University of Pittsburgh

School of Computing and Information

CS2410

Computer Architecture

Out-Of-Order Execution CPU Simulator Using Tomasulo Algorithm and Speculation

Date: April 25, 2023

Jacob Hoffman

Advisors:

Xulong Tang PhD

1. Introduction

1.1. Purpose

Using out-of-order execution allows a processor to execute on a set of instructions based on the availability of data or operands, which dramatically increases performance over in-order execution. This is due to the avoidance of stalls/idling in the processor pipeline that arise due to the handling of true data dependencies, and out-of-order execution may execute the instructions in a different order to avoid these stalls. Computer architecture simulators are able to lower costs for research and development of new hardware designs by allowing the architect to evaluate their proposed architecture without building the physical hardware system.

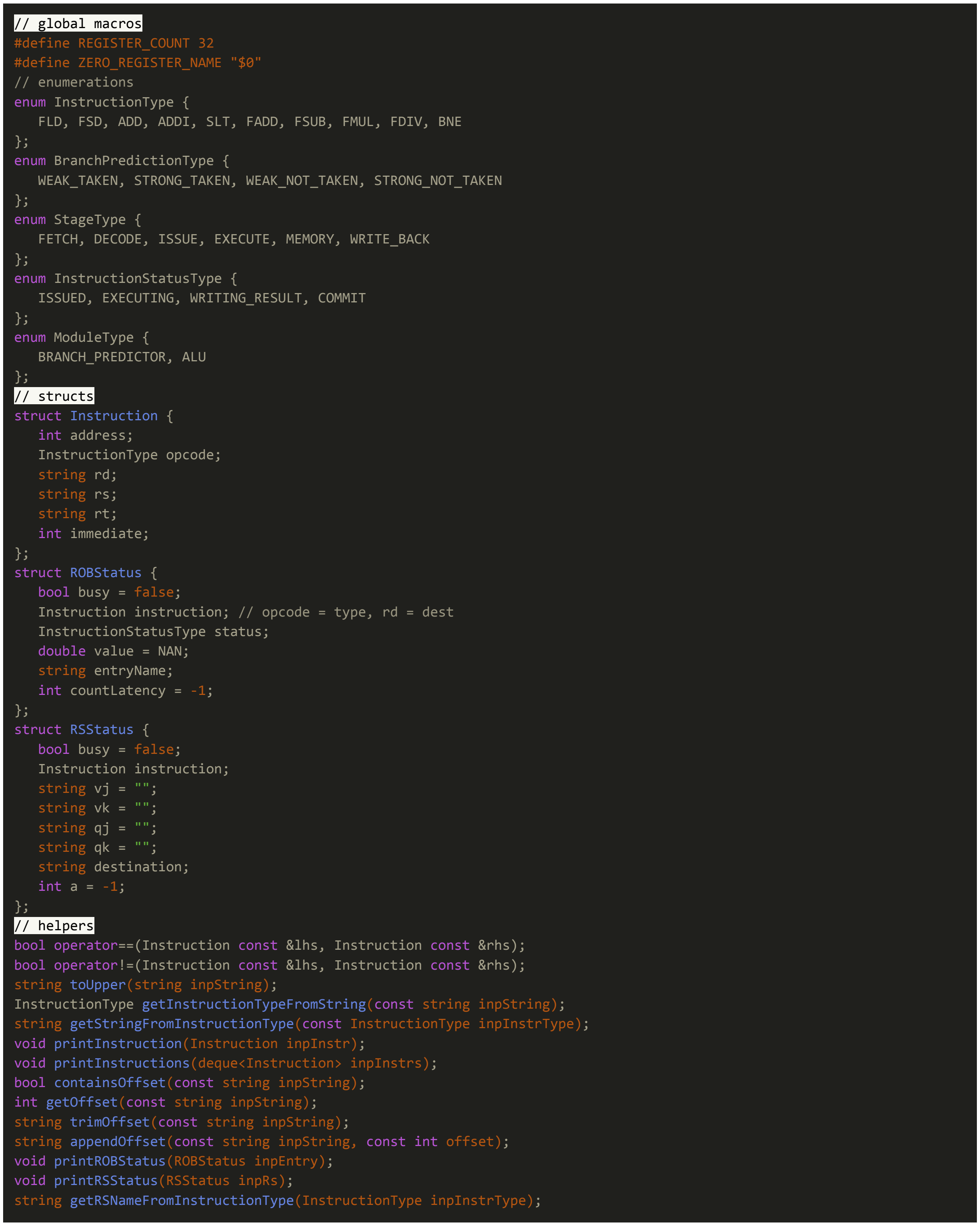
1.2. Scope of Project

The project required the implementation of a dynamically scheduled processor simulator based on a simplified version of the PowerPC 604 RISC Microprocessor [1]. Only a specified subset of the RISC V instruction set was required (fld, fsd, add, addi, slt, fadd, fsub, fmul, fdiv, bne), and certain parameters for controlling system latency/bandwidth had to be implemented with input parameterization capabilities for the user (NF, NI, NW, NB, NR). Then, certain changes in latency and issue capabilities on the simulated system were evaluated by adjusting these parameters.

2. Project Description

2.1. Structs, Enums, and Classes

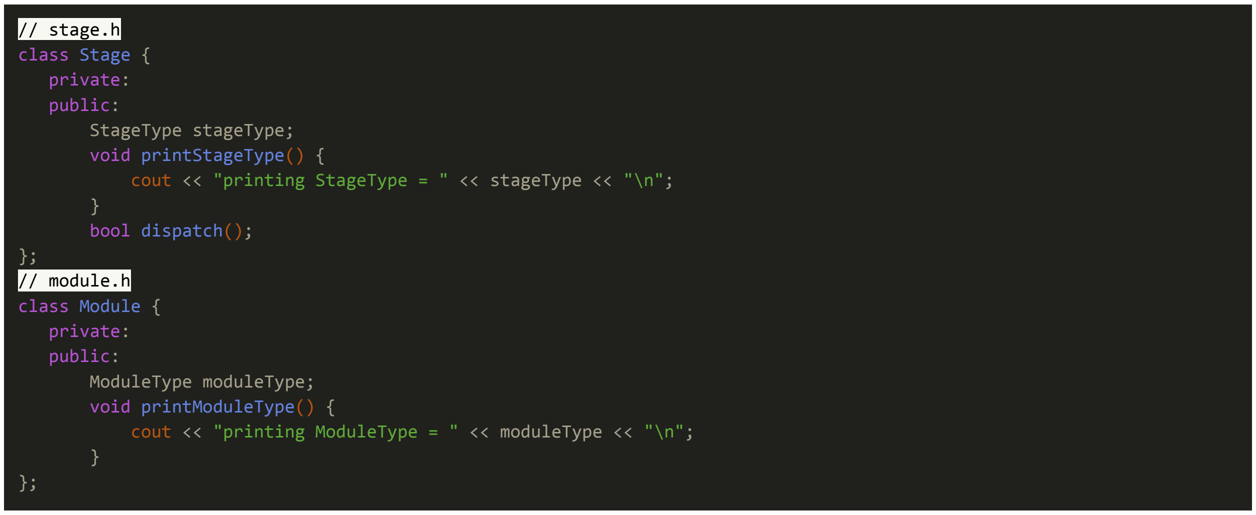
To allow the simulator design to be as modular as possible, various data structures were designed and employed throughout the project. To globally scope these structures, the common header and .cpp files were created, and the common header was included within all the primary classes in the project. The contents of the common header file are displayed in *Figure 1* and outlines the global declarations for the project. There are many self-explanatory, enumerated types, three important structs, and many global helper functions. The Instruction struct object is used throughout the project to define the contents of a particular instruction. The ROBStatus struct represents an entry in the reorder buffer, and the RSStatus struct defines a reservation station entry contained within a reservation station unit. The ROBStatus and RSStatus structs were based on the course material covered during class [2].



```
// global macros
#define REGISTER_COUNT 32
#define ZERO_REGISTER_NAME "$0"
// enumerations
enum InstructionType {
   FLD, FSD, ADD, ADDI, SLT, FADD, FSUB, FMUL, FDIV, BNE
};
enum BranchPredictionType {
   WEAK_TAKEN, STRONG_TAKEN, WEAK_NOT_TAKEN, STRONG_NOT_TAKEN
};
enum StageType {
   FETCH, DECODE, ISSUE, EXECUTE, MEMORY, WRITE_BACK
};
enum InstructionStatusType {
   ISSUED, EXECUTING, WRITING_RESULT, COMMIT
};
enum ModuleType {
   BRANCH_PREDICTOR, ALU
};
// structs
struct Instruction {
   int address;
   InstructionType opcode;
   string rd;
   string rs;
   string rt;
   int immediate;
};
struct ROBStatus {
   bool busy = false;
   Instruction instruction; // opcode = type, rd = dest
   InstructionStatusType status;
   double value = NAN;
   string entryName;
   int countLatency = -1;
};
struct RSStatus {
   bool busy = false;
   Instruction instruction;
   string vj = "";
   string vk = "";
   string qj = "";
   string qk = "";
   string destination;
   int a = -1;
};
// helpers
bool operator==(Instruction const &lhs, Instruction const &rhs);
bool operator!=(Instruction const &lhs, Instruction const &rhs);
string toUpper(string inpString);
InstructionType getInstructionTypeFromString(const string inpString);
string getStringFromInstructionType(const InstructionType inpInstrType);
void printInstruction(Instruction inpInstr);
void printInstructions(deque<Instruction> inpInstrs);
bool containsOffset(const string inpString);
int getOffset(const string inpString);
string trimOffset(const string inpString);
string appendOffset(const string inpString, const int offset);
void printROBStatus(ROBStatus inpEntry);
void printRSStatus(RSStatus inpRs);
string getRSNameFromInstructionType(InstructionType inpInstrType);
```

