, i3, clock, reset : in std_logic;
             saida: out std_logic_vector(3 downto 0) );
end periferico;

architecture arch1 of periferico is
   signal si, s3, s2, s1, s0 : std_logic;
   signal cont : std_logic_vector(1 downto 0);
begin
       saida <= s3 & s2 & s1 & s0;

       process(reset, clock)
       begin
              if reset='1' then
                     s3<='0';    s2<='0';   s1<='0';   s0<='0';
              elsif clock'event and clock='0' then
                     s3 <= si;   s2 <= s3;  s1 <= s2;  s0 <= s1;
              end if;
       end process;

       process(reset, clock)
       begin
              if reset='1' then
                     cont <= "00";
              elsif clock'event and clock='1' then
                     cont <= cont + 1;
              end if;
       end process;

si <=  i0 when cont="00" else       i1 when cont="01" else
       i2 when cont="10" else       i3 when cont="11" ;

end arch1;
```

33. Considere o programa abaixo, escrito na linguagem de descrição de hardware VHDL, composto da descrição do circuito (entity teste) e do módulo de teste (entity tb) .

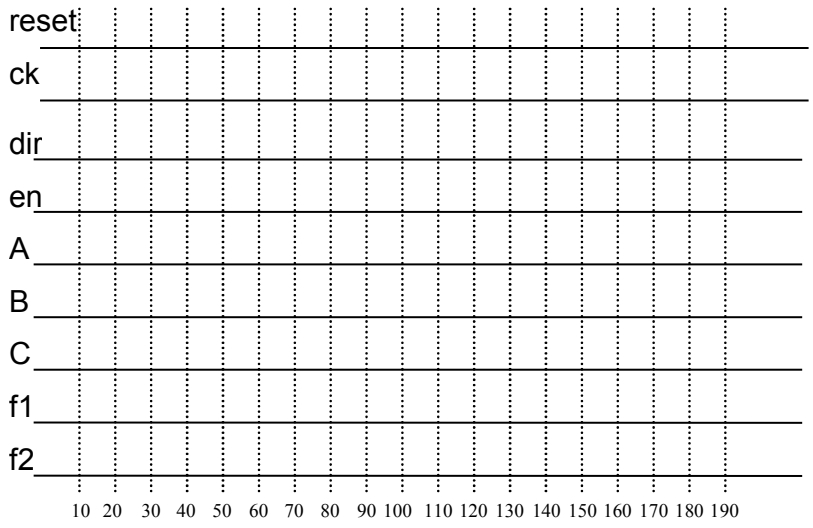
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity teste is
 port( clock, reset, dir : in std_logic;
       entrada: in STD_LOGIC_VECTOR(3 downto 0);
       f1,f2 : out STD_LOGIC_VECTOR(3 downto 0)
      );
end teste;

architecture a1 of teste is
  signal A, B, C : STD_LOGIC_VECTOR(3 downto 0);
begin

  process(clock, reset)
    begin
      if   reset = '1'   then
        A <= "0000";  B <= "0000";   C <= "0000";
   elsif clock'event and clock='1' then
        if dir='1' then
           A <= entrada;   B <= A;   C <= B;
        else
           A <= B;         B <= C;   C <= A;
        end if;
        f2 <= A + B;
   end if;
 end process;

 f1 <= A + B;

end a1;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity tb is
end tb;

architecture a2 of tb is
 signal ck, dir, reset : std_logic;
 signal entr,f1: STD_LOGIC_VECTOR(3 downto 0) ;
begin

  cont1: teste port map
    ( clock=>ck, reset=>reset,
      dir=>dir, entrada=>entr, f1=>f1);

   reset<=  '1', '0' after 8ns;
   dir<= '1', '0' after  70 ns, '1' after 170ns;
   entr <= " 0001",  "0110" after 30ns,
             "1001" after 50 ns,
             "1111" after 170ns;

    process
        begin
            ck <= '1', '0' after 10ns;
            wait for 20ns;
        end process;

  end a2;

-- assuma que a declaração do componente está feita
--  em um package, não mostrado
```

Pede-se, a partir do código VHDL acima:

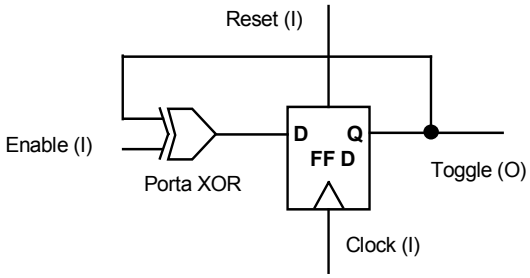
a) Mostre uma implementação do circuito teste (desconsidere o test_bench) sob a forma de um diagrama de esquemáticos, usando registradores, multiplexadores, entradas e saídas, operadores lógicos e aritméticos, etc. Assinale claramente quais são as entradas e saídas no diagrama. Todos os sinais da descrição devem aparecer no diagrama.

b) Complete o diagrama de tempos abaixo, para pelo menos os 180 ns iniciais de simulação.

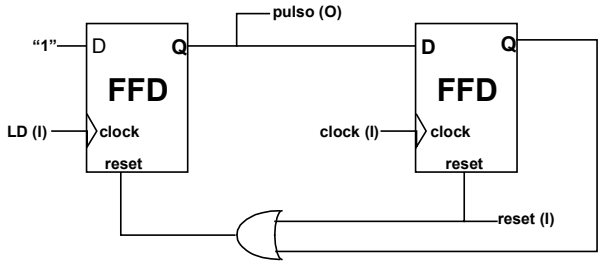


c) O que diferencia as atribuições aos sinais “f1” e “f2”? Como eles comportam-se no diagrama de tempos, e qual seria a atribuição correta? Por quê ?

