DESIGNING A RUNTIME SIMULATOR FOR QUEUE CORE PROCESSOR

A
THESIS
Presented to
African University of Science and Technology

in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

By

Dwumfour, Abdullai

Supervised

By

Prof. Ben Abdallah Abderazek

December, 2010

Abuja, Nigeria

DESIGNING A RUNTIME SIMULATOR FOR THE QUEUE CORE PROCESSOR

A THESIS APPROVED BY THE COMPUTER SCIENCE

DEPARTMENT

RECOMMENDED:

_____

Dr. Ben A. Abdallah, Advisory Committee Chair

_____

Dr. Ekpe Okorafor

_____

Dr. Alpheus Igbokoyi

_____

Dr. Guy Degla

APPROVED :

_____

Chief Academic Officer

_____

Date

ACKNOWLEDGEMENT

What shall I say to Allah, the omnipotent, the most gracious and ever merciful? All I have to say is ***"Allahu Akbar".*** I thank Allah, the Almighty for his ineffable guidance, wisdom and support for me throughout my education up to this far.

I wish to express my heartfelt gratitude to my supervisor, Prof. Ben Abdallah Abderazek of University of Aizu, Japan for his wonderful supervision, invaluable advice and inspiring encouragements without which this work would not have been a success.

My unreserved appreciation goes to Prof. Mamadou Kaba Traore, University of Blaise Pascal, France, for his constructive guidance and assistance toward this work.

Special thanks go to all the lecturers in the computer science department for nurturing me and impacting their incalculable knowledge into me throughout my studies in AUST. Further acknowledgment goes the entire staff of AUST for diverse supports.

I will like to show a special appreciation to Pete Sanderson and Ken Vollmar, the developers of MARS simulator and James Larus, the developer of SPIM. Their simulators were beneficial to this work.

To my colleague computer science students, I say God bless you all for the interesting and sad moments were shared together throughout our studentship. Furthermore, I cannot understate the brotherly care and support rendered to me by Aminu Mahdi. May Allah bless you abundantly. To Dorothy Maduagwu, I appreciate all the care and times shared together. I shall forever remember you.

This acknowledgement is never complete without mentioning the supports received from my colleague students of AUST.

To my Ghanaian brothers and sisters, I say kudos for strongly standing behind me in the days when all odds were against me. To Juliet Sackey, you have been so wonderful to me. I am very grateful to you all.

DEDICATION

I dedicate this work to my lovely sisters, wonderful niece, special nephews and above all my dear brother-in-law, Mr. Musah Ibrahim. I love you all.

I cannot forget the entire family.

CONTENTS

CHAPTER 2

CHAPTER 3

CHAPTER 5

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ABBREVIATIONS

AUST	African University of Science and Technology
QC-2	Queue Core Processor
QSIM	Queue core SIMulator
QASM	Queue core Assembler
QCM	Queue Computation Model
QC	Queue Core
QCU	Queue Computation Unit
QREG	Queue Register
QH	Queue Head
QT	Queue Tail
PQP	Parallel Queue Processor
FIFO	First In First Out
ILP	Instruction Level Parallelism
CNBR	Number of Consumed Data
PNBR	Number of Produced Data

LQH	Live Queue Head
OPQ	Operand Queue
HDL	Hardware Description Language
FPU	Floating Point Unit
ALU	Arithmetic and Logical Unit
LD	Load
ST	Store
RISC	Reduce Instruction Set Computer
CISC	Complex Instruction Set Computer
BF	Barrier Flag
PC	Program Counter
UML	Unified Modeling language
IDE	Integrated Development Environment
GUI	Graphical User Interface
ISA	Instruction Set Architecture
SRC	Source

ABSTRACT

In today's era of modern architectures, high performance computing has received a lot of attention from diverse architectural levels. The performance crises in the computing arena have forced researchers to look for alternative architectures that will foster high performance whiles minimizing tradeoffs.

The Queue Core processor is a novel 32-bit microprocessor that emerged as a result of the urgent desire for high performance microprocessor. The advent of the Queue Core processor has curb the performance crises due to its interesting features such as high Instruction Level Parallelism (ILP), dense program size, low power consumption, and elimination register concept.

However, the demand in further improving the performance of the Queue Core processor has consequently triggered the growth in complexity. This has imposed a lot of constraints on the evaluation of Queue Processor with regards to timing, computations etc. Understanding of the internal dynamic mechanisms of the Queue Core Processor and their design space exploration therefore rely extensively on simulation tools which are traditionally software.

In this research, we propose QSIM, a trace-driven and runtime simulator for the Queue Core processor. In QSIM, only a subset of the Queue Core Instruction Set Architecture has been implemented using the JAVA programming language.

It is interesting to mention that this research work falls in the cross-road of two different domains: System Architecture and Software Engineering. The research will therefore be accomplished using methodical approach through the competent background knowledge of the Queue Core system architecture and application of Software Engineering principles.

The advents of the QSIM amongst other benefits will principally offer an attractive opportunity to quickly evaluate the performance of the Queue Core Processor and serves as a tool to explore more Queue computation.

CHAPTER ONE (1)

1.0 INTRODUCTION

1.1 OVERVIEW OF QUEUE COMPUTING

Nowadays, the shift in Hardware and Software technology has compelled designers and users to look at micro-architecture that process instructions stream with high performance, low power consumption, and short program length. In striving for high performance, micro-architecture researchers have emphasized Instruction-Level Parallelism (ILP) processing, which has been established in superscalar architectures without major changes to Software. Since the program contains no explicit information about available ILP, it must be discovered by the Hardware, which must then also construct a plan of actions for exploiting parallelism. In short, computers have far achieved this goal at the expense of tremendous Hardware complexity – a complexity that has grown so large as to challenge the industry's ability to deliver ever-higher performance.

Queue Computing emerged as a new paradigm with an attractive alternative seeking to achieve the compulsive demand of high performance in micro-architectures.

Queue Computing in simpler terms is the use of the Queues in processing/computation of data. Queue Computing uses the Queue Computation Model for its data processing.

It has received much attention in recent years as an alternative architecture due to interesting features such as high parallelism capabilities, less instruction set, low complexity etc.

Queue Computation Model (QCM) refers to the evaluation of expressions using a First-In First-Out (FIFO) Queue [9, 11], called Operand Queue. In this model, operands are inserted, or Enqueued, at the Tail of the Queue and removed, or Dequeued, from the Head of the Queue. Two references are needed to track the location of the Head and the Tail of the Queue. The Queue Head (**QH**), points to the head of the Queue. And Queue Tail (**QT**), points to the rear of the Queue.

The concept of Enqueueing and Dequeueing extends to the operations allowed by the QCM. For example, binary and unary operations require two and one operands, respectively, to be Dequeued. After the operation is performed, the result is Enqueued back to the FIFO queue

as illustrated in the diagram below (***a+b*** and **(a+b)/c** are Enqueued after ***ADD*** and ***DIV*** operations respectively). We say that binary and unary operations ***Consume*** (Dequeued) and ***Produce*** (Enqueue) data. Some type of operations is Produce-only such as Load operation. Other types of operations are Consume-only such as Store operation. Queue Code is defined as the set of instructions executed by the QCM.

Fig1 below shows the evaluation of the expression $(a+b)/C$ using the Queue Computation Model

QUEUE COMPUTING

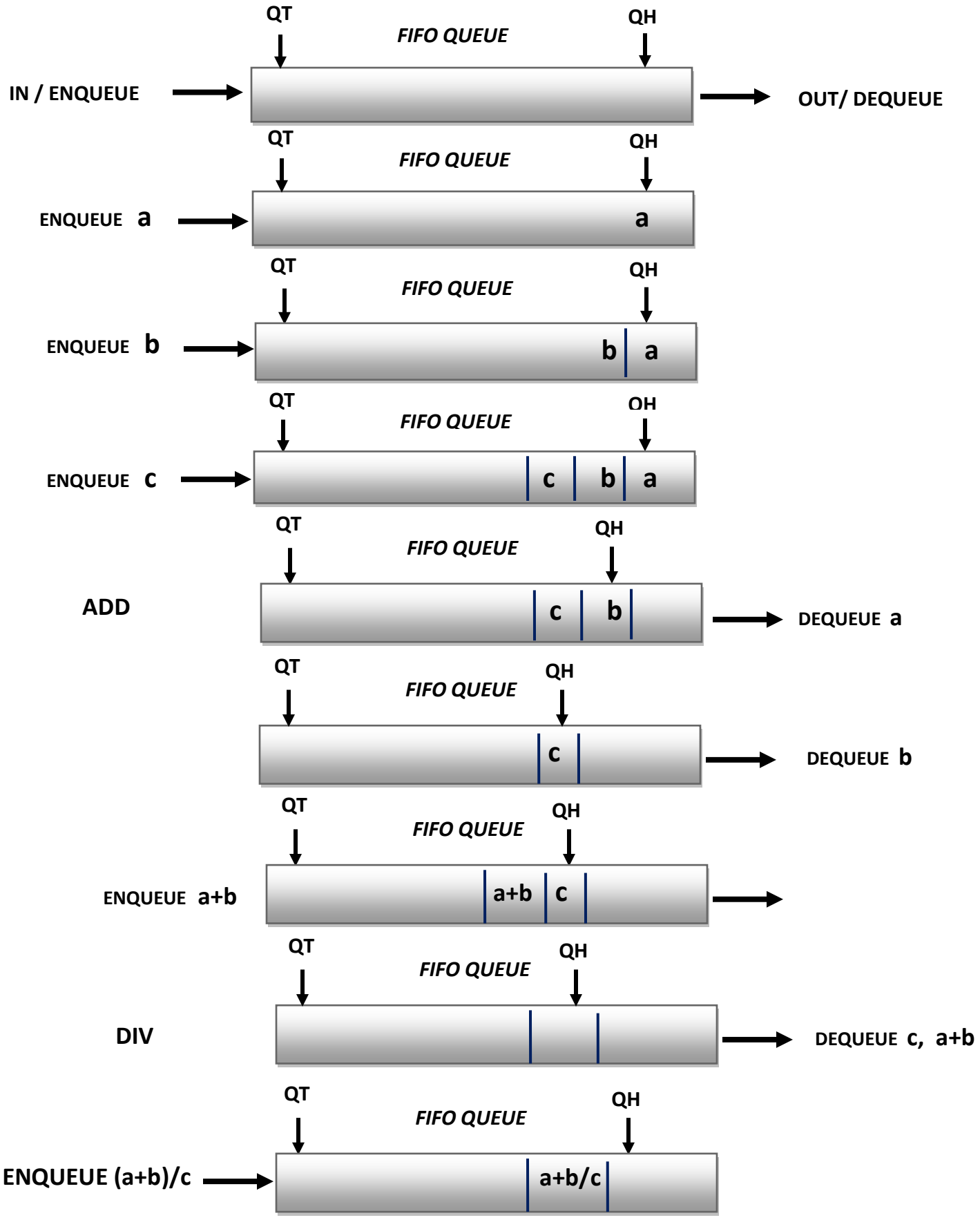


Fig 1:QUEUE COMPUTING EVALUATION

1.2 OVERVIEW OF THE QUEUE CORE PROCESSOR

The quest for high performance in traditional register architecture led to the development of complex compiler and Hardware technique for the exploitation of Instruction Level Parallelism (ILP) [6] in microprocessors. Instruction Level Parallelism is the key to improve the performance of modern architectures. Several Hardware and compiler techniques have been proposed including provision of more architectural registers, modulo scheduling, usage of register windows among others in order to improve the performance of related architectures [5].

Upon the relentless efforts made by several proposed architectural techniques among those above, performance was still a major setback in those architectures. This has accelerated the urgent desire of microprocessor designers to seek for alternative architectures that can extract maximum parallelism and foster high performance whiles minimizing tradeoffs.

The Queue Core Processor emerged as an alternative microprocessor to combat the performance crises in the Computing world. It is an entirely new Processor based on the Producer Order Queue Computation Model.
The key ideas of the Produced Order Queue Computation Model are the operands and results manipulation schemes.

1.2.1 PRODUCED ORDER QUEUE COMPUTATION MODEL

The Produced Order Queue Computing Model uses a circular Queue Register (Operand Queue) instead of Random Access Register to store intermediate results. Data is inserted in a Queue Register (QREG) in a Produced Order Scheme and can be reused.

A special register, called Queue Head pointer (***QH),*** points to the first data in the QREG. Another pointer, named Queue Tail pointer (***QT***), points to the location of the QREG in which the result is stored. Data Dequeued at QH are called Consume data whiles data Enqueued at the QT are called Produced data. In the Queue Computation Model, Produced data can be reused (given forth the name Produced Order Queue Computation Model).

Live Queue Head pointer (***LQH***) is also used to keep used data that could be reused and thus should not be overridden. These data, which are found between QH and LQH pointers, are called

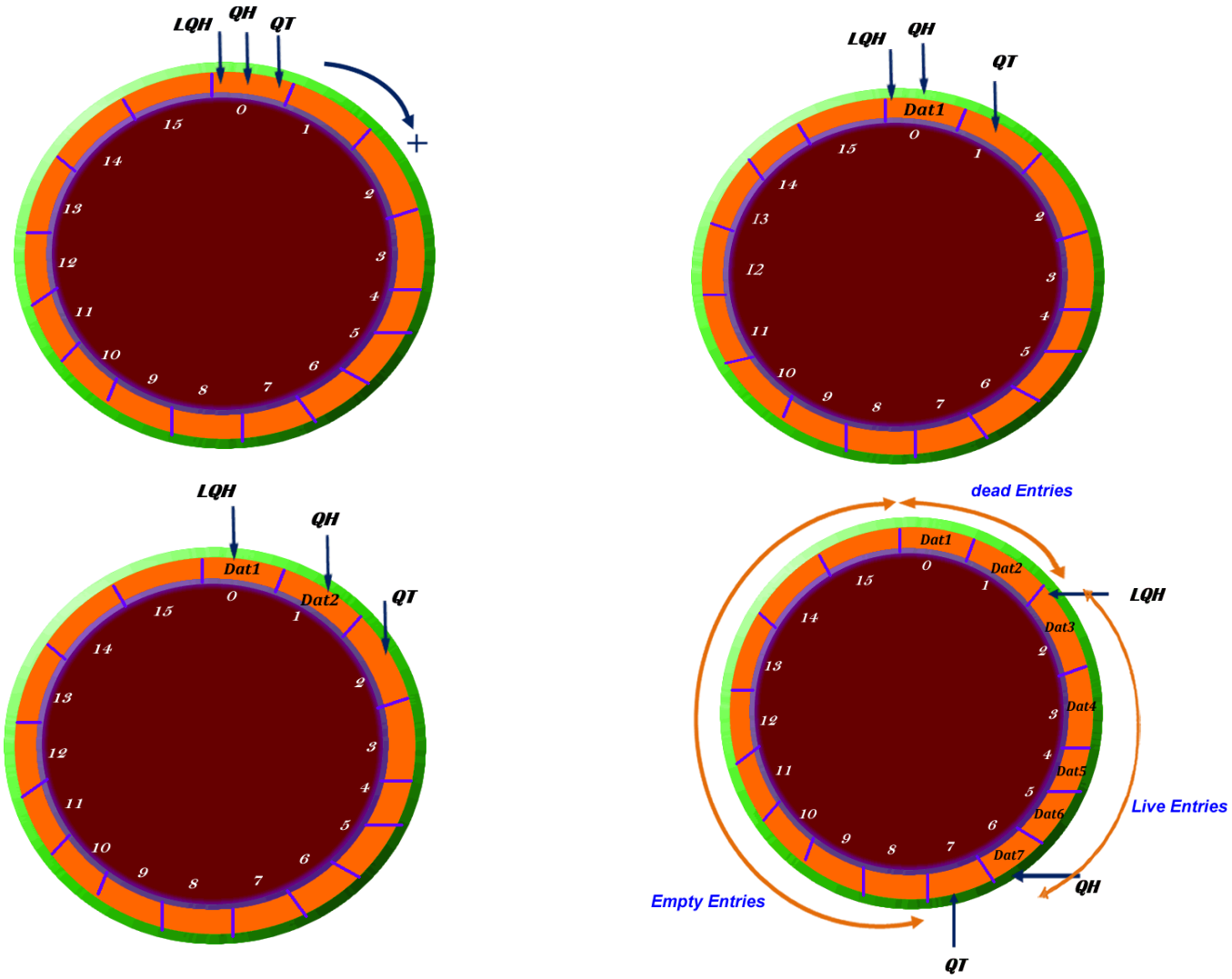
Live-Data (data still reusable). The L*ive-Data* entries in the QREG are statically controlled. *Dead-Entries* signify those data that are no longer needed and can be overridden. *Empty-Entries* contained no data.

Two special instructions are used to *stop* or *release* the LQH pointer. *Stplqh* (stop LQH) instruction is implemented to stop the automatic movement of LQH whiles *autlqh* (automatic LQH) instruction is used to release the LQH pointer.

Immediately after using the data, the QH is incremented so that it points to the data for the next instruction. QT is also incremented after the result is stored.

The diagram below show the structure of the Circular Queue Register (QREG) showing the QH, QT, LQH, Live Entries, Dead Entries.

FIG 2:CIRCULAR QUEUE REGISTER STRUCTURE



The Queue Core Processor offers attractive features like reduction in code size since operands are implicitly specified, the concept of registers does not exist here, hence eliminating register renaming and avoiding false dependencies.

