CSEN 601 Computer Architecture

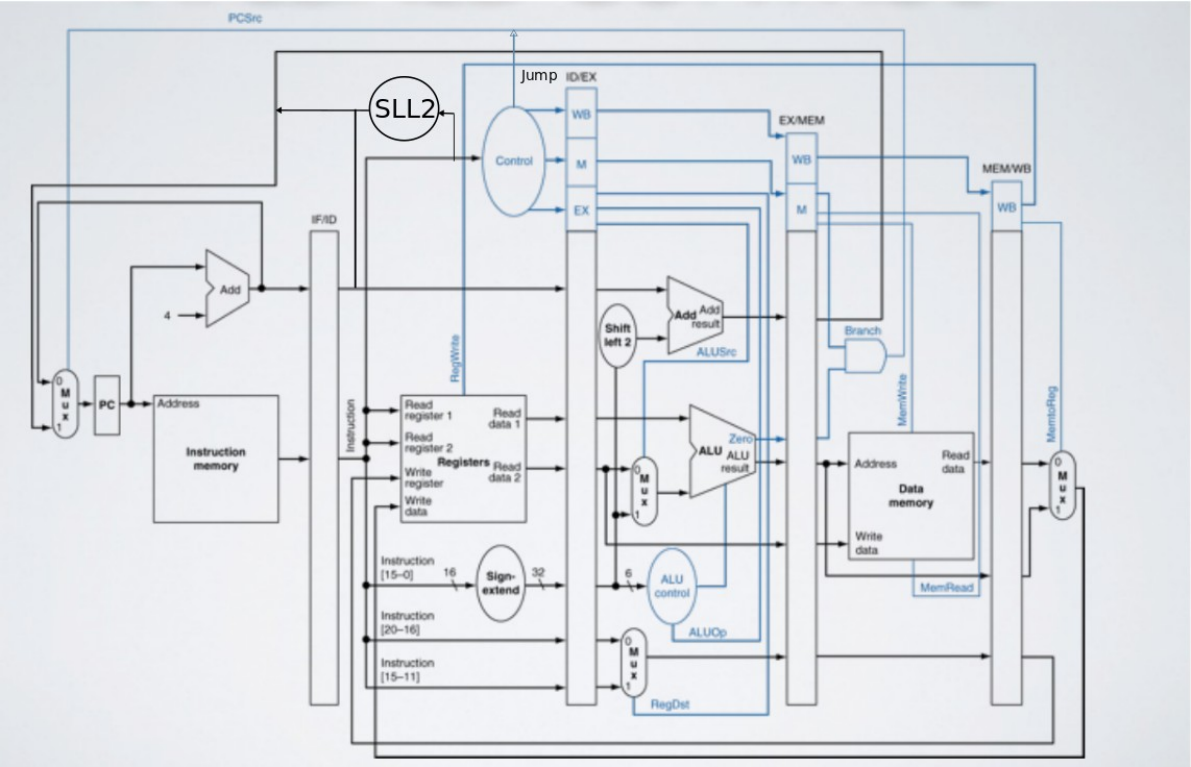
Pipelined MIPS compiler and simulator using JRuby

Name	ID
Ahmed Abd El Badie	28-0605
Ahmed Hesham	28-0708
Ahmed Raafat	28-7826
Mo Agamia	28-9537