34. A partir do diagrama de esquemáticos abaixo, gere uma descrição VHDL com a mesma funcionalidade, sob a forma de um par entidade-arquitetura. Não se esqueça de definir as entradas e saídas corretamente. Para facilitar, saídas estão marcadas com (O) e entradas com (I). Todos os fios são simples, e não barramentos. Caso necessite, defina sinais internos e anote-os no esquemático.



35. A partir do diagrama de esquemáticos abaixo, gere uma descrição VHDL com a mesma funcionalidade, sob a forma de um par entidade-arquitetura. Não se esqueça de definir as entradas e saídas corretamente. Para facilitar, a saída está marcada com (O) e entradas com (I). Todos os fios são simples, e não barramentos. Caso necessite, defina sinais internos e anote-os no esquemático.



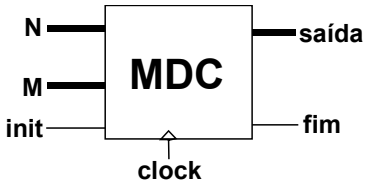
36. Implemente o algoritmo descrito abaixo em linguagem VHDL, assim como o test_bench para a simulação. A implementação VHDL pode ser comportamental.

Algoritmo de Euclides para cálculo de MDC (máximo divisor comum):

Dados dois inteiros positivos, m e n, opere (**m pode ser maior que n, conforme exemplo abaixo**):

a) Encontre o resto r da divisão de m por n;
b) Se r=0 o algoritmo termina e n é a resposta;
c) Senão faça m←n e n←r. Volte ao passo (1).

Dicas: Use operadores VHDL de cálculo de resto da divisão tais como rem ou mod.



- ➔ N, M: entradas (integer)
- ➔ saída: valor do máximo divisor comum (integer)
- ➔ o sinal init sinaliza que o algoritmo pode começar, e o sinal fim deve subir quando o algoritmo terminar.
- ➔ importante: o laço do algoritmo deve ser controlado pelo sinal de clock.

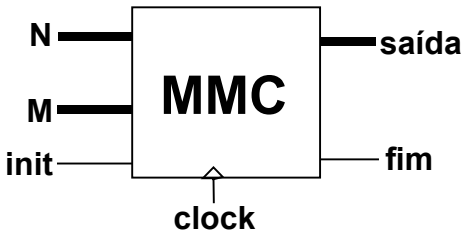
Exemplo:

nn	mm	resto	saída
20	36	1	
16	20	16	
4	16	4	
0	14	0	**4**

ou

nn	mm	resto	saída
36	20	1	
20	36	20	
16	20	16	
4	16	4	
0	14	0	**4**

37. Implemente o algoritmo descrito abaixo em linguagem VHDL, assim como o test_bench para a simulação. A implementação VHDL deve utilizar os comandos + para soma, rem para resto da divisão, / para divisão, * para multiplicação, etc. A interface do circuito deve ser:



- ➔ N, M: entradas (integer)
- ➔ saída: valor do mínimo múltiplo comum (integer)
- ➔ o sinal init sinaliza que o algoritmo pode começar, e o sinal fim deve subir quando o algoritmo terminar
- ➔ importante: o laço do algoritmo deve ser controlado pelo sinal de clock

Algoritmo para cálculo de MMC (mínimo múltiplo comum):

Observação: o algoritmo consultará uma tabela ordenada de números primos [2,3,5,7,11,13, etc], armazenada em uma memória tipo ROM.

Dados dois inteiros positivos, m e n, opere:

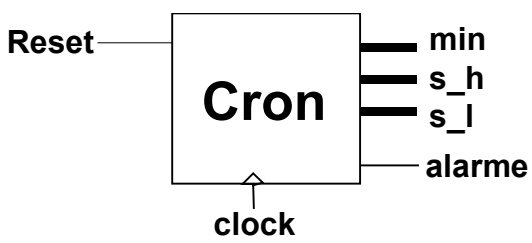
1) Crie um ponteiro para a posição inicial da tabela de números primos;
2) Inicializar o resultado em 1;
3) Caso m seja divisível pelo número primo corrente (resto=0), substitua-o pelo quociente correspondente;
4) Caso n seja divisível pelo número primo corrente (resto=0), substitua-o pelo quociente correspondente;
5) Caso alguma substituição tenha sido feita nos passos (2) ou (3) resultado = resultado*primo corrente, senão incremente o ponteiro da tabela de primos;
6) Se m e n forem iguais a 1 terminar a execução, caso contrário voltar ao passo (3).

