UNIVERSIDADE FEDERAL DO PIAUÍ

Curso de Graduação Ciência da Computação

Relatório 1 – Teste do MIC feito em sala

Adaías Abner Brito Silva

Emanuel Mendes Monteiro

Felipe Santiago Gama

Marcos Vinícius Ribeiro Alencar

Teresina-PI

2021

Adaias Abner Brito Silva

Emanuel Mendes Monteiro

Felipe Santiago Gama

Marcos Vinícius Ribeiro Alencar

Relatório 1 – Teste do MIC feito em sala

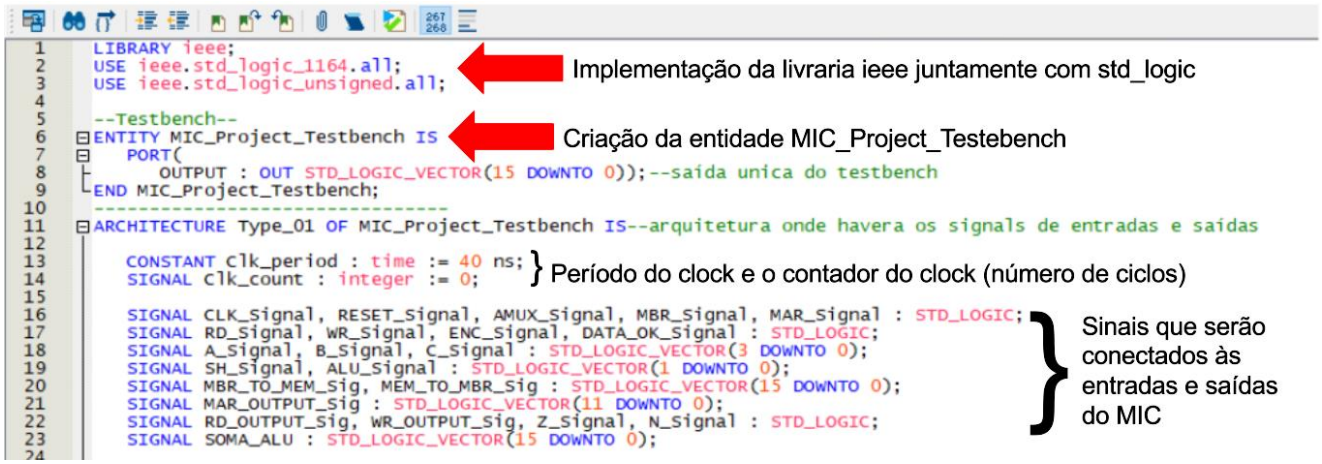
Relatório 1 – Teste do MIC proposto em sala, apresentado ao Curso de Ciência da Computação como parte do requisito para a aprovação na disciplina Arquitetura de Computadores

Professor: Ivan Saraiva Silva

Teresina-PI

2021

Criou-se inicialmente um novo file do tipo vhdl onde irá se criar o testbench do microprocessador (MIC_Project) desenvolvido em sala de aula. No código do testbench, após a implementação da biblioteca padrão ieee e da criação da entidade MIC_Project_Testsbench junto com a saída OUTPUT, única saída visível do teste, criou-se a arquitetura Type_01 que possui os sinais (Signal) de todas as entradas e saídas do microprocessador e a constante dos ciclos do MIC juntamente com o sinal que contará os ciclos.




```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.std_logic_unsigned.all;
4
5  --Testbench--
6  ENTITY MIC_Project_Testbench IS
7      PORT(
8          OUTPUT : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));--saida unica do testbench
9  END MIC_Project_Testbench;
10 ---------------------------------
11 ARCHITECTURE Type_01 OF MIC_Project_Testbench IS--arquitetura onde havera os signals de entradas e saidas
12
13     CONSTANT Clk_period : time := 40 ns;
14     SIGNAL Clk_count : integer := 0;
15
16     SIGNAL CLK_Signal, RESET_Signal, AMUX_Signal, MBR_Signal, MAR_Signal : STD_LOGIC;
17     SIGNAL RD_Signal, WR_Signal, ENC_Signal, DATA_OK_Signal : STD_LOGIC;
18     SIGNAL A_Signal, B_Signal, C_Signal : STD_LOGIC_VECTOR(3 DOWNTO 0);
19     SIGNAL SH_Signal, ALU_Signal : STD_LOGIC_VECTOR(1 DOWNTO 0);
20     SIGNAL MBR_TO_MEM_Sig, MEM_TO_MBR_Sig : STD_LOGIC_VECTOR(15 DOWNTO 0);
21     SIGNAL MAR_OUTPUT_Sig : STD_LOGIC_VECTOR(11 DOWNTO 0);
22     SIGNAL RD_OUTPUT_Sig, WR_OUTPUT_Sig, Z_Signal, N_Signal : STD_LOGIC;
23     SIGNAL SOMA_ALU : STD_LOGIC_VECTOR(15 DOWNTO 0);
```

