Advance Computer Architecture.

Design and Simulation of a CPU, Cache, Bus and Memory Datapath.

Due to: 04/11/2013

Adalberto Claudio.
A20294552.

Index.

Abstract.

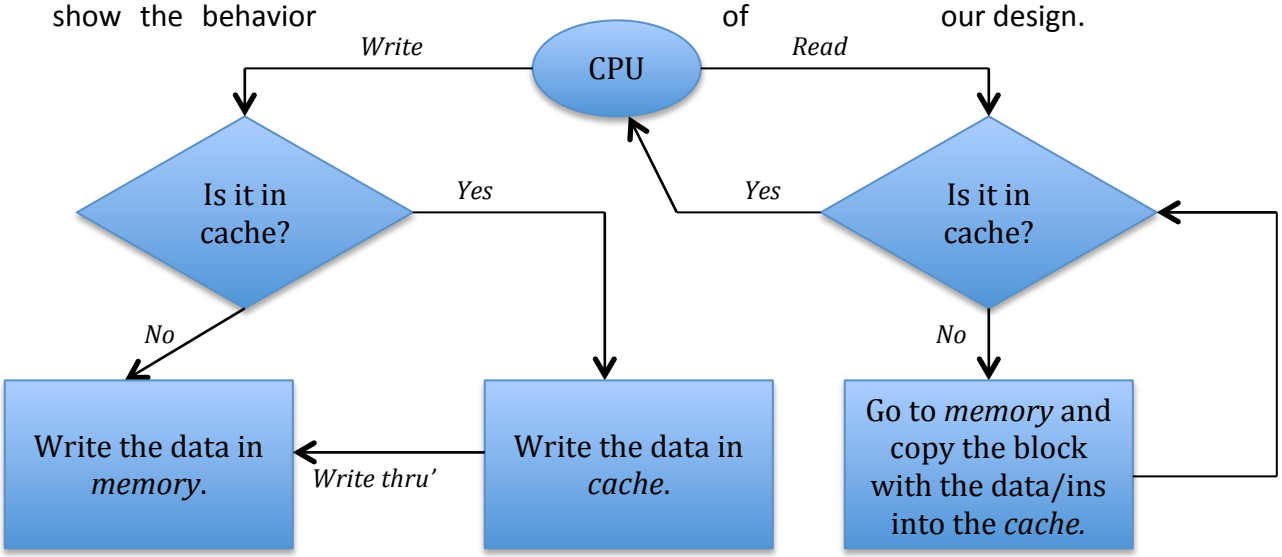
In this project we are going to design how the memory flows in a MIPS processor. It is going to be analyzed how instructions and data goes from *memory* to *cache* and then to CPU to be executed. To test the correctness of the design there are going to be implemented a few instructions that will provide us enough information to ensure the correctness of the design. The main construction is going to be the cache, which is going to be implemented following some specifications depending on our student ID number. The tool to develop the design it is going to be used VHDL.

Introduction.

To implement a CPU, memory and a cache, the first thing that we have to do is to study how different block work and how they communicate with others. In main *memory* we have all the values of data and instructions. It is connected with *CPU* and *cache* through the data bus that allow us to send and receive data and instruction to any block that we want.

Cache memory is a memory of fast access. It will store the data and instructions that have been used more recently and the access to this memory from the *CPU* is supposed to be very fast. It is divided in two parts: one for data and other for instructions. Data cache memory is smaller (128 Bytes) than Instruction cache (256 Bytes).

In the *CPU* is where it is going to take place the operation of the instructions. The data that is going to be used in the instruction is going to be transfer from cache. If we go to cache and there is not the data that we are looking on the cache, we will have a *read miss*. Therefore, we have to go to *memory* to look for the data and bring it to *cache* and then to *CPU*. If the data is found in *cache*, we have a *read hit* and take the data from *cache* and work with it. Once we have finished operating and we want to save the data (with a store instruction) we go to the *cache* to store the data. If the memory direction is not found in the *cache* we will obtain a *write miss*. In the case of *write miss* we will go to *memory* to write data, but we will not copy a block from *memory* to *cache* as we did in *read miss* case. This is call **write no allocate**. If the memory direction where we want to write is in *cache* we will obtain a *write hit* and it will be written in *cache*. As part of our specification we have to do a **write thru'**. It just only means that we have to update the memory direction whether we have a *write miss* or *write hit.* The following picture will show the behavior of our design.



Modules.

To implement the whole design we have decided to divide it in multiple modules. Each module will realise a scpecific function, with concrete outputs and inputs to the correctness of our design. In this section we are going to explain how works each module of the design. We have built 6 modules: CPU, Cache, Memory, PC, IR, clock and Final State Machine (FSM).

- **PC:** it is a module that it is going to receive the an instruction and depending on a value (*IWrite*) it is going to write the next instruction or not.

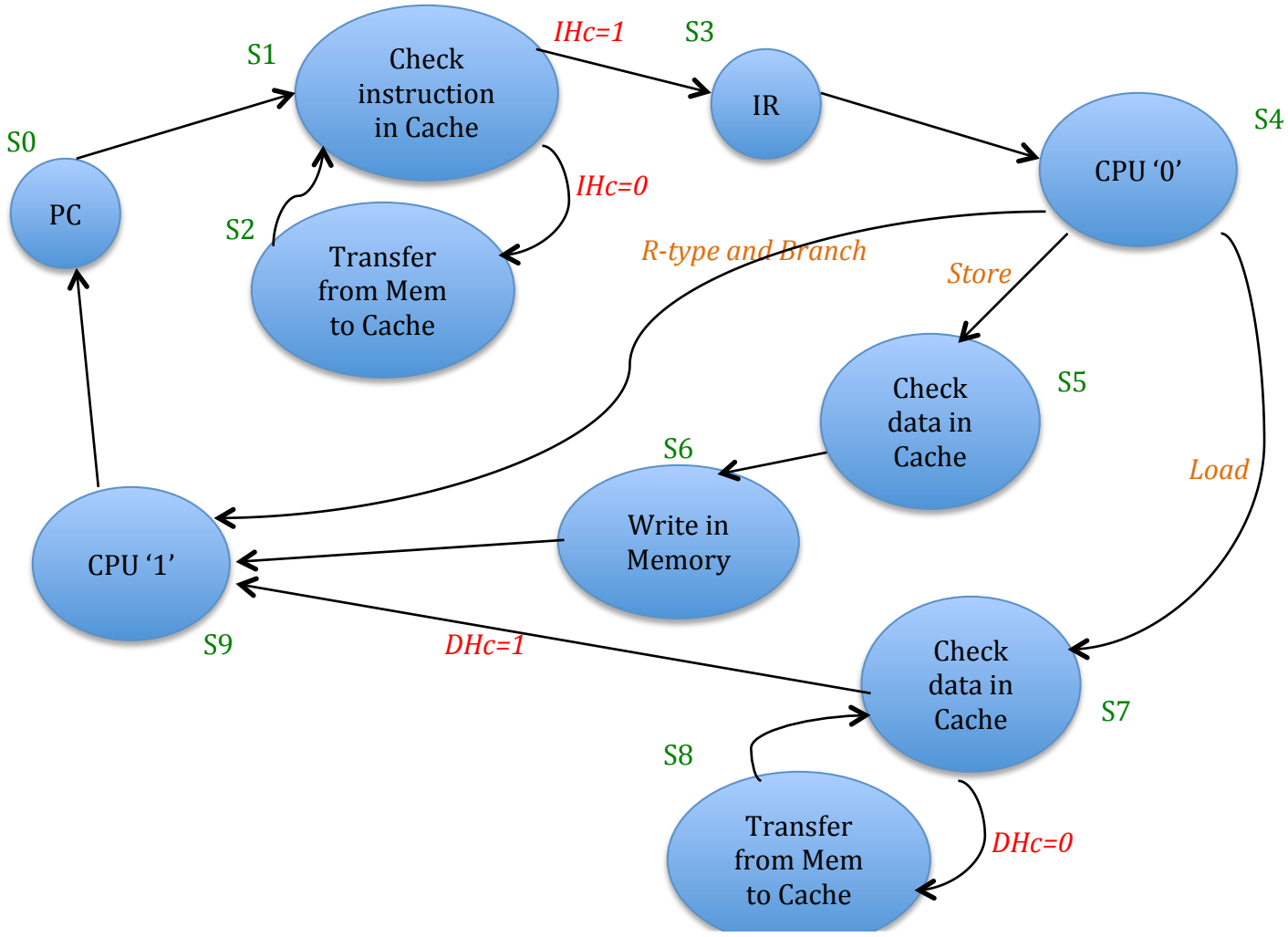
- **IR:** it is instruction register and, such as in the *PC*, we are going to have as input the instruction that it is going to be executed. Again, we have a parameter that will allow us to write in IR.

- **CPU**: in this module are going to take place all the operation that the *IR* sends to it. Depending on the *opCode* we are going to have several option: *R*-type, *I*-type or *J*-type. In the case of *R*-type we are going to execute an *add, nor* or *sll.* In the case of *I*-type we have *store* and *load*. This two operation are going to interact with *Cache* and *Memory.* Therefore, the most important challenge of this project is to study how interact the *Cache*, the *Memory* and the *CPU* with the different conditions that we have to implement: write thru' and write allocate. In case of *load*, we have to send the result to *Cache* and to *Memory.*

- **Memory:** in *Memory* we have stored all data and instructions. When *CPU* need a data (load) or an instruction and *Cache* doesn't have it, we have to take a block of 8 words and introduce it in *Cache* (write thru'). In the case of store instruction, we have to go to *Memory* and write there de value that we want to store (write no allocate). For *Memory* we have two parameter to decide when we want to write or read: *MemWrite* and *MemRead.*

- **Cache:** it has to different areas as we have explained before: I-cache (for instructions) and D-cache (for data). This module is going to receive data from *CPU* and *Memory*, and the address where we have to look for the data/instruction it is imposed by the address in that comes from *PC*, *CPU* or *FSM.* We are going to send data from *Cache* to *Memory* or *CPU* and to *IR* it is going to be sent instructions. Finally to *Memory* we are going to send the address where data has to be stored. If we have a hit (*IHc/DHc* equal to '1') the data/ instruction that we are looking for is in *Cache*. On the other hand, if we obtain a miss (*IHc/DHc* equal to '0'), we have to go to memory and take a block of memory that conctain that data/instruction. This is made with the address that is provided from *PC*.

- **FSM:** this is the state machine. Depending on the inputs and outputs this module is the responsible to make the instruction go step by step. First taking the instruction, leaving the *CPU* does whatever it has to do, if it is necessary to *Cache* or *Memory*, *FSM* will put the values in the outputs that are needed to acces to them. Finally, it will let come in the new addres to the *PC* to look for next instruction.



