UNIVERSIDADE FEDERAL DE SANTA CATARINA

MICROPROCESSADOR

Descrito em VHDL

Pedro Augusto Ceriotti

Florianópolis, junho de 2011.

SUMÁRIO

OBJETIVOS *3*

1.0 ESPECIFICAÇÃO *4*

2.0 DESCRIÇÃO DAS FERRAMENTAS UTILIZADAS *5*

3.0 PROJETO *7*

4.0 FLUXO DE ESTADOS *16*

5.0 DIAGRAMA DE BLOCOS *17*

6.0 CONCLUSÃO *18*

7.0 ANEXO *19*

OBJETIVOS

Temos como objetivo final no desenvolvimento deste projeto a simulação da arquitetura de um microprocessador na linguagem de descrição de hardware VHDL, com um conjunto reduzido de instruções e todos os seus componentes.

ESPECIFICAÇÃO

Partindo de um conjunto de instruções pré-determinado, o microprocessador deverá possuir as seguintes características:

- *Registrador PC (Program Counter):* contém o endereço da próxima instrução a ser executada;
- *Registrador IR (Instruction Register)*: que contém a instrução de máquina em uso na arquitetura em um determinado instante;
- *Registrador AC (accumulator):* registrador que armazena resultados intermediários de operações;
- *Memória de dados;*
- *Memória de instruções;*
- *Conjunto de instruções*: listadas na tabela 1 a seguir (onde *loc* representa um endereço de memória de 8 bits).

Código de Máquina	Instrução	Descrição
00000000 loc	LDA loc	AC <- memoria[loc]
00000001 loc	STA loc	memoria[loc] <- AC
00000010 loc	ADD loc	AC <-AC + memoria[loc]
00000011 loc	AND loc	AC <-AC AND memoria[loc]
00000100 loc	JMP loc	PC <- loc
00000101 loc	BRN loc	AC < 0 ? loc : PC+1
00000110 loc	BRZ loc	AC == 0 ? loc : PC+1
00000111 XX	HALT	HALT

Tabela 1 - Conjunto de Instruções

DESCRIÇÃO DAS FERRAMENTAS UTILIZADAS

Abaixo serão descritas as ferramentas utilizadas na realização do projeto.

i. **Modelsim SE Plus 6.2c:** foi o simulador utilizado no desenvolvimento da descrição de hardware e também para fazer a verificação/debug do código em VHDL.