Escreveu-se posteriormente o componente (Component) MIC_Project. Esse componente irá importar as entradas e saídas dessa arquitetura para serem utilizadas no teste.


```
22     SIGNAL RD_OUTPUT_Sig, WR_OUTPUT_Sig, Z_Signal, N_Signal : STD_LOGIC;
23     SIGNAL SOMA_ALU : STD_LOGIC_VECTOR(15 DOWNTO 0);
24
25 COMPONENT MIC_Project IS --Entradas e saidas do MIC_Project.vhd
26     PORT (
27         RESET : IN STD_LOGIC;
28         CLK : IN STD_LOGIC;
29         AMUX   : IN STD_LOGIC;
30         ALU    : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
31         MBR    : IN STD_LOGIC;
32         MAR    : IN STD_LOGIC;
33         RD     : IN STD_LOGIC;
34         WR     : IN STD_LOGIC;
35         ENC    : IN STD_LOGIC;
36         C      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
37         B      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
38         A      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
39         SH     : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
40        MEM_TO_MBR : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
41        DATA_OK : IN STD_LOGIC;
42
43        MBR_TO_MEM : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
44        MAR_OUTPUT : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
45        RD_OUTPUT : OUT STD_LOGIC;
46        WR_OUTPUT : OUT STD_LOGIC;
47        Z         : OUT STD_LOGIC;
48        N         : OUT STD_LOGIC);
49 END COMPONENT;
50 ---------------------------------
51 BEGIN
```

A posteriori, escreveu-se o port map para ligar os sinais da arquitetura com os do componente, para que assim possamos manipulá-los.

```
51  BEGIN
52  MIC : MIC_Project
53     PORT MAP (--ligando signals da arquitetura com as entradas do component
54        CLK => CLK_Signal, RESET => RESET_Signal, AMUX => AMUX_Signal,
55        MBR => MBR_Signal, MAR => MAR_Signal, RD => RD_Signal, WR => WR_Signal,
56        ENC => ENC_Signal, DATA_OK => DATA_OK_Signal, A => A_Signal, B => B_Signal,
57        C => C_Signal, SH => SH_Signal, MBR_TO_MEM => OUTPUT,
58        MEM_TO_MBR => MEM_TO_MBR_Sig, MAR_OUTPUT => MAR_OUTPUT_Sig,
59        RD_OUTPUT => RD_OUTPUT_Sig, WR_OUTPUT => WR_OUTPUT_Sig, Z => Z_Signal,
60        N => N_Signal, ALU => ALU_Signal
61        );
62  ---------------------------------
```

Escreveu-se depois os processos clock e reset. No process do clock, o sinal começa com o valor “0” e após a metade do tempo do período (20 nanosegundos) o valor do clock irá para “1” e o contador de ciclos (Clk_count) irá aumentar em 1 também. Então o process irá esperar mais 20 ns (metade do período). Quando o contador chegar a 20 ciclos, o teste será encerrado.

```
63  ------------------------CLOCK
64  Clock_Process : PROCESS
65     Begin
66       CLK_Signal <= '0';
67       wait for Clk_period/2;  --for 0.5 ns signal is '0'.
68       CLK_Signal  <= '1';
69       Clk_count <= Clk_count + 1;
70       wait for Clk_period/2;  --for next 0.5 ns signal is '1'.
71
72  IF (Clk_count = 20) THEN
73  REPORT "Stopping simulation after 20 cycles";
74          Wait;
75  END IF;
76
77  End Process Clock_Process;
78  ------------------------CLOCK
```

Valor inicial em 0 do sinal do Clock

Espera metade do período (clk_period)

Valor do sinal do Clock vai para 1

Contador aumenta em 1.

