ITESO

Universidad Jesuita de Guadalajara

Design and simulation of a pipeline processor based on the MIPS architecture

Raúl Méndez Álvarez

7/19/19

Arquitectura Computacional

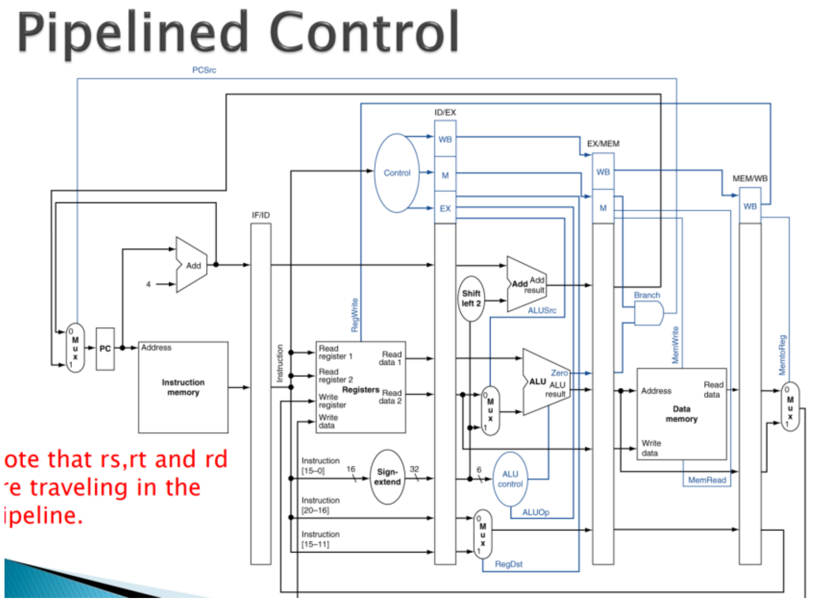
Rodrigo Aldana López

Repository: https://github.com/RaulMendezA/Practica-3---AC

1 – Design changes and process:

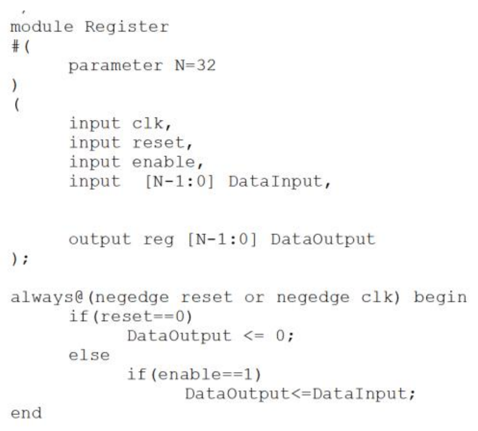
For the pipeline, I first had to create some sort of internal language or system in order to name each of the new components for each pipeline. For example: ID_Instruction_wire_EX, both for entries and exits, in this way it is easier to realize where exactly is a certain cable; basically, all of the new component had the cable or module type in the middle and its connecting components on both sides.

I also made the decision to only pass the fundamental cables through the pipeline so that the forward unit could work. For this, the diagram provided in the presentation "Digital design with verilog HDL" was followed since using a different approach would be a bit confusing.


Pipelined Control

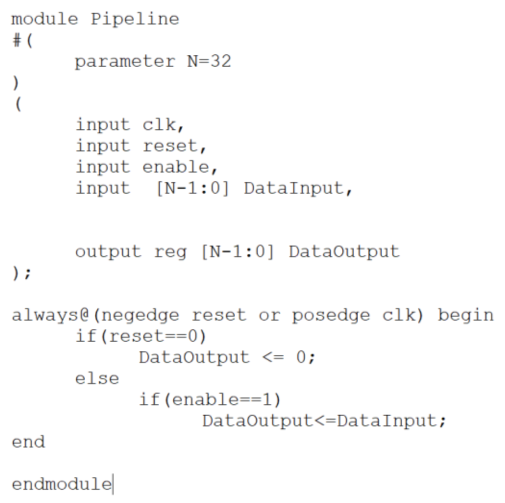
ote that rs,rt and rd
re traveling in the
ipeline.

In order to begin migrating my original sincle-cycle model to a pipelined/multi-cycle approach, I had to change the register from posedge to negedge:



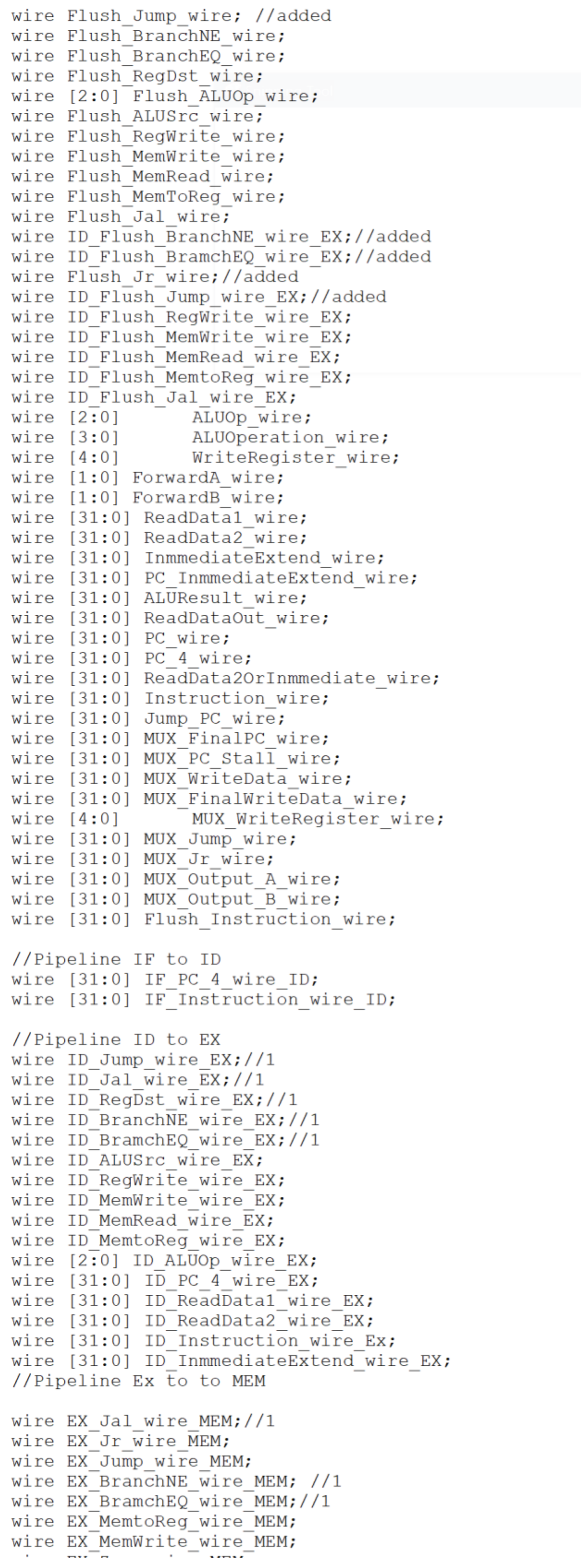
```
module Register
#(
      parameter N=32
)
(
      input clk,
      input reset,
      input enable,
      input  [N-1:0] DataInput,


      output reg [N-1:0] DataOutput
);

always@(negedge reset or negedge clk) begin
      if(reset==0)
            DataOutput <= 0;
      else
            if(enable==1)
                  DataOutput<=DataInput;
end
```