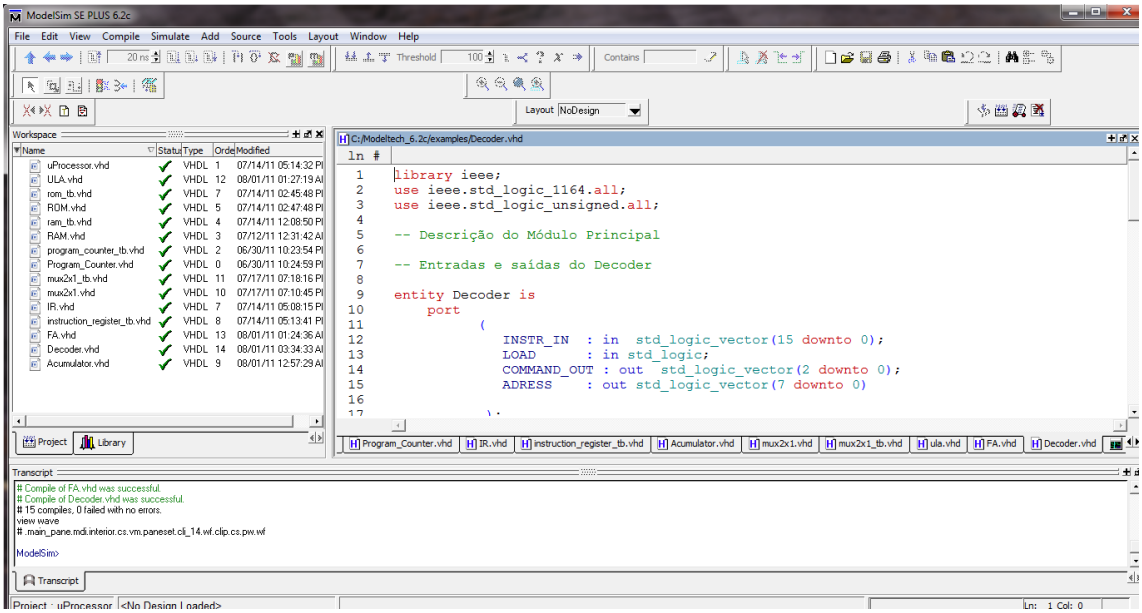

Figura 1 - Modelsim SE Plus 6.2c

ii. **VHDL:** foi a linguagem de descrição de hardware utilizada, a escolha se deu por já ter sido utilizada em outros trabalhos.

PROJETO

Abaixo será descrito como se deu o desenvolvimento do projeto, bem como os componentes descritos em VHDL com suas respectivas entradas e saídas.

- ***Program Counter (PC):*** é um registrador utilizado para armazenar o endereço da próxima instrução a ser executada.

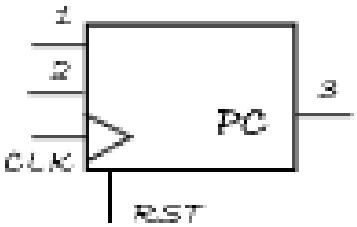

Figura 2 - Program Counter

Legenda da figura 2, contendo as entradas e saídas do Program Counter:

1	Recebe o endereço passado na instrução JMP ou condicional
2	Bit de controle da instrução JMP e condição
3	Saída contendo o endereço da instrução a ser executada
CLK	Load recebido da FSM
RST	Reset

- ***Instruction Register (IR):*** é um registrador utilizado para armazenar o mnemônico da instrução que está sendo executada em determinado instante.

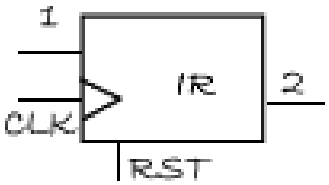

Figura 3 - Instruction Register

Legenda da figura 3, contendo as entradas e saídas do Instruction Register:

1	Recebe a próxima instrução a ser executada
2	Saída contendo a instrução em uso
CLK	Load recebido da FSM
RST	Reset

- ***Acumulador (ACC):*** é um registrador que armazena resultados intermediários de operações.

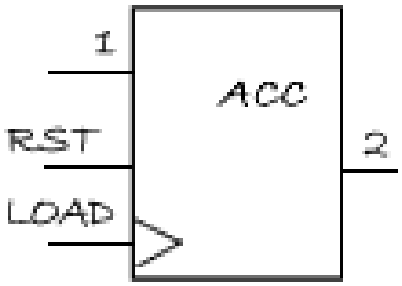

Figura 4 - Acumulador

Legenda da figura 4, contendo as entradas e saídas do Acumulador:

1	Recebe o valor vindo do mux 2x1, vindo da memória ou da ULA
2	Saída contendo o valor que está armazenado no Acumulador
CLK	Load recebido da FSM
RST	Reset

No pino 1, está ligada a saída de um Mux 2x1, tal como representado na figura abaixo:

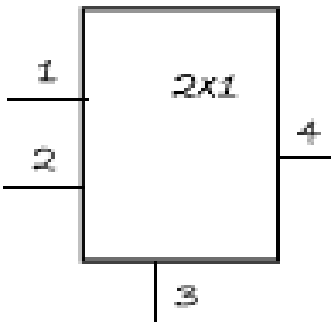

Figura 5 - Mux 2x1

Legenda da figura 5, contendo as entradas e saídas do Mux:

1	Valor contendo o resultado de uma operação realizada na ULA
2	Valor contendo o dado de um determinado endereço de memória
3	Bit de controle do mux que seleciona um dos valores
4	Saída do Mux

- ***Memória de Instruções:*** dispositivo utilizado para armazenar as instruções do programa.