Espera metade do período (clk_period)

Após 20 ciclos, o teste será encerrado

O reset apenas limpará as entradas, alterando o valor do sinal do reset de “0” para “1” e depois para “0” de novo, tudo dentro de um ciclo (40 ns), no início do programa, e depois ficará aguardando o teste encerrar.

```
80  ------------------------RESET
81  Reset_Process : PROCESS
82     Begin
83        RESET_Signal <= '0';
84        Wait for 10 ns;
85        RESET_Signal <= '1';
86        Wait for 30 ns;
87        RESET_Signal <= '0';
88        wait;
89
90  End Process Reset_Process;
91  ------------------------RESET
92  ----------------------------
```

Sinal reset começa em 0

Muda para 1

Volta para 0

Espera até o final do teste

Terminando a escrita do reset, fez-se o process do teste em si. Esse terá 8 ciclos, porém pulando o primeiro ciclo, que será destinado ao reset para retornar os sinais a um estado conhecido. Dessa forma, o process do teste iniciará no segundo ciclo do programa, por isso haverá uma espera de 40 nanosegundos (1 ciclo) no início, e todas as próximas instruções irão esperar o mesmo tempo antes de executarem. Após a espera de 1 ciclo, o processo irá começar a dar valores aos sinais de entrada (menos ao reset e ao clock por possuírem processos exclusivos para suas funções), para realizar os comandos do teste.

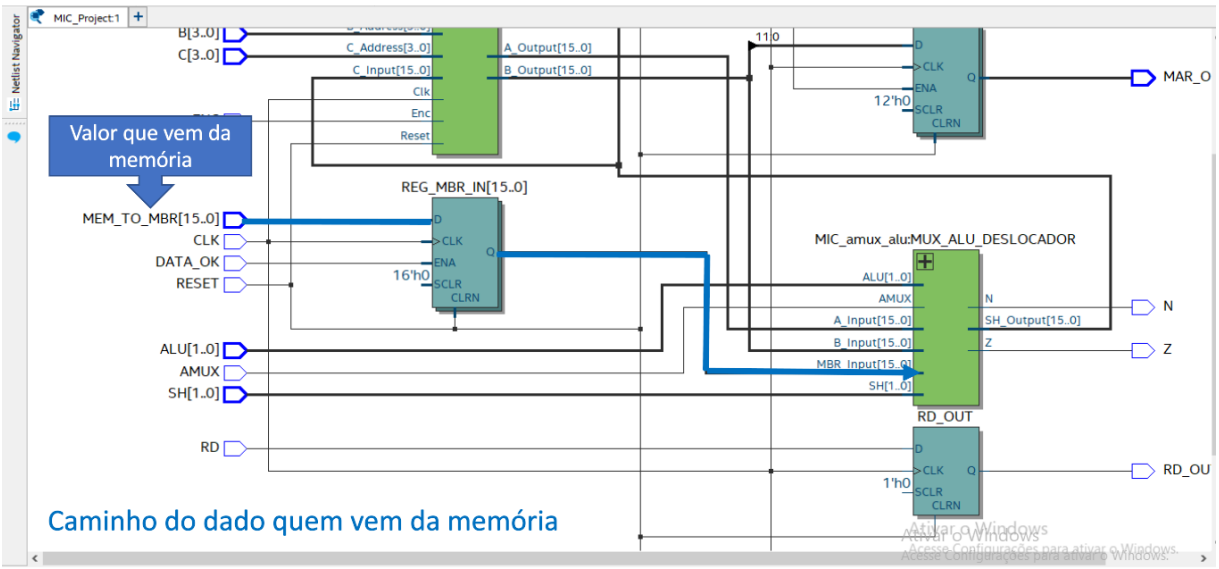
O segundo ciclo do programa será onde o MIC receberá um valor da memória e o guardará no MBR. O sinal AMUX_Signal recebe o valor 1 para que a entrada A_Input receba o valor do MBR_Input (ambas entradas da arquitetura MIC_amux_alu). O sinal MEM_TO_MBR_Sig recebe o valor da memória e coloca em MBR_Input, que é a entrada da ligação MEM_TO_MBR com o amux, e o sinal DATA_OK_Signal recebe "1" para que seja possível levar o conteúdo de MEM_TO_MBR para o amux. Todos os demais sinais recebem o valor padrão 0.

```
92  ------------------------------
93  PROCESS
94      BEGIN
95      -- SEGUNDO CICLO: Recebe o valor 1 da memoria e guarda no MBR.
96         wait for 40ns;
97          AMUX_Signal  <= '1';
98          ALU_Signal   <= "00";
99          MBR_Signal   <= '0';
100         MAR_Signal   <= '0';
101         RD_Signal    <= '0';
102         WR_Signal    <= '0';
103         ENC_Signal   <= '0';
104         C_Signal     <= "0000";
105         B_Signal    <= "0000";
106         A_Signal    <= "0000";
107         SH_Signal     <= "00";
108         MEM_TO_MBR_Sig <= "0000000000000001";
109         DATA_OK_Signal <= '1';
110
111
```

Sinal do AMUX está em 1 para usar-se o MBR e não a entrada A

1 em binário que vem da memória

Valor em 1 para indicar que o valor da memória estar pronto para uso



No terceiro ciclo, pega-se o valor de MBR (1 em binário) e o coloca no registrador A (registrador de número 10). Para isso o AMUX_Signal continuará em 1 pois A_Input continuará recebendo o valor do MBR_Input. O ALU_Signal receberá o valor "10" para

haver transparência de A_Input na ULA (esse que está com o valor do MBR), ou seja, a saída da ULA será o valor de A_Input. O ENC_Signal sobe em 1 para permitir a escrita no banco de registradores. E por fim, o sinal C_Signal recebe o valor “1010” (10 decimal em binário) indicando em que registrador deverá ser escrito o MBR, no caso, o registrador de número 10.