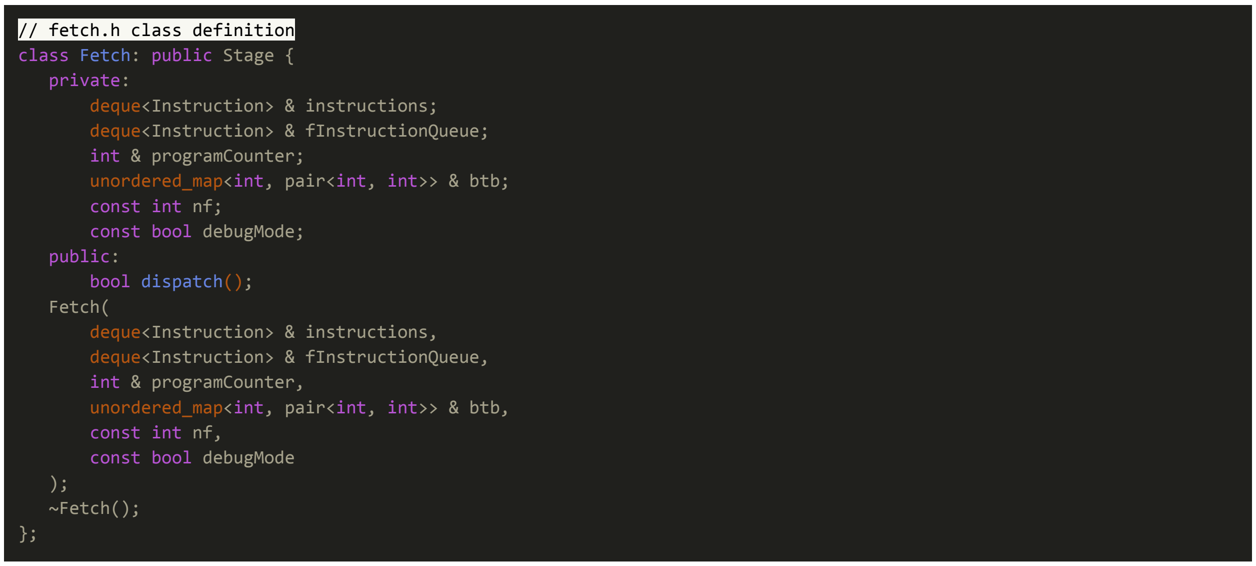
Figure 1: the common.h file, which defines the project's global values, enums, structs, and helper functions.

The project defines multiple classes, which all (barring the simulator class) extend from two possible base classes named stage and module. These base classes are respectively defined in the stage and module header files and are displayed in *Figure 2*. Then, a new class was created for each distinct stage required for the simulator—fetch, decode, issue, and execute. The stage classes' corresponding class definitions are displayed in *Figure 3 - Figure 6.* The class definitions are useful for listing the simulator data structures that are "wired" into each stage. There are two classes defined that extend the module class-–branchPredictor and alu. The branchPredictor class is referenced within the decode and execute stage, and the alu is initialized within the execute stage (highlighted in *Figure 4* and *Figure 6*). In *Figure 7*, the dynamic branch prediction "2-bit" state machine is described. The alu module can handle all of the required operations based on instruction type, and the alu logic is displayed in *Figure 8*. Finally, a simulator class is defined to contain, interact, and output data for the entire system. The simulator's class definition is displayed in *Figure 9*.



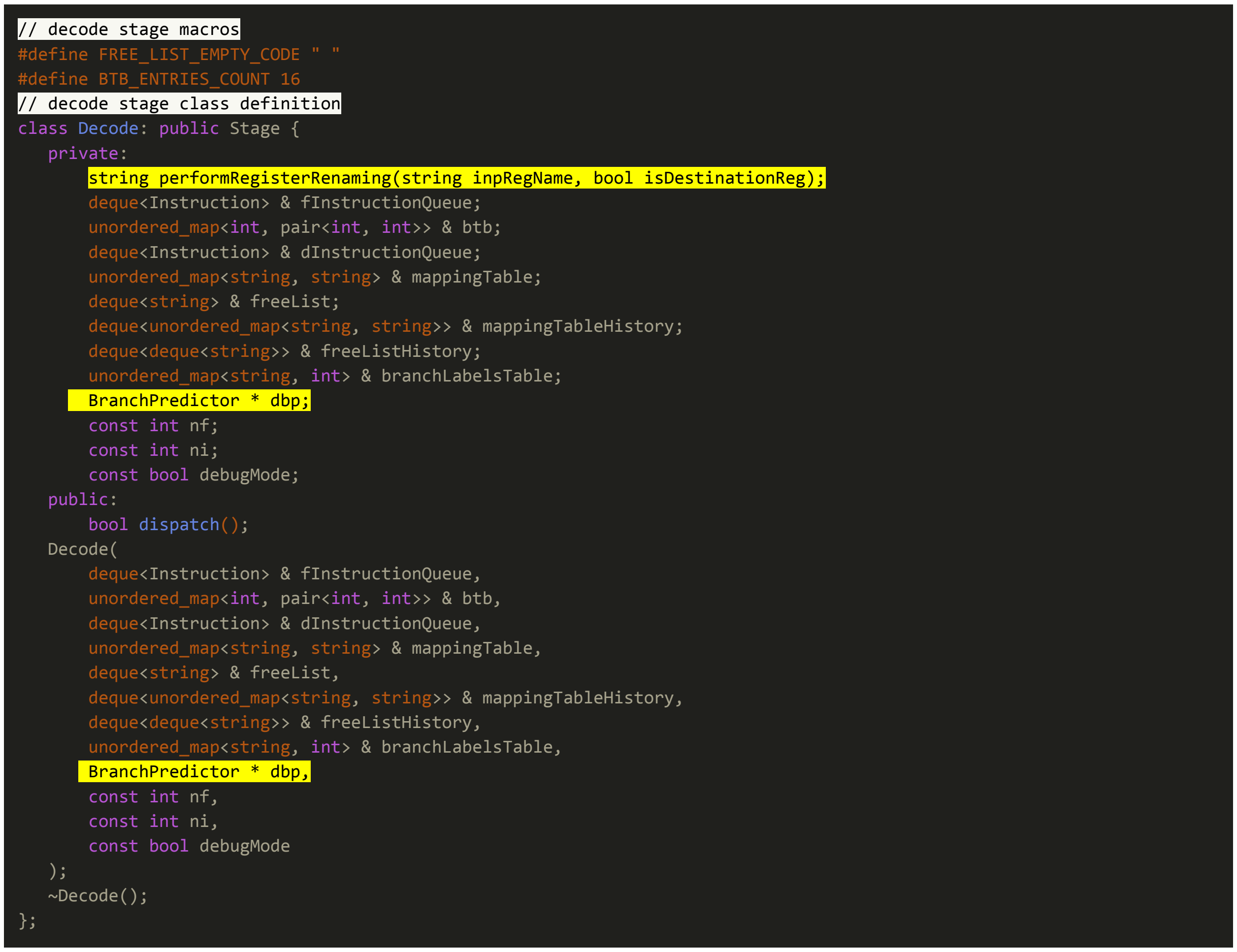
```
// stage.h
class Stage {
   private:
   public:
       StageType stageType;
       void printStageType() {
           cout << "printing StageType = " << stageType << "\n";
       }
       bool dispatch();
};
// module.h
class Module {
   private:
   public:
       ModuleType moduleType;
       void printModuleType() {
           cout << "printing ModuleType = " << moduleType << "\n";
       }
};
```

Figure 2: the stage and module header files, which defines the base classes that all modules/stages extend on.



```cpp
// fetch.h class definition
class Fetch: public Stage {
   private:
       deque<Instruction> & instructions;
       deque<Instruction> & fInstructionQueue;
       int & programCounter;
       unordered_map<int, pair<int, int>> & btb;
       const int nf;
       const bool debugMode;
   public:
       bool dispatch();
   Fetch(
       deque<Instruction> & instructions,
       deque<Instruction> & fInstructionQueue,
       int & programCounter,
       unordered_map<int, pair<int, int>> & btb,
       const int nf,
       const bool debugMode
   );
   ~Fetch();
};
```

Figure 3: The fetch stage class definition.