Figura 6 - Memória de Instruções

Legenda da figura 6, contendo as entradas e saídas da Memória de Instruções:

1	Entrada contendo o endereço da instrução
2	Load vindo da FSM
3	Saída contendo a instrução localizada no endereço solicitado pelo PC

- ***Memória de Dados:*** dispositivo utilizar para armazenar dados temporários utilizados durante a execução do programa.

Figura 7 - Memória de Dados

Legenda da figura 7, contendo as entradas e saídas da Memória de Dados:

1	Load vindo da FSM
2	Entrada contendo um endereço de memória
3	Entrada contendo um dado a ser guardado na memória
4	Sinal de controle vindo da FSM, indicando operação de escrita ou leitura
5	Saída contendo um dado que será enviado ao acumulador
RST	Reset

- ***Decoder:*** dispositivo utilizado para fazer a decodificação da instrução e depois enviar um sinal de controle para a máquina de estados.

Figura 8 - Decoder

Legenda da figura 8, contendo as entradas e saídas do Decoder:

1	Entrada contendo a instrução a ser decodificada (16 bits)
2	Load vindo da FSM
3	Saída contendo um sinal de controle para a máquina de estados
4	Saída contendo o endereço [loc] contido na instrução (8 bits)
CLK	Load vindo da máquina de estados

- ***Unidade Lógico Aritmética (ULA):*** componente utilizado para realizar operações lógicas e aritméticas entre dois valores de 8 bits.

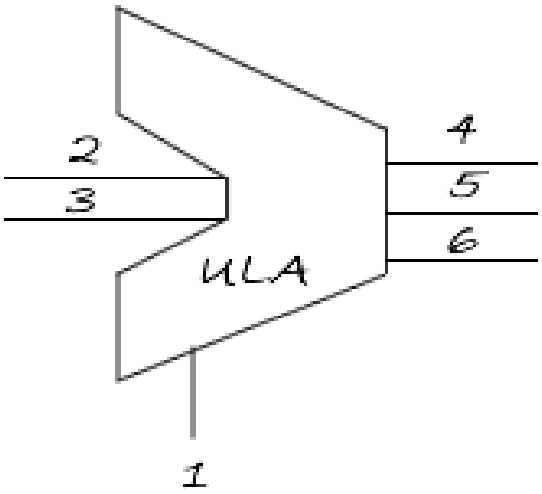

Figura 9 - ULA

Legenda da figura 9, contendo as entradas e saídas da ULA:

1	Entrada contendo o bit de controle que seleciona a operação a ser executada (AND ou ADD)
2	Entrada contendo o primeiro operando
3	Entrada contendo o segundo operando
4	Saída contendo o resultado da operação

5	Saída contendo o valor do FLAG_ZERO
6	Saída contendo o valor do FLAG_NEG

- ***Full Adder (FA):*** componente utilizado para realizar a operação de soma dentro da ULA.

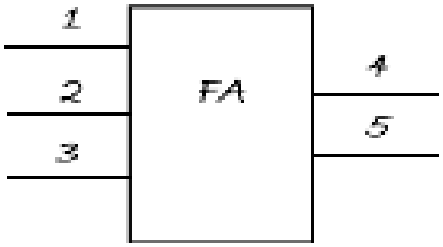

Figura 10 - FA

Legenda da figura 10, contendo as entradas e saídas do FA:

1	Primeiro operando
2	Segundo operando
3	Carry-in
4	Resultado da soma
5	Carry-out

Para realizar a soma de 8 bits, são utilizados 8 FA's, realizando-se a operação bit-a-bit e mapeando um resultado num sinal chamado SOMA_AUX.

- ***Máquina de Estados (FSM):*** é a unidade de controle do microprocessador, é a partir dela que serão enviados os sinais que controlam a operação dos demais componentes.

