Universidade do Vale do Itajaí
Projeto de Sistemas Digitais

Avaliação 6 – Projeto de processadores em VHDL

Aluno: Lucas Mateus Gonçalves
Professores : Cesar Albenes Zeferino e Douglas Rossi de Melo

Julho
2020

Universidade do Vale do Itajaí

Avaliação 6 – Projeto de processadores em VHDL

Relatório

Sexto Relatório apresentado ao Curso Projeto de Sistemas Sequenciais da Universidade do Vale do Itajaí, como requisito parcial para Avaliação 6.

Aluno: Lucas Mateus Gonçalves

Professores : Cesar Albenes Zeferino e Douglas Rossi de Melo

Julho
2020

Sumário

1 Resumo

Esse papel tem a intenção de mostrar e explicar um processador de estados finitos.

O projeto foi criado para funcionar com o `FPGA Cyclone V: 5CGXFC7C7F23C8`. O projeto foi compilado usando `Quartus 19.1 Lite` e `Modelsim 10.5b`.

Foram usados `333 registradores` , `8192 blocos de memória` , `273 ALM's` , `317 ALUT's` e possui uma frequência de `99.65 MHZ` .

Alguns componentes encontrados nesse projeto já foram mostrados e explicados em trabalhos anteriores, então sua simulação será omitida, porem o componente em si será explicado para que seus sinais sejam compreendidos.

A sequencia de componentes a ser explicado será do menos complexo ao mais complexo, começando com o bloco de operações *DataPath*.

O autor de todos os componentes, incluindo seus `testbenches` foi o aluno *Lucas Mateus Gonçalves*.

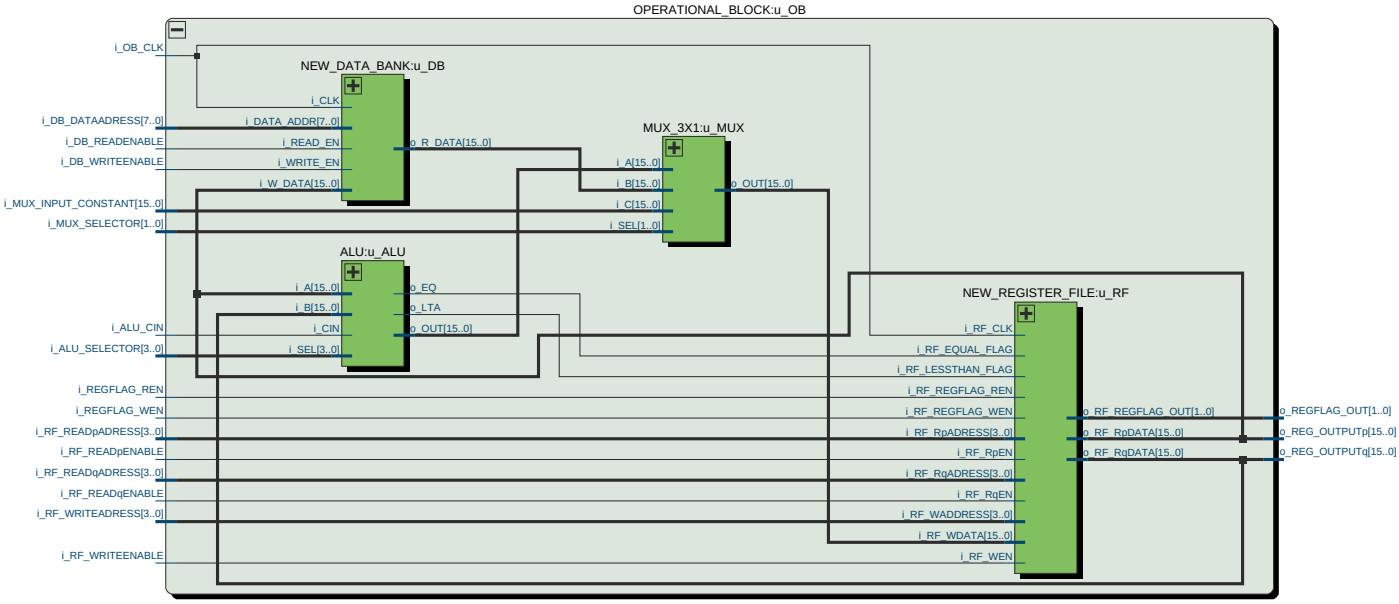
Os códigos do projeto podem ser encontrados no seguinte link:

`https://www.edaplayground.com/x/43FQ`

Pelas limitações da ferramenta, a simulação não mostra o resultado final no banco de dados, portanto o valor binário resultante da operação deve ser encontrado na saída do banco de registradores.

2 Bloco de Operações

O bloco de operações serve como o *DataPath*, faz cálculos baseado nas informações dadas e devolve sinais e valores.



`RTL do componente Operational_Block` - A descrição do componente multiplexador será omitida por sua simplicidade, sua função sera descrita em componentes que a usam.

2.1 ALU

A unidade de logica aritmética é usada para calcular e relatar informações importantes sobre dois valores, esses cálculos podem ser soma, subtração, alguma transformação lógica entre os dois valores, e relata qual é maior, ou se são iguais.

2.1.1 Explicação do código

```
port(

 i_A    : in std_logic_vector( 15 downto 0 )    ;
 i_B    : in std_logic_vector( 15 downto 0 )    ;
 i_CIN  : in integer range 0 to 1               ; ---Carry in for the shift and
     for the adder

 i_SEL  : in std_logic_vector( 3 downto 0 )     ;

 o_OUT  : out std_logic_vector( 15 downto 0 )   ;
 o_EQ   : out std_logic                         ; ---CMP two vectors , EQUAL FLAG
 o_LTA  : out std_logic                           ---CMP two vectors , If B less
     than A , o_LTA = 1

);
```

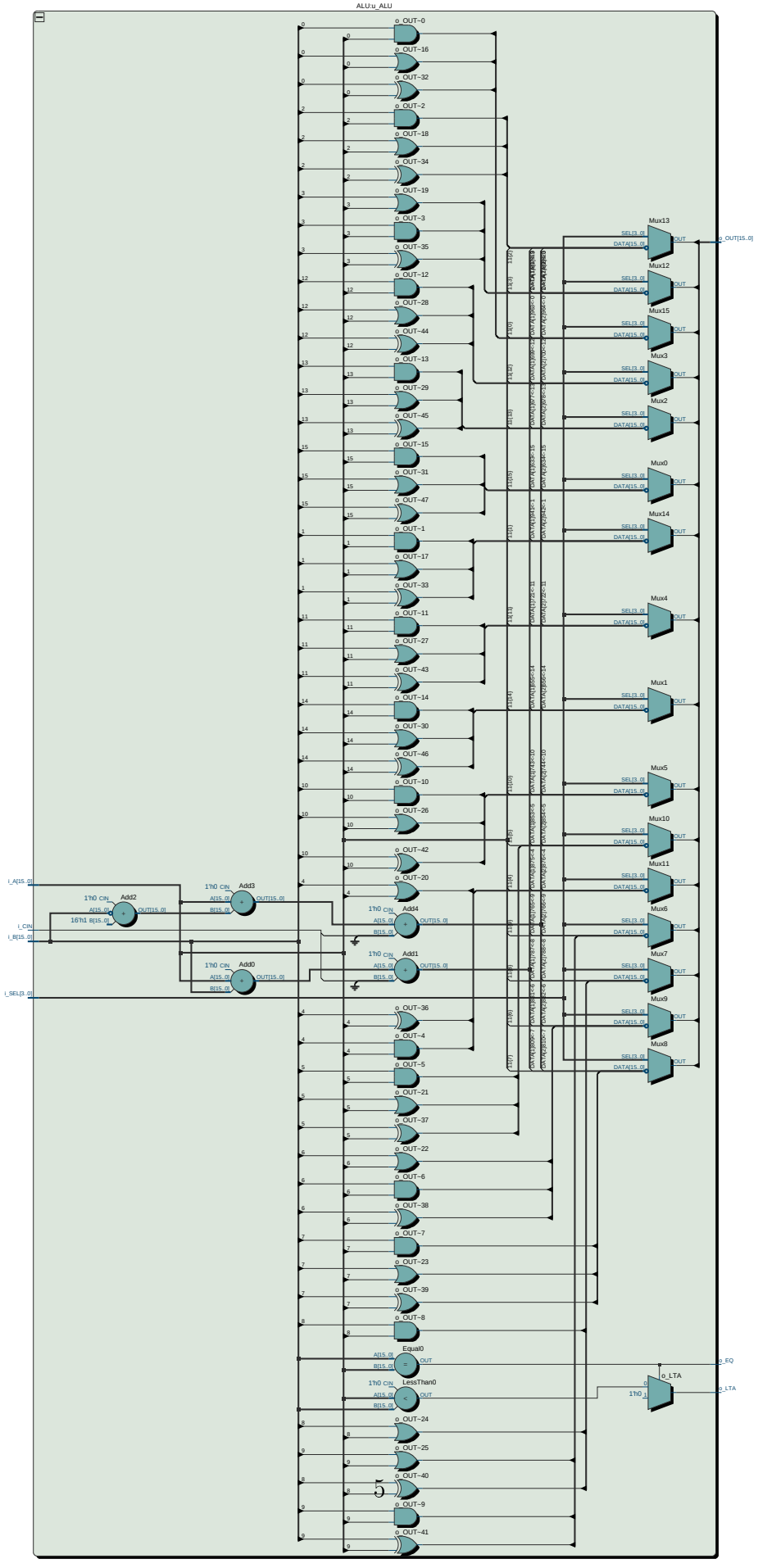
Os valores A e B são os valores vindos do registrador, apenas o registrador tem permissão de entregar valores ao componente, qualquer valor guardado em uma memória separada tem que ser guardado primeiro no registrador para que um calculo seja feito.

```
process (i_SEL , i_A , i_B ) begin

        case i_SEL is

                when "0000" => o_OUT <= i_A ;   --- NO OP

                when "0001" => o_OUT <= std_logic_vector( unsigned( i_A ) +
                    unsigned( i_B ) + i_CIN );  ---UNSIGNED ADD

                when "0010" => o_OUT <= std_logic_vector( unsigned( i_A ) +
                    unsigned( unsigned(not i_B) + 1 ) + i_CIN ); ---UNSIGNED SUB
                    WITH 2 COM

                when "0101" => o_OUT <= i_A and i_B;    --AND

                when "0110" => o_OUT <= i_A or i_B;     --OR

                when "0111" => o_OUT <= i_A xor i_B;    --XOR

                when "1000" => o_OUT <= not i_A;        --NOT A

                when others => o_OUT <= i_A ;   --- NO OP

        end case;       --I_SEL
```

O seletor decide que aritmética ou transformação lógica fazer sobre os valores contidos nas suas entradas, é responsabilidade do bloco de controle mandar esse sinal para que a função correta seja feita.

```
if( i_A = i_B ) then

        o_EQ    <= '1';
        o_LTA <= '0';

else

        o_EQ <= '0';

        if ( unsigned(i_A) < unsigned(i_B) ) then

                o_LTA <= '1';

        else

                o_LTA <= '0';

        end if; --A<B

end if; ---A=B

end process;
```

As três saídas são para a saída do valor final após sua transformação, a outra seria o sinal de que os dois valores de entrada são iguais, e o ultimo se o valor de A é menor que B. Essas servem para atualizar o registrador de `FLAGS` se o seu `ENABLE` permitir.


ALU:u_ALU
i_A[15..0]
i_B[15..0]
i_CIN
i_SEL[3..0]
Add0
Add1
Add2
Add3
Add4
Equal0
LessThan0
Mux0
Mux1
Mux2
Mux3
Mux4
Mux5
Mux6
Mux7
Mux8
Mux9
Mux10
Mux11
Mux12
Mux13
Mux14
Mux15
o_OUT[15..0]
o_EQ
o_LTA

2.1.2 Código do componente ALU

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

entity ALU is   ---DONE

port(

        i_A             : in std_logic_vector( 15 downto 0 )    ;
        i_B             : in std_logic_vector( 15 downto 0 )    ;
        i_CIN           : in integer range 0 to 1               ;
            ---Carry in for the shift and for the adder

        i_SEL           : in std_logic_vector( 3 downto 0 )     ;

        o_OUT           : out std_logic_vector( 15 downto 0 )   ;
        o_EQ            : out std_logic                         ;
            ---CMP two vectors , EQUAL FLAG
        o_LTA           : out std_logic
            ---CMP two vectors , If B less than A , o_LTA = 1



);

end ALU;


architecture ARCH_1 of ALU is

begin

process (i_SEL , i_A , i_B ) begin

        case i_SEL is

                when "0000" => o_OUT <= i_A ;   --- NO OP

                when "0001" => o_OUT <= std_logic_vector( unsigned( i_A ) +
                    unsigned( i_B ) + i_CIN );  ---UNSIGNED ADD

                when "0010" => o_OUT <= std_logic_vector( unsigned( i_A ) +
                    unsigned( unsigned(not i_B) + 1 ) + i_CIN ); ---UNSIGNED SUB
                    WITH 2 COM

                when "0101" => o_OUT <= i_A and i_B;    --AND

                when "0110" => o_OUT <= i_A or i_B;     --OR

                when "0111" => o_OUT <= i_A xor i_B;    --XOR

                when "1000" => o_OUT <= not i_A;        --NOT A

                when others => o_OUT <= i_A ;   --- NO OP

        end case;       --I_SEL
```

```
        if( i_A = i_B ) then

                o_EQ    <= '1';
                o_LTA <= '0';

        else

                o_EQ <= '0';

                if ( unsigned(i_A) < unsigned(i_B) ) then

                        o_LTA <= '1';

                else

                        o_LTA <= '0';

                end if; --A<B

        end if; ---A=B


end process;


end ARCH_1;
```

2.2 Banco de Dados

O banco de dados está integrado na CPU, portanto pode ser visto como um CACHE, ou uma RAM integrada. Esse componente guarda valores dados pelo usuário ou pelo Banco de Registradores.

Esse componente pode apenas ler ou escrever em um ciclo de CLOCK, não podendo fazer os dois ao mesmo tempo.

2.2.1 Explicação do código

```
port(

        i_CLK           : in std_logic;

        i_DATA_ADDR     : in std_logic_vector( 7 downto 0 );

        i_READ_EN       : in std_logic;
        i_WRITE_EN      : in std_logic;

        i_W_DATA        : in std_logic_vector( 15 downto 0 );
        o_R_DATA        : out std_logic_vector( 15 downto 0 )

);
```

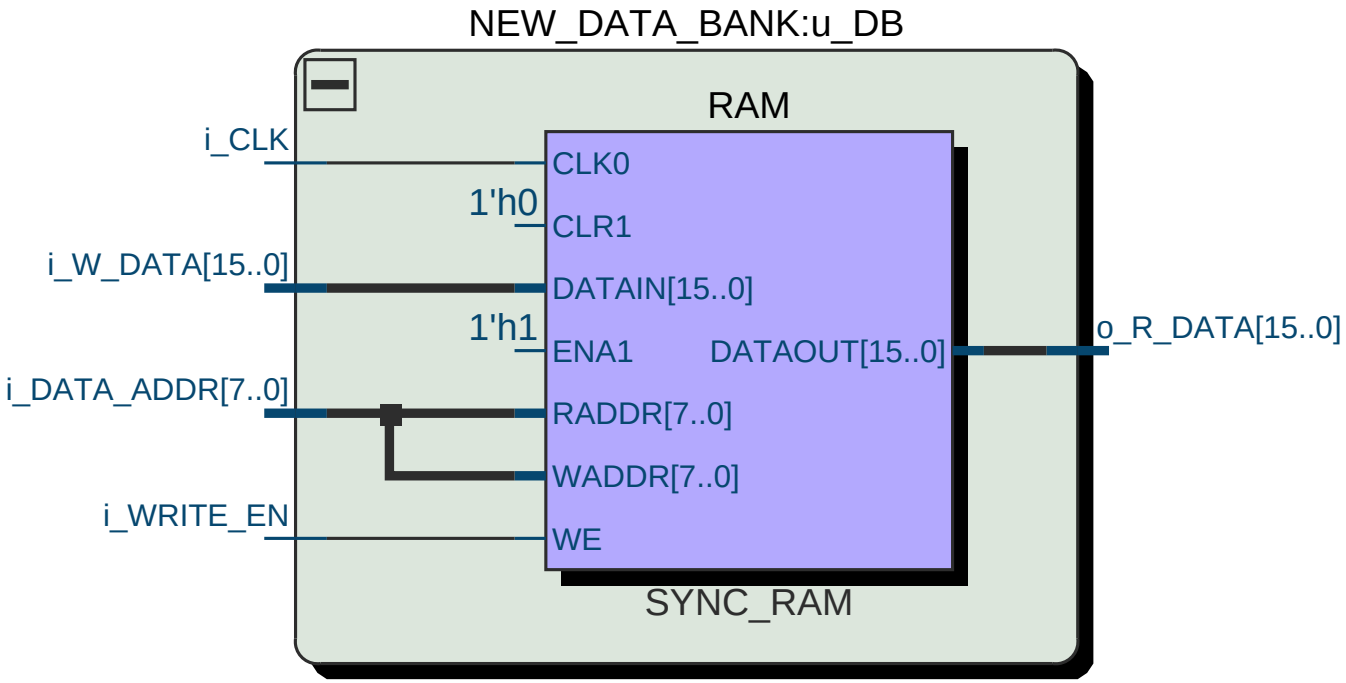
Esse componente é sincronizado com o relógio para a leitura e escrita de dados, também tem uma entrada de endereços ligada ao Bloco de Controle.

```
        type t_RAM is array ( 0 to 255 ) of std_logic_vector( 15 downto 0 );
        signal RAM      :       t_RAM;

begin
```

Foi utilizado o tipo RAM para guardar os dados.

```
if( rising_edge(i_CLK) ) then

        if i_WRITE_EN = '1' then

                RAM( to_integer(unsigned(i_DATA_ADDR)) ) <= i_W_DATA;

        end if; ---WRITE ENABLE

end if; ---CLK

if i_READ_EN = '1' then

        o_R_DATA        <= RAM( to_integer(unsigned(i_DATA_ADDR)) );

else o_R_DATA <= "UUUUUUUUUUUUUUUU";

end if; ---READ ENABLE
```

A escrita e a leitura são feitas com a permissão do seu sinal equivalente, cujo registrador a ser operado é determinado pelo endereço providenciado pelo Bloco de Controle.



2.2.2 Código do componente Banco de Dados

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

entity NEW_DATA_BANK is ---DONE

port(

        i_CLK                   : in std_logic;

        i_DATA_ADDR     : in std_logic_vector( 7 downto 0 );

        i_READ_EN       : in std_logic;
        i_WRITE_EN      : in std_logic;

        i_W_DATA                : in std_logic_vector( 15 downto 0 );
        o_R_DATA                : out std_logic_vector( 15 downto 0 )

);

end NEW_DATA_BANK;

architecture ARCH_1 of NEW_DATA_BANK is

        type t_RAM is array ( 0 to 255 ) of std_logic_vector( 15 downto 0 );
        signal RAM      :       t_RAM;

begin

process (i_DATA_ADDR , i_CLK , i_WRITE_EN , i_READ_EN , i_W_DATA ) begin

if( rising_edge(i_CLK) ) then

        if i_WRITE_EN = '1' then

                RAM( to_integer(unsigned(i_DATA_ADDR)) ) <= i_W_DATA;

        end if; ---WRITE ENABLE

end if; ---CLK

if i_READ_EN = '1' then

        o_R_DATA        <= RAM( to_integer(unsigned(i_DATA_ADDR)) );

else o_R_DATA <= "UUUUUUUUUUUUUUUU";

end if; ---READ ENABLE

end process;

end ARCH_1;
```

2.3 Banco de Registradores

O Banco de Registradores é o componente mais complexo do Bloco de Operações, controlando o movimento de dados internos e sinais para o Bloco de Controle. O Banco de Registradores consiste em dezesseis registradores de dezesseis bits, e um registrador de `FLAGS`, que fala diretamente com o Bloco de Controle.

```
port(

        i_RF_CLK                : in std_logic                          ;

        i_RF_REGFLAG_WEN        : in std_logic                          ;
            ---REGISTER FLAG WRITE ENABLE
        i_RF_REGFLAG_REN        : in std_logic                          ;
            ---REGISTER FLAG READ ENABLE
        i_RF_EQUAL_FLAG         : in std_logic                          ;
        i_RF_LESSTHAN_FLAG      : in std_logic                          ;
        o_RF_REGFLAG_OUT        : out std_logic_vector( 1 downto 0 )    ;
            ---REGISTER FLAG OUT

        i_RF_WDATA              : in std_logic_vector (15 downto 0)     ;
            ---WRITE DATA IN
        i_RF_WADDRESS           : in std_logic_vector (3 downto 0)      ;
            ---WRITE ADDRESS
        i_RF_WEN                : in std_logic                          ;
            ---WRITE ANABLE

        o_RF_RpDATA             : out std_logic_vector(15 downto 0)     ;
            ---READ DATA P
        i_RF_RpADRESS           : in std_logic_vector(3 downto 0)       ;
            ---READ ADDRESS
        i_RF_RpEN               : in std_logic                          ;

        o_RF_RqDATA             : out std_logic_vector(15 downto 0)     ;
            ---READ DATA Q
        i_RF_RqADRESS           : in std_logic_vector(3 downto 0)       ;
            ---READ ADDRESS
        i_RF_RqEN               : in std_logic

);
```