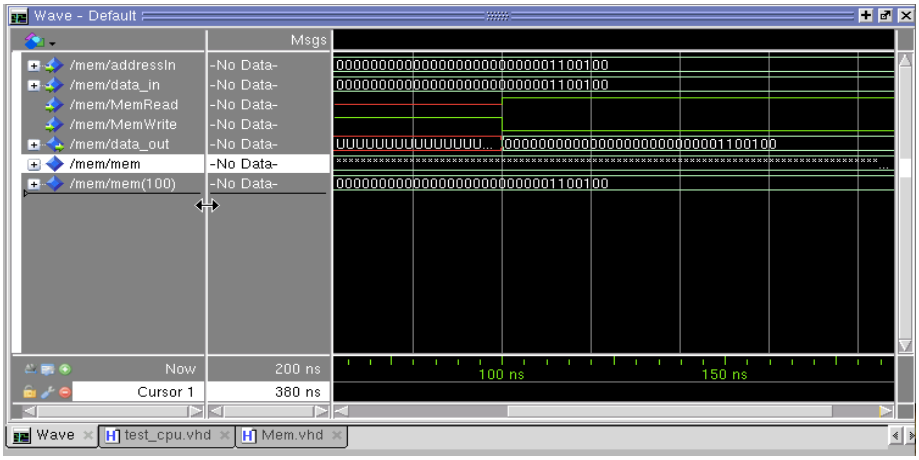
In the previous diagram we can observe states that our design has. In S0 we read the address in *PC*. Then we go to *Cache* to look for it. Depending on if it is or not in *Cache* we will go to S3 (the instruction is in *Cache*) or to load a block of memory from *Memory* to *Cache.* In S4 we are in *stateCPU* equal to 0, we will only calculate the operation that are required. Later, depending on the *opCode* we can go to different states:

1. Add/Branch: we will go to S9 where the address out is calculated. It could be the previous address plus 4 or in case that the branch condition is true, the one that the branch instruction says.
2. Store: we will go to S5 and look for the memory address in *Cache* to store the value. Afterwards we have to update that same value in *Memory* (write thru'). If the address is not found in *Cache* we only have to write the value in *Memory* (write no allocate).
3. Load: the following state is S7. As in S1, we have to know if we have a hit or a miss and then do what we have to do. If we have a miss, it id necessary to go to *Memory* and load the block where the data is stored to *Cache*. If what we have obtained is a hit, we just go to S9 and continue as the rest of the instructions.

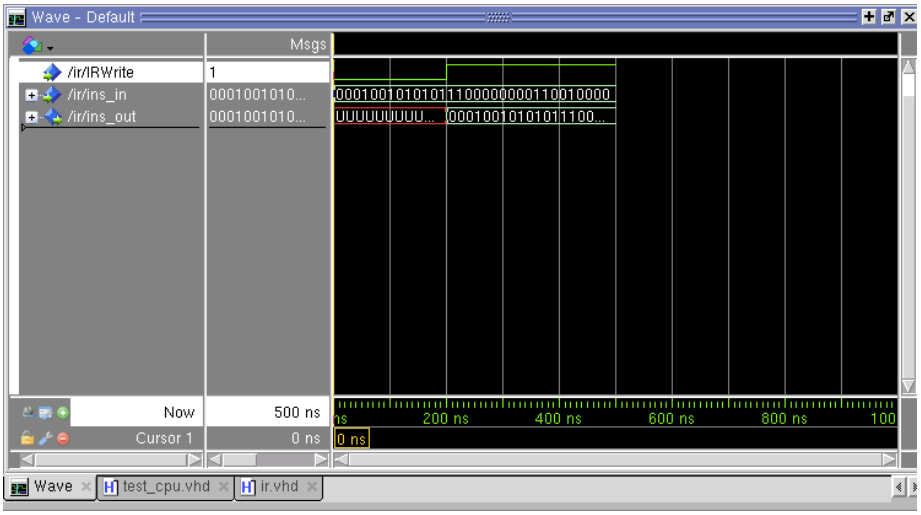
Results.

In this section we are going to examine all the modules that were tested separately and see what we have obtained in running specific test benches for each one. Finally we will analyze the main block where all modules where added as components. Therefore, we will test the whole program at once.

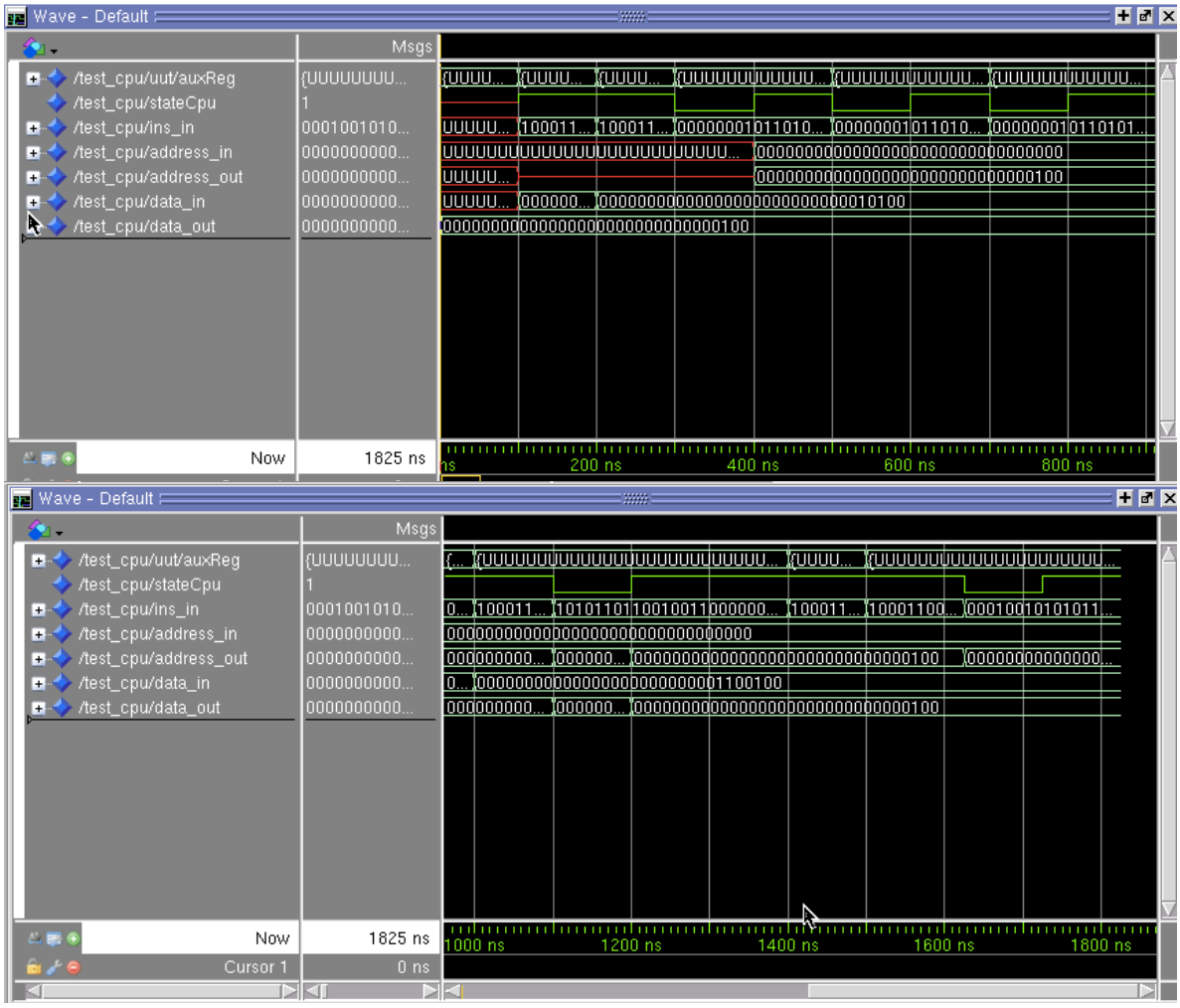
- **PC:** in the following image we can apreciate that when we introduce a address in addressin an a data in data_in, with *MemWrite* equal to 1, the data is writen in the address provided. An when we want to read it, making *MemRead* equal to one we obtain the data_out.



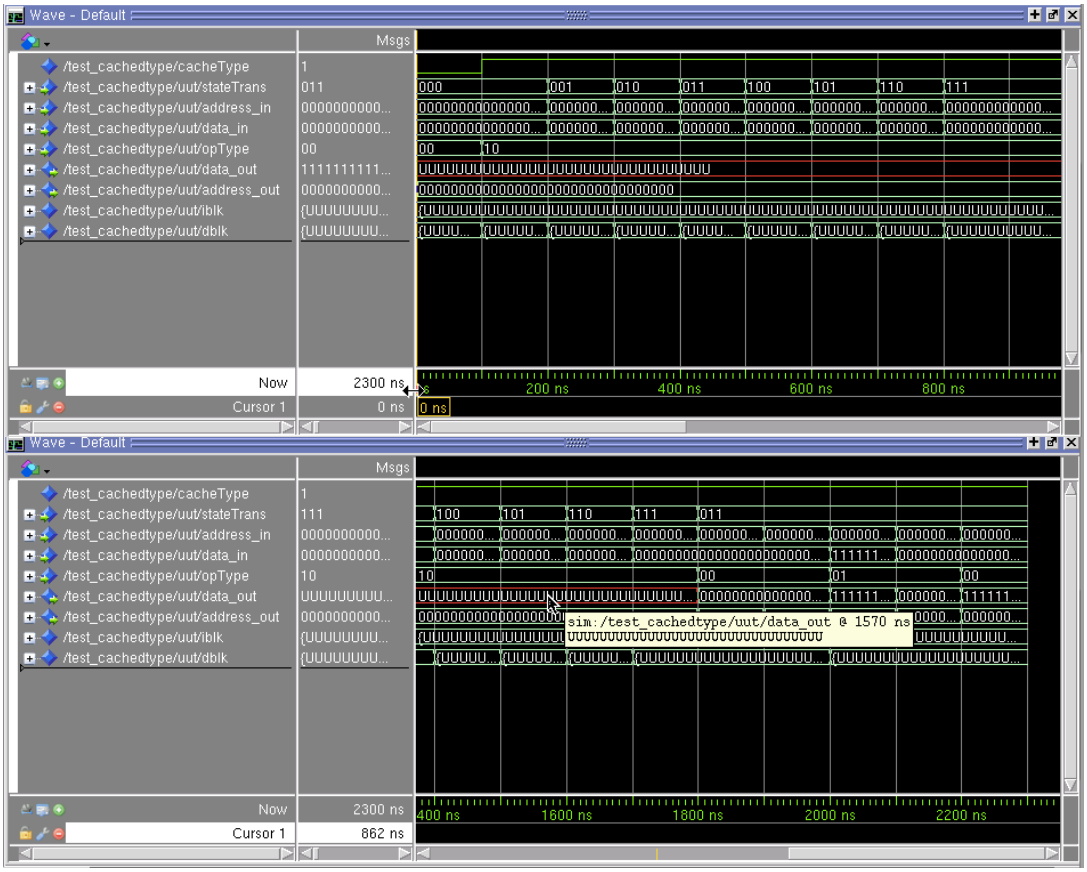
- **IR:** when we have an *IRWrite* equal to 1 we read the instruction that is in ins_in and put it on ins_out



- **CPU:** here we can appreciate the behaviour of the *CPU* when we use different instructions. Firstly we load some values in the registers that are going to be used. After that, the first instruction is an ADD, then a NOR and later a SLL. Afterwards, we load again values in a register that is going to be used in store instruction, then we run a STORE instruction, and finally a LOAD. We can appreciate in each case the outputis the right operation. Therefore, we are making the rights operations.