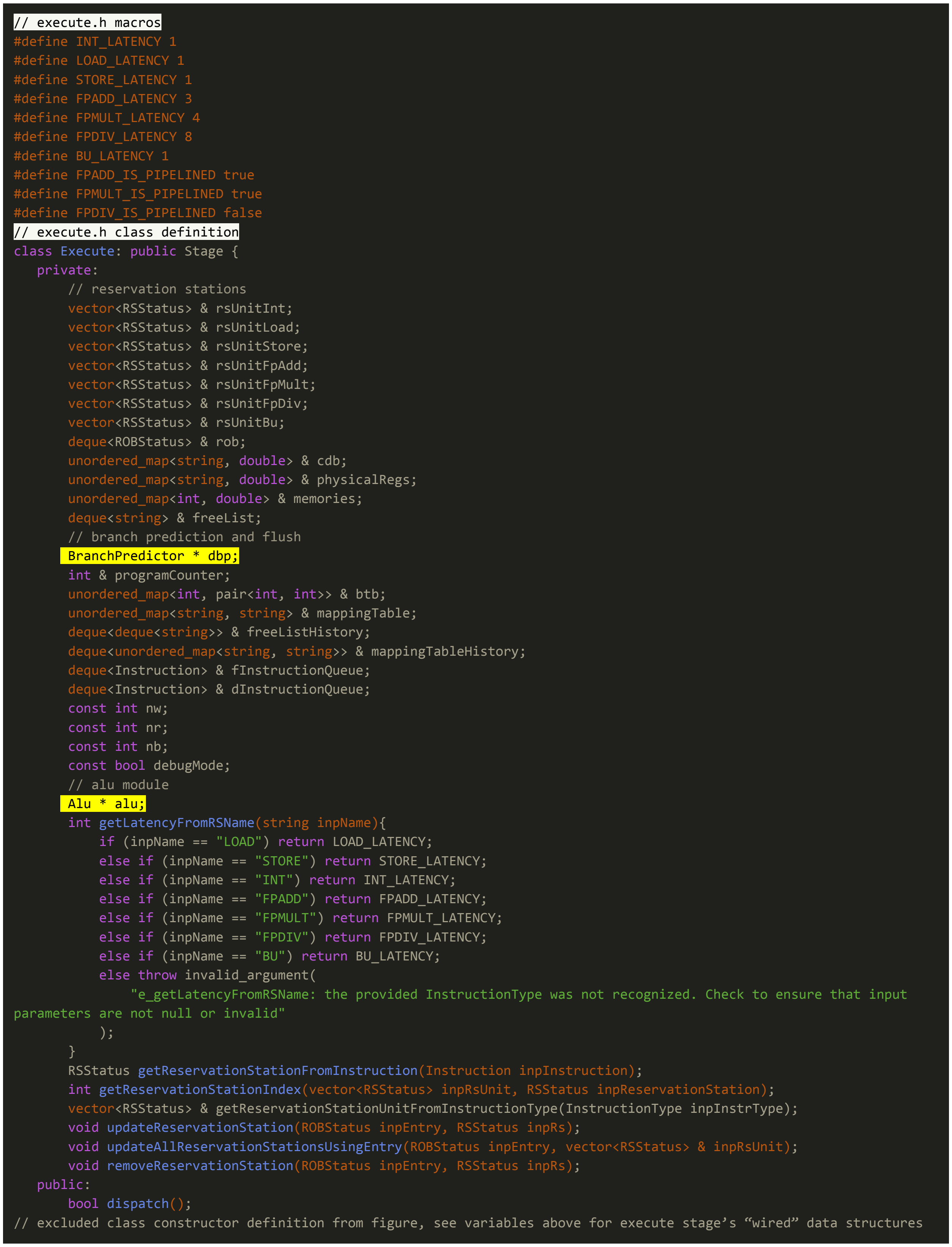


```cpp
// decode stage macros
#define FREE_LIST_EMPTY_CODE " "
#define BTB_ENTRIES_COUNT 16
// decode stage class definition
class Decode: public Stage {
   private:
       string performRegisterRenaming(string inpRegName, bool isDestinationReg);
       deque<Instruction> & fInstructionQueue;
       unordered_map<int, pair<int, int>> & btb;
       deque<Instruction> & dInstructionQueue;
       unordered_map<string, string> & mappingTable;
       deque<string> & freeList;
       deque<unordered_map<string, string>> & mappingTableHistory;
       deque<deque<string>> & freeListHistory;
       unordered_map<string, int> & branchLabelsTable;
       BranchPredictor * dbp;
       const int nf;
       const int ni;
       const bool debugMode;
   public:
       bool dispatch();
   Decode(
       deque<Instruction> & fInstructionQueue,
       unordered_map<int, pair<int, int>> & btb,
       deque<Instruction> & dInstructionQueue,
       unordered_map<string, string> & mappingTable,
       deque<string> & freeList,
       deque<unordered_map<string, string>> & mappingTableHistory,
       deque<deque<string>> & freeListHistory,
       unordered_map<string, int> & branchLabelsTable,
       BranchPredictor * dbp,
       const int nf,
       const int ni,
       const bool debugMode
   );
   ~Decode();
};
```

Figure 4: The decode stage class definition. The branch predictor provides the decode stage with a dynamic prediction. Additionally, the decode stage internally performs register renaming.

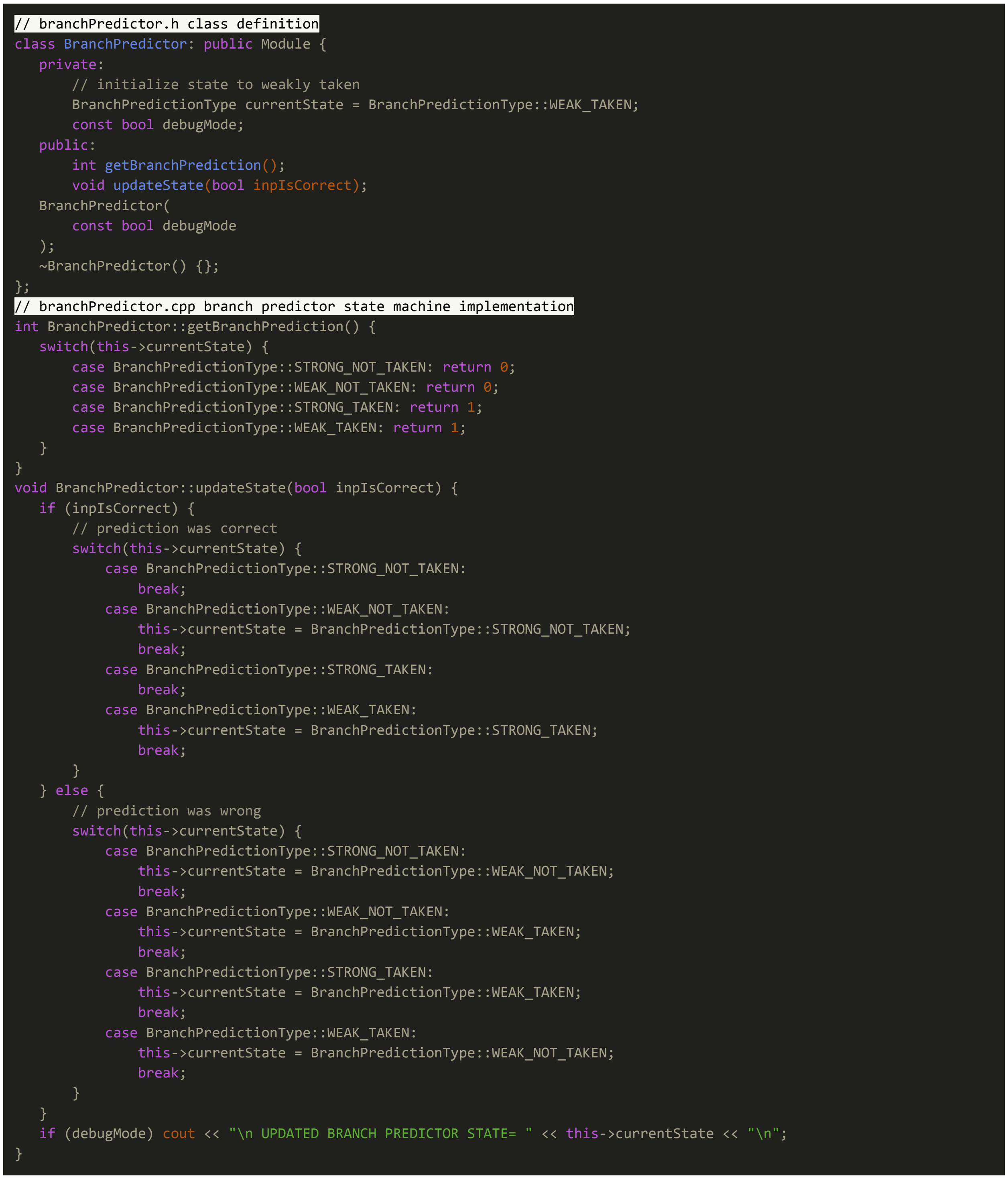
```
// issue.h macros
#define RS_COUNT_INT 4
#define RS_COUNT_LOAD 2
#define RS_COUNT_STORE 2
#define RS_COUNT_FPADD 3
#define RS_COUNT_FPMULT 3
#define RS_COUNT_FPDIV 2
#define RS_COUNT_BU 2
// issue.h class definition
class Issue: public Stage {
   private:
       deque<Instruction> & dInstructionQueue;
       // reservation stations
       vector<RSStatus> & rsUnitInt;
       vector<RSStatus> & rsUnitLoad;
       vector<RSStatus> & rsUnitStore;
       vector<RSStatus> & rsUnitFpAdd;
       vector<RSStatus> & rsUnitFpMult;
       vector<RSStatus> & rsUnitFpDiv;
       vector<RSStatus> & rsUnitBu;
       deque<ROBStatus> & rob;
       const int nw;
       const int nr;
       const bool debugMode;

       bool insertInstructionInReservationStation(
           vector<RSStatus> & inpReservationStation,
           int inpReservationStationCount,
           Instruction inpInstruction
       );
       RSStatus getReservationStationForInstruction(
           Instruction inpInstruction
       );
       int getReservationStationIndex(
           vector<RSStatus> inpRsUnit,
           RSStatus inpReseravtionStation
       );
       ROBStatus & getROBStatusEntryForInstruction(
           Instruction inpInstruction
       );
       int getROBStatusEntryIndex(
           deque<ROBStatus> inpRob,
           ROBStatus inpEntry
       );
       vector<RSStatus> & getReservationStationUnitFromInstructionType(InstructionType inpInstrType);
       string generateROBStatusEntryName(Instruction inpInstr);
       void updateReservationStation(Instruction inpInstr, RSStatus inpRs);
   public:
       bool dispatch();
   Issue(
       deque<Instruction> & dInstructionQueue,
       vector<RSStatus> & rsUnitInt,
       vector<RSStatus> & rsUnitLoad,
       vector<RSStatus> & rsUnitStore,
       vector<RSStatus> & rsUnitFpAdd,
       vector<RSStatus> & rsUnitFpMult,
       vector<RSStatus> & rsUnitFpDiv,
       vector<RSStatus> & rsUnitBu,
       deque<ROBStatus> & rob,
       const int nw,
       const int nr,
       const bool debugMode
   );
   ~Issue();
};
```

Figure 5: The issue stage class definition. Instructions are taken from the decode instruction queue every cycle, and if possible, they are inserted into their corresponding reservation stations.