Whereas the new pipeline module created would then be updating on posedge:



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


      output reg [N-1:0] DataOutput
);

always@(negedge reset or posedge clk) begin
      if(reset==0)
            DataOutput <= 0;
      else
            if(enable==1)
                  DataOutput<=DataInput;
end

endmodule
```

I created the new pipeline registers as well as instantiated their respective wires:



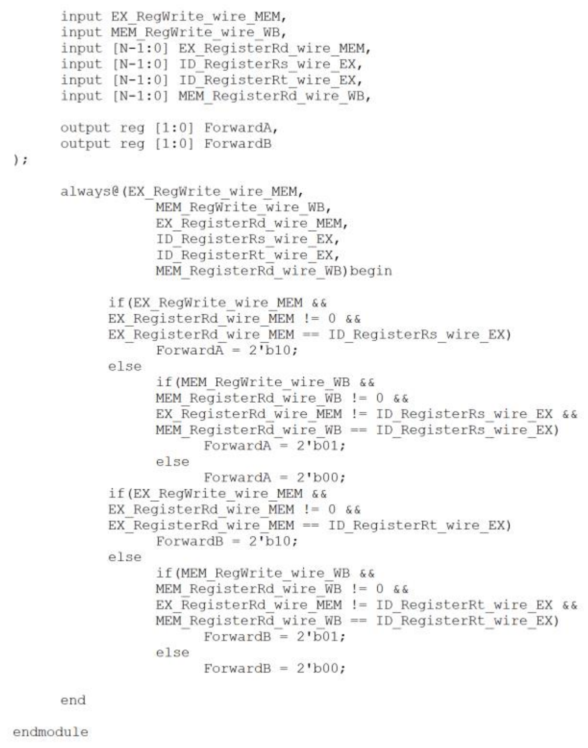
```
wire Flush_Jump_wire; //added
wire Flush_BranchNE_wire;
wire Flush_BranchEQ_wire;
wire Flush_RegDst_wire;
wire [2:0] Flush_ALUOp_wire;
wire Flush_ALUSrc_wire;
wire Flush_RegWrite_wire;
wire Flush_MemWrite_wire;
wire Flush_MemRead_wire;
wire Flush_MemToReg_wire;
wire Flush_Jal_wire;
wire ID_Flush_BranchNE_wire_EX;//added
wire ID_Flush_BramchEQ_wire_EX;//added
wire Flush_Jr_wire;//added
wire ID_Flush_Jump_wire_EX;//added
wire ID_Flush_RegWrite_wire_EX;
wire ID_Flush_MemWrite_wire_EX;
wire ID_Flush_MemRead_wire_EX;
wire ID_Flush_MemtoReg_wire_EX;
wire ID_Flush_Jal_wire_EX;
wire [2:0]      ALUOp_wire;
wire [3:0]      ALUOperation_wire;
wire [4:0]      WriteRegister_wire;
wire [1:0] ForwardA_wire;
wire [1:0] ForwardB_wire;
wire [31:0] ReadData1_wire;
wire [31:0] ReadData2_wire;
wire [31:0] InmmediateExtend_wire;
wire [31:0] PC_InmmediateExtend_wire;
wire [31:0] ALUResult_wire;
wire [31:0] ReadDataOut_wire;
wire [31:0] PC_wire;
wire [31:0] PC_4_wire;
wire [31:0] ReadData2OrInmmediate_wire;
wire [31:0] Instruction_wire;
wire [31:0] Jump_PC_wire;
wire [31:0] MUX_FinalPC_wire;
wire [31:0] MUX_PC_Stall_wire;
wire [31:0] MUX_WriteData_wire;
wire [31:0] MUX_FinalWriteData_wire;
wire [4:0]      MUX_WriteRegister_wire;
wire [31:0] MUX_Jump_wire;
wire [31:0] MUX_Jr_wire;
wire [31:0] MUX_Output_A_wire;
wire [31:0] MUX_Output_B_wire;
wire [31:0] Flush_Instruction_wire;

//Pipeline IF to ID
wire [31:0] IF_PC_4_wire_ID;
wire [31:0] IF_Instruction_wire_ID;

//Pipeline ID to EX
wire ID_Jump_wire_EX;//1
wire ID_Jal_wire_EX;//1
wire ID_RegDst_wire_EX;//1
wire ID_BranchNE_wire_EX;//1
wire ID_BramchEQ_wire_EX;//1
wire ID_ALUSrc_wire_EX;
wire ID_RegWrite_wire_EX;
wire ID_MemWrite_wire_EX;
wire ID_MemRead_wire_EX;
wire ID_MemtoReg_wire_EX;
wire [2:0] ID_ALUOp_wire_EX;
wire [31:0] ID_PC_4_wire_EX;
wire [31:0] ID_ReadData1_wire_EX;
wire [31:0] ID_ReadData2_wire_EX;
wire [31:0] ID_Instruction_wire_Ex;
wire [31:0] ID_InmmediateExtend_wire_EX;
//Pipeline Ex to to MEM

wire EX_Jal_wire_MEM;//1
wire EX_Jr_wire_MEM;
wire EX_Jump_wire_MEM;
wire EX_BranchNE_wire_MEM; //1
wire EX_BramchEQ_wire_MEM;//1
wire EX_MemtoReg_wire_MEM;
wire EX_MemWrite_wire_MEM;
```

```
Pipeline
#(
      .N(143)
)
EX_Pipeline_MEM
(
      .clk(clk),
      .reset(reset),
      .enable(1),
      .DataInput({
                              ID_ReadData1_wire_EX,
                              ID_Flush_Jal_wire_EX,
                              ID_PC_4_wire_EX,
                              ID_Flush_BranchNE_wire_EX,//added
                              ID_Flush_BramchEQ_wire_EX,//added
                              Flush_Jr_wire,//added
                              ID_Flush_Jump_wire_EX,//added
                              ID_Flush_RegWrite_wire_EX,
                              ID_Flush_MemWrite_wire_EX,
                              ID_Flush_MemRead_wire_EX,
                              ID_Flush_MemtoReg_wire_EX,
                              Zero_wire,
                              ALUResult_wire,
                              MUX_Output_B_wire,
                              WriteRegister_wire
                              }),
      .DataOutput({
                              EX_ReadData1_wire_MEM,
                              EX_Jal_wire_MEM,
                              EX_PC_4_wire_MEM,
                              EX_BranchNE_wire_MEM, //1
                              EX_BramchEQ_wire_MEM,//1
                              EX_Jr_wire_MEM,
                              EX_Jump_wire_MEM,
                              EX_RegWrite_wire_MEM,
                              EX_MemWrite_wire_MEM,
                              EX_MemRead_wire_MEM,
                              EX_MemtoReg_wire_MEM,
                              EX_Zero_wire_MEM,
                              EX_ALUResult_wire_MEM,
                              EX_ReadData2_wire_MEM,
                              EX_MUX_ForRTypeAndIType_wire_MEM
                              })
);