Exemplo:

m	n	primo corrente	resultado	
12	9	2	1	→ condições iniciais
6	9	2	2	→ final do passo 5, primeiro loop
3	9	3	4	
1	3	3	12	
1	1	3	(36)	→ resultado final

38. Esta questão consiste na implementação de um cronômetro decrescente com tempo fixo de 3 minutos, para uso em jogo de xadrez contra o relógio em VHDL. A especificação da interface do cronômetro aparece na figura abaixo. O uso do cronômetro é o seguinte. O jogador com as pretas aciona o reset, que dispara a contagem de um período de 3 minutos. Este é o tempo que as brancas têm para iniciar o jogo. Após realizar o movimento, o jogador das brancas aciona o reset para iniciar a contagem do tempo para as pretas realizarem a jogada e assim por diante até o fim jogo. Caso alguém demore mais do que 3 minutos para jogar e acionar o reset, o cronômetro tranca no valor 0:00 e coloca a saída alarme em 1, o que dispara um sinal sonoro. Pede-se:

a) O código VHDL do cronômetro;
b) O código VHDL do TestBench;
c) Responda se o relógio especificado tem uma freqüência adequada ou não e justifique;
d) Descreva e justifique com palavras as alterações necessárias para que este cronômetro possa ser usado em partidas onde o tempo para fazer cada jogada seja diferente de três minutos, por exemplo 0:30, 1:00, 1:30 ou 5:00 minutos.



- ➔ Reset: sinal de entrada de um bit, onde cada pulso com duração mínima de um ciclo de relógio reinicializa o cronômetro e a contagem.
- ➔ clock: sinal de relógio de freqüência 1Hz usado com referência para a contagem.
- ➔ min, s_h, s_l: sinais de saidas de 4 bits contendo o valor de contagem em minutos, dezenas de segundos e unidades de segundo, respectivamente, em código BCD.
- ➔ alarme: sinal de um bit que vai para um sempre que o contador chegar a 0.

39. Gere o código VHDL equivalente ao conjunto de esquemáticos abaixo e diga o que faz o circuito. Preserve a hierarquia de dois níveis dos esquemáticos em VHDL.

Informações úteis:

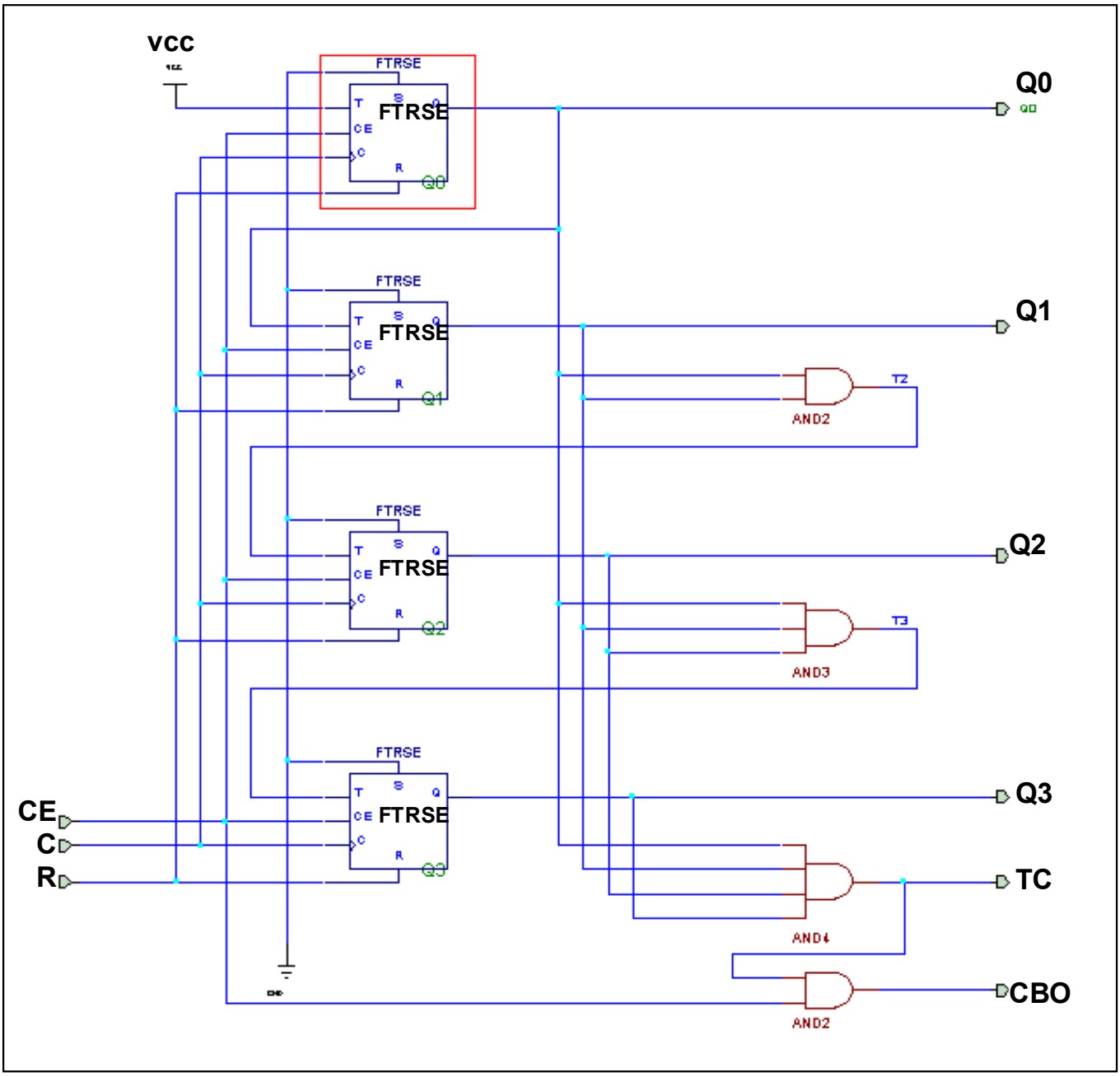
a) O primeiro esquemático tem o nome CB4RE;

b) A entrada T do componente FTRSE no topo do primeiro esquemático está conectada ao valor lógico constante '1' (vcc).