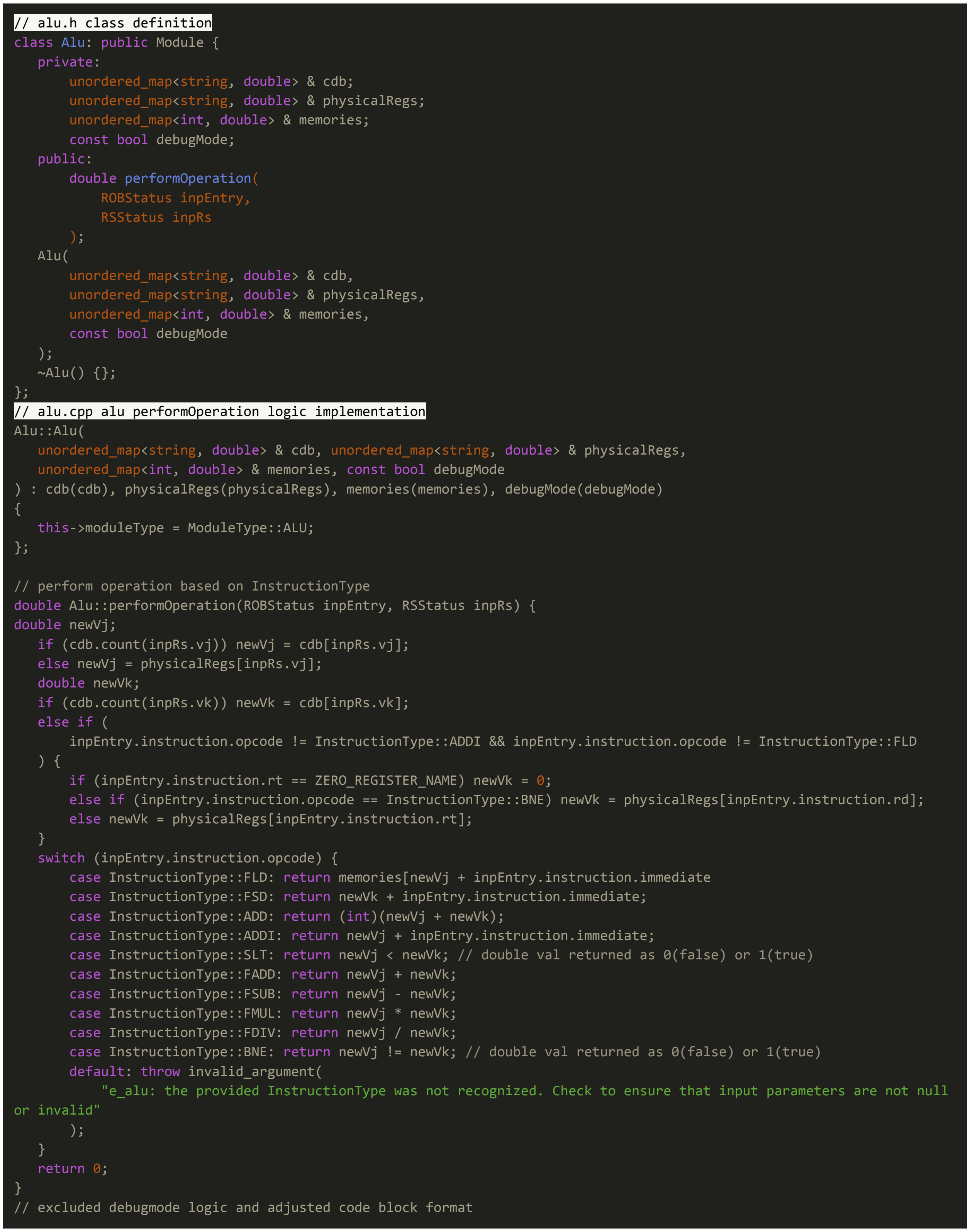
```
// execute.h macros
#define INT_LATENCY 1
#define LOAD_LATENCY 1
#define STORE_LATENCY 1
#define FPADD_LATENCY 3
#define FPMULT_LATENCY 4
#define FPDIV_LATENCY 8
#define BU_LATENCY 1
#define FPADD_IS_PIPELINED true
#define FPMULT_IS_PIPELINED true
#define FPDIV_IS_PIPELINED false
// execute.h class definition
class Execute: public Stage {
   private:
        // reservation stations
        vector<RSStatus> & rsUnitInt;
        vector<RSStatus> & rsUnitLoad;
        vector<RSStatus> & rsUnitStore;
        vector<RSStatus> & rsUnitFpAdd;
        vector<RSStatus> & rsUnitFpMult;
        vector<RSStatus> & rsUnitFpDiv;
        vector<RSStatus> & rsUnitBu;
        deque<ROBStatus> & rob;
        unordered_map<string, double> & cdb;
        unordered_map<string, double> & physicalRegs;
        unordered_map<int, double> & memories;
        deque<string> & freeList;
        // branch prediction and flush
        BranchPredictor * dbp;
        int & programCounter;
        unordered_map<int, pair<int, int>> & btb;
        unordered_map<string, string> & mappingTable;
        deque<deque<string>> & freeListHistory;
        deque<unordered_map<string, string>> & mappingTableHistory;
        deque<Instruction> & fInstructionQueue;
        deque<Instruction> & dInstructionQueue;
        const int nw;
        const int nr;
        const int nb;
        const bool debugMode;
        // alu module
        Alu * alu;
        int getLatencyFromRSName(string inpName){
            if (inpName == "LOAD") return LOAD_LATENCY;
            else if (inpName == "STORE") return STORE_LATENCY;
            else if (inpName == "INT") return INT_LATENCY;
            else if (inpName == "FPADD") return FPADD_LATENCY;
            else if (inpName == "FPMULT") return FPMULT_LATENCY;
            else if (inpName == "FPDIV") return FPDIV_LATENCY;
            else if (inpName == "BU") return BU_LATENCY;
            else throw invalid_argument(
                "e_getLatencyFromRSName: the provided InstructionType was not recognized. Check to ensure that input 
parameters are not null or invalid"
            );
        }
        RSStatus getReservationStationFromInstruction(Instruction inpInstruction);
        int getReservationStationIndex(vector<RSStatus> inpRsUnit, RSStatus inpReservationStation);
        vector<RSStatus> & getReservationStationUnitFromInstructionType(InstructionType inpInstrType);
        void updateReservationStation(ROBStatus inpEntry, RSStatus inpRs);
        void updateAllReservationStationsUsingEntry(ROBStatus inpEntry, vector<RSStatus> & inpRsUnit);
        void removeReservationStation(ROBStatus inpEntry, RSStatus inpRs);
   public:
        bool dispatch();
// excluded class constructor definition from figure, see variables above for execute stage's "wired" data structures
```

Figure 6: The execute stage class definition. The branch predictor module has its current state updated from the execute stage based on whether the predictor is correct. The alu module is instantiated within the execute stage to handle any of the determined instruction operations.