Pipeline
      #(
            .N(104)
      )
MEM_Pipeline_WB
(
            .clk(clk),
            .reset(reset),
            .enable(1),
            .DataInput({
                              EX_Jal_wire_MEM,
                              EX_PC_4_wire_MEM,
                              EX_RegWrite_wire_MEM,
                              EX_MemtoReg_wire_MEM,
                              ReadDataOut_wire,
                              EX_ALUResult_wire_MEM,
                              EX_MUX_ForRTypeAndIType_wire_MEM
                              }),
```

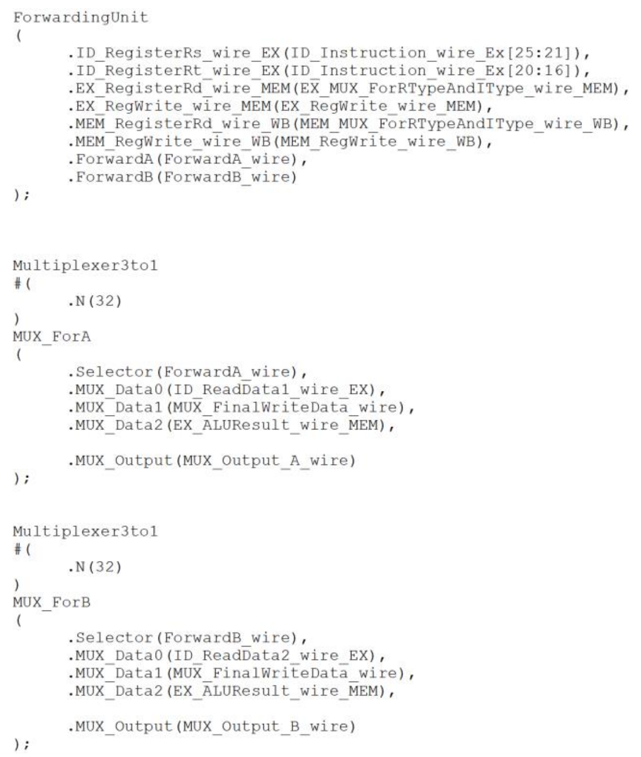
I also had to change all of the previous modules´ inputs and outputs in order to work with the new naming schema/system:

```
MUX_ForRTypeAndIType        //rt rd
(
      .Selector(ID_RegDst_wire_EX),
      .MUX_Data0(ID_Instruction_wire_Ex[20:16]),
      .MUX_Data1(ID_Instruction_wire_Ex[15:11]),
      .MUX_Output(MUX_WriteRegister_wire)

);



RegisterFile
Register_File
(
      .clk(clk),
      .reset(reset),
      .RegWrite(MEM_RegWrite_wire_WB),
      .WriteRegister(MEM_MUX_ForRTypeAndIType_wire_WB),
      .ReadRegister1(IF_Instruction_wire_ID[25:21]),    //rs
      .ReadRegister2(IF_Instruction_wire_ID[20:16]),    //rt
      .WriteData(MUX_FinalWriteData_wire),
      .ReadData1(ReadData1_wire),
      .ReadData2(ReadData2_wire)

);

SignExtend
SignExtendForConstants
(
      .DataInput(IF_Instruction_wire_ID[15:0]),
   .SignExtendOutput(InmmediateExtend_wire)
);



Multiplexer2to1
#(
      .N(32)
)
MUX_ForReadDataAndInmediate
(
      .Selector(ID_ALUSrc_wire_EX),
      .MUX_Data0(MUX_Output_B_wire),
      .MUX_Data1(ID_InmmediateExtend_wire_EX),
      .MUX_Output(ReadData2OrInmmediate_wire)

);

ALUControl
ArithmeticLogicUnitControl
(
      .ALUOp(ID_ALUOp_wire_EX),
      .ALUFunction(ID_InmmediateExtend_wire_EX[5:0]),
      .ALUOperation(ALUOperation_wire),
      .Jr(Jr_wire)
);



ALU
ArithmeticLogicUnit
(
      .ALUOperation(ALUOperation_wire),
      .A(MUX_Output_A_wire),
      .B(ReadData2OrInmmediate_wire),
```

For the forwarding unit, I created a new register and added the combinational algorithm proposed in the course´s presentation:



```
        input EX_RegWrite_wire_MEM,
        input MEM_RegWrite_wire_WB,
        input [N-1:0] EX_RegisterRd_wire_MEM,
        input [N-1:0] ID_RegisterRs_wire_EX,
        input [N-1:0] ID_RegisterRt_wire_EX,
        input [N-1:0] MEM_RegisterRd_wire_WB,

        output reg [1:0] ForwardA,
        output reg [1:0] ForwardB
    );

        always@(EX_RegWrite_wire_MEM,
                MEM_RegWrite_wire_WB,
                EX_RegisterRd_wire_MEM,
                ID_RegisterRs_wire_EX,
                ID_RegisterRt_wire_EX,
                MEM_RegisterRd_wire_WB)begin

            if(EX_RegWrite_wire_MEM &&
            EX_RegisterRd_wire_MEM != 0 &&
            EX_RegisterRd_wire_MEM == ID_RegisterRs_wire_EX)
                ForwardA = 2'b10;
            else
                if(MEM_RegWrite_wire_WB &&
                MEM_RegisterRd_wire_WB != 0 &&
                EX_RegisterRd_wire_MEM != ID_RegisterRs_wire_EX &&
                MEM_RegisterRd_wire_WB == ID_RegisterRs_wire_EX)
                    ForwardA = 2'b01;
                else
                    ForwardA = 2'b00;
            if(EX_RegWrite_wire_MEM &&
            EX_RegisterRd_wire_MEM != 0 &&
            EX_RegisterRd_wire_MEM == ID_RegisterRt_wire_EX)
                ForwardB = 2'b10;
            else
                if(MEM_RegWrite_wire_WB &&
                MEM_RegisterRd_wire_WB != 0 &&
                EX_RegisterRd_wire_MEM != ID_RegisterRt_wire_EX &&
                MEM_RegisterRd_wire_WB == ID_RegisterRt_wire_EX)
                    ForwardB = 2'b01;
                else
                    ForwardB = 2'b00;

        end

    endmodule
```



```
ForwardingUnit
(
      .ID_RegisterRs_wire_EX(ID_Instruction_wire_Ex[25:21]),
      .ID_RegisterRt_wire_EX(ID_Instruction_wire_Ex[20:16]),
      .EX_RegisterRd_wire_MEM(EX_MUX_ForRTypeAndIType_wire_MEM),
      .EX_RegWrite_wire_MEM(EX_RegWrite_wire_MEM),
      .MEM_RegisterRd_wire_WB(MEM_MUX_ForRTypeAndIType_wire_WB),
      .MEM_RegWrite_wire_WB(MEM_RegWrite_wire_WB),
      .ForwardA(ForwardA_wire),
      .ForwardB(ForwardB_wire)
);