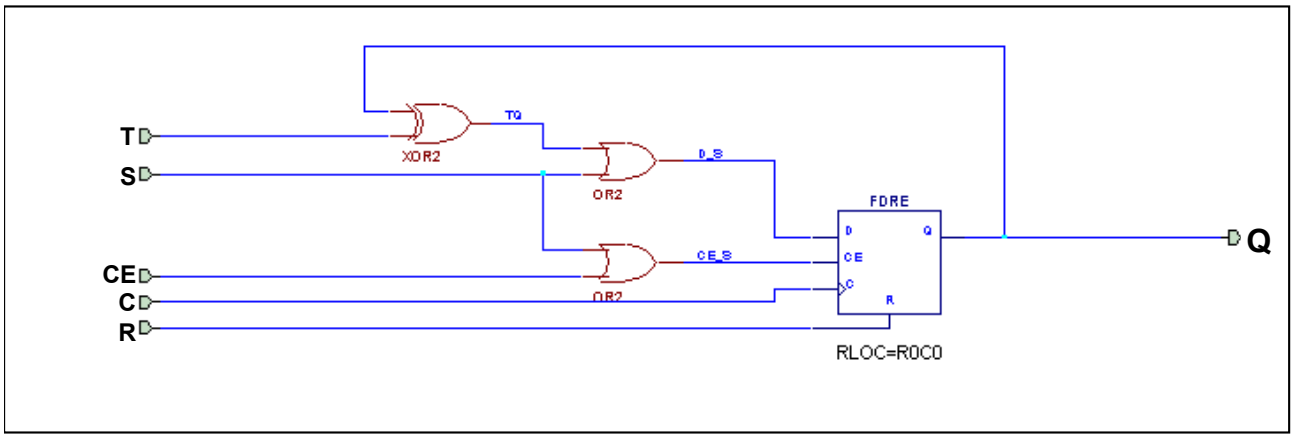
c) As entradas e saídas estão claramente especificadas em ambos esquemáticos. Não existem outras além das identificadas nos esquemáticos. Note que o segundo esquemático define a estrutura interna dos componentes FTRSE do primeiro esquemático. A entrada C do primeiro esquemático é (obviamente) o sinal de relógio do circuito.

d) O flip-flop FDRE do segundo esquemático possui entrada de reset assíncrono (R) ativo em 1 e uma entrada de habilitação de escrita (CE), também ativa em 1.

