

PUC-Minas - Ciência da Computação
ARQ1 – Guia 02
Período: 18-22/02/2013

Tema: Introdução à linguagem Verilog
Atividade: Descrição de módulos

01.) Editar e salvar um esboço de programa em Verilog,
o nome do arquivo deverá ser Exemplo0011.v,
tal como o nome do módulo abaixo,
concordando maiúsculas e minúsculas, sem espaços em branco:

```
// -----  
// Exemplo0011 - BASE  
// Nome: xxx yyy zzz  
// Matricula: 999999  
// -----  
  
// -----  
// test number system  
// -----  
  
module test_base_01;  
// ----- definir dados  
    reg [2:0] a;  
    reg [3:0] b;  
    reg [4:0] c;  
    reg [4:0] d;  
  
// ----- parte principal  
initial begin  
    $display("Exemplo0011 - xxx yyy zzz - 999999");  
    $display("Test number system");  
  
    a = 5;  
    b = 10;  
    c = 15;  
    d = 20;  
  
    $display("\nPositive value");  
    $display("a = %d = %3b", a, a);  
    $display("b = %d = %4b", a, a);  
    $display("c = %d = %5b", a, a);  
  
    $display("b = %d = %4b", b, b);  
    $display("c = %d = %5b", c, c);  
    $display("d = %d = %5b", d, d);  
    $display("d = %d = %5o", d, d);  
    $display("d = %d = %5h", d, d);  
end  
  
endmodule // test_base
```

- 02.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 03.) Executar o programa.
Observar as saídas.
- 04.) Copiar a versão atual do programa para outra (nova) – Exemplo0012.v.
- 05.) Acrescentar ao programa o trecho indicado abaixo.
Incluir na documentação complementar as alterações feitas,
acrescentar indicações de mudança de versão e
prever novos testes.

```
// ----- parte principal
initial begin
    $display("Exemplo0011 - xxx yyy zzz - 999999");
    $display("Test number system");

    a = 5;
    b = 10;
    c = 15;
    d = 20;

    $display("\nPositive value");
    $display("a = %d = %3b", a, a);
    $display("b = %d = %4b", a, a);
    $display("c = %d = %5b", a, a);

    $display("b = %d = %4b", b, b);
    $display("c = %d = %5b", c, c);
    $display("d = %d = %5b", d, d);
    $display("d = %d = %5o", d, d);
    $display("d = %d = %5h", d, d);

    a = -5;
    b = -5;
    c = -5;

    $display("\nNegative value");
    $display("a = -5 [3] = %d = %3b", a, a);
    $display("b = -5 [4] = %d = %4b", b, b);
    $display("c = -5 [5] = %d = %5b", c, c);
end

endmodule // test_base
```

- 06.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 07.) Executar o programa.
Observar as saídas.

08.) Copiar a versão atual do programa para outra (nova) – Exemplo0013.v.

09.) Acrescentar ao programa o trecho indicado abaixo.

Incluir na documentação complementar as alterações feitas,
acrescentar indicações de mudança de versão e
prever novos testes.

```
a = ~5+1;  
b = ~5+1;  
c = ~5+1;  
  
$display("\nNegative mixed expression");  
$display("a = %d = %3b", a, a);  
$display("b = %d = %4b", b, b);  
$display("c = %d = %5b", c, c);
```

10.) Compilar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

11.) Executar o programa.

Observar as saídas.

12.) Copiar a versão atual do programa para outra (nova) – Exemplo0014.v.

13.) Acrescentar ao programa o trecho indicado abaixo.

Incluir na documentação complementar as alterações feitas,
acrescentar indicações de mudança de versão e
prever novos testes.

```
a = 5 + 3;  
b = 5 + 3;  
c = 10 - 5 + 25 + 3 + 1;  
  
$display("\nOverflow");  
$display("a = %d = %3b", a, a);  
$display("b = %d = %4b", b, b);  
$display("c = %d = %5b", c, c);
```

14.) Compilar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

15.) Executar o programa.

Observar as saídas.

16.) Copiar a versão atual do programa para outra (nova) – Exemplo00015.v.

17.) Acrescentar ao programa o trecho indicado abaixo.

Incluir na documentação complementar as alterações feitas, acrescentar indicações de mudança de versão e prever novos testes.

```
$display("\nComplements");  
$display("0= %d = %3b = %3b", ~1 , (1-1), ~1 );  
$display("1= %d = %3b = %3b", ~0 , (2-1), ~0 );  
$display("2= %d = %3b = %3b", (1+1), (3-1), ~0+~0);
```

18.) Compilar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.

Se não houver erros, seguir para o próximo passo.

19.) Executar o programa.

Observar as saídas.

Exercícios:

DICAS GERAIS: Consultar a referência para Verilog na apostila para outros exemplos.

Para cada um dos enunciados abaixo,
definir o módulo correspondente e os procedimentos de testes.

01.) Executar as operações abaixo,
armazenar seus dados e resultados em registradores
e mostrar os valores resultantes em binário,
com a menor quantidade de bits necessária:

- a.) $3 + 14$
- b.) $3 * 8$
- c.) $33 / 5$
- d.) $25 - 11$
- e.) $2 * 8 + 7 - 1$

02.) Executar as operações abaixo,
armazenar seus dados e resultados em registradores
e mostrar os valores resultantes em binário,
com a menor quantidade de bits necessária:

- a.) $100110_{(2)} + 11011_{(2)}$
- b.) $11101_{(2)} + 25_{(8)}$
- c.) $1234_{(8)} / 4C_{(16)}$
- d.) $1BA_{(16)} - 101110011_{(2)}$
- e.) $25 * 32_{(8)} + 8A_{(16)}$

03.) Calcular e mostrar o complemento de 2
de cada um dos valores abaixo armazenados em registrador(es):

- a.) $101010_{(2)}$
- b.) $25_{(8)}$
- c.) $27_{(10)}$
- d.) $2C_{(16)}$
- e.) $25 - 35$

04.) Calcular e mostrar o complemento de 2
de cada um dos valores abaixo armazenados em registrador(es):

- a.) $101111_{(2)}$ com 8 bits de comprimento
- b.) $321_{(4)}$ com 7 bits de comprimento
- c.) $25_{(10)}$ com 6 bits de comprimento
- d.) $D_{(16)}$ com 5 bits de comprimento
- e.) $24_{(8)}$ com 8 bits de comprimento

05.) Executar as operações abaixo,
armazenar seus dados e resultados em registradores
e mostrar os valores resultantes em binário,
usar 8 bits e complemento de 2 nas subtrações:

- a.) $101011_{(2)} - 1001_{(2)}$
- b.) $10011_{(2)} - 15_{(8)}$
- c.) $35_{(8)} - C_{(16)}$
- d.) $BA_{(16)} - 10111001_{(2)}$
- e.) $37 - 1B_{(16)}$

Extras

- 06.) Definir e testar um módulo para calcular o complemento de 1,
de um valor qualquer contido em um byte.
É recomendável simular o mesmo em Logisim.
- 07.) Definir e testar um módulo para calcular o complemento de 2,
de um valor qualquer contido em um byte.
É recomendável simular o mesmo em Logisim.