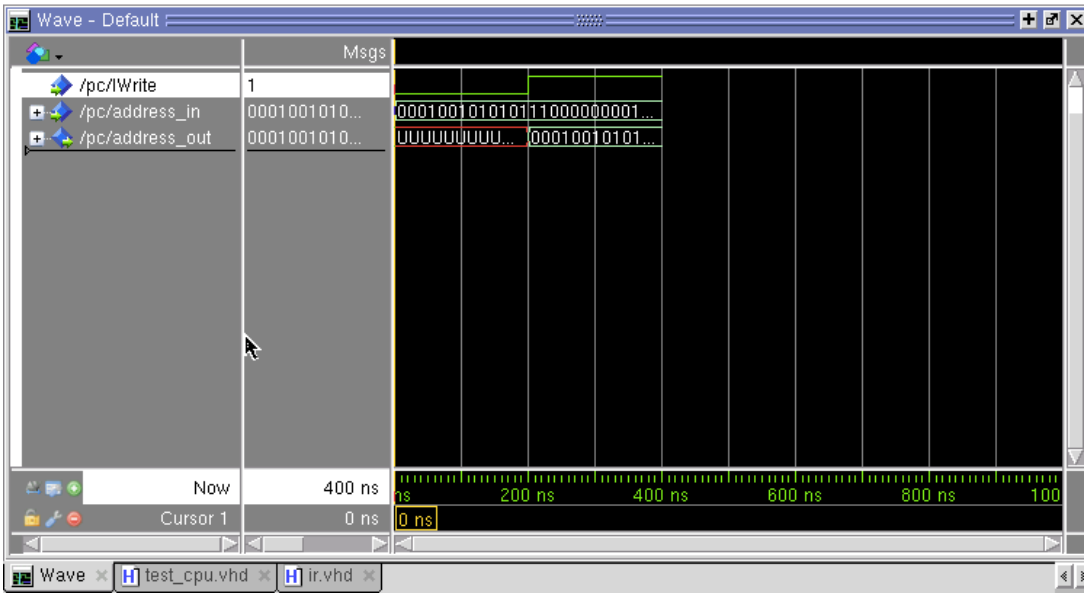


- **Cache:** we are going to show first the D-Cache and after the I-Cache. With a '1' in cacheType, we know that we are in D-Cache. We compare the tag of each direction with the address of address_in. If it matches with any tag of the *Cache*, we have found the data that we want and depending on the opType, we can read ("00"), write ("01") or transfer ("10"). In the case of transfer for D-Cache, we introduce values in the *Cache.* Then, with a "00" we can read the values that have been introduced before.

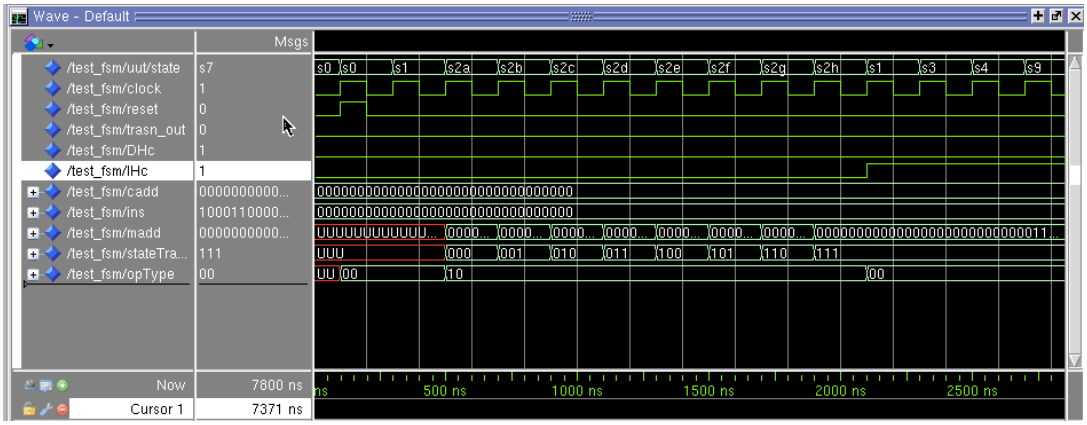


The case for I-Cache is similar, so we won't introduce more images.

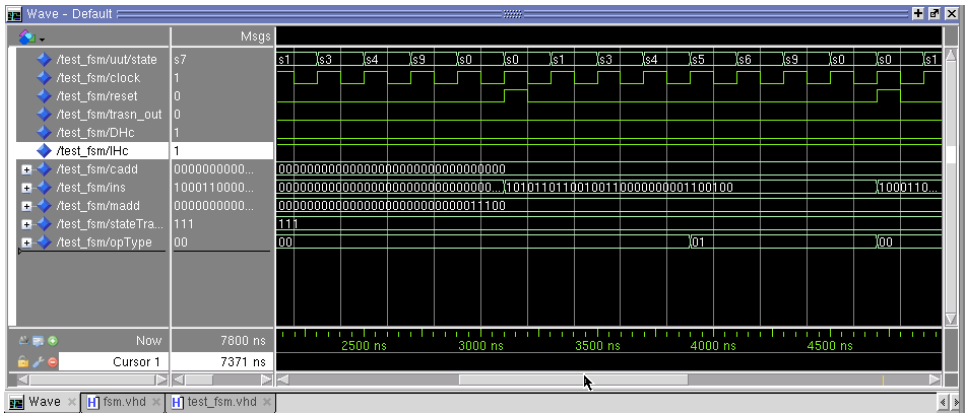
- **Memory:** this case is very simple, it has been tested that with a '1' in *IWrite* we write in *Memory* what it is in address_in.



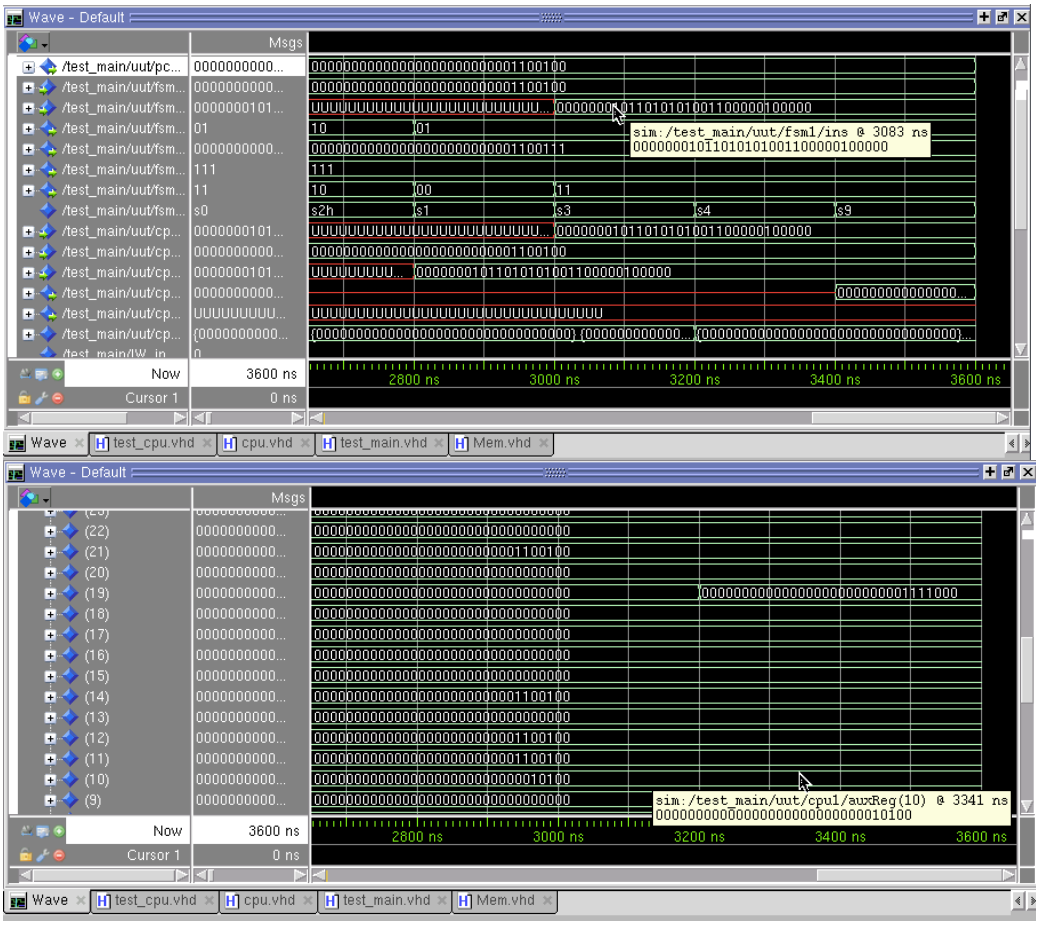
- **FSM:** in this case we have several images. In all we can appreciate that depending on the kind of operation (Add, Nor, Sll, Store, Load or Branch) we are going to activate some signals. This first image is an example of Add. The states S2a, S2b, etc are in total 8 and each one is a transfer of 8 word we have to do transfer from *Memory* to *Cache.*



The following is for a Store. Again it could be appreciate that the signals take the value that they have to.

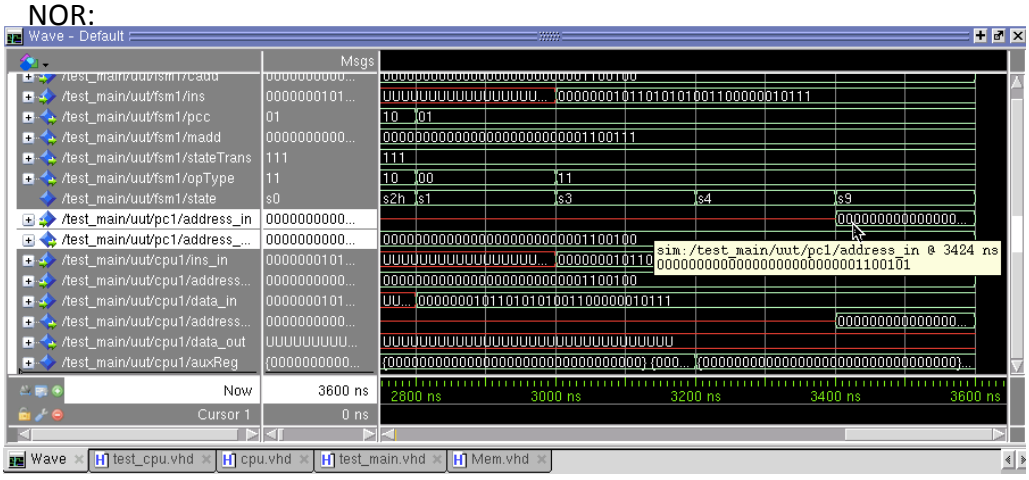


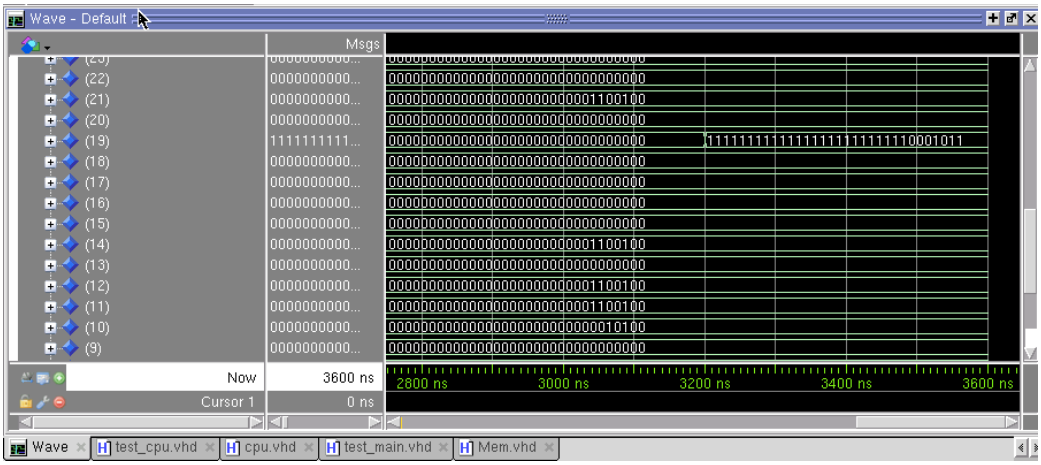
- **Main:** in this case it has been implemented all modules in the same file, so all of them are connected between them. It is going to be shown instruction one by one because when we tried to run it all 6 instructions we have warnings and the running stops because of that. We think that this warnings where caused because the *Cache* and *Memory* were full of undefined values. The first case is the Add instruction. In the first one we can see how the instruction is read, the program goes step by step (it can be appreciate that it goes form S1 to S3 to S4 and finally S9) and in the second one we can see how the registers change theirs values.in concrete, register number 19 changes and the value of the addition is stored in it.