CB4RE

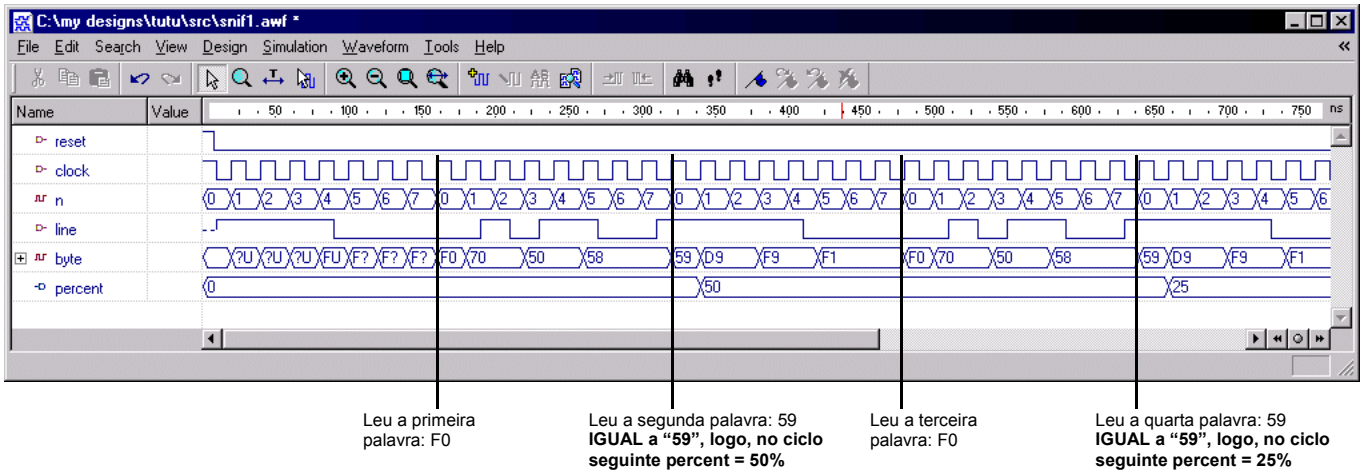


FTRSE



40. Implemente em VHDL um sistema digital “sniffer” (farejador), que compute o percentual de um caracter ASCII dado, que está passando numa linha de dados. Suponha que este cálculo deve ser inicializado por um sinal reset e sincronizado com o sinal de clock do sniffer. Lembre-se que a linha é serial (transmite um bit de cada vez) e que um caracter ASCII é representado por um código de 8 bits. Use a entidade abaixo e assuma que o caracter em questão é a maiúscula Y (em hexa, 59, em binário 0101 1001). Como dica, dá-se abaixo a forma de onda do funcionamento do circuito.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity sniffer is
port( clock, reset : in STD_LOGIC;
          line: in STD_LOGIC;
           percent : out INTEGER
         );
end teste;
```



41. Considere o código VHDL abaixo:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is port(X, clock : in std_logic; Z: out std_logic);
end;

architecture A of fsm is
      type STATES is (S0, S1, S2, S3);
      signal scurrent, snext : STATES;
begin
      controle: process
      begin
            wait until clock'event and clock='1';
            scurrent <= snext;
      end process;
```

```
        combinacional: process(scurrent, X)
        begin
            case scurrent is
              when S0 =>  if X='0' then Z<='0'; snext <= S1;
                                   else Z<='1'; snext <= S2;
                          end if;
              when S1 =>  if X='0' then Z<='1'; snext <= S2;
                                   else Z<='0'; snext <= S1;
                          end if;
              when S2 =>  if X='0' then Z<='1'; snext <= S2;
                                   else Z<='0'; snext <= S3;
                          end if;
              when S3 =>  if X='0' then Z<='0'; snext <= S0;
                                   else Z<='0'; snext <= S1;
                          end if;
            end case;
        end process;
end A;
```

Pede-se:

a) Diagrama de transição de estados da máquina de estados finitos acima.

b) Trata-se de uma máquina de Moore ou Mealy? Justifique.

c) Implemente um *test bench* para esta máquina de estados, testando todos os casos possíveis. Faça o diagrama de tempos para o seu test_bench, com ao menos 15 transições do clock, explicando cada transição.

PIPELINES

42. Suponha que os trechos de programas abaixo sejam executados no pipeline visto em aula (DLX):

Seqüência 1:

```
ADD   $16, $18, $15    ; soma os conteúdos dos regs 15 e 18, armazenando o resultado no reg 16
LX    $24, 4($16)      ; busca o conteúdo apontado por reg 16 + 4, armazenando no reg 24
LX    $25, 8($16)      ; busca o conteúdo apontado por reg 16 + 8, armazenando no reg 25
SLT   $1, $25, $24     ; reg1 recebe 0 se reg25 ≥ reg24, senão reg1 recebe 1
BEQ   $1, $0,  saida   ; se o reg1 for igual a 0 (reg0 é constante 0) realiza um salto para a "saída"
```

Seqüência 2:

```
SUB   $2,   $1,  $13   ;  reg2 ← reg1 – reg3
AND   $12, $2,  $5     ;  reg12 ← reg2 and reg4
OR    $13, $6,  $2     ;  reg13 ← reg6 or reg2
ADD   $14, $12, $12    ;  reg14 ← reg12 +  reg12
SW    $15, 100($13)    ;  posição de memória( reg13 + 100) ← reg15
```

Para cada seqüência acima:

a) Qual o número mínimo **ideal** de ciclos de clock para a execução deste programa no pipeline do DLX?

b) Determine o número **real** de ciclos de clock para executar este programa, desenhando a execução em um diagrama de tempos e explique a causa das bolhas, caso estas existam;

c) Quais as soluções possíveis para reduzir o número de bolhas (apenas cite-as) ?

43. Partindo da implementação pipeline do processador DLX, segundo a bibliografia de Patterson e Hennessy, suponha os seguintes tempos para os estágios do pipeline da máquina:

Estágio	Busca de Instrução	Leitura dos Registradores Fonte	Operação com a ULA	Leitura/Escrita na memória de Dados	Atualização dos Registradores Destino
Duração da Execução (em ns)	13	9	12	15	11

Dados estes tempos, suponha a execução do seguinte trecho de programa, durante a qual **certamente** não ocorrerão **bolhas** no pipeline:

```
ADD   R1, R2, R3       ; R1 ← R2 + R3
LW    R4, 30(R2)       ; R4 ← PMEM(R2+30)
BEQZ R2, 10            ; PC ← PC + 10 se R2=0, senão PC ← PC+4
```

Pede-se:

a) O tempo de execução deste programa, caso executado na máquina sem pipeline.

b) Tempo de execução deste programa na implementação pipeline descrita na tabela.

c) Quantas vezes a implementação pipeline é mais rápida que a implementação sem pipeline para este programa, ou seja, qual é o ganho da máquina pipeline em relação à não pipeline?

d) Responda o item (c) para um programa de 1000 instruções, igualmente sem bolhas.

e) Existe algum programa para o qual o ganho seja **pelo menos** igual ao número de estágios do pipeline? Justifique sua resposta.

44. O tempo ideal ***T*** de execução de um programa em uma máquina com pipeline de ***p*** estágios, cujo estágio mais lento consome ***t*** segundos, para um programa com ***n*** instruções é:

$$T = t.[p+(n-1)]$$

Supondo dois trechos de código sem dependência. O primeiro trecho de programa possui 500 instruçoes e executa em 50,4 μs (tempo decorrido entre a busca da primeira instrução e o término da última instrução do programa). O segundo trecho de programa possui 1000 instruções e executa em 50,4 100,4 μs. Pergunta-se:

a) qual o número ***p*** de estágios?

b) qual a latência ***L*** do pipeline

c) qual o tempo de execução do estágio mais lento deste pipeline?