O Banco de Registradores aceita como entrada valores para serem escritos, esses valores vem do multiplexador que decide se o valor vem do `ALU`, do `Banco de Dados` ou do Bloco de Controle como um `Imediato`.

```
if rising_edge( i_RF_CLK ) then

        ---WRITE INTO

        if i_RF_WEN = '1' then

                REG( to_integer( unsigned( i_RF_WADDRESS ) ) ) <= i_RF_WDATA;
                    ---WRITE DATA IN

        end if; --i_RF_WEN

        if i_RF_REGFLAG_WEN = '1' then
```

```
                r_FLAGS(1)      <= i_RF_EQUAL_FLAG;
                r_FLAGS(0)      <= i_RF_LESSTHAN_FLAG;

        end if; --i_RF_REGFLAG_WEN

        if i_RF_REGFLAG_REN = '1' then

                o_RF_REGFLAG_OUT        <= r_FLAGS;

        end if;

        ---WRITE

end if; ---CLK
```

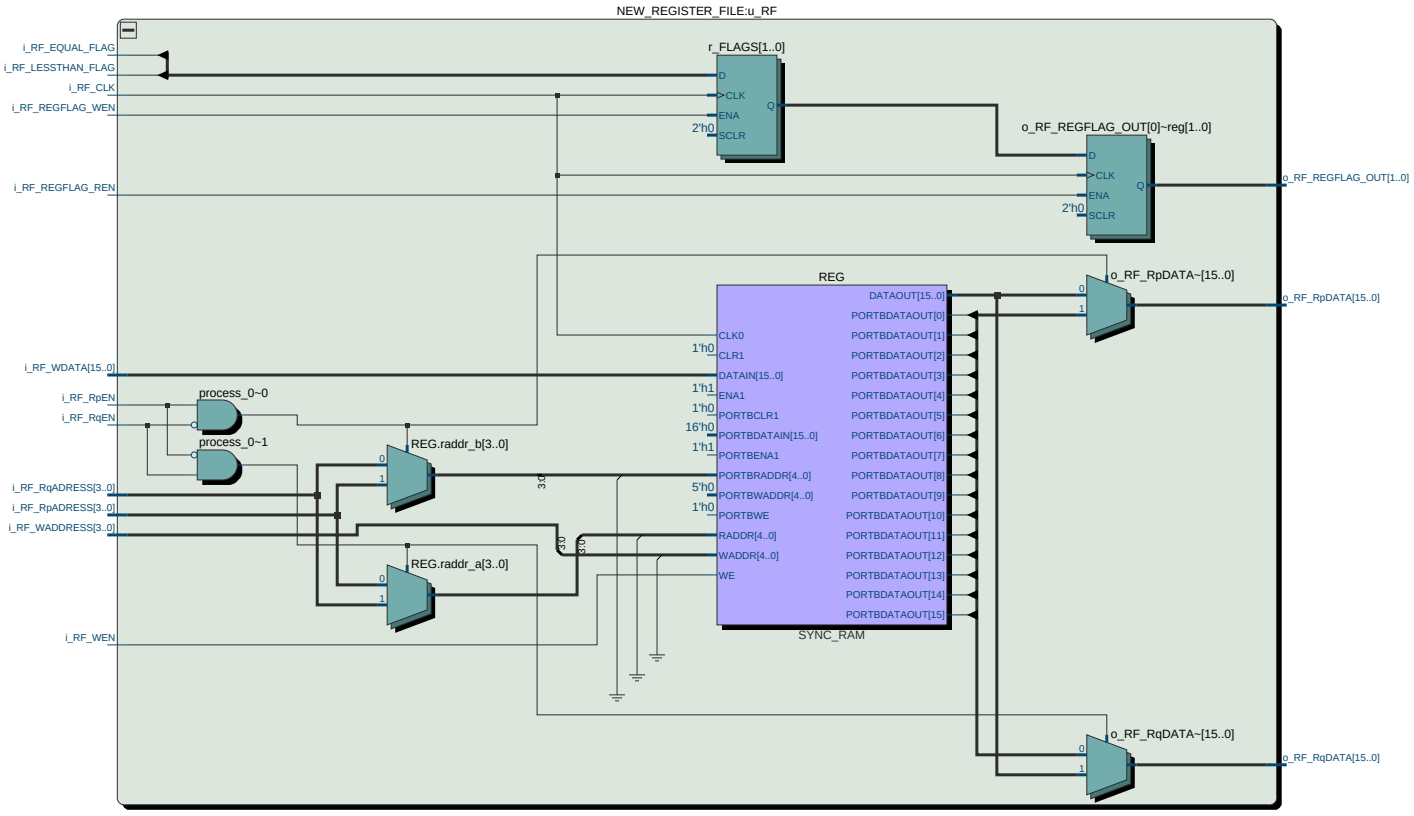
Os registradores são atualizados com valores novos quando seus sinais `ENABLE` respectivos estão altos, o mesmo vale para o registrador de `FLAGS` que tem suas entradas as saídas do `ALU`.

```
if i_RF_RpEN = '1' and i_RF_RqEN = '0' then

        o_RF_RpDATA <= REG( to_integer( unsigned( i_RF_RpADRESS ) ) );
        o_RF_RqDATA <= "UUUUUUUUUUUUUUUU";

        elsif i_RF_RqEN = '1' and i_RF_RpEN = '0' then

        o_RF_RqDATA <= REG( to_integer( unsigned( i_RF_RqADRESS ) ) );
        o_RF_RpDATA <= "UUUUUUUUUUUUUUUU";

        elsif i_RF_RpEN = '1' and i_RF_RqEN = '1' then

        o_RF_RpDATA <= REG( to_integer( unsigned( i_RF_RpADRESS ) ) );
        o_RF_RqDATA <= REG( to_integer( unsigned( i_RF_RqADRESS ) ) );

        else

        o_RF_RqDATA <= "UUUUUUUUUUUUUUUU";
        o_RF_RpDATA <= "UUUUUUUUUUUUUUUU";

end if;
```

As saídas não são sincronizadas com o `CLOCK` podendo ser ativadas a qualquer momento. A saída `o_RF_RpDATA` tem múltiplos destinos, servindo para alimentar a `ALU` e como a entrada para a escrita para o Banco de Dados.

As duas saídas `o_RF_RpDATA` e `o_RF_RqDATA` também são saídas externas do processador, pois como o `Quartus` não compila o projeto se não houver pelo menos uma saída, essas foram escolhidas.

A outra saída do Banco de Registradores que sai do Bloco de Operações é o registrador de `FLAGS` que serve como entrada para o Bloco de Controle, que dita para o mesmo se deve pular ou não instruções no processo.

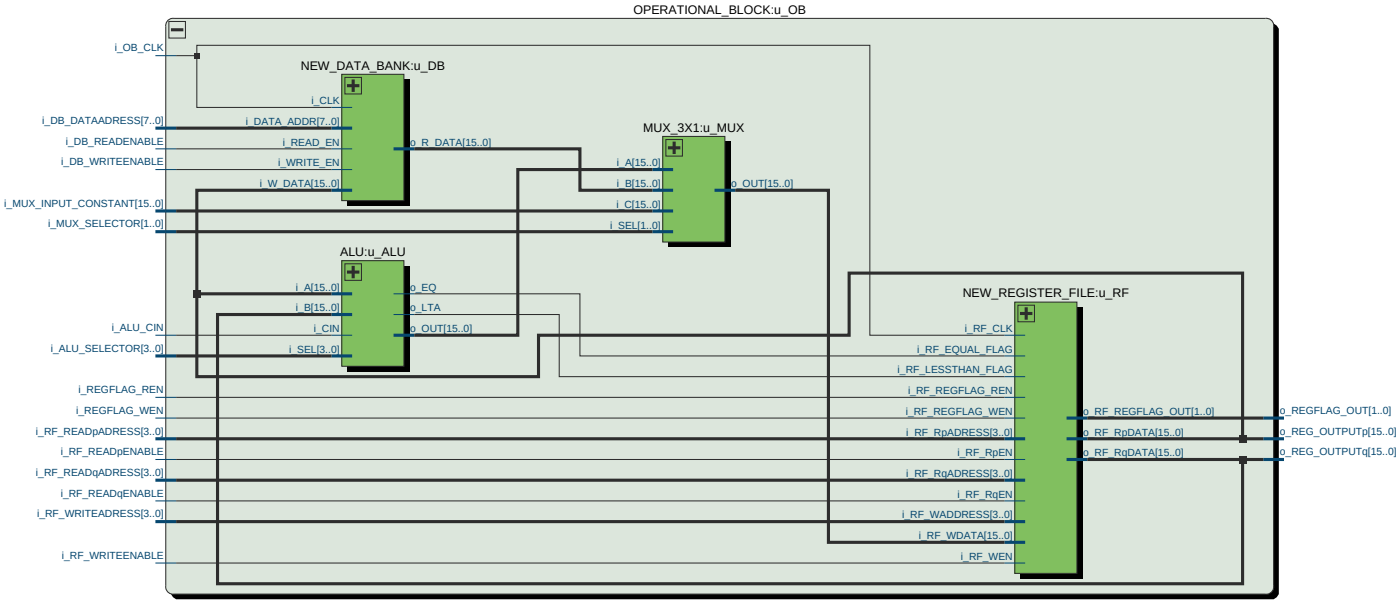

NEW_REGISTER_FILE:u_RF
i_RF_EQUAL_FLAG
i_RF_LESSTHAN_FLAG
i_RF_CLK
i_RF_REGFLAG_WEN
i_RF_REGFLAG_REN
i_RF_WDATA[15..0]
i_RF_RpEN
i_RF_RqEN
i_RF_RqADRESS[3..0]
i_RF_RpADRESS[3..0]
i_RF_WADDRESS[3..0]
i_RF_WEN
r_FLAGS[1..0]
D
CLK
ENA
SCLR
Q
2'h0
o_RF_REGFLAG_OUT[0]~reg[1..0]
o_RF_REGFLAG_OUT[1..0]
process_0~0
process_0~1
REG.raddr_b[3..0]
REG.raddr_a[3..0]
3:0
REG
CLK0
CLR1
DATAIN[15..0]
ENA1
PORTBCLR1
PORTBDATAIN[15..0]
PORTBENA1
PORTBRADDR[4..0]
PORTBWADDR[4..0]
PORTBWE
RADDR[4..0]
WADDR[4..0]
WE
1'h0
1'h1
16'h0
5'h0
DATAOUT[15..0]
PORTBDATAOUT[0]
PORTBDATAOUT[1]
PORTBDATAOUT[2]
PORTBDATAOUT[3]
PORTBDATAOUT[4]
PORTBDATAOUT[5]
PORTBDATAOUT[6]
PORTBDATAOUT[7]
PORTBDATAOUT[8]
PORTBDATAOUT[9]
PORTBDATAOUT[10]
PORTBDATAOUT[11]
PORTBDATAOUT[12]
PORTBDATAOUT[13]
PORTBDATAOUT[14]
PORTBDATAOUT[15]
SYNC_RAM
o_RF_RpDATA~[15..0]
o_RF_RpDATA[15..0]
o_RF_RqDATA~[15..0]
o_RF_RqDATA[15..0]

2.3.1 Código do componente Banco de Registradores

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

entity NEW_REGISTER_FILE is     ---DONE

port(

        i_RF_CLK                : in std_logic                          ;

        i_RF_REGFLAG_WEN        : in std_logic                          ;
            ---REGISTER FLAG WRITE ENABLE
        i_RF_REGFLAG_REN        : in std_logic                          ;
            ---REGISTER FLAG READ ENABLE
        i_RF_EQUAL_FLAG         : in std_logic                          ;
        i_RF_LESSTHAN_FLAG      : in std_logic                          ;
        o_RF_REGFLAG_OUT        : out std_logic_vector( 1 downto 0 )    ;
            ---REGISTER FLAG OUT

        i_RF_WDATA              : in std_logic_vector (15 downto 0)     ;
            ---WRITE DATA IN
        i_RF_WADDRESS           : in std_logic_vector (3 downto 0)      ;
            ---WRITE ADDRESS
        i_RF_WEN                : in std_logic                          ;
            ---WRITE ANABLE

        o_RF_RpDATA             : out std_logic_vector(15 downto 0)     ;
            ---READ DATA P
        i_RF_RpADRESS           : in std_logic_vector(3 downto 0)       ;
            ---READ ADDRESS
        i_RF_RpEN               : in std_logic                          ;

        o_RF_RqDATA             : out std_logic_vector(15 downto 0)     ;
            ---READ DATA Q
        i_RF_RqADRESS           : in std_logic_vector(3 downto 0)       ;
            ---READ ADDRESS
        i_RF_RqEN               : in std_logic

);

end NEW_REGISTER_FILE;

architecture ARCH_1 of NEW_REGISTER_FILE is

        type t_REG is array ( 0 to 16 ) of std_logic_vector( 15 downto 0 );
        signal REG      :       t_REG;

        signal r_FLAGS  :       std_logic_vector( 1 downto 0 );

begin

        process ( i_RF_CLK , i_RF_WEN , i_RF_WADDRESS , i_RF_RpEN , i_RF_RpADRESS
             , i_RF_RqADRESS , i_RF_RqEN , i_RF_WDATA ) begin

                if rising_edge( i_RF_CLK ) then

                        ---WRITE INTO
```

```
                if i_RF_WEN = '1' then

                        REG( to_integer( unsigned( i_RF_WADDRESS ) ) ) <=
                             i_RF_WDATA;   ---WRITE DATA IN

                end if; --i_RF_WEN

                if i_RF_REGFLAG_WEN = '1' then

                        r_FLAGS(1)      <= i_RF_EQUAL_FLAG;
                        r_FLAGS(0)      <= i_RF_LESSTHAN_FLAG;

                end if; --i_RF_REGFLAG_WEN

                if i_RF_REGFLAG_REN = '1' then

                        o_RF_REGFLAG_OUT        <= r_FLAGS;

                end if;

                ---WRITE

        end if; ---CLK

        ---READ ONTO


        if i_RF_RpEN = '1' and i_RF_RqEN = '0' then

        o_RF_RpDATA <= REG( to_integer( unsigned( i_RF_RpADRESS ) ) );
        o_RF_RqDATA <= "UUUUUUUUUUUUUUUU";

        elsif i_RF_RqEN = '1' and i_RF_RpEN = '0' then

        o_RF_RqDATA <= REG( to_integer( unsigned( i_RF_RqADRESS ) ) );
        o_RF_RpDATA <= "UUUUUUUUUUUUUUUU";

        elsif i_RF_RpEN = '1' and i_RF_RqEN = '1' then

        o_RF_RpDATA <= REG( to_integer( unsigned( i_RF_RpADRESS ) ) );
        o_RF_RqDATA <= REG( to_integer( unsigned( i_RF_RqADRESS ) ) );

        else

        o_RF_RqDATA <= "UUUUUUUUUUUUUUUU";
        o_RF_RpDATA <= "UUUUUUUUUUUUUUUU";

        end if;

        ---READ


    end process;

end ARCH_1;
```

2.4 Código do componente Bloco Operacional



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

entity OPERATIONAL_BLOCK is     ---DONE

port(

        i_OB_CLK                :       in std_logic                            ;

        i_REGFLAG_WEN           :       in std_logic                            ;
            ---REGISTER FLAG WRITE ENABLE
        i_REGFLAG_REN           :       in std_logic                            ;
            ---REGISTER FLAG READ ENABLE
        o_REGFLAG_OUT           :       out std_logic_vector                    ;
            ---REGISTER FLAG OUT

        i_MUX_INPUT_CONSTANT    :       in std_logic_vector (15 downto 0)       ;
            --- Input constant for multiplexer i_C
        i_MUX_SELECTOR          :       in std_logic_vector ( 1 downto 0 )      ;
            --- Selector input for multiplexer

        i_RF_WRITEADRESS        :       in std_logic_vector(3 downto 0)         ;
            --- Adress for writing to RF
        i_RF_WRITEENABLE        :       in std_logic                            ;
            --- Enable for writing to RF

        i_RF_READpADRESS        :       in std_logic_vector(3 downto 0)         ;
            --- Adress for Readp from RF
        i_RF_READpENABLE        :       in std_logic                            ;
                    --- Enable for Readp from RF

        i_RF_READqADRESS        :       in std_logic_vector(3 downto 0)         ;
            --- Adress for Readq from RF
        i_RF_READqENABLE        :       in std_logic                            ;
            --- Enable for Readq from RF

        i_ALU_CIN               :       in integer range 0 to 1                 ;
            --- Carry for ALU
        i_ALU_SELECTOR          :       in std_logic_vector( 3 downto 0 )       ;
            --- Selector for ALU

        i_DB_DATAADRESS         :       in std_logic_vector( 7 downto 0 )       ;
            --- Data adress input for Data Bank
        i_DB_READENABLE         :       in std_logic                            ;
            --- Read enable for Data Bank
        i_DB_WRITEENABLE        :       in std_logic                            ;
                    --- Write enable for Data Bank

        o_REG_OUTPUTp           :       out std_logic_vector (15 downto 0);
            ---Data output from reg_file p
        o_REG_OUTPUTq           :       out std_logic_vector (15 downto 0)
            ---Data output from reg_file q

);

end OPERATIONAL_BLOCK;

architecture ARCH_1 of OPERATIONAL_BLOCK is

component NEW_REGISTER_FILE is
```

```
        port(

        i_RF_CLK                : in std_logic                          ;

        i_RF_REGFLAG_WEN        : in std_logic                          ;
            ---REGISTER FLAG WRITE ENABLE
        i_RF_REGFLAG_REN        : in std_logic                          ;
            ---REGISTER FLAG READ ENABLE
        i_RF_EQUAL_FLAG         : in std_logic                          ;
        i_RF_LESSTHAN_FLAG      : in std_logic                          ;
        o_RF_REGFLAG_OUT        : out std_logic_vector( 1 downto 0 )    ;
            ---REGISTER FLAG OUT

        i_RF_WDATA              : in std_logic_vector (15 downto 0)     ;
            ---WRITE DATA IN
        i_RF_WADDRESS           : in std_logic_vector (3 downto 0)      ;
            ---WRITE ADDRESS
        i_RF_WEN                : in std_logic                          ;
            ---WRITE ANABLE

        o_RF_RpDATA             : out std_logic_vector(15 downto 0)     ;
            ---READ DATA P
        i_RF_RpADRESS           : in std_logic_vector(3 downto 0)       ;
            ---READ ADDRESS
        i_RF_RpEN               : in std_logic                          ;

        o_RF_RqDATA             : out std_logic_vector(15 downto 0)     ;
            ---READ DATA Q
        i_RF_RqADRESS           : in std_logic_vector(3 downto 0)       ;
            ---READ ADDRESS
        i_RF_RqEN               : in std_logic

        );

end component;

component MUX_3X1 is

        port(

                i_A     : in std_logic_vector ( 15 downto 0 )   ;
                    ---ALU
                i_B     : in std_logic_vector ( 15 downto 0 )   ;
                    ---DATA BANK
                i_C     : in std_logic_vector ( 15 downto 0 )   ;
                    ---CONSTANT

                i_SEL   : in std_logic_vector ( 1 downto 0 )    ;
                    --- 0000 load ALU , 0001 load DATA BANK , 0010 load CONSTANT

                o_OUT   : out std_logic_vector ( 15 downto 0 )

                );

end component;

component ALU is

        port(

        i_A     : in std_logic_vector( 15 downto 0 )    ;
```