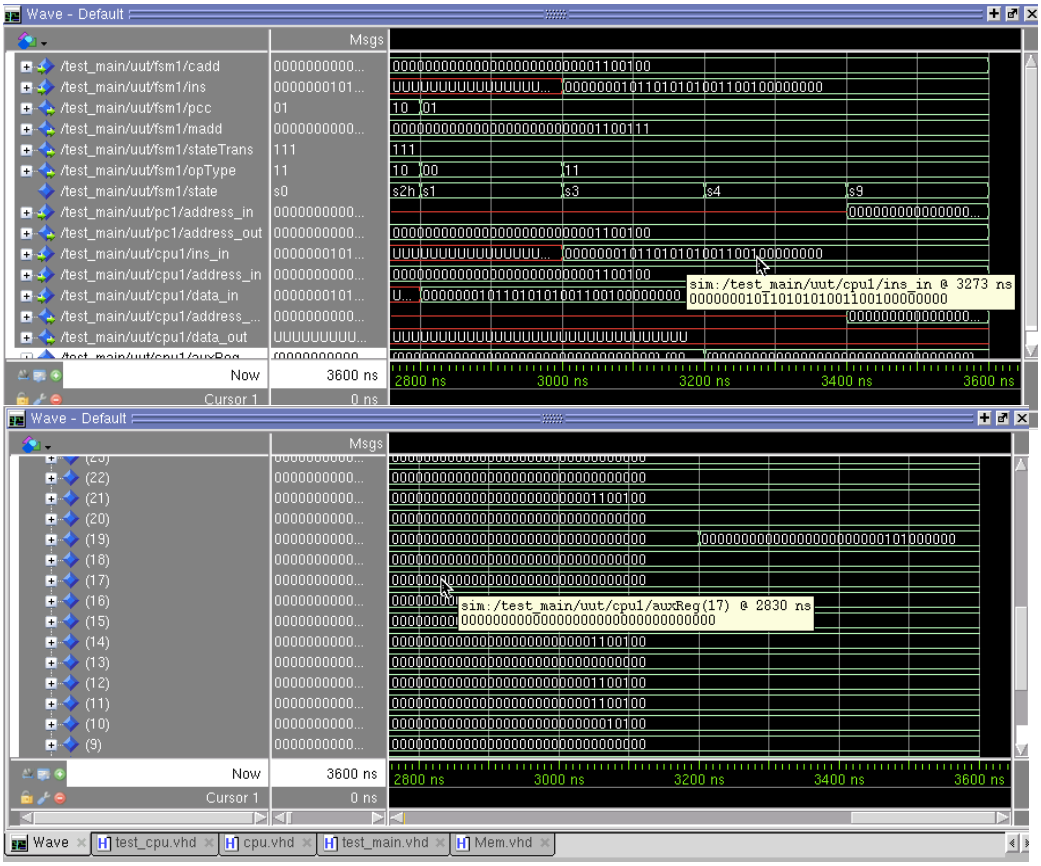
In the following images we are going to see the behavior of Nor and Sll instructions. As in the Add case, there are two images for each instruction. In the first one we will see the operation and all the states, and in the second one how the registers change.

NOR:

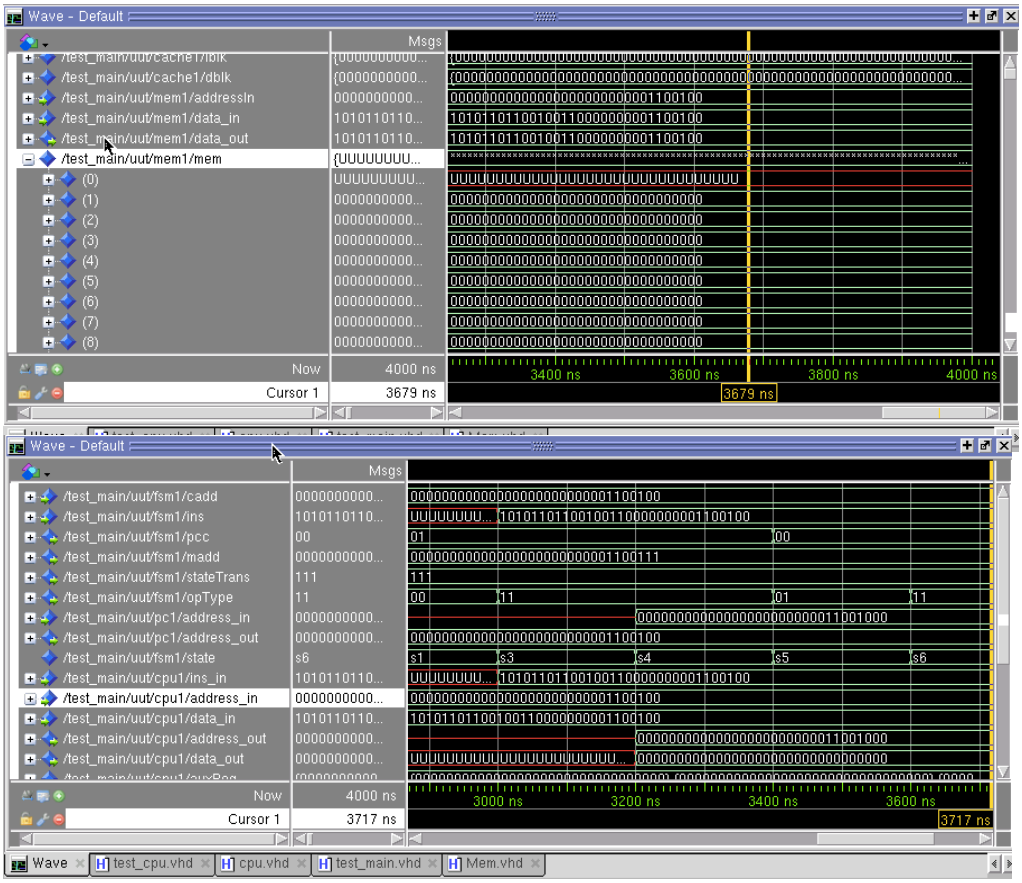




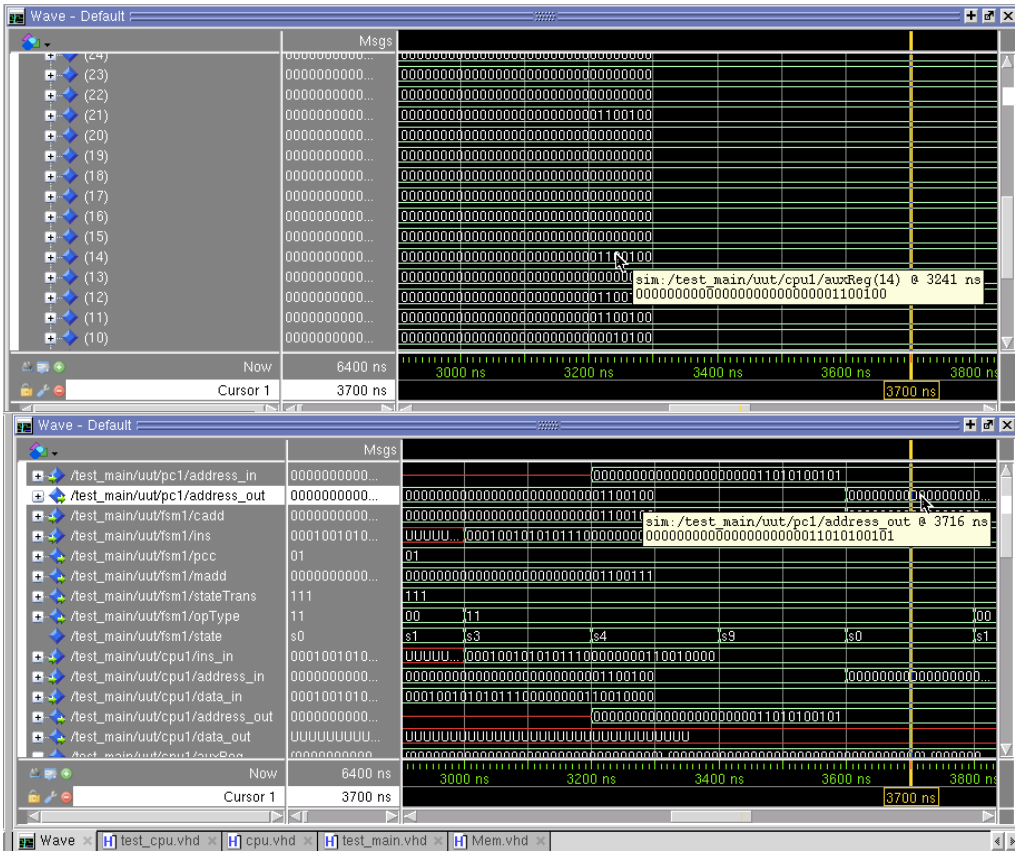
Sll:



Now we are going to study a Store instruction. In the first one we see that we have a miss so we have to implement a write thru': we just write in *Memory* but not in *Cache*. We will look for that address position in *Memory* and then write there. The second one shows the change in the states and how the following instruction's address is ready to go to *PC*.



The following case of studying is the Branch. In the first one we see the value of the two registers that are going to be compare to do the jump if the are equal. In this case we can see that both are equal. In the second image we see that address_out has been calculated and the *PC* takes it.



The last instruction is a load. As it has been seen with the cases tested, we have to load values to start the execution of the instructions. Therefore, we are not going to study this case as we did with the previous to reduce the information provided in this report.

Appendix.

