Tarea 12: implementación pipeline

Arquitectura Computacional | Pedro Saldaña Zepeda | O. 2019

Eduardo Ethandrake Castillo Pulido
Ie714410@iteso.mx

Para esta tarea se creó el modulo PipelineRegister el cual es parecido a uno de Register aunque con el cambio del flanco del reloj por uno negedge.

```
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
```

Figura 1 Modulo Pipeline

Posteriormente se continuó a instancias el modulo creado en la figura 1 para cada etapa de Pipeline, teniendo al final 4 instancias, IF_ID, ID_EX, EX_MEM y MEM_WB. A continuación se muestran los wires declarados para su funcionamiento. Para la resolución de esta tarea se tomaron en cuenta las fases que toma la información y control dentro del MIPS, las cuales son:

- Instruction fetch
- Instruction decode / register file read
- Execute / address calculation
- Memory Access
- Write Back

```
wire [31:0] PipeID_instruction_bus_wire;
wire [31:0] PipeID_pc_plus_4_wire;

wire PipeEX_reg_write_wire;
wire PipeEX_branch_ne_wire;
wire PipeEX_branch_eq_wire;
wire [2:0] PipeEX_aluop_wire;
wire PipeEX_alu_src_wire;
wire PipeEX_wMemRead;
wire PipeEX_wMemtoReg;
wire PipeEX_wMemWrite;
wire [31:0] PipeEX_read_data_1_wire;
wire [31:0] PipeEX_read_data_2_wire;
wire [31:0] PipeEX_Inmmediate_extend_wire;
wire [31:0] PipeEX_instruction_bus_wire;
wire PipeEX_reg_dst_wire;

wire PipeMEM_reg_write_wire;
wire PipeMEM_wMemRead;
wire PipeMEM_wMemtoReg;
wire PipeMEM_wMemWrite;
wire [31:0] PipeMEM_instruction_bus_wire;
wire [31:0] PipeMEM_alu_result_wire;
wire [31:0] PipeMEM_read_data_2_wire;
wire [4:0] PipeMEM_write_register_wire;

wire PipeWB_reg_write_wire;
wire PipeWB_wMemtoReg;
wire [31:0] PipeWB_instruction_bus_wire;
wire [31:0] PipeWB_wReadData;
wire [31:0] PipeWB_alu_result_wire;
wire [4:0] PipeWB_write_register_wire;
```

Figura 2 Wires declarados para el funcionamiento de los Pipelines

```
PipelineRegister
#(
    .N(64),
    .start(0)
)
IF_ID_Pipeline(
    .clk(clk),
    .enable(1),
    .reset(reset),
    .DataInput({instruction_bus_wire,pc_plus_4_wire}),
    .DataOutput({PipeID_instruction_bus_wire,PipeID_pc_plus_4_wire})

);
```

Figura 3 Declaración del Pipeline IF_ID, junto con sus entradas y salidas

```
PipelineRegister
#(
.N(139),
.start(0)
)
ID_EX_Pipeline(
.clk(clk),
.enable(1),
.reset(reset),
.DataInput({reg_write_wire,branch_ne_wire,branch_eq_wire,aluop_wire,alu_src_w
ire,wMemRead,wMemtoReg,wMemWrite,read_data_1_wire,read_data_2_wire,Inmmediate
_extend_wire,PipeID_instruction_bus_wire,reg_dst_wire}),
.DataOutput({PipeEX_reg_write_wire,PipeEX_branch_ne_wire,PipeEX_branch_eq_wir
e,PipeEX_aluop_wire,PipeEX_alu_src_wire,PipeEX_wMemRead,PipeEX_wMemtoReg,Pipe
EX_wMemWrite,PipeEX_read_data_1_wire,PipeEX_read_data_2_wire,PipeEX_Inmmediat
e_extend_wire,PipeEX_instruction_bus_wire,PipeEX_reg_dst_wire})
);
```

Figura 4 Declaración del Pipeline ID_EX junto con sus entradas y salidas

```
PipelineRegister
#(
.N(105),
.start(0)
)
EX_MEM_Pipeline(
.clk(clk),
.enable(1),
.reset(reset),
.DataInput({PipeEX_reg_write_wire,PipeEX_wMemRead,PipeEX_wMemtoReg,PipeEX_wMe
mWrite,PipeEX_instruction_bus_wire,alu_result_wire,PipeEX_read_data_2_wire,wr
ite_register_wire}),
.DataOutput({PipeMEM_reg_write_wire,PipeMEM_wMemRead,PipeMEM_wMemtoReg,PipeME
M_wMemWrite,PipeMEM_instruction_bus_wire,PipeMEM_alu_result_wire,PipeMEM_read
_data_2_wire,PipeMEM_write_register_wire})
);
```