The extraction of maximum Instruction Level Parallelism is a key feature since its programs are constructed from Level-order Traversal. It has small instruction window and consumes less power. These features are depicted below:

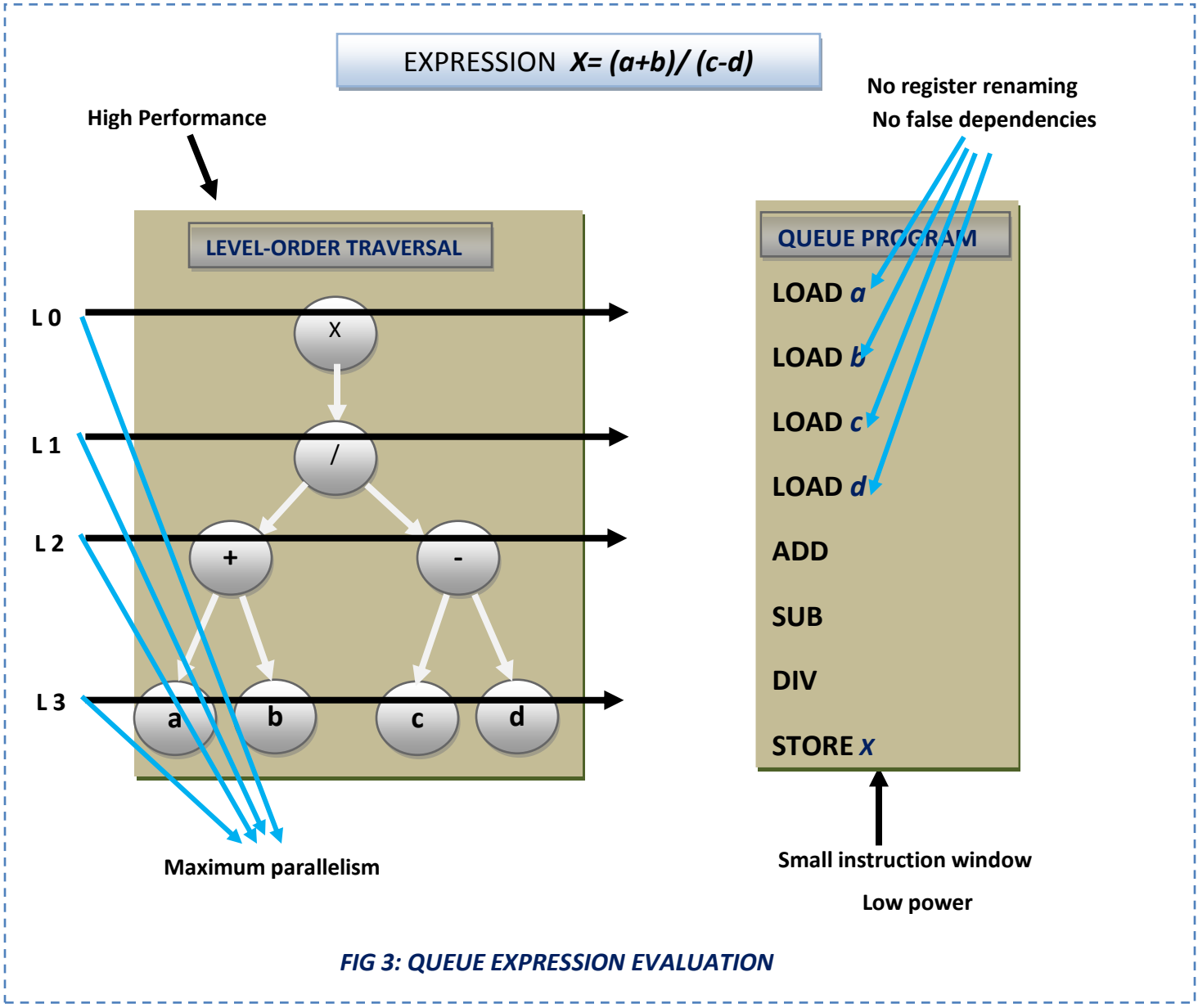


FIG 3: QUEUE EXPRESSION EVALUATION

1.2.2 QUEUE CORE ARCHITECTURE

The QC-2 Processor is a 32-bit microprocessor which implements a Produced Order Instruction Set architecture. All instructions are 16-bit wide and each instruction can encode at most two (2) operands specifying the location in the operand queue from where to read the operands.

Below is the Queue Core Processor architecture block diagram showing clearly all the core modules and how they are interconnected. The block diagram primarily contains the Queue Core Memory, Instruction Pipeline Units, Execution Units and Architectural registers (Circular Queue Registers).

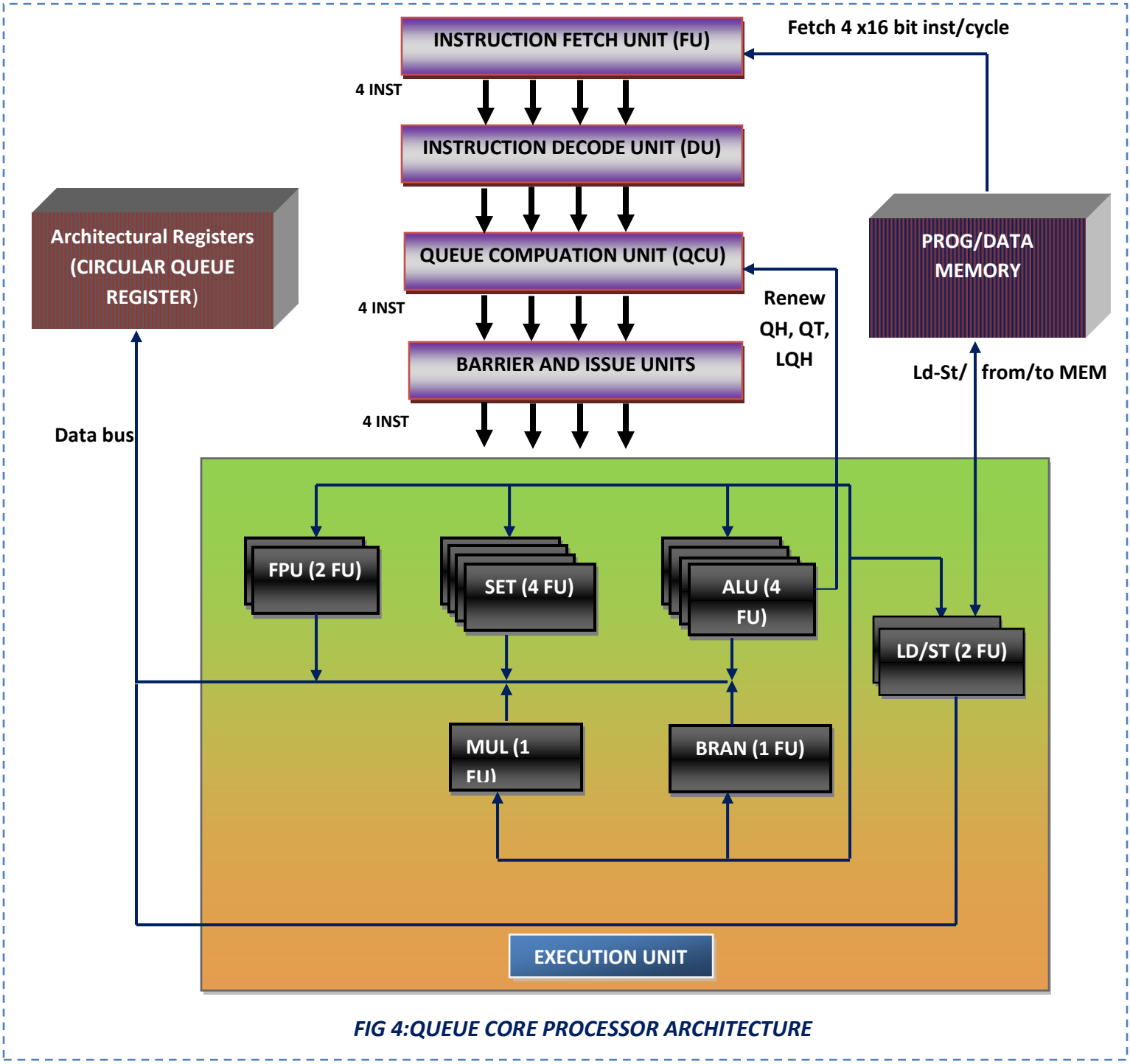


FIG 4:QUEUE CORE PROCESSOR ARCHITECTURE

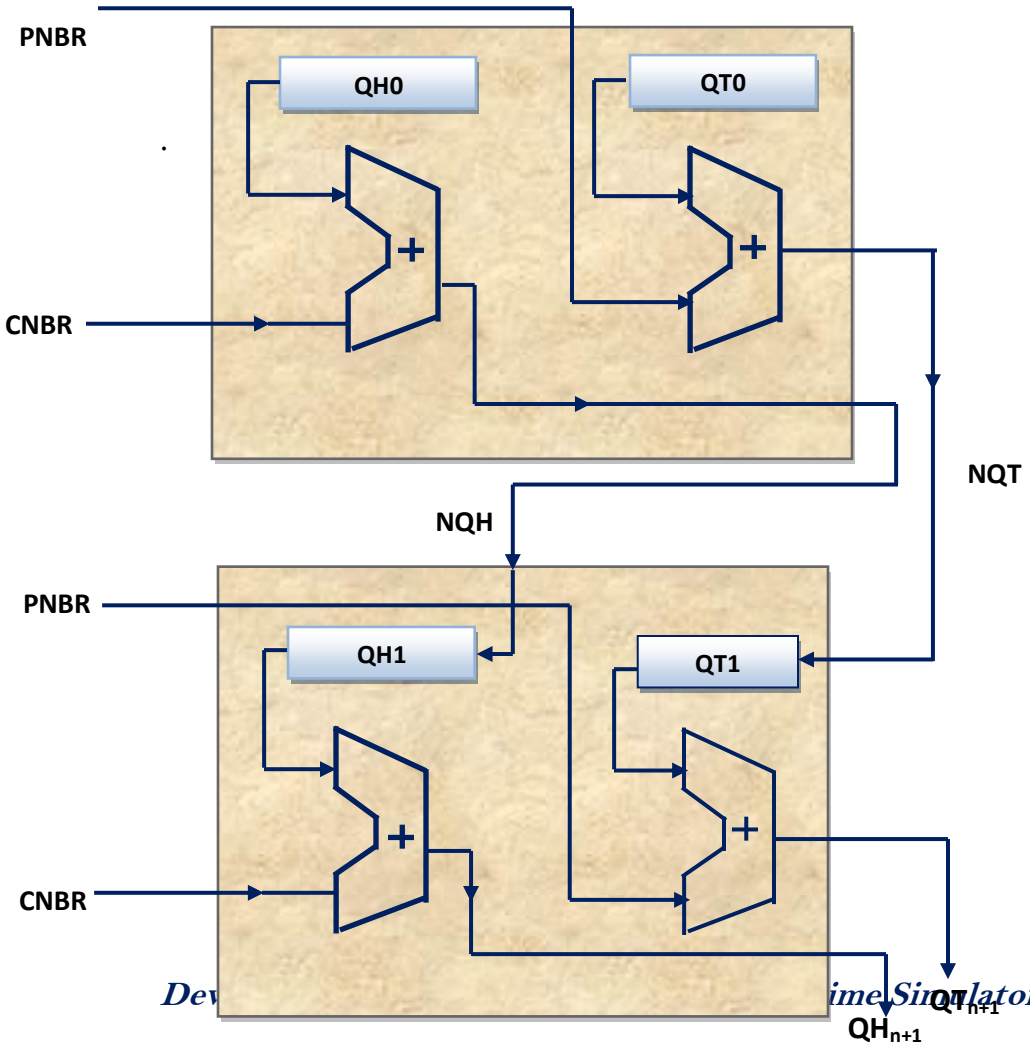
The Queue Core Processor has six Pipeline Stages combined with five pipeline-buffers to smoothen the flow of instructions through the pipeline [3].

Fetch Unit (FU): The pipeline begins with the Fetch Stage which fetches instruction from the Instruction Memory and delivers four instructions to the Decode unit each cycle.

Decode Unit (DU): The DU decodes four instructions in parallel during the second phase and writes them into the Decode buffer. This stage also calculates the number of Consumed (CNBR) and Produced (PNBR) data for each instruction. The CNBR and PNBR are used by the next pipeline stage to calculate the sources (*source1* and *source2*) and destination locations for each instruction.

Queue Computation Unit (QCU): The QCU is a unique computation Unit that distinguishes the Queue Core Processor's pipeline stages from registers based Processors. Four instructions arrive at the QCU unit each cycle. The QCU calculates the first operand (*source1*) and destination addresses (DEST) for each instruction. It is interesting to mention that these addresses are calculated dynamically by the Hardware mechanism in the QCU.

The figure below shows the Hardware mechanism for source1 address calculation by the QCU:



DESCRIPTION

PNBR: Number of Produced data

CNBR: Number of Consumed data

QH0: Initial Queue Head Value

QT0: Initial Queue Tail Value

NQH: Next Queue Head Value

NQT: Next Queue Tail Value

QH_{n+1}: Next Queue Head Value

QT_{n+1}: Next Queue Head Value

FIG 5: SOURCE 1 ADDRESS HARDWARE

***Barrier and Issue Units*:** These are other special units that characterize the Queue Core Processor from others. The Barrier Unit inserts barrier flags for dependency resolutions.

The Issue Unit (IU) on the other hand issues four (4) instructions for execution each cycle. In this stage, the second operand (*source2*) of a given instruction is first calculated by adding the address of *source1* to the displacement that comes with the instruction. The second operand address calculation is performed in the QCU stage. However, for a balanced pipeline consideration, the *source2* is calculated at the beginning of the Issue Stage.

The Hardware mechanism for the source2 address calculation is shown below:

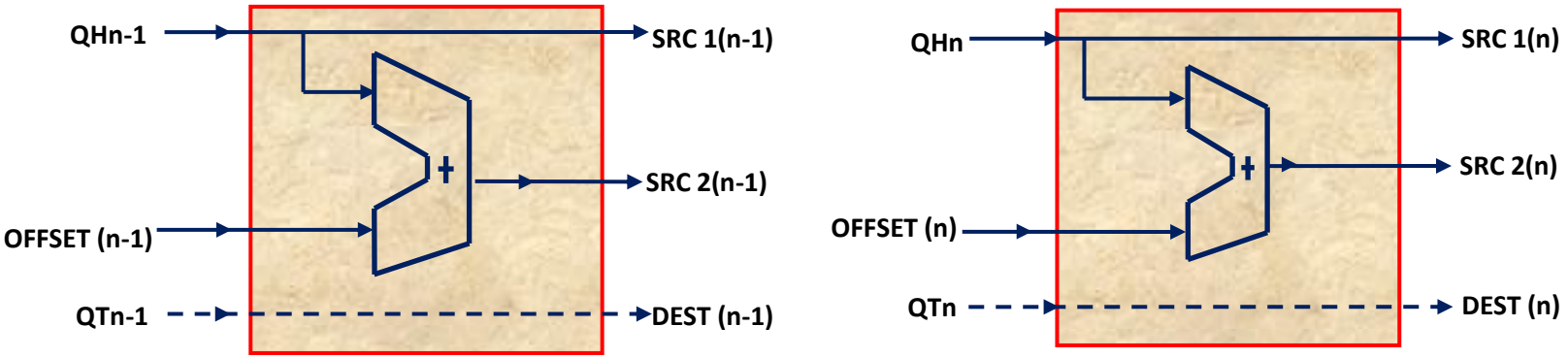


FIG 6: SOURCE 2 ADDRESS HARDWARE

DESCRIPTION

OFFSET: +/- Integer value that indicate the location of SRC2 (n-1) from QH (n-1)

QT (n): Queue Tail value of instruction n, **DEST (n):** Destination location of instruction n

SRC 1: Source address 1 of instruction (n-1), **SRC 2:** Source address 2 of instruction (n-1)

The Processor reads the operands from the Queue register, and execution begins in the next stage.

***Execution Unit* (*EU*):** The macro data flow execution core consists of four Integer ALU units, two Floating-point units, one Branch unit, one Multiply unit, four Set units, and two Load/Store units. These units are shown in the QC-2 block diagram above (Fig 4). The results from the execution phase is either stored in Memory or written into the Queue Register.

It is worth mentioning that the Circular Queue register is one of the most distinguishing features of the QC-2 from other Processor architectures. It totally eliminates the need for random access registers hence eliminating false dependencies and register renaming.

1.3 RESEARCH PROBLEM STATEMENT

Queue Core Processor has demonstrated high performance against conventional microprocessors (RISC & CISC).

As complexity of QC-2 increases due to the desire to further enhance the performance and throughput, there is the apparent need to explore the QC-2 and incorporate other features. The dynamics and characteristics of QC-2 will have to be studied based on the new added features. It becomes very pertinent to execute and debug Queue programs in order to evaluate Queue computations. The performance of the Queue Processor needs to be evaluated.

Undoubtedly, the actual QC-2 Hardware will not be the best platform to evaluate these features. Hence, we need a virtual system to implement the above features and functionalities before affecting them on the actual Hardware.

Many modern Processor architectures unlike the Queue Core Processor have simulators that will emulate them and render these functionalities.

This thesis was therefore inspired by the need to design a simulator that will emulate the Queue Core Processor.

1.4 RESEARCH GOAL

The goal of this research is to design Runtime Simulator for the Queue Core Processor.

1.5 RESEARCH OBJECTIVES

This research seeks to achieve the following objectives

- To study and understand the general underlying principles of simulators, and their benefits most especially in the design of microprocessors
- To study and understand the internal dynamic mechanism of the QC-2 microprocessor
- To model the system architecture of the QC-2 microprocessor
- Design the conceptual model of the QC-2 simulator (QSIM)
- Based on the conceptual design, build a simulator that will emulate the QC-2 microprocessor and implement a subset of the QC-2 Instruction Set Architecture

1.6 EXPECTED OUTPUT OF THIS RESEARCH

When this research is completed successfully, the following are expected:

QSIM

- would be designed for the QC-2 Processor
- would be able to load and execute Assembly Language programs for the QC-2 Processor
- would provide the enabling platform for studying the interactions between the components of the QC-2 Processor
- would offer an attractive opportunity to quickly evaluate the performance of the Queue Processor
- would serve as a tool to explore more Queue computation

1.7 OVERVIEW OF SIMULATORS

Simulation has become a useful and integral part in the modeling of many natural systems in a wide range of fields including Manufacturing, Education, Health, Technology etc. By virtue of the diverse use of the term, it has been defined in many contexts differently.

In [18], simulation has been termed as the imitation of some real thing, state of affairs or process. According to [21], it is the artificial creation of condition in order to study or experiment something that could exist in reality.

Simulation has been viewed in [20, 22] as the mathematical representation of interaction between real-world object.

In spite of the varied use of the terminology, it can be coherently viewed as the act of depicting the structure and functionality of a real system using a virtual system.

The expanding role played by simulation in different arena of life has triggered the desire of computer scientist to embrace the concept. Computer simulation has therefore been defined as the use of a computer-generated system to represent the dynamic responses, structure and behavior of a real proposed system.

In the Computing paradigm, simulation becomes indispensable and desirable when we need to explore and gain insight into new technology and to estimate the performance of systems too complex for analytical solutions. It is used to show eventual effect of alternative conditions and course of actions. More importantly, when the real system cannot be engaged, simulation becomes the ultimate choice.

Simulator can be described as the tool that provides the needed platform for simulation. Computer Simulator stemmed from the term computer simulation and can be viewed as a piece of Software to model computer device or predict outputs and performance metrics on a given input.

Computer simulator provides the opportunity to make a virtual representation of the reality and makes a way to study the dynamics and behavior of this virtual system which depicts the real system.

The tool has attracted wide range of use in the computer science domain as complexity in modern architectures rises due to new technological trend. In order to ride along with the current trend, microprocessor designers have adopted the entire concept. The term emulator has been used to describe the computer simulators that simulate microprocessors. The emulator enables designers to evaluate complex Hardware designs without building. It helps to access non-existent components or systems. Detailed performance metrics can be obtained through this tool. More importantly, it allows the debugging of micro-programs. The dynamics and characteristics of a proposed system can be explored in details. Because they are implemented in Software, not silicon, they can be easily modified to add new instructions, build new systems such as multiprocessors.