ALGUMAS SOLUÇÕES PARA PROGRAMAÇÃO ASSEMBLY

Observação: todas foram testadas no simulador distribuído na página da disciplina.

```
;*********************************************************************
;  SOMA UMA CONSTANTE A UM VETOR
;*********************************************************************
;
        lda eend    ; inicializa ponteiro
        sta pvet    ;
repete: lda n       ; verifica se atualizou todos elementos
        jz  fim     ; salta se vetor inicialmente vazio ou se fim
        lda pvet,I  ; senão, toma elemento e põe no acumulador
        add conste  ; adiciona constante
        sta pvet,I  ; armazena de volta na mesma posição
        lda n       ; atualiza contador de elementos subtraindo 1
        add #0ffh
        sta n
        lda pvet    ; atualiza ponteiro para próximo elemento
        add #01h
        sta pvet
        jmp repete
fim:    hlt

; Area de dados

        org 30h
.DATA
vet:    db #03h     ; vetor, um elemento por linha
        db #08h
        db #3bh
        db #0f5h
n:      db #04h     ; tamanho do vetor
eend:   db vet     ; endereço do endereço de início do vetor end1
pvet:   db #00h     ; ponteiro para avançar no vetor
conste: db #10h     ; constante a somar
.ENDDATA

;*********************************************************************
;SOMA DE DOIS VETORES  VET3 <- VET1 + VET2
;*********************************************************************
.code
       LDA     F3
       ADD     #0FFH
       STA     F3

loop:  LDA     F0,I      ; soma vet1 com vet2 armazenando em vet3
       ADD     F1,I
       STA     F2,I

       LDA     F0         ; incrementa o ponteiro de vet1
       ADD     #01H
       STA     F0

       LDA     F1         ; incrementa o ponteiro de vet2
       ADD     #01H
       STA     F1

       LDA     F2         ; incrementa o ponteiro de vet4
       ADD     #01H
       STA     F2

       LDA     F3          ; decrementa o número de elementos
       ADD     #0FFH
       STA     F3

       JN      FIM,R
       JMP     LOOP,R
FIM:   HLT
.endcode

.data
F0:    DB      vet1
F1:    DB      vet2
F2:    DB      vet3
F3:    DB      #05H
```

```
vet1:   DB      #01H, #02H, #03H, #04H, #05H    ; declaração dos vetores
vet2:   DB      #01H, #02H, #03H, #04H, #05H
vet3:   DB      #00H, #00H, #00H, #00H, #00H
.enddata

;*********************************************************************
;  encontra o máximo e o mínimo de dois INTEIROS POSITIVOS
;*********************************************************************
.code
LOOP:   LDA A
        NOT
        ADD #01
        ADD B
       JN  AMAIOR
       LDA B
        STA MAX
        LDA A
        STA MIN
        HLT
AMAIOR:
       LDA A
        STA MAX
        LDA B
        STA MIN
        HLT
.endcode

.data
A:      DB #61H
B:      DB #60H
max:    DB #00H
min:    DB #00H
.enddata


;*********************************************************************
;  somatório dos máximos entre dois vetores
;*********************************************************************
.code
LOOP:     LDA v1,I    ; encontra que é o maior entre v1 e v2
          NOT
          ADD #01
          ADD v2,I
          JN  v1maior
          LDA v2,I
          JMP L1
v1maior:  LDA v1,I

L1:       ADD maxant  ; soma o máximo atual com o max anterior
          STA v3,I    ; atualiza o vetor v3 com o máximo anterior

          STA maxant  ; atualiza o máximo anterior

          LDA V1      ; incrementa os ponteiros
          ADD #01h
          STA V1
          LDA V2
         ADD #01h
          STA V2
          LDA V3
         ADD #01h
          STA V3

          LDA n
         ADD #0FFh
          STA n
          JZ  FIM,R
          JMP LOOP,R
FIM:      HLT
.endcode

.data
maxant: db #00h
tmp:    db #00h
n:       db #05h

v1:     db E1
v2:     db E2
v3:     db E3
E1:     db #08h, #03h, #50h, #12h, #03h
```

```
E2:     db #12h, #01h, #30h, #07h, #06h
E3:      db #00H

.enddata


;*********************************************************************
;  cálculo dos N primeiros números da série de Fibonnacci
;*********************************************************************
.code
INI:     LDA #00H
        STA IX,I
        JSR INC
        LDA #01H
        STA IX,I
        JSR INC
LOOP:    LDA A1,I
        ADD A2,I
        STA IX,I
        JSR INC
        JMP LOOp
INC:     LDA N
         ADD #0FFH
         JZ FIM
        STA N
         LDA A1
        STA A2
        LDA IX
        STA A1
        ADD #1H
        STA IX
        RTS
FIM: HLT
.endcode

.data
 N: DB #10H
IX: DB P0
A1: DB #1H
A2: DB #0H
P0: DB #0H
.enddata

;*********************************************************************
;Programa que calcula a multiplicação entre dois inteiros através de
;sucessivas somas
;*********************************************************************
.code
 begin:
        lda n2
        jz fim
        lda n1
        add ml
        sta ml
        jc inc,r
 volta:
        lda n2
        add #0ffh
        sta n2
        jz fim,r
        jmp begin,r
 inc:
        lda mh
        add #01h
        sta mh
        jmp volta,r

 fim:
        hlt
.endcode

.data
        n1: db #12h  ; multiplicando
        n2: db #34h  ; multiplicador
        ml: db #00h  ; resultado final
        mh: db #00h  ; resultado final
.enddata
```