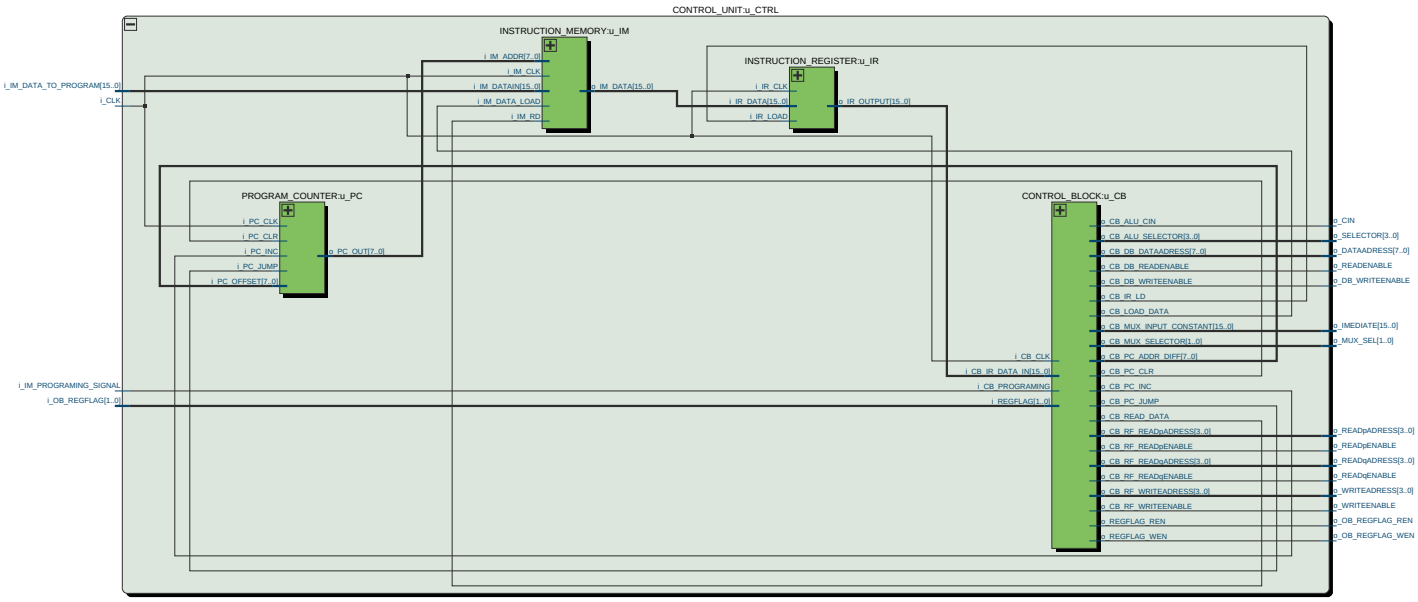
```
        i_B     : in std_logic_vector( 15 downto 0 )    ;
        i_CIN   : in integer range 0 to 1               ;
            ---Carry in for the shift and for the adder

        i_SEL   : in std_logic_vector( 3 downto 0 )     ;

        o_OUT   : out std_logic_vector( 15 downto 0 )   ;
        o_EQ    : out std_logic                         ;
            ---CMP two vectors , EQUAL FLAG
        o_LTA   : out std_logic          ---CMP two vectors , If B less than A ,
            o_LTA = 1

                );

end component;

component NEW_DATA_BANK is      ---DONE

        port(

                i_CLK           : in std_logic;

                i_DATA_ADDR     : in std_logic_vector( 7 downto 0 );

                i_READ_EN       : in std_logic;
                i_WRITE_EN      : in std_logic;

                i_W_DATA        : in std_logic_vector( 15 downto 0 );
                o_R_DATA        : out std_logic_vector( 15 downto 0 )

                );

end component;

        signal w_OUTPUT_FROM_ALU        :       std_logic_vector (15 downto 0)  ;
        signal w_OUTPUT_FROM_MUX        :       std_logic_vector ( 15 downto 0 );

        signal w_OUTPUTp_FROM_RF        :       std_logic_vector(15 downto 0)   ;
        signal w_OUTPUTq_FROM_RF        :       std_logic_vector(15 downto 0)   ;

        signal w_OUTPUT_FROM_DATA_BANK  :       std_logic_vector(15 downto 0)   ;

        signal w_EQUAL_FLAG             :       std_logic       ;
        signal w_LESSTHAN_FLAG          :       std_logic       ;

begin

        u_MUX   :       MUX_3X1 port map(

                i_A     =>      w_OUTPUT_FROM_ALU       ,
                i_B     =>      w_OUTPUT_FROM_DATA_BANK ,
                i_C     =>      i_MUX_INPUT_CONSTANT    ,

                i_SEL   =>      i_MUX_SELECTOR          ,

                o_OUT   => w_OUTPUT_FROM_MUX

                );

        u_RF    :       NEW_REGISTER_FILE port map(

                i_RF_CLK                =>      i_OB_CLK                ,
```

```
                i_RF_REGFLAG_WEN        =>      i_REGFLAG_WEN           ,
                i_RF_REGFLAG_REN        =>      i_REGFLAG_REN           ,
                i_RF_EQUAL_FLAG         =>      w_EQUAL_FLAG            ,
                i_RF_LESSTHAN_FLAG      =>      w_LESSTHAN_FLAG         ,
                o_RF_REGFLAG_OUT        =>      o_REGFLAG_OUT           ,

                i_RF_WDATA              =>      w_OUTPUT_FROM_MUX       ,
                i_RF_WADDRESS           =>      i_RF_WRITEADRESS        ,
                i_RF_WEN                =>      i_RF_WRITEENABLE        ,

                o_RF_RpDATA             =>      w_OUTPUTp_FROM_RF       ,
                i_RF_RpADRESS           =>      i_RF_READpADRESS        ,
                i_RF_RpEN               =>      i_RF_READpENABLE        ,

                o_RF_RqDATA             =>      w_OUTPUTq_FROM_RF       ,
                i_RF_RqADRESS           =>      i_RF_READqADRESS        ,
                i_RF_RqEN               =>      i_RF_READqENABLE

                );

        u_ALU   :       ALU port map(

                i_A     =>      w_OUTPUTp_FROM_RF       ,
                i_B     =>      w_OUTPUTq_FROM_RF       ,
                i_CIN   =>      i_ALU_CIN               ,

                i_SEL   =>      i_ALU_SELECTOR          ,

                o_OUT   =>      w_OUTPUT_FROM_ALU       ,
                o_EQ    =>      w_EQUAL_FLAG            ,
                o_LTA   =>      w_LESSTHAN_FLAG


                );

        u_DB    :       NEW_DATA_BANK port map(

                i_CLK           =>      i_OB_CLK                ,

                i_DATA_ADDR     =>      i_DB_DATAADRESS         ,

                i_READ_EN       =>      i_DB_READENABLE         ,
                i_WRITE_EN      =>      i_DB_WRITEENABLE        ,

                i_W_DATA        =>      w_OUTPUTp_FROM_RF       ,
                o_R_DATA        =>      w_OUTPUT_FROM_DATA_BANK

                );

        o_REG_OUTPUTp   <= w_OUTPUTp_FROM_RF;
        o_REG_OUTPUTq   <= w_OUTPUTq_FROM_RF;

end ARCH_1;
```

3 Unidade de Controle

A Unidade de Controle é um componente que contem outros componentes necessários para mandar sinais para o Bloco de Operações, dessa forma o Bloco pode saber o que fazer com a informação providenciada.

Note que o Bloco de Controle e a Unidade de Controle são componentes diferentes. A Unidade de Controle contem o Bloco de Controle, o Bloco de Controle é o componente principal da Unidade de Controle.



3.1 Registrador de Instruções

O Registrador de Instruções recebe um vetor binário que contem as informações que vão ser executadas.

Os quatro bits mais significativos são a operação a ser executada, essas podem ser a escrita de um valor imediato a um registrador, a escrita de um registrador a um espaço de memória, a soma dos valores de dois registradores, um pulo sobre um a condição, etc.

Os quatro bits subsequentes são o endereço de um registrador caso a operação precise de um, ou zeros se não for necessário.

Os últimos oito bits servem como o endereço para o Banco de Dados, ou pode ser dividido em dois endereços para o Banco de Registradores, como na operação de adição e subtração.

A seguir são todos os `OpCodes`:

OpCode	4 Bits	4 Bits	4 Bits	Definição
0000	Endereço de Registro [X]	Endereço de Data [Y]		Reg[X] = DB[Y]
0001	Endereço de Registro [X]	Endereço de Data [Y]		DB[Y] = Reg[X]
0010	Endereço de Registro [X]	Endereço de Registro [Y]	Endereço de Registro [Z]	Reg[X] = Reg[Y] + Reg[Z]
0011	Endereço de Registro [X]	Valor Imediato [#C]		Reg[X] = #C -1
0100	*0000*	Endereço de Registro [X]	Endereço de Registro [Y]	CMP Reg[X] , Reg[Y]
0101	*0000*	Diferença de enderço[#C]		JE #C -1
0110	Endereço de Registro [X]	Endereço de Registro [Y]	Endereço de Registro [Z]	Reg[X] = Reg[Y] - Reg[Z]
0111	*0000*	Diferença de enderço[#C]		JLT #C -1
1000	*0000*	Diferença de enderço[#C]		JNE #C -1
1001	*0000*	Diferença de enderço[#C]		JNLT #C -1

Note como os pulos condicionais `JE , JLT , JNE , JNLT` tem a diferença de endereço diminuída por um, pois logo após o pulo o próximo estado incrementará o contador de programa por um. Essa subtração deve ser feita no código, pois o circuito não faz essa subtração.

Também pode ser notado nos quatro bits de endereço de registro dos pulos combinacionais não é usado, em retrospectiva esse espaço poderia ser usado para ditar qual tipo de condição pra fazer o pulo.

3.1.1 Explicação do código

```
port(

        i_IR_CLK        :       in std_logic                            ;
        i_IR_DATA       :       in std_logic_vector( 15 downto 0 )      ;
        i_IR_LOAD       :       in std_logic                            ;

        o_IR_OUTPUT     :       out std_logic_vector( 15 downto 0 )

);
```

Esse é o mais simples circuito na Unidade de Controle, seu RTL é apenas um registrador, como deveria ser, mas esse componente tem uma grande importância pois sua saída está diretamente ligado ao Bloco de Controle.

```
signal r_INS_REGIST     :       std_logic_vector( 15 downto 0 );

begin

process ( i_IR_CLK , i_IR_LOAD , r_INS_REGIST , i_IR_DATA ) begin

if rising_edge(i_IR_CLK) then

        if i_IR_LOAD = '1' then

        r_INS_REGIST <= i_IR_DATA;

        end if; ---i_IR_LOAD

end if; ---i_IR_CLK

end process;

        o_IR_OUTPUT <= r_INS_REGIST;
```

Isso significa que instruções podem ser puladas por acidente, ou deliberadamente pelo sinal de escrita que uma instrução pode ter.A entrada de informações vem diretamente da saída da Memória de Instruções.

3.1.2 Código do componente Registrador de Instruções

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

entity INSTRUCTION_REGISTER is  ---DONE

port(

        i_IR_CLK                :       in std_logic;
        i_IR_DATA       :       in std_logic_vector( 15 downto 0 );
        i_IR_LOAD       :       in std_logic;

        o_IR_OUTPUT     :       out std_logic_vector( 15 downto 0 )

);

end INSTRUCTION_REGISTER;

architecture ARCH_1 of INSTRUCTION_REGISTER is

signal r_INS_REGIST     :       std_logic_vector( 15 downto 0 );

begin

        process ( i_IR_CLK , i_IR_LOAD , r_INS_REGIST , i_IR_DATA ) begin

        if rising_edge(i_IR_CLK) then

                if i_IR_LOAD = '1' then

                r_INS_REGIST <= i_IR_DATA;

                end if; ---i_IR_LOAD

        end if; ---i_IR_CLK

        end process;

        o_IR_OUTPUT <= r_INS_REGIST;

end ARCH_1;
```

3.2 Contador de Programa

O Contador de Programa serve para atualizar o Registrador de Instruções com o endereço de novos comandos para serem usados, dessa forma todo o programa é executado se não houver nenhum tipo de condicional, e caso haja uma condicional, o Contador de Programas pula para a instrução ditada pelo pulo.

3.2.1 Explicação do código

```
port(

        i_PC_CLK        : in std_logic;
        i_PC_CLR        : in std_logic;
        i_PC_INC        : in std_logic;

        i_PC_JUMP       : in std_logic;
        i_PC_OFFSET     : in std_logic_vector ( 7 downto 0 );

        o_PC_OUT        : out std_logic_vector ( 7 downto 0 )

);
```

Normalmente o contador apenas incrementa por um, pois um programa não pode ser feito de apenas pulos condicionais.

```
signal r_COUNTER : std_logic_vector( 7 downto 0 ) := "00000000";

begin

process ( i_PC_CLK , i_PC_CLR , i_PC_INC ) begin

if rising_edge(i_PC_CLK) then

        if i_PC_JUMP = '0' then

                if i_PC_CLR = '1' then r_COUNTER <= "00000000"; end if;
                if i_PC_INC = '1' then r_COUNTER <= std_logic_vector( signed(
                    r_COUNTER ) + 1 );end if;

        else r_COUNTER <= std_logic_vector( signed( r_COUNTER ) + signed(
            i_PC_OFFSET ) );

        end if;

end if;
```

Porem caso seja necessário dar um pulo, seja mais adiante ou não, o sinal de pulo será ativado, com o valor de diferença entre onde o contador está e onde deve ir. Note como o processo usa um valor com sinal, isso serve para indicar se o valor é negativo ou não.

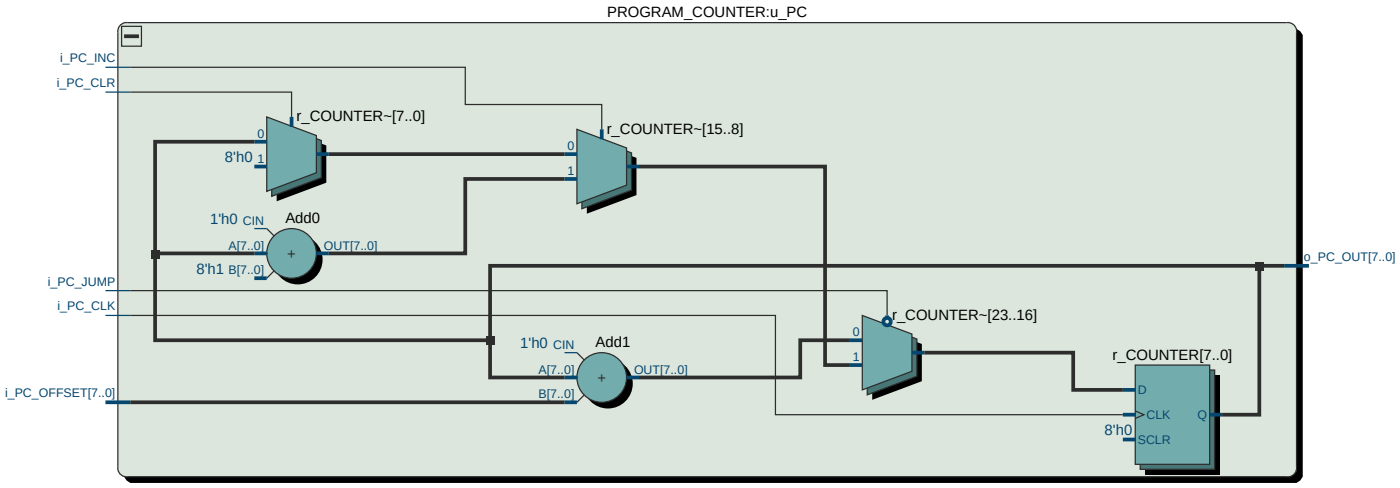
Apesar de que o valor guardado no registrador será sempre positivo pois não há uma instrução em um endereço negativo, a biblioteca não permite aritmética entre valores com sinal e sem sinal, portanto qunado um pulo é feito, apenas sete bits podem ser usados para a contagem de endereço pois o bit mais significativo na aritmética é o sinal.

```
end process;
```

```
        o_PC_OUT <= r_COUNTER;

end ARCH_1;
```

O valor do registrador é mostrado a todos os tempos, a saída é levada a entrada de endereços da Memória de Instruções.

3.2.2 Código do componente Contador de Programa



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

entity PROGRAM_COUNTER is

port(

        i_PC_CLK        : in std_logic;
        i_PC_CLR        : in std_logic;
        i_PC_INC        : in std_logic;

        i_PC_JUMP       : in std_logic;
        i_PC_OFFSET     : in std_logic_vector ( 7 downto 0 );

        o_PC_OUT        : out std_logic_vector ( 7 downto 0 )

);

end PROGRAM_COUNTER;

architecture ARCH_1 of PROGRAM_COUNTER is

signal r_COUNTER : std_logic_vector( 7 downto 0 ) := "00000000";

begin

process ( i_PC_CLK , i_PC_CLR , i_PC_INC ) begin

if rising_edge(i_PC_CLK) then

        if i_PC_JUMP = '0' then

                if i_PC_CLR = '1' then r_COUNTER <= "00000000"; end if;
                if i_PC_INC = '1' then r_COUNTER <= std_logic_vector( signed(
                    r_COUNTER ) + 1 );end if;

        else r_COUNTER <= std_logic_vector( signed( r_COUNTER ) + signed(
            i_PC_OFFSET ) );

        end if;

end if;

end process;

        o_PC_OUT <= r_COUNTER;

end ARCH_1;
```

3.3 Memória de Instruções

A Memória de Instruções guarda as instruções dada pelo usuário no inicio da execução do Processador, Normalmente essa memória é apenas para leitura durante a execução, porem pode ser mudada durante a execução do programa, podendo ser criado um código que escreve em si mesmo, porem precisaria ser mudado os sinais de alguns estados do Bloco de Controle.

Como o Processador não foi testado em um FPGA, o processo de escrever os comandos são feitos apenas no TestBench.

3.3.1 Explicação do código

```
port(

        i_IM_CLK        :       in std_logic                            ;
        i_IM_ADDR       :       in std_logic_vector( 7 downto 0 )       ;
            ---MEMORY TAKEN DOWN TO 8 BITS, or 255 possibilities
        i_IM_RD         :       in std_logic                            ;
            ---ENABLE FOR OUTPUT
        i_IM_DATA_LOAD  :       in std_logic                            ;
            ---ENABLE FOR COMMAND INPUT
        i_IM_DATAIN     :       in std_logic_vector( 15 downto 0 )      ;
            ---COMMANDS IN
        o_IM_DATA       :       out std_logic_vector( 15 downto 0 )

);
```

O endereço usado para cada instrução é dado pelo Contador de Programas, durante a escrita do código o Contador aumenta normalmente, mas quando a escrita acaba o contador volta ao zero, preparando-se para a execução.

```
type t_INS_MEM is array ( 0 to 255 ) of std_logic_vector( 15 downto 0 );
signal INS_MEM  :       t_INS_MEM;

begin

process ( i_IM_CLK , i_IM_DATAIN , i_IM_ADDR , INS_MEM ) begin

        if rising_edge(i_IM_CLK) then

                if i_IM_DATA_LOAD = '1' then

                        INS_MEM( to_integer( unsigned( i_IM_ADDR ) ) )  <=
                            i_IM_DATAIN;

                end if; ---i_IM_DATA_LOAD

                if i_IM_RD = '1' then

                        o_IM_DATA       <=      INS_MEM( to_integer( unsigned(
                            i_IM_ADDR ) ) );

                else

                        o_IM_DATA       <=      "UUUUUUUUUUUUUUUU";

                end if; ---i_IM_RD
```

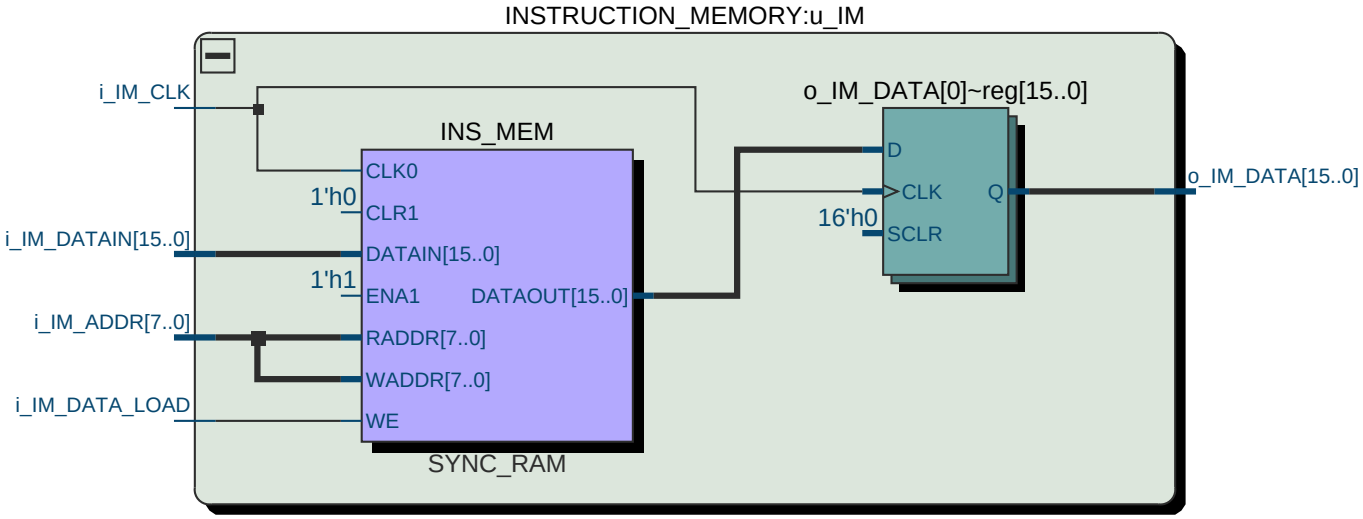
```
    end if; ---i_IM_CLK
```

Apesar de que o componente pode ler e escrever, não é recomendado a não ser que a intenção seja criar um algorítimo que se escreve, em qualquer momento a Memoria de Instrução vai estar executando alguma instrução, ou escrevendo em algum endereço, não os dois.

A entrada de dados a ser gravado na memória vem de fora do componente TOP, ou seja, vem do usuário, ou algum outro circuito que o usuário poderia estar usando para passar informações.

Esse componente não faz parte do Processador, mas é necessário para testa-lo, o algorítimo sendo executado e escrito deveria estar em uma memória RAM separado do Processador.