PC code:

```
 library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity PC is
    port(
        IWrite        : in std_logic;
        address_in : in  std_logic_vector(31 downto 0);
        address_out: out std_logic_vector(31 downto 0)
   );
end entity;

Architecture behav of PC is

    begin

        process(IWrite, address_in)

        begin

            if IWrite = '1' then
                address_out <= address_in;
            end if;

        end process;
end architecture;
```

IR code:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity IR is
    port(
        IRWrite        : in std_logic;
        ins_in : in  std_logic_vector(31 downto 0);
        ins_out: out std_logic_vector(31 downto 0)
   );
end entity;

Architecture behav of IR is

    begin

        process(IRWrite, ins_in)
```

```
        begin

            if IRWrite = '1' then
                ins_out <= ins_in;
            end if;

        end process;
end architecture;
```

Memory code:

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

entity Mem is generic ( DATA_WIDTH :integer := 32; ADDR_WIDTH :integer
:= 10 );
  port(
        addressIn : in std_logic_vector(31 downto 0); --address Input
        data_in : in std_logic_vector(DATA_WIDTH-1 downto 0);
        MemRead : in std_logic;
        MemWrite : in std_logic;
        data_out : out std_logic_vector(DATA_WIDTH-1 downto 0)
        );
  end entity;

architecture rtl of Mem is
--Internal Variables--

  constant RAM_DEPTH :integer := 2**(ADDR_WIDTH);
    type RAM is array (integer range<>) of
std_logic_vector(DATA_WIDTH-1 downto 0);
  signal mem : RAM (0 to RAM_DEPTH-1):= (others =>(others=>'0'));


begin



--Memory Write Block
MEM_WRITE:
    process (addressIn, data_in, MemWrite)

     variable address1 : std_logic_vector(ADDR_WIDTH-1 downto 0);
     variable index : integer;
      begin
        address1 := addressIn(ADDR_WIDTH-1 downto 0);
        index := conv_integer(address1);
        if( MemWrite='1') then
          mem(100) <= "00000001011010101001100000100000";
          mem(101) <= "00000001011010101001100000010111";
          mem(102) <= "00000001011010101001100100000000";
          mem(103) <= "10101101100100110000000001100100";
          mem(104) <= "00010010101011100000000110010000";
```

```
          mem( index) <= data_in;
        end if;
    end process;

MEM_READ:
    process (addressIn, MemRead)

     variable address1 : std_logic_vector(ADDR_WIDTH-1 downto 0);

     begin
        address1 := addressIn(ADDR_WIDTH-1 downto 0);
        if(MemRead='1') then
          data_out <= mem(conv_integer(address1));
        end if;
    end process;

end architecture;
```

Cache code:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Cache is
    port(
        cacheType  : in  std_logic;                    -- Icahe=0;
Dcache=1
        stateTrans : in  std_logic_vector(2 downto 0);
        address_in : in  std_logic_vector(31 downto 0);
        data_in    : in  std_logic_vector(31 downto 0);
        opType     : in  std_logic_vector(1 downto 0);
        data_out   : out std_logic_vector(31 downto 0);
        address_out: out std_logic_vector(31 downto 0);
        IHc        : out std_logic;
        DHc        : out std_logic;
        dbset      : out std_logic;
        trasn_out  : out std_logic
    );
end entity;

Architecture behav of Cache is

    type iCacheMem is array(31 downto 0) of std_logic_vector(282
downto 0);
    type dCacheMem is array(15 downto 0) of std_logic_vector(282
downto 0);
    type prob is array(31 downto 0) of std_logic_vector(26 downto 0);
    signal pa : prob;
    signal pr  : std_logic_vector(282 downto 0);
    signal iblk: iCacheMem := (others =>(others=>'0'));
    signal dblk: dCacheMem := (others =>(others=>'0'));
```

```
    signal addr: std_logic_vector(26 downto 0);
    signal comp: std_logic_vector(26 downto 0);

    begin
        --Read Instruction Cache
        process( cacheType, stateTrans, opType, address_in, data_in)
            variable temp: std_logic_vector(26 downto 0);

            --ICache aux variables
            variable iaddress_in: integer;
            --DCache aux variables
            variable daddress_in: integer;

            begin
            --Read
            if opType="00" then
                -- ICache
                if cacheType='0' then
                    --Compares the tag into the cache.
                    for I in 0 to 31 loop
                        if (address_in(31 downto 5) = iblk(31-I)(282
downto 256)) then
                            --- Hit case
                            case address_in(4 downto 2) is
                                when "111" =>   data_out <= iblk(31-
I)(255 downto 224); IHc <= '1';--word 7
                                when "110" =>   data_out <= iblk(31-
I)(223 downto 192); IHc <= '1'; --word 6
                                when "101" =>   data_out <= iblk(31-
I)(191 downto 160); IHc <= '1'; --word 5
                                when "100" =>   data_out <= iblk(31-
I)(159 downto 128); IHc <= '1'; --word 4
                                when "011" =>   data_out <= iblk(31-
I)(127 downto 96);  IHc <= '1'; --word 3
                                when "010" =>   data_out <= iblk(31-
I)(95 downto 64);   IHc <= '1'; --word 2
                                when "001" =>   data_out <= iblk(31-
I)(63 downto 32);   IHc <= '1'; --word 1
                                when "000" =>   data_out <= iblk(31-
I)(31 downto 0);    IHc <= '1'; --word 0
                                when others => data_out <=
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
                            end case;
                            exit;
                        else
                            --- Miss case
                            IHc <='0';
                            address_out <= address_in;
                        end if;
                    end loop;
                end if;
                -- DCache
                if cacheType = '1' then
                     --Compares the tag into the cache.
                    for I in 0 to 15 loop
```

```
                        if (address_in(31 downto 5) = dblk(15-I)(282
downto 256)) then
                            --- Hit case
                            case address_in(4 downto 2) is
                                when "111" =>   data_out <= dblk(15-
I)(255 downto 224); DHc <= '1'; --word 7
                                when "110" =>   data_out <= dblk(15-
I)(223 downto 192); DHc <= '1';--word 6
                                when "101" =>   data_out <= dblk(15-
I)(191 downto 160); DHc <= '1';--word 5
                                when "100" =>   data_out <= dblk(15-
I)(159 downto 128); DHc <= '1';--word 4
                                when "011" =>   data_out <= dblk(15-
I)(127 downto 96);  DHc <= '1';--word 3
                                when "010" =>   data_out <= dblk(15-
I)(95 downto 64);   DHc <= '1';--word 2
                                when "001" =>   data_out <= dblk(15-
I)(63 downto 32);   DHc <= '1';--word 1
                                when "000" =>   data_out <= dblk(15-
I)(31 downto 0);    DHc <= '1';--word 0
                                when others => data_out <=
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
                            end case;
                            exit;
                        else
                            --- Miss case
                            DHc <='0';
                            address_out <= address_in;
                        end if;
                    end loop;
                end if;
            end if;

            --Write
            if opType="01" then
                -- DCache
                if cacheType='1' then
                    for I in 0 to 15 loop
                        if address_in(31 downto 5) = dblk(15-I)(282
downto 256) then
                            --- Hit case
                            case address_in(4 downto 2) is
                                when "111" =>   dblk(15-I)(255 downto
224) <= data_in; DHc<='1';--word 7
                                when "110" =>   dblk(15-I)(223 downto
192) <= data_in; DHc<='1';--word 6
                                when "101" =>   dblk(15-I)(191 downto
160) <= data_in; DHc<='1';--word 5
                                when "100" =>   dblk(15-I)(159 downto
128) <= data_in; DHc<='1';--word 4
                                when "011" =>   dblk(15-I)(127 downto
96)  <= data_in; DHc<='1';--word 3
                                when "010" =>   dblk(15-I)(95 downto
64)   <= data_in; DHc<='1';--word 2
                                when "001" =>   dblk(15-I)(63 downto
32)   <= data_in; DHc<='1';--word 1
```

```
                                when others=>   dblk(15-I)(31 downto
0)    <= data_in; DHc<='1';--word 0
                                end case;
                                exit;
                            else
                                --- Miss case
                                DHc<='0';
                            end if;
                        end loop;
                        address_out <= address_in;
                        data_out <= data_in;
                    end if;
                end if;

                --Transfer
                if opType="10" then
                    -- ICahce
                    if cacheType='0' then
                        iaddress_in:= conv_integer(address_in(9 downto
5));
                        iblk(iaddress_in)(282 downto 256) <= address_in(31
downto 5);
                        case stateTrans is
                            when "000" => iblk(iaddress_in)(31 downto 0)
<= data_in;
                            when "001" => iblk(iaddress_in)(63 downto 32)
<= data_in;
                            when "010" => iblk(iaddress_in)(95 downto 64)
<= data_in;
                            when "011" => iblk(iaddress_in)(127 downto 96)
<= data_in;
                            when "100" => iblk(iaddress_in)(159 downto
128)<= data_in;
                            when "101" => iblk(iaddress_in)(191 downto
160)<= data_in;
                            when "110" => iblk(iaddress_in)(223 downto
192)<= data_in;
                            when others => iblk(iaddress_in)(255 downto
224)<= data_in;trasn_out<='1';

                        end case;
                    end if;
                    -- DCache
                    if cacheType='1' then
                       daddress_in:= conv_integer(address_in(8 downto 5));
                        dblk(daddress_in)(282 downto 256) <= address_in(31
downto 5);
                        case stateTrans is
                            when "000" => dblk(daddress_in)(31 downto 0)
<= data_in;
                            when "001" => dblk(daddress_in)(63 downto 32)
<= data_in;
                            when "010" => dblk(daddress_in)(95 downto 64)
<= data_in;
                            when "011" => dblk(daddress_in)(127 downto 96)
<= data_in;
```

```
                        when "100" => dblk(daddress_in)(159 downto
128) <= data_in;
                        when "101" => dblk(daddress_in)(191 downto
160) <= data_in;
                        when "110" => dblk(daddress_in)(223 downto
192) <= data_in;
                        when others => dblk(daddress_in)(255 downto
224)<= data_in;trasn_out<='1';