In respect of this, many simulators have been built over the years for different microprocessors. These simulators among other benefits are able to read instructions of a particular microprocessor and execute those instruction based on the internal mechanisms of such simulators.

CHATPTER TWO (2)

2.0 RELATED WORKS

Queue machines are abstract definition of a computer that uses First-In –First-Out (FIFO) data structure for intermediate storage of operands for Queue computations. Instructions read and write the ***Operand Queue*** implicitly thus instruction takes its operand from the Queue Head (QH) of the FIFO queue and writes the results to the rear (QT) of the queue.

In [32] Bruno R. Preiss was the first to propose a simple Queue machine, Department of Electrical Engineering, University of Toronto in 1987. Data- flow computer architectures have been suggested as a means to achieve increased computational throughput via the automatic detection and exploitation of potential parallelism. In his thesis, a new approach to data-flow, called ***pseudo-static*** dataflow was described. One way of realizing the pseudo-static data-flow approach was to use a queue machine. It has however been shown that the instruction sequences on a queue machine are natural execution model for computational tasks specified by acyclic data-flow graphs. He identified the problem of evaluating expressions from their Directed Acyclic Graphs (DAGs) [5]. To resolve this problem, he proposed an index Queue machine which uses an index reference to produce and place intermediate results in their right position for correct evaluation of the expression.

His proposed execution model based on the Queue machine supports program reentrancy, recursion, and automatic run-time loop unraveling. By virtue of this, a multiprocessor architecture that supports his execution model was proposed. The proposed architecture has been simulated in Software and a number of programs have been developed and executed to evaluate the performance of this proposed architecture.

In [8], S. Okamoto, University of Electro-Communication, proposed the Hardware design of a superscalar Processor model using the Queue Machine Execution Model. The Queue Model was further used to backup and restore data for loop and procedure calls.

Schmitt proposed another solution on the use a Queue Machine as the execution layer for reconfigurable Hardware. They transformed the program's Data Flow Graph (DFG) into a spatial

representation that can be executed in a simple Queue Machine. This transformation inserts extra special instructions to guarantee correct execution by allowing every variable to be Produced and Consumed only once. Their experiments showed that the execution of programs in their Queue Machine have the potential of exploiting high levels of parallelism while keeping code size less than a RISC Instruction Set.

In [35], further investigation on the usage of Queue Register File for Very Large Instruction Word (VLIW) machines was carried out at the University of Edinburgh in 1998. It was asserted that Instruction Level Parallelism (ILP) is a set of Hardware and Software techniques that allow parallel execution of machine operations. Superscalar architectures rely most heavily upon Hardware schemes to identify parallelism among operations. Although successful in terms of performance, the Hardware complexity involved might limit the scalability of this model. VLIW architectures use a different approach to exploit ILP. In this case all data dependence analyses and scheduling of operations is performed at compile-time, resulting in a simpler Hardware organization. This allows the inclusion of a larger number of Functional Units (FUs) into a single chip. In spite of this relative simplification, the scalability of VLIW architectures can be constrained by the size and number of ports of the Register File. VLIW machines often use Software pipelining techniques to improve the execution of loop structures, which can increase the register pressure. Furthermore, the access time of a Register File can be compromised by the number of ports, causing a negative impact on the machine cycle time.

There was therefore the need to have parallel FUs which will render immense benefits. This motivated the urgent desire to investigate for alternative machine designs. By virtue of this, a scalar VLIW architecture comprising clusters of FUs and private register files was proposed. Register Files organised as Q***ueue Structures*** are used as a mechanism for inter-cluster communication, allowing the enforcement of fixed latency in the process. This scheme presented better possibilities in terms of scalability as the size of the individual Register Files is not determined by the total number of FUs, suggesting that the silicon area may grow only linearly with respect to the total number of FUs.

However, the effectiveness of such an organization depends on the efficiency of the code partitioning strategy. An algorithm for a clustered VLIW architecture integrating both Software pipelining and code partitioning in a single procedure was developed. Experimental results show

it may allow performance levels close to an unclustered machine without communication restraints. Finally, silicon area and cycle time models were developed to quantify the scalability of performance and cost for this class of architecture.

In 1999, the superscalar Queue machine was proposed at the University of Ibaraki.

It is very interesting to mention that none of these researches actually investigated real model or architecture base on Queue until in 1999, when Abderazek, in his PhD proposed a novel Queue Processor model based on Circular Queue Register. He presented the architecture and the design of a Produced Order Queue Processor which uses the Queue Computation Model (QCM).

In [33], 2001 recorded another transformation when superscalar machines with Queue Register File were proposed at the University of Michigan by Gary S., Mikhail S. and Edward S. They identified that many code transformations performed in optimizing compiler trade off an increase in register pressure for some desirable effect such as low instruction count etc. several pipeline schedule approaches were proposed in order to reduce register pressure in Software pipeline loop. Modulo Variable Expansion (MVE) and Rotating Register (RR) were two common register renaming schemes adopted.

MVE is a Software-only approach which gives each simultaneously live variable instances it own name, unrolling the loop body as necessary to ensure that any later uses can directly specify the correct instance. MVE however, both increases the architecture register requirements and expands the loop body to accommodate the renaming constraints of the Software pipelined loop. RR on the other hand is a Hardware-managed register renaming scheme that eliminates the code expansion problem by dynamically renaming the register specifier for each instance of a loop variable. In essence, efficient Software pipeline schedules that account for realistic memory latencies are difficult, and often impossible to achieve with MVE or RR. One solution was to dynamically increase the number of architected registers available through the proposition of new Instruction Set Architecture.

By introducing ***Register Queue (RQs)*** and the ***rq-connect*** instructions the architected register space was no longer a limiting factor in achieving efficient Software pipelined loop schedules. The design of these register queues was derived from the interprocessor queues that support

asynchronous communication in decoupled architecture. Software pipelining using queues has also been studied in VLIW architectures and decoupled Processors. The use of Queue Register File to support the execution of Software pipelined loops in VLIW was proposed.

Carnegie Melon University reported in [34] the usage of a simple Queue Machine for dynamic compilation of Software (SW) to Hardware (HW). It was hypothesized that reconfigurable computing was no more widely used due to the logistical difficulties caused by the close coupling of applications and Hardware platforms. As an alternative, computing machines that use a single, serial instruction representation for the entire reconfigurable application was proposed. It was shown how possible it is to convert, at run-time, the parallel portions of the application into a spatial representation for execution on reconfigurable fabric. The conversion to the spatial representation was facilitated by the use of Instruction Set Architecture base on the queue machine. The techniques to generate code for Queue Machines and Hardware visualization techniques necessary to allow any application to execute on any platform were described.

It was also argued in [34] that a Queue Machine application can be easily mapped to an appropriate Hardware. However, no real Hardware was proposed or designed and only theoretical techniques were proposed to virtualize the Hardware.

The possibility of designing a Processor architecture that could offer simple Hardware, high performance and small code size, while maintaining other characteristics is what led to the development of a Parallel Queue Processor (PQP) architecture [36,37,38], which stores intermediate data in FIFO Queue Registers (Operand Queue (OPQ)) and exploits parallelism dynamically. The assignment of OPQ word obeys *single assignment rule,* where datum is stored at once into the OPQ word.

In [38, 39, 40], Consumed Order Parallel Queue Processor (PQPcv) architecture was proposed which offers much lower Hardware complexity and execution speed than conventional

architectures. PQPcv stores data in the OPQ in *consumed order scheme*. However, PQPcv has several restrictions, which lead to large program size and low execution speed. To overcome this and other architectural design problems, high performance Produced Order Parallel Queue Processor (PQPpv) architecture was proposed in [1, 41, 42]. The PQPpv adopts a new data manipulation technique named *produced order scheme*. In this scheme, data produced by instructions are inserted in the OPQ in their executions order and can be reused by other instructions.

The Queue Execution Model stores intermediate results in a Circular Queue-Register (QREG). A given instruction implicitly reads its first operand from the Head (QH) of the QREG, its second operand from a location explicitly addressed with an offset from the first operand location. The computed result is finally written into the QREG at a position pointed by a Queue Tail pointer (QT).

In [3], 32-bit Synthesizable QueueCore (QC-2) was proposed. The QC-2 is an optimized and improved architecture of the PQP and implements all Hardware features found in PQP core. It supports single precision Floating-Point Accelerator (FPA), and uses *QCaEXT* scheme which is a novel technique used to extend immediate values and memory instruction offsets that were otherwise not representable because of bit-width constraints.

Moreover, the QC-2 core implements new instructions for controlling the QREG unit

A prototype implementation is produced by synthesizing the high-level model for a target FPGA device

Several works have been done on the QC-2 Processor since its development up to date.

2.1 OVERVIEW OF RELATED WORK ON QUEUE CORE PROCESSOR

The present day Queue Core Processor has achieved the goal of reducing the performance crises in the microprocessor industry by virtue of the Queue Computing infrastructure it exhibits.

QC-2 has received tremendous development over the past years including the development of the Queue Compiler, Queue Assembler, efficient algorithms targeted for improvement.

2.1.1 INSTRUCTION SET ARCHITECTURE OF THE QUEUE CORE PROCESSOR

The QC-2 Processor implements a Produced Order Instruction Set Architecture. Here, each instruction can encode at most 2 operands that specify the location in the Register File where to read the operands. The Processor determines the physical location of operands by adding the offset reference in the instruction to the current QH pointer.

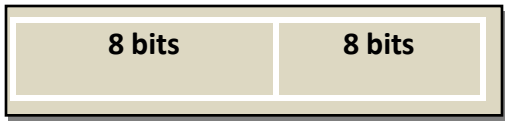
In [16], the Queue Core Instruction Set Architecture was presented. All instructions of QC-2 Processor are 16-bit wide, allowing simple fetch and decode stages and facilitating pipelining of the Processor. In the case of insufficient bit to express large constants, memory offsets or offset references, the Queue Core Processor implements the QCaEXT which inserts a ***"covop"*** instruction. This special instruction is used to extend load and store instructions offsets and also extends immediate values when necessary. This consequently curbs the bid- width constraints in the Queue Core Processor.

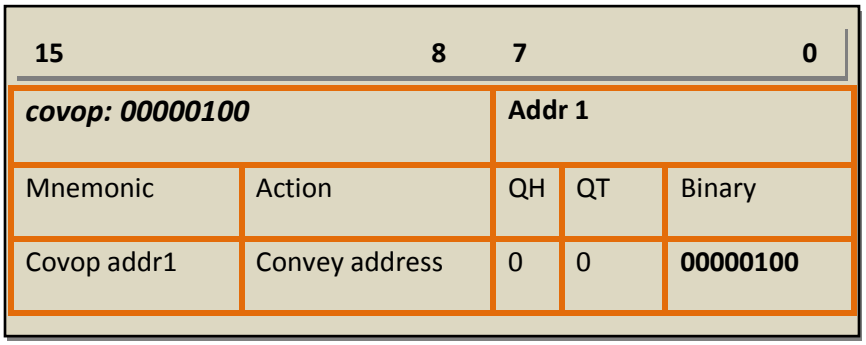
QC-2 Instruction Set Architecture uses mnemonics to denote operations. Each operation has a unique operation code (opcode) which specifies the exact operation needed to be performed by that instruction (the opcode ***add, sub*** indicates addition and subtraction operations respectively)

The first six (6) or eight (8) bits of any QC-2 instruction specifies the opcode whiles the rest of the bits denote instructions to be executed.

Below are the instruction formats for "COVOP" instruction and "LOAD / STORE" instructions.

COVOP INSTRUCTION





15 8	7 0			
covop: 00000100		**Addr 1**		
Mnemonic	Action	QH	QT	Binary
Covop addr1	Convey address	0	0	**00000100**

FIG 7: COVOP INSTRUCTION FORMAT

This instruction is used to convey an 8-bit address to a load/store instruction to extend the addressing bits from 8 to 16 bit

LOAD/ STORE INSTRUCTION

6 -bits	2 -bits	8-bits

LOAD INSTRUCTION: ***Ld k: k=byte, string, word (k could be unsigned or signed)***

15 10	9 8	7 0		
Load :opcode	**d**	**addr 0**		
Mnemonic	Action	QH	QT	Binary
Ld*k addr0(d)*	QTw ← m((d)+addr1.addr0)	0	1	***opcode***

FIG 8: LOAD/ STORE INSTRUCTION FORMAT

The above instruction load ***K*** (byte, string or word-signed/unsigned) to the operand queue pointed by the QT from memory address ((d)+addr0)

Details of the QC-2 instruction set architecture are made available in Appendix I.

2.1.2 QC-2 SPECIAL PURPOSE REGISTERS

Queue Core defines a set of special purpose registers available to the programmer to be used as the frame pointer register ($fp), stack pointer register ($sp), and return address register ($ra).

Frame pointer register serves as base register to access local variables, incoming parameters, and saved registers. Stack pointer register is used as the base address for outgoing parameters to other functions.

2.1.3 SYNTHESIS OF THE QUEUE CORE PROCESSOR (QC-2)

To make the QC-2 design easy to debug, modify, and adapt, a high-level description was used in the design process, which was also used by other system designers.

The QC-2 core was developed in Verilog HDL. After synthesizing the HDL code, the designed Processor has characteristics that enable investigation of the actual Hardware performance and functional correctness. It also gives the possibility to study the effect of coding style and Instruction Set Architectures over various optimizations. For the QC-2 Processor to be useful for these purposes, the following requirements were identified:

- High-level description: the format of the QC-2 description should be easy to understand and modify;
- Modular: to add or remove new instructions, only the relevant parts should have to be modified. A monolithic design would make experiments difficult;
- The Processor description should be synthesizable to derive actual implementations. The QC-2 has been designed with a distributed controller to facilitate debugging and future adaptation for specific application requirements since it was targeted for embedded applications.

2.1.4 QUEUE CORE COMPILER DESIGN

A lot of efforts were made towards the development of an efficient compiler for the QC-2 after the implementation of the Queue Core Processor was studied thoroughly. With regards to this, several compiling techniques were investigated for the Queue machine by Abderazek, Canedo and Sowa. The Queue Compiler was however not developed at that time. All previous attempts to develop the Queue Compiler were focused on using a retargetable code generator for register machines to map and rearrange low level intermediate register code into Queue code. This approach led to complex algorithms and unacceptable quality of the generated queue code.

A GCC based Queue compiler was then developed to evaluate the Queue programs in [4]. The Queue Compiler generates assembly code from C programs. This compiler is, for the best of knowledge, the first automated code generation tool designed for the Queue Computation Model. It has the higher capability of exposing natural Instruction Level Parallelism (ILP) from the input programs to the Queue Core Processor.

The GCC's 4.0.2 C front-end is used to build an Abstract Syntax Tree (AST) from the C program. The AST is transformed to GIMPLE representation (allows operations to have at most 2 operands) by the GCC's middle-end. The GIMPLE tree is reconstructed to form QTrees. The back-end which is the developed Queue Compiler takes the QTrees as input to produce Level Directed Acyclic Graphs (LDAGs).

The 2-offset P-code generation algorithm [9] in the code generator component of the Queue Compiler takes LDAGs and produce a low level linear intermediate representation called QIR. The assembly code is then generated from the QIR instructions.

The Queue Core Compiler was developed for the Queue Computation Model with a suitable data structure. It produces a good code with no artifacts. The infrastructure it provides is clean and easy to maintain. Its ability to compile large amount of programs garnishes its interesting features.

The uniqueness of the developed compiler is from the 1-offset code generation algorithm [10] implemented as the first and second phases in the back-end. This algorithm transforms the data flow graph to assure that the program can be executed using a one-offset queue instruction set. The algorithm then statically determines the offset values for all instructions by measuring the

distance of QH relative position with respect of each instruction. Each offset value is computed once and remains the same until the final assembly code is generated

Among other crucial roles played by the developed Queue Compiler encompasses constraining all instructions to have at most one offset reference, computation of offset reference values, Queue register allocation and Scheduling of the Queue program expressions in Level-Order manner.

The figure below shows the structure of the Queue Compiler

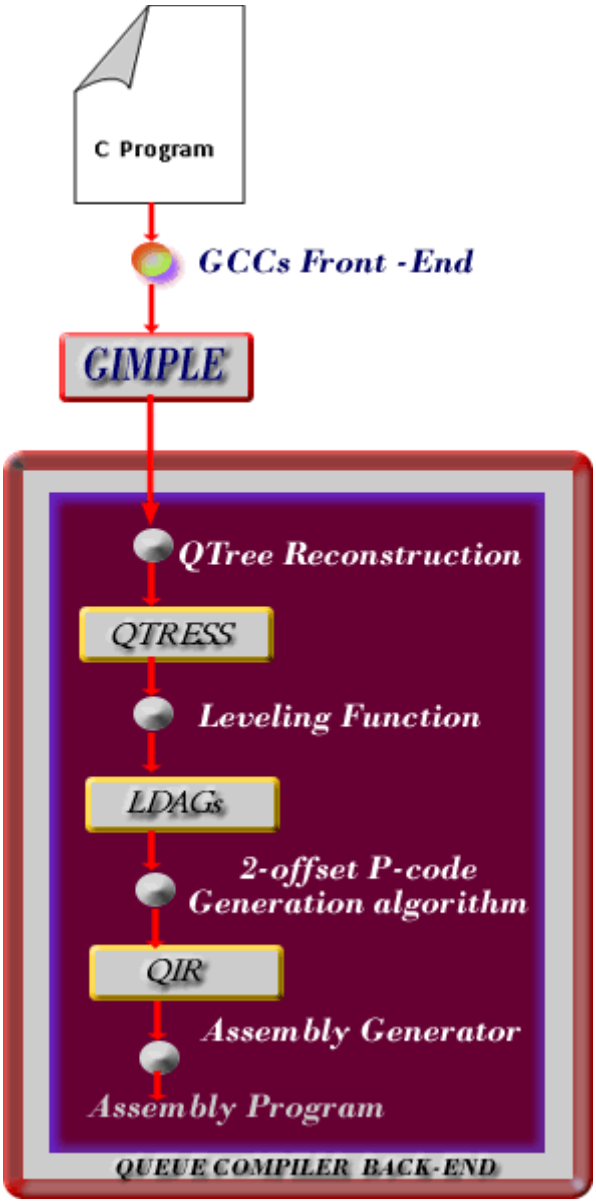


FIG 9: QUEUE COMPILER INFRASTRUCTURE

2.1.5 QUEUE CORE ASSEMBLER

The utmost need to map the assembly program produced by the Queue Compiler to the equivalent machine code for the Queue Processor to execute motivated the development of Queue Core Assembler (Qasm) [13]. Qasm supports two computing models: Queue Core (QC-2) model and Dual Execution Processor (DEP) model;

It supports the output of the Queue Compiler with the help of a preprocessor.

The figure below represents the structure of the Qasm.