Figura 5 Declaración del Pipeline EX_MEM con sus entradas y salidas

```
PipelineRegister
#(
.N(103),
.start(0)
)
MEM_WB_Pipeline(
.clk(clk),
.enable(1),
.reset(reset),
.DataInput({PipeMEM_reg_write_wire,PipeMEM_wMemtoReg,PipeMEM_instruction_bus_
wire,wReadData,PipeMEM_alu_result_wire,PipeMEM_write_register_wire}),
.DataOutput({PipeWB_reg_write_wire,PipeWB_wMemtoReg,PipeWB_instruction_bus_wi
re,PipeWB_wReadData,PipeWB_alu_result_wire,PipeWB_write_register_wire})
);
```

Figura 6 Declaracion del Pipeline MEM_WB con sus entradas y salidas

Una vez declarados los Pipelines junto con sus wires de entrada y de salida se actualizaron los módulos que se encontraban entre los bancos de registros del pipeline.

```
Control
ControlUnit
(
    .OP(PipeID_instruction_bus_wire[31:26]),//From Pipeline ID
    .FUN(PipeID_instruction_bus_wire[5:0]),//From Pipeline ID
    .RegDst(reg_dst_wire),
    .BranchNE(branch_ne_wire),
    .BranchEQ(branch_eq_wire),
    .ALUOp(aluop_wire),
    .ALUSrc(alu_src_wire),
    .RegWrite(reg_write_wire),
    .MemWrite(wMemWrite),
    .MemRead(wMemRead),
    .MemtoReg(wMemtoReg),
    .Jump(wJump),
    .Jump_R(wJump_R),
    .JAL(wJAL)
);

RegisterFile
Register_File
(
    .clk(clk),
    .reset(reset),
    .RegWrite(PipeWB_reg_write_wire),////From Pipeline WB
    .WriteRegister(PipeWB_write_register_wire),//From Pipeline WB
    .ReadRegister1(PipeID_instruction_bus_wire[25:21]),//From Pipeline ID
    .ReadRegister2(PipeID_instruction_bus_wire[20:16]),//From Pipeline ID
    .WriteData(wRamAluMux),
    .ReadData1(read_data_1_wire),
    .ReadData2(read_data_2_wire)

);

SignExtend
SignExtendForConstants
(
    .DataInput(PipeID_instruction_bus_wire[15:0]),//From Pipeline ID
    .SignExtendOutput(Inmmediate_extend_wire)
);
```

```
Multiplexer2to1
#(
    .NBits(5)
)
MUX_ForRTypeAndIType
(
    .Selector(PipeEX_reg_dst_wire),//From Pipeline EX
    .MUX_Data0(PipeEX_instruction_bus_wire[20:16]),//From Pipeline EX
    .MUX_Data1(PipeEX_instruction_bus_wire[15:11]),//From Pipeline EX

    .MUX_Output(write_register_wire)

);


Multiplexer2to1
#(
    .NBits(32)
)
MUX_ForReadDataAndInmediate
(
    .Selector(PipeEX_alu_src_wire),//From Pipeline EX
    .MUX_Data0(PipeEX_read_data_2_wire),//From Pipeline EX
    .MUX_Data1(PipeEX_Inmmediate_extend_wire),//From Pipeline EX

    .MUX_Output(read_data_2_orr_inmmediate_wire)

);

ALUControl
ArithmeticLogicUnitControl
(
    .ALUOp(PipeEX_aluop_wire),//From Pipeline EX
    .ALUFunction(PipeEX_instruction_bus_wire[5:0]),//From Pipeline EX
    .ALUOperation(alu_operation_wire)

);
ALU
ArithmeticLogicUnit
(
    .shamt(PipeEX_instruction_bus_wire[10:6]),//From Pipeline EX
    .ALUOperation(alu_operation_wire),
    .A(PipeEX_read_data_1_wire),//From Pipeline EX
    .B(read_data_2_orr_inmmediate_wire),
    .Zero(zero_wire),
    .ALUResult(alu_result_wire)
);
```

```
DataMemory
#( .DATA_WIDTH(32),
   .MEMORY_DEPTH(256)
)
RamMemory
(
   .WriteData(PipeMEM_read_data_2_wire),//From Pipeline MEM
   .Address(PipeMEM_alu_result_wire),//From Pipeline MEM
   .MemWrite(PipeMEM_wMemWrite),//From Pipeline MEM
   .MemRead(PipeMEM_wMemRead),//From Pipeline MEM
   .clk(clk),
   .ReadData(wReadData)
);

Multiplexer2to1
#(
   .NBits(32)
)
MUX_ForAluAndRamMemory
(
   .Selector(PipeWB_wMemtoReg),//From Pipeline WB
   .MUX_Data0(PipeWB_alu_result_wire),//From Pipeline WB
   .MUX_Data1(PipeWB_wReadData),//From Pipeline WB
   .MUX_Output(wRamAluMux)
);
```