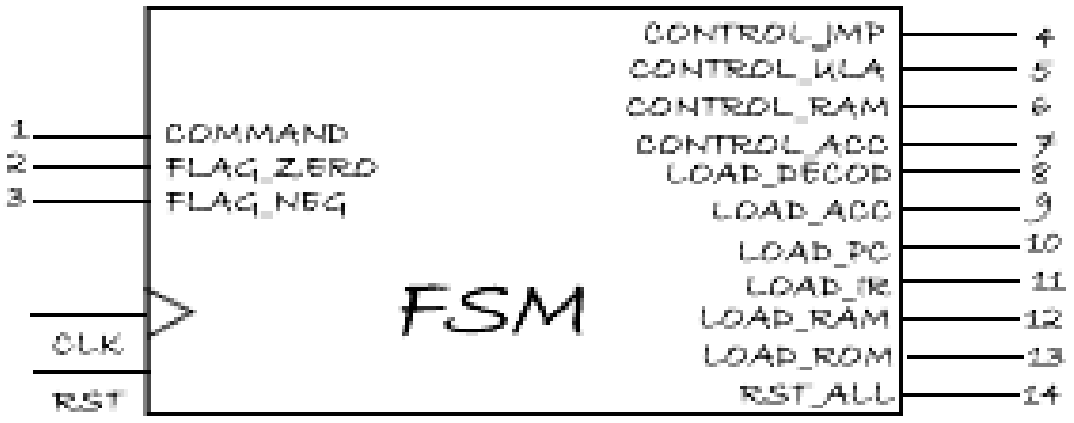

Figura 11 - FSM

Legenda da figura 11, contendo as entradas e saídas da máquina de estados:

1	Entrada contendo um comando de instrução vindo do Decoder
2	Entrada contendo o valor do FLAG_ZERO vindo da ULA
3	Entrada contendo o valor do FLAG_NEG vindo da ULA
4	Controle do PC, quando há uma instrução de JMP
5	Controle da ULA, que seleciona a operação desejada
6	Controle da memória RAM, no caso de uma instrução de READ/WRITE
7	Controla o Mux do acumulador, escolhendo entre um dado vindo da ULA ou da memória de dados
8	Load do Decoder
9	Load do Acumulador
10	Load do PC
11	Load do IR
12	Load da RAM
13	Load da ROM
14	Sinal de reset dos demais componentes
CLK	Saída contendo um sinal de controle para a máquina de estados
RST	Load vindo da máquina de estados

Legenda
ENTRADA
SAÍDA

A máquina de estados neste projeto possui doze estados, sendo eles descritos abaixo:

- ***START:*** é o estado inicial do microprocessador, acionado quando há uma operação de RESET, neste estado todos os valores dos registradores e os LOAD's enviados pela FSM são zero. Abaixo segue o código VHDL que descreve o estado START:

```
when START =>  LOAD_ACC   <= '0';
               LOAD_PCAUX <= '0';
               LOAD_PC    <= '0';
               LOAD_IR    <= '0';
               LOAD_ROM   <= '0';
              LOAD_DECOD <= '0';
              RST_ALL    <= '1';

              PE <= SINST;
```

- ***SINST:*** é o estado em que a máquina de estado enviará os sinais para o Instruction Register e para o Program Counter, para que estes sejam atualizados, dando início à busca por uma instrução. Abaixo segue o código VHDL que descreve o estado SINST:

```
when SINST =>  RST_ALL     <= '0';
               LOAD_ACC    <= '0';
               LOAD_RAM    <= '0';
               CONTROL_JMP <= '0';
               LOAD_PC     <= not(LOAD_PCAUX);
               LOAD_ROM    <= '1';

               PE <= LOADIR;
```

- ***LOADIR:*** esse estado faz apenas com que a FSM envie um sinal para o Instruction Register, para que seja carregada a instrução a ser executada no ciclo atual.

```
when LOADIR =>  LOAD_IR <= '1';

               PE <= DECOD;
```