3.3.2 Código do componente Memória de Instruções

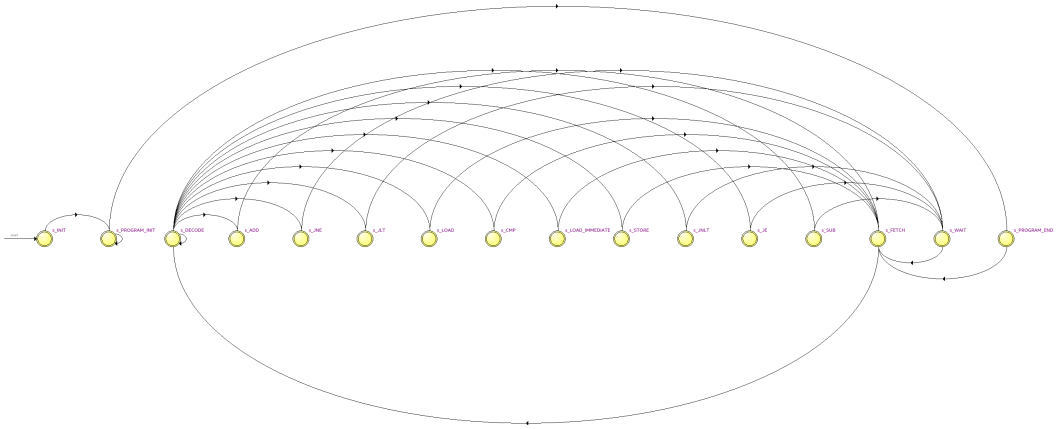


```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity INSTRUCTION_MEMORY is

port(

        i_IM_CLK        :       in std_logic                            ;
        i_IM_ADDR       :       in std_logic_vector( 7 downto 0 )       ;
            ---MEMORY TAKEN DOWN TO 8 BITS, or 255 possibilities
        i_IM_RD         :       in std_logic                            ;
            ---ENABLE FOR OUTPUT
        i_IM_DATA_LOAD  :       in std_logic                            ;
            ---ENABLE FOR COMMAND INPUT
        i_IM_DATAIN     :       in std_logic_vector( 15 downto 0 )      ;
            ---COMMANDS IN
        o_IM_DATA       :       out std_logic_vector( 15 downto 0 )

);

end INSTRUCTION_MEMORY;

architecture ARCH_1 of INSTRUCTION_MEMORY is

type t_INS_MEM is array ( 0 to 255 ) of std_logic_vector( 15 downto 0 );
signal INS_MEM  :       t_INS_MEM;

begin

process ( i_IM_CLK , i_IM_DATAIN , i_IM_ADDR , INS_MEM ) begin

        if rising_edge(i_IM_CLK) then

                if i_IM_DATA_LOAD = '1' then

                        INS_MEM( to_integer( unsigned( i_IM_ADDR ) ) )  <=
                            i_IM_DATAIN;

                end if; ---i_IM_DATA_LOAD

                if i_IM_RD = '1' then

                        o_IM_DATA       <=      INS_MEM( to_integer( unsigned(
                            i_IM_ADDR ) ) );

                else

                        o_IM_DATA       <=      "UUUUUUUUUUUUUUUU";

                end if; ---i_IM_RD

        end if; ---i_IM_CLK

end process;

end ARCH_1;
```

3.4 Bloco de Controle

O Bloco de Controle é o componente mais complexo da Unidade de Controle, esse componente guarda os estados e respectivos sinais. Esse componente separa os dados no Registrador de Instruções em pedaços que possam ser usados e decide o que fazer baseado neles.

O Bloco de Controle também recebe e manda sinais para o Bloco de Operações para que seja executado o algorítimo, o Bloco de Operações é responsável em mandar informações sobre os dois últimos valores trabalhados em sua ALU, e o Bloco de Controle é responsável em mandar sinais e endereços para que sejam trabalhados.



A imagem é muito grande, aproxime-se para vê-la

3.4.1 Explicação do código

```
port(

        i_CB_CLK                :       in std_logic                            ;
        ---INSTRUCTION REGISTER

        i_CB_IR_DATA_IN         :       in std_logic_vector( 15 downto 0 )      ;
                    ---Instructions from instruction register
        o_CB_IR_LD              :       out std_logic                           ;
                    ---Signal for instruction register to load contents

        ---/INSTRUCTION REGISTER

        ---PROGRAM COUNTER

        o_CB_PC_CLR                     :       out std_logic                   ;
                    ---Signal for Program Counter to clear
        o_CB_PC_INC                     :       out std_logic                   ;
                    ---Signal for Program Counter to increment
        o_CB_PC_JUMP                    :       out std_logic                   ;
                    ---Signal for program counter to jump
```

```
o_CB_PC_ADDR_DIFF               :       out std_logic_vector ( 7 downto 0
     )      ;       ---Difference that PC has to account for, use for
    conditional jumps


---/PROGRAM COUNTER

---INSTRUCTION MEMORY

i_CB_PROGRAMING                 :       in std_logic                    ;
            ---If 1 , programing
o_CB_LOAD_DATA                  :       out std_logic                   ;
            ---Signal for Instruction Memory to load Data at addres from
    PC
o_CB_READ_DATA                  :       out std_logic                   ;
            ---Signal for Instruction Memory to dum its contents into
    Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT         :       out std_logic_vector(15 downto 0)
    ;       --- Input constant for multiplexer i_C
o_CB_MUX_SELECTOR               :       out std_logic_vector ( 1 downto 0
     )      ;       --- Selector input for multiplexer

o_CB_RF_WRITEADRESS             :       out std_logic_vector(3 downto 0);
            --- Adress for writing to RF
o_CB_RF_WRITEENABLE             :       out std_logic                   ;
            --- Enable for writing to RF

o_CB_RF_READpADRESS             :       out std_logic_vector(3 downto 0);
            --- Adress for Readp from RF
o_CB_RF_READpENABLE             :       out std_logic                   ;
            --- Enable for Readp from RF

o_CB_RF_READqADRESS             :       out std_logic_vector(3 downto 0);
            --- Adress for Readq from RF
o_CB_RF_READqENABLE             :       out std_logic                   ;
            --- Enable for Readq from RF

o_CB_ALU_CIN                    :       out integer range 0 to 1        ;
            --- Carry for ALU
o_CB_ALU_SELECTOR               :       out std_logic_vector( 3 downto 0
    );      --- Selector for ALU

o_CB_DB_DATAADRESS              :       out std_logic_vector( 7 downto 0
    )       ;       --- Data adress input for Data Bank
o_CB_DB_READENABLE              :       out std_logic                   ;
            --- Read enable for Data Bank
o_CB_DB_WRITEENABLE             :       out std_logic                   ;
            --- Write enable for Data Bank

o_REGFLAG_WEN                   :       out std_logic                   ;
            --- Write enable for comand block's flag register
o_REGFLAG_REN                   :       out std_logic                   ;
            --- Read enable for comand block's flag register
i_REGFLAG                       :       in std_logic_vector ( 1 downto 0
    )               --- Comand Block's flag register

---/BLOCO OPERACIONAL
```

```
);
architecture ARCH_1 of CONTROL_BLOCK is

        type t_STATE is ( s_INIT , s_PROGRAM_INIT , s_PROGRAM_END , s_FETCH ,
            s_DECODE , s_LOAD , s_STORE , s_LOAD_IMMEDIATE , s_ADD , s_CMP , s_JE
             , s_SUB , s_JLT , s_JNE , s_JNLT , s_WAIT );

        signal r_STATE          : t_STATE := s_INIT ;
        signal w_NEXT           : t_STATE := s_INIT ;
```

Os nomes dos estados usados são auto descritivos em sua maioria, porem há alguns que precisam ser explicados em mais detalhes.

Cada estado manda um punhado de sinais para múltiplos componentes do Bloco Operacional ou para a Unidade de Controle, esses sinais são, em ordem que foram programados:

Definição	Possível valor
Permissão para o registrador de FLAGS atualizar	X
Permissão para o registrador de FLAGS escrever	X
Permissão para o registrador de Instruções atualizar	X
Permissão para o Contador de Programas zerar	X
Permissão para o Contador de Programas incrementar	X
Permissão para o Contador de Programas pular	X
Diferença de enderço para o Contador de Programas	XXXXXXXX
Permissão para memória carregar instruções do usuário	X
Permissão para memória escrever ao Registrador de Instruções	X
Valor da constante a ser usado pela ALU	XXXXXXXXXXXXXXXX
Valor do seletor do multiplexador no Bloco Operacional	XX
Endereço a escrever no Banco de Registradores	XXXX
Permissão para escrever no Banco de Registradores	X
Endereço a ler do Banco de Registradores na saída P	XXXX
Permissão para ler do Banco de Registradores na saída P	X
Endereço a ler do Banco de Registradores na saída Q	XXXX
Permissão para ler do Banco de Registradores na saída Q	X
Carry para aritmética do ALU	X
Seletor do multiplexador do ALU	XXXX
Endereço para escrita ou leitura do Banco de Dados	XXXXXXXX
Permissão para a leitura do Banco de Dados	X
Permissão para a escrita ao Banco de Dados	X

Por questão de brevidade e na tentativa de não ofuscar o significado de cada estado, não será mostrado o código de cada estado e seus respectivos sinais, mas sim a lista acima para que fique mais fácil a compreensão das suas funções. Caso seja necessário ver o código, ele estará disponível mais adiante.

```
if rising_edge( i_CB_CLK ) then

        case r_STATE is

        when s_INIT     =>

                ---CLEAR PC
                ---EVERYTHING IS OFF
                w_NEXT <= s_PROGRAM_INIT;
```

O estado inicial, ou seja, o estado que o Processador tem quando ligado pela primeira vez é o estado `s_INIT`, ele não faz muito, apenas permite que os valores iniciais sejam colocados nos seus lugares.

Também zera o Contador de Programas, para ter certeza que realmente está em zero.

s_INIT	Possível valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	0
Permissão para o registrador de Instruções atualizar	0
Permissão para o Contador de Programas zerar	1
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	00000000
Permissão para memória carregar instruções do usuário	0
Permissão para memória escrever ao Registrador de Instruções	0
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	UU
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

```
when s_PROGRAM_INIT     =>

        ---SEND SIGNAL TO INSTRUCTION MEMORY TO UPDATE ITSELF BASED ON OFF SITE
            DATA
        ---ALLOW FOR PC TO INCREMENT

        if i_CB_PROGRAMING = '1' then

                w_NEXT <= s_PROGRAM_INIT;

        else

                w_NEXT <= s_PROGRAM_END;

        end if;
```

O estado seguinte é o estado de inicio de programação, `s_PROGRAM_INIT`. Esse estado permite a escrita do algorítimo em pedaços de código que podem ser gravados na Memória de Instruções um pedaço de cada vez, nenhum outro componente está fazendo algo importante nesse tempo. Também é necessário que o usuário avise o Bloco de Controle que está programando, e ainda não terminou.

s_PROGRAM_INIT	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	0
Permissão para o registrador de Instruções atualizar	0
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	1
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	00000000
Permissão para memória carregar instruções do usuario	1
Permissão para memória escrever ao Registrador de Instruções	0
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	UU
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

```
when s_PROGRAM_END              =>

        ---CLEAR PC
        ---SEND SIGNAL TO INSTRUCTION MEMORY TO STOP WRITING, AND START READING
        ---SEND SIGNAL TO INSTRUCTION REGISTER TO START READING

        w_NEXT <= s_FETCH;
```

Foi necessário criar um estado novo unicamente para o fim da programação, pois era necessário limpar o Contador de Programas para que a execução do algorítimo começasse do zero. Como o proximo estado seria `s_FETCH`, limpar o Contador de Programas nesse estado resultaria no Contador sempre reiniciando.

s_PROGRAM_END	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	0
Permissão para o registrador de Instruções atualizar	1
Permissão para o Contador de Programas zerar	1
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	00000000
Permissão para memória carregar instruções do usuario	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	UU
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

```
when s_FETCH    =>

        ---LOAD w_OPERATOR AND w_OPERAND_ FOR NEXT STATE , INCREASE PC

        w_NEXT <= s_DECODE;
```

O estado de Busca serve para carregar uma instrução da Memória de Instruções e guarda-la no Registrador de Instrução, também incrementa o Contador de Programas. Esse também é o motivo pelo qual o endereço do pulo condicional deve ser subtraído por um.

s_FETCH	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	1
Permissão para o registrador de Instruções atualizar	1
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	1
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	UUUUUUUU
Permissão para memória carregar instruções do usuário	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	UU
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

```
when s_DECODE                   =>

        ----DEPENDING ON OPCODE FROM w_OPERATOR , CHANGE STATES
            ACCORDINGLY

        if      i_CB_IR_DATA_IN( 15 downto 12 ) = "0000" then   ---LOAD
            FROM DATA_BANK INTO REGISTER

            w_NEXT <= s_LOAD;

        elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0001" then     ---STORE
            FROM REGISTER INTO DATA_BANK

            w_NEXT <= s_STORE;

        elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0010" then     ---TAKE
            FROM 2 REGISTERS AND ADD TEHM INTO ANOTHER REGISTER

            w_NEXT <= s_ADD;

        elsif   i_CB_IR_DATA_IN( 15 downto 12 ) = "0011" then   ---LOAD
            IMMEDIATE INTO REGISTER

            w_NEXT <= s_LOAD_IMMEDIATE;

        elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0100" then

            w_NEXT <= s_CMP;

        elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0101" then

            w_NEXT <= s_JE;

        elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0110" then

            w_NEXT <= s_SUB;

        elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0111" then

            w_NEXT <= s_JLT;

        elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "1000" then

            w_NEXT <= s_JNE;

        elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "1001" then

            w_NEXT <= s_JNLT;

        end if;
```

O estado de decodificação separa o vetor do Registrador de Instruções em dois pedaços, o `OpCode`, e o resto. O `OpCode` decide qual estado será o próximo baseado no seu valor.

s_DECODE	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	1
Permissão para o registrador de Instruções atualizar	1
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	UUUUUUUU
Permissão para memória carregar instruções do usuário	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	UU
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

Adendo: Apesar de que a permissão para o registrador de `FLAGS` atualizar está alto, o registrador não será usado nesse estado ou no estado anterior, apenas no próximo, por isso está sendo atualizado.

```
when s_LOAD     =>

        ---LOAD FROM DATA_BANK TO A REGISTER

        w_NEXT <= s_FETCH;
```

Um dos possíveis estados escolhidos pode ser o `LOAD` cuja função é carregar um valor do Banco de Dados no endereço encontrado nos bits `7 a 0` e guarda-lo em um registrador no endereço nos bits `11 a 8`.

Esses bits são encontrados no vetor `i_CB_IR_DATA_IN`.

s_LOAD	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	0
Permissão para o registrador de Instruções atualizar	0
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	UUUUUUUU
Permissão para memória carregar instruções do usuário	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	01
Endereço a escrever no Banco de Registradores	i_CB_IR_DATA_IN(11 downto 8)
Permissão para escrever no Banco de Registradores	1
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	i_CB_IR_DATA_IN(7 downto 0)
Permissão para a leitura do Banco de Dados	1
Permissão para a escrita ao Banco de Dados	0

```
when s_STORE    =>

        ---STORE FROM REGISTER INTO DATA_BANK

        w_NEXT <= s_FETCH;
```

Outro possível estado pode ser o `STORE` que faz o oposto do estado anterior, guardando o valor de um registrador cujo endereço pode ser encontrado em `11 a 8` e guardado no Banco de Dados cujo endereço pode ser encontrado em `7 a 0`.

s_STORE	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	0
Permissão para o registrador de Instruções atualizar	0
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	UUUUUUUU
Permissão para memória carregar instruções do usuário	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	01
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	i_CB_IR_DATA_IN(11 downto 8)
Permissão para ler do Banco de Registradores na saída P	1
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	i_CB_IR_DATA_IN(7 downto 0)
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	1

Note que a saída sendo usada é a saída `P` pois só ela está conectada com a entrada do Banco de Dados.

```
when s_LOAD_IMMEDIATE   =>

        ---TAKE FROM MUX AND STORE INTO A REGISTER

        w_NEXT <= s_FETCH;
```

Outra possibilidade é a gravação de um valor imediato dado pelo usuário a um registrador que pode ser encontrado em `11 a 8`. O valor dado pelo usuário vai estar em `7 a 0` porem como o registrador possui a capacidade de dezesseis bits, e apenas oito estão sendo gravados, todos os bits a frente do valor carregado serão zerados, isso significa que o valor não pode ser negativo pois não há como ativar o bit de sinal.

Esse erro foi causado por tentar seguir ao máximo o diagrama dado durante o design e não foi percebido a discrepância de capacidades, e pela falta de tempo não foi possível ser consertado.

s_LOAD_IMMEDIATE	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	0
Permissão para o registrador de Instruções atualizar	1
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	UUUUUUUU
Permissão para memória carregar instruções do usuário	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	"00000000"& i_CB_IR_DATA_IN(7 downto 0)
Valor do seletor do multiplexador no Bloco Operacional	10
Endereço a escrever no Banco de Registradores	i_CB_IR_DATA_IN(11 downto 8)
Permissão para escrever no Banco de Registradores	1
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

```
when s_ADD      =>
        ---Operand to will have both register adresses
        w_NEXT <= s_WAIT
when s_SUB      =>
        w_NEXT <= s_WAIT;;
```

Outra possibilidade é a soma ou a subtração, essas serão explicadas juntas pois são praticamente iguais, a unica diferença sendo o valor de endereçamento da `ALU`.

Ambos esses estados guardam em um registrador cujo o endereço pode ser encontrado em `11 a 8` a aritmética entre dois registradores que podem ser encontrados em `7 a 4` e `3 a 0` respectivamente. Note que a aritmética é `Registrador[X] +- Registrador[Y]` então é importante tomar cuidado com a ordem.

s_ADD, s_SUB	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	0
Permissão para o registrador de Instruções atualizar	0
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	UUUUUUUU
Permissão para memória carregar instruções do usuario	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	00
Endereço a escrever no Banco de Registradores	i_CB_IR_DATA_IN(11 downto 8)
Permissão para escrever no Banco de Registradores	1
Endereço a ler do Banco de Registradores na saída P	i_CB_IR_DATA_IN(7 downto 4)
Permissão para ler do Banco de Registradores na saída P	1
Endereço a ler do Banco de Registradores na saída Q	i_CB_IR_DATA_IN(3 downto 0)
Permissão para ler do Banco de Registradores na saída Q	1
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	0001 se ADD, 0010 se SUB
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

```
when s_CMP      =>

        w_NEXT <= s_FETCH;
```

Outra possibilidade é a comparação entre dois registradores que podem ser encontrados em `7 a 4` e `3 a 0` respectivamente. Essa comparação dita qual é o maior, ou se são iguais, esses valores são dados ao Banco de Registradores como sinais separados mas são guardados em um registrador próprio para ser dado ao Bloco de Controle para as operações de pulos condicionais.

Algo importante a ser notado é de que, muito como a aritmética de adição e subtração a ordem dos dois registradores é importante, pois a equação sendo usada é a seguinte, `Registrador[X] = Registrador[Y]` e `Registrador[X] < Registrador[Y]`. Caso sejam verdade, seus respectivos sinais serão altos.

Apesar de que esses testes de comparação estão sendo feitos a todo momento apenas durante o estado de comparação esses sinais são guardados no registrador de `FLAGS`. A entrada do registrador `i_REGFLAG(1)` sendo o sinal de igualdade, e `i_REGFLAG(0)` sendo o sinal de maioridade.

s_CMP	Valor
Permissão para o registrador de FLAGS atualizar	1
Permissão para o registrador de FLAGS escrever	1
Permissão para o registrador de Instruções atualizar	0
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	UUUUUUUU
Permissão para memória carregar instruções do usuario	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	00
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	i_CB_IR_DATA_IN(7 downto 4)
Permissão para ler do Banco de Registradores na saída P	1
Endereço a ler do Banco de Registradores na saída Q	i_CB_IR_DATA_IN(3 downto 0)
Permissão para ler do Banco de Registradores na saída Q	1
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

```
when s_JE               =>      --- Pulo se igual
        w_NEXT <= s_WAIT;
when s_JLT      =>      --- Pulo se maior
        w_NEXT <= s_WAIT;
when s_JNE      =>      --- Pulo se nao igual
        w_NEXT <= s_WAIT;
when s_JNLT     =>      --- Pulo se nao maior
        w_NEXT <= s_WAIT;
```

A seguir são as possibilidades de pulos condicionais, será mostrado apenas dois, o pulo se sinal de igualdade está alto, e pulo se sinal de maioridade está alto. Todos os estados tme seus sinais praticamente iguais com exceção de uma condicional, portanto seria redundante mostra-los todos.