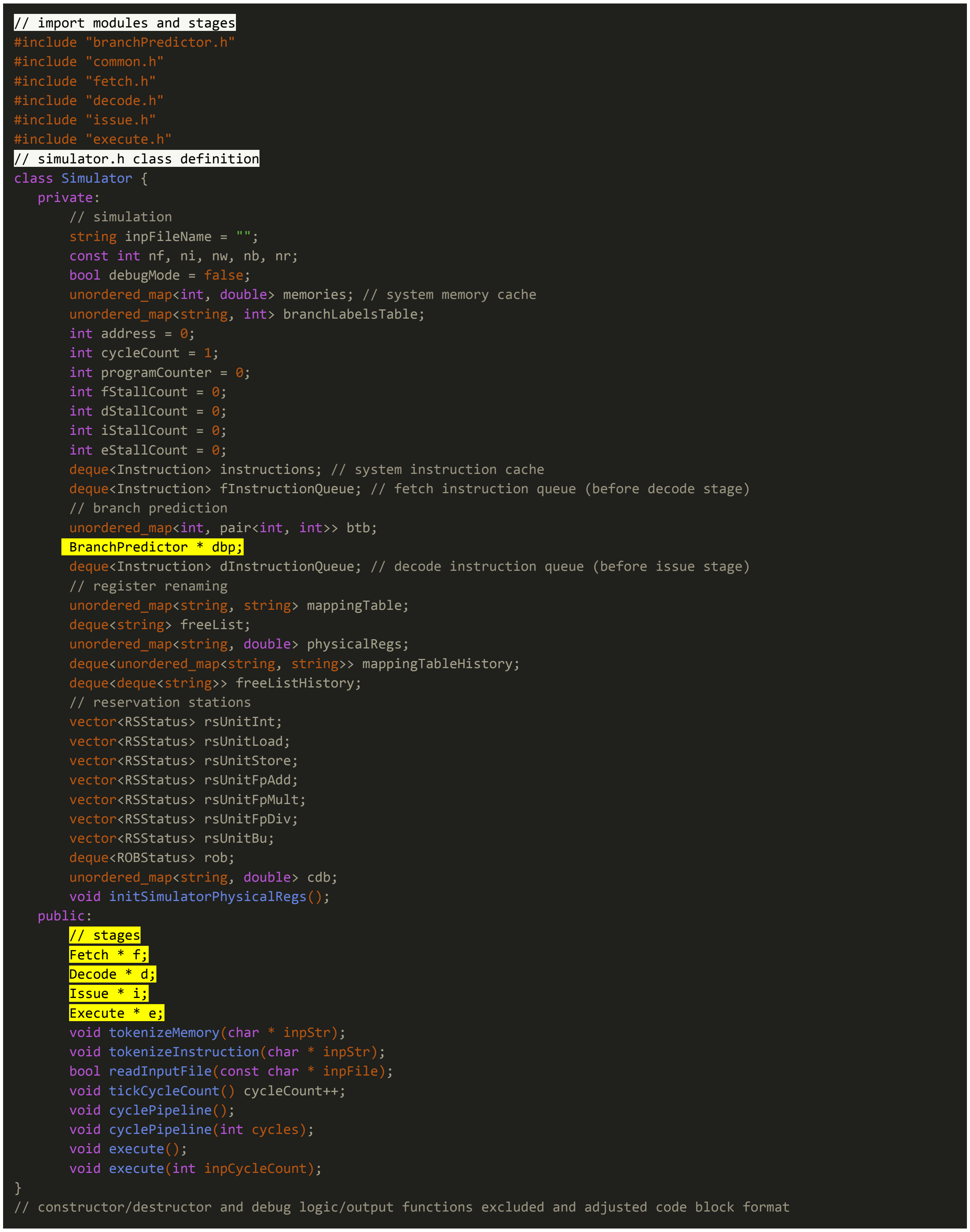
```cpp
// branchPredictor.h class definition
class BranchPredictor: public Module {
   private:
       // initialize state to weakly taken
       BranchPredictionType currentState = BranchPredictionType::WEAK_TAKEN;
       const bool debugMode;
   public:
       int getBranchPrediction();
       void updateState(bool inpIsCorrect);
   BranchPredictor(
       const bool debugMode
   );
   ~BranchPredictor() {};
};
// branchPredictor.cpp branch predictor state machine implementation
int BranchPredictor::getBranchPrediction() {
   switch(this->currentState) {
       case BranchPredictionType::STRONG_NOT_TAKEN: return 0;
       case BranchPredictionType::WEAK_NOT_TAKEN: return 0;
       case BranchPredictionType::STRONG_TAKEN: return 1;
       case BranchPredictionType::WEAK_TAKEN: return 1;
   }
}
void BranchPredictor::updateState(bool inpIsCorrect) {
   if (inpIsCorrect) {
       // prediction was correct
       switch(this->currentState) {
           case BranchPredictionType::STRONG_NOT_TAKEN:
               break;
           case BranchPredictionType::WEAK_NOT_TAKEN:
               this->currentState = BranchPredictionType::STRONG_NOT_TAKEN;
               break;
           case BranchPredictionType::STRONG_TAKEN:
               break;
           case BranchPredictionType::WEAK_TAKEN:
               this->currentState = BranchPredictionType::STRONG_TAKEN;
               break;
       }
   } else {
       // prediction was wrong
       switch(this->currentState) {
           case BranchPredictionType::STRONG_NOT_TAKEN:
               this->currentState = BranchPredictionType::WEAK_NOT_TAKEN;
               break;
           case BranchPredictionType::WEAK_NOT_TAKEN:
               this->currentState = BranchPredictionType::WEAK_TAKEN;
               break;
           case BranchPredictionType::STRONG_TAKEN:
               this->currentState = BranchPredictionType::WEAK_TAKEN;
               break;
           case BranchPredictionType::WEAK_TAKEN:
               this->currentState = BranchPredictionType::WEAK_NOT_TAKEN;
               break;
       }
   }
   if (debugMode) cout << "\n UPDATED BRANCH PREDICTOR STATE= " << this->currentState << "\n";
}
```

Figure 7: the branch prediction module class definition and the dynamic branch predictor's state machine implementation.



```cpp
// alu.h class definition
class Alu: public Module {
   private:
       unordered_map<string, double> & cdb;
       unordered_map<string, double> & physicalRegs;
       unordered_map<int, double> & memories;
       const bool debugMode;
   public:
       double performOperation(
           ROBStatus inpEntry,
           RSStatus inpRs
       );
   Alu(
       unordered_map<string, double> & cdb,
       unordered_map<string, double> & physicalRegs,
       unordered_map<int, double> & memories,
       const bool debugMode
   );
   ~Alu() {};
};
// alu.cpp alu performOperation logic implementation
Alu::Alu(
   unordered_map<string, double> & cdb, unordered_map<string, double> & physicalRegs,
   unordered_map<int, double> & memories, const bool debugMode
) : cdb(cdb), physicalRegs(physicalRegs), memories(memories), debugMode(debugMode)
{
   this->moduleType = ModuleType::ALU;
};

// perform operation based on InstructionType
double Alu::performOperation(ROBStatus inpEntry, RSStatus inpRs) {
double newVj;
   if (cdb.count(inpRs.vj)) newVj = cdb[inpRs.vj];
   else newVj = physicalRegs[inpRs.vj];
   double newVk;
   if (cdb.count(inpRs.vk)) newVk = cdb[inpRs.vk];
   else if (
       inpEntry.instruction.opcode != InstructionType::ADDI && inpEntry.instruction.opcode != InstructionType::FLD
   ) {
       if (inpEntry.instruction.rt == ZERO_REGISTER_NAME) newVk = 0;
       else if (inpEntry.instruction.opcode == InstructionType::BNE) newVk = physicalRegs[inpEntry.instruction.rd];
       else newVk = physicalRegs[inpEntry.instruction.rt];
   }
   switch (inpEntry.instruction.opcode) {
       case InstructionType::FLD: return memories[newVj + inpEntry.instruction.immediate
       case InstructionType::FSD: return newVk + inpEntry.instruction.immediate;
       case InstructionType::ADD: return (int)(newVj + newVk);
       case InstructionType::ADDI: return newVj + inpEntry.instruction.immediate;
       case InstructionType::SLT: return newVj < newVk; // double val returned as 0(false) or 1(true)
       case InstructionType::FADD: return newVj + newVk;
       case InstructionType::FSUB: return newVj - newVk;
       case InstructionType::FMUL: return newVj * newVk;
       case InstructionType::FDIV: return newVj / newVk;
       case InstructionType::BNE: return newVj != newVk; // double val returned as 0(false) or 1(true)
       default: throw invalid_argument(
           "e_alu: the provided InstructionType was not recognized. Check to ensure that input parameters are not null
or invalid"
       );
   }
   return 0;
}
// excluded debugmode logic and adjusted code block format
```

Figure 8: the ALU module class definition.



```
// import modules and stages
#include "branchPredictor.h"
#include "common.h"
#include "fetch.h"
#include "decode.h"
#include "issue.h"
#include "execute.h"
// simulator.h class definition
class Simulator {
   private:
       // simulation
       string inpFileName = "";
       const int nf, ni, nw, nb, nr;
       bool debugMode = false;
       unordered_map<int, double> memories; // system memory cache
       unordered_map<string, int> branchLabelsTable;
       int address = 0;
       int cycleCount = 1;
       int programCounter = 0;
       int fStallCount = 0;
       int dStallCount = 0;
       int iStallCount = 0;
       int eStallCount = 0;
       deque<Instruction> instructions; // system instruction cache
       deque<Instruction> fInstructionQueue; // fetch instruction queue (before decode stage)
       // branch prediction
       unordered_map<int, pair<int, int>> btb;
       BranchPredictor * dbp;
       deque<Instruction> dInstructionQueue; // decode instruction queue (before issue stage)
       // register renaming
       unordered_map<string, string> mappingTable;
       deque<string> freeList;
       unordered_map<string, double> physicalRegs;
       deque<unordered_map<string, string>> mappingTableHistory;
       deque<deque<string>> freeListHistory;
       // reservation stations
       vector<RSStatus> rsUnitInt;
       vector<RSStatus> rsUnitLoad;
       vector<RSStatus> rsUnitStore;
       vector<RSStatus> rsUnitFpAdd;
       vector<RSStatus> rsUnitFpMult;
       vector<RSStatus> rsUnitFpDiv;
       vector<RSStatus> rsUnitBu;
       deque<ROBStatus> rob;
       unordered_map<string, double> cdb;
       void initSimulatorPhysicalRegs();
   public:
       // stages
       Fetch * f;
       Decode * d;
       Issue * i;
       Execute * e;
       void tokenizeMemory(char * inpStr);
       void tokenizeInstruction(char * inpStr);
       bool readInputFile(const char * inpFile);
       void tickCycleCount() cycleCount++;
       void cyclePipeline();
       void cyclePipeline(int cycles);
       void execute();
       void execute(int inpCycleCount);
}
// constructor/destructor and debug logic/output functions excluded and adjusted code block format
```

Figure 9: the simulator class definition. The pipeline stages and branch predictor class objects are data members of the simulator class.