Multiplexer3to1
#(
      .N(32)
)
MUX_ForA
(
      .Selector(ForwardA_wire),
      .MUX_Data0(ID_ReadData1_wire_EX),
      .MUX_Data1(MUX_FinalWriteData_wire),
      .MUX_Data2(EX_ALUResult_wire_MEM),

      .MUX_Output(MUX_Output_A_wire)
);


Multiplexer3to1
#(
      .N(32)
)
MUX_ForB
(
      .Selector(ForwardB_wire),
      .MUX_Data0(ID_ReadData2_wire_EX),
      .MUX_Data1(MUX_FinalWriteData_wire),
      .MUX_Data2(EX_ALUResult_wire_MEM),

      .MUX_Output(MUX_Output_B_wire)
);
```

For hazard detection, I created a new register type where the ins and outs would depend on the

```
module HazardUnit
#(
      parameter N=5
)
(
      input ID_MemRead_wire_EX,
      input [N-1:0] ID_RegisterRt_wire_EX,
      input [N-1:0] IF_RegisterRs_wire_ID,
      input [N-1:0] IF_RegisterRt_wire_ID,

      output reg Flush_wire,
      output reg IF_Enable_wire_ID,
      output reg Stall_wire
);

      always@(ID_MemRead_wire_EX,
                  ID_RegisterRt_wire_EX,
                  IF_RegisterRs_wire_ID,
                  IF_RegisterRt_wire_ID)
                  begin

                  if    (ID_MemRead_wire_EX &&
                        (ID_RegisterRt_wire_EX ==
IF_RegisterRs_wire_ID ||ID_RegisterRt_wire_EX ==
IF_RegisterRt_wire_ID))
                        begin
                              Flush_wire = 1;
                              IF_Enable_wire_ID = 0;
                              Stall_wire = 1;
                        end
                  else
                        begin
                              Flush_wire = 0;
                              IF_Enable_wire_ID = 1;
                              Stall_wire = 0;
                        end
                  end
endmodule
```

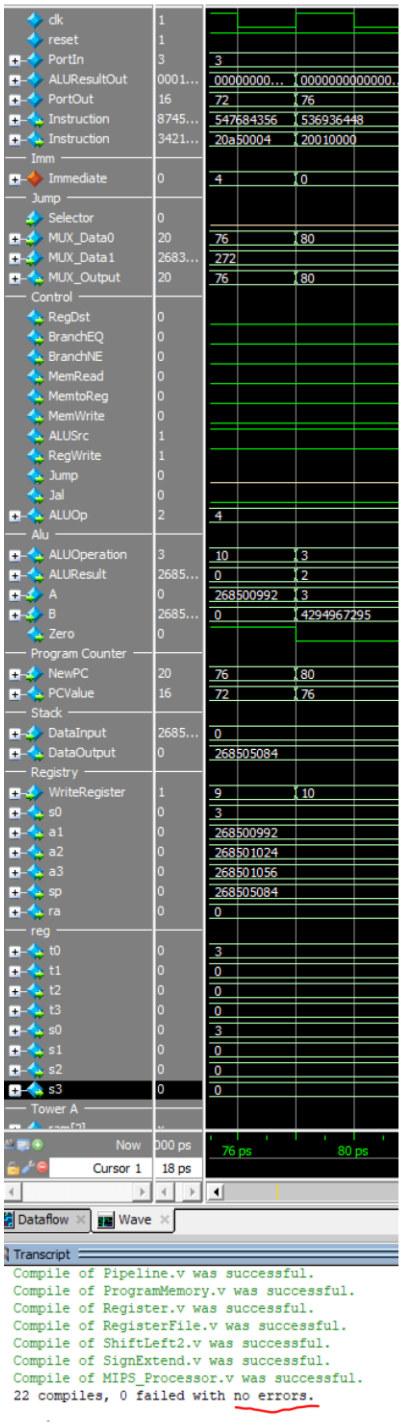
For both of the new HDU/FWU, I had to create a and update new mux types that would either reset a signal after ID or to just pass 0 as a predefined parameter (flush or stall the stage) according to the instruction scenario – for the branch prediction status, I would always assume the branch would not be taken so that a flush mux would then pass a predefined 0 signal to the IF/ID out pipe in order to “cancel” the rest of the conflicting/not needed instruction:

```
module Multiplexer3to1
#
(
	parameter N=32
)
(
	input [1:0] Selector,
	input [N-1:0] MUX_Data0,
	input [N-1:0] MUX_Data1,
	input [N-1:0] MUX_Data2,
	output reg [N-1:0] MUX_Output
);

	always@(Selector,MUX_Data0,MUX_Data1,MUX_Data2)
	begin
		case (Selector)
			2'b00: MUX_Output <= MUX_Data0;
			2'b01: MUX_Output <= MUX_Data1;
			2'b10: MUX_Output <= MUX_Data2;
			default: MUX_Output <= 'b0;
		endcase
	end

endmodule
```

```
Hazard
(

      .ID_MemRead_wire_EX(ID_MemRead_wire_EX),
      .ID_RegisterRt_wire_EX(ID_Instruction_wire_Ex[20:16]),
      .IF_RegisterRs_wire_ID(IF_Instruction_wire_ID[25:21]),
      .IF_RegisterRt_wire_ID(IF_Instruction_wire_ID[20:16]),
      .Flush_wire(Flush_wire),
      .IF_Enable_wire_ID(IF_Enable_wire_ID),
      .Stall_wire(Stall_wire)
);


Multiplexer2to1
#(
      .N(32)
)
MUX_Stall
(
      .Selector(Stall_wire),
      .MUX_Data0(MUX_FinalPC_wire),
      .MUX_Data1(PC_wire),
      .MUX_Output(MUX_PC_Stall_wire)
);


Multiplexer2to1
MUX_IF_Flush_ID
(
      .Selector(PCtoBranch_wire | Jump_wire | Jr_wire),
      .MUX_Data0(Instruction_wire),
      .MUX_Data1(0),

      .MUX_Output(Flush_Instruction_wire)
);


Multiplexer2to1
#(
      .N(12)
)
MUX_ID_Flush_EX
(
      .Selector(Flush_wire | PCtoBranch_wire | EX_Jump_wire_MEM |
EX_Jr_wire_MEM),
      .MUX_Data0({
                              Jal_wire,
                              Jump_wire, //1
                              BranchNE_wire,//1
                              BranchEQ_wire,//1
                              RegDst_wire,//1
                              ALUOp_wire,//3
                              ALUSrc_wire,//1
                              RegWrite_wire,//1
                              MemWrite_wire,//1
                              MemRead_wire,          //1
                              MemToReg_wire          //1
      }),
      .MUX_Data1(0),
      .MUX_Output({
                              Flush_Jal_wire,
                              Flush_Jump_wire, //added
                              Flush_BranchNE_wire,
                              Flush_BranchEQ_wire,
                              Flush_RegDst_wire,