Esses estados testam sua condição e se for verdadeiro, pulam um valor que pode ser encontrado em `7 a 0`, o sinal de permissão de pulo também é ativado para que o pulo aconteça.

s_JE	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	1
Permissão para o registrador de Instruções atualizar	0
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	Se X = Y, 1 : 0
Diferença de enderço para o Contador de Programas	i_CB_IR_DATA_IN(7 downto 0)
Permissão para memória carregar instruções do usuário	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	00
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

s_JLT	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	1
Permissão para o registrador de Instruções atualizar	0
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	Se X <Y, 1 : 0
Diferença de enderço para o Contador de Programas	i_CB_IR_DATA_IN(7 downto 0)
Permissão para memória carregar instruções do usuário	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	00
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

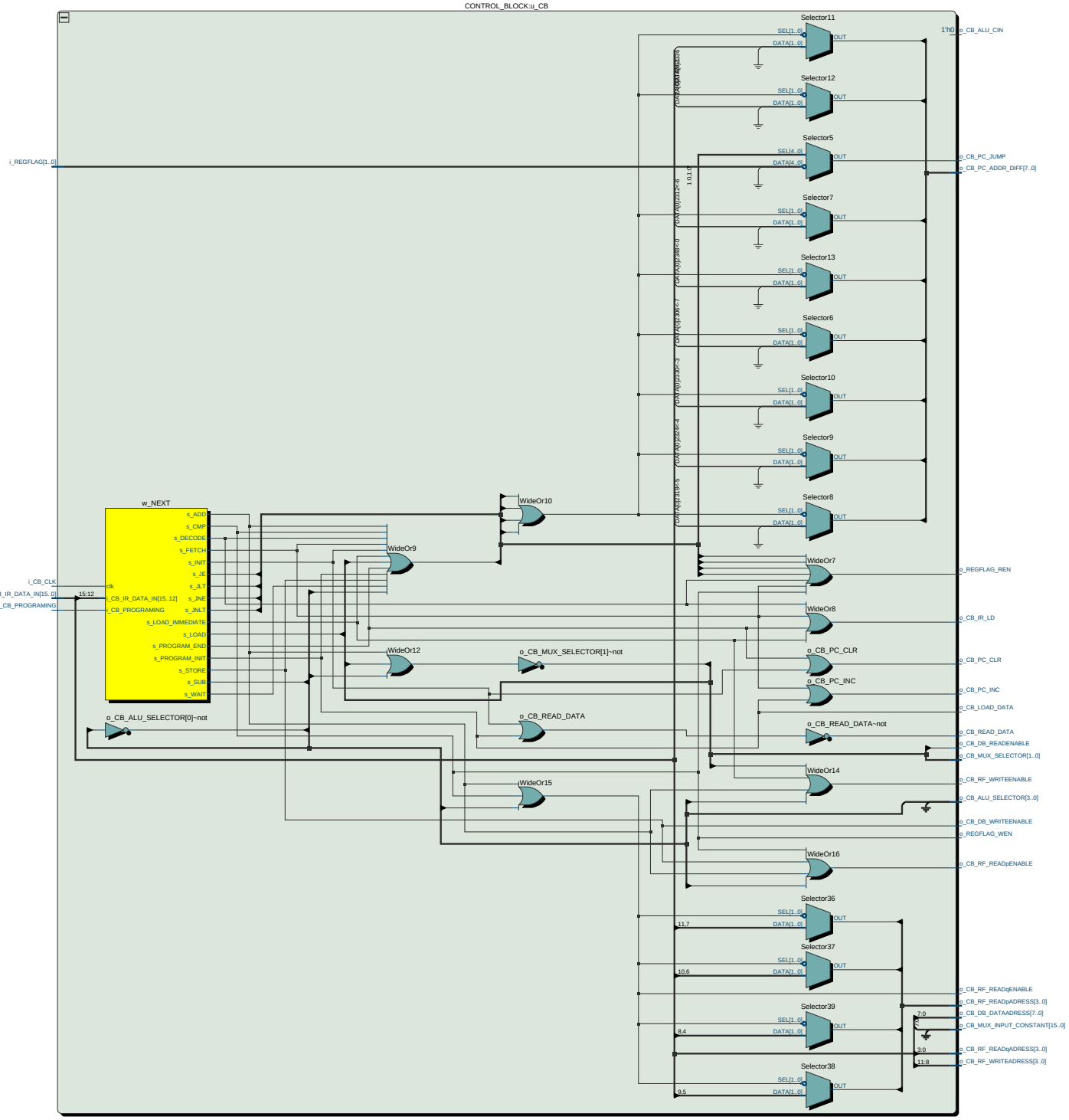
Os outros dois pulos condicionais são praticamente idênticos, sendo sua única diferença a reversão da sua condicional.

```
when s_WAIT     =>

        w_NEXT <= s_FETCH;
```

Depois de todos os pulos condicionais ou processos aritméticos vem o estado de espera, esse foi criado especificamente para permitir que os registradores atualizem com os valores novos, pois como cada estado toma apenas um ciclo de CLOCK é necessário esperar para que o mesmo suba novamente e permitir os registradores atualizarem.

s_WAIT	Valor
Permissão para o registrador de FLAGS atualizar	0
Permissão para o registrador de FLAGS escrever	0
Permissão para o registrador de Instruções atualizar	0
Permissão para o Contador de Programas zerar	0
Permissão para o Contador de Programas incrementar	0
Permissão para o Contador de Programas pular	0
Diferença de enderço para o Contador de Programas	UUUUUUUU
Permissão para memória carregar instruções do usuário	0
Permissão para memória escrever ao Registrador de Instruções	1
Valor da constante a ser usado pela ALU	UUUUUUUUUUUUUUUU
Valor do seletor do multiplexador no Bloco Operacional	00
Endereço a escrever no Banco de Registradores	UUUU
Permissão para escrever no Banco de Registradores	0
Endereço a ler do Banco de Registradores na saída P	UUUU
Permissão para ler do Banco de Registradores na saída P	0
Endereço a ler do Banco de Registradores na saída Q	UUUU
Permissão para ler do Banco de Registradores na saída Q	0
Carry para aritmética do ALU	0
Seletor do multiplexador do ALU	UUUU
Endereço para escrita ou leitura do Banco de Dados	UUUUUUUU
Permissão para a leitura do Banco de Dados	0
Permissão para a escrita ao Banco de Dados	0

3.4.2 Código do componente Bloco de Controle



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

entity CONTROL_BLOCK is ---DONE

port(

        i_CB_CLK                :       in std_logic                            ;
        ---INSTRUCTION REGISTER

        i_CB_IR_DATA_IN         :       in std_logic_vector( 15 downto 0 )      ;
                    ---Instructions from instruction register
        o_CB_IR_LD              :       out std_logic                           ;
                    ---Signal for instruction register to load contents

        ---/INSTRUCTION REGISTER

        ---PROGRAM COUNTER

        o_CB_PC_CLR                     :       out std_logic                   ;
                    ---Signal for Program Counter to clear
        o_CB_PC_INC                     :       out std_logic                   ;
                    ---Signal for Program Counter to increment
        o_CB_PC_JUMP                    :       out std_logic                   ;
                    ---Signal for program counter to jump
        o_CB_PC_ADDR_DIFF               :       out std_logic_vector ( 7 downto 0
             )      ;       ---Difference that PC has to account for, use for
            conditional jumps


        ---/PROGRAM COUNTER

        ---INSTRUCTION MEMORY

        i_CB_PROGRAMING                 :       in std_logic                    ;
                    ---If 1 , programing
        o_CB_LOAD_DATA                  :       out std_logic                   ;
                    ---Signal for Instruction Memory to load Data at addres from
            PC
        o_CB_READ_DATA                  :       out std_logic                   ;
                    ---Signal for Instruction Memory to dum its contents into
            Insruction Register

        ---/INSTRUCTION MEMORY

        ---BLOCO OPERACIONAL

        o_CB_MUX_INPUT_CONSTANT         :       out std_logic_vector(15 downto 0)
            ;       --- Input constant for multiplexer i_C
        o_CB_MUX_SELECTOR               :       out std_logic_vector ( 1 downto 0
             )      ;       --- Selector input for multiplexer

        o_CB_RF_WRITEADRESS             :       out std_logic_vector(3 downto 0);
                    --- Adress for writing to RF
        o_CB_RF_WRITEENABLE             :       out std_logic                   ;
                    --- Enable for writing to RF

        o_CB_RF_READpADRESS             :       out std_logic_vector(3 downto 0);
                    --- Adress for Readp from RF
        o_CB_RF_READpENABLE             :       out std_logic                   ;
                    --- Enable for Readp from RF
```

```
        o_CB_RF_READqADRESS             :       out std_logic_vector(3 downto 0);
                    --- Adress for Readq from RF
        o_CB_RF_READqENABLE             :       out std_logic                   ;
                    --- Enable for Readq from RF

        o_CB_ALU_CIN                    :       out integer range 0 to 1        ;
                    --- Carry for ALU
        o_CB_ALU_SELECTOR               :       out std_logic_vector( 3 downto 0
            );      --- Selector for ALU

        o_CB_DB_DATAADRESS              :       out std_logic_vector( 7 downto 0
            )       ;       --- Data adress input for Data Bank
        o_CB_DB_READENABLE              :       out std_logic                   ;
                    --- Read enable for Data Bank
        o_CB_DB_WRITEENABLE             :       out std_logic                   ;
                    --- Write enable for Data Bank

        o_REGFLAG_WEN                   :       out std_logic                   ;
                    --- Write enable for comand block's flag register
        o_REGFLAG_REN                   :       out std_logic                   ;
                    --- Read enable for comand block's flag register
        i_REGFLAG                       :       in std_logic_vector ( 1 downto 0
            )               --- Comand Block's flag register

        ---/BLOCO OPERACIONAL

);
architecture ARCH_1 of CONTROL_BLOCK is

        type t_STATE is ( s_INIT , s_PROGRAM_INIT , s_PROGRAM_END , s_FETCH ,
            s_DECODE , s_LOAD , s_STORE , s_LOAD_IMMEDIATE , s_ADD , s_CMP , s_JE
             , s_SUB , s_JLT , s_JNE , s_JNLT , s_WAIT );

        signal r_STATE          : t_STATE := s_INIT ;
        signal w_NEXT           : t_STATE := s_INIT ;

begin

        process ( i_CB_CLK , r_STATE , i_CB_PROGRAMING , w_NEXT , i_CB_IR_DATA_IN
             , i_REGFLAG ) begin

        r_STATE <= w_NEXT;

        if rising_edge( i_CB_CLK ) then

                case r_STATE is

                when s_INIT     =>

                        ---CLEAR PC
                        ---EVERYTHING IS OFF
                        w_NEXT <= s_PROGRAM_INIT;

                when s_PROGRAM_INIT     =>

                        ---SEND SIGNAL TO INSTRUCTION MEMORY TO UPDATE ITSELF
                            BASED ON OFF SITE DATA
                        ---ALLOW FOR PC TO INCREMENT

                        if i_CB_PROGRAMING = '1' then
```

```
                w_NEXT <= s_PROGRAM_INIT;

        else

                w_NEXT <= s_PROGRAM_END;

        end if;

when s_PROGRAM_END      =>

        ---CLEAR PC
        ---SEND SIGNAL TO INSTRUCTION MEMORY TO STOP WRITING, AND
             START READING
        ---SEND SIGNAL TO INSTRUCTION REGISTER TO START READING

        w_NEXT <= s_FETCH;

when s_FETCH    =>

        ---LOAD w_OPERATOR AND w_OPERAND_ FOR NEXT STATE ,
            INCREASE PC

        w_NEXT <= s_DECODE;

when s_DECODE   =>

        ---DEPENDING ON OPCODE FROM w_OPERATOR , CHANGE STATES
            ACCORDINGLY

            if      i_CB_IR_DATA_IN( 15 downto 12 ) = "0000"
                then   ---LOAD FROM DATA_BANK INTO REGISTER

                    w_NEXT <= s_LOAD;

            elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0001"
                then     ---STORE FROM REGISTER INTO
                DATA_BANK

                    w_NEXT <= s_STORE;

            elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0010"
                then     ---TAKE FROM 2 REGISTERS AND ADD
                TEHM INTO ANOTHER REGISTER

                    w_NEXT <= s_ADD;

            elsif   i_CB_IR_DATA_IN( 15 downto 12 ) = "0011"
                then   ---LOAD IMMEDIATE INTO REGISTER

                    w_NEXT <= s_LOAD_IMMEDIATE;

            elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0100"
                then

                    w_NEXT <= s_CMP;

            elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0101"
                then

                    w_NEXT <= s_JE;

            elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0110"
```

```
                    then

                        w_NEXT <= s_SUB;

                elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "0111"
                    then

                        w_NEXT <= s_JLT;

                elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "1000"
                    then

                        w_NEXT <= s_JNE;

                elsif i_CB_IR_DATA_IN( 15 downto 12 ) = "1001"
                    then

                        w_NEXT <= s_JNLT;

                end if;

when s_LOAD     =>

        ---LOAD FROM DATA_BANK TO A REGISTER

        w_NEXT <= s_FETCH;

when s_STORE    =>

        ---STORE FROM REGISTER INTO DATA_BANK

        w_NEXT <= s_FETCH;

when s_LOAD_IMMEDIATE   =>

        ---TAKE FROM MUX AND STORE INTO A REGISTER

        w_NEXT <= s_FETCH;

when s_ADD      =>

        ---Operand to will have both register adresses

        w_NEXT <= s_WAIT;

when s_CMP      =>

        w_NEXT <= s_FETCH;

when s_JE       =>

        w_NEXT <= s_WAIT;

when s_SUB      =>

        w_NEXT <= s_WAIT;

when s_JLT      =>

        w_NEXT <= s_WAIT;

when s_JNE      =>
```

```
                w_NEXT <= s_WAIT;

        when s_JNLT     =>

                w_NEXT <= s_WAIT;

        when s_WAIT     =>

                w_NEXT <= s_FETCH;

        when others     =>

                w_NEXT <= s_INIT;

        end case;

end if; ---i_CB_CLK

case r_STATE is

when s_WAIT     =>


o_REGFLAG_WEN   <= '0' ;        ---Signal for Register file's Flag
    register to write from signals
o_REGFLAG_REN   <= '0' ;        ---Signal for Register file's Flag
    register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction register to
    load contents


---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to clear
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program Counter to
    jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for PC to
    jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory to load
    Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory to dum
    its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU"   ;       --- Input
    constant for multiplexer i_C
```

```
o_CB_MUX_SELECTOR       <= "UU"
            ;       --- Selector input for multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU"                                       ;
           --- Adress for writing to RF
o_CB_RF_WRITEENABLE     <= '0'                                          ;
           --- Enable for writing to RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from RF
o_CB_RF_READpENABLE     <= '0'                                          ;
           --- Enable for Readp from RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from RF
o_CB_RF_READqENABLE     <= '0'                                          ;
           --- Enable for Readq from RF

o_CB_ALU_CIN    <= 0                                                    ;
           --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU"                                       ;
           --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU"                           ;
    --- Data adress input for Data Bank
o_CB_DB_READENABLE      <= '0'                                          ;
           --- Read enable for Data Bank
o_CB_DB_WRITEENABLE     <= '0'                                          ;
           --- Write enable for Data Bank

---/BLOCO OPERACIONAL

        when    s_INIT                          =>

        ---CLEAR PC
        ---EVERYTHING IS OFF



        o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
            file's Flag register to write from signals
        o_REGFLAG_REN   <= '0' ;                ---Signal for Register
            file's Flag register to output its contents

        ---INSTRUCTION REGISTER

        o_CB_IR_LD      <= '0' ;        ---Signal for instruction
            register to load contents


        ---/INSTRUCTION REGISTER

        ---PROGRAM COUNTER

        o_CB_PC_CLR     <= '1' ;        ---Signal for Program Counter to
            clear
        o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
            increment
        o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
            Counter to jump
        o_CB_PC_ADDR_DIFF               <= "00000000";  ---Difference for
             PC to jump

        ---/PROGRAM COUNTER
```

```
---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '0' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when    s_PROGRAM_INIT          =>

        ---SEND SIGNAL TO INSTRUCTION MEMORY TO UPDATE ITSELF
            BASED ON OFF SITE DATA
        ---ALLOW FOR PC TO INCREMENT

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '0' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER
```

```
---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '1' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '1' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '0' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL


when    s_PROGRAM_END           =>

        ---CLEAR PC
        ---SEND SIGNAL TO INSTRUCTION MEMORY TO STOP WRITING, AND
             START READING
```

```
        ---SEND SIGNAL TO INSTRUCTION REGISTER TO START READING

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '0' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '1' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '1' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
```

```
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when    s_FETCH         =>

        ---LOAD w_OPERATOR AND w_OPERAND_ FOR NEXT STATE ,
            INCREASE PC

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '1' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '1' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '1' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
```

```
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when    s_DECODE                =>

        ---DEPENDING ON OPCODE FROM w_OPERATOR , CHANGE STATES
            ACCORDINGLY

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '1' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '1' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
```

```
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when    s_LOAD  =>

                                                ---LOAD FROM
                                                    DATA_BANK TO
                                                    A REGISTER

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '0' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY
```

```
o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "01" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= i_CB_IR_DATA_IN( 11 downto 8 ) ;
    --- Adress for writing to RF
o_CB_RF_WRITEENABLE     <= '1' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= i_CB_IR_DATA_IN( 7 downto 0 ) ;
    --- Data adress input for Data Bank
o_CB_DB_READENABLE      <= '1' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when    s_STORE =>

        ---STORE FROM REGISTER INTO DATA_BANK

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '0' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear
```

```
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "UU" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= i_CB_IR_DATA_IN( 11 downto 8 ) ;
    --- Adress for Readp from RF
o_CB_RF_READpENABLE     <= '1' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= i_CB_IR_DATA_IN( 7 downto 0 ) ;
    --- Data adress input for Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '1' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when    s_LOAD_IMMEDIATE        =>

                                                ---TAKE FROM MUX
                                                    AND STORE
                                                    INTO A
                                                    REGISTER

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '0' ;                ---Signal for Register
```

```
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '1' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "00000000" & i_CB_IR_DATA_IN( 7 downto
     0 ) ; --- Input constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= i_CB_IR_DATA_IN( 11 downto 8 ) ;
    --- Adress for writing to RF
o_CB_RF_WRITEENABLE     <= '1' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank
```

```
---/BLOCO OPERACIONAL


when    s_ADD   =>

        ---Operand to will have both register adresses

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '0' ;                ---Signal for Register
    file's Flag register to output its contents

        ---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
 o_CB_READ_DATA <= '1' ;        ---Signal for Instruction Memory
     to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "00" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= i_CB_IR_DATA_IN( 11 downto 8 ) ;
    --- Adress for writing to RF
o_CB_RF_WRITEENABLE     <= '1' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= i_CB_IR_DATA_IN( 7 downto 4 ) ;
    --- Adress for Readp from RF
o_CB_RF_READpENABLE     <= '1' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= i_CB_IR_DATA_IN( 3 downto 0 ) ;
    --- Adress for Readq from RF
```

```
o_CB_RF_READqENABLE     <= '1' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "0001" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when s_CMP      =>

        ---COMPARE TWO REGISTERS IN OPERAND TWO

o_REGFLAG_WEN   <= '1' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '1' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents


---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU"   ;       --- Input
     constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "UU"
                    ;       --- Selector input for multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU"
```

```
            ;       --- Adress for writing to RF
o_CB_RF_WRITEENABLE     <= '0'
            ;       --- Enable for writing to RF

o_CB_RF_READpADRESS     <= i_CB_IR_DATA_IN( 7 downto 4) ;
    --- Adress for Readp from RF
o_CB_RF_READpENABLE     <= '1'
            ;       --- Enable for Readp from RF

o_CB_RF_READqADRESS     <= i_CB_IR_DATA_IN( 3 downto 0) ;
    --- Adress for Readq from RF
o_CB_RF_READqENABLE     <= '1'
            ;       --- Enable for Readq from RF

o_CB_ALU_CIN    <= 0
            ;       --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU"
            ;       --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU"                           ;
          --- Data adress input for Data Bank
o_CB_DB_READENABLE      <= '0'
            ;       --- Read enable for Data Bank
o_CB_DB_WRITEENABLE     <= '0'
            ;       --- Write enable for Data Bank

---/BLOCO OPERACIONAL

when s_JE               =>

        ---SEND PC A SIGNAL TO JUMP IF EQ REGISTER IS 1

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '1' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents


---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear

if i_REGFLAG(1) = '1' then      ---IF EQ = '1' , jump

        o_CB_PC_INC     <= '0' ;        ---Signal for Program
            Counter to increment
        o_CB_PC_JUMP            <= '1' ;        ---Signal for
            Program Counter to jump
        o_CB_PC_ADDR_DIFF               <= i_CB_IR_DATA_IN( 7
            downto 0 );       ---Difference for PC to jump

else

        o_CB_PC_INC     <= '0' ;        ---Signal for Program
```

```
            Counter to increment
        o_CB_PC_JUMP            <= '0' ;        ---Signal for
            Program Counter to jump
        o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---
            Difference for PC to jump

end if;

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when s_SUB      =>

        ---Operand to will have both register adresses

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '0' ;                ---Signal for Register
    file's Flag register to output its contents
```

```
        ---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA          <= '1' ;        ---Signal for Instruction
     Memory to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "00" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= i_CB_IR_DATA_IN( 11 downto 8 ) ;
    --- Adress for writing to RF
o_CB_RF_WRITEENABLE     <= '1' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= i_CB_IR_DATA_IN( 7 downto 4 ) ;
    --- Adress for Readp from RF
o_CB_RF_READpENABLE     <= '1' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= i_CB_IR_DATA_IN( 3 downto 0 ) ;
    --- Adress for Readq from RF
o_CB_RF_READqENABLE     <= '1' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "0010" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL
```

```
when s_JLT      =>

        ---SEND PC A SIGNAL TO JUMP IF LT REGISTER IS 1

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '1' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents


---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear

if i_REGFLAG(0) = '1' then      ---IF LT = '1' , jump

        o_CB_PC_INC     <= '0' ;        ---Signal for Program
            Counter to increment
        o_CB_PC_JUMP            <= '1' ;        ---Signal for
            Program Counter to jump
        o_CB_PC_ADDR_DIFF               <= i_CB_IR_DATA_IN( 7
            downto 0 );       ---Difference for PC to jump

else

        o_CB_PC_INC     <= '0' ;        ---Signal for Program
            Counter to increment
        o_CB_PC_JUMP            <= '0' ;        ---Signal for
            Program Counter to jump
        o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---
            Difference for PC to jump

end if;

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
```

```
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when s_JNE      =>

        ---SEND PC A SIGNAL TO JUMP IF EQ REGISTER IS 0

o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '1' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents


---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear

if i_REGFLAG(1) = '0' then      ---IF EQ = '0' , jump

        o_CB_PC_INC     <= '0' ;        ---Signal for Program
            Counter to increment
        o_CB_PC_JUMP            <= '1' ;        ---Signal for
            Program Counter to jump
        o_CB_PC_ADDR_DIFF               <= i_CB_IR_DATA_IN( 7
            downto 0 );       ---Difference for PC to jump

else

        o_CB_PC_INC     <= '0' ;        ---Signal for Program
            Counter to increment
        o_CB_PC_JUMP            <= '0' ;        ---Signal for
```

