Alejandro Gudiño Gallegos
Tarea 12

Código en Mars con burbujas:

```
.text
        add $zero, $zero, $zero
        add $zero, $zero, $zero
        addi $t0, $zero, 5

        add $zero, $zero, $zero
        add $zero, $zero, $zero
        add $zero, $zero, $zero

        add $t1, $t0, $zero

        add $zero, $zero, $zero
        add $zero, $zero, $zero
        add $zero, $zero, $zero

        addi $t1, $t1, 2

        add $zero, $zero, $zero
        add $zero, $zero, $zero
        add $zero, $zero, $zero
        addi $t2, $t1, 3
        add $zero, $zero, $zero
        add $zero, $zero, $zero
        add $zero, $zero, $zero
        lui $1, 4097
        add $zero, $zero, $zero
        add $zero, $zero, $zero
```

```
add $zero, $zero, $zero
ori $1, $1, 0
add $zero, $zero, $zero
add $zero, $zero, $zero
add $zero, $zero, $zero
add $11, $11, $1
add $zero, $zero, $zero
add $zero, $zero, $zero
add $zero, $zero, $zero

sw $t2, 0($t3)
add $zero, $zero, $zero
add $zero, $zero, $zero
add $zero, $zero, $zero
add $s0, $t2, $t1
add $zero, $zero, $zero
add $zero, $zero, $zero
add $zero, $zero, $zero
sub $s1, $s0, $t3
add $zero, $zero, $zero
add $zero, $zero, $zero
add $zero, $zero, $zero
lw $t4, 0($t3)
add $zero, $zero, $zero
add $zero, $zero, $zero
add $zero, $zero, $zero
addi $s2, $t4, -2
add $zero, $zero, $zero
add $zero, $zero, $zero
```

```
add $zero, $zero, $zero
or $s2, $s2, $t5
add $zero, $zero, $zero
add $zero, $zero, $zero
add $zero, $zero, $zero
sll $s7, $s2, 2

exit:
```

Bloques modificados de código

Primero se agregaron los wires necesarios para que el pipeline pudiera funcionar, dividiendo en secciones dependiendo de la parte a la que pertenecieran.

```
//////////PIPELINE FETCH_DECODE WIRES//////
    wire [31:0] ID_instruction_bus_wire;
    wire [31:0] ID_pc_plus_4_wire;
//////////////PIPELINE DECODE_EXECUTE/////////////
    wire EX_wMemtoReg;
    wire EX_reg_dst_wire;
    wire [2:0] EX_aluop_wire;
    wire EX_alu_src_wire;
    wire EX_reg_write_wire;
    wire EX_wMemWrite;
    wire EX_wMemRead;
    wire [31:0] EX_pc_plus_4_wire;
    wire [31:0] EX_read_data_1_wire;
    wire [31:0] EX_read_data_2_wire;
    wire [31:0] EX_Inmmediate_extend_wire;
    wire [31:0] EX_instruction_bus_wire;
```

```
//////////////////////PIPELINE EXECUTE_MEMORY ACCESS////////////////
        wire [4:0] MEM_write_register_wire;
        wire [31:0] MEM_read_data_2_wire;
        wire MEM_wMemWrite;
        wire MEM_wMemRead;
        wire MEM_wMemtoReg;
        wire [31:0] MEM_pc_plus_4_wire;
        wire MEM_reg_write_wire;
        wire [31:0] MEM_alu_result_wire;
//////////////////////////PIPELINE MEMORY ACCESS _ WRITE BACK/////////////////
        wire [4:0] WB_write_register_wire;
        wire WB_wMemtoReg;
        wire [31:0] WB_pc_plus_4_wire;
        wire [31:0] WB_wReadData;
        wire WB_reg_write_wire;
        wire [31:0] WB_alu_result_wire;
```

Después de agregar las secciones necesarias, se hizo el archivo en verilog del pipeline, el cual se muestra a continuación. Para la realización de este archivo se tomo como base el register file ya realizado antes.

```
module PipelineRegister
#(
        parameter N=32,
        parameter start=0
)
(
        input clk,
        input enable,
        input reset,
        input  [N-1:0] DataInput,
```

```
        output reg [N-1:0] DataOutput
);
        always@(negedge reset or negedge clk) begin
                if(reset==0)
                        DataOutput <=start;
                else
                        if(enable==1)
                                DataOutput<=DataInput;
        end


endmodule
```

Después de crear el archivo del pipeline, se conectaron los wires necesarios para que funcionara. Dichos bloques se muestran a continuación.