                    end case;
                end if;
            end if;
        end process;
end architecture;
```

Cache Test bench:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity test_cache is
end entity;

architecture behav of test_cache is

    --Component Declaration of UUT
    component Cache
    port(
        cacheType  : in  std_logic;                    -- Icahe=0;
Dcache=1
        stateTrans : in  std_logic_vector(2 downto 0);
        address_in : in  std_logic_vector(31 downto 0);
        data_in    : in  std_logic_vector(31 downto 0);
        opType     : in  std_logic_vector(1 downto 0);
        data_out   : out std_logic_vector(31 downto 0);
        address_out: out std_logic_vector(31 downto 0);
        IHc        : out std_logic;
        DHc        : out std_logic;
        dbset      : out std_logic;
        trasn_out  : out std_logic
    );
    end component;

    signal cacheType   : std_logic := '0';
    signal stateTrans  : std_logic_vector(2 downto 0) := (others
=>'0');
    signal address_in  : std_logic_vector(31 downto 0) := (others
=>'0');
    signal data_in     : std_logic_vector(31 downto 0) := (others
=>'0');
    signal opType      : std_logic_vector(1 downto 0) := (others
=>'0');
```

```
    signal data_out    : std_logic_vector(31 downto 0);
    signal address_out : std_logic_vector(31 downto 0);
    signal IHc         : std_logic;
    signal DHc         : std_logic;
    signal dbset       : std_logic;
    signal trasn_out   : std_logic;

    begin

        --UUT
        uut: Cache port map(
                cacheType  => cacheType,
                stateTrans    =>    stateTrans,
                address_in   => address_in,
                data_in  => data_in,
                opType    => opType,
                data_out     =>data_out,
                address_out  => address_out,
                IHc    =>    IHc,
                DHc   => DHc,
                dbset  => dbset,
                trasn_out    => trasn_out
                );

        tb: process
        begin



        --Transfer for ICache, Test if the word is correctly written
in his corresponding place of the row.
        --Block, word 0.
        wait for 100 ns;
        cacheType <= '0'; opType<="10";stateTrans<="000"; address_in
<= "00000000000000000000000000000000"; data_in <=
"00000000000000000000000000000000";
        wait for 100 ns;
        --Block, word 1.
        cacheType <= '0'; opType<="10";stateTrans<="001"; address_in
<= "00000000000000000000000000000100"; data_in <=
"00000000000000000000000000000001";
        wait for 100 ns;
        --Block, word 2.
        cacheType <= '0'; opType<="10";stateTrans<="010"; address_in
<= "00000000000000000000000000001000"; data_in <=
"00000000000000000000000000000010";
        wait for 100 ns;
        --Block, word 3.
        cacheType <= '0'; opType<="10";stateTrans<="011"; address_in
<= "00000000000000000000000000001100"; data_in <=
"00000000000000000000000000000011";
        wait for 100 ns;
        --Block, word 4.
        cacheType <= '0'; opType<="10";stateTrans<="100"; address_in
<= "00000000000000000000000000010000"; data_in <=
"00000000000000000000000000000100";
```

```
        wait for 100 ns;
        --Block, word 5.
        cacheType <= '0'; opType<="10";stateTrans<="101"; address_in
<= "00000000000000000000000000010100"; data_in <=
"00000000000000000000000000000101";
        wait for 100 ns;
        --Block, word 6.
        cacheType <= '0'; opType<="10";stateTrans<="110"; address_in
<= "00000000000000000000000000011000"; data_in <=
"00000000000000000000000000000110";
        wait for 100 ns;
        --Block, word 7.
        cacheType <= '0'; opType<="10";stateTrans<="111"; address_in
<= "00000000000000000000000000011100"; data_in <=
"00000000000000000000000000000111";
        wait for 100 ns;


        --Transfer for ICache, Test if the word is correctly written
in his corresponding row. Row = 1.
        --Block, word 0.
        wait for 100 ns;
        cacheType <= '0'; opType<="10";stateTrans<="000"; address_in
<= "00000000000000000000000000100000"; data_in <=
"00000000000000000000000000000000";
        wait for 100 ns;
        --Block, word 1.
        cacheType <= '0'; opType<="10";stateTrans<="001"; address_in
<= "00000000000000000000000000100100"; data_in <=
"00000000000000000000000000000001";
        wait for 100 ns;
        --Block, word 2.
        cacheType <= '0'; opType<="10";stateTrans<="010"; address_in
<= "00000000000000000000000000101000"; data_in <=
"00000000000000000000000000000010";
        wait for 100 ns;
        --Block, word 3.
        cacheType <= '0'; opType<="10";stateTrans<="011"; address_in
<= "00000000000000000000000000101100"; data_in <=
"00000000000000000000000000000011";
        wait for 100 ns;
        --Block, word 4.
        cacheType <= '0'; opType<="10";stateTrans<="100"; address_in
<= "00000000000000000000000000110000"; data_in <=
"00000000000000000000000000000100";
        wait for 100 ns;
        --Block, word 5.
        cacheType <= '0'; opType<="10";stateTrans<="101"; address_in
<= "00000000000000000000000000110100"; data_in <=
"00000000000000000000000000000101";
        wait for 100 ns;
        --Block, word 6.
        cacheType <= '0'; opType<="10";stateTrans<="110"; address_in
<= "00000000000000000000000000111000"; data_in <=
"00000000000000000000000000000110";
        wait for 100 ns;
```

```
        --Block, word 7.
        cacheType <= '0'; opType<="10";stateTrans<="111"; address_in
<= "00000000000000000000000000111100"; data_in <=
"00000000000000000000000000000111";
        wait for 100 ns;


        --Read for ICache, test the Hit case.
        cacheType <= '0'; opType<="00";stateTrans<="011"; address_in
<= "00000000000000000000000000101100";
        wait for 100 ns;

        --Read for ICache, test the miss case.
        cacheType <= '0'; opType<="00";stateTrans<="011"; address_in
<= "00000000000000000000000011101100";
        wait for 100 ns;


            wait;
        end process;
end architecture;
```

CPU code:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity cpu is
    port(
        stateCpu    : in  std_logic;
        ins_in      : in  std_logic_vector(31 downto 0);
        address_in  : in  std_logic_vector(31 downto 0);
        data_in     : in  std_logic_vector(31 downto 0);
        address_out : out std_logic_vector(31 downto 0);
        data_out    : out std_logic_vector(31 downto 0)
   );
end entity;

architecture behav of cpu is

    type regs is array(31 downto 0) of std_logic_vector(31 downto 0);
    signal auxReg : regs:= (others =>(others=>'0'));
    signal opCode  : std_logic_vector(5 downto 0);
    signal funCode : std_logic_vector(5 downto 0);

        begin
        process( stateCpu, ins_in, address_in, data_in)
            variable dirRs   : integer;
            variable dirRt   : integer;
            variable dirRd   : integer;
            variable auxImm : std_logic_vector(31 downto 0);
```

```
            variable auxBran : std_logic_vector(31 downto 0);




            begin

                opCode     <= ins_in(31 downto 26);
                funCode    <= ins_in(5 downto 0);
                dirRs      := conv_integer(ins_in(25 downto 21));
                dirRt      := conv_integer(ins_in(20 downto 16));
                dirRd      := conv_integer(ins_in(15 downto 11));

                if ( stateCPU='0') then
                    -- Add
                    if ((ins_in(31 downto 26) = "000000") and
(ins_in(5 downto 0) = "100000")) then
                        auxReg(dirRd) <= auxReg(dirRs) +
auxReg(dirRt);
                    end if;

                    -- Load
                    if ins_in(31 downto 26) = "100011" then
                        if (ins_in(15)='1') then
                            auxImm := (others =>'1');
                            auxImm(15 downto 0) := ins_in(15 downto
0);
                        else
                            auxImm := (others =>'0');
                            auxImm(15 downto 0)  := ins_in(15 downto
0);
                        end if;
                        address_out <= ( auxReg(conv_integer(ins_in(25
downto 21))) + auxImm);
                    end if;

                    -- Store
                    if ins_in(31 downto 26) = "101011" then
                        if (ins_in(15)='1') then
                            auxImm(31 downto 0) := (others =>'1');
                            auxImm(15 downto 0) := ins_in(15 downto
0);
                        else
                            auxImm(31 downto 0) := (others =>'0');
                            auxImm(15 downto 0) := ins_in(15 downto
0);
                        end if;
                        data_out <= auxReg(conv_integer(ins_in(20
downto 16)));
                        address_out <= (auxReg(conv_integer(ins_in(25
downto 21))) + auxImm);
                    end if;

                    -- Branch
```

```
                    if ins_in(31 downto 26) = "000100" then
                        if (ins_in(15)='1') then
                            auxImm(31 downto 0) := (others =>'1');
                            auxImm(15 downto 0) :=ins_in(15 downto 0);
                        else
                            auxImm(31 downto 0) := (others =>'0');
                            auxImm(15 downto 0) := ins_in(15 downto
0);
                        end if;
                        auxBran(31 downto 2) := auxImm(29 downto 0);
                        auxBran(1 downto 0)  := "00";
                        address_out <= ((address_in +
"00000000000000000000000000000001") + auxBran);
                    end if;

                    -- Sll
                    if ((ins_in(31 downto 26) = "000000") and
(ins_in(5 downto 0) = "000000")) then
                        auxReg(conv_integer(ins_in(15 downto 11))) <=
(others =>'0');
                        auxReg(conv_integer(ins_in(15 downto 11)))(31
downto conv_integer(ins_in(10 downto 6))) <=
auxReg(conv_integer(ins_in(20 downto 16)))((31-conv_integer(ins_in(10
downto 6))) downto 0);
                    end if;

                    -- Nor
                    if ((ins_in(31 downto 26) = "000000") and
(ins_in(5 downto 0) = "010111")) then
                        auxReg(conv_integer(ins_in(15 downto 11))) <=
not( auxReg(conv_integer(ins_in(25 downto 21))) or
auxReg(conv_integer(ins_in(20 downto 16))) );
                    end if;
            --end if;
        end if;
        if(stateCPU='1') then
                case ins_in(31 downto 26) is
                when "000100" =>

                    if (not(auxReg(conv_integer(ins_in(25 downto 21)))
= auxReg(conv_integer(ins_in(20 downto 16))))) then
                        address_out <= (address_in +
"00000000000000000000000000000001");
                    end if;

              when "100011" =>
                        auxReg(dirRt) <= data_in;
                        address_out <= (address_in +
"00000000000000000000000000000001");
              when others => address_out <= (address_in +
"00000000000000000000000000000001");
            end case;
         end if;

        end process;
  end architecture;
```

CPU Test Bench:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity test_cpu is
end entity;

architecture behav of test_cpu is

    --Component Declaration of UUT
    component cpu
    port(
        stateCpu    : in  std_logic;
        ins_in      : in  std_logic_vector(31 downto 0);
        address_in  : in  std_logic_vector(31 downto 0);
        data_in     : in  std_logic_vector(31 downto 0);
        address_out : out std_logic_vector(31 downto 0);
        data_out    : out std_logic_vector(31 downto 0)
    );
    end component;

    signal stateCpu      : std_logic;
    signal ins_in        : std_logic_vector(31 downto 0);
    signal address_in    : std_logic_vector(31 downto 0);
    signal address_out   : std_logic_vector(31 downto 0);
    signal data_in       : std_logic_vector(31 downto 0);
    signal data_out      : std_logic_vector(31 downto 0);

    begin

        --UUT
        uut: cpu port map(
                stateCpu  => stateCpu,
                ins_in    =>    ins_in,
                address_in   => address_in,
                data_in    => data_in,
                address_out  => address_out,
                data_out     =>data_out
                );

        tb: process
        begin


        --load values for add ($t3)
        wait for 100 ns;
        stateCpu<='1'; ins_in <= "10001100001010110000000000000000";
data_in<= "00000000000000000000000001100100";--100
        wait for 100 ns;
        --($t2)
```

```
        stateCPU<='1'; ins_in <= "10001100001010100000000000000000";
data_in<= "00000000000000000000000000010100";--20
        --ins Add
        wait for 100 ns;
        stateCPU <= '0'; ins_in <= "00000001011010101001100000100000";
        wait for 100 ns;
        stateCPU <= '1';address_in <=
"00000000000000000000000000000000";
        wait for 100 ns;

        --ins NOR
        stateCPU <= '0';
        ins_in <= "00000001011010101001100000010111";
        wait for 100 ns;
        stateCPU <= '1';address_in <=
"00000000000000000000000000000011";
        wait for 100 ns;

        --ins SLL
        stateCPU <= '0';
        ins_in <= "00000001011010101001100100000000";
        wait for 100 ns;
        stateCPU <= '1';address_in <=
"00000000000000000000000000000000";
        wait for 100 ns;

        --load values for SW ($t4)
        wait for 100 ns;
        ins_in <= "10001100001011000000000000000000";
        data_in<= "00000000000000000000000001100100";--100
        stateCPU<='1';
        wait for 100 ns;

        --ins SW
        stateCPU <= '0';
        ins_in <= "10101101100100110000000001100100";
        wait for 100 ns;
        stateCPU <= '1';address_in <=
"00000000001100000000000000000000";
        wait for 100 ns;

        --load values for beq ($s5)
        wait for 100 ns;
        ins_in <= "10001100001101010000000000000000";
        data_in<= "00000000000000000000000001100100";--100
        stateCPU<='1';
        wait for 100 ns;
        --($t2)
        ins_in <= "10001100001011100000000000000000";
        data_in<= "00000000000000000000000001100100";--100
        stateCPU<='1';
        wait for 25 ns;

        --ins beq
        wait for 100 ns;
        stateCPU <= '0';
```

```
        address_in <= "00000000000000000000000000000000";
        ins_in <= "00010010101011100000000110010000";
        wait for 100 ns;
        stateCPU <= '1';
        wait for 100 ns;

            wait;
        end process;
end architecture;
```

FSM code:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity fsm is
    port(
        trasn_out : in std_logic;
        reset     : in std_logic;
        clock     : in bit;
        IHc       : in std_logic;
        DHc       : in std_logic;
        cadd      : in std_logic_vector(31 downto 0);
        ins       : in std_logic_vector(31 downto 0);
        pcc       : out std_logic_vector(1 downto 0 );
        ccm       : out std_logic;
        stateCPU  : out std_logic;
        IWrite    : out std_logic;
        IRWrite   : out std_logic;
        madd      : out std_logic_vector(31 downto 0);
        stateTrans: out std_logic_vector(2 downto 0);
        opType    : out std_logic_vector(1 downto 0);
        cacheType : out  std_logic;
        MemWrite  :out   std_logic;
        MemRead   :out    std_logic
    );
end entity;