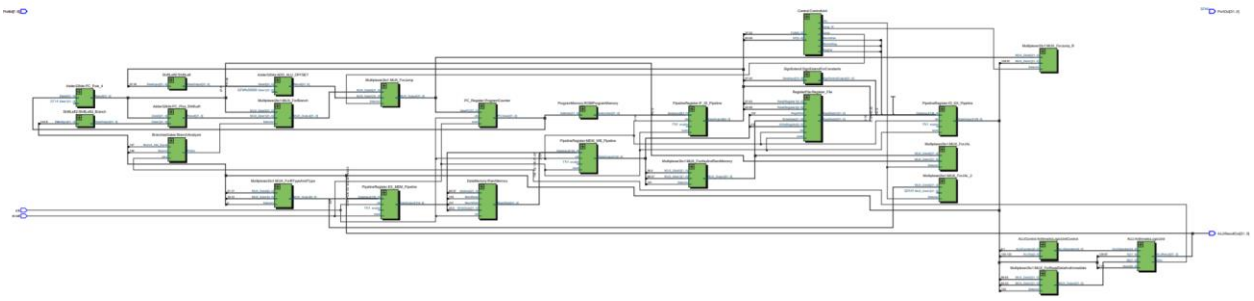

Figura 7 Diagrama RTL

A continuación se presenta el código en Mars utilizado incluyendo las burbujas necesarias para evitar los hazards

```
.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:
```

Finalmente, como se puede observar a continuación dentro de la figura 8 hasta la 11, el código implementado dentro de Mars corrió de la manera deseada dentro del MIPS con pipelines incluidos. Además, previamente se tuvo que modificar el código

proporcionado debido a la dependencia de datos y se cambio la línea de código (addi $t3, $t3, 0x010010000) por sus tres líneas equivalentes :

```
lui $1, 4097
ori $1, $1, 0
add $11, $11, $1
```

Esto debido a que existía una dependencia de datos entre las tres instrucciones previas y era necesario implementar también burbujas.