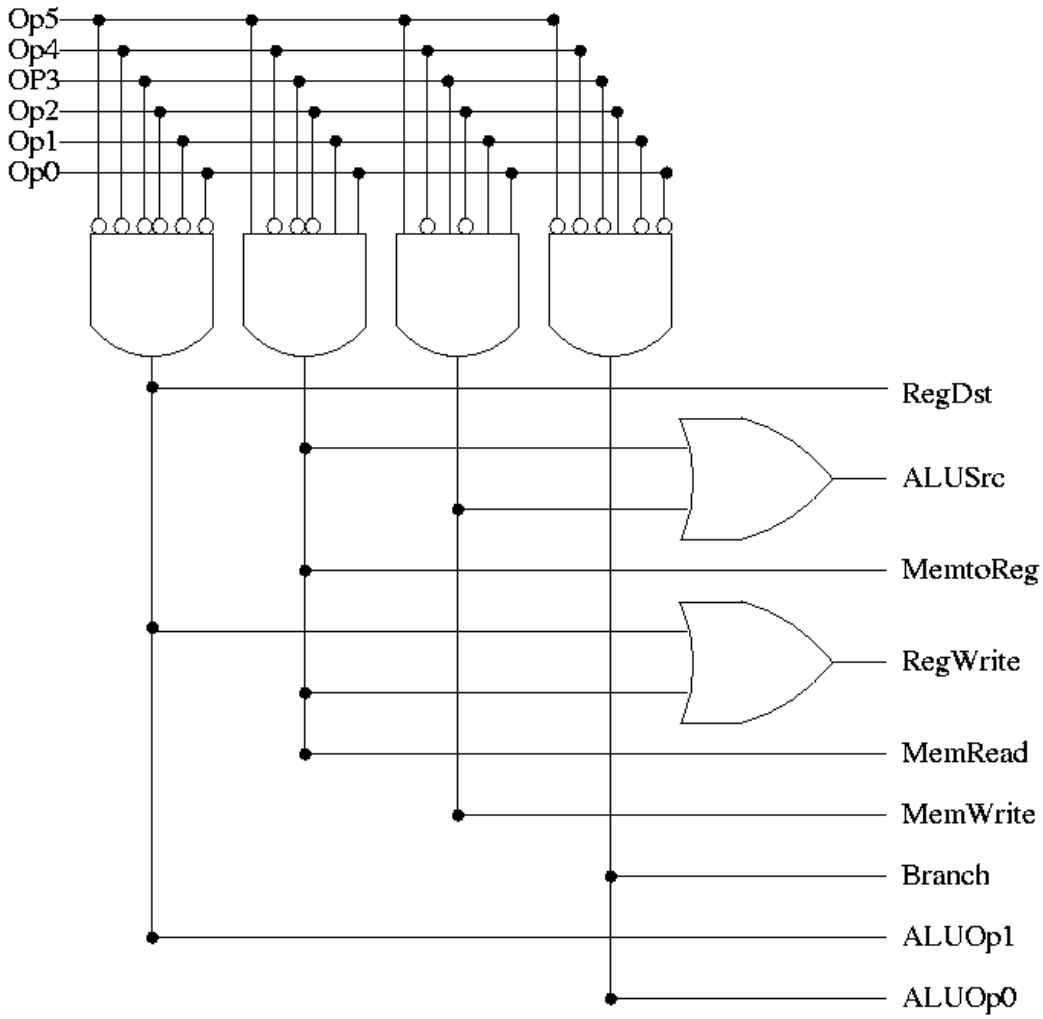
Description:

We implemented the pipelined simulator using JRuby. The user can enter a full MIPS code and even place comments (as hash tags) and the program will compile normally. We implemented the bonus feature of handling data hazards in branches and jumps by placing NOP operations where possible. The entire project was implemented in a modular manner to help ease the implementation and error and detection. Another bonus feature we implemented was the GUI, which is where we used the Java side of JRuby.

Datapath:



Control Unit:



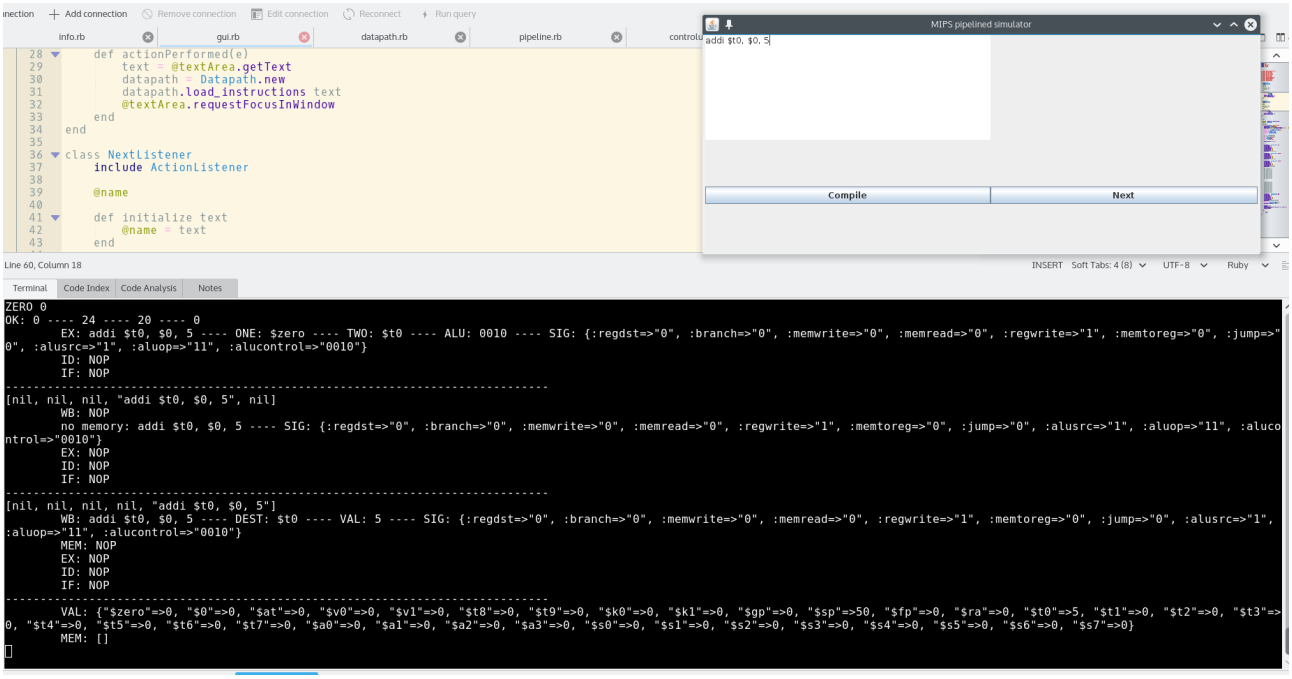
Assumptions:

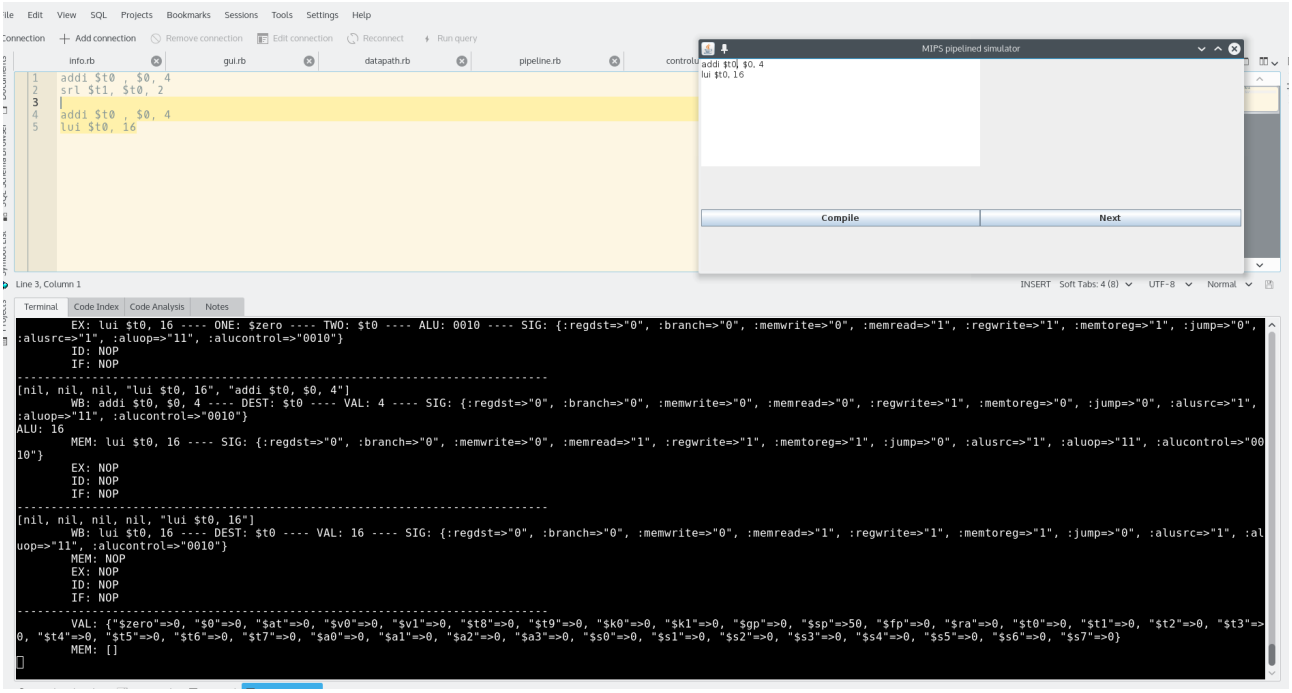
We assumed values for the AluControl signals as many were missing.