--
-- MDC
--

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mdcB is
    port (
        n, m: in INTEGER;
        ck: in std_logic;
        saida: out INTEGER
    );
end mdcB;

architecture mdcB of mdcB is
  signal mm, nn : integer;            --  sinais temporarios
begin

    process
      variable resto : integer;
     begin
      resto := 1;
      nn <= n;
      mm <= m;
      wait until ck'event and ck='1';   -- inicializa as variaveis

      while resto /= 0  loop
          if(nn/=0) then
             resto := mm rem nn;
             saida <= nn;
             mm <= nn;
             nn <= resto;
             wait until ck'event and ck='1';  -- atualiza sinais
          end if;
      end loop;

    end process;
end mdcB;
```

--
-- MMC
--

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mmc is
    port (
        N,M: in integer;
        init,ck : in STD_LOGIC;
        saida: out integer;
        fim: out STD_LOGIC
    );
end mmc;

architecture mmc of mmc is
   type mem_rom is array (0 to 127) of integer;
   constant primo : mem_rom := (2, 3, 5, 7, 11, 13, 17, 19, 21, 23, 29, others=>0);
begin

    process
        variable i, flag, nn, mm, mmc  : integer;
    begin

       wait until init'event and init='1';  -- reset
       mmc := 1;
       nn  := N;
       mm  := M;
       i := 0;

       loop
        flag := 0;
```

```
               if (nn rem primo(i))=0 then  nn := nn / primo(i);  flag := 1;  end if;

              if (mm rem primo(i))=0 then  mm := mm / primo(i);  flag := 1;  end if;

              if flag=1 then   mmc := mmc * primo(i);
                   else    i := i + 1;
                end if;

               if (nn=1) and (mm=1) then  fim<='1';  else fim<='0'; end if;

              saida <= mmc;
               wait until ck'event and ck='1';   -- atualiza os sinais

       exit when fim='1';

      end loop;
       end process;

end mmc;


library IEEE;
use IEEE.std_logic_1164.all;

entity tb is
end tb;

architecture teste of tb is
    type mem_rom is array (0 to 127) of integer;
    constant N : mem_rom := (12,  8, 5, 27, 11, 13, 17, 19, 21, 23, 29, others=>0);
    constant M : mem_rom := ( 9, 54, 3, 15, 7, 11, 13, 17, 19, 21, 23, 29, others=>0);

    component mmc
       port ( N,M: in integer;
        init,ck : in STD_LOGIC;
        saida: out integer;
        fim: out STD_LOGIC
       );
    end component;

    signal a,b, saida, cont : integer := 0;
    signal reset, init, ck, fim : STD_LOGIC;

    begin

       mm1:  mmc port map (N=>a, M=>b, init=>init, ck=>ck, saida=>saida, fim=>fim);

       -- inicio do processo
       reset <= '0', '1' after 2ns, '0' after 5ns;

       -- gerador de clock
       process
        begin
         ck <= '1' after 10ns, '0' after 20ns;
         wait for 20ns;
       end process;

       -- fornece os dados quando sobe o reset ou o fim
       process
        begin

         wait on fim, reset;

          if (fim'event and fim='1') or (reset'event and reset='1')
            then cont <= cont + 1;
                 a <= N(cont);
                 b <= M(cont);
                 init <= '1' after 30ns, '0' after 35ns;
                 wait for 10ns;
         end if;
       end process;

    end teste;
```

--
-- sniffer
--

library IEEE;

library IEEE;

```
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity teste is
          port( clock, reset, line : in std_logic;
                percent: out integer
              );
end teste;

architecture a1 of teste is
   signal c1, c2, n : integer;
   signal byte : STD_LOGIC_VECTOR(7 downto 0);
begin

  process(clock, reset)
   begin
     if  reset = '1'  then
          n <= 0;
          c1 <= 0;
          c2 <= 0;
          percent <= 0;
    elsif clock'event and clock='1' then

          byte(7-n) <= line;        -- montagem do byte

          -- teste de igualdade HINT: quando n=0 !!!!
          if n=0 and  byte=x"59" then
               c2 <= c2 +1;
              if c1>0 then  percent <= (c2+1*100)/c1; end if;
          end if;

          if n=7 then         --- controle de palavras recebidas
                n <= 0;
                c1 <= c1 +1;
             else
                n <= n +1;
           end if;

       end if;
   end process;
end a1;
```

```
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity tb is
end tb;

architecture a2 of tb is

  component teste
       port( clock, reset, line : in std_logic;  percent: out integer);
  end component;

  constant stringao : std_logic_vector(63 downto 0) :=
        "1111000001011001111100000101100111110000010110011111000001011001";
   signal ck, line, reset : std_logic;
   signal percent : integer ;

begin
     cont1:  teste port map(clock=>ck, reset=>reset, line=>line,     percent=>
percent);

      reset <=  '1', '0' after 8ns;

      process
          begin
             wait for 10 ns;
             for i in 63 downto 0 loop
               line <= stringao(i);
               wait for 20 ns;
             end loop;
          end process;

          process
             begin
               ck <= '1', '0' after 10ns;
               wait for 20ns;
           end process;
end a2;
```