2.2. Product Design

The simulator's pipeline is arranged as a sub-system contained within the simulator class, which is also where all shared data structures are instantiated. Each stage class references various data structures, which are all associated with the same data structures contained within the simulator class. Additionally, the branch predictor module is contained within the simulator class. The instantiation of the simulator's stages and module are highlighted in *Figure 9*. Besides containing the system during runtime, the simulator class allows initializing and interacting with the pipeline. A simulator object becomes instantiated within the program's main function at runtime and is provided the command-line input parameters, which includes the input instruction/memory .dat file. After tokenizing this file, the simulator can initialize the system instruction and memory caches. Then, the simulator class allows for independently cycling the pipeline (once or many times), or indefinitely executing the pipeline until the fetch queue, decode queue, and reorder buffer are all empty. By designing the simulator class to contain the pipeline as a sub-system, the simulator object effectively orchestrates the communication between the stages, modules, and data structures. The system diagram displayed in *Figure 10* illustrates the wiring between the system components comprising the simulator/pipeline classes.

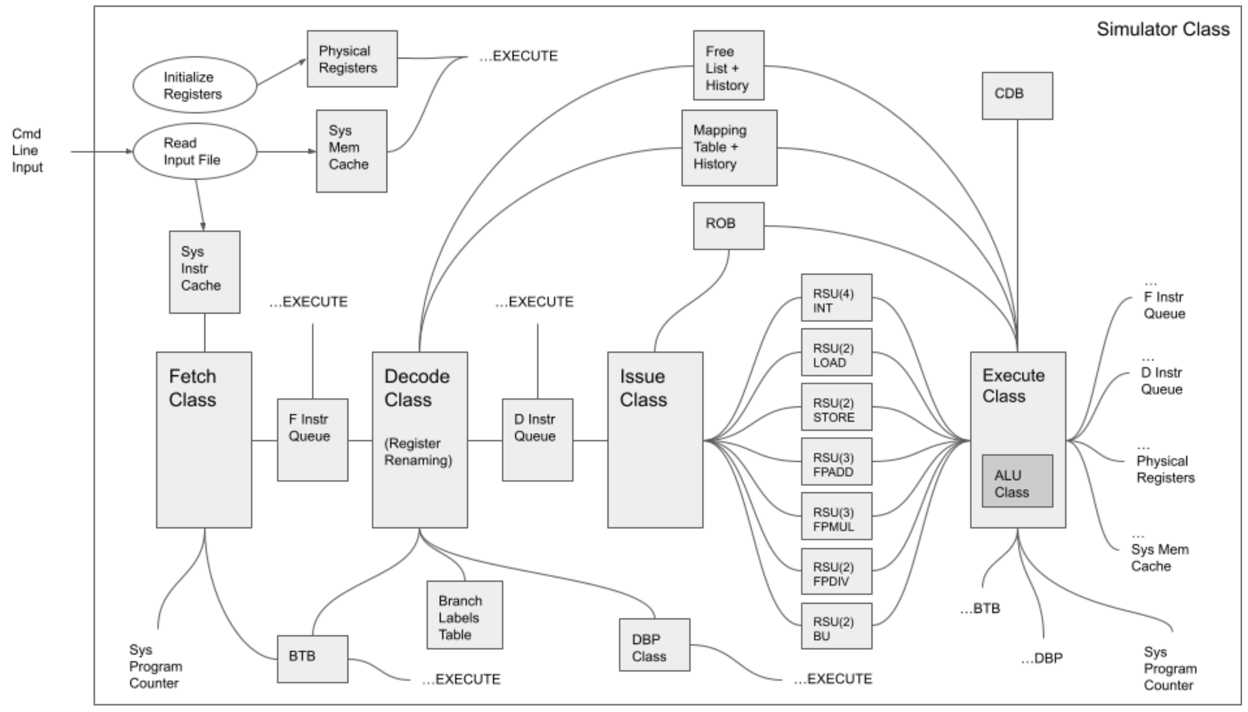


Figure 10: The simulator system diagram, which illustrates the various data structures and their interconnections to form the pipeline sub-system within the simulator..

2.3. How to Run

The project includes a Makefile to handle compilation of the various .cpp files. Furthermore, the Makefile can run the program in two different modes—run or debug. The Makefile is displayed in *Figure 11*. The program is compiled using g++ by default, but the compiler can be configured in the Makefile. Additionally, the user may configure their desired values for the project's required input parameters (INP_FILE_NAME, NF, NI, NW, NB, NR) within the Makefile.

The following list of instructions guides the user through **compiling and running the program in "run" mode**:

1. Download the project and unzip the files, if necessary.
2. Traverse into the project directory from the terminal/command window.
3. Run the following commands:

```
make all
make run
```

The following list of instructions guides the user through **compiling and running the program in "debug" mode**, which will provide additional console outputs about the simulation:

1. Download the project and unzip the files, if necessary.
2. Traverse into the project directory from the terminal/command window.
3. Run the following commands:

```
make all
make debug
```

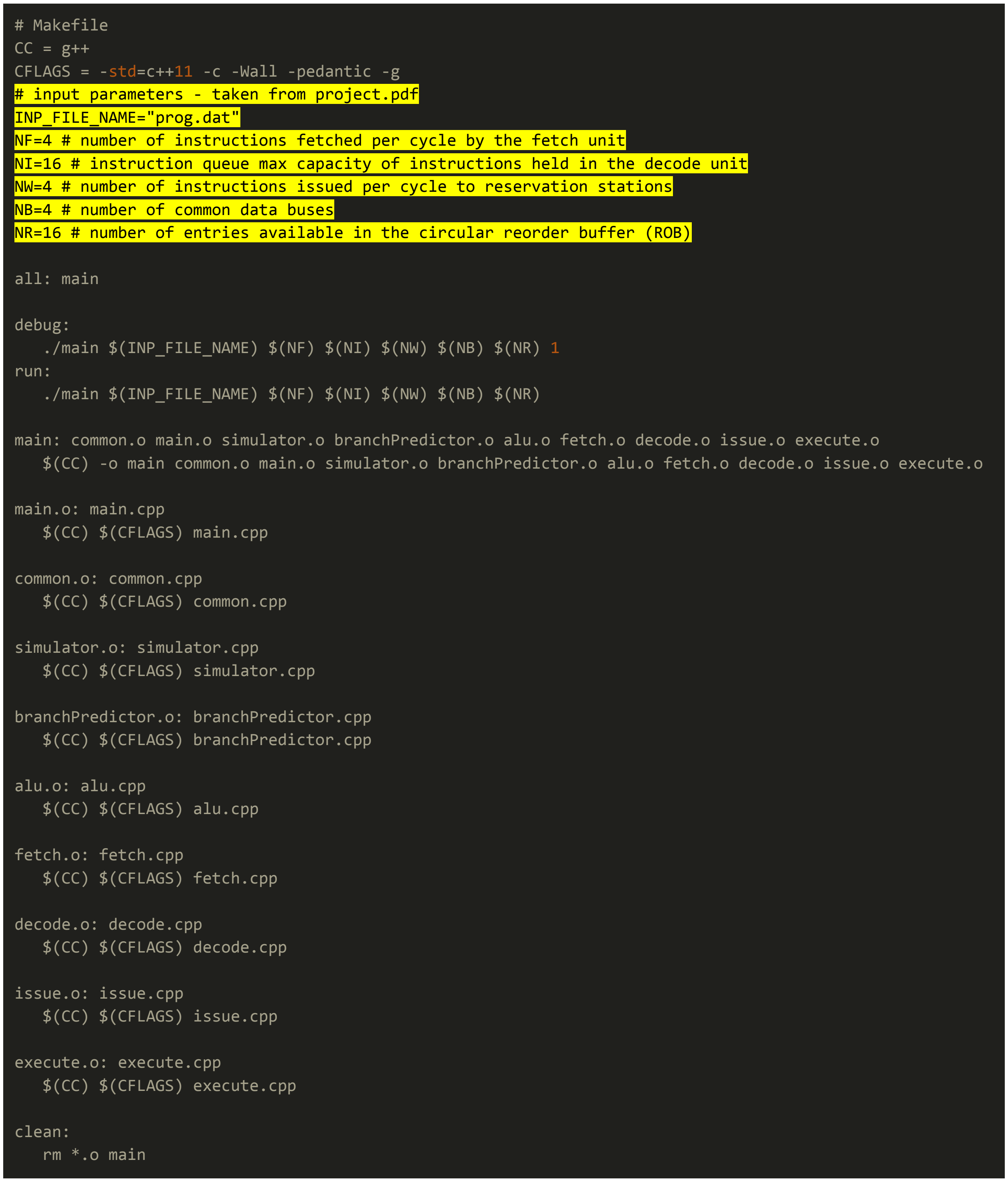
The following list of instructions guides the user through **re-compiling the program**:

1. Download the project and unzip the files, if necessary.
2. Traverse into the project directory from the terminal/command window.
3. Run the following commands:

```
make clean
make all
make run // or make debug
```