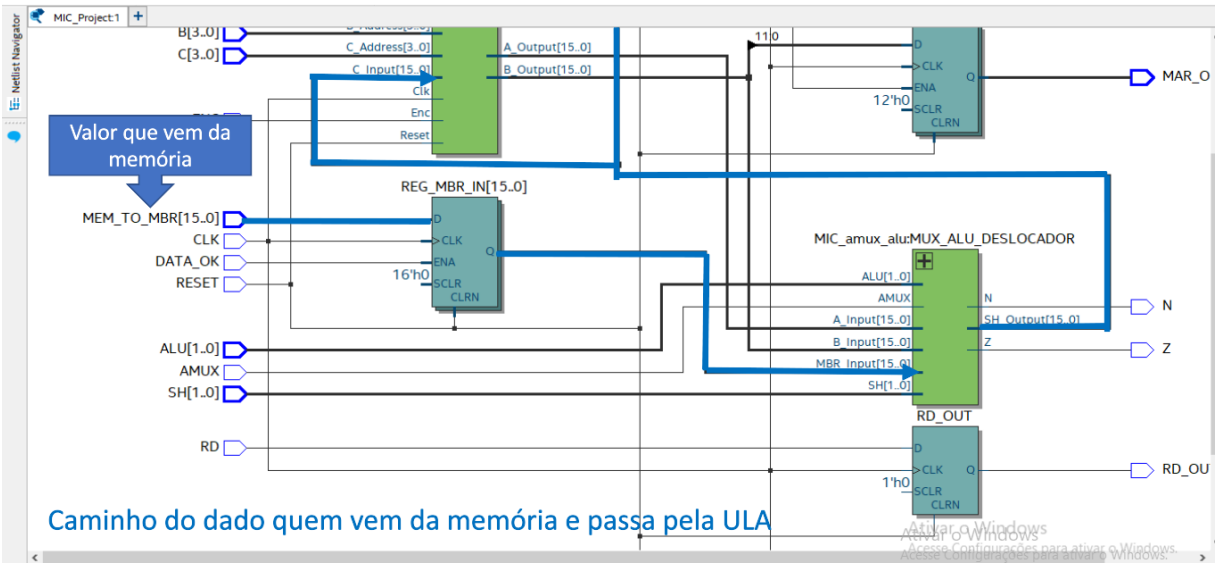
```
112
113     --TERCEIRO CICLO:Escreve o conteudo de MBR no reg A (numero 10).
114     wait for 40ns;
115      --CLK_Signal <= '1';
116      --RESET_Signal <= '0';
117      AMUX_Signal  <= '1';
118      ALU_Signal   <= "10";
119      MBR_Signal   <= '0';
120      MAR_Signal   <= '0';
121      RD_Signal    <= '0';
122      WR_Signal    <= '0';
123      ENC_Signal   <= '1';
124      C_Signal     <= "1010";
125      B_Signal    <= "0000";
126      A_Signal    <= "0000";
127      SH_Signal     <= "00";
128      MEM_TO_MBR_Sig <= "0000000000000000";
129      DATA_OK_Signal <= '0';
```

Sinal do AMUX continua em 1, A_Input ainda recebe o valor do MBR

Transparência de A_Input

Habilitando a escrita no banco de registradores

Endereço do registrador A (10)



O quarto ciclo é o mesmo do terceiro com a diferença de que o conteúdo de MBR será guardado no registrador B (endereço 11) ao invés do registrador A. Dessa forma, o valor do C_Signal será “1011” (11 decimal em binário).

```
131     --QUARTO CICLO:Escreve o conteudo de MBR no reg B (numero 11).
132     wait for 40ns;
133     --CLK_Signal <= '1';
134     --RESET_Signal <= '0';
135     AMUX_Signal  <= '1';
136     ALU_Signal   <= "10";
137     MBR_Signal   <= '0';
138     MAR_Signal   <= '0';
139     RD_Signal    <= '0';
140     WR_Signal    <= '0';
141     ENC_Signal   <= '1';
142     C_Signal     <= "1011";
143     B_Signal    <= "0000";
144     A_Signal    <= "0000";
145     SH_Signal     <= "00";
146     MEM_TO_MBR_Sig <= "0000000000000000";
147     DATA_OK_Signal <= '0';
```