```
            Program Counter to jump
        o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---
            Difference for PC to jump

end if;

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL

when s_JNLT     =>

        ---SEND PC A SIGNAL TO JUMP IF LT REGISTER IS 1


o_REGFLAG_WEN   <= '0' ;                ---Signal for Register
    file's Flag register to write from signals
o_REGFLAG_REN   <= '1' ;                ---Signal for Register
    file's Flag register to output its contents

---INSTRUCTION REGISTER
```

```
o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents


---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '0' ;        ---Signal for Program Counter to
    clear

if i_REGFLAG(0) = '0' then      ---IF LT = '0' , jump

        o_CB_PC_INC     <= '0' ;        ---Signal for Program
            Counter to increment
        o_CB_PC_JUMP            <= '1' ;        ---Signal for
            Program Counter to jump
        o_CB_PC_ADDR_DIFF               <= i_CB_IR_DATA_IN( 7
            downto 0 );       ---Difference for PC to jump

else

        o_CB_PC_INC     <= '0' ;        ---Signal for Program
            Counter to increment
        o_CB_PC_JUMP            <= '0' ;        ---Signal for
            Program Counter to jump
        o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---
            Difference for PC to jump

end if;

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '1' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
     RF

o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
     RF
o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
     RF

o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
     RF
```

```
o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
     RF

o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
     Data Bank
o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
    Bank
o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
     Bank

---/BLOCO OPERACIONAL



when others     =>

        ---CLEAR PC
        ---EVERYTHING IS OFF

---INSTRUCTION REGISTER

o_CB_IR_LD      <= '0' ;        ---Signal for instruction
    register to load contents

---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR     <= '1' ;        ---Signal for Program Counter to
    clear
o_CB_PC_INC     <= '0' ;        ---Signal for Program Counter to
    increment
o_CB_PC_JUMP            <= '0' ;        ---Signal for Program
    Counter to jump
o_CB_PC_ADDR_DIFF               <= "UUUUUUUU";  ---Difference for
     PC to jump

---/PROGRAM COUNTER

---INSTRUCTION MEMORY

o_CB_LOAD_DATA  <= '0' ;        ---Signal for Instruction Memory
    to load Data at addres from PC
o_CB_READ_DATA  <= '0' ;        ---Signal for Instruction Memory
    to dum its contents into Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT <= "UUUUUUUUUUUUUUUU" ; --- Input
    constant for multiplexer i_C
o_CB_MUX_SELECTOR       <= "10" ;       --- Selector input for
    multiplexer

o_CB_RF_WRITEADRESS     <= "UUUU" ;     --- Adress for writing to
     RF
o_CB_RF_WRITEENABLE     <= '0' ;        --- Enable for writing to
```

```
                    RF

                o_CB_RF_READpADRESS     <= "UUUU" ;     --- Adress for Readp from
                    RF
                o_CB_RF_READpENABLE     <= '0' ;        --- Enable for Readp from
                    RF

                o_CB_RF_READqADRESS     <= "UUUU" ;     --- Adress for Readq from
                    RF
                o_CB_RF_READqENABLE     <= '0' ;        --- Enable for Readq from
                    RF

                o_CB_ALU_CIN    <= 0 ;  --- Carry for ALU
                o_CB_ALU_SELECTOR       <= "UUUU" ;     --- Selector for ALU

                o_CB_DB_DATAADRESS      <= "UUUUUUUU" ; --- Data adress input for
                    Data Bank
                o_CB_DB_READENABLE      <= '0' ;        --- Read enable for Data
                   Bank
                o_CB_DB_WRITEENABLE     <= '0' ;        --- Write enable for Data
                    Bank

                ---/BLOCO OPERACIONAL

                end case;

                end process;

end ARCH_1;
```

3.4.3 Código do componente Unidade de Controle

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

entity CONTROL_UNIT is

port(

        i_CLK                   :       in std_logic    ;
        i_IM_PROGRAMING_SIGNAL  :       in std_logic    ;
        i_IM_DATA_TO_PROGRAM    :       in std_logic_vector( 15 downto 0 )      ;
                    ---COMMANDS IN

        o_OB_REGFLAG_WEN        :       out std_logic   ;       ---FLAG REGISTER
            WRITE ENABLE SIGNAL
        o_OB_REGFLAG_REN        :       out std_logic   ;       ---FLAG REGISTER
            READ ENABLE SIGNAL
        i_OB_REGFLAG            :       in std_logic_vector ( 1 downto 0 )      ;
                    ---FLAG REGISTER

        o_IMEDIATE              :       out std_logic_vector (15 downto 0)      ;
        o_MUX_SEL               :       out std_logic_vector ( 1 downto 0 )     ;

        o_WRITEADRESS           :       out std_logic_vector(3 downto 0)        ;
        o_WRITEENABLE           :       out std_logic   ;

        o_READpADRESS           :       out std_logic_vector(3 downto 0)        ;
        o_READpENABLE           :       out std_logic   ;

        o_READqADRESS           :       out std_logic_vector(3 downto 0)        ;
        o_READqENABLE           :       out std_logic   ;

        o_CIN                   :       out integer range 0 to 1        ;
        o_SELECTOR              :       out std_logic_vector( 3 downto 0 )      ;

        o_DATAADRESS            :       out std_logic_vector( 7 downto 0 )      ;
        o_READENABLE            :       out std_logic   ;
        o_DB_WRITEENABLE        :       out std_logic


);

end CONTROL_UNIT;

architecture ARCH_1 of CONTROL_UNIT is

component CONTROL_BLOCK is

        port(

        i_CB_CLK                :       in std_logic                            ;
        ---INSTRUCTION REGISTER

        i_CB_IR_DATA_IN         :       in std_logic_vector( 15 downto 0 )      ;
                    ---Instructions from instruction register
        o_CB_IR_LD              :       out std_logic                           ;
                    ---Signal for instruction register to load contents
```

```
---/INSTRUCTION REGISTER

---PROGRAM COUNTER

o_CB_PC_CLR                     :       out std_logic                   ;
            ---Signal for Program Counter to clear
o_CB_PC_INC                     :       out std_logic                   ;
            ---Signal for Program Counter to increment
o_CB_PC_JUMP                    :       out std_logic                   ;
            ---Signal for program counter to jump
o_CB_PC_ADDR_DIFF               :       out std_logic_vector ( 7 downto 0
     )      ;       ---Difference that PC has to account for, use for
    conditional jumps


---/PROGRAM COUNTER

---INSTRUCTION MEMORY

i_CB_PROGRAMING                 :       in std_logic                    ;
            ---If 1 , programing
o_CB_LOAD_DATA                  :       out std_logic                   ;
            ---Signal for Instruction Memory to load Data at addres from
    PC
o_CB_READ_DATA                  :       out std_logic                   ;
            ---Signal for Instruction Memory to dum its contents into
    Insruction Register

---/INSTRUCTION MEMORY

---BLOCO OPERACIONAL

o_CB_MUX_INPUT_CONSTANT         :       out std_logic_vector(15 downto 0)
    ;       --- Input constant for multiplexer i_C
o_CB_MUX_SELECTOR               :       out std_logic_vector ( 1 downto 0
     )      ;       --- Selector input for multiplexer

o_CB_RF_WRITEADRESS             :       out std_logic_vector(3 downto 0);
            --- Adress for writing to RF
o_CB_RF_WRITEENABLE             :       out std_logic                   ;
            --- Enable for writing to RF

o_CB_RF_READpADRESS             :       out std_logic_vector(3 downto 0);
            --- Adress for Readp from RF
o_CB_RF_READpENABLE             :       out std_logic                   ;
            --- Enable for Readp from RF

o_CB_RF_READqADRESS             :       out std_logic_vector(3 downto 0);
            --- Adress for Readq from RF
o_CB_RF_READqENABLE             :       out std_logic                   ;
            --- Enable for Readq from RF

o_CB_ALU_CIN                    :       out integer range 0 to 1        ;
            --- Carry for ALU
o_CB_ALU_SELECTOR               :       out std_logic_vector( 3 downto 0
    );      --- Selector for ALU

o_CB_DB_DATAADRESS              :       out std_logic_vector( 7 downto 0
    )       ;       --- Data adress input for Data Bank
o_CB_DB_READENABLE              :       out std_logic                   ;
            --- Read enable for Data Bank
o_CB_DB_WRITEENABLE             :       out std_logic                   ;
```

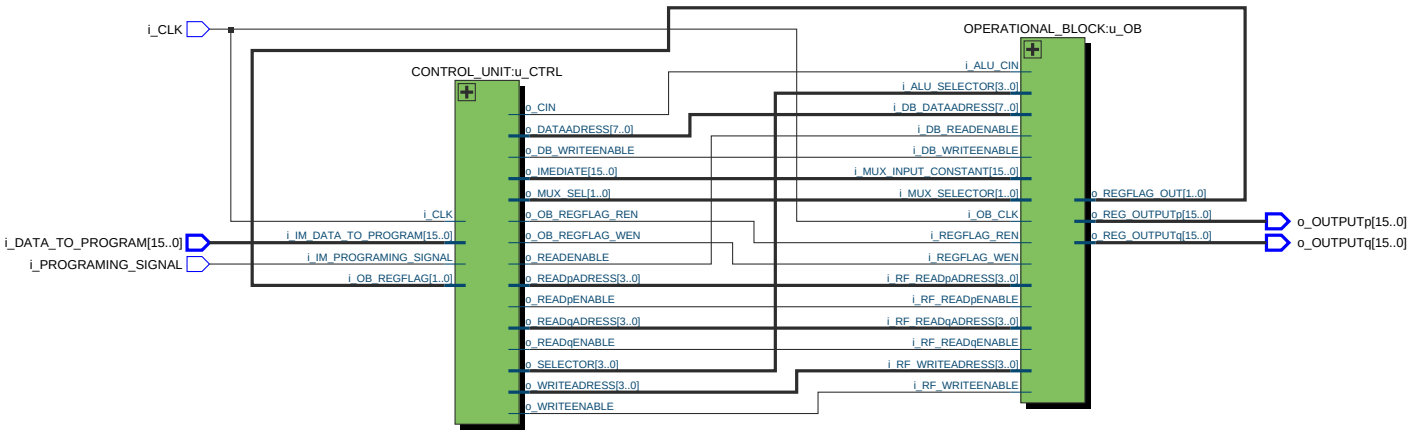
```
                --- Write enable for Data Bank

        o_REGFLAG_WEN                   :       out std_logic                   ;
                --- Write enable for comand block's flag register
        o_REGFLAG_REN                   :       out std_logic                   ;
                --- Read enable for comand block's flag register
        i_REGFLAG                       :       in std_logic_vector ( 1 downto 0
            )               --- Comand Block's flag register

        ---/BLOCO OPERACIONAL

);

end component;

component PROGRAM_COUNTER is

        port(

                i_PC_CLK        : in std_logic;
                i_PC_CLR        : in std_logic;
                i_PC_INC        : in std_logic;

                i_PC_JUMP       : in std_logic;
                i_PC_OFFSET     : in std_logic_vector ( 7 downto 0 );

                o_PC_OUT        : out std_logic_vector ( 7 downto 0 )

        );

end component;

component INSTRUCTION_REGISTER is

        port(

                i_IR_CLK        :       in std_logic;
                i_IR_DATA       :       in std_logic_vector( 15 downto 0 );
                i_IR_LOAD       :       in std_logic;

                o_IR_OUTPUT     :       out std_logic_vector( 15 downto 0 )

        );

end component;

component INSTRUCTION_MEMORY is

        port(

                i_IM_CLK        :       in std_logic;

                i_IM_ADDR       :       in std_logic_vector( 7 downto 0 );
                    ---MEMORY TAKEN DOWN TO 8 BITS, or 255 possibilities

                i_IM_RD         :       in std_logic;
                                                ---ENABLE FOR OUTPUT

                i_IM_DATA_LOAD  :       in std_logic;
                                                ---ENABLE FOR COMMAND INPUT

                i_IM_DATAIN     :       in std_logic_vector( 15 downto 0 );
```

```
                ---COMMANDS IN

                o_IM_DATA       :       out std_logic_vector( 15 downto 0 )

        );

end component;


        signal  w_INST_MEM_READ_ENABLE  :       std_logic;
        signal  w_INST_MEM_PROGRAM      :       std_logic;
        signal  w_INST_MEM_OUTPUT       :       std_logic_vector( 15 downto 0 );

        signal  w_INST_REG_LOADSIG      :       std_logic;
        signal  w_INST_REG_OUTPUT       :       std_logic_vector( 15 downto 0 );

        signal  w_PC_CLEAR              :       std_logic;
        signal  w_PC_INCREMENT          :       std_logic;
        signal  w_PROGRAM_COUNTER_ADRESS:       std_logic_vector ( 7 downto 0 );

        signal  w_CB_PC_JUMP            :       std_logic       ;
        signal  w_PC_CB_ADDR_DIFFERENCE :       std_logic_vector ( 7 downto 0 );

        begin

        u_IM    :       INSTRUCTION_MEMORY port map(

                i_IM_CLK        => i_CLK        ,
                i_IM_ADDR       => w_PROGRAM_COUNTER_ADRESS     ,
                i_IM_RD         => w_INST_MEM_READ_ENABLE       ,
                i_IM_DATA_LOAD  => w_INST_MEM_PROGRAM   ,
                i_IM_DATAIN     => i_IM_DATA_TO_PROGRAM ,
                o_IM_DATA       => w_INST_MEM_OUTPUT

                );

        u_IR    :       INSTRUCTION_REGISTER port map   (

                i_IR_CLK        =>      i_CLK   ,
                i_IR_DATA       =>      w_INST_MEM_OUTPUT       ,
                i_IR_LOAD       =>      w_INST_REG_LOADSIG      ,
                o_IR_OUTPUT     =>      w_INST_REG_OUTPUT

                );

        u_PC    :       PROGRAM_COUNTER port map(

                i_PC_CLK                =>      i_CLK   ,
                i_PC_CLR                => w_PC_CLEAR           ,
                i_PC_INC                =>      w_PC_INCREMENT  ,
                i_PC_JUMP       =>      w_CB_PC_JUMP    ,
                i_PC_OFFSET     =>      w_PC_CB_ADDR_DIFFERENCE ,
                o_PC_OUT                =>      w_PROGRAM_COUNTER_ADRESS

                                                                                )
                                                                                    ;


        u_CB    :       CONTROL_BLOCK port map(

                i_CB_CLK        =>      i_CLK   ,
```

```
                ---INSTRUCTION REGISTER

                i_CB_IR_DATA_IN =>      w_INST_REG_OUTPUT       ,
                o_CB_IR_LD      =>      w_INST_REG_LOADSIG      ,

                ---/INSTRUCTION REGISTER

                ---PROGRAM COUNTER

                o_CB_PC_CLR     =>      w_PC_CLEAR      ,
                o_CB_PC_INC     =>      w_PC_INCREMENT  ,
                o_CB_PC_JUMP            =>      w_CB_PC_JUMP    ,
                o_CB_PC_ADDR_DIFF               =>      w_PC_CB_ADDR_DIFFERENCE ,

                ---/PROGRAM COUNTER

                ---INSTRUCTION MEMORY

                i_CB_PROGRAMING =>      i_IM_PROGRAMING_SIGNAL  ,
                o_CB_LOAD_DATA  =>      w_INST_MEM_PROGRAM              ,
                o_CB_READ_DATA  =>      w_INST_MEM_READ_ENABLE  ,

                ---/INSTRUCTION MEMORY

                ---BLOCO OPERACIONAL

                o_CB_MUX_INPUT_CONSTANT =>      o_IMEDIATE      ,
                o_CB_MUX_SELECTOR       =>      o_MUX_SEL       ,

                o_CB_RF_WRITEADRESS     =>      o_WRITEADRESS   ,
                o_CB_RF_WRITEENABLE     =>      o_WRITEENABLE   ,

                o_CB_RF_READpADRESS     =>      o_READpADRESS   ,
                o_CB_RF_READpENABLE     =>      o_READpENABLE   ,

                o_CB_RF_READqADRESS     =>      o_READqADRESS   ,
                o_CB_RF_READqENABLE     =>      o_READqENABLE   ,

                o_CB_ALU_CIN            =>      o_CIN   ,
                o_CB_ALU_SELECTOR       =>      o_SELECTOR      ,

                o_CB_DB_DATAADRESS      =>      o_DATAADRESS            ,
                o_CB_DB_READENABLE      =>      o_READENABLE            ,
                o_CB_DB_WRITEENABLE     =>      o_DB_WRITEENABLE        ,

                o_REGFLAG_WEN   =>      o_OB_REGFLAG_WEN        ,
                o_REGFLAG_REN   =>      o_OB_REGFLAG_REN                ,
                i_REGFLAG       =>      i_OB_REGFLAG

                --/BLOCO OPERACIONAL

        );

end ARCH_1;
```

4 TOP

O componente de topo liga os dois componentes principais e permite a entrada e saída de valores.



4.1 Código do componente TOP

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

entity TOP is

port(

        i_CLK                   :       in std_logic    ;
        i_PROGRAMING_SIGNAL     :       in std_logic    ;
        i_DATA_TO_PROGRAM       :       in std_logic_vector( 15 downto 0 )      ;
                    ---COMMANDS IN

        o_OUTPUTp               :       out std_logic_vector (15 downto 0)      ;
                    ---Data output from reg_file p
        o_OUTPUTq               :       out std_logic_vector (15 downto 0)
                     ---Data output from reg_file q

);

end TOP;

architecture ARCH_1 of TOP is

component CONTROL_UNIT is

        port(

                i_CLK                   :       in std_logic    ;
                i_IM_PROGRAMING_SIGNAL  :       in std_logic    ;
                i_IM_DATA_TO_PROGRAM    :       in std_logic_vector( 15 downto 0
                    )       ;       ---COMMANDS IN

                o_OB_REGFLAG_WEN        :       out std_logic   ;       ---FLAG
                    REGISTER WRITE ENABLE SIGNAL
                o_OB_REGFLAG_REN        :       out std_logic   ;       ---FLAG
                    REGISTER READ ENABLE SIGNAL
                i_OB_REGFLAG            :       in std_logic_vector ( 1 downto 0
                    )       ;       ---FLAG REGISTER

                o_IMEDIATE              :       out std_logic_vector (15 downto
                    0)      ;
                o_MUX_SEL               :       out std_logic_vector ( 1 downto 0
                     )      ;

                o_WRITEADRESS           :       out std_logic_vector(3 downto 0)
                            ;
                o_WRITEENABLE           :       out std_logic   ;

                o_READpADRESS           :       out std_logic_vector(3 downto 0)
                            ;
                o_READpENABLE           :       out std_logic   ;

                o_READqADRESS           :       out std_logic_vector(3 downto 0)
                            ;
                o_READqENABLE           :       out std_logic   ;

                o_CIN                   :       out integer range 0 to 1        ;
```

```
                o_SELECTOR              :       out std_logic_vector( 3 downto 0
                    )       ;

                o_DATAADRESS            :       out std_logic_vector( 7 downto 0
                    )       ;
                o_READENABLE            :       out std_logic   ;
                o_DB_WRITEENABLE        :       out std_logic


        );

end component;

component OPERATIONAL_BLOCK is

        port(

                i_OB_CLK                : in std_logic  ;

                i_REGFLAG_WEN           :       in std_logic    ;       ---
                    REGISTER FLAG WRITE ENABLE
                i_REGFLAG_REN           :       in std_logic    ;       ---
                    REGISTER FLAG READ ENABLE
                o_REGFLAG_OUT           :       out std_logic_vector    ;
                    ---REGISTER FLAG OUT

                i_MUX_INPUT_CONSTANT    :       in std_logic_vector (15 downto 0)
                            ;       --- Input constant for multiplexer i_C
                i_MUX_SELECTOR          :       in std_logic_vector ( 1 downto 0
                    )       ;       --- Selector input for multiplexer

                i_RF_WRITEADRESS        :       in std_logic_vector(3 downto 0) ;
                            --- Adress for writing to RF
                i_RF_WRITEENABLE        :       in std_logic    ;       ---
                    Enable for writing to RF

                i_RF_READpADRESS        :       in std_logic_vector(3 downto 0) ;
                            --- Adress for Readp from RF
                i_RF_READpENABLE        :       in std_logic    ;       ---
                    Enable for Readp from RF

                i_RF_READqADRESS        :       in std_logic_vector(3 downto 0) ;
                            --- Adress for Readq from RF
                i_RF_READqENABLE        :       in std_logic    ;       ---
                    Enable for Readq from RF

                i_ALU_CIN               :       in integer range 0 to 1 ;
                    --- Carry for ALU
                i_ALU_SELECTOR          :       in std_logic_vector( 3 downto 0 )
                            ;       --- Selector for ALU

                i_DB_DATAADRESS         :       in std_logic_vector( 7 downto 0 )
                            ;       --- Data adress input for Data Bank
                i_DB_READENABLE         :       in std_logic    ;       --- Read
                    enable for Data Bank
                i_DB_WRITEENABLE        :       in std_logic    ;       --- Write
                     enable for Data Bank

                o_REG_OUTPUTp           :       out std_logic_vector (15 downto
                    0)      ;       ---Data output from reg_file p
                o_REG_OUTPUTq           :       out std_logic_vector (15 downto
                    0)              ---Data output from reg_file q
```