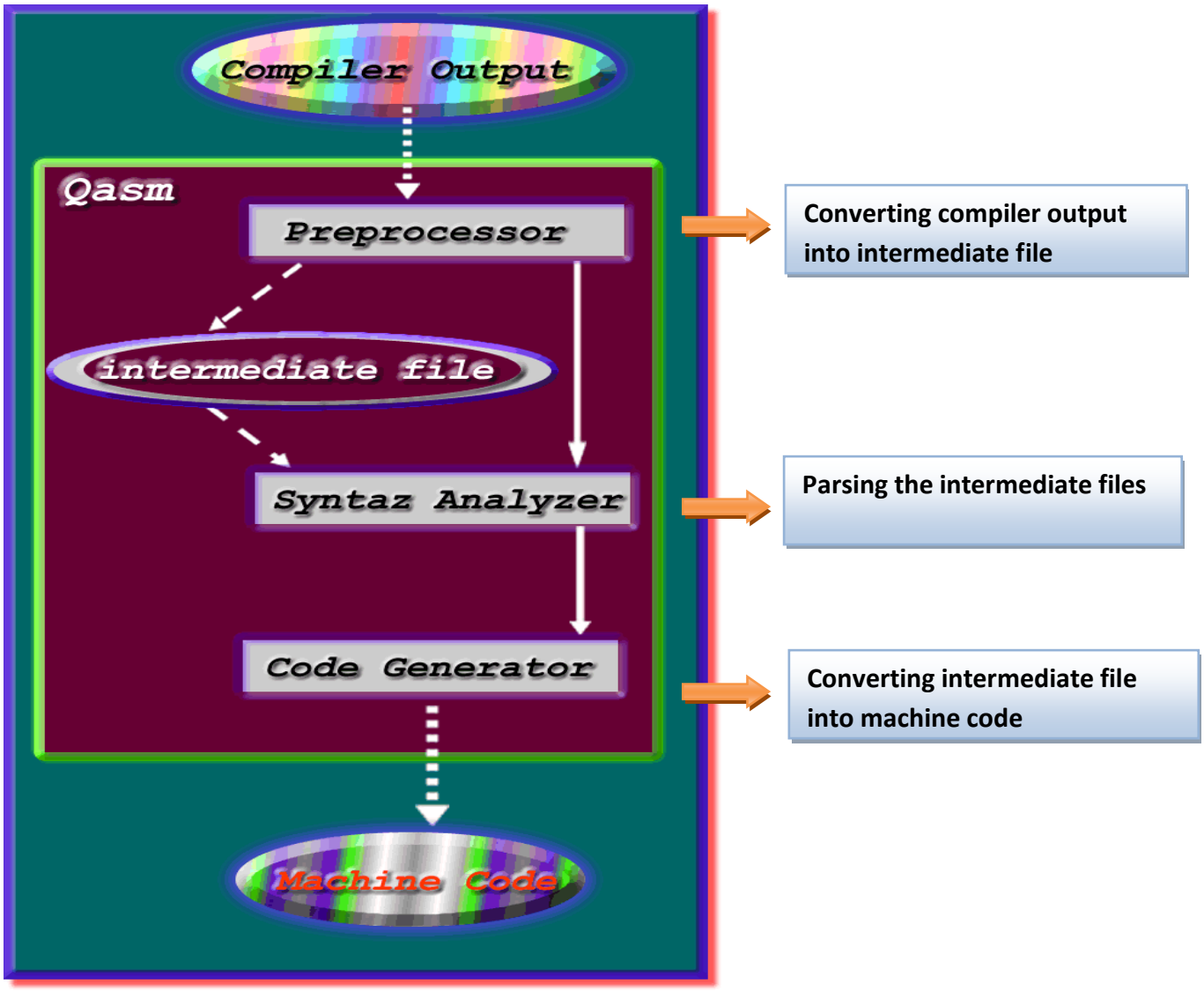


FIG 10: QUEUE ASSEMBLER INFRASTRUCTURE

2.1.6 CURRENT STATE OF THE ART OF THE QUEUE CORE PROCESSOR

The Queue Core Processor has gone through series of dramatic transformations from the time the idea of Queue Processor was conceived and implemented up to date. The transformation in the Queue Processor traverses improvements in its system architecture, development of the Queue Core Compiler, and the Queue Core Assembler.

The Queue Compiler has been successfully developed for the Queue Core Processor. It accepts any program written in C and generates a corresponding Assembly language program for the QC-2 Assembler.

The Queue Core Assembler was developed in order to translate the Assembly language of the Queue Processor into the equivalent machine language.

Simulator? Several simulators exist for different microprocessors. These simulators are built to emulate the microprocessors in diverse respects. Simulators are generally architecture specific in the sense that a simulator design for a particular architecture may not simulate a microprocessor with different architecture.
Several simulators have been built over the years for different microprocessors such as the RISC and CISC architectures with special reference to SPIM for the RISC architecture.

Unfortunately, the existing simulators cannot simulate the Queue Core Processor. This is primarily due to the differences in their architectures and Instruction sets.

CHAPTER THREE (3)

3.0 SIMULATOR DESIGN INFRASTRUCTURE

This research falls in the cross-road of two major disciplines; System Architecture and Software Engineering. By virtue of this, the research technique adopted will cut across both domains.

The simulator design infrastructure will primarily base on the Queue Core Processor architecture. The system architecture of the Queue Core Processor was studied in details to ascertain the functional units and their design infrastructure.

It is worth mentioning that other existing simulators [46,47] were extensively studied to gain in-depth knowledge, features and functionalities which were incorporated in our simulator (QSIM).

3.1. INFRASTRUCTURE OF THE INSTRUCTION PIPELINE STAGES

The six instruction pipeline stages of the QC-2 are shown in the diagram below:

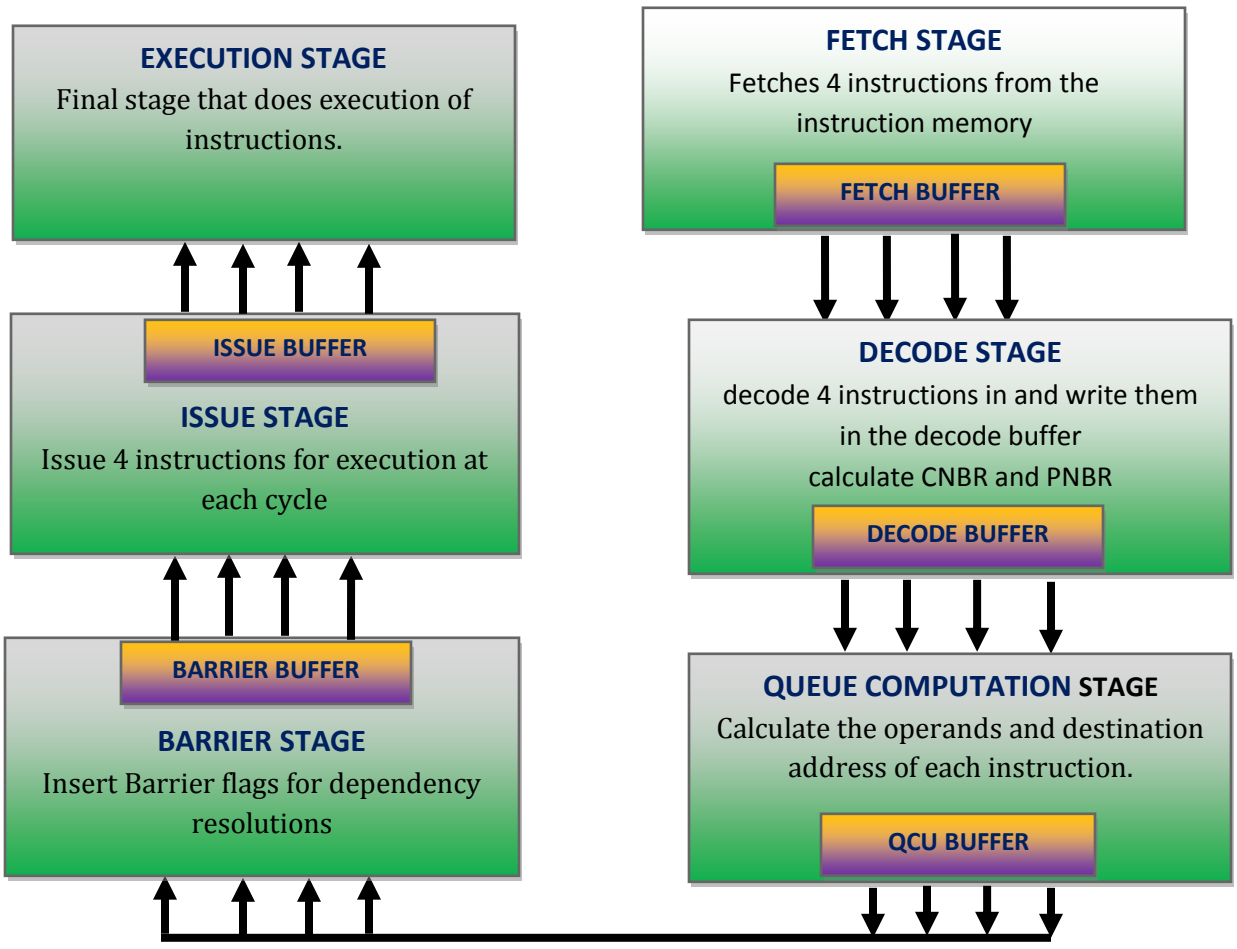


FIG 11: INSTRUCTION PROCESSING STAGES INFRASTRUCTURE FOR THE

FIG 12: FETCH PIPELINE STAGE FLOWCHART

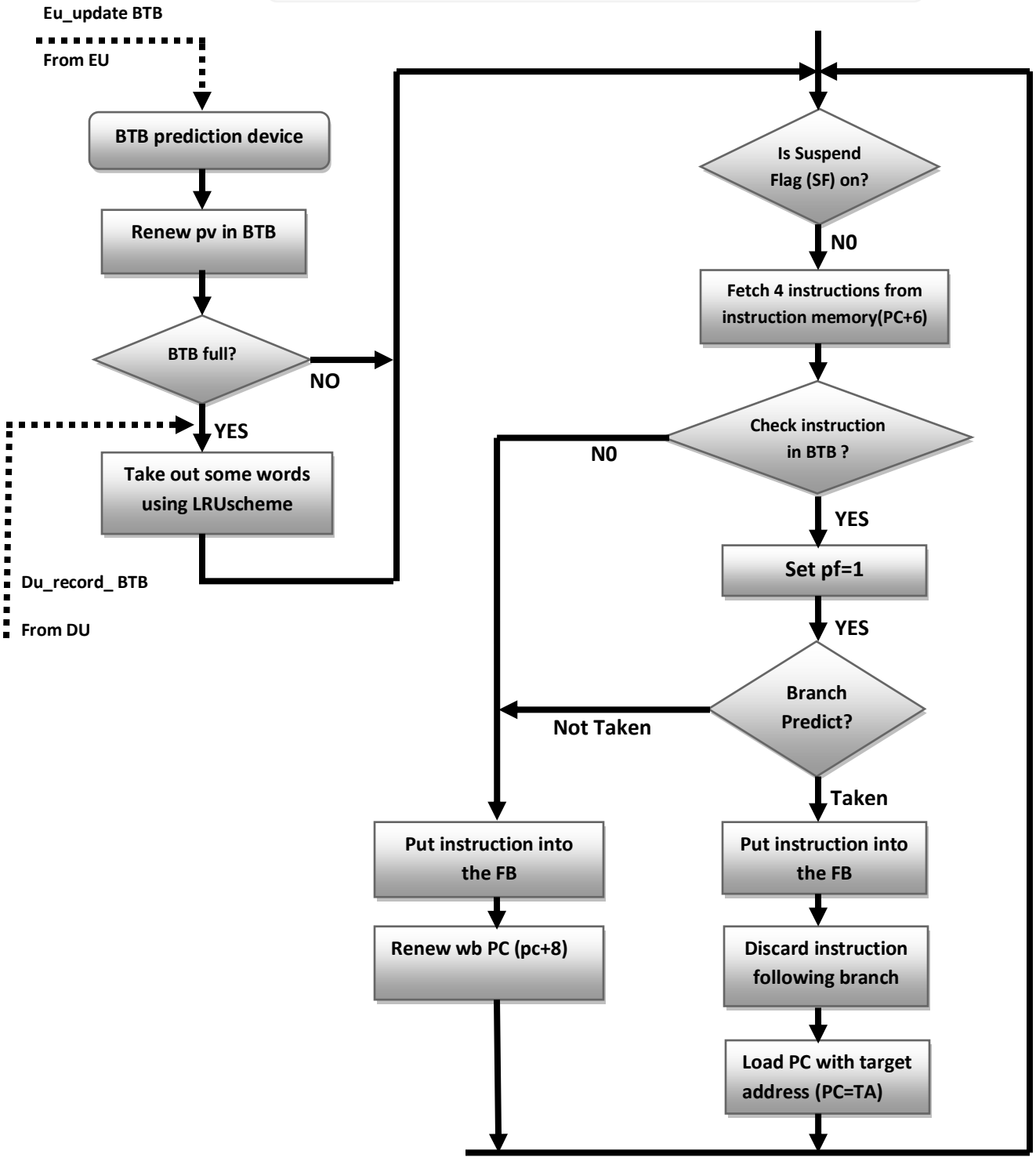
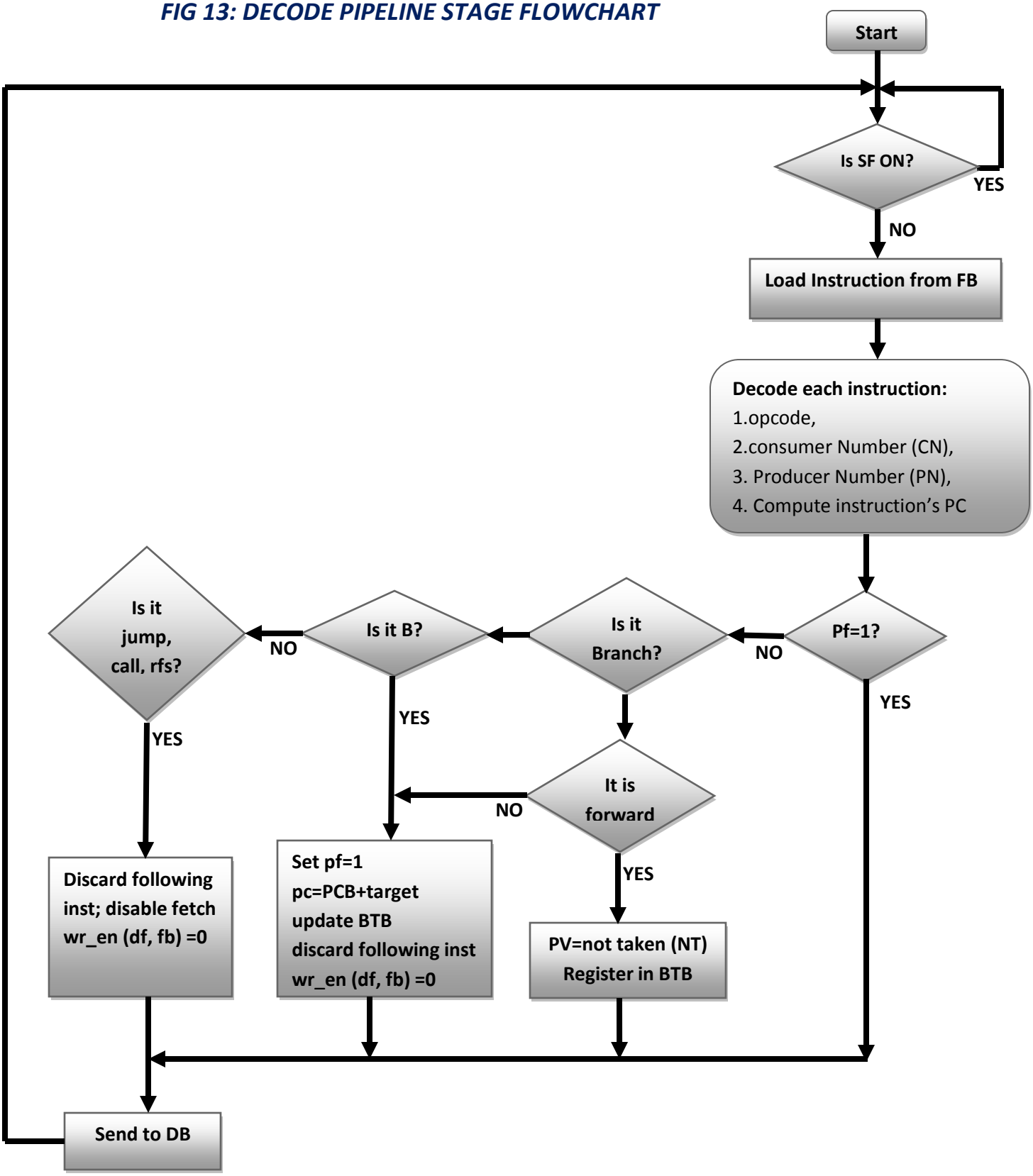