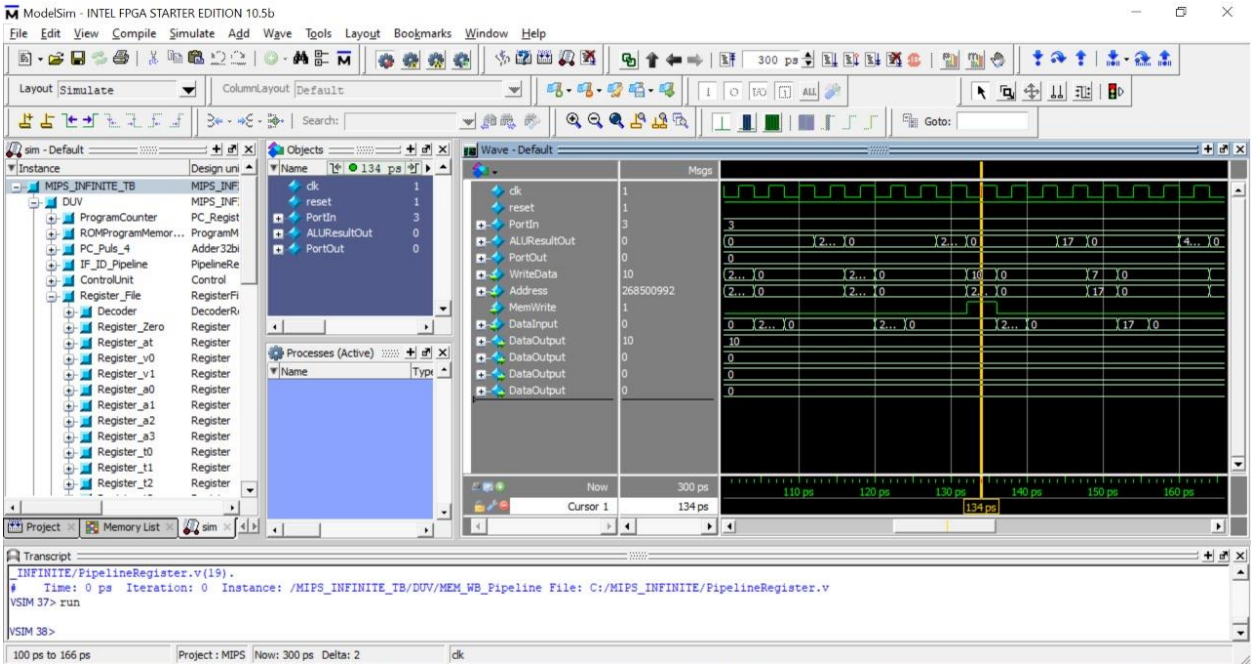

Figura 8 Ejecución del codigo en ModelSim

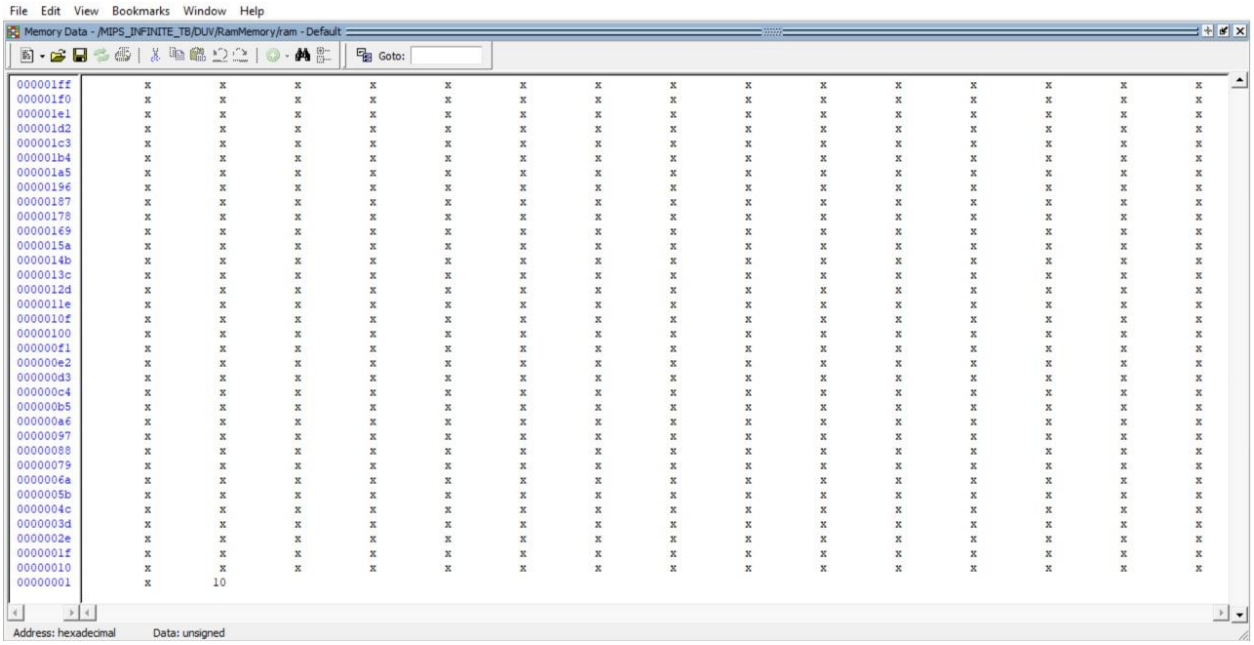

Figura 9 Vista de memoria RAM con el dato esccrito en la localidad 0