                              Flush_ALUOp_wire,

                              Flush_ALUSrc_wire,

                              Flush_RegWrite_wire,

                              Flush_MemWrite_wire,

                              Flush_MemRead_wire,

                              Flush_MemToReg_wire
      })
```

Test and debugging:

For simulation on ModelSim, I first created a wave viewer/map where I would be adding new registers ins and outs from time to time according to my needs for debugging each stage:

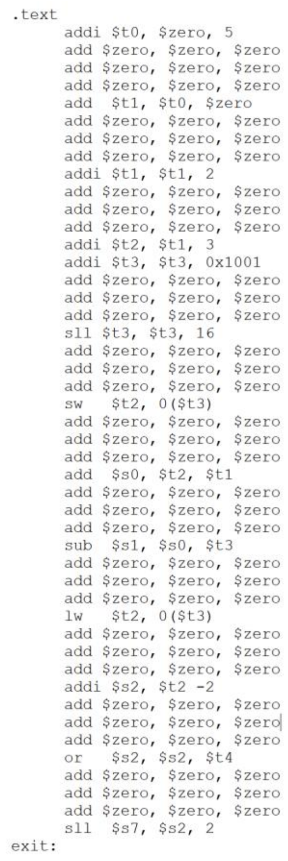


```
Compile of Pipeline.v was successful.
Compile of ProgramMemory.v was successful.
Compile of Register.v was successful.
Compile of RegisterFile.v was successful.
Compile of ShiftLeft2.v was successful.
Compile of SignExtend.v was successful.
Compile of MIPS_Processor.v was successful.
22 compiles, 0 failed with no errors.
```

Instruction tests:

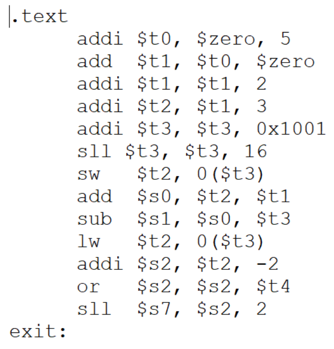
I created a set of programs that would test each of the initially implemented instruction at its most basic operation level. All of them are included in the archive.

pipeTest (test with nop´s stalling)



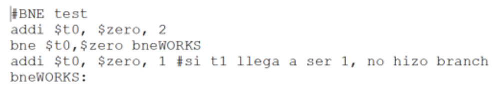
```
.text
      addi $t0, $zero, 5
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      add  $t1, $t0, $zero
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      addi $t1, $t1, 2
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      addi $t2, $t1, 3
      addi $t3, $t3, 0x1001
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      sll $t3, $t3, 16
      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   $t2, 0($t3)
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      addi $s2, $t2 -2
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      or   $s2, $s2, $t4
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      add $zero, $zero, $zero
      sll  $s7, $s2, 2
exit:
```

pipefwTest (test no nop)



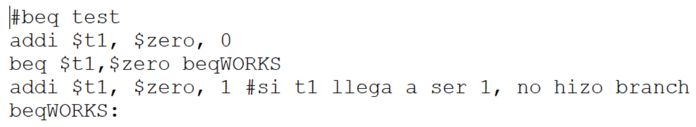
```
.text
      addi $t0, $zero, 5
      add  $t1, $t0, $zero
      addi $t1, $t1, 2
      addi $t2, $t1, 3
      addi $t3, $t3, 0x1001
      sll $t3, $t3, 16
      sw   $t2, 0($t3)
      add  $s0, $t2, $t1
      sub  $s1, $s0, $t3
      lw   $t2, 0($t3)
      addi $s2, $t2, -2
      or   $s2, $s2, $t4
      sll  $s7, $s2, 2
exit:
```

Bne (test flush after ID on conflicting instruction)



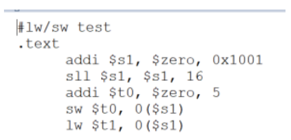
```
#BNE test
addi $t0, $zero, 2
bne $t0,$zero bneWORKS
addi $t0, $zero, 1 #si t1 llega a ser 1, no hizo branch
bneWORKS:
```

Beq (test flush)



```
#beq test
addi $t1, $zero, 0
beq $t1,$zero beqWORKS
addi $t1, $zero, 1 #si t1 llega a ser 1, no hizo branch
beqWORKS:
```

lwswTest

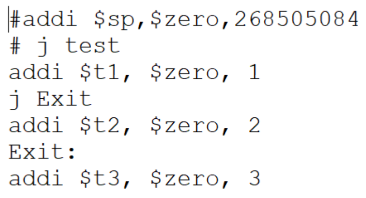