```
        );

end component;

        signal w_IMEDIATE       :       std_logic_vector (15 downto 0)  ;
        signal w_MUX_SEL        :       std_logic_vector ( 1 downto 0 ) ;
        signal w_WRITEADRESS    :       std_logic_vector(3 downto 0)    ;
        signal w_WRITEENABLE    :       std_logic       ;
        signal w_READpADRESS    :       std_logic_vector(3 downto 0)    ;
        signal w_READpENABLE    :       std_logic       ;
        signal w_READqADRESS    :       std_logic_vector(3 downto 0)    ;
        signal w_READqENABLE    :       std_logic       ;
        signal w_CIN            :       integer range 0 to 1    ;
        signal w_SELECTOR       :       std_logic_vector( 3 downto 0 )  ;
        signal w_DATAADRESS     :       std_logic_vector( 7 downto 0 )  ;
        signal w_READENABLE     :       std_logic       ;
        signal w_DB_WRITEENABLE :       std_logic       ;

        signal w_FLAG_REGIST_WRITEEN    :       std_logic       ;
        signal w_FLAG_REGIST_READEN     :       std_logic       ;
        signal w_FLAG_REGIST    :       std_logic_vector( 1 downto 0 )  ;

        begin

        u_CTRL  :       CONTROL_UNIT port map(

                i_CLK                   =>      i_CLK   ,
                i_IM_PROGRAMING_SIGNAL  =>      i_PROGRAMING_SIGNAL     ,
                i_IM_DATA_TO_PROGRAM    =>      i_DATA_TO_PROGRAM       ,

                o_OB_REGFLAG_WEN        =>      w_FLAG_REGIST_WRITEEN   ,
                o_OB_REGFLAG_REN        =>      w_FLAG_REGIST_READEN    ,
                i_OB_REGFLAG            =>      w_FLAG_REGIST           ,

                o_IMEDIATE              =>      w_IMEDIATE      ,
                o_MUX_SEL               =>      w_MUX_SEL       ,

                o_WRITEADRESS           =>      w_WRITEADRESS   ,
                o_WRITEENABLE           =>      w_WRITEENABLE   ,

                o_READpADRESS           =>      w_READpADRESS   ,
                o_READpENABLE           =>      w_READpENABLE   ,

                o_READqADRESS           =>      w_READqADRESS   ,
                o_READqENABLE           =>      w_READqENABLE   ,

                o_CIN                           =>      w_CIN   ,
                o_SELECTOR              =>      w_SELECTOR      ,

                o_DATAADRESS            =>      w_DATAADRESS    ,
                o_READENABLE            =>      w_READENABLE    ,
                o_DB_WRITEENABLE        =>      w_DB_WRITEENABLE

                );

        u_OB    :       OPERATIONAL_BLOCK port map      (

                i_OB_CLK                =>      i_CLK   ,

                i_REGFLAG_WEN           =>      w_FLAG_REGIST_WRITEEN   ,
```

```
                i_REGFLAG_REN           =>      w_FLAG_REGIST_READEN    ,
                o_REGFLAG_OUT           =>      w_FLAG_REGIST   ,

                i_MUX_INPUT_CONSTANT    =>      w_IMEDIATE      ,
                i_MUX_SELECTOR          =>      w_MUX_SEL       ,

                i_RF_WRITEADRESS        =>      w_WRITEADRESS   ,
                i_RF_WRITEENABLE        =>      w_WRITEENABLE   ,

                i_RF_READpADRESS        =>      w_READpADRESS   ,
                i_RF_READpENABLE        =>      w_READpENABLE   ,

                i_RF_READqADRESS        =>      w_READqADRESS   ,
                i_RF_READqENABLE        =>      w_READqENABLE   ,

                i_ALU_CIN               =>      w_CIN   ,
                i_ALU_SELECTOR          =>      w_SELECTOR      ,

                i_DB_DATAADRESS         =>      w_DATAADRESS    ,
                i_DB_READENABLE         =>      w_READENABLE    ,
                i_DB_WRITEENABLE        =>      w_DB_WRITEENABLE        ,

                o_REG_OUTPUTp           =>      o_OUTPUTp       ,
                o_REG_OUTPUTq           =>      o_OUTPUTq

        );

end ARCH_1;
```

4.2 TestBench do Componente TOP e algorítimo de Mínimo Divisor Comum

O TestBench foi feito normalmente com um CLOCK de 200 ps e o algorítimo foi criado. A cada pulso do relógio, com exceção do primeiro que está no estado inicial, carrega o código de máquina nos pinos de entrada, junto com o sinal de programação para que o computador saiba que o próximo pulso não comece o algorítimo prematuramente.

O algorítimo guarda nos dois primeiros registradores os valores a serem usados, e por ultimo guarda o resultado do primeiro registrador no primeiro endereço do Banco de Dados.

```
            ---ASSEMBLY:

            ---     REG[0] = 00000010
            ---     REG[1] = 00000100
            ---     CMP REG[0] , REG[1]
            ---     JEQ END
            ---LOOPHEAD:
            ---     JLT XLTY
            ---     REG[0] = REG[0] - REG[1]
            ---     JNLT LOOPBOTTOM
            ---XLTY:
            ---     REG[1] = REG[1] - REG[0]
            ---LOOPBOTTOM:
            ---     CMP REG[0] , REG[1]
            ---     JNEQ LOOPHEAD
            ---END:
            ---     DB[0] = REG[0]

w_PROGRAMING_SIGNAL     <= 'U' ;
w_DATA_TO_PROGRAM       <= "UUUUUUUUUUUUUUUU" ;wait for c_CLK_PERIOD;

w_PROGRAMING_SIGNAL     <= '1' ;

w_DATA_TO_PROGRAM       <= "0011000000011011" ;wait for c_CLK_PERIOD;
            ---REG[0] = 00011011 27
w_DATA_TO_PROGRAM       <= "0011000000011011" ;wait for c_CLK_PERIOD;
            ---REG[0] = 00011011 27
w_DATA_TO_PROGRAM       <= "0011000100101101" ;wait for c_CLK_PERIOD;
            ---REG[1] = 00101101 45 = 9