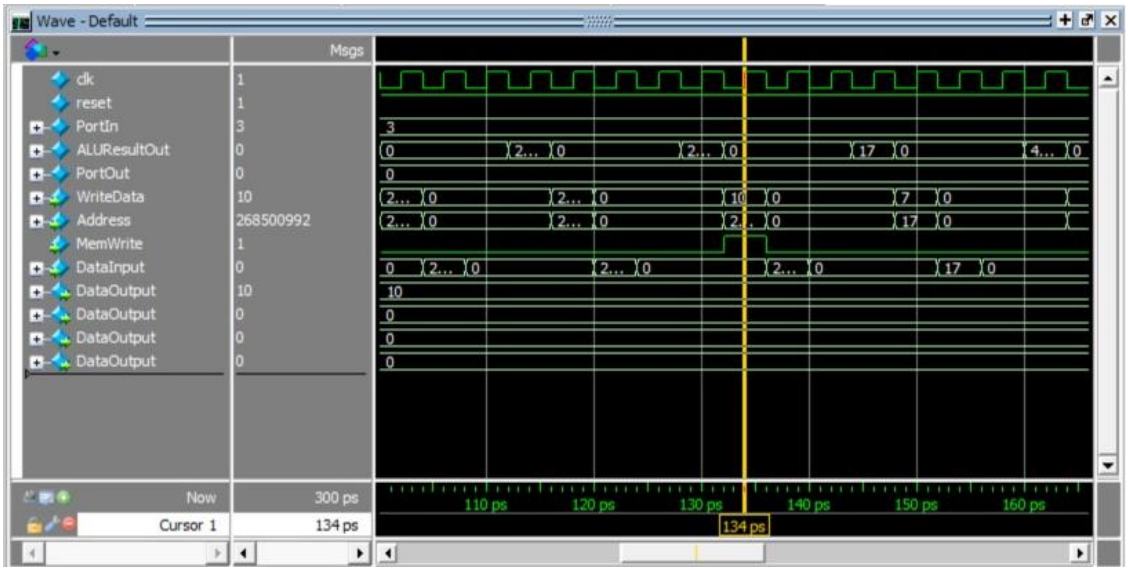
```
//////////////////////////PIPELINE_IF_ID//////////////////////////////
        PipelineRegister
        #(
                .N(32)
        )
        IF_ID_Pipeline(
                .clk(clk),
                .enable(1),
                .reset(reset),
                .DataInput(instruction_bus_wire),
                .DataOutput(ID_instruction_bus_wire)
        );
//////////////////////////////////PIPELINE DECODE_EXECUTE ////////////////////////
        PipelineRegister
        #(
```

```
        .N(137)//174
)
ID_EX_Pipeline(
        .clk(clk),
        .enable(1),
        .reset(reset),
        .DataInput({
                                        wMemtoReg,
                                        reg_dst_wire,
                                        aluop_wire,
                                        alu_src_wire,
                                        reg_write_wire,
                                        wMemWrite,
                                        wMemRead,
                                        read_data_1_wire,
                                        read_data_2_wire,
                                        Inmmediate_extend_wire,
                                        ID_instruction_bus_wire
                                        }),
        .DataOutput({
                                        EX_wMemtoReg,
                                        EX_reg_dst_wire,
                                        EX_aluop_wire,
                                        EX_alu_src_wire,
                                        EX_reg_write_wire,
                                        EX_wMemWrite,
                                        EX_wMemRead,
                                        EX_read_data_1_wire,
                                        EX_read_data_2_wire,
```