FIG 13: DECODE PIPELINE STAGE FLOWCHART



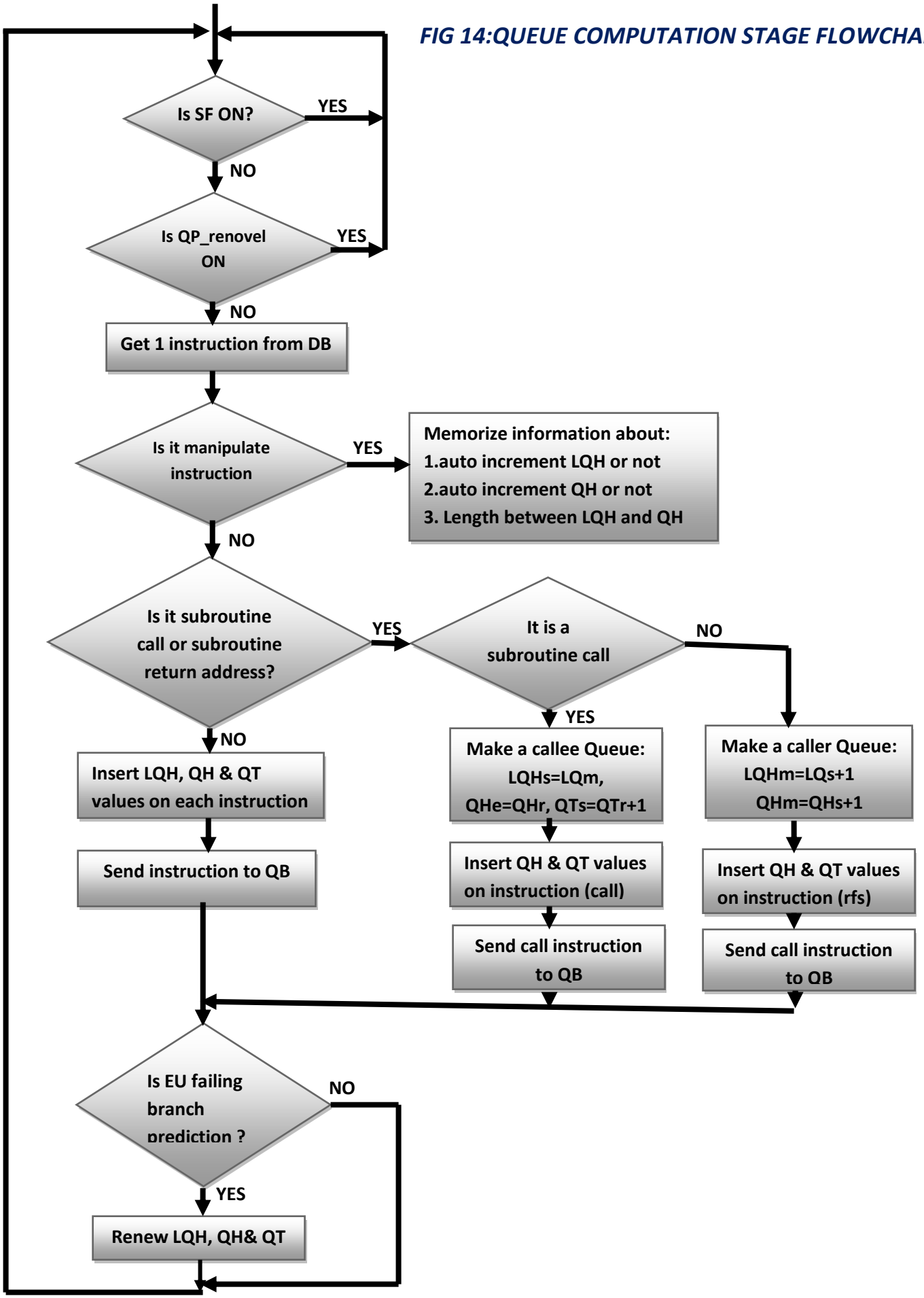


FIG 14:QUEUE COMPUTATION STAGE FLOWCHART

FIG 15: BARRIER AND QUEUE STAGE FLOWCHART

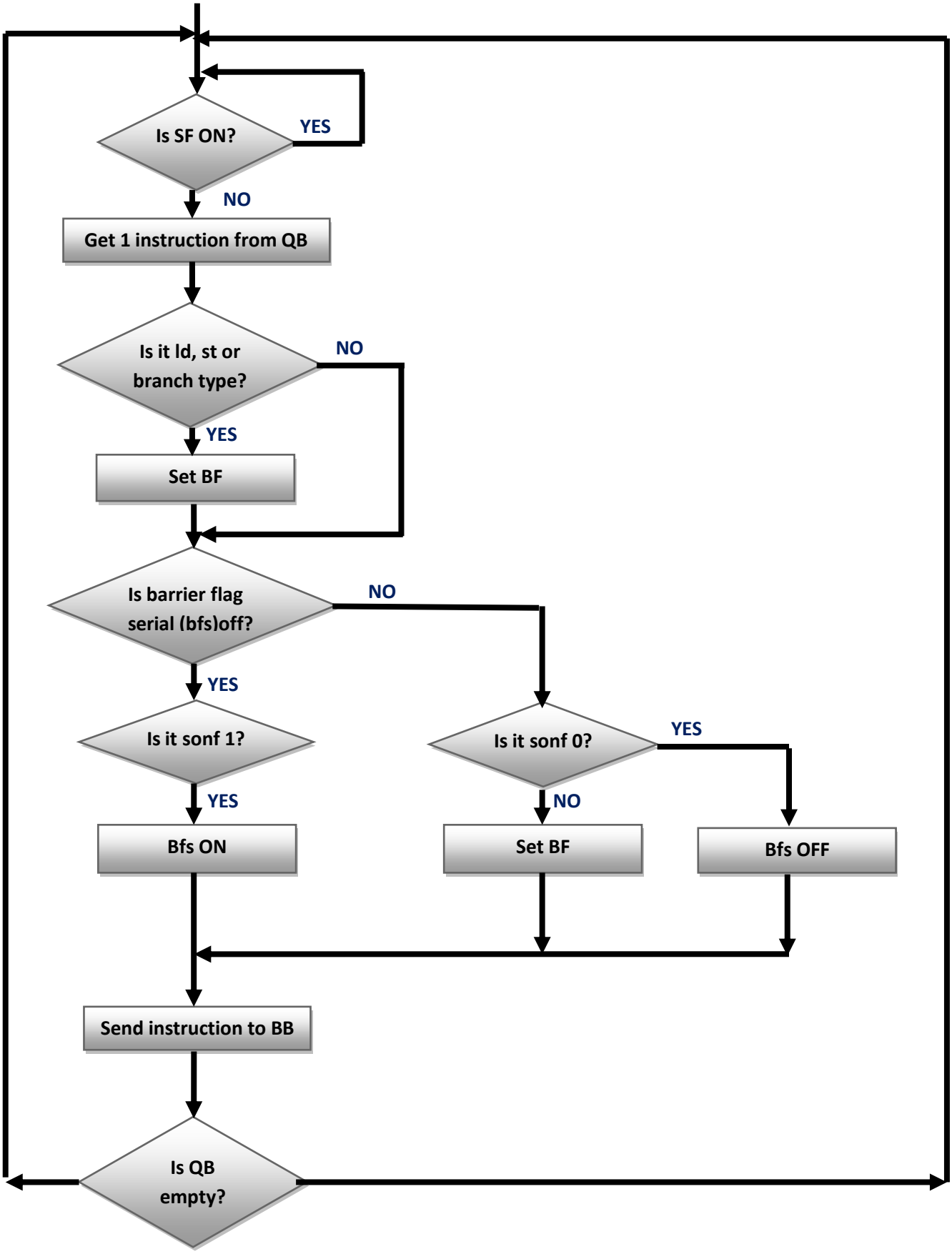


FIG 16: ISSUE PIPELINE STAGE FLOWCHART

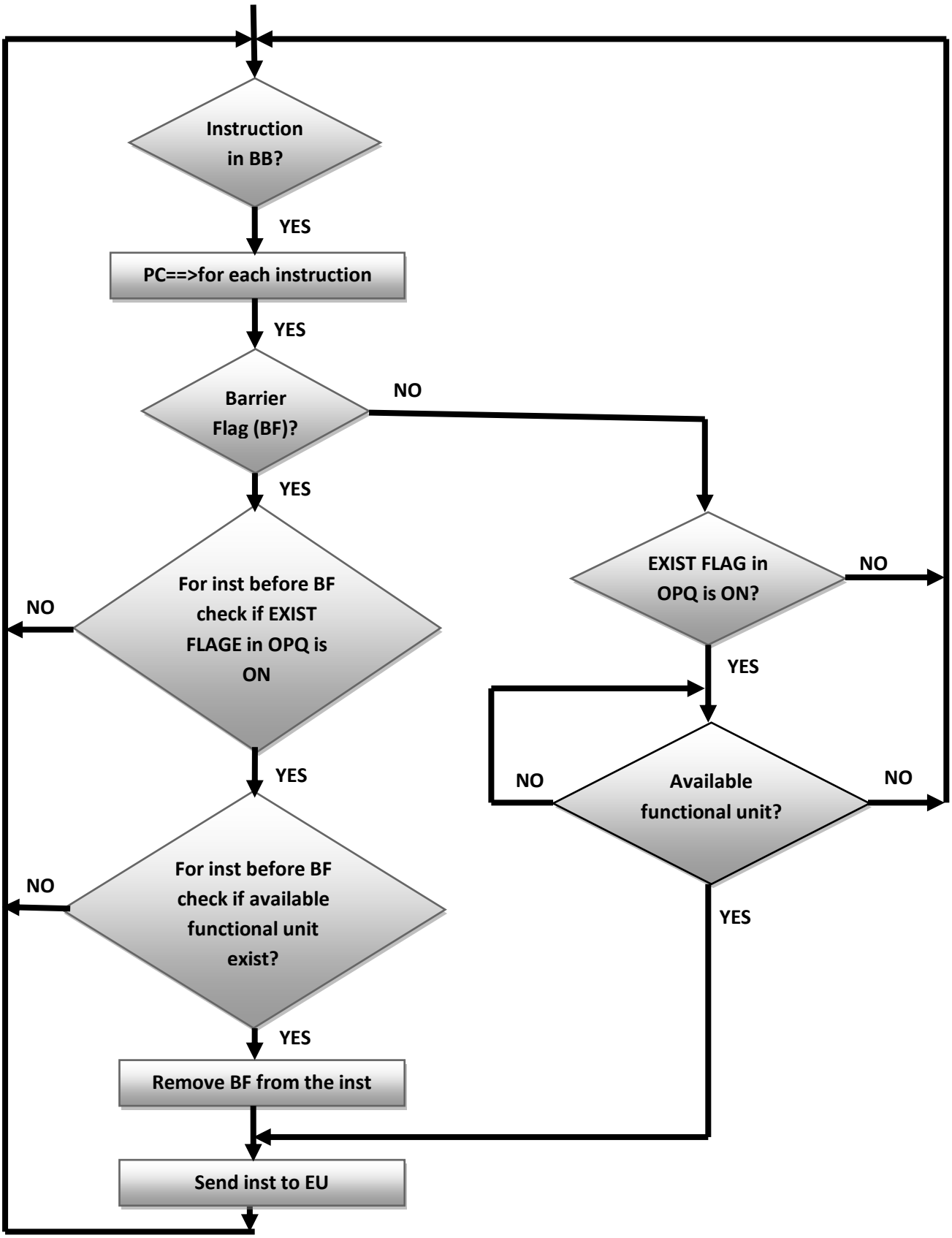
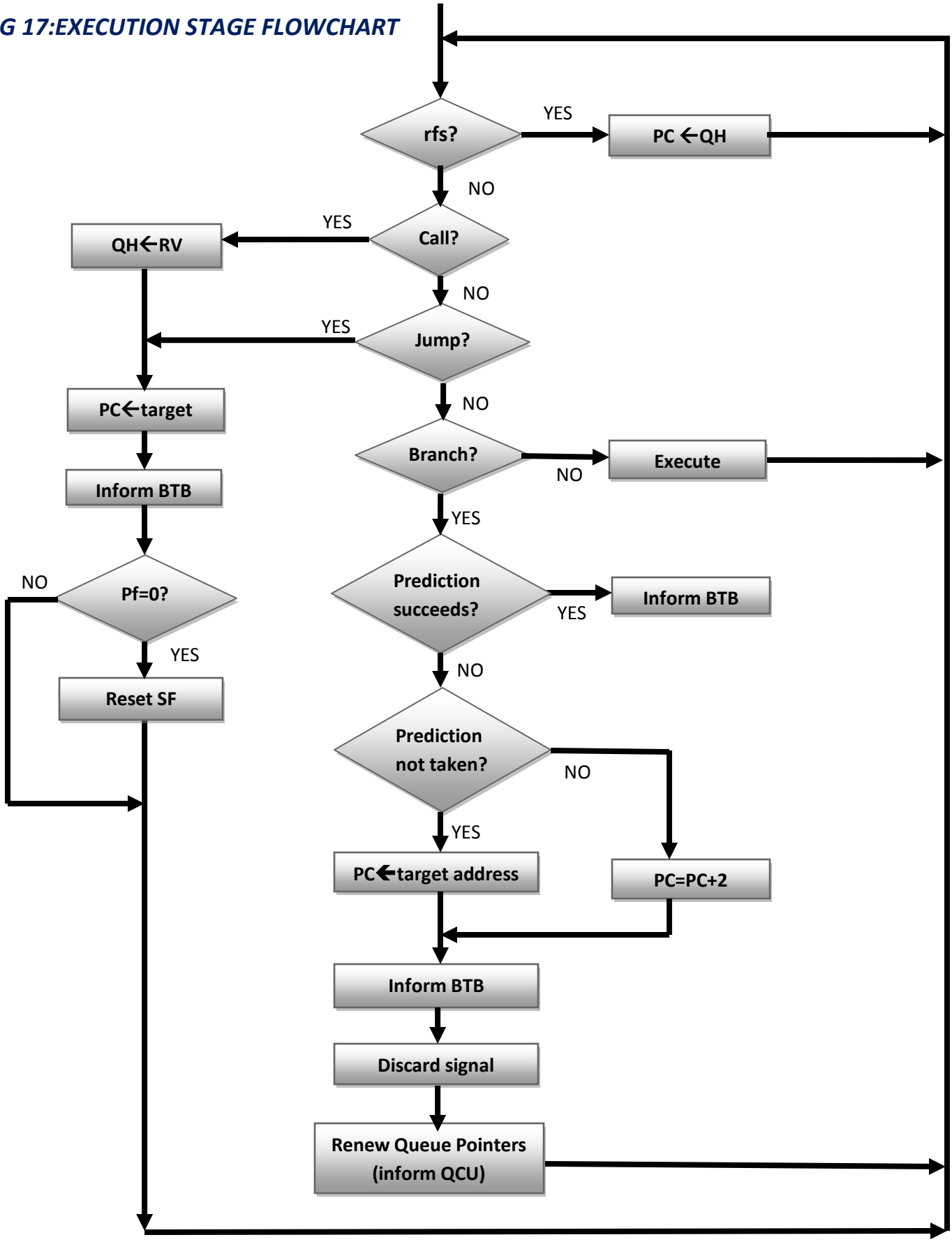