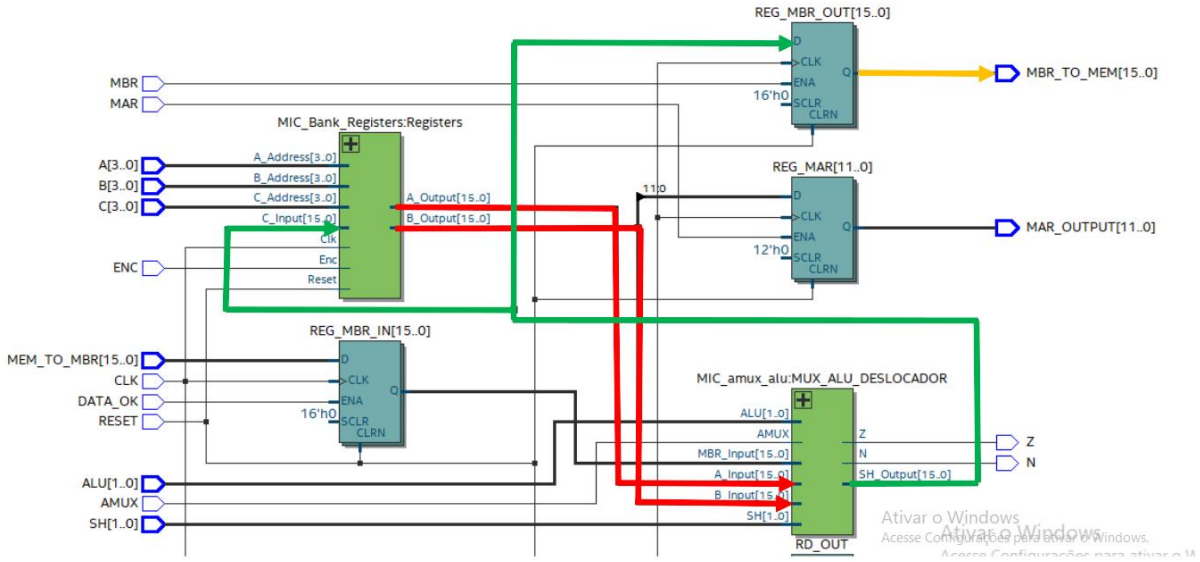
Endereço do registrador B (11)

No quinto ciclo fez-se a soma entre os valores do registrador A e B e o resultado é inserido no registrador AC (endereço 1). Assim, os sinais ficam: O AMUX_Signal recebe o valor ”0”, pois agora a ULA receberá o valor de A ao invés do valor do MBR. O ALU_Signal recebe o valor ”00” que indica soma entre o conteúdo das entradas A_Input com B_Input. O ENC_Signal recebe o valor 1 para que se escreva a soma da ALU no registrador AC. O MBR_Signal recebe 1 para que a soma da ULA vá para MBR_TO_MEM que é ligado a saída do teste OUTPUT. O C_Signal recebe ”0001”, que é o endereço de escrita (AC). A_Signal e B_Signal recebem ”1010” (10 decimal em binário) e ”1011” (11 decimal em binário) respectivamente. Esses valores indicam de qual registrador A_Output e B_Output irão levar os dados. Nesse caso, A_Output levará o valor do registrador 10 que possui o valor ”1” armazenado e B_Output levará o do 11, que também possui armazenado o mesmo valor. Finalmente, o SH_Signal recebe ”00” para não haver deslocamentos na ULA.

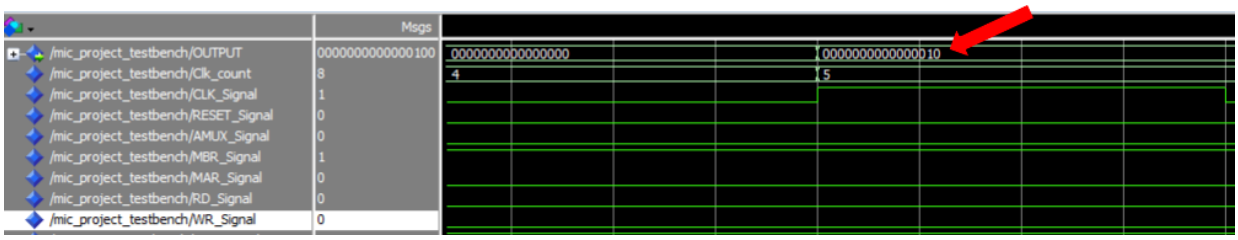
```
148
149     --QUINTO CICLO:Faz soma 1+1(A+B) e coloca no reg AC (1) e em MBR.
150     wait for 40ns;
151     --CLK_Signal <= '1';
152     --RESET_Signal <= '0';
153     AMUX_Signal  <= '0';
154     ALU_Signal   <= "00";
155     MBR_Signal   <= '1';
156     MAR_Signal   <= '0';
157     RD_Signal    <= '0';
158     WR_Signal    <= '0';
159     ENC_Signal   <= '1';
160     C_Signal     <= "0001";
161     B_Signal    <= "1011";
162     A_Signal    <= "1010";
163     SH_Signal     <= "00";
164     MEM_TO_MBR_Sig <= "0000000000000000";
165     DATA_OK_Signal <= '0';
166     --OUTPUT <= MBR_TO_MEM_Sig(15 DOWNTO 0);
```