```
#lw/sw test
.text
      addi $s1, $zero, 0x1001
      sll $s1, $s1, 16
      addi $t0, $zero, 5
      sw $t0, 0($s1)
      lw $t1, 0($s1)
```

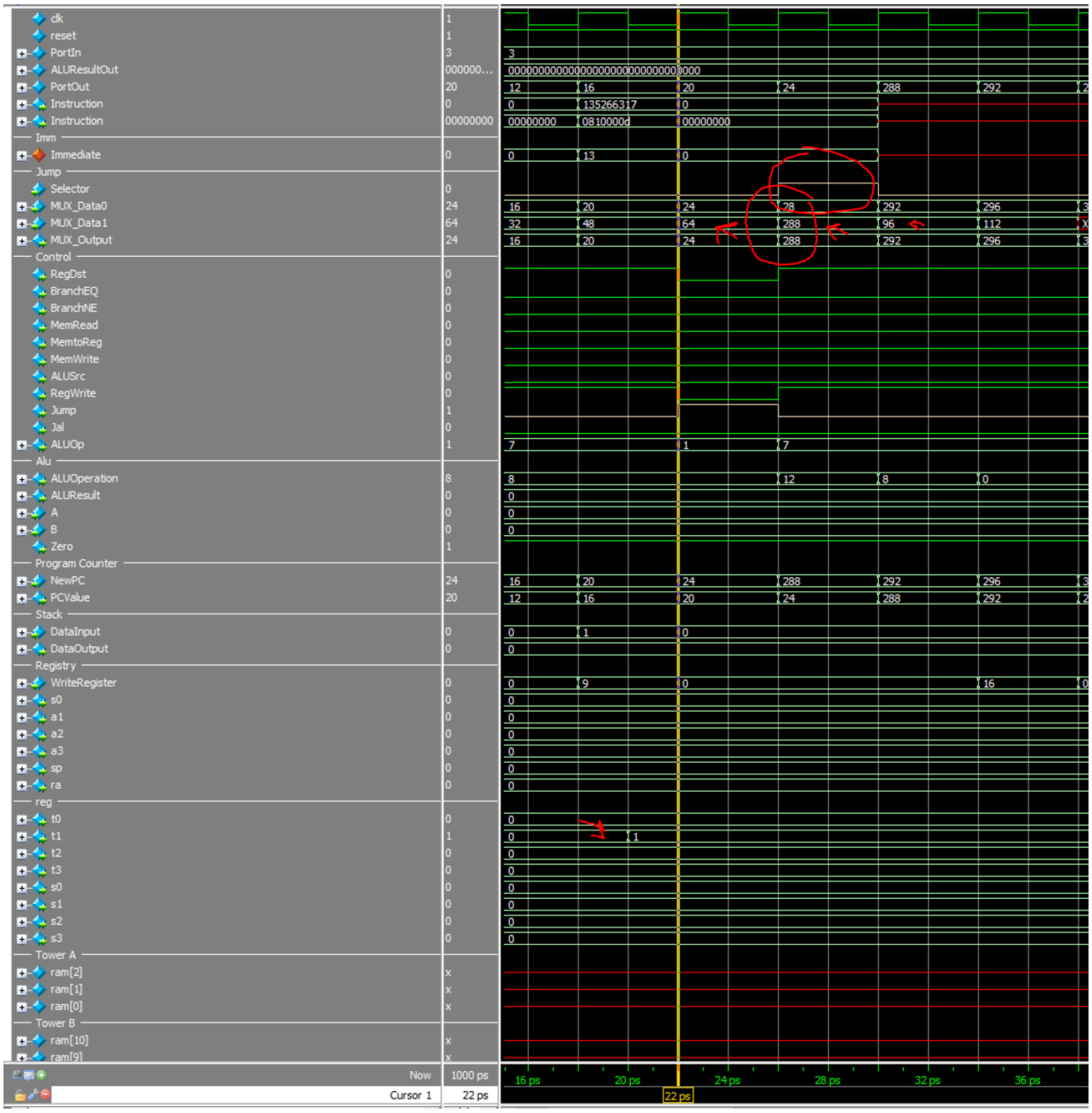
J types and Hanoi:

Even though the above test did work/pass, unfortunately, and here´s where I failed, I never seemed to either fix or even understand the J type instructions through a pipeline approach; I couldn’t seem to figure the Datapath that a jump address would have to go through/by in order to go to the stage it is needed at the precise cycle it is needed. And yes, I tried using nop´s…

jTest



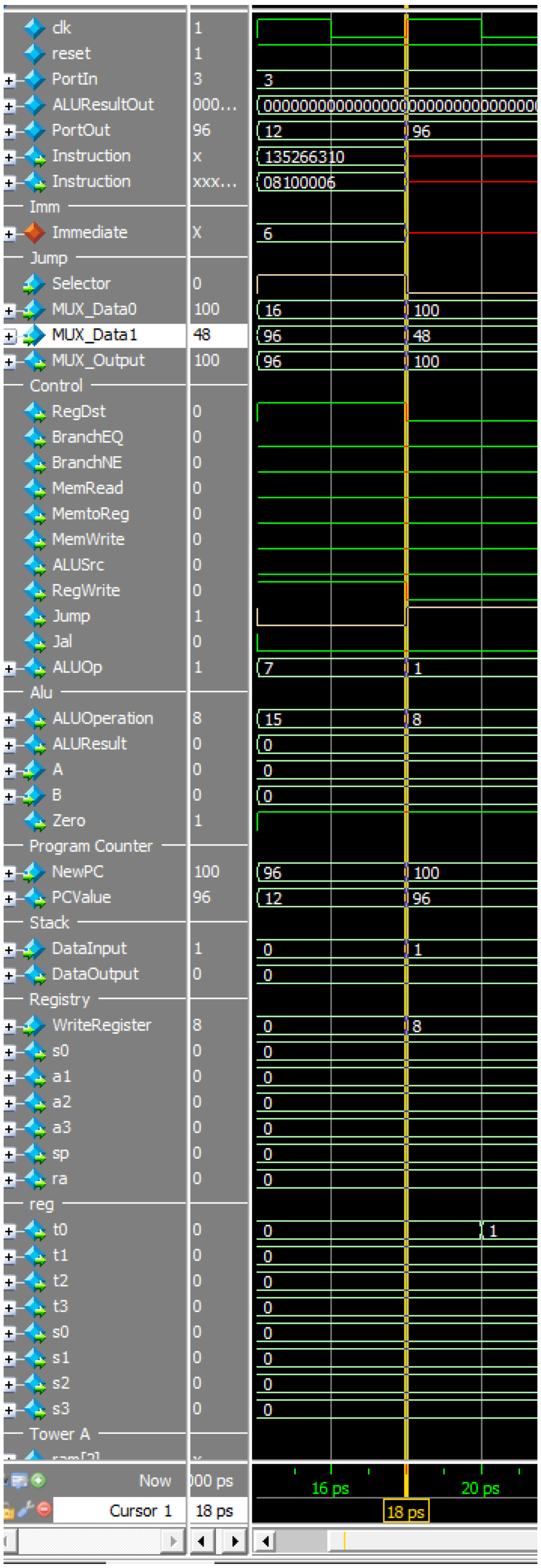
```
#addi $sp,$zero,268505084
# j test
addi $t1, $zero, 1
j Exit
addi $t2, $zero, 2
Exit:
addi $t3, $zero, 3
```