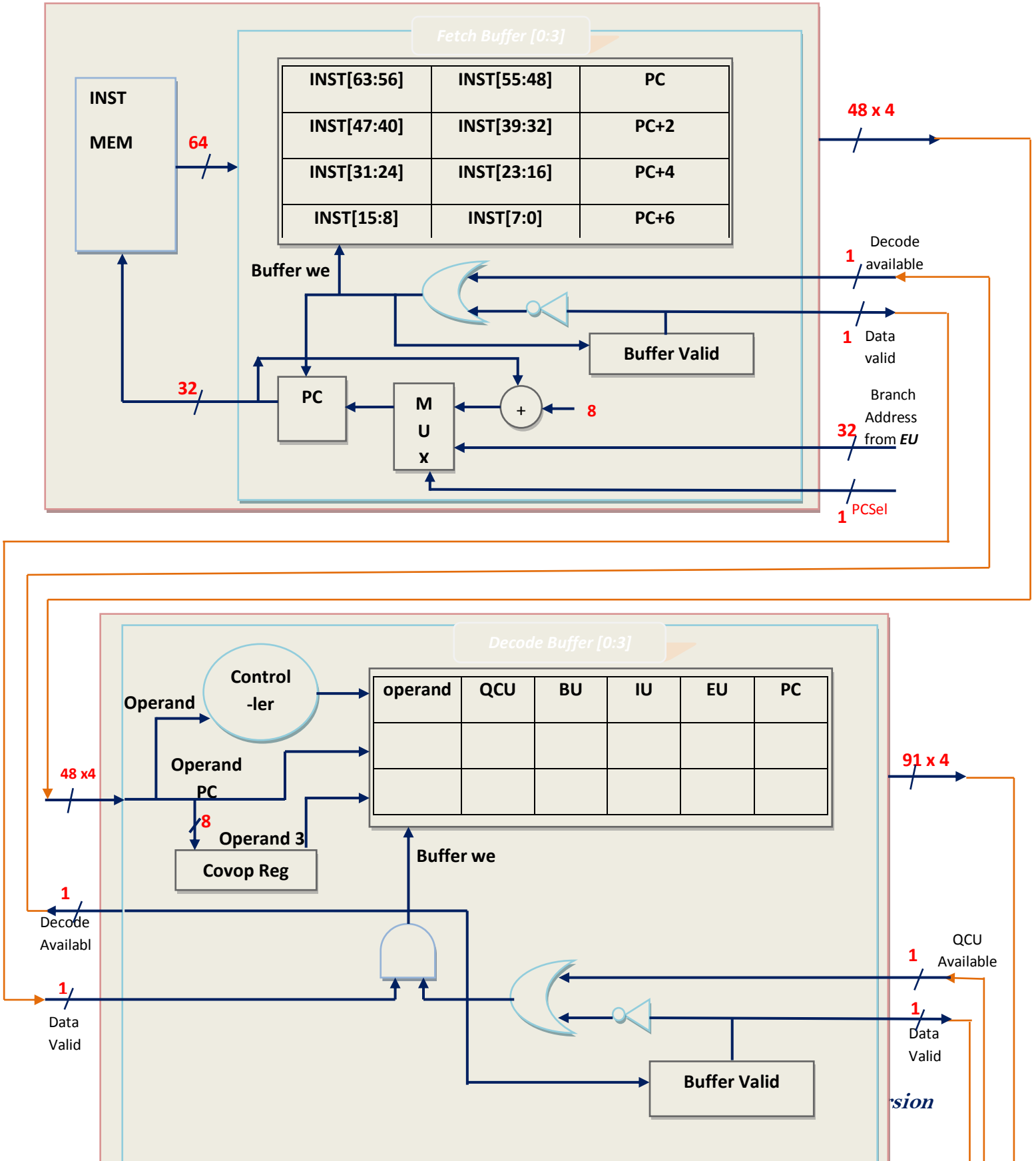
FIG 17:EXECUTION STAGE FLOWCHART

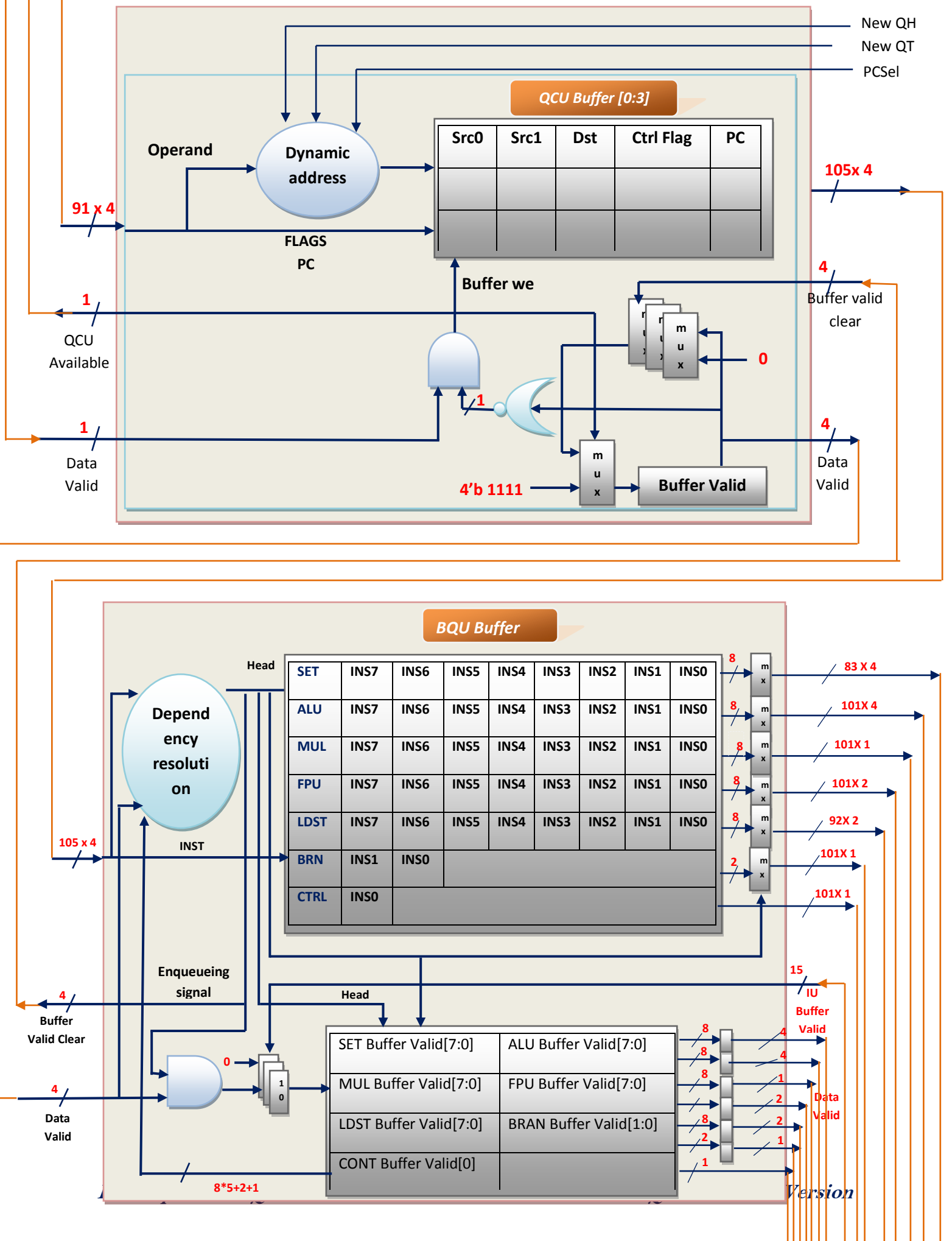


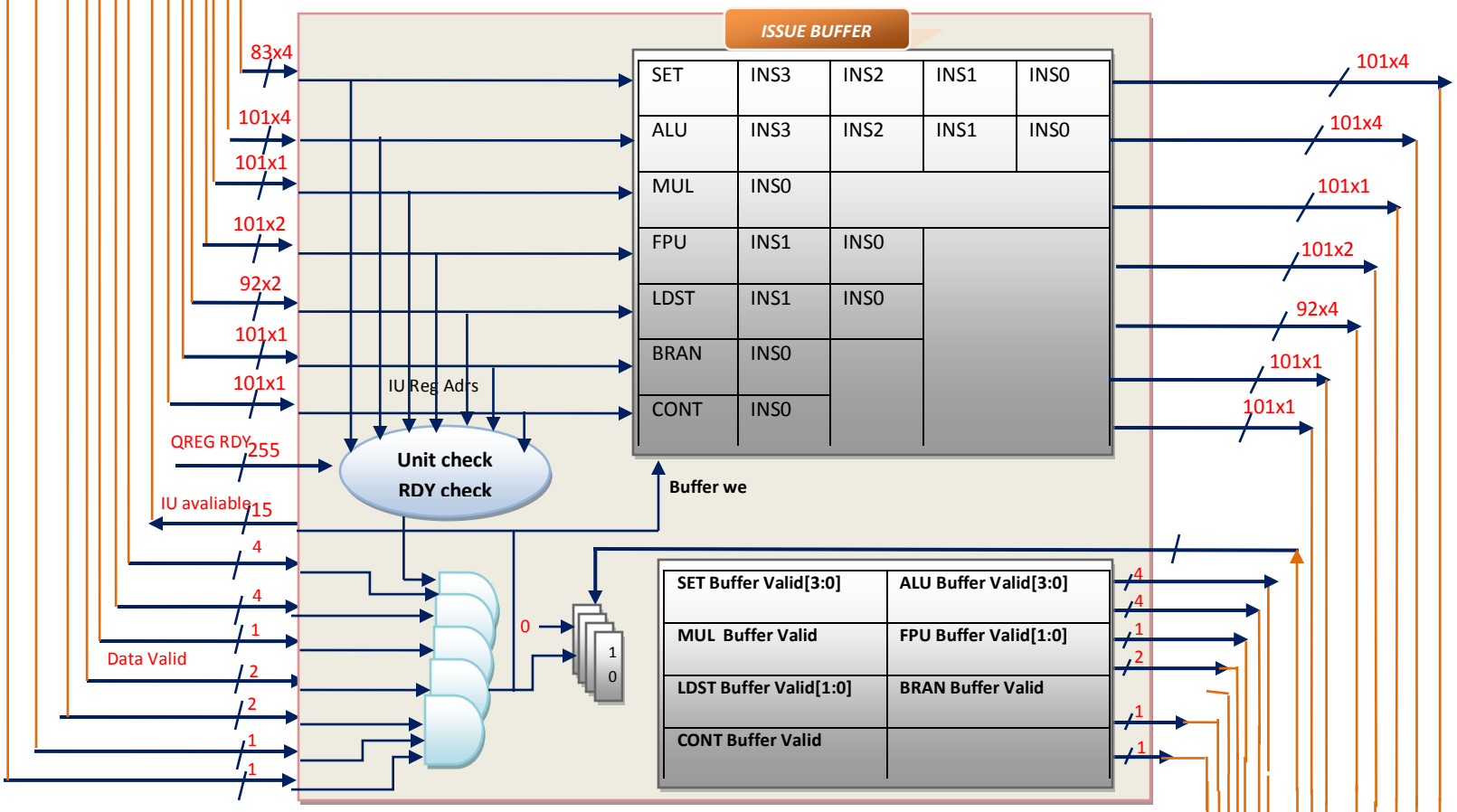
3.2 DATA PATH INFRASTRUCTURE OF THE QSIM

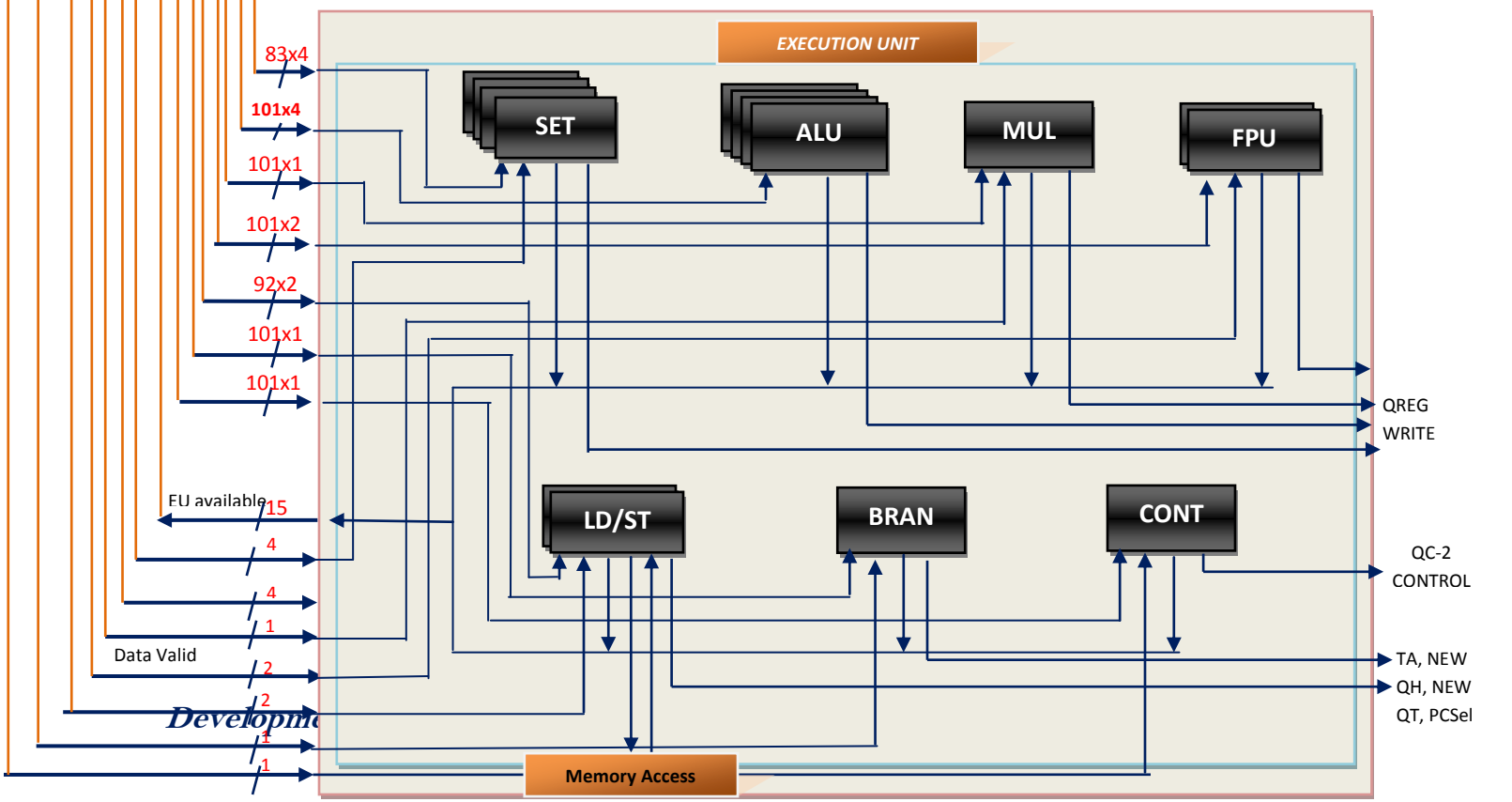
QSIM implemented a section of the data path of the QC-2 Processor. This is shown in the diagram below;

FIG 18: OC-2 DATAPATH




New QH
New QT
PCSel
QCU Buffer [0:3]
Src0
Src1
Dst
Ctrl Flag
PC
Operand
Dynamic address
91 x 4
FLAGS
PC
105x 4
Buffer we
4
Buffer valid clear
1
QCU Available
0
1
1
Data Valid
4
Data Valid
4'b 1111
Buffer Valid
BQU Buffer
Head
SET INS7 INS6 INS5 INS4 INS3 INS2 INS1 INS0
ALU INS7 INS6 INS5 INS4 INS3 INS2 INS1 INS0
MUL INS7 INS6 INS5 INS4 INS3 INS2 INS1 INS0
FPU INS7 INS6 INS5 INS4 INS3 INS2 INS1 INS0
LDST INS7 INS6 INS5 INS4 INS3 INS2 INS1 INS0
BRN INS1 INS0
CTRL INS0
83 X 4
101X 4
101X 1
101X 2
92X 2
101X 1
101X 1
Dependency resolution
105 x 4
INST
Enqueueing signal
Head
15
IU Buffer Valid
4
Buffer Valid Clear
0
4
Data Valid
SET Buffer Valid[7:0]
ALU Buffer Valid[7:0]
MUL Buffer Valid[7:0]
FPU Buffer Valid[7:0]
LDST Buffer Valid[7:0]
BRAN Buffer Valid[1:0]
CONT Buffer Valid[0]
Data Valid
8*5+2+1

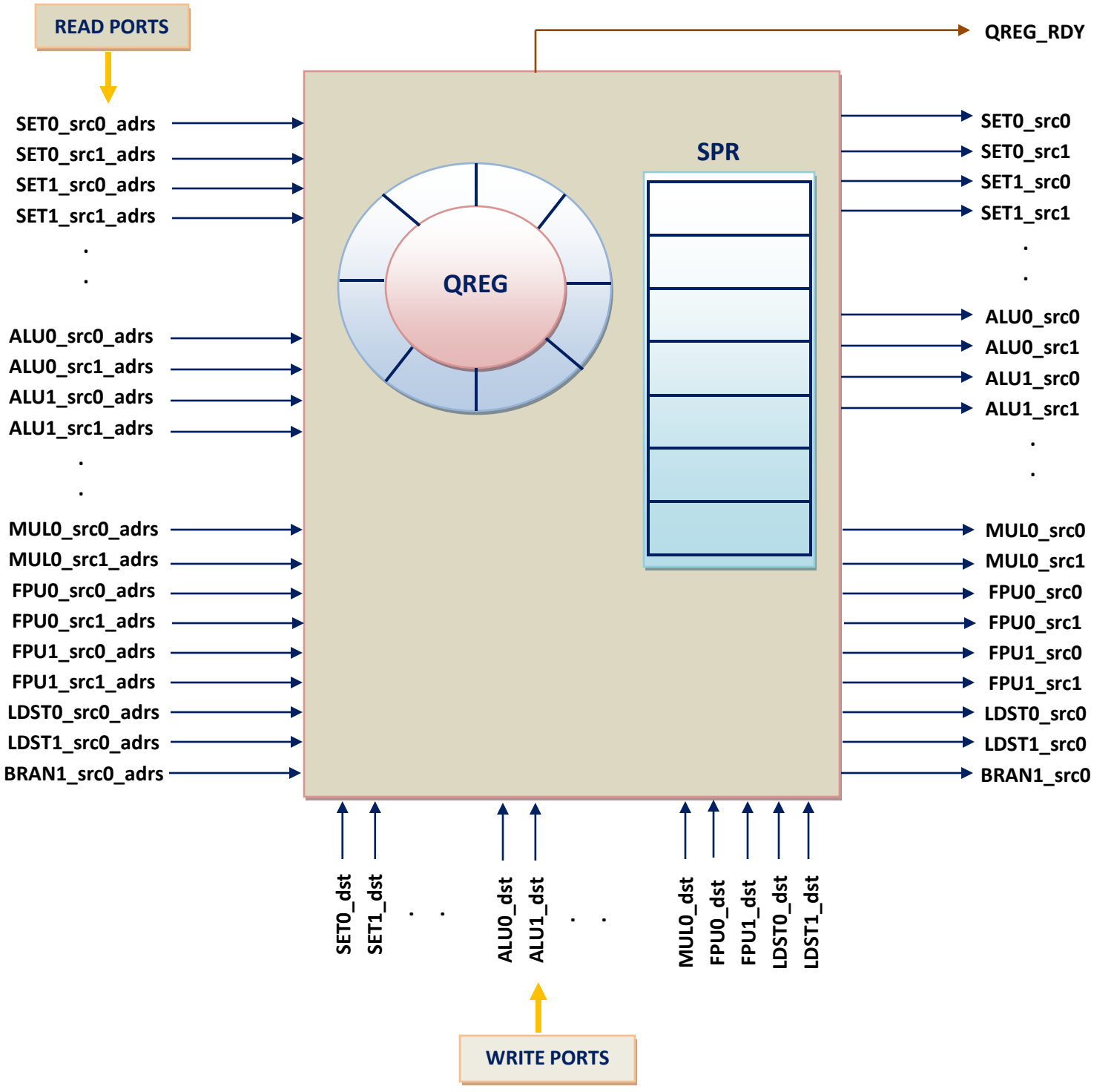

ISSUE BUFFER
SET INS3 INS2 INS1 INS0
ALU INS3 INS2 INS1 INS0
MUL INS0
FPU INS1 INS0
LDST INS1 INS0
BRAN INS0
CONT INS0
83x4
101x4
101x1
101x2
92x2
101x1
101x1
IU Reg Adrs
QREG RDY
255
Unit check
RDY check
Buffer we
IU avaliable
15
4
4
1
2
2
1
1
Data Valid
0
1
0
SET Buffer Valid[3:0]
ALU Buffer Valid[3:0]
MUL Buffer Valid
FPU Buffer Valid[1:0]
LDST Buffer Valid[1:0]
BRAN Buffer Valid
CONT Buffer Valid
101x4
101x4
101x1
101x2
92x4
101x1
101x1
4
4
1
2
1
1


EXECUTION UNIT
83x4
101x4
101x1
101x2
92x2
101x1
101x1
SET
ALU
MUL
FPU
QREG
WRITE
EU available
15
4
4
1
2
2
1
1
Data Valid
LD/ST
BRAN
CONT
QC-2
CONTROL
TA, NEW
QH, NEW
QT, PCSel
Memory Access

3.3 QUEUE REGISTER INFRASTRUCTURE OF THE QSIM

The design implementation of QSIM includes the Circular Queue Register. This is depicted in the diagram below:

FIG 19: QREGISTER INFRASTRUCTURE



3.4. SOFTWARE ENGINEERING INFRASTRUCTURE

Software Engineering principles were solicited in developing a conceptual model based on the understanding of the QC-2 System Architecture.

In the Software Engineering paradigm, there are two basic approaches in analyzing and designing a system: Traditional Approach and Object Oriented Approach. However, the desirable characteristics of the Object Oriented Approach made its choice in this project paramount over the Traditional Approach. The Object Oriented Approach uses the Unified Modeling Language (UML) in its design model.

Unified Modeling Language (UML) is a very dominant visual modeling language which provides a comprehensive notation for communicating the requirements, architecture, implementation, deployment, and states of a system. The Unified Modeling Language (UML) [43] has now become the de-facto industry standard for Object-Oriented (OO) Software development. UML provides a set of diagrams to model structural and behavioral aspects of an Object-Oriented system [44, 45].

The objective of UML is to provide a common vocabulary of Object Oriented terms and diagramming techniques that are rich enough to model any systems development project from analysis through implementation. There are nine (9) diagramming techniques in UML that can be used to model the existing and proposed system. These include Class, Package, Use Case, Sequence, Communication, State, Activity, Component and Deployment diagrams.

In this research, we employed only the UML Class and Package diagramming techniques in our modeling and design process.

3.4.1 UML CLASS DIAGRAM

A class diagram partitions the system into areas of responsibility or functions (classes). It is used to depict the static structure and view of the system to be designed and describes the attributes and behavior of the system.

The UML class diagram was used to model each functional unit of QSIM, indicating explicitly the relationships between the various functional units. Each of these functional units is modeled as a separate Class with attributes and methods representing operations performed by the unit.

In QSIM, we identified fifty-six (56) Classes that represent the entire functional units of our system.

However, due to the large number of Classes identified with our system, it was quite daunting managing these Classes. To provide a comprehensive and easy way to handle all these classes, we have therefore employed a very powerful UML constructs called Package diagram that allows a designer to organize Classes into groups called Packages indicating the dependencies between them.

3.4.1 UML PACKAGE DIAGRAM

In QSIM, UML Package diagrams were used to depict the high-level overview of the architecture and design. It was typically used to logically modularize our system by organizing related Classes into the same package with one namespace.