"00" é a instrução de soma da ULA
Disponibilizando a saída do programa para vizualização

Endereço do registrador AC (1)
Endereço do registrador B (11)
Endereço do registrador A (10)



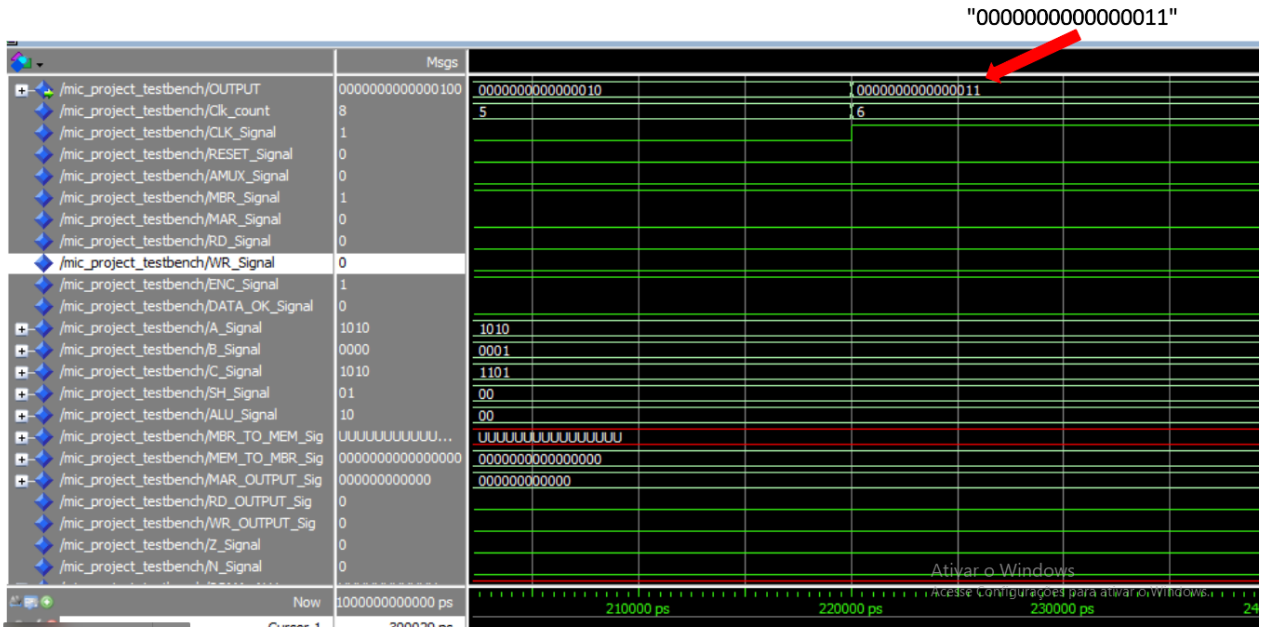
As linhas vermelhas mostram os valores a serem somados saindo do banco de registradores e indo para a ULA. Já a linha verde representa o caminho que o resultado dessa soma vai percorrer, voltando ao banco de registradores para ser guardado em AC e também indo para REG_MBR_OUT de onde irá para a saída do teste, representado pela seta amarela.



Podemos observar o valor “0000000000000010” na saída OUTPUT no ModelSim, que representa 2 decimal na linguagem binária. Esse valor está correto pois OUTPUT é a saída do teste, que está ligado ao REG_MBR_OUT, onde esse possui o resultado da soma da ULA. Dessa forma, como os registradores A e B possuíam o valor 1, a soma de ambos resulta em 2 na linguagem binária.