```
clk            1
reset          1
PortIn         3
ALUResultOut   000000...
PortOut        20
Instruction    0
Instruction    00000000
Imm
Immediate      0
Jump
Selector       0
MUX_Data0      24
MUX_Data1      64
MUX_Output     24
Control
RegDst         0
BranchEQ       0
BranchNE       0
MemRead        0
MemtoReg       0
MemWrite       0
ALUSrc         0
RegWrite       0
Jump           1
Jal            0
ALUOp          1
Alu
ALUOperation   8
ALUResult      0
A              0
B              0
Zero           1
Program Counter
NewPC          24
PCValue        20
Stack
DataInput      0
DataOutput     0
Registry
WriteRegister  0
s0             0
a1             0
a2             0
a3             0
sp             0
ra             0
reg
t0             0
t1             1
t2             0
t3             0
s0             0
s1             0
s2             0
s3             0
Tower A
ram[2]         x
ram[1]         x
ram[0]         x
Tower B
ram[10]        x
ram[9]         x
Now            1000 ps
Cursor 1       22 ps
```

jrTest

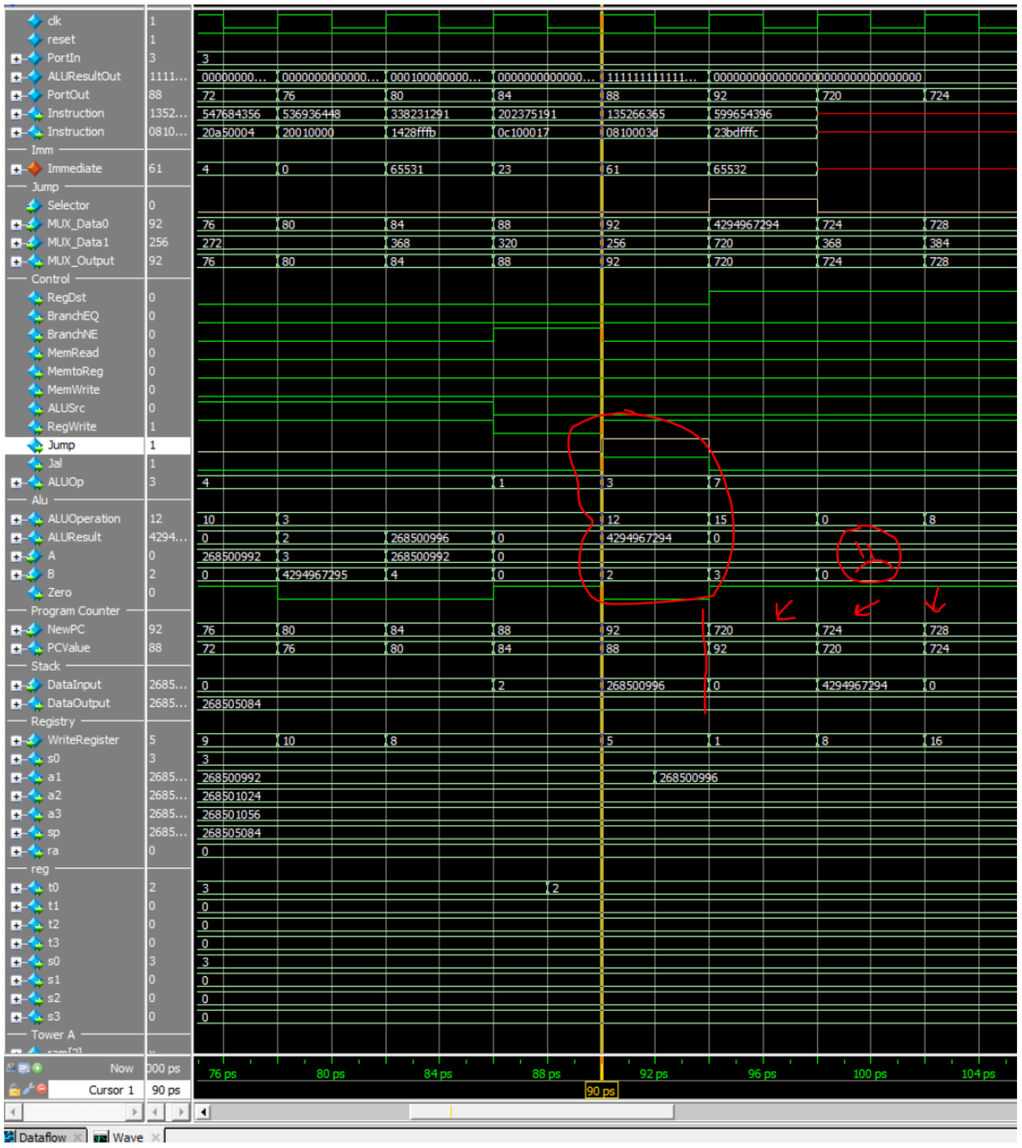
```
#JR test
addi $t0, $zero, 1
jal TEST
addi $t0, $zero, 3
j Exit
TEST:
addi $t0, $zero, 2
jr $ra
Exit:
```



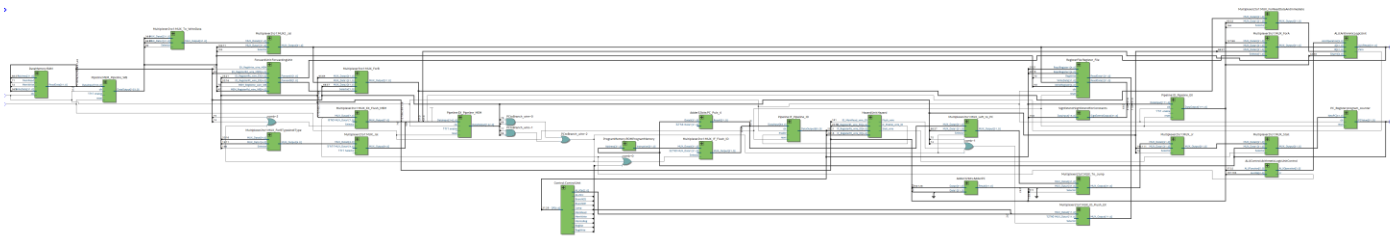
```
clk               1
reset             1
PortIn            3        3
ALUResultOut      000...   0000000000000000000000000000000
PortOut           96       12                 96
Instruction       x        135266310
Instruction       xxx...   08100006
Imm
Immediate         X        6
Jump
Selector          0
MUX_Data0         100      16                 100
MUX_Data1         48       96                 48
MUX_Output        100      96                 100
Control
RegDst            0
BranchEQ          0
BranchNE          0
MemRead           0
MemtoReg          0
MemWrite          0
ALUSrc            0
RegWrite          0
Jump              1
Jal               0
ALUOp             1        7                  1
Alu
ALUOperation      8        15                 8
ALUResult         0        0
A                 0        0
B                 0        0
Zero              1
Program Counter
NewPC             100      96                 100
PCValue           96       12                 96
Stack
DataInput         1        0                  1
DataOutput        0        0
Registry
WriteRegister     8        0                  8
s0                0        0
a1                0        0
a2                0        0
a3                0        0
sp                0        0
ra                0        0
reg
t0                0        0                            1
t1                0        0
t2                0        0
t3                0        0
s0                0        0
s1                0        0
s2                0        0
s3                0        0
Tower A
ram[?]            x