If the user would like to **fine-tune the debug console output for the program**, they may adjust the commented/uncommented print functions within the simulator.cpp execute()/ cyclePipeline() functions and then re-compiling the program:

```
// adjust simulator.cpp cyclePipeline function for post-cycle console output
void Simulator::cyclePipeline() {
   // TODO: set up Stage::debug() and use here for each stage
   cout << "\n";
   printSimulatorCurrentCycleCount();
   // run stages backwards
   if (!e->dispatch()) eStallCount++;
   if (!i->dispatch()) iStallCount++;
   if (!d->dispatch()) dStallCount++;
   if (!f->dispatch()) fStallCount++;
   tickCycleCount();
   if (debugMode) {
       cout << "\nPOST CYCLE RESULTS\n";
       // debug per cycle
       // printSimulatorMemories();
       printSimulatorDecodeInstructionQueue();
       // printSimulatorBranchLabelsTable();
       // printSimulatorBtbMap();
       // printSimulatorPhysicalRegs();
       // printSimulatorMappingTable();
       // printSimulatorFreeList();
       // printSimulatorMappingTableHistory();
       // printSimulatorFreeListHistory();
       printSimulatorReservationStations();
       printSimulatorROB();
       // printSimulatorStallCounts();
   }
}
// adjust simulator.cpp execute function for post execution console output
void Simulator::execute() {
   cout << "\nstart\n";
   do cyclePipeline();
   while (fInstructionQueue.size() > 0 || dInstructionQueue.size() > 0 || rob.size() > 0);
   cout << "\ndone\n\nPOST EXECUTE RESULTS:\n";
   // post execution
   printSimulatorMemories();
   // printSimulatorDecodeInstructionQueue();
   // printSimulatorBranchLabelsTable();
   // printSimulatorBtbMap();
   printSimulatorPhysicalRegs();
   printSimulatorMappingTable();
   // printSimulatorFreeList();
   // printSimulatorMappingTableHistory();
   // printSimulatorFreeListHistory();
   // printSimulatorReservationStations();
   // printSimulatorROB();
   printSimulatorStallCounts();
   cout << "\n";
```



```
# Makefile
CC = g++
CFLAGS = -std=c++11 -c -Wall -pedantic -g
# input parameters - taken from project.pdf
INP_FILE_NAME="prog.dat"
NF=4 # number of instructions fetched per cycle by the fetch unit
NI=16 # instruction queue max capacity of instructions held in the decode unit
NW=4 # number of instructions issued per cycle to reservation stations
NB=4 # number of common data buses
NR=16 # number of entries available in the circular reorder buffer (ROB)

all: main

debug:
   ./main $(INP_FILE_NAME) $(NF) $(NI) $(NW) $(NB) $(NR) 1
run:
   ./main $(INP_FILE_NAME) $(NF) $(NI) $(NW) $(NB) $(NR)

main: common.o main.o simulator.o branchPredictor.o alu.o fetch.o decode.o issue.o execute.o
   $(CC) -o main common.o main.o simulator.o branchPredictor.o alu.o fetch.o decode.o issue.o execute.o

main.o: main.cpp
   $(CC) $(CFLAGS) main.cpp

common.o: common.cpp
   $(CC) $(CFLAGS) common.cpp

simulator.o: simulator.cpp
   $(CC) $(CFLAGS) simulator.cpp

branchPredictor.o: branchPredictor.cpp
   $(CC) $(CFLAGS) branchPredictor.cpp

alu.o: alu.cpp
   $(CC) $(CFLAGS) alu.cpp

fetch.o: fetch.cpp
   $(CC) $(CFLAGS) fetch.cpp

decode.o: decode.cpp
   $(CC) $(CFLAGS) decode.cpp

issue.o: issue.cpp
   $(CC) $(CFLAGS) issue.cpp

execute.o: execute.cpp
   $(CC) $(CFLAGS) execute.cpp

clean:
   rm *.o main
```

Figure 11: The program's Makefile specification. The user may pre-configure the program's input parameters using the command line arguments highlighted in the Makefile.

3. Comparative Analysis

3.1. Lower Issue and Commit Width (NW=2, NB=2)

	Cycle Count	Fetch Stalls	Decode Stalls	Issue Stalls	Execute Stalls
Default Parameters	19	0	0	4	1
3.1	20	0	0	3	1

Console Output: See Appendix 5.2.

3.2. Lower Fetch and Decode Width (NF=2)

	Cycle Count	Fetch Stalls	Decode Stalls	Issue Stalls	Execute Stalls
Default Parameters	19	0	0	4	1
3.2	20	0	0	3	1

Console Output: See Appendix 5.3.

3.3. Lower Instruction Queue Capacity (NI=4)

	Cycle Count	Fetch Stalls	Decode Stalls	Issue Stalls	Execute Stalls
Default Parameters	19	0	0	4	1
3.3	19	0	4	4	1

Console Output: See Appendix 5.4.

3.4. Varying Reorder Buffer Capacity (NR=4, NR=8, NR=32)

	Cycle Count	Fetch Stalls	Decode Stalls	Issue Stalls	Execute Stalls
Default Parameters	19	0	0	4	1
3.4.1	X	X	X	X	X
3.4.2	19	0	0	4	1
3.4.3	19	0	0	4	1

Console Output: See Appendix 5.5.

As mentioned earlier in this report, as well as in the project assignment document [3], a dynamically scheduled processor is tolerable to changes in latency or issue capability. The results for this section seem to indicate this.

4. References

[1] PowerPC 605 RISC Microprocessor Documentation, https://www.nxp.com/docs/en/data-sheet/MPC604.pdf

[2] CS2410 Computer Architecture: 8-SpeculativeOOO.pdf (slide 132), Xulong Tang

[3] CS2410 Computer Architecture Spring 2023 Course Project.pdf, Xulong Tang

[4] Computer Architecture, A Quantitative Approach (Chapter 3), John L. Hennessy

[5] RISCV-Simulator, Alex Chi, https://github.com/skyzh/RISCV-Simulator

[6] “Tomasulo Algorithm Take 3”, @jreiss999 (YouTube), https://www.youtube.com/watch?v=zS9ngvUQPNM&t=0s

[7] Project Source Code, Jacob Hoffman, https://github.com/Jacob-Hoff-man/cs2410

5. Appendix

5.1. Default Console Output

```
./main "prog.dat" 4  16  4  4  16
start
cycleCount = 1
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 2
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 3
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 4
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 5
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 6
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 7
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 8
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 9
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 10
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 11
e_dispatch called (nw=4 nr=16 nb=4)=
stall e because branch prediction was incorrect
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 12
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 13
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 14
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 15
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 16
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 17
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 18
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 19
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
done
POST EXECUTE RESULTS:
cycleCount = 19
MEMORIES CACHE (size=9)
   124 128
   116 3
   108 27
   200 12
   24 10
   16 5
   8 14
   100 2
   0 111
```

```
PHYSICAL REGS (size=32)
    name=p30
    val=0
    name=p29
    val=0
    name=p28
    val=0
    name=p27
    val=0
    name=p31
    val=0
    name=p24
    val=0
    name=p23
    val=0
    name=p22
    val=0
    name=p20
    val=0
    name=p16
    val=0
    name=p25
    val=0
    name=p15
    val=0
    name=p13
    val=0
    name=p18
    val=0
    name=p14
    val=0
    name=p12
    val=0
    name=p6
    val=120
    name=p1
    val=0
    name=p17
    val=0
    name=p11
    val=0
    name=p9
    val=16
    name=$0
    val=0
    name=p5
    val=10
    name=p8
    val=128
    name=p21
    val=0
    name=p4
    val=12
    name=p2
    val=24
    name=p10
    val=116
    name=p26
    val=0
    name=p7
    val=8
    name=p19
    val=0
    name=p3
    val=124
MAPPING TABLE (size=6)
  F4 p7
  F0 p8
  R0 p1
  F2 p4
  R1 p9
  R2 p10
FETCH STAGE STALLS= 0
DECODE STAGE STALLS= 0
ISSUE STAGE STALLS= 4
EXECUTE STAGE STALLS= 1
```

5.2. Lower Issue and Commit Width (NW=2, NB=2) Console Output

```
./main "prog.dat" 4  16  2  2  16
start
cycleCount = 1
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 2
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 3
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 4
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 5
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 6
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 7
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 8
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 9
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 10
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 11
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 12
e_dispatch called (nw=2 nr=16 nb=2)=
stall e because branch prediction was incorrect
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 13
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 14
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 15
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 16
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 17
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 18
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 19
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 20
e_dispatch called (nw=2 nr=16 nb=2)=
i_dispatch called (nw=2 nr=16)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
done
```

```
POST EXECUTE RESULTS:
cycleCount = 20
MEMORIES CACHE (size=9)
  124 128
  116 3
  108 27
  200 12
  24 10
  16 5
  8 14
  100 2
  0 111
PHYSICAL REGS (size=32)
    name=p30
    val=0
    name=p29
    val=0
    name=p28
    val=0
    name=p27
    val=0
    name=p31
    val=0
    name=p24
    val=0
    name=p23
    val=0
    name=p22
    val=0
    name=p20
    val=0
    name=p16
    val=0
    name=p25
    val=0
    name=p15
    val=0
    name=p13
    val=0
    name=p18
    val=0
    name=p14
    val=0
    name=p12
    val=0
    name=p6
    val=120
    name=p1
    val=0
    name=p17
    val=0
    name=p11
    val=0
    name=p9
    val=16
    name=$0
    val=0
    name=p5
    val=10
    name=p8
    val=128
    name=p21
    val=0
    name=p4
    val=12
    name=p2
    val=24
    name=p10
    val=116
    name=p26
    val=0
    name=p7
    val=8
    name=p19
    val=0
    name=p3
    val=124
MAPPING TABLE (size=6)
  F4 p7
  F0 p8
  R0 p1
  F2 p4
  R1 p9
  R2 p10
FETCH STAGE STALLS= 0
DECODE STAGE STALLS= 0
ISSUE STAGE STALLS= 3
EXECUTE STAGE STALLS= 1
```