Architecture behav of fsm is

    type state_type is ( s0,s1,s2a, s2b, s2c, s2d, s2e, s2f, s2g, s2h
                        ,s3,s4,s5,s6,s7,s8a, s8b, s8c, s8d, s8e, s8f,
s8g, s8h
                        ,s9 );
    signal state: state_type;

    begin

        process( clock, reset, trasn_out, IHc, DHc, ins, cadd, state )

            variable opCode: std_logic_vector(5 downto 0);
```

```
begin

    if( reset='1') then
        state <= s0;
    end if;

    case state is
    when s0 => IWrite <= '1'; opType <= "11";
    when s1 =>  cacheType <= '0';IWrite <= '0';
                opType <= "00";
                pcc <= "01";
    when s2a => madd(31 downto 5) <= cadd(31 downto 5);
                madd(4 downto 0) <= "00000";
                cacheType  <= '0';
                opType <= "10";
                stateTrans <= "000";
                MemRead<='1';
                pcc <= "10";
                ccm <= '0';
    when s2b => madd(31 downto 5) <= cadd(31 downto 5);
                    madd(4 downto 0) <= "00001";
                    cacheType  <= '0';
                    opType <= "10";
                    stateTrans <= "001";
                    MemRead<='1';
                    pcc <= "10";
                ccm <= '0';
    when s2c => madd(31 downto 5) <= cadd(31 downto 5);
                    madd(4 downto 0) <= "00010";
                    cacheType  <= '0';
                    opType <= "10";
                    stateTrans <= "010";
                    MemRead<='1';
                    pcc <= "10";
                ccm <= '0';
    when s2d => madd(31 downto 5) <= cadd(31 downto 5);
                    madd(4 downto 0) <= "00011";
                    cacheType  <= '0';
                    opType <= "10";
                    stateTrans <= "011";
                    MemRead<='1';
                    pcc <= "10";
                ccm <= '0';
    when s2e => madd(31 downto 5) <= cadd(31 downto 5);
                    madd(4 downto 0) <= "00100";
                    cacheType  <= '0';
                    opType <= "10";
                    stateTrans <= "100";
                    MemRead<='1';
                    pcc <= "10";
                ccm <= '0';
    when s2f => madd(31 downto 5) <= cadd(31 downto 5);
                    madd(4 downto 0) <= "00101";
                    cacheType  <= '0';
                    opType <= "10";
                    stateTrans <= "101";
```

```
                MemRead<='1';
                pcc <= "10";
            ccm <= '0';
 when s2g => madd(31 downto 5) <= cadd(31 downto 5);
                madd(4 downto 0) <= "00110";
                cacheType  <= '0';
                opType <= "10";
                stateTrans <= "110";
                MemRead<='1';
                pcc <= "10";
            ccm <= '0';
when s2h => madd(31 downto 5) <= cadd(31 downto 5);
                    madd(4 downto 0) <= "00111";
                    cacheType  <= '0';
                    opType <= "10";
                    stateTrans <= "111";
                    MemRead<='1';
                    pcc <= "10";
            ccm <= '0';
when s3 =>  IRWrite <= '1';opType <= "11";
when s4 =>  stateCPU <= '0';opType <= "11";
            opCode := ins(31 downto 26);
when s5 =>  cacheType <= '0';
                opType <= "01";
                pcc <= "00";
            ccm <= '1';
when s6 =>  MemWrite<='1';opType <= "11";
            pcc <= "00";
            ccm <= '1';
when s7 =>  cacheType <= '1';
                opType <= "00";
                pcc <= "00";
when s8a => madd(31 downto 5) <= cadd(31 downto 5);
                madd(4 downto 0) <= "00000";
                cacheType  <= '1';
                opType <= "10";
                stateTrans <= "000";
                MemRead<='1';
                pcc <= "00";
            ccm <= '0';
when s8b => madd(31 downto 5) <= cadd(31 downto 5);
                madd(4 downto 0) <= "00001";
                cacheType  <= '1';
                opType <= "10";
                stateTrans <= "001";
                MemRead<='1';
                pcc <= "00";
            ccm <= '0';
when s8c => madd(31 downto 5) <= cadd(31 downto 5);
                madd(4 downto 0) <= "00010";
                cacheType  <= '1';
                opType <= "10";
                stateTrans <= "010";
                MemRead<='1';
                pcc <= "00";
            ccm <= '0';
```

```
  when s8d => madd(31 downto 5) <= cadd(31 downto 5);
                  madd(4 downto 0) <= "00011";
                  cacheType  <= '1';
                  opType <= "10";
                  stateTrans <= "011";
                  MemRead<='1';
                  pcc <= "00";
              ccm <= '0';
  when s8e => madd(31 downto 5) <= cadd(31 downto 5);
                  madd(4 downto 0) <= "00100";
                  cacheType  <= '1';
                  opType <= "10";
                  stateTrans <= "100";
                  MemRead<='1';
                  pcc <= "00";
              ccm <= '0';
  when s8f => madd(31 downto 5) <= cadd(31 downto 5);
                  madd(4 downto 0) <= "00101";
                  cacheType  <= '1';
                  opType <= "10";
                  stateTrans <= "101";
                  MemRead<='1';
                  pcc <= "00";
              ccm <= '0';
  when s8g => madd(31 downto 5) <= cadd(31 downto 5);
                  madd(4 downto 0) <= "00110";
                  cacheType  <= '1';
                  opType <= "10";
                  stateTrans <= "110";
                  MemRead<='1';
                  pcc <= "00";
              ccm <= '0';
  when s8h => madd(31 downto 5) <= cadd(31 downto 5);
                  madd(4 downto 0) <= "00111";
                  cacheType  <= '1';
                  opType <= "10";
                  stateTrans <= "111";
                  MemRead<='1';
                  pcc <= "00";
              ccm <= '0';
  when s9 => stateCPU <= '1';opType <= "11";
end case;




  if (clock'event and clock='1') then

      case state is

      when s0 => state <= s1;

      when s1 =>  if (IHc='1') then
```

```
                                    state <= s3;
                                else
                                    state <= s2a;
                                end if;

                    when s2a => state <= s2b;
                    when s2b => state <= s2c;
                    when s2c => state <= s2d;
                    when s2d => state <= s2e;
                    when s2e => state <= s2f;
                    when s2f => state <= s2g;
                    when s2g => state <= s2h;
                    when s2h => state <= s1;

                    when s3 =>  state <= s4;

                    when s4 =>  if ( (opCode = "000000") or (opCode =
"000100") ) then
                                state <= s9;
                                end if;
                                if ( opCode = "101011" ) then
                                state <= s5;
                                end if;
                                if ( opCode = "100011" ) then
                                state <= s7;
                                end if;
                    when s5 =>  state <= s6;
                    when s6 =>  state <= s9;
                    when s7 =>  if (DHc='1') then
                                    state <= s9;
                                else
                                    state <= s8a;
                                end if;

                    when s8a => state <= s8b;
                    when s8b => state <= s8c;
                    when s8c => state <= s8d;
                    when s8d => state <= s8e;
                    when s8e => state <= s8f;
                    when s8f => state <= s8g;
                    when s8g => state <= s8h;
                    when s8h => state <= s7;
                    when s9 =>  state <= s0;
                    end case;
            end if;
        end process;
    end architecture;
```

FSM Test Bench:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
entity test_fsm is
end entity;

architecture behav of test_fsm is

    --Component Declaration of UUT
    component fsm
    port(
        trasn_out : in std_logic;
        reset     : in std_logic;
        clock     : in bit;
        IHc       : in std_logic;
        DHc       : in std_logic;
        cadd      : in std_logic_vector(31 downto 0);
        ins       : in std_logic_vector(31 downto 0);
        pcc       : out std_logic_vector(1 downto 0 );
        ccm       : out std_logic;
        stateCPU  : out std_logic;
        IWrite    : out std_logic;
        IRWrite   : out std_logic;
        madd      : out std_logic_vector(31 downto 0);
        stateTrans: out std_logic_vector(2 downto 0);
        opType    : out std_logic_vector(1 downto 0);
        cacheType : out std_logic;
        MemWrite  : out std_logic;
        MemRead   : out std_logic
    );
    end component;

    --Inputs
    signal clock     : bit := '0';
    signal reset     : std_logic := '0';
    signal trasn_out : std_logic := '0';
    signal DHc       : std_logic := '0';
    signal IHc       : std_logic := '0';
    signal cadd      : std_logic_vector(31 downto 0) := (others
=>'0');
    signal ins       : std_logic_vector(31 downto 0) := (others
=>'0');

    --Outputs
    signal pcc          : std_logic_vector(1 downto 0 );
    signal ccm          : std_logic;
    signal stateCPU     : std_logic;
    signal IW           : std_logic;
    signal IRW          : std_logic;
    signal cacheType    : std_logic;
    signal MemWrite     : std_logic;
    signal MemRead      : std_logic;
    signal madd         : std_logic_vector(31 downto 0);
    signal stateTrans   : std_logic_vector(2 downto 0);
    signal opType       : std_logic_vector(1 downto 0);

    begin
```

```
--UUT
uut: fsm port map(
        trasn_out =>trasn_out,
        reset =>reset,
        clock =>clock,
        IHc =>IHc,
        DHc =>DHc,
        cadd =>cadd,
        ins =>ins,
        pcc => pcc,
        ccm =>ccm,
        stateCPU =>stateCPU,
        IWrite =>IW,
        IRWrite =>IRW,
        madd =>madd,
        stateTrans =>stateTrans,
        opType =>opType,
        cacheType =>cacheType,
        MemWrite =>MemWrite,
        MemRead =>MemRead
);

tb: process
begin

-------------R Type----------------
    --s0
    wait for 100 ns;
    clock<='1'; reset<='1';
    wait for 100 ns;
    clock<='0'; reset <='0';
    --s1
    wait for 100 ns;
    clock<='1'; IHc<='0';
    wait for 100 ns;
    clock<='0';
    --s2a
    wait for 100 ns;
    clock<='1';
    wait for 100 ns;
    clock<='0';
    --s2b
    wait for 100 ns;
    clock<='1';
    wait for 100 ns;
    clock<='0';
    --s2c
    wait for 100 ns;
    clock<='1';
    wait for 100 ns;
    clock<='0';
    --s2d
    wait for 100 ns;
    clock<='1';
    wait for 100 ns;
    clock<='0';
```

```
            --s2e
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s2f
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s2g
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s2h
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s1
            wait for 100 ns;
            clock<='1'; IHc <='1';
            wait for 100 ns;
            clock<='0';
            --s3
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s4
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s9
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s0
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';

        -------------Store----------------

            --s0
            wait for 100 ns;
            clock<='1'; reset<='1';
ins<="10101101100100110000000001100100";
            wait for 100 ns;
            clock<='0'; reset <='0';
            --s1
            wait for 100 ns;
```

```
            clock<='1'; IHc<='1';
            wait for 100 ns;
            clock<='0';
            --s3
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s4
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s5
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s6
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s9
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s0
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';

        -------------Load----------------

            --s0
            wait for 100 ns;
            clock<='1'; reset<='1';
ins<="10001100001101010000000000000000";
            wait for 100 ns;
            clock<='0'; reset <='0';
            --s1
            wait for 100 ns;
            clock<='1'; IHc<='1';
            wait for 100 ns;
            clock<='0';
            --s3
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';
            --s4
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
```

```
clock<='0';
--s7
wait for 100 ns;
clock<='1'; DHc <='0';
wait for 100 ns;
clock<='0';
--s8a
wait for 100 ns;
clock<='1';
wait for 100 ns;
clock<='0';
--s8b
wait for 100 ns;
clock<='1';
wait for 100 ns;
clock<='0';
--s8c
wait for 100 ns;
clock<='1';
wait for 100 ns;
clock<='0';
--s8d
wait for 100 ns;
clock<='1';
wait for 100 ns;
clock<='0';
--s8e
wait for 100 ns;
clock<='1';
wait for 100 ns;
clock<='0';
--s8f
wait for 100 ns;
clock<='1';
wait for 100 ns;
clock<='0';
--s8g
wait for 100 ns;
clock<='1';
wait for 100 ns;
clock<='0';
--s8h
wait for 100 ns;
clock<='1';
wait for 100 ns;
clock<='0';
--s7
wait for 100 ns;
clock<='1'; DHc <='1';
wait for 100 ns;
clock<='0';
--s9
wait for 100 ns;
clock<='1';
wait for 100 ns;
clock<='0';
```

```
            --s0
            wait for 100 ns;
            clock<='1';
            wait for 100 ns;
            clock<='0';

            wait;
        end process;
end architecture;
```