--
-- TRANSMISSÃO DE DADOS PELA SERIAL – apenas exemplo, não está na lista
--

```
    --
    -- protocolo de transmissao de dados por uma serial
    --
    library IEEE;
    use IEEE.std_logic_1164.all;

    entity transmissor is
        port ( clock,reset,send: in STD_LOGIC;
               palavra: in STD_LOGIC_VECTOR (7 downto 0);
               busy, linha: out STD_LOGIC
             );
    end transmissor;

    architecture transmissor of transmissor is
        type STATES is (REP, S0, S1, S2,S3, S4, S5, S6, S7, S8, S9);
        signal EA, PE : STATES;
    begin
       process(send,clock)
       begin
              if reset='1' then EA <= REP;
              elsif clock'event and clock='1' then
                   EA <= PE;
              end  if;
       end process;

       process(EA, send)
       begin
              case EA is
                     when REP =>    if send='1' then PE<=S0; else PE<=REP;   end if;
                                 busy <= '0';
                                 linha <= '1';
                     when S0 =>     PE<=S1;       busy <= '1';   linha <= '0';
                     when S1 =>     PE<=S2;       busy <= '1';   linha <=  palavra(7);
```

```
                    when S2 =>     PE<=S3;       busy <= '1';   linha <=  palavra(6);
                    when S3 =>     PE<=S4;       busy <= '1';   linha <=  palavra(5);
                    when S4 =>     PE<=S5;       busy <= '1';   linha <=  palavra(4);
                    when S5 =>     PE<=S6;       busy <= '1';   linha <=  palavra(3);
                    when S6 =>     PE<=S7;       busy <= '1';   linha <=  palavra(2);
                    when S7 =>     PE<=S8;       busy <= '1';   linha <=  palavra(1);
                     when S8 =>    PE<=S9;       busy <= '1';   linha <=  palavra(0);
                    when S9 =>     PE<=REP;      busy <= '1';   linha <=  '0';
            end case;
   end process;
end transmissor;

--
-- A TEST_BENCH
---
library ieee;
use ieee.std_logic_1164.all;

entity transmissor_tb is
end transmissor_tb;

architecture TB_ARCHITECTURE of transmissor_tb is
   component transmissor
   port ( clock,reset,send: in STD_LOGIC;
           palavra: in STD_LOGIC_VECTOR (7 downto 0);
           busy, linha: out STD_LOGIC
         );
   end component;

  signal busy, linha, clock,reset,send: std_logic;
  signal palavra : std_logic_vector(7 downto 0);
begin
   UUT : transmissor
          port map
                 (clock => clock,
                 reset => reset,
                 send => send,
                 palavra => palavra,
                 busy => busy,
                 linha => linha );

   -- gerador de clock
      process
       begin
        clock <= '1' after 5ns, '0' after 10ns;
        wait for 10ns;
      end process;

      reset <= '1', '0' after 3 ns;

      send <= '0', '1' after 23 ns, '0' after 50 ns, '1' after 160ns, '0' after 200 ns;

      palavra <= "11010001", "00100110" after 150ns;

end TB_ARCHITECTURE;
```

--

-- CONTADOR COM TEST_BENCH – apenas exemplo, não está na lista

--

```
            library IEEE;
            use IEEE.std_logic_1164.all;
            use IEEE.std_logic_unsigned.all;
            entity contup is
                        port ( clock, reset, Load, Enable: In std_logic;
                               DATABUS : In Std_logic_Vector (5 downto 0);
                               Upcount2 : Out Std_logic_Vector (5 downto 0));
            end contup;

            architecture RTL of contup is
                   Signal Upcount : std_logic_Vector (5 downto 0);
            begin
                   Upcount2 <= Upcount;

                   Upcounter : Process (clock, reset)
                   begin
                          if reset = '1' then
                                 Upcount <= "000000";
                          elsif clock'event and clock='1' then
```

```
                        if ENABLE = '1' then
                                if LOAD = '1'     then Upcount <= DATABUS;
                                   else Upcount <= Upcount + 1;
                                                        -- precisa e.g. ieee_std_logic_unsigned
                                   end if;                      -- para a soma de bit
                                end if;
                        end if;
                end process Upcounter;
        end RTL;


library ieee;
use ieee.STD_LOGIC_UNSIGNED.all;
use ieee.std_logic_1164.all;


entity contup_tb is
end contup_tb;

architecture TB_ARCHITECTURE of contup_tb is

        component contup
        port(
                clock : in std_logic;
                reset : in std_logic;
                Load : in std_logic;
                Enable : in std_logic;
                DATABUS : in std_logic_vector(5 downto 0);
                Upcount2 : out std_logic_vector(5 downto 0) );
   end component;

        signal clock : std_logic;
        signal reset : std_logic;
        signal Load : std_logic;
        signal Enable : std_logic;
        signal DATABUS : std_logic_vector(5 downto 0);
        signal Upcount2 : std_logic_vector(5 downto 0);

begin

    UUT : contup
                port map
                        (clock => clock,
                        reset => reset,
                        Load => Load,
                        Enable => Enable,
                        DATABUS => DATABUS,
                        Upcount2 => Upcount2 );

    reset <= '0', '1' after 2ns, '0' after 8ns;

    enable <= '1', '0' after 27ns, '1' after 48ns;

    databus <= "011111", "010010" after 35 ns;

    load<= '0', '1' after 29ns, '0' after 34 ns,'1' after 76ns, '0' after 82ns;

    process
         begin
            clock <= '1', '0' after 5ns;
            wait for 10ns;
    end process;

end TB_ARCHITECTURE;
```