**Project** **2**

*By*

***Mohamed waleed elshemy*** *&* ***Noureldeen Amr Allama***

**Project Description**

In this project, our purpose was to design, build, and implement an architectural simulator with the capacity of performance assessment for a simplified (out of order) 16-Bit RISC processor that is using the dynamic (out of order) (single-issued) Tomasulo Algorithm without speculation (No Committing). Our programming language of choice was C++. As demanded by the ISA of the given project document, we will assume a simplified computer (RISC-16) with 16-Bit word size and 8 Regs. The memory is also word addressable. The Instructions implemented are: Load/Store, BNE, Call/Return, and some Arithmetic and Logic operations. The numbers of both, the reservation stations and needed cycles are also given in a table in the project doc. No bonus feature was included. We also took some simplifying assumptions from the doc.

Our work to implement this simulator mainly depended on:

* Designing a structure for the Reservation Stations
* Designing a structure for the Functional Units status
* Designing a structure for ROB and a class for the Tomasulo Processor
* Building multiple constructors to initialize each of the aforementioned
* Implementing multiple functions for each function required in Tomasulo and the required simulation outputs (Metrics, Clock Cycles, …)
* Supporting assessments for multiple Instructions (Load/Store, BNE, ADDI, …)
* Multiple test programs for the simulator

**Reservation Stations**

We designed the Reservation Station Entry as a structure that takes the different information on the instruction’s execution. (operation type(op), operands(Vj, Vk), dependencies( Qj, Qk), Destination Reg (Dest), Branch Address(Addrs), availability flag (busy), Clock Cycles (cycles))

struct Instruction {

OpType op;

int rs,rt,rd,imm;

Instruction(OpType \_op,int \_rs,int \_rt,int \_rd,int \_imm): op(\_op), rs(\_rs), rt(\_rt), rd(\_rd), imm(\_imm) {} //Change values/start with values of your choice for each

};

//Reservation Station Entry Structure (Instrucion's Execution Status)

struct RS{

OpType op;

int Vj;

int Vk;

int Qj,

int Qk;

int Dest;

int Addrs;

bool busy;

int cycles;

RS() : busy(false), cycles(0) {} //Constructor Initialization

};

**Functional Units and ROB**

A structure for Functional Units’ status and its constructor, and a structure for the ROB entry and its readiness flag.

//Functional Unit Status Structure

struct FU\_Stat {

int cycles\_to\_go;

bool busy;

FU\_Stat() : cycles\_to\_go(0), busy(false) {}

};

//Reorder Buffer Entry Structure

struct ROB {

bool valid;

int Dest;

};

**Tomasulo Processor**

A class defining the Tomasulo algorithm with multiple public and private members and multiple stage functions.

//Tomasulo Algorithm Processor Class

class Tomasulo {

public:

Tomasulo(const vector<Instruction> & program, const vector<pair<int,int>> & data);

void simulate();

private:

vector<RS> reservation\_stations;

vector<ROB> reorder\_buffer;

vector<FU\_Stat> functional\_units;

unordered\_map<int,int> mem;

vector<int> regs;

vector<int> tags;

int pc;

int clock\_cycles;

int miss\_branch;

void issue(const Instruction & instruction);

void exe();

void exe\_instruction(RS & rs);

void write();

void metrics();

void print\_rs\_stat();

};

**Instruction Cases**

The Instruction Cases for the required simulation assessment.

void Tomasulo::issue(const Instruction& instruction) {

RS new\_rs;

new\_rs.op = instruction.op;

new\_rs.Dest = instruction.rd;

new\_rs.busy = true;

switch (instruction.op) {

case OpType::load:

case OpType::store:

new\_rs.Vj = regs[instruction.rs];

new\_rs.Qj = tags[instruction.rs];

new\_rs.Vk = instruction.imm;

new\_rs.Qk = -1;

new\_rs.cycles = 3;

break;

case OpType::BNE:

new\_rs.Vj = regs[instruction.rs];

new\_rs.Qj = tags[instruction.rs];

new\_rs.Vk = regs[instruction.rt];

new\_rs.Qk = tags[instruction.rt];

new\_rs.Addrs = instruction.imm;

new\_rs.cycles = 1;

break;

case OpType::ADD:

case OpType::ADDI:

case OpType::NAND:

case OpType::DIV:

new\_rs.Vj = regs[instruction.rs];

new\_rs.Qj = tags[instruction.rs];

new\_rs.Vk = regs[instruction.rt];

new\_rs.Qk = tags[instruction.rt];

new\_rs.cycles = 2;

break;

case OpType::CALL:

new\_rs.Vj = pc + 1; // PC+1

new\_rs.Qj = -1;

new\_rs.cycles = 1;

break;

case OpType::RET:

new\_rs.Vj = regs[1];

new\_rs.Qj = tags[1];

new\_rs.cycles = 1;

break;

**Tests**