w_DATA_TO_PROGRAM       <= "0100000000000001" ;wait for c_CLK_PERIOD;
            ---CMP REG[0] , REG[1]
w_DATA_TO_PROGRAM       <= "0101000000000110" ;wait for c_CLK_PERIOD;
            ---JEQ END      *6*
        ---LOOPHEAD:
w_DATA_TO_PROGRAM       <= "0111000000000010" ;wait for c_CLK_PERIOD;
            ---JLT XLTY *2*
w_DATA_TO_PROGRAM       <= "0110000000000001" ;wait for c_CLK_PERIOD;
            ---REG[0] = REG[0] - REG[1]
w_DATA_TO_PROGRAM       <= "1001000000000001" ;wait for c_CLK_PERIOD;
            ---JNLT LOOPBOTTOM *1*
        ---XLTY:
w_DATA_TO_PROGRAM       <= "0110000100010000" ;wait for c_CLK_PERIOD;
            ---REG[1] = REG[1] - REG[0]
        ---LOOPBOTTOM:
w_DATA_TO_PROGRAM       <= "0100000000000001" ;wait for c_CLK_PERIOD;
            ---CMP REG[0] , REG[1]
w_DATA_TO_PROGRAM       <= "1000000011111010" ;wait for c_CLK_PERIOD;
```

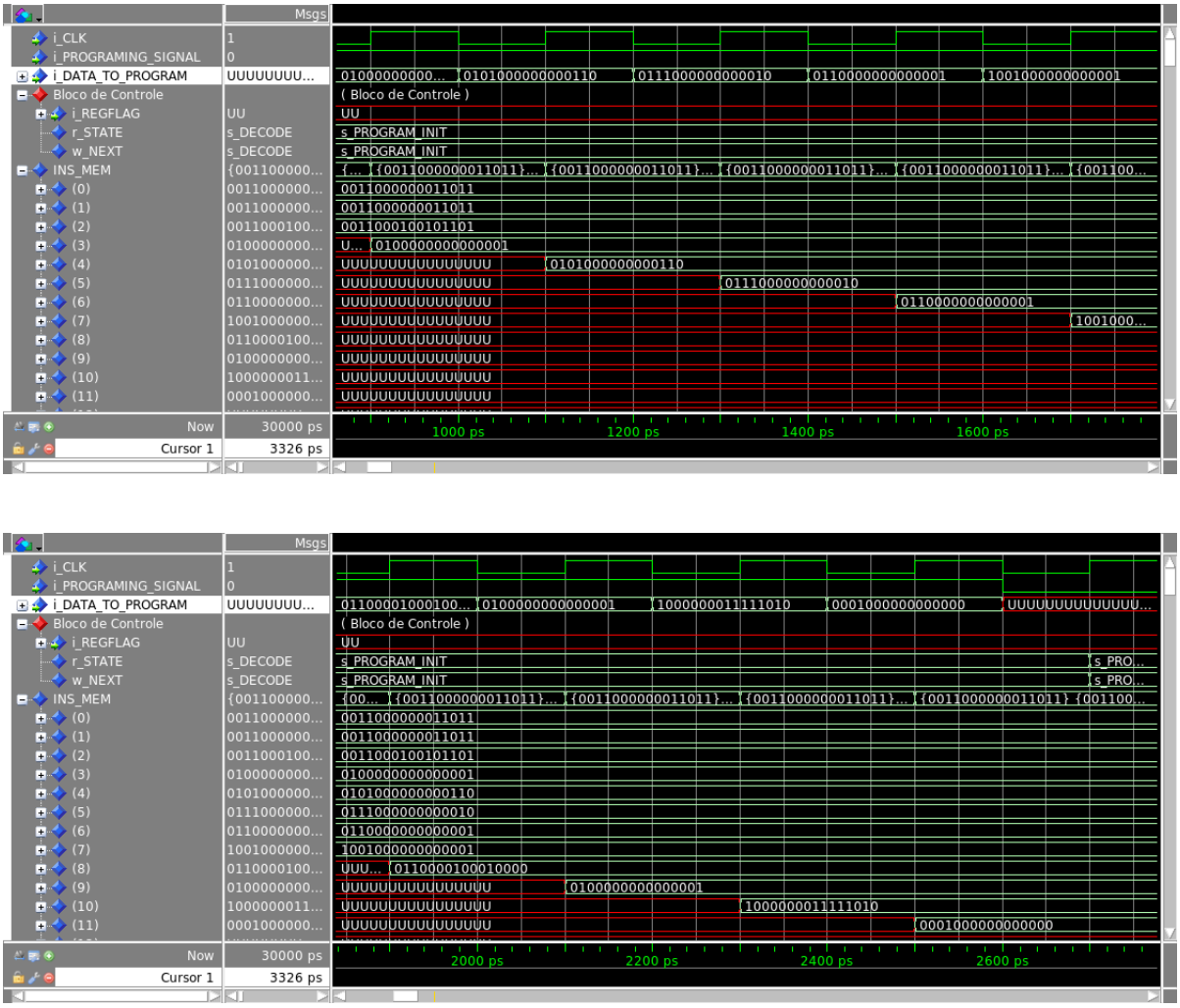
```
                ---JNEQ LOOPHEAD *-6*
            ---END:
    w_DATA_TO_PROGRAM       <= "0001000000000000" ;wait for c_CLK_PERIOD;
                ---DB[0] = REG[0]
```

Note como as duas primeiras instruções são repetidas, isso seria porque sempre que um valor é colocado na primeira instrução, a instrução é mal formada, o motivo disso não foi encontrado e acontece apenas na primeira instrução se for um carregamento de um imediado.

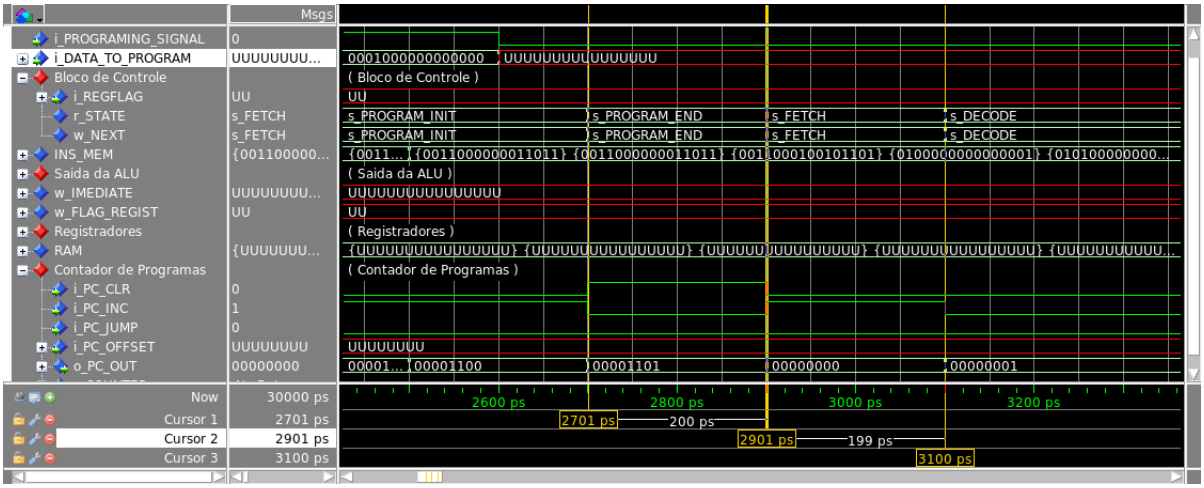
Os dois valores sendo testados são vinte e sete e quarenta e cinco, cujo mínimo divisor comum é nove.

27	45	3
9	15	3
3	5	3
1	5	5
1	1	1
3*3 = 9		

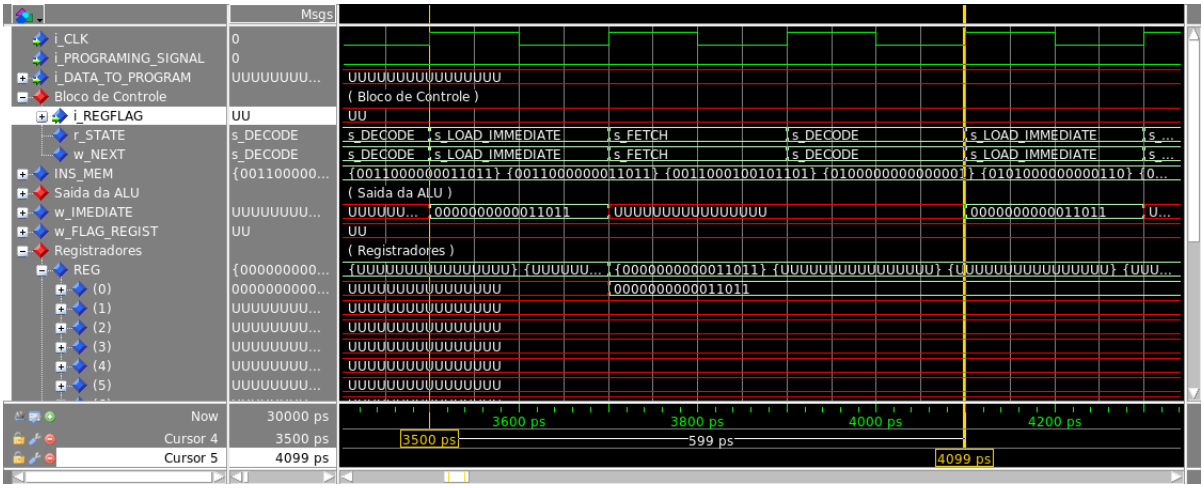
Note as duas primeiras instruções iguais.



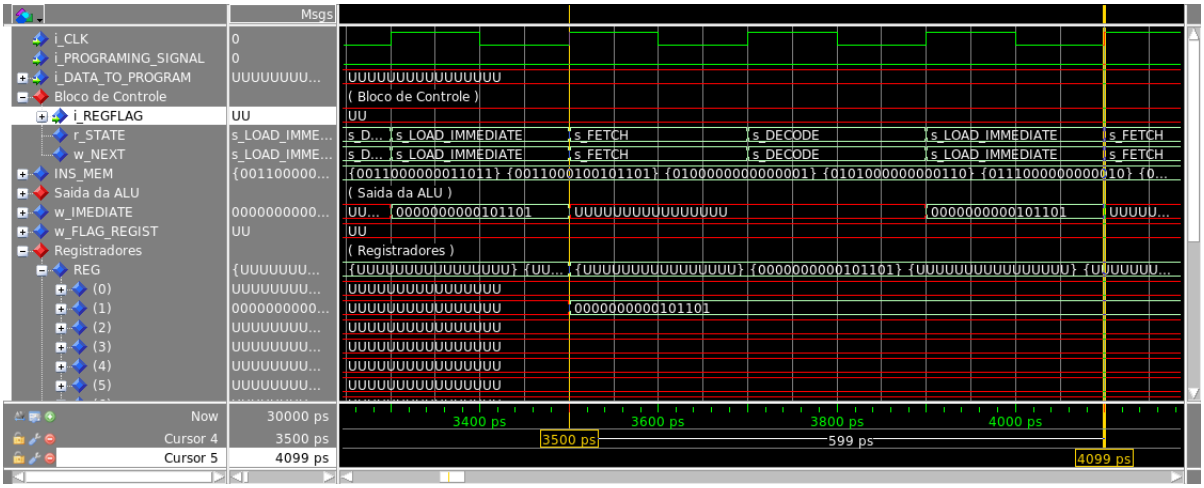
Todas as instruções foram carregadas, note o sinal de programação abaixando.



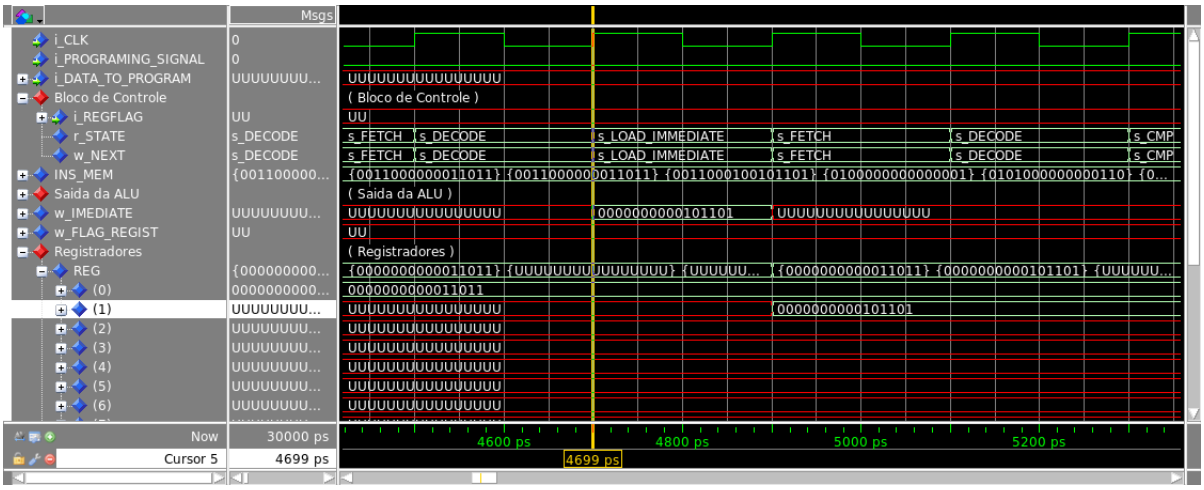
Assim que o algorítimo terminou de ser gravado o contador foi zerado e o estado seguinte foi a busca, que incrementa o contador. Por isso foi necessário o estado de fim de programação, pois caso contrario o contador nunca seria zerado.



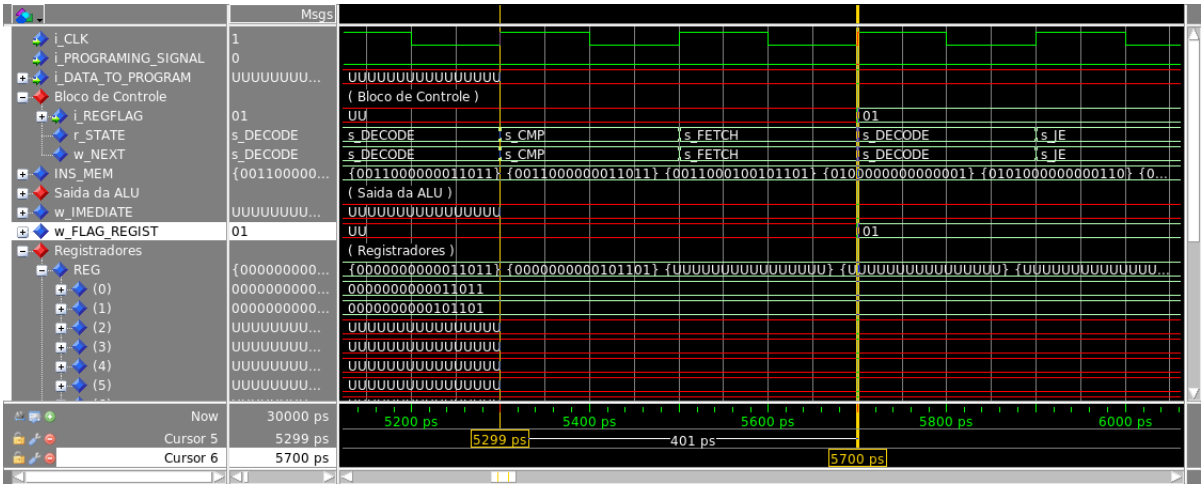
Quando a instrução de carregamento de imediato é a primeira instrução e é repetida duas vezes, a primeira vez que é executada ela funciona corretamente.



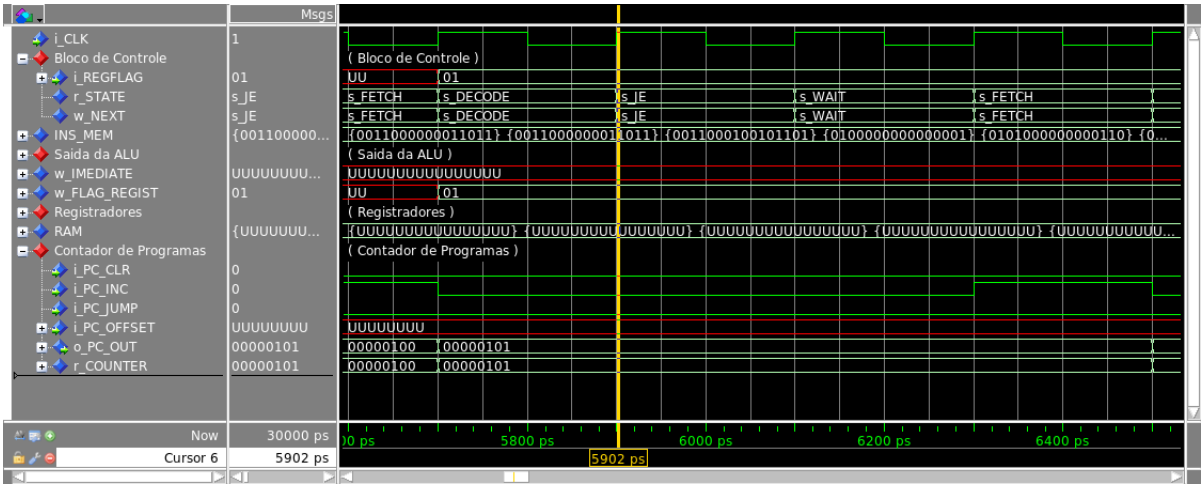
Porem quando não repetida, ela carrega a próxima instrução, que nesse caso seria carregar o segundo valor. Por algum motivo o sinal de carregamento de instruções fica alto depois do estado de fim de programação, Carregando assim os valores da instrução seguinte quando não deveria.



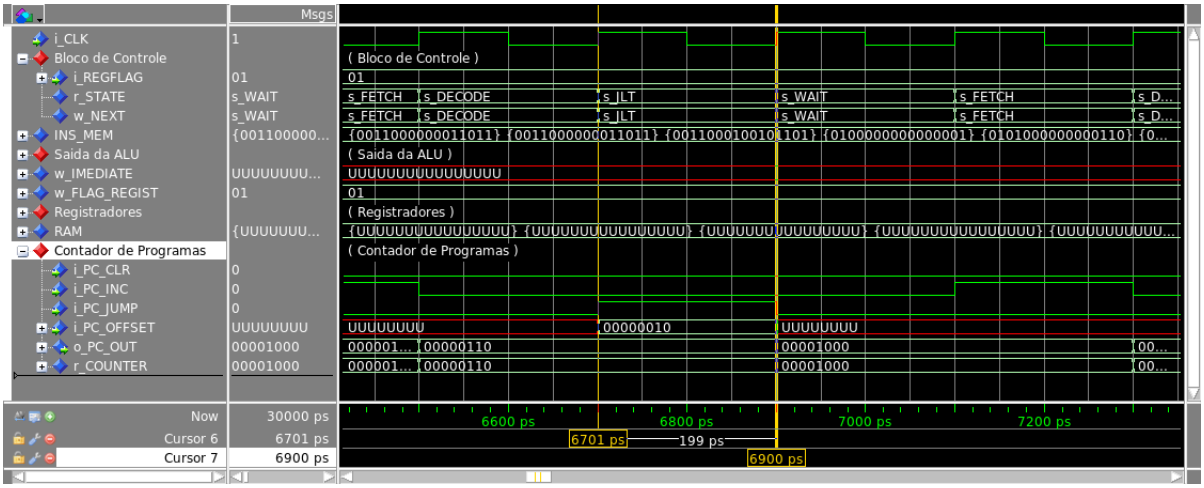
O segundo valor é guardado no registrador.



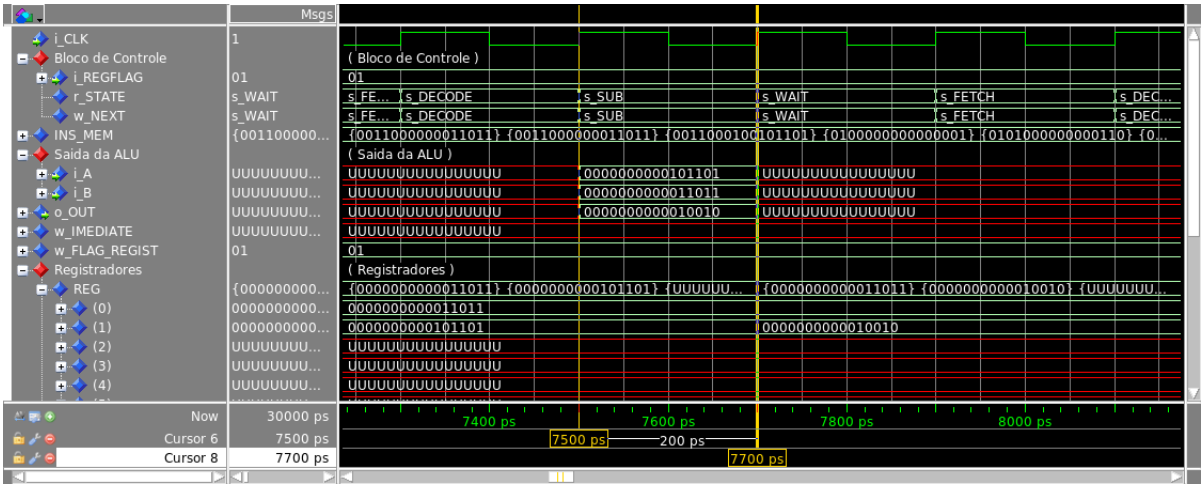
Os valores dos dois registradores são comparados e o registrador de FLAGS atualizado.



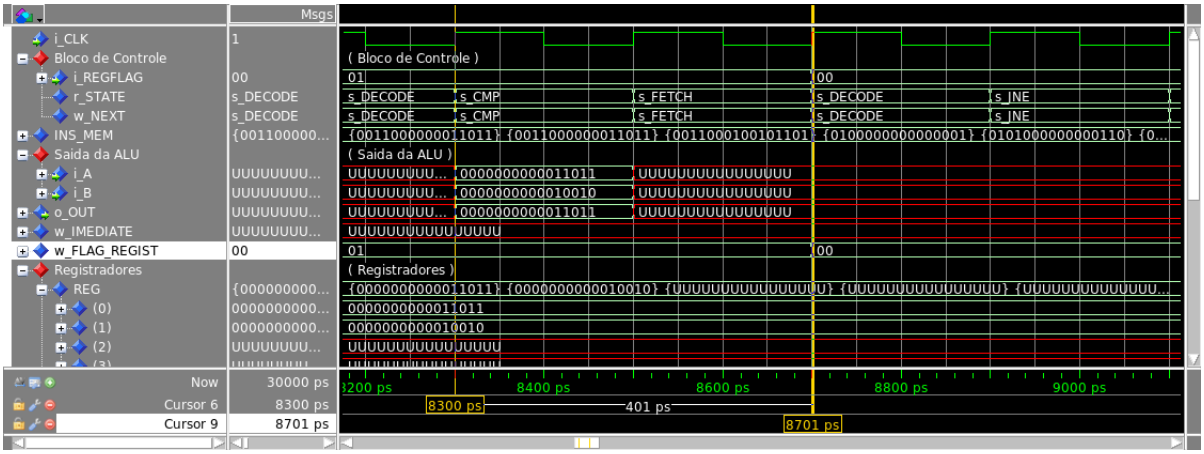
É testado a possibilidade de igualdade e o pulo não é feito pois o registrador indica falso.



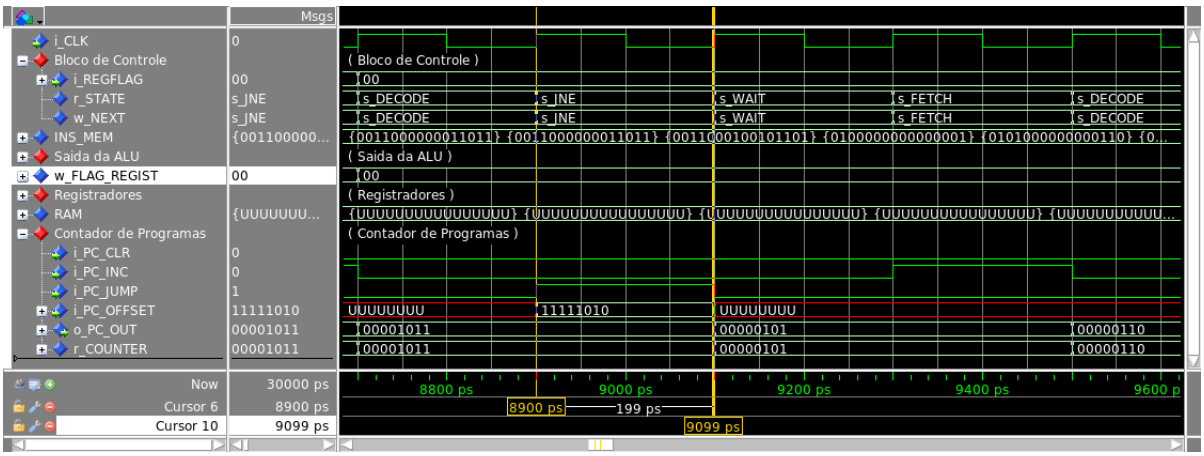
O algorítimo continua e testa maioridade, o registrador indica verdadeiro, portanto um pulo acontece, incrementando o contador por dois.



O algorítimo então subtrai dois valores e os guarda em um registrador.

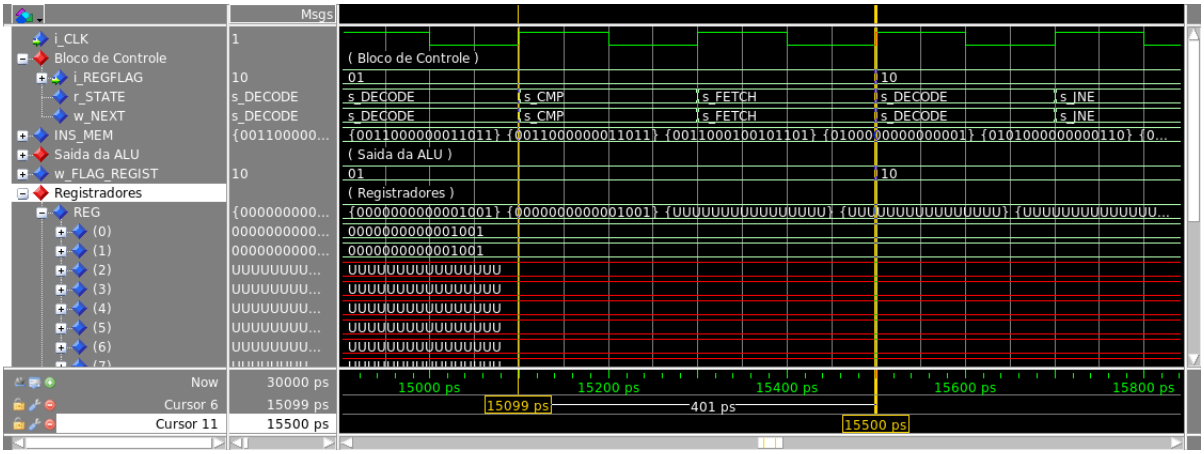


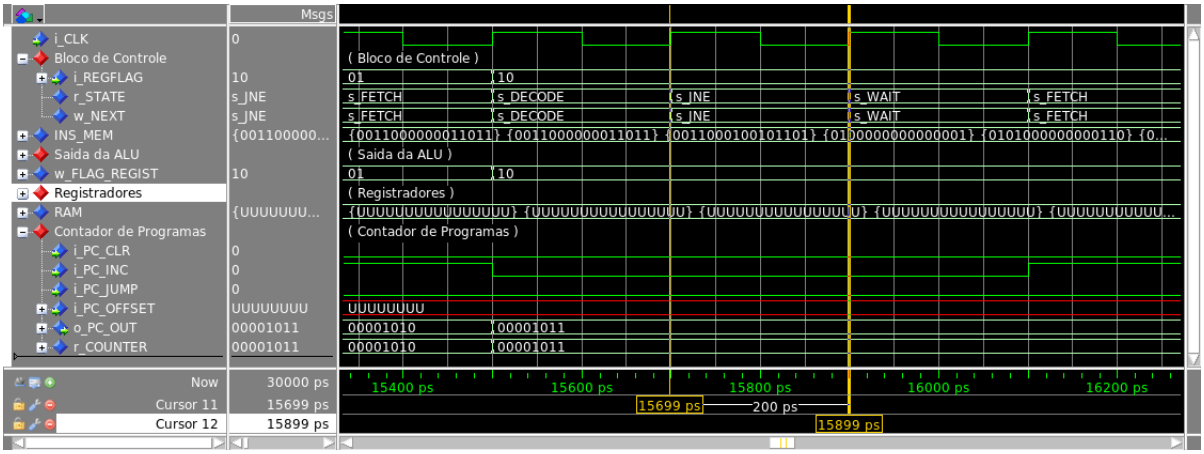
Os registradores são novamente comparados e o registrador de FLAGS é atualizado para o próximo pulo.



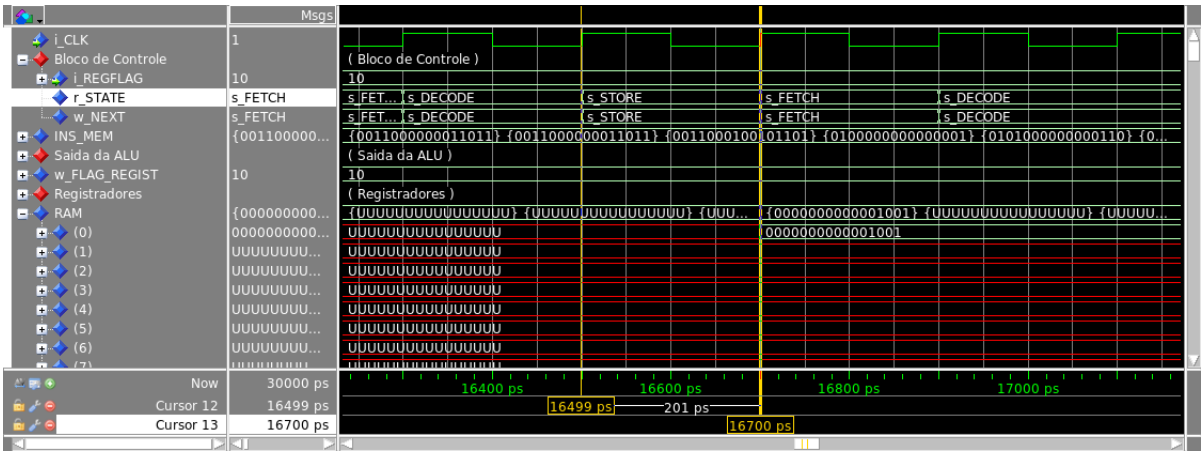
É testado a igualdade e um pulo de -6 é feito.

O algorítimo se repete múltiplas vezes em um LOOP até que os valores nos dois registradores forem iguais.





Os registradores são iguais, o algorítimo então vai guardar o que está no primeiro registrador e parar.

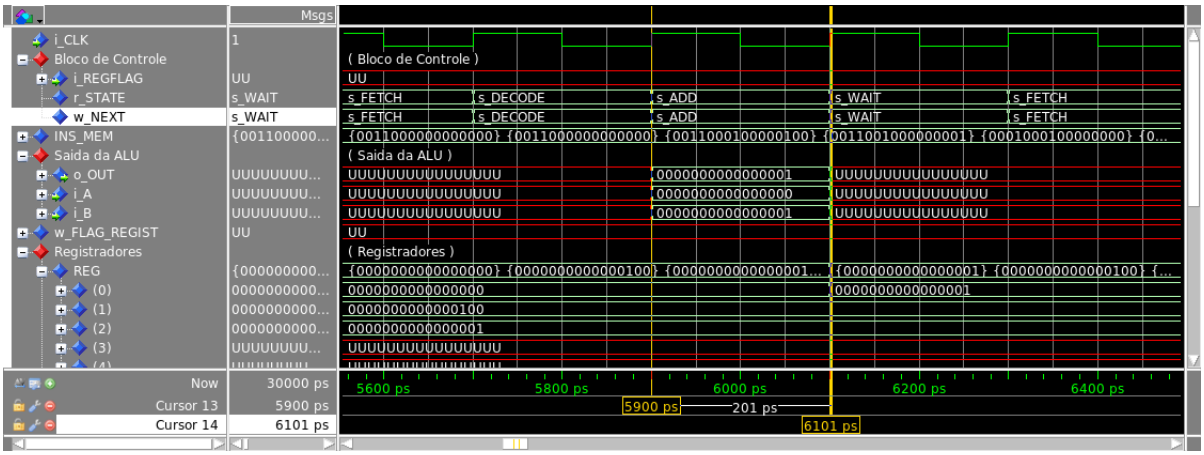


O resultado final é guardado, de agora em diante o processador dicará travado no estado de decodificação até que o contador receba um `overflow`, começando tudo novamente.

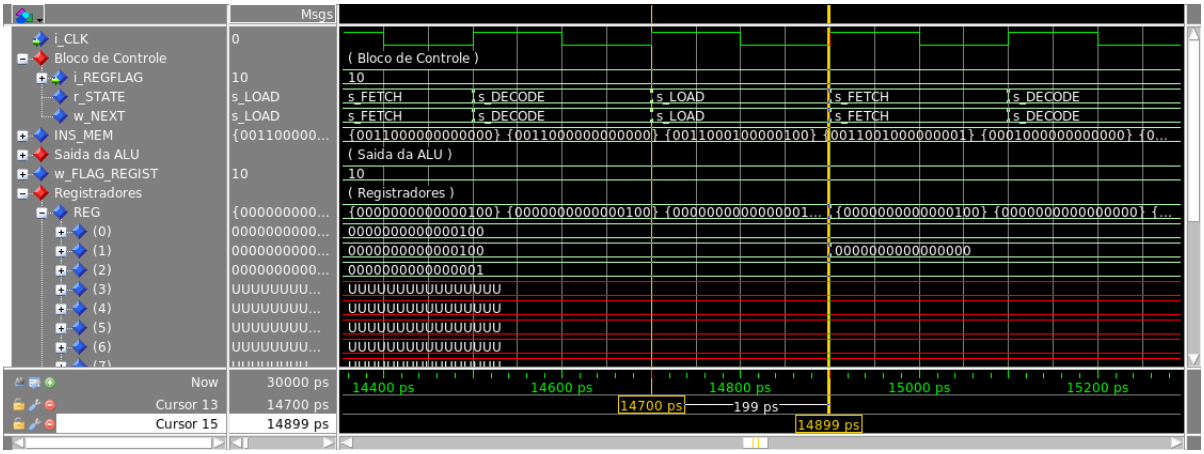
Como demonstrado todos os comandos funcionam como deveriam e algorítimos podem ser programados nesse processador, alguns comandos ficaram sem ser mostrados nesse algorítimo, e serão mostrados mais adiante. Os comandos não mostrados no algorítimo acima foram o carregamento do Banco de Dados ao Banco de Registradores e a soma entre dois registradores.

O seguinte algorítimo mostrará esses comandos:

```
---REG[0] = 00000000
---REG[0] = 00000000
---REG[1] = 00000100
---REG[2] = 00000001
---DB[0] = REG[0]
---ADD REG[0] , REG[0] + REG[2] *JNE -3*
---CMP REG[0] , REG[1]
---JNE -3
---REG[1] = DB[0]

w_DATA_TO_PROGRAM       <= "0011000000000000" ;wait for c_CLK_PERIOD;
    ---REG[0] = 00000000
        w_DATA_TO_PROGRAM       <= "0011000000000000" ;wait for
            c_CLK_PERIOD;   ---REG[0] = 00000000
        w_DATA_TO_PROGRAM       <= "0011000100000100" ;wait for
            c_CLK_PERIOD;   ---REG[1] = 00000100
        w_DATA_TO_PROGRAM       <= "0011001000000001" ;wait for
            c_CLK_PERIOD;   ---REG[2] = 00000001
        w_DATA_TO_PROGRAM       <= "0001000000000000" ;wait for
            c_CLK_PERIOD;   ---DB[0] = REG[0]
        w_DATA_TO_PROGRAM       <= "0010000000000010" ;wait for
            c_CLK_PERIOD;   ---ADD REG[0] , REG[0] + REG[2] *JNE -3*
        w_DATA_TO_PROGRAM       <= "0100000000000001" ;wait for
            c_CLK_PERIOD;   ---CMP REG[0] , REG[1]
        w_DATA_TO_PROGRAM       <= "1000000011111101" ;wait for
            c_CLK_PERIOD;   ---JNE -3
        w_DATA_TO_PROGRAM       <= "0000000100000000" ;wait for
            c_CLK_PERIOD;   ---REG[1] = DB[0]
```



É somado os registradores 0 e 2 e o valor é guardado no registrador 0.



É carregado o que está no endereço 0 do Bando de Dados e guardado no registrador 1. O valor guardado no Banco de dados é 0 como mostrado na imagem seguinte.