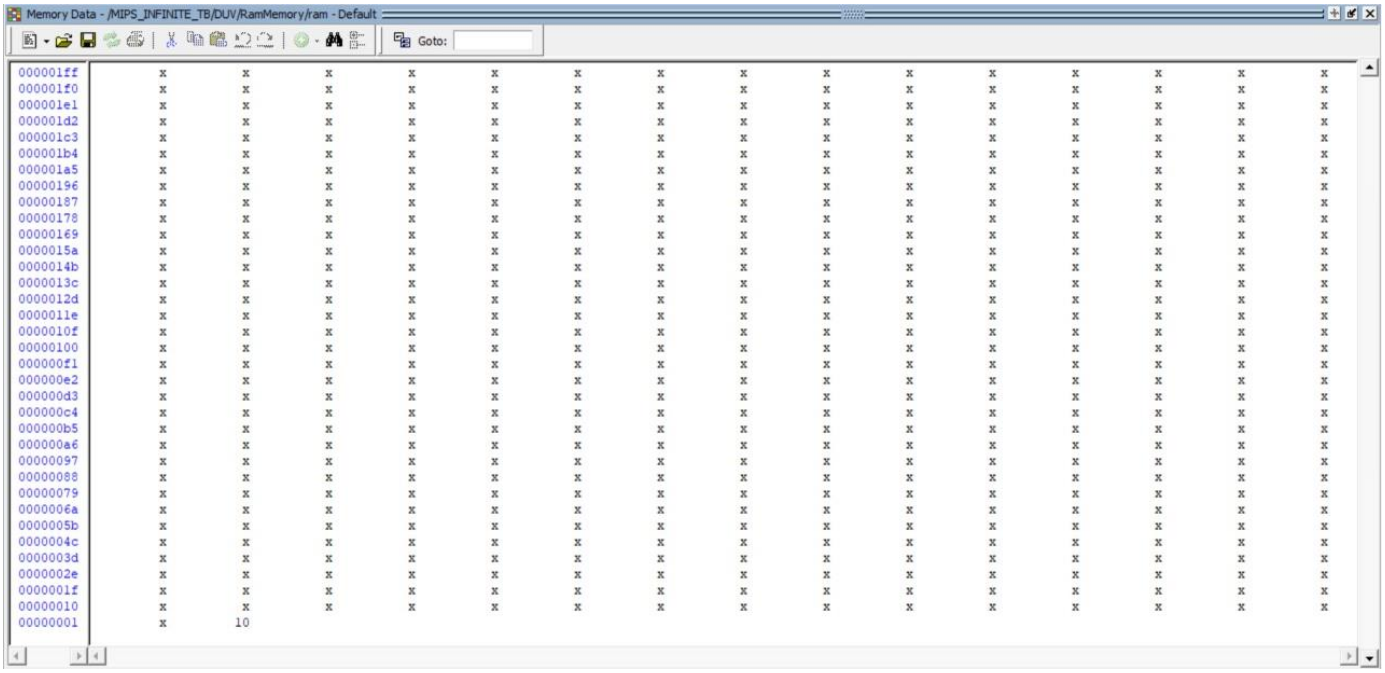
```
                                        EX_Inmmediate_extend_wire,
                                        EX_instruction_bus_wire
                                        })
        );
////////////////////////////////////// PIPELINE EXECUTE_MEMORY ACCESS//////////////////
        PipelineRegister
        #(
                .N(73)//202
        )
        EX_MEM_Pipeline(
                .clk(clk),
                .enable(1),
                .reset(reset),
                .DataInput({
                                        write_register_wire,
                                        EX_read_data_2_wire,
                                        EX_wMemWrite,
                                        EX_wMemRead,
                                        EX_wMemtoReg,
                                        EX_reg_write_wire,
                                        alu_result_wire
                                        }),
                .DataOutput({
                                        MEM_write_register_wire,
                                        MEM_read_data_2_wire,
                                        MEM_wMemWrite,
                                        MEM_wMemRead,
                                        MEM_wMemtoReg,
                                        MEM_reg_write_wire,
```

```
                                        MEM_alu_result_wire
                                        })
        );
//////////////////////////////////////PIPELINE MEMORY ACCES _ WRITE BACK/////////////////
PipelineRegister
#(
        .N(71)//198
)
MEM_WB_Pipeline(
        .clk(clk),
        .enable(1),
        .reset(reset),
        .DataInput({
                                        MEM_write_register_wire,
                                        MEM_wMemtoReg,
                                        wReadData,
                                        MEM_reg_write_wire,
                                        MEM_alu_result_wire
                                        }),
        .DataOutput({
                                        WB_write_register_wire,
                                        WB_wMemtoReg,
                                        WB_wReadData,
                                        WB_reg_write_wire,
                                        WB_alu_result_wire
                                        })
);
```