The UML Package diagram has been used to cluster our system into four (4) packages with each package containing several related Classes.

The detailed structure of our system via UML package diagram is shown below.
The broken lines with arrow head indicate the relationship between packages. In the diagram, the stereotype: << **access**>> was used to indicate that all the members of the imported package will be added to the namespace of the importing package whiles keeping the visibility private.

The arrow head points from the importing package to the imported package.
In our package diagrams, the contents of the packages were not shown for clarity reasons and also due to the large number Classes for some of the packages. However, details of each of the packages will be discussed further.

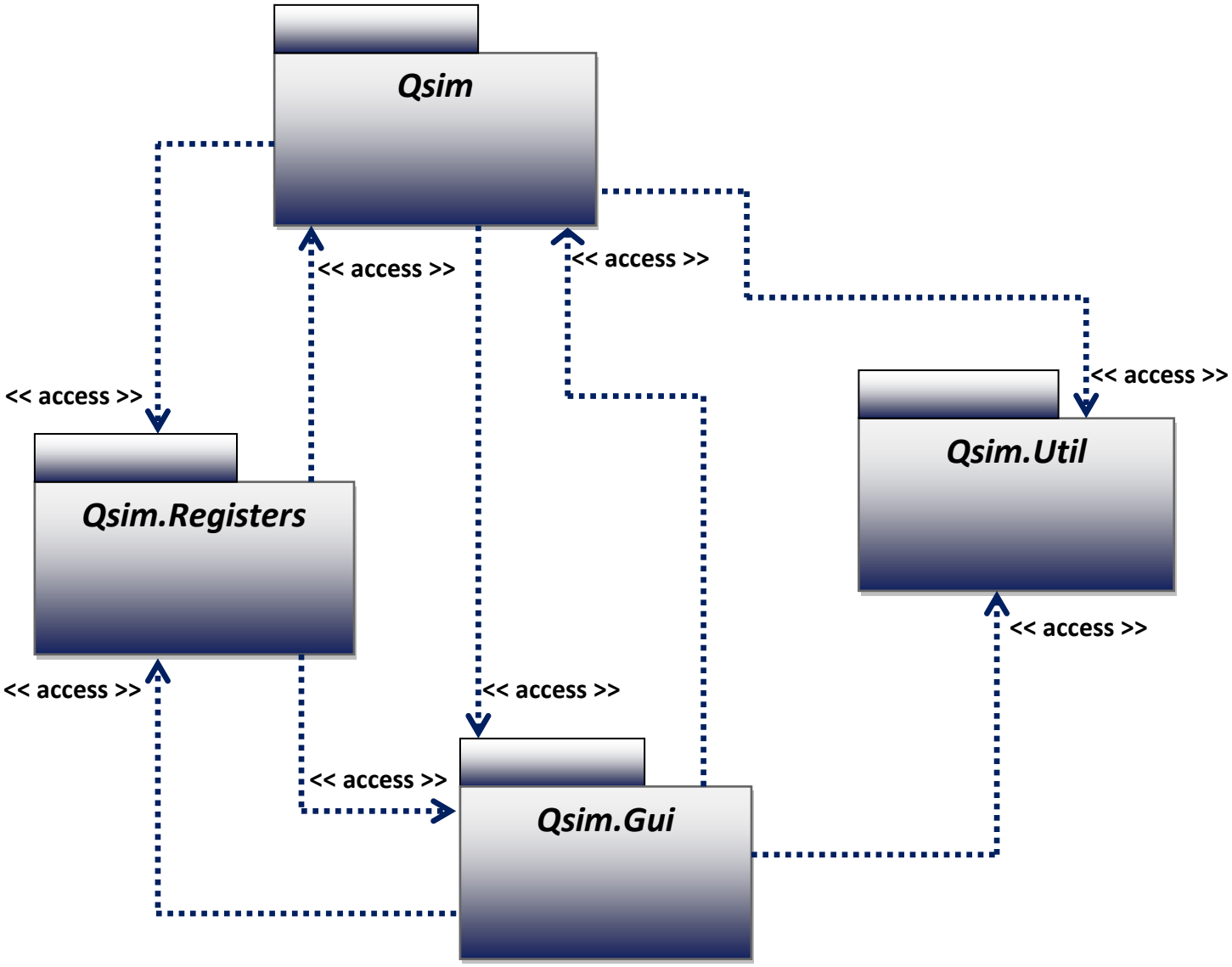


Fig: UML Package Diagrams for QSIM

3.4.3 ANALYSIS OF UML PACKAGE DIAGRAMS

Here, we give a detailed breakdown of each of the UML packages above by clearly outlining the main members of the package.

3.4.3.1 QSIM.REGISTER PACKAGE

This package consists of six (6) Classes. These Classes are depicted at the high abstraction level hence we have not indicated the attributes and methods of each Class. The plus (+) sign in the circle attached to the namespace is used to indicate that all the Classes connected by the plus sign are members of the package.

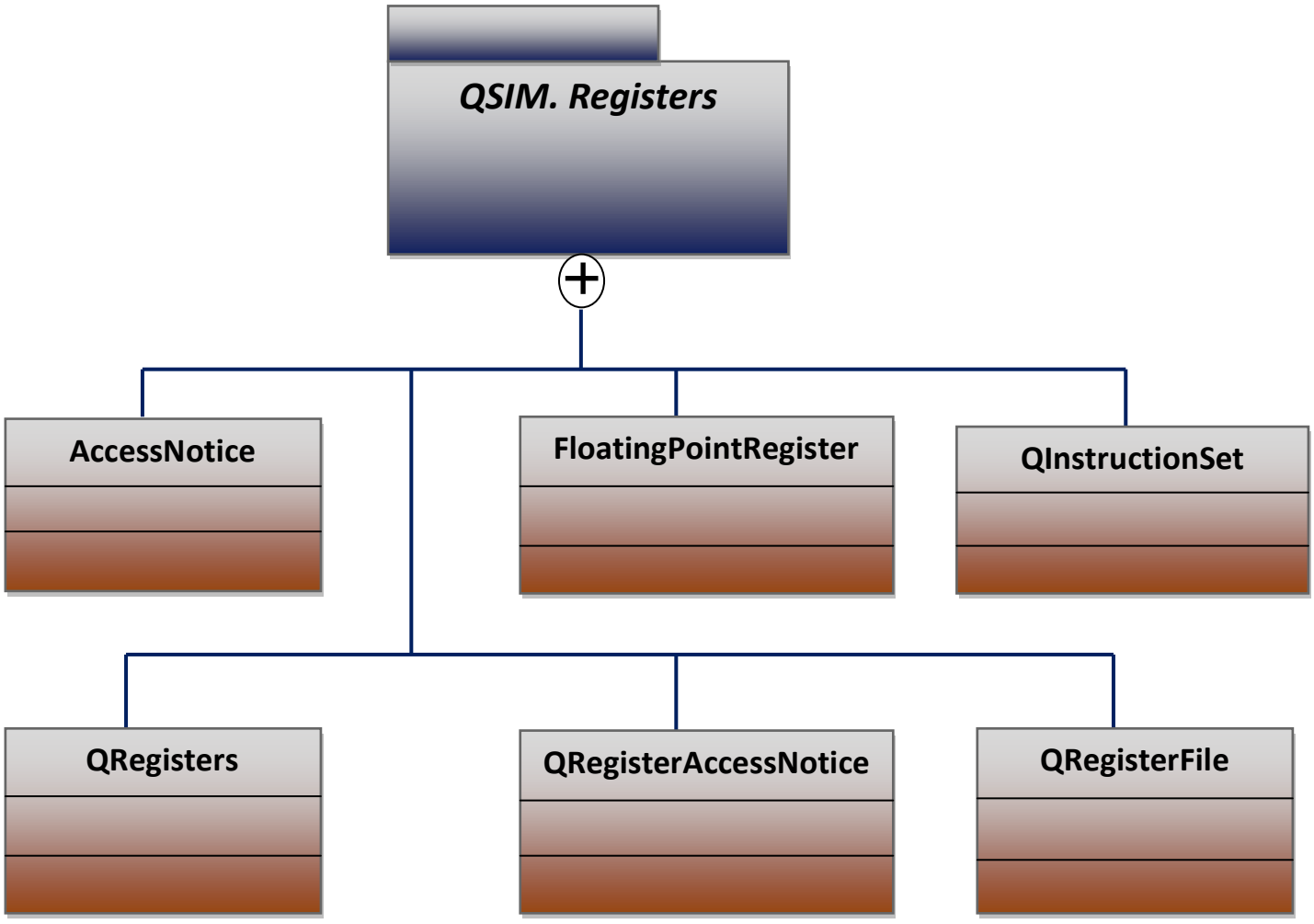


Fig 21: UML Class Diagrams for QSIM.REGISTERS PACKAGE

3.4.3.2 QSIM PACKAGE

This package also consists of seven (7) Classes and these Classes are depicted at the high abstraction level hence the attributes and methods of each Class are not shown. This package contains the Main Class which initializes the entire program execution.

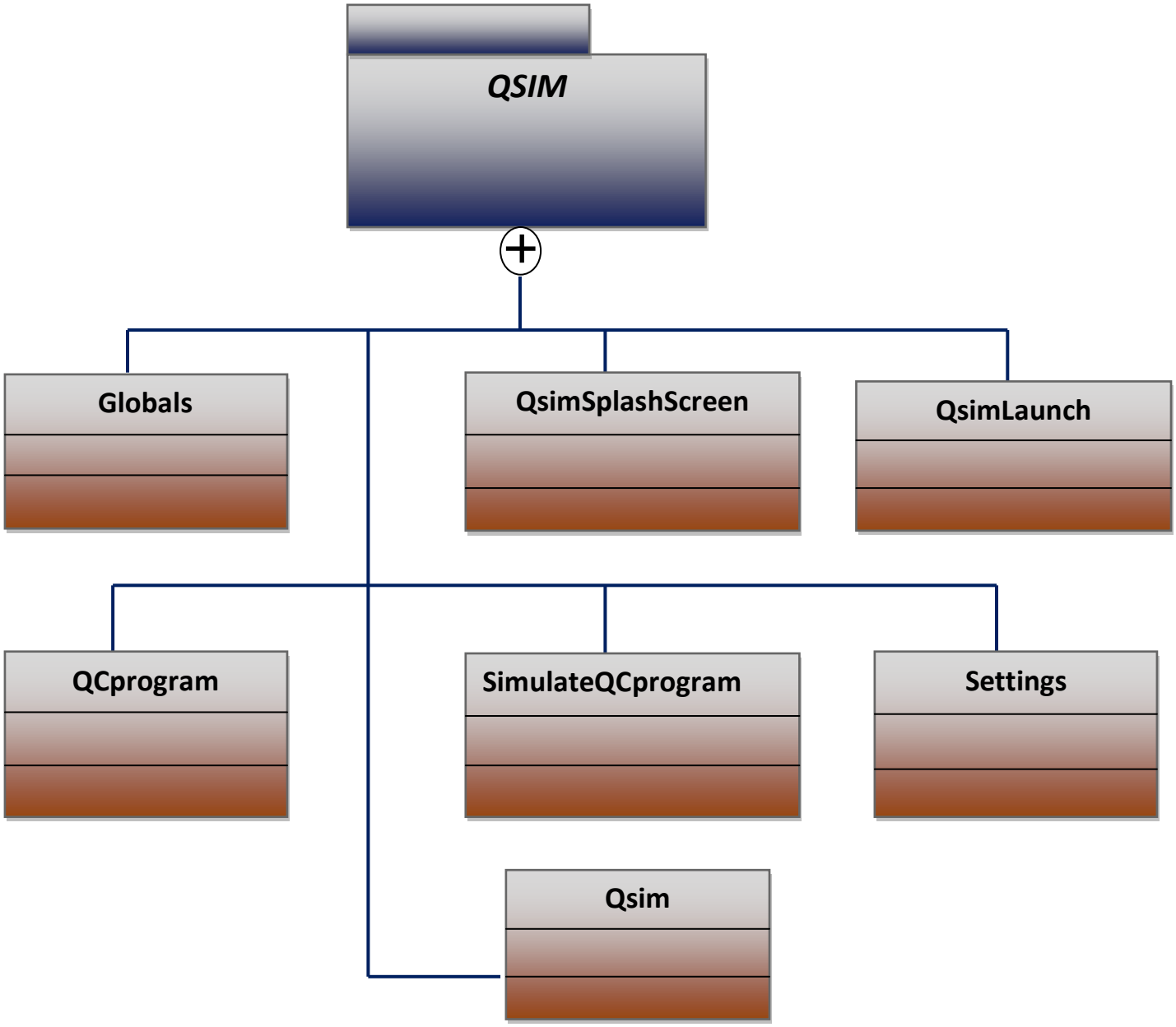


Fig22 : UML Class Diagrams for QSIM PACKAGE

3.4.3.3 QSIM .UTIL PACKAGE

This package contains three (3) Classes. As usual these Classes are depicted at the high abstraction level hence the attributes and methods of each Class are not shown.

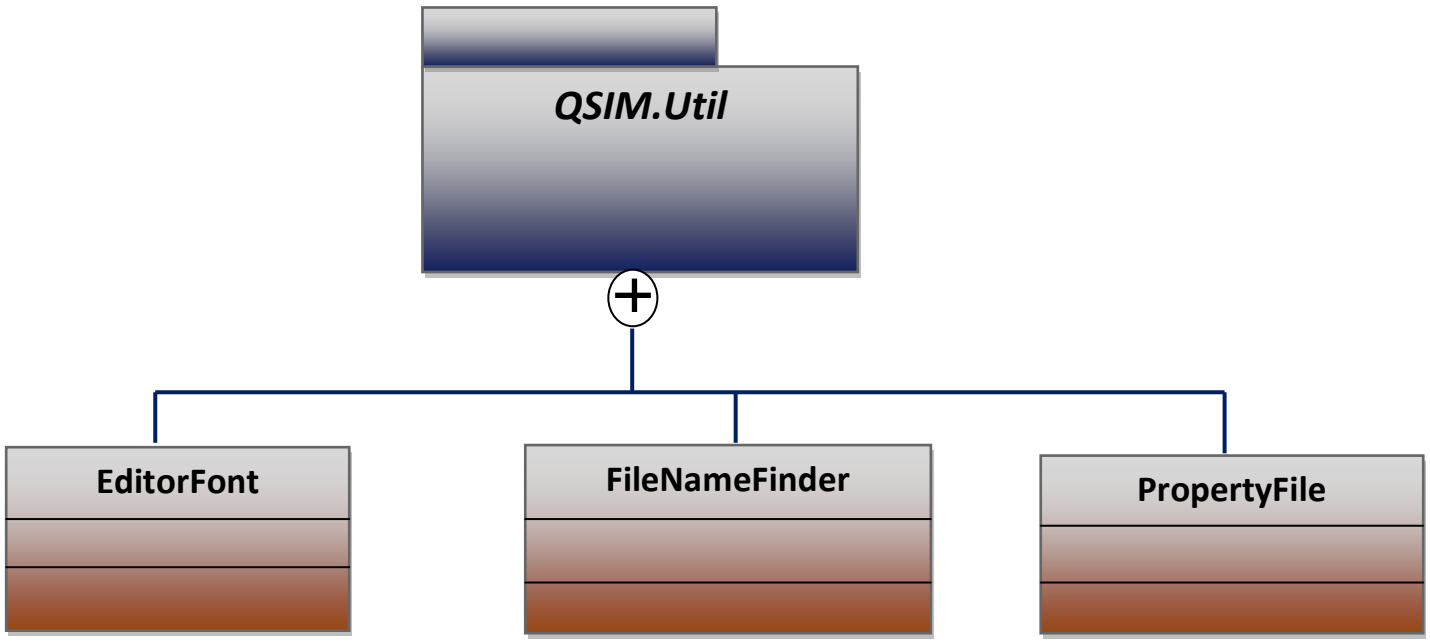


Fig 23: UML Class Diagrams for QSIM.Util PACKAGE

3.4.3.4 QSIM .GUI PACKAGE

This package consists of 40 Classes. However, these Classes have been sub-grouped into main Classes and Action Object Classes. The Action Object Classes provides action listener object to various components of the GUI including Menu and Tool items.

The Main Classes are the various components of the GUI. Ten (10) of such Classes were identified with our system. Due to the large number of the Action Classes, we have grouped them using the UML package diagram labeled ***ActionObejects***. This package has been merged with the GUI package using the << **merge**>> stereotype. This will allow the content of the two packages to be merged as if they were one holistic package with the same namespace.