- ***DECOD:*** é o estado em que a instrução atual será decodificada, e um sinal de comando será enviado para a máquina de estados, assim como o endereço [loc], contido na instrução, será enviado à memória de dados. Abaixo segue o código VHDL que descreve o estado DECOD:

```
when DECOD => LOAD_PC    <= '0';
              LOAD_IR    <= '0';
              LOAD_ROM   <= '0';
              LOAD_DECOD <= '1';

              PE <= ID;
```

- ***ID:*** é o estado em que máquina de estados irá identificar o comando enviado pelo decoder e, então, prosseguir para o próximo estado de acordo com o tipo de instrução a ser executada . Abaixo segue o código VHDL que descreve o estado ID:

```
when ID   => LOAD_DECOD <= '0';

        case COMMAND is
          when x"1" => PE <= RW;
          when x"2" => PE <= RW;
          when x"3" => PE <= OP;
          when x"4" => PE <= OP;
          when x"5" => PE <= JMPR;
          when x"6" => PE <= COND;
          when x"7" => PE <= COND;
          when x"F" => PE <= BREAK;
```

```
        when others => PE <= BREAK;
    end case;
```

- ***RW:*** neste estado a instrução em andamento é de leitura ou escrita na memória, deste modo serão enviados os sinais correspondentes para que cada operação seja realizada. Abaixo segue o código VHDL que descreve o estado RW:

```
when RW    =>
            if(COMMAND = x"1") then
               CONTROL_RAM <= '0'; -- Leitura
               CONTROL_ACC <= '0';
               LOAD_RAM  <= '1';
               PE <= CACC;

            elsif(COMMAND = x"2") then
               CONTROL_RAM <= '1'; -- Escrita
               LOAD_RAM <= '1';
               PE <= SINST;

            end if;
```

- ***OP:*** este é o estado em que a instrução em andamento é uma operação a ser realizada na ULA, deste modo serão enviados os sinais correspondentes para que se dê a operação. Abaixo segue o código VHDL que descreve o estado OP:

```
when OP   =>      LOAD_RAM <= '1';

            PE <= RESULT;
```

- ***RESULT:*** esse é o estado em que o resultado da operação realizada anteriormente será enviado ao acumulador. Abaixo segue o código VHDL que descreve o estado RESULT:

```
when RESULT  =>    if(COMMAND = x"3") then
                     CONTROL_ULA <= '0'; -- OP ADD
                   elsif(COMMAND = x"4") then
                     CONTROL_ULA <= '1'; -- OP AND
                   end if;

                    CONTROL_ACC <= '1';
                    PE <= CACC;
```

- ***COND:*** esse é o estado em que a instrução em execução é uma instrução de jump condicional. Para sua execução são verificados os flags vindos da ULA após a operação. Abaixo segue o código VHDL que descreve o estado COND:

```
when COND  =>
                if((COMMAND = x"6" and FLAG_NEG = '1') or
                (COMMAND = x"7" and FLAG_ZERO = '1')) then
                   PE <= JMPR;
                end if;

                PE <= SINST;
```

- ***JMPR:*** esse estado corresponde a execução de uma instrução de JMP e, também, quando é verificado um jump condicional. Abaixo segue o código VHDL que descreve o estado JMPR:

```
when JMPR  =>
                  CONTROL_JMP <= '1';
                  LOAD_PCAUX <= '1';
                  LOAD_PC <= '1';

                  PE <= SINST;
```

- ***BREAK:*** esse é o estado em que o microprocessador permanece inativo até que seja dado um comando de reset, pode ser originado da instrução HALT ou de uma instrução não especificada previamente. Abaixo segue o código VHDL que descreve o estado BRAKE:

```
when BREAK  =>
              if(RESET = '1') then
               PE <= START;
              else
               PE <= BREAK;
              end if;
```

FLUXO DE ESTADOS

Nesta seção apresentaremos um modelo que representa a transição de um estado para o outro, bem como as suas possibilidades.