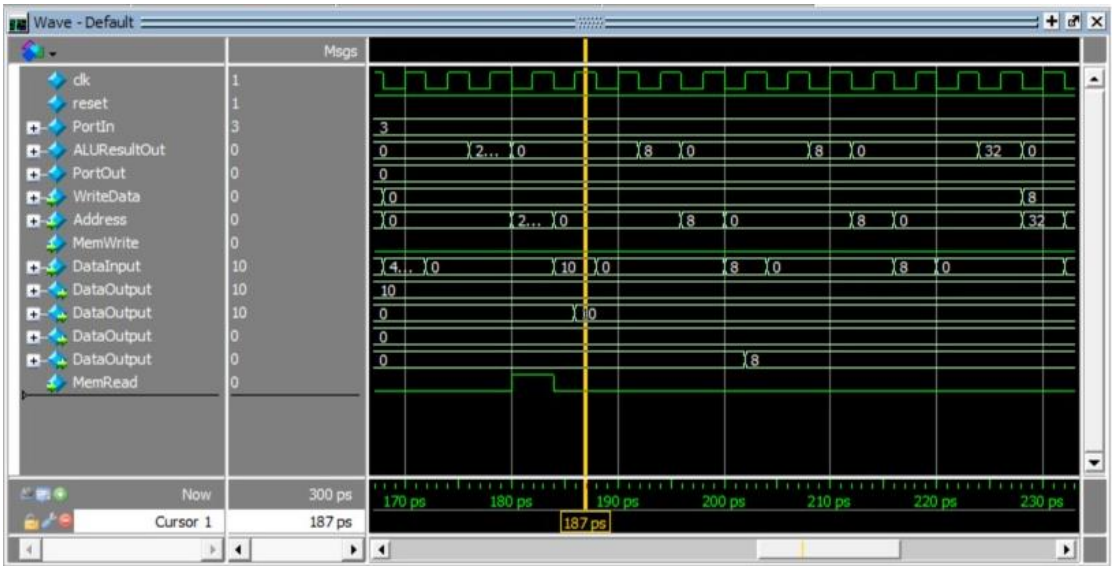
También se modificaron los wires de algunos bloques para que todo el código pudiera funcionar sin problemas. A continuación, se adjuntan imágenes de simulaciones para comprobar el funcionamiento.



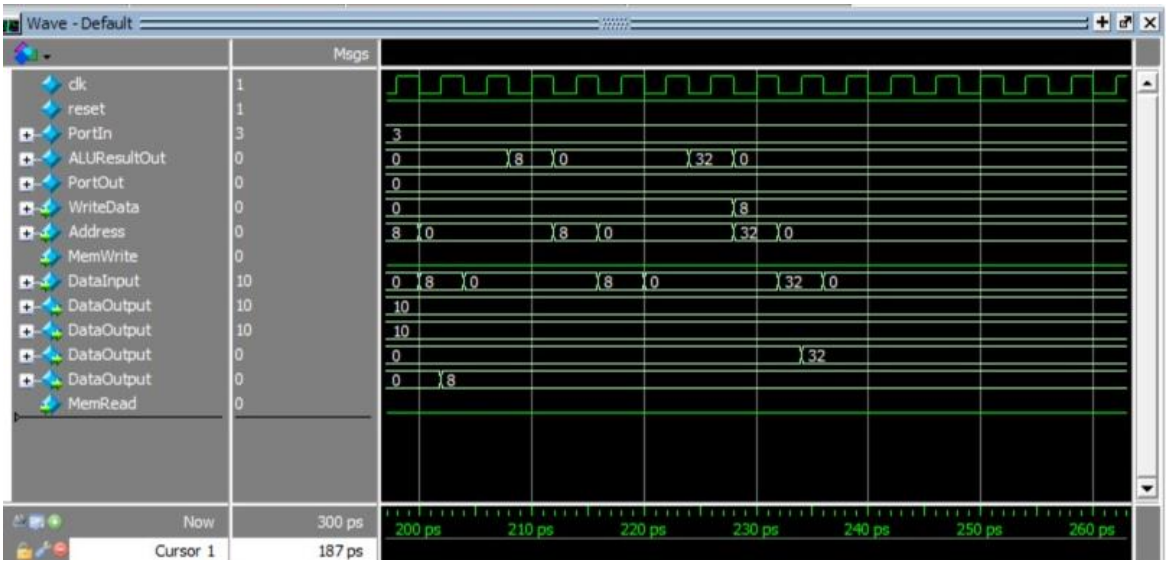
En la primera imagen se puede observar que las operaciones se hacen bien y da como resultado un 10, mismo 10 que se quiere escribir en memoria. Se puede observar que si se tiene la dirección y el dato deseados a guardar.



Aquí se puede observar el 10 ya escrito en memoria.



Aquí se puede observar que ahora se quiere leer de la memoria, la señal que se necesita esta activa y el 10 aparece en el registro seleccionado, mismo 10 que estaba en memoria.



En la ultima se pueden observar los datos ya guardados en los registros la terminar el programa.