No sexto ciclo, tomou-se o conteúdo do registrador AC, que possui o valor da soma do quinto ciclo, portanto 2, e pegou-se o conteúdo do registrador A (valor 1). Depois disso, a soma foi feita e o resultado foi colocado no registrador D (Endereço 13) e na saída do teste. Essa operação é semelhante à do quinto ciclo, então grande parte dos sinais permaneceram inalterados e a lógica é exatamente a mesma. Os valores que irão mudar são de C_Signal, B_Signal e A_Signal. Esses receberam respectivamente: “1101”, endereço para a escrita no registrador D; “0001”, endereço do registrador AC que será recebido pela ULA; “1010”, endereço do registrador A que será recebido pela ULA.

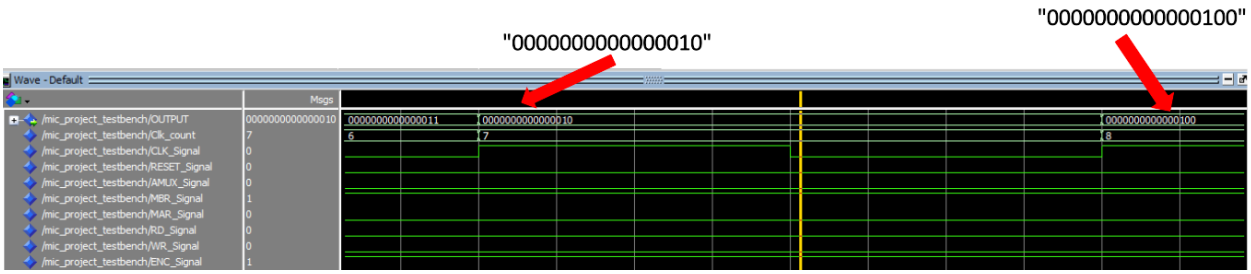
```
168       --SEXTO CICLO:Faz soma 2+1(AC+A) e coloca no reg D (13) e em MBR.
169       wait for 40ns;
170       --CLK_Signal <= '1';
171       --RESET_Signal <= '0';
172       AMUX_Signal  <= '0';
173       ALU_Signal   <= "00";
174       MBR_Signal   <= '1';
175       MAR_Signal   <= '0';
176       RD_Signal    <= '0';
177       WR_Signal    <= '0';
178       ENC_Signal   <= '1';
179       C_Signal     <= "1101";   <- Registrador D(13)
180       B_Signal    <= "0001";    <- Registrador AC(1)
181       A_Signal    <= "1010";    <- Registrador A(10)
182       SH_Signal     <= "00";
183       MEM_TO_MBR_Sig <= "0000000000000000";
184       DATA_OK_Signal <= '0';
```



Observa-se que o valor da saída OUTPUT continua correto, o valor “0000000000000011” (3 decimal em binário) é a soma dos valores 2 (vindo do registrador AC) mais 1 (vindo do registrador A).

O sétimo e oitavo ciclo são iguais, pois em ambos o MIC deverá pegar o conteúdo do registrador A, leva-lo a ULA, desloca-lo à esquerda e guardá-lo novamente em A. O deslocamento à esquerda em binário equivale a uma multiplicação de um número por 2. Dessa maneira, toma-se o número de A (1) e o multiplica por 2 duas vezes, resultando em uma multiplicação por 4. Para isso: AMUX_Signal continua em 0, pois precisamos de uma transparência em A; ALU_Signal fica em “00” que indica transparência de A_Input na ULA; MBR_Signal permanece em 1 para que possamos visualizar o resultado do deslocamento na saída; ENC_Signal também permanece em 1 para haver escrita no banco de registradores; C_Signal recebe “1010” (10) para informar em que endereço será escrito o deslocamento; A_Signal recebe “1010” para indicar que o valor que irá para a ULA será do registrador A (10); SH_Signal recebe “01” que indica descolamento de bits à esquerda.