Now        000 ps          16 ps              20 ps
Cursor 1   18 ps                 18 ps
```

Hanoi (“hanoiTest”)

For Hanoi, as the j type instructions were broken the moment I tried to pipeline my original approach, the program did not work right at the moment where the first Jal is used: it calls the Hanoi Tower recursively after linking its current address, but there were lots of different problems – sometimes it even wrote the $ra data on a random register like $so. In this case, I also tried to nop the hell out of the original program and no, neither did work:

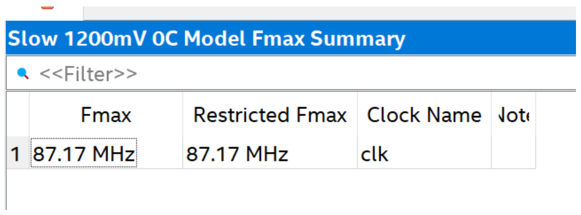


3. RTL DIagram



4. Clock rate and CPU comparison:

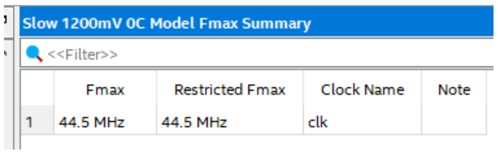
Single Cycled and Pipelined



Slow 1200mV 0C Model Fmax Summary

<<Filter>>

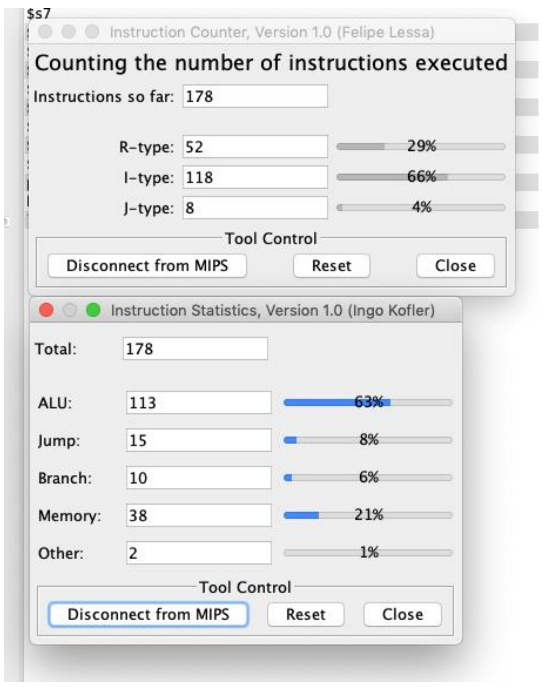
	Fmax	Restricted Fmax	Clock Name	Note
1	87.17 MHz	87.17 MHz	clk	



Slow 1200mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	44.5 MHz	44.5 MHz	clk	



$s7

Instruction Counter, Version 1.0 (Felipe Lessa)

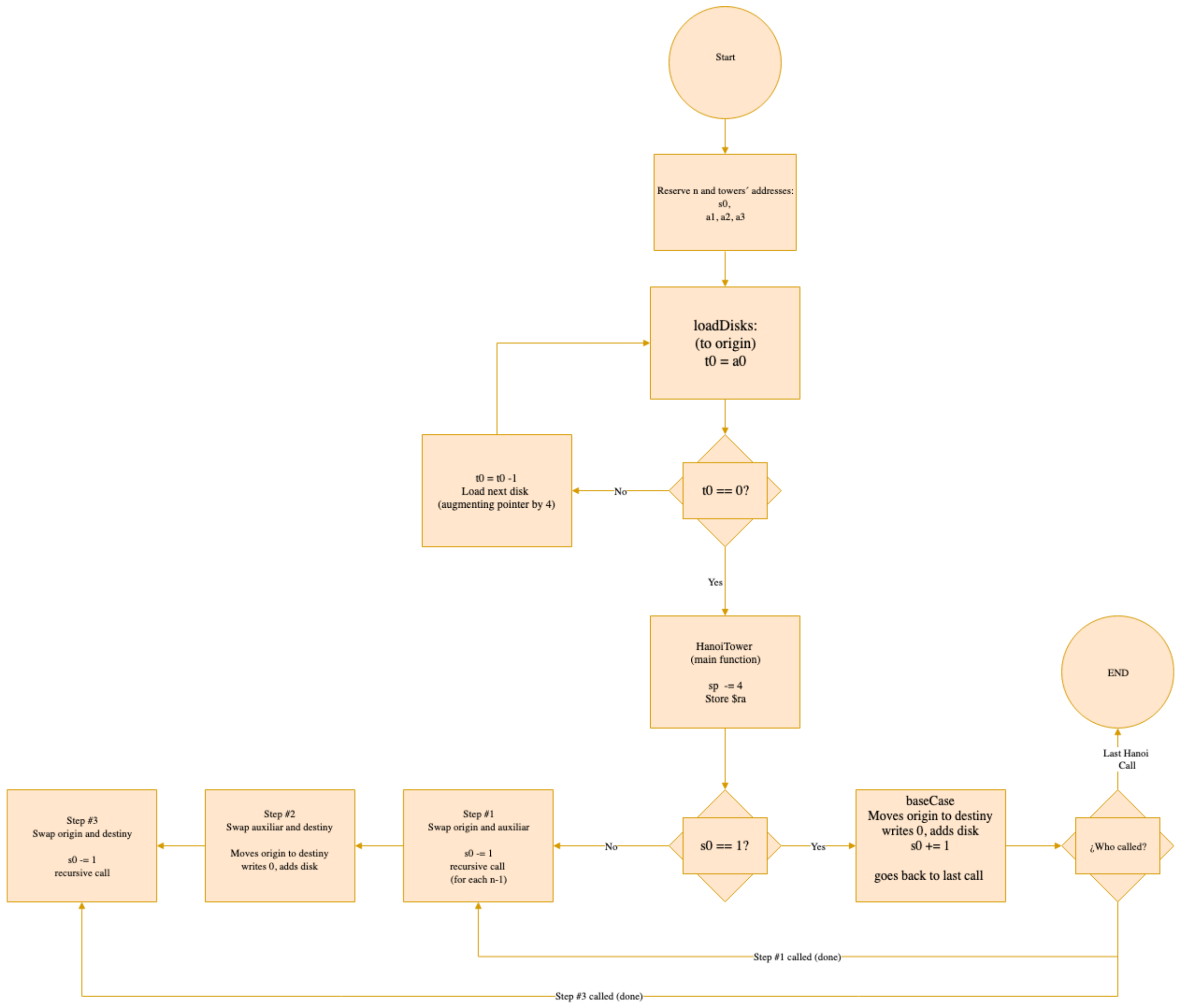
Counting the number of instructions executed

Instructions so far:	178	
R-type:	52	29%
I-type:	118	66%
J-type:	8	4%

Instruction Statistics, Version 1.0 (Ingo Kofler)

Total:	178	
ALU:	113	63%
Jump:	15	8%
Branch:	10	6%
Memory:	38	21%
Other:	2	1%

5. Flow diagram and description:



Flow description:

My algorithm would begin by associating each of the addresses to each of its respective tower as well as register. We´d also always have n stored in s0, even though such variable is inputed by the user beforehand.

The loadDisks function or subroutine would then begin to load each and every disk onto its respective space or place on tower A/origin tower until the amount of disks are complete. And then it would proceed to call the HanoiTower function for the first time, which would initially allocate the memory necessary to store the ra registry onto the top stack position.

The function would first try to validate whether the amount of disks is or isn´t 1 in order to jump to the base case (which would move the respective tower to its according position) or to the internal function steps (which would swap the pointers or addresses on each of the towers accordingly in order to follow the algorithm described on the assignment).

The process would loop in and out of the steps, unwinding and then rewinding when its ready to swap the disks to their respective next tower, step by step, all by switching the auxiliary tower accordingly. Once the algorithms takes us to the last step or to the top of the rewinding process, it would put the last disk onto its final place and finally it would restore the size of the sp as well as s0 to finally be done with the process.

9 – Files

Check the included archive, where the top and rest of the v files, pdf architecture and flow diagrams are also included.