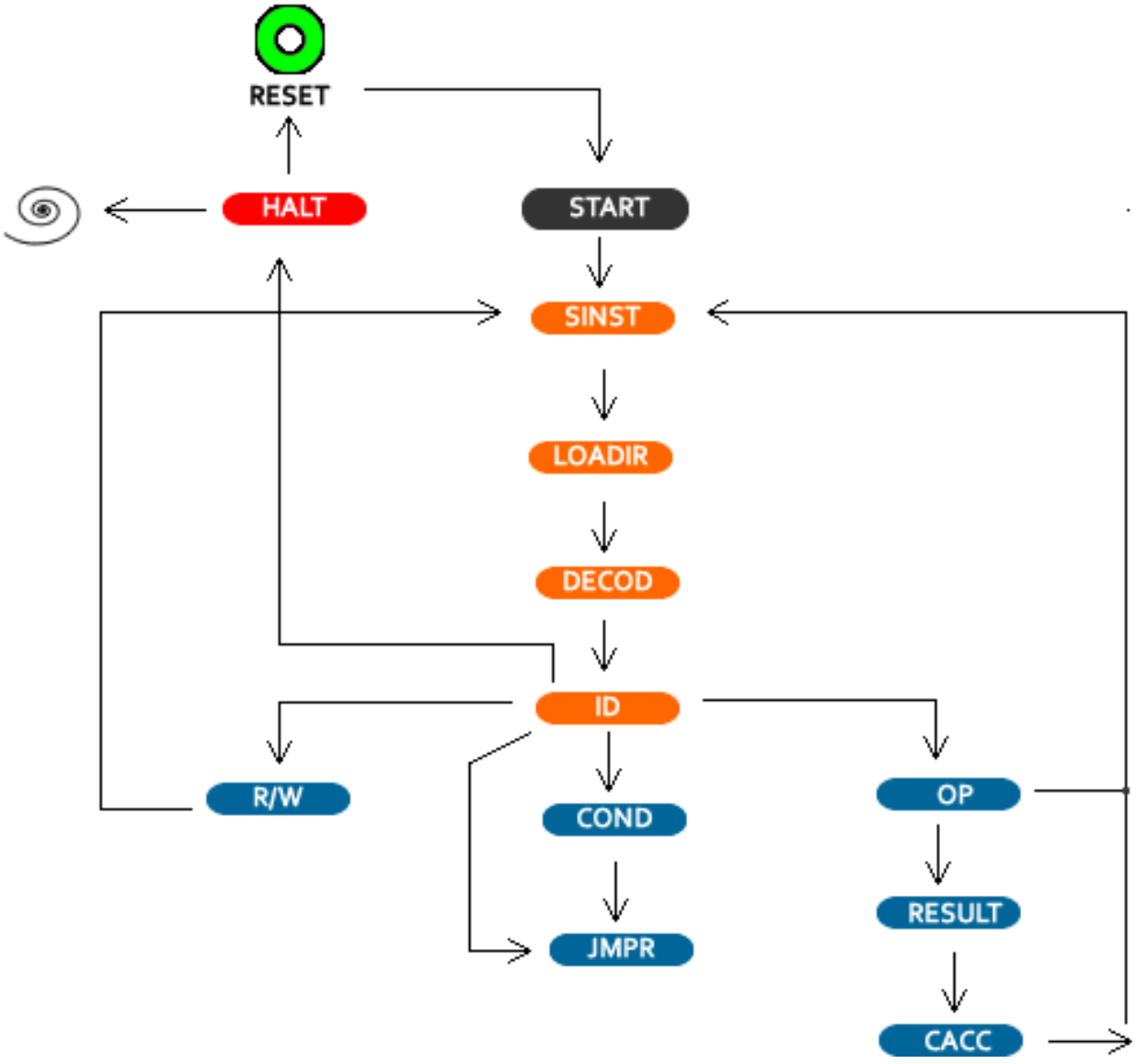

Figura 12 - Fluxo de Estados

DIAGRAMA DE BLOCOS

Nesta seção apresentaremos o diagrama de blocos do microprocessador com todos os seus componenentes.