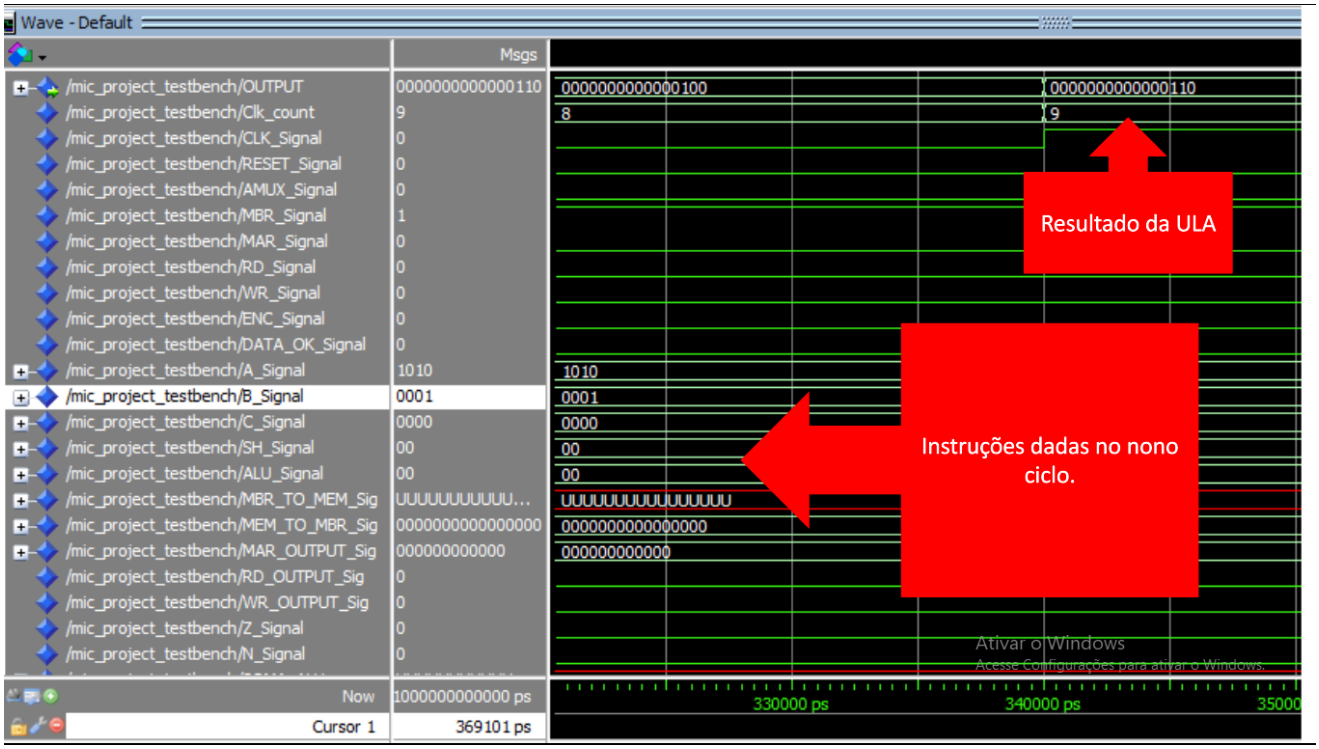
```
186     --SETIMO CICLO: Multiplica A por 2 e guarda em A(2).
187     wait for 40ns;
188     AMUX_Signal  <= '0';
189     ALU_Signal   <= "10";   <- Transparência da ULA
190     MBR_Signal   <= '1';
191     MAR_Signal   <= '0';
192     RD_Signal    <= '0';
193     WR_Signal    <= '0';
194     ENC_Signal   <= '1';
195     C_Signal     <= "1010"; <- Escrever no registrador A
196     B_Signal    <= "0000";
197     A_Signal    <= "1010";  <- Enviar valor do registrador A para a ULA
198     SH_Signal     <= "01";  <- Deslocamento à esquerda
199     MEM_TO_MBR_Sig <= "0000000000000000";
200     DATA_OK_Signal <= '0';
201
```



Resultados do sétimo e oitavo ciclo respectivamente, tendo no final do oitavo ciclo o número 4 armazenado no registrador A.

O nono é o último ciclo com instruções no teste do MIC e também será uma soma. Essa soma será dos valores dos registradores A e AC (4+2 respectivamente). Dessa forma, a maioria dos sinais serão iguais às do quinto e sexto ciclo, mudando apenas os sinais C_Signal, B_Signal, A_Signal e ENC_Signal, já que o resultado não será armazenado em um registrador dessa vez.

```
218         --NONO CICLO: Soma A+AC(4+2).
219         wait for 40ns;
220         AMUX_Signal  <= '0';
221         ALU_Signal   <= "00";   <- Somar valores na ULA
222         MBR_Signal   <= '1';
223         MAR_Signal   <= '0';
224         RD_Signal    <= '0';
225         WR_Signal    <= '0';
226         ENC_Signal   <= '0';    <- Não haverá escrita nesse ciclo
227         C_Signal     <= "0000";
228         B_Signal    <= "0001";  <- Conteúdo de AC será levado para a ULA
229         A_Signal    <= "1010";  <- Conteúdo de A será levado para a ULA
230         SH_Signal     <= "00";
231         MEM_TO_MBR_Sig <= "0000000000000000";
232         DATA_OK_Signal <= '0';
233
234         wait;
235
236 END process;                    <- Fim do programa e da arquitetura
237
238 END Type_01;
```



Como pode se observar, o resultado foi o esperado, a saída do programa indica o valor ”0000000000000110”, que é 6 em binário.