5.3. Lower Fetch and Decode Width (NF=2) Console Output

```
./main "prog.dat" 2  16  4  4  16
start
cycleCount = 1
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 2
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 3
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 4
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 5
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 6
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 7
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 8
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 9
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 10
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 11
e_dispatch called (nw=4 nr=16 nb=4)=
stall e because branch prediction was incorrect
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 12
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 13
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 14
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 15
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 16
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 17
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 18
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 19
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
cycleCount = 20
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=2 ni=16)=
f_dispatch called (nf=2)=
done
```

```
POST EXECUTE RESULTS:
cycleCount = 20
MEMORIES CACHE (size=9)
  124 128
  116 3
  108 27
  200 12
  24 10
  16 5
  8 14
  100 2
  0 111
PHYSICAL REGS (size=32)
    name=p30
    val=0
    name=p29
    val=0
    name=p28
    val=0
    name=p27
    val=0
    name=p31
    val=0
    name=p24
    val=0
    name=p23
    val=0
    name=p22
    val=0
    name=p20
    val=0
    name=p16
    val=0
    name=p25
    val=0
    name=p15
    val=0
    name=p13
    val=0
    name=p18
    val=0
    name=p14
    val=0
    name=p12
    val=0
    name=p6
    val=120
    name=p1
    val=0
    name=p17
    val=0
    name=p11
    val=0
    name=p9
    val=16
    name=$0
    val=0
    name=p5
    val=10
    name=p8
    val=128
    name=p21
    val=0
    name=p4
    val=12
    name=p2
    val=24
    name=p10
    val=116
    name=p26
    val=0
    name=p7
    val=8
    name=p19
    val=0
    name=p3
    val=124
MAPPING TABLE (size=6)
  F4 p7
  F0 p8
  R0 p1
  F2 p4
  R1 p9
  R2 p10
FETCH STAGE STALLS= 0
DECODE STAGE STALLS= 0
ISSUE STAGE STALLS= 3
EXECUTE STAGE STALLS= 1
```

5.4. Lower Instruction Queue Capacity (NI=4) Console Output

```
./main "prog.dat" 4  4  4  4  16
start
cycleCount = 1
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 2
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 3
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 4
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 5
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 6
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 7
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 8
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 9
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 10
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 11
e_dispatch called (nw=4 nr=16 nb=4)=
stall e because branch prediction was incorrect
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 12
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 13
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 14
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 15
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 16
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 17
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 18
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
cycleCount = 19
e_dispatch called (nw=4 nr=16 nb=4)=
i_dispatch called (nw=4 nr=16)=
d_dispatch called (nf=4 ni=4)=
f_dispatch called (nf=4)=
done
```

```
POST EXECUTE RESULTS:
cycleCount = 19
MEMORIES CACHE (size=9)
  124 128
  116 3
  108 27
  200 12
  24 10
  16 5
  8 14
  100 2
  0 111
PHYSICAL REGS (size=32)
    name=p30
    val=0
    name=p29
    val=0
    name=p28
    val=0
    name=p27
    val=0
    name=p31
    val=0
    name=p24
    val=0
    name=p23
    val=0
    name=p22
    val=0
    name=p20
    val=0
    name=p16
    val=0
    name=p25
    val=0
    name=p15
    val=0
    name=p13
    val=0
    name=p18
    val=0
    name=p14
    val=0
    name=p12
    val=0
    name=p6
    val=120
    name=p1
    val=0
    name=p17
    val=0
    name=p11
    val=0
    name=p9
    val=16
    name=$0
    val=0
    name=p5
    val=10
    name=p8
    val=128
    name=p21
    val=0
    name=p4
    val=12
    name=p2
    val=24
    name=p10
    val=116
    name=p26
    val=0
    name=p7
    val=8
    name=p19
    val=0
    name=p3
    val=124
MAPPING TABLE (size=6)
  F4 p7
  F0 p8
  R0 p1
  F2 p4
  R1 p9
  R2 p10
FETCH STAGE STALLS= 0
DECODE STAGE STALLS= 4
ISSUE STAGE STALLS= 4
EXECUTE STAGE STALLS= 1
```

5.5. Varying Reorder Buffer Capacity (NR=4, NR=8, NR=32) Console Output

5.5.1. NR=4

Setting NR=4 results in the program infinitely looping from a stall occurring in the issue stage caused by the reorder buffer size being equal to NR (reorder buffer with max size 4 is full):

```
./main "prog.dat" 4  16  4  4  4
start
cycleCount = 1
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 2
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 3
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 4
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 5
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 6
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 7
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 8
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 9
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 10
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 11
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 12
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 13
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 14
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 15
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 16
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 17
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 18
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 19
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 20
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 21
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 22
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 23
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 24
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 25
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 26
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 27
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 28
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 29
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 30
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 31
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 32
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 33
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 34
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 35
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 36
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 37
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 38
e_dispatch called (nw=4 nr=4 nb=4)=
i_dispatch called (nw=4 nr=4)=
stall i because rob size == nr
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
…
…
…
```

5.5.2. NR=8

```
./main "prog.dat" 4  16  4  4  8
start
cycleCount = 1
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 2
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 3
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 4
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 5
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 6
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 7
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 8
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 9
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 10
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 11
e_dispatch called (nw=4 nr=8 nb=4)=
stall e because branch prediction was incorrect
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 12
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 13
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 14
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 15
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 16
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 17
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 18
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 19
e_dispatch called (nw=4 nr=8 nb=4)=
i_dispatch called (nw=4 nr=8)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
done
```

```
POST EXECUTE RESULTS:
cycleCount = 19
MEMORIES CACHE (size=9)
  124 128
  116 3
  108 27
  200 12
  24 10
  16 5
  8 14
  100 2
  0 111
PHYSICAL REGS (size=32)
    name=p30
    val=0
    name=p29
    val=0
    name=p28
    val=0
    name=p27
    val=0
    name=p31
    val=0
    name=p24
    val=0
    name=p23
    val=0
    name=p22
    val=0
    name=p20
    val=0
    name=p16
    val=0
    name=p25
    val=0
    name=p15
    val=0
    name=p13
    val=0
    name=p18
    val=0
    name=p14
    val=0
    name=p12
    val=0
    name=p6
    val=120
    name=p1
    val=0
    name=p17
    val=0
    name=p11
    val=0
    name=p9
    val=16
    name=$0
    val=0
    name=p5
    val=10
    name=p8
    val=128
    name=p21
    val=0
    name=p4
    val=12
    name=p2
    val=24
    name=p10
    val=116
    name=p26
    val=0
    name=p7
    val=8
    name=p19
    val=0
    name=p3
    val=124
MAPPING TABLE (size=6)
  F4 p7
  F0 p8
  R0 p1
  F2 p4
  R1 p9
  R2 p10
FETCH STAGE STALLS= 0
DECODE STAGE STALLS= 0
ISSUE STAGE STALLS= 4
EXECUTE STAGE STALLS= 1
```

5.5.3. NR=32

```
./main "prog.dat" 4  16  4  4  32
start
cycleCount = 1
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 2
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 3
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 4
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 5
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 6
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 7
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
stall i because rs insert was unsuccessful (LOAD full)
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 8
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 9
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 10
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 11
e_dispatch called (nw=4 nr=32 nb=4)=
stall e because branch prediction was incorrect
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 12
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 13
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 14
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 15
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 16
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 17
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 18
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
cycleCount = 19
e_dispatch called (nw=4 nr=32 nb=4)=
i_dispatch called (nw=4 nr=32)=
d_dispatch called (nf=4 ni=16)=
f_dispatch called (nf=4)=
done
```

```
POST EXECUTE RESULTS:
cycleCount = 19
MEMORIES CACHE (size=9)
  124 128
  116 3
  108 27
  200 12
  24 10
  16 5
  8 14
  100 2
  0 111
PHYSICAL REGS (size=32)
    name=p30
    val=0
    name=p29
    val=0
    name=p28
    val=0
    name=p27
    val=0
    name=p31
    val=0
    name=p24
    val=0
    name=p23
    val=0
    name=p22
    val=0
    name=p20
    val=0
    name=p16
    val=0
    name=p25
    val=0
    name=p15
    val=0
    name=p13
    val=0
    name=p18
    val=0
    name=p14
    val=0
    name=p12
    val=0
    name=p6
    val=120
    name=p1
    val=0
    name=p17
    val=0
    name=p11
    val=0
    name=p9
    val=16
    name=$0
    val=0
    name=p5
    val=10
    name=p8
    val=128
    name=p21
    val=0
    name=p4
    val=12
    name=p2
    val=24
    name=p10
    val=116
    name=p26
    val=0
    name=p7
    val=8
    name=p19
    val=0
    name=p3
    val=124
MAPPING TABLE (size=6)
  F4 p7
  F0 p8
  R0 p1
  F2 p4
  R1 p9
  R2 p10
FETCH STAGE STALLS= 0
DECODE STAGE STALLS= 0
ISSUE STAGE STALLS= 4
EXECUTE STAGE STALLS= 1
```