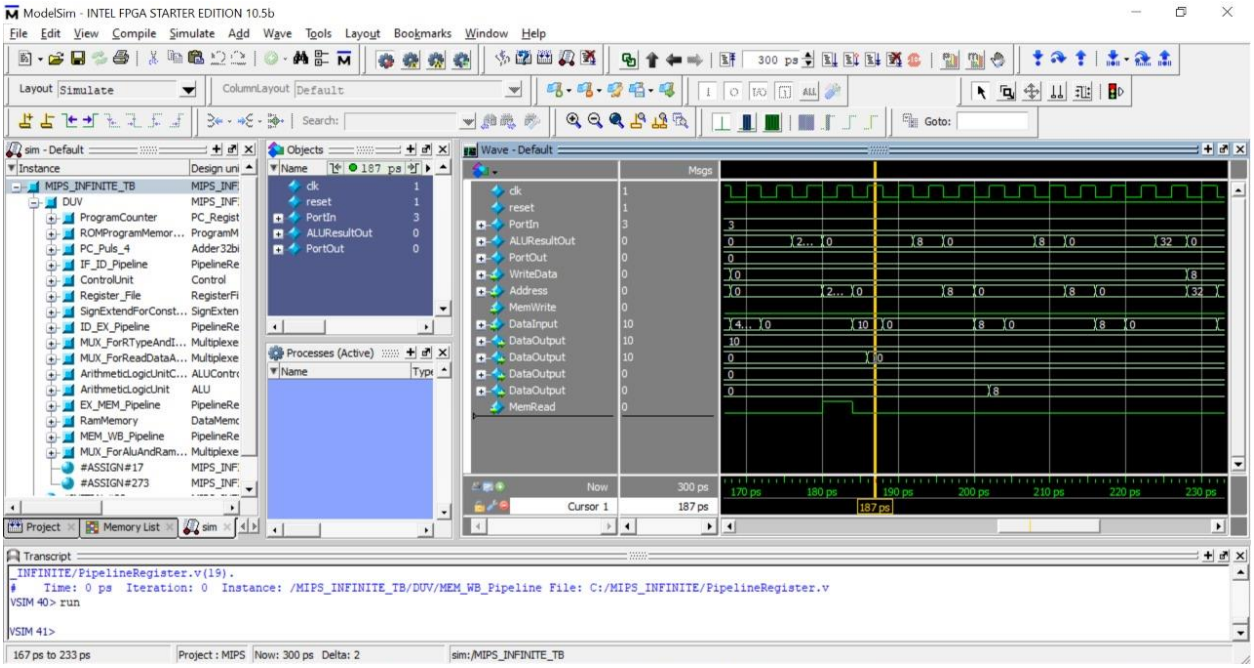

Figura 10 Simulacion del codigo mostrando que funciona el lw

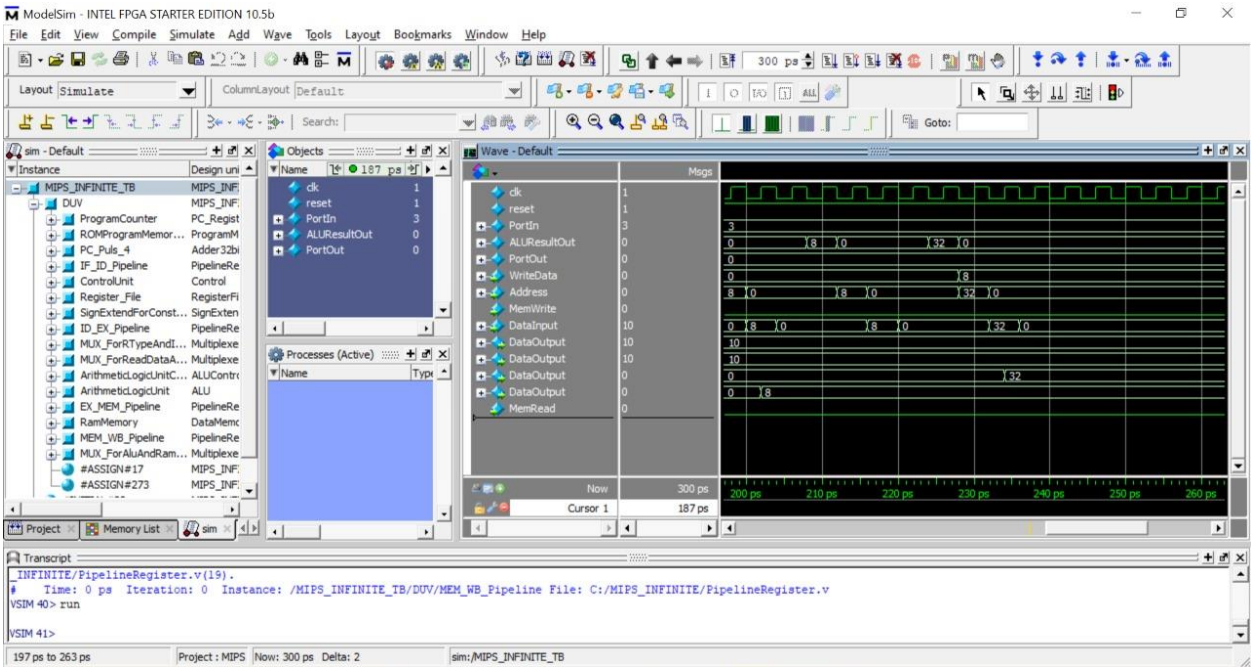

Figura 11 Resto del codigo en mars ejecutandose dentro de la simulacion