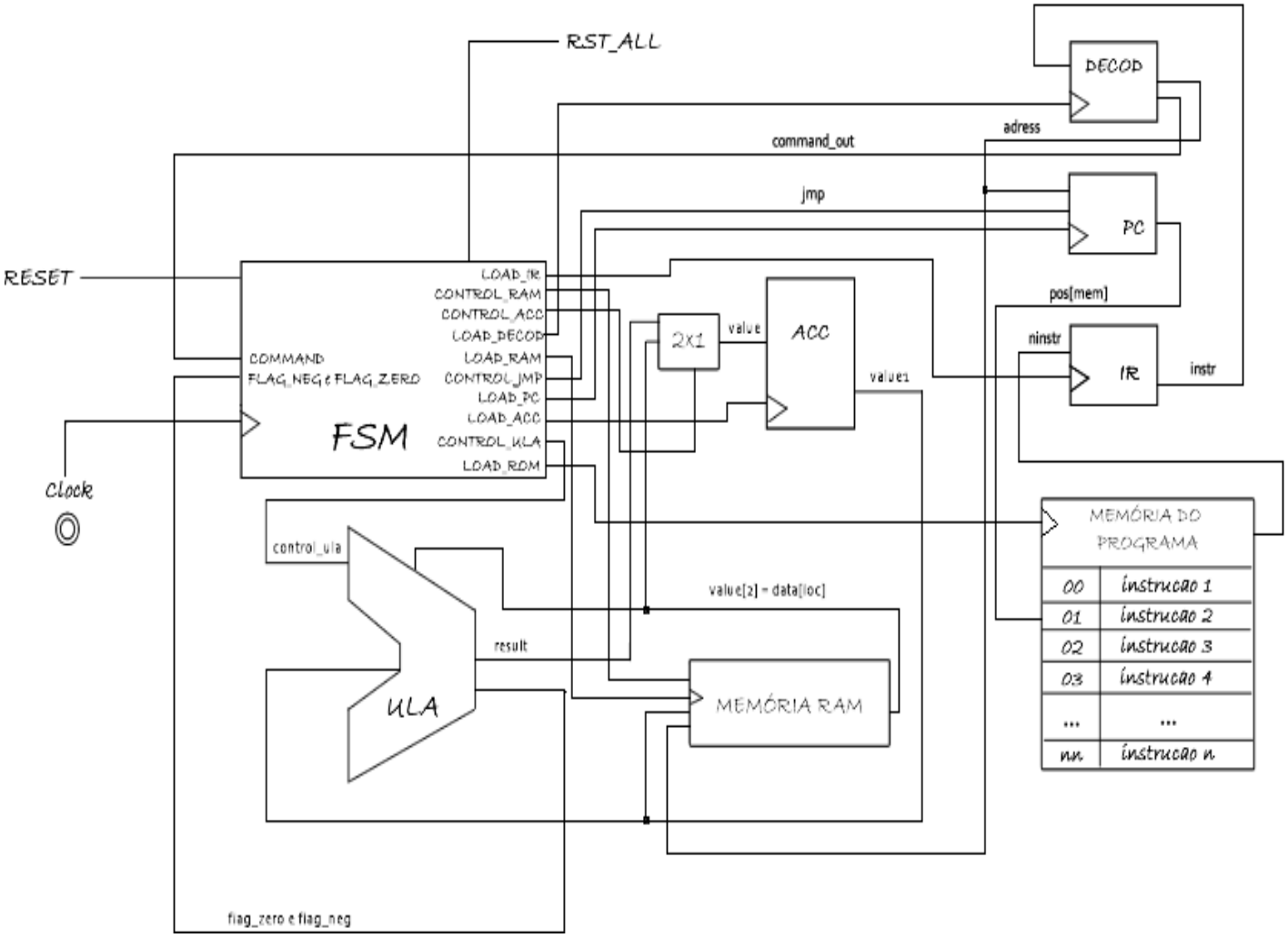

Figura 13 - Diagrama de Blocos

ANEXO

Segue anexo o código fonte em VHDL.