Main code:

```
library work;
use work.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity main is
    port(
        rst    : in std_logic;
        IW_in  : in std_logic;
        crtl   : in std_logic;
        MW     : in std_logic;
        MWcrtl : in std_logic;
        stateC : in std_logic;
        stateCC: in std_logic;
        dataC  : in std_logic;
        insC   : in std_logic;
        sc     : in std_logic;
        pc_in  : in std_logic_vector( 31 downto 0);
        data   : in std_logic_vector( 31 downto 0);
        ins    : in std_logic_vector( 31 downto 0)
    );
end entity;

architecture structural of main is

    component cpu is
        port(
        stateCpu    : in  std_logic;
        ins_in      : in  std_logic_vector(31 downto 0);
        address_in  : in  std_logic_vector(31 downto 0);
        data_in     : in  std_logic_vector(31 downto 0);
        address_out : out std_logic_vector(31 downto 0);
        data_out    : out std_logic_vector(31 downto 0)
        );
    end component;

    component Cache is
        port(
        cacheType  : in  std_logic;                    -- Icahe=0;
Dcache=1
```

```
    stateTrans : in  std_logic_vector(2 downto 0);
    address_in : in  std_logic_vector(31 downto 0);
    data_in    : in  std_logic_vector(31 downto 0);
    opType     : in  std_logic_vector(1 downto 0);
    data_out   : out std_logic_vector(31 downto 0);
    address_out: out std_logic_vector(31 downto 0);
    IHc        : out std_logic;
    DHc        : out std_logic;
    dbset      : out std_logic;
    trasn_out  : out std_logic
);
end component;

component clock is
    port(
        clk : inout bit
    );
end component;

component IR is
    port(
    IRWrite        : in std_logic;
    ins_in : in  std_logic_vector(31 downto 0);
    ins_out: out std_logic_vector(31 downto 0)
    );
end component;

component PC is
    port(
    IWrite        : in std_logic;
    address_in : in  std_logic_vector(31 downto 0);
    address_out: out std_logic_vector(31 downto 0)
    );
end component;

component mux3to1 is
    port(
        A : in std_logic_vector(31 downto 0);
        B : in std_logic_vector(31 downto 0);
        C : in std_logic_vector(31 downto 0);
        S : in std_logic_vector(1 downto 0);
        Y : out std_logic_vector(31 downto 0)
    );
end component;

component mux2to1 is
    port(
        A : in std_logic_vector(31 downto 0);
        B : in std_logic_vector(31 downto 0);
        S : in std_logic;
        Y : out std_logic_vector(31 downto 0)
    );
end component;

component Mem is
    port(
```

```
        addressIn : in std_logic_vector(31 downto 0); --address Input
        data_in : in std_logic_vector(31 downto 0);
        MemRead : in std_logic;
        MemWrite : in std_logic;
        data_out : out std_logic_vector(31 downto 0)
        );
    end component;

    component fsm is
        port(
        trasn_out : in std_logic;
        reset     : in std_logic;
        clock     : in bit;
        IHc       : in std_logic;
        DHc       : in std_logic;
        cadd      : in std_logic_vector(31 downto 0);
        ins       : in std_logic_vector(31 downto 0);
        pcc       : out std_logic_vector(1 downto 0 );
        ccm       : out std_logic;
        stateCPU  : out std_logic;
        IWrite    : out std_logic;
        IRWrite   : out std_logic;
        madd      : out std_logic_vector(31 downto 0);
        stateTrans: out std_logic_vector(2 downto 0);
        opType    : out std_logic_vector(1 downto 0);
        cacheType : out  std_logic;
        MemWrite  :out   std_logic;
        MemRead   :out    std_logic
    );
    end component;

    component MUX1 is
        port(
            A : in std_logic;
            B : in std_logic;
            S : in std_logic;
            Y : out std_logic
        );
    end component;

    signal w0, w1, w2, w3,w4, w5, w7, w8, w9, w10, w11, w28, w29 :
std_logic_vector(31 downto 0);
    signal w25, w21 : std_logic_vector(1 downto 0);
    signal w20 : std_logic_vector(2 downto 0);
    signal w13 : bit;
    signal w14, w15, w16, w17, w18, w19, w22, w23, w24, w26, w27, w30,
w6, w31 : std_logic;

    begin

        cache1 : Cache   port map(w22, w20, w7, w10, w21, w4, w8, w14,
w15, w30, w16 );
        mem1   : Mem     port map(w8, w4, w24, w27, w9);
        fsm1   : fsm     port map(w16, rst, w13, w14, w15, w1, w3,
w25, w26, w17, w18, w19, w11, w20, w21, w22, w23, w24);
        cpu1   : cpu     port map(w31, w28, w1, w29, w0, w5);
```

```
        clk1   : clock   port map(w13);
        ir1    : IR      port map(w19, w4, w3);
        pc1    : PC      port map(w6, w2, w1);
        mux    : mux2to1 port map(w5, w9, w26, w10);
        mux2   : mux3to1 port map(w0, w1, w11, w25, w7);
        mux3   : mux2to1 port map(w0, pc_in, sc, w2);
        mux4   : MUX1    port map(w18, IW_in, crtl, w6);
        mux5   : MUX1    port map(w23, MW, MWcrtl, w27);
        mux6   : mux2to1 port map(w3, ins, insC, w28);
        mux7   : mux2to1 port map(w4, data, dataC, w29);
        mux8   : MUX1     port map(w17, stateC, stateCC, w31);

end structural;
```

FSM Test Bench:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity test_main is
end entity;

architecture behav of test_main is

    --Component Declaration of UUT
    component main
    port(
        rst    : in std_logic;
        IW_in  : in std_logic;
        crtl   : in std_logic;
        MW     : in std_logic;
        MWcrtl : in std_logic;
        stateC : in std_logic;
        stateCC: in std_logic;
        dataC  : in std_logic;
        insC   : in std_logic;
        sc     : in std_logic;
        pc_in  : in std_logic_vector( 31 downto 0);
        data   : in std_logic_vector( 31 downto 0);
        ins    : in std_logic_vector( 31 downto 0)
    );
    end component;

    signal rst     : std_logic;
    signal IW_in   : std_logic;
    signal crtl    : std_logic;
    signal MW      : std_logic;
    signal MWcrtl  : std_logic;
    signal stateC  : std_logic;
    signal stateCC : std_logic;
    signal dataC   : std_logic;
    signal insC    : std_logic;
```

```
    signal sc      : std_logic;
    signal pc_in   : std_logic_vector(31 downto 0);
    signal data    : std_logic_vector(31 downto 0);
    signal ins     : std_logic_vector(31 downto 0);

    begin

        --UUT
        uut: main port map(
                rst  => rst,
                IW_in    =>    IW_in,
                crtl   => crtl,
                MW    => MW,
                MWcrtl  => MWcrtl,
                stateC  => stateC,
                stateCC    =>    stateCC,
                dataC   => dataC,
                insC    => insC,
                sc => sc,
                pc_in     =>pc_in,
                data   => data,
                ins    => ins
                );

        tb: process
        begin

            wait for 100 ns;
            MW<='1'; MWcrtl<='0';
            wait for 100 ns;
            MWcrtl<='1';
            wait for 100 ns;
            dataC<='0'; stateCC<='0';insC<='0';
            stateC<='1';ins <= "10001100001010110000000000000000";
data<= "00000000000000000000000001100100";--100
            wait for 100 ns;
            dataC<='0'; stateCC<='0';insC<='0';
            stateC<='1';ins <= "10001100001010100000000000000000";
data<= "00000000000000000000000000010100";--100
            wait for 100 ns;
            dataC<='0'; stateCC<='0';insC<='0';
            stateC<='1';ins <= "10001100001011000000000000000000";
data<= "00000000000000000000000001100100";--100
            wait for 100 ns;
            dataC<='0'; stateCC<='0';insC<='0';
            stateC<='1';ins <= "10001100001101010000000000000000";
data<= "00000000000000000000000001100100";--100
            wait for 100 ns;
            dataC<='0'; stateCC<='0';insC<='0';
            stateC<='1';ins <= "10001100001011100000000000000000";
data<= "00000000000000000000000001100100";--100
            wait for 100 ns;
            dataC<='1'; stateCC<='1';insC<='1';
            wait for 100 ns;
            rst<='1';
```

```
            pc_in <="00000000000000000000000001100100";IW_in
<='1';crtl<='0';sc<='0';
            wait for 10 ns;
            rst <='0';IW_in <='0';
            wait for 250 ns;
            crtl<='1'; sc <='1';
            wait for 100000 ns;
            rst<='1';
            wait for 100 ns;


        wait;
        end process;
end architecture;
```

Other components:

Clock:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity clock is
  port(
    clk : inout bit
  );
end entity;

architecture behav of clock is
  begin

    process
      begin
        clk <= not(clk);
        wait for 100 ns;
    end process;
end architecture;
```

Mux:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity mux is
  port(
    A : in std_logic;
    B : in std_logic;
    S : in std_logic;
    Y : out std_logic
```

```
  );
end entity;

architecture behav of mux is
  begin
    process(A,B,S)
      begin
        if ( S = '1' ) then
          Y <= A;
        else
          Y <= B;
        end if;
    end process;
end architecture;
```

Mux2to1:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity mux2to1 is
  port(
    A : in std_logic_vector(31 downto 0 );
    B : in std_logic_vector(31 downto 0 );
    S : in std_logic;
    Y : out std_logic_vector(31 downto 0)
  );
end entity;

architecture behav of mux2to1 is
  begin
    process(A,B,S)
      begin
        if ( S = '1' ) then
          Y <= A;
        else
          Y <= B;
        end if;
    end process;
end architecture;
```

Mux3to1:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity mux3to1 is
  port(
    A : in std_logic_vector(31 downto 0);
    B : in std_logic_vector(31 downto 0);
    C : in std_logic_vector(31 downto 0);
```

```
    S : in std_logic_vector(1 downto 0);
    Y : out std_logic_vector(31 downto 0)
  );
end entity;

architecture behav of mux3to1 is
  begin
    process(A,B,S)
      begin
        if ( S = "00" ) then
          Y <= A;
        elsif( S="01" ) then
          Y <= B;
        elsif(S="10" ) then
          Y <= C;
        end if;
    end process;
end architecture;
```