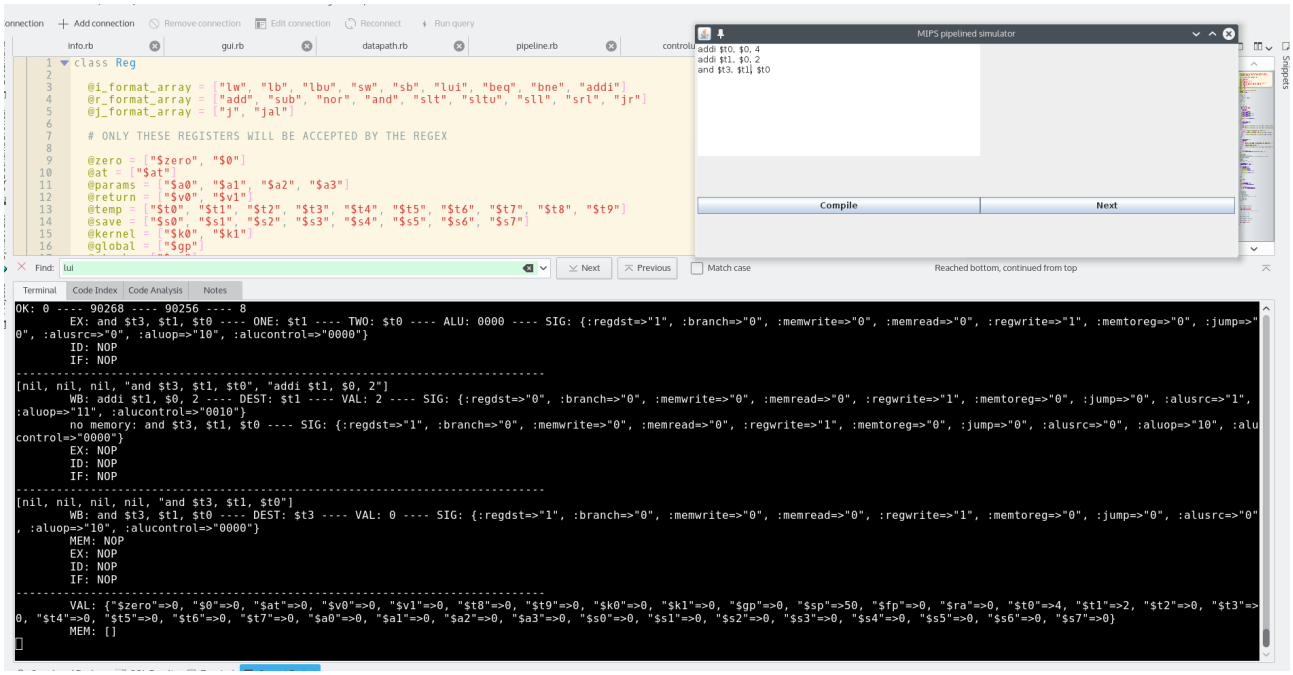
Sll	0011
Srl	0100
Nor	0101

Furthermore we assumed that when branching/jumping to labels in MIPS code, that we have to

Examples:


MIPS pipelined simulator
addi $t0, $0, 5
Compile
Next
ZERO 0
OK: 0 ---- 24 ---- 20 ---- 0
EX: addi $t0, $0, 5 ---- ONE: $zero ---- TWO: $t0 ---- ALU: 0010 ---- SIG: {:regdst=>"0", :branch=>"0", :memwrite=>"0", :memread=>"0", :regwrite=>"1", :memtoreg=>"0", :jump=>"0", :alusrc=>"1", :aluop=>"11", :alucontrol=>"0010"}
ID: NOP
IF: NOP
[nil, nil, nil, "addi $t0, $0, 5", nil]
WB: NOP
no memory: addi $t0, $0, 5 ---- SIG: {:regdst=>"0", :branch=>"0", :memwrite=>"0", :memread=>"0", :regwrite=>"1", :memtoreg=>"0", :jump=>"0", :alusrc=>"1", :aluop=>"11", :alucontrol=>"0010"}
EX: NOP
ID: NOP
IF: NOP
[nil, nil, nil, nil, "addi $t0, $0, 5"]
WB: addi $t0, $0, 5 ---- DEST: $t0 ---- VAL: 5 ---- SIG: {:regdst=>"0", :branch=>"0", :memwrite=>"0", :memread=>"0", :regwrite=>"1", :memtoreg=>"0", :jump=>"0", :alusrc=>"1", :aluop=>"11", :alucontrol=>"0010"}
MEM: NOP
EX: NOP
ID: NOP
IF: NOP
VAL: {"$zero"=>0, "$0"=>0, "$at"=>0, "$v0"=>0, "$v1"=>0, "$t8"=>0, "$t9"=>0, "$k0"=>0, "$k1"=>0, "$gp"=>0, "$sp"=>50, "$fp"=>0, "$ra"=>0, "$t0"=>5, "$t1"=>0, "$t2"=>0, "$t3"=>0, "$t4"=>0, "$t5"=>0, "$t6"=>0, "$t7"=>0, "$a0"=>0, "$a1"=>0, "$a2"=>0, "$a3"=>0, "$s0"=>0, "$s1"=>0, "$s2"=>0, "$s3"=>0, "$s4"=>0, "$s5"=>0, "$s6"=>0, "$s7"=>0}
MEM: []


MIPS pipelined simulator
addi $t0, $0, 4
lui $t0, 16
Compile
Next
EX: lui $t0, 16 ---- ONE: $zero ---- TWO: $t0 ---- ALU: 0010 ---- SIG: {:regdst=>"0", :branch=>"0", :memwrite=>"0", :memread=>"1", :regwrite=>"1", :memtoreg=>"1", :jump=>"0", :alusrc=>"1", :aluop=>"11", :alucontrol=>"0010"}
ID: NOP
IF: NOP
[nil, nil, nil, "lui $t0, 16", "addi $t0, $0, 4"]
WB: addi $t0, $0, 4 ---- DEST: $t0 ---- VAL: 4 ---- SIG: {:regdst=>"0", :branch=>"0", :memwrite=>"0", :memread=>"0", :regwrite=>"1", :memtoreg=>"0", :jump=>"0", :alusrc=>"1", :aluop=>"11", :alucontrol=>"0010"}
ALU: 16
MEM: lui $t0, 16 ---- SIG: {:regdst=>"0", :branch=>"0", :memwrite=>"0", :memread=>"1", :regwrite=>"1", :memtoreg=>"1", :jump=>"0", :alusrc=>"1", :aluop=>"11", :alucontrol=>"0010"}
EX: NOP
ID: NOP
IF: NOP
[nil, nil, nil, nil, "lui $t0, 16"]
WB: lui $t0, 16 ---- DEST: $t0 ---- VAL: 16 ---- SIG: {:regdst=>"0", :branch=>"0", :memwrite=>"0", :memread=>"1", :regwrite=>"1", :memtoreg=>"1", :jump=>"0", :alusrc=>"1", :aluop=>"11", :alucontrol=>"0010"}
MEM: NOP
EX: NOP
ID: NOP
IF: NOP
VAL: {"$zero"=>0, "$0"=>0, "$at"=>0, "$v0"=>0, "$v1"=>0, "$t8"=>0, "$t9"=>0, "$k0"=>0, "$k1"=>0, "$gp"=>0, "$sp"=>50, "$fp"=>0, "$ra"=>0, "$t0"=>0, "$t1"=>0, "$t2"=>0, "$t3"=>0, "$t4"=>0, "$t5"=>0, "$t6"=>0, "$t7"=>0, "$a0"=>0, "$a1"=>0, "$a2"=>0, "$a3"=>0, "$s0"=>0, "$s1"=>0, "$s2"=>0, "$s3"=>0, "$s4"=>0, "$s5"=>0, "$s6"=>0, "$s7"=>0}
MEM: []



Work Distribution:

The work was distributed evenly amongst the team members. Ahmed Abd El Badie created the pipelined architecture base and the output. Ahmed Hesham generated the datapath elements and the GUI. Ahmed Raafat handled the generation of binary strings. Mohamed Agamia generated the regular expressions and parser for the assembly code.