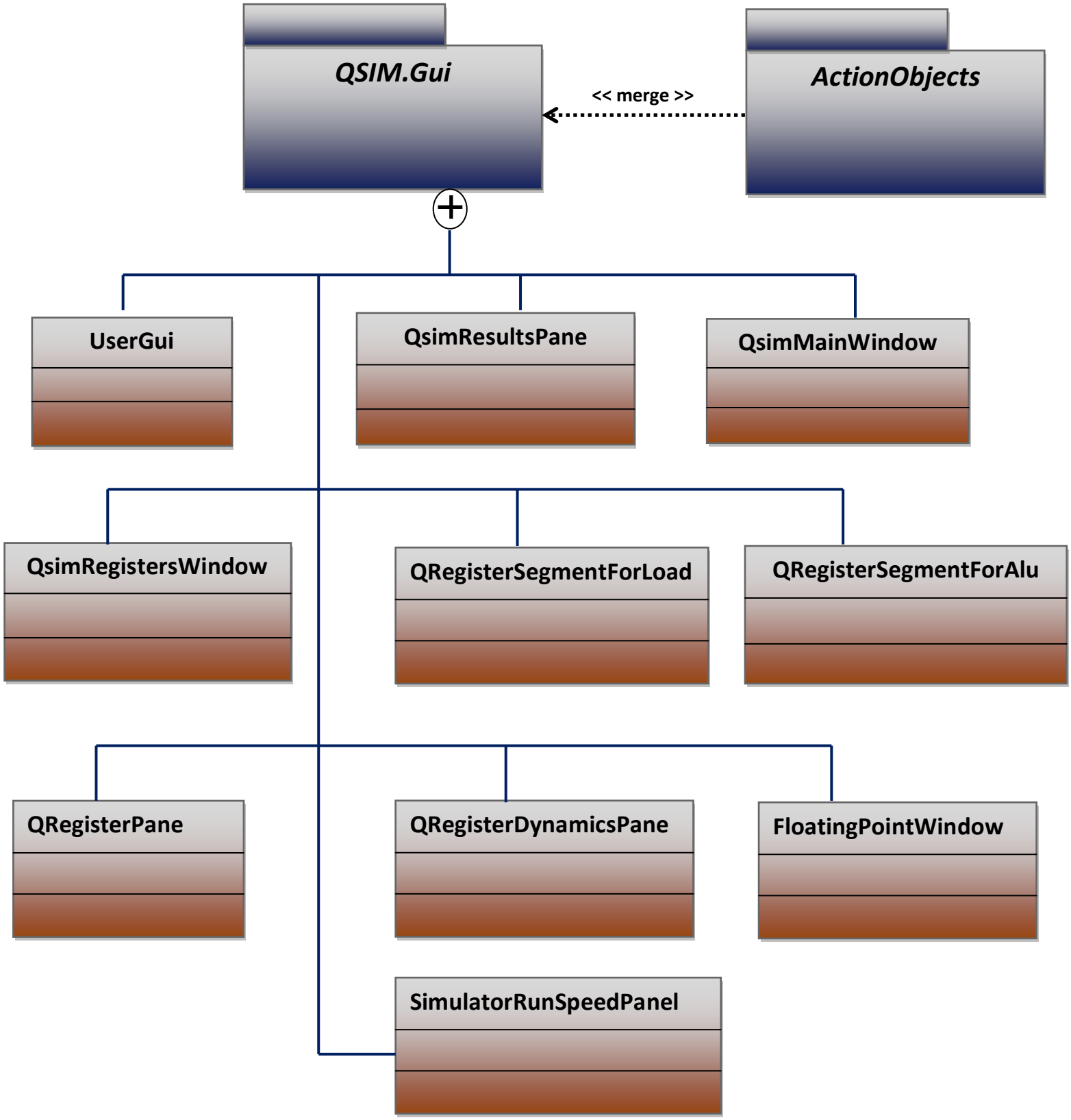


Fig 24 : UML Class Diagrams for QSIM.Gui PACKAGE

Based on the UML diagrams discussed above, a conceptual model in a form of prototype was design to depict the User Interface and with associated components. This is shown in the figure below:

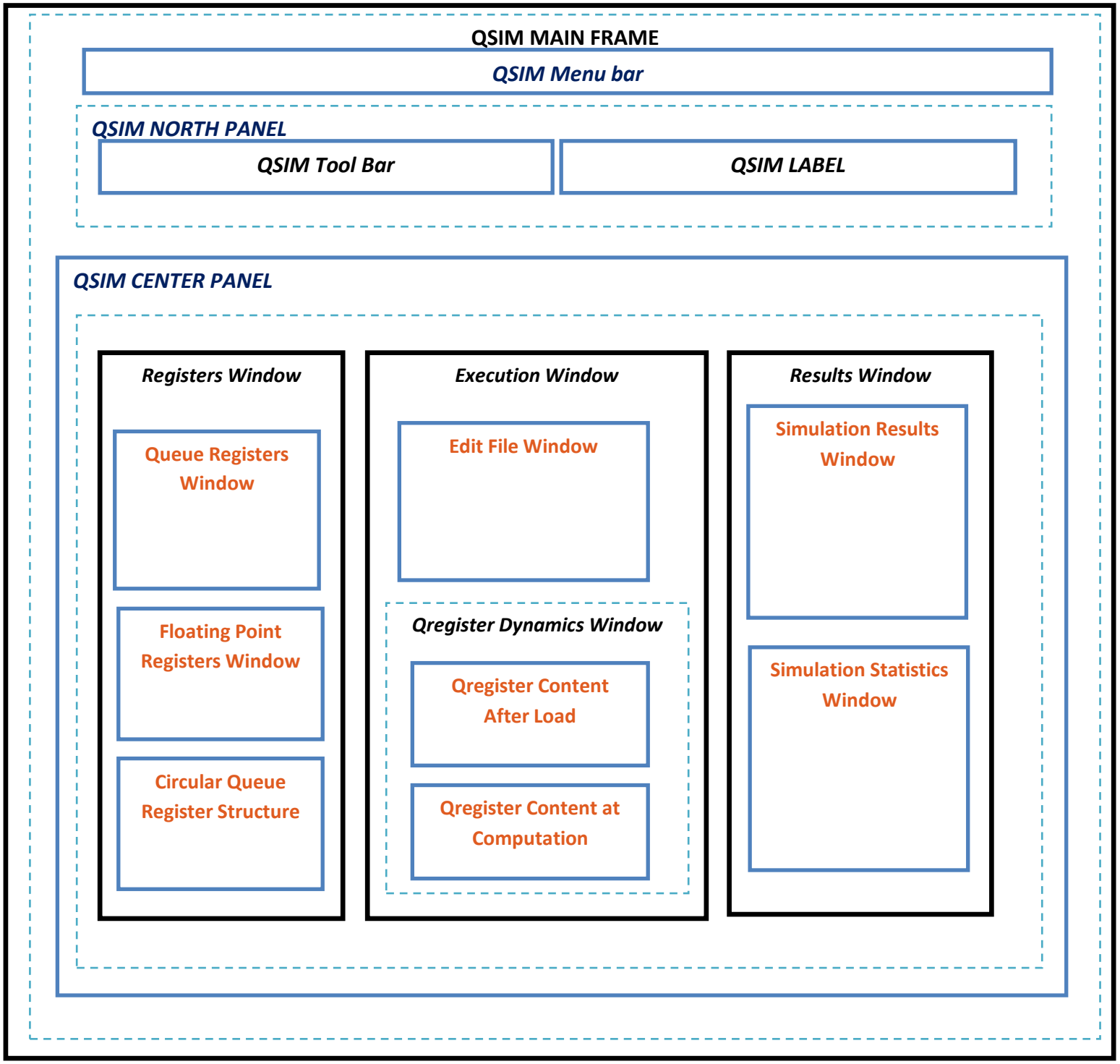


Fig 25: GUI Conceptual Design of QSIM

The above conceptual model was implemented using the Java programming language as a tool. To ensure the correctness of each functional unit, a unit testing was adopted to test each successfully designed functional unit. This was followed by debugging as needed.

In order to solicit a methodical and comprehensive approach in both the design and implementation of QSIM, we adopted the ***Software Life Cycle Model*** described below.

3.4.4 SOFTWARE LIFE CYCLE MODEL

Software Life-cycle model plays an indispensable role in any Software design process. The explicit models of Software evolution date back to the earliest projects developing large Software systems in the 1950's and 1960's. Since the 1960's, many descriptions of the classic Software life cycle have emerged including Hosier 1960, Royce 1970, Boehm 1976, Distaso 1980, Scacchi 1984, Somerville 1999 [26].

A Software / System Life-cycle model is a description of the significant phases or sequence of activities carried out in a Software Engineering project, and the relative order of these activities [24]. It provides a fixed generic framework that can be tailored to a specific project and specifies the relationships between project phases, including transition criteria, feedback mechanisms, milestones, baselines, reviews, and deliverables [27].

The apparent purpose of early Software Life-cycle model was to provide a conceptual scheme for rationally managing the development of Software systems [25]. The Software Life-cycle model provides a standardized format for planning, organizing and running a new development project [24].

Different Life-cycle models exist in the Software Engineering paradigm which can be adopted in a Software development process. Among the commonly used ones include Waterfall Model, V-Shaped Model, Spiral Model, Evolutionary Model, Rapid Prototyping, Unified Process (UP) [24,25].

In this research, the Iterative Waterfall Model, a modification of the Waterfall model introduced by Royce in 1970 [24] was adopted in that it is simple and easy to understand and implement. It is one of the most widely known and used in theory. The model is characterized by the interesting slogan which reinforces good habit: ***define-before-design and design-before-code.***

Each phase in this model has a specific deliverable and milestones. The iterative feature of the model makes the activity non-linear and allows backtracking from any of the phases when needed.

In [28], the Iterative Waterfall Model was described as depicted below;

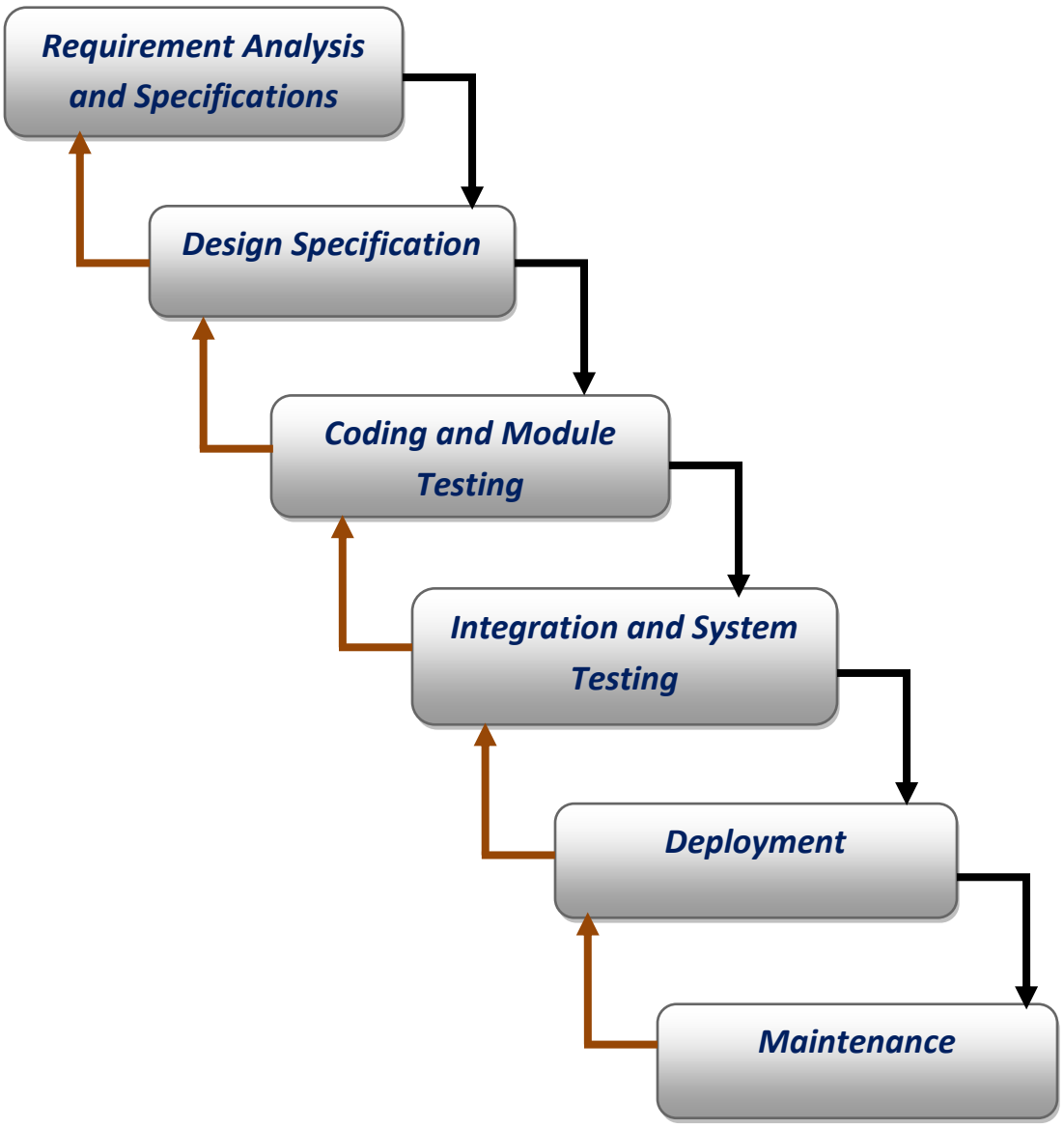


FIG 26: ITERATIVE WATERFALL MODEL

3.4.5 DEVELOPMENT ENVIRONMENT AND LANGUAGE INFRASTRUCTURE

In Software development, the choice of suitable language has immense impact on the resulting product. The Software industry has witnessed a myriad of programming languages over the past decades. The choice of a programming language for Software development is therefore an explicit decision of the developer.

In this research, the Java programming language was chosen for the development of the Queue Core Simulator. Java has gained enormous popularity since it first appeared. Its rapid ascension and wide acceptance can be traced to the powerful technology and platform it renders to programmers. Sun Microsystems identified ***"write once, run anywhere"*** slogan as the core value proposition of the Java platform [29]. This feature allows a programmer to write Software that can run on any platform (windows, Linux, Mac etc).

As stated in the Java language white paper by Sun Microsystems, "java is a simple, Object-Oriented, distributed, interpreted, robust, secure, architecture neutral, portable, multithreaded, and dynamic [30].

The choice of Java as a development language in this research was motivated by the well emulating features stated above.

In computing, Integrated Development Environment (IDE) is a Software application that provides comprehensive facilities and features to computer programmers for Software development [31].

The ***Eclipse*** IDE was used in the development of QSIM. The choice of this IDE stemmed from its simplicity in usage and multipurpose features it provide. It supports other languages aside Java. Eclipse employs plug-ins to provide different functionalities. The plug-in architecture supports writing any desired extension to the environment.

More importantly, the Eclipse SDK includes the Eclipse Java Development Tools (JDK), offering an IDE with a build-in Java compiler and a full model of the Java source file. This allows for advanced refactoring techniques and code analysis.

ECLIPSE INFRASTRUCTURE

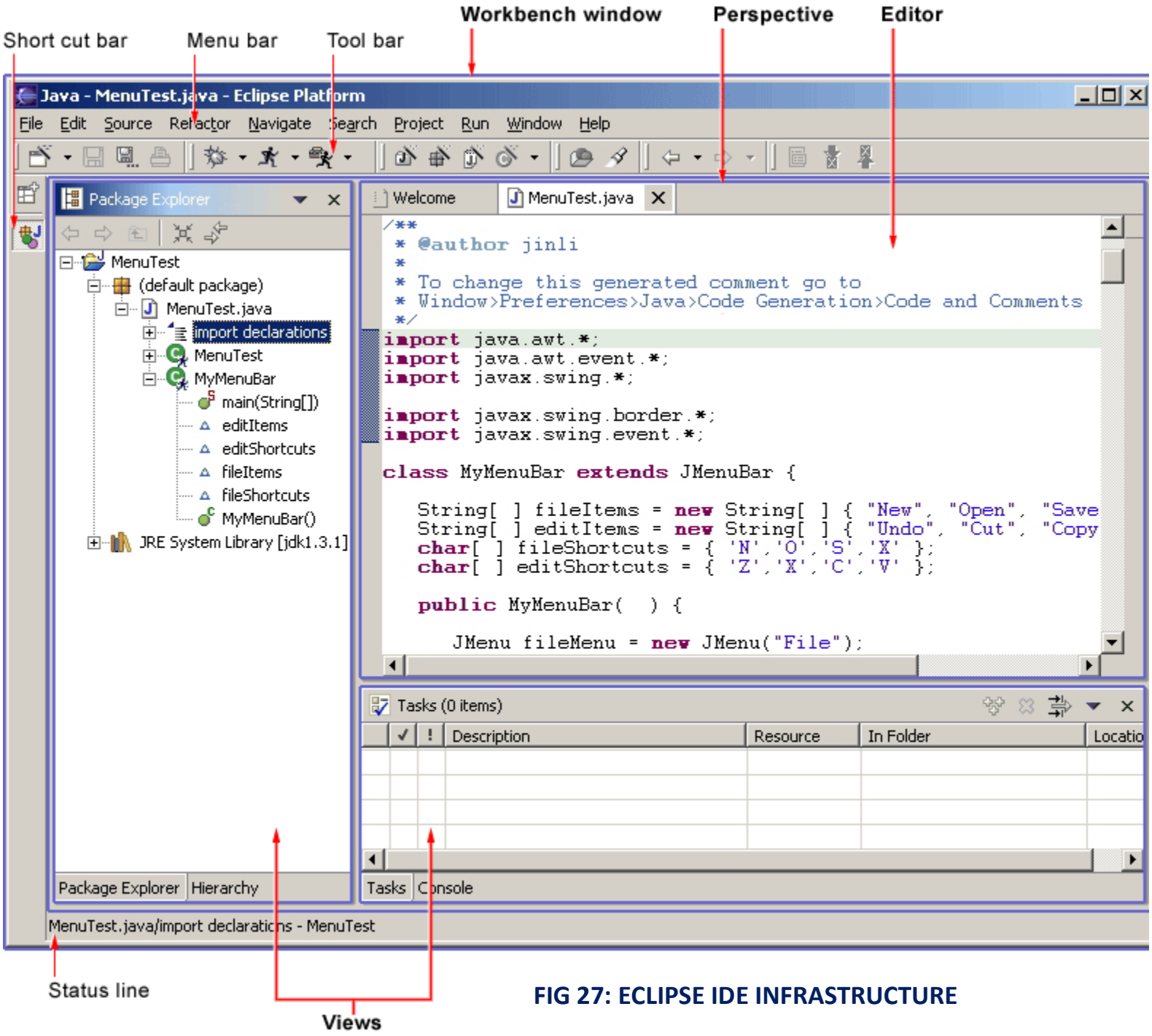


FIG 27: ECLIPSE IDE INFRASTRUCTURE

Source >>*http://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html*

CHATPTER FOUR (4)

4.0 RESULTS AND DISCUSSIONS

This research is aimed at designing QSIM, a Queue Core Assembler and Runtime Simulator for the Queue Core Processor. In QSIM, we have implemented a subset of the Queue Core Instruction Set Architecture.

QSIM has a user friendly Graphical Interface via which a user can interact with the Simulator and carry out desired operations rendered by the Simulator.

Our Simulator provides a platform (File Edit window) for a user to write Queue Core assembly language program, save them before running. Users can alternatively load files containing Queue Core assembly language programs and simulate them.

Without limiting the functionality of our Simulator, we have provided visualization to specific components of QSIM allowing users to view content of such components whiles computation is in progress. In the light of this, the Queue Register window displays the initial content of the Queue Registers before load operation and program execution begin.

The Qregister Dynamics window displays the content of the Queue Register after load and ALU operations. The results of QSIM program execution is displayed at the Qsim Results window.

Our Simulator fetches and executes one instruction at a time.

It is worth mentioning at this point that, QSIM is platform independent owing to the powerful feature provide by its development language (JAVA).

4.1 BENEFITS OF QSIM

QSIM will among other benefits emulate the QC-2 Processor and enable users to debug and execute QC-2 Assembly Programs. Most importantly, QSIM will serve as an educational and research tool hence can be used by students to understand computer architecture, especially microprocessors.

More Queue computations could be explored using the QSIM. It will create the enabling platform to evaluate new Hardware features before implementing them into the actual QC- 2 Hardware.

Furthermore, QSIM will help in determining the efficiency of code generated by the Queue Compiler since it gives statistics of instructions being executed.

4.2. QSIM FEATURES

4.2.1 SPLASH SCREEN

The splash screen is a fore-screen that is displayed and stays for a given numbers of seconds before the actual QSIM application is loaded. This was actually implemented to relief the user of having to wait in idle while the QSIM application is being loaded.

The QSIM splash screen is shown in the diagram below:

FIG 28 : QSIM SPLASH SCREEN

The Graphic User Interface of the QSIM with various components is shown below:

The major components include the QSIM File Editor Window, Qregisters Window, Results Window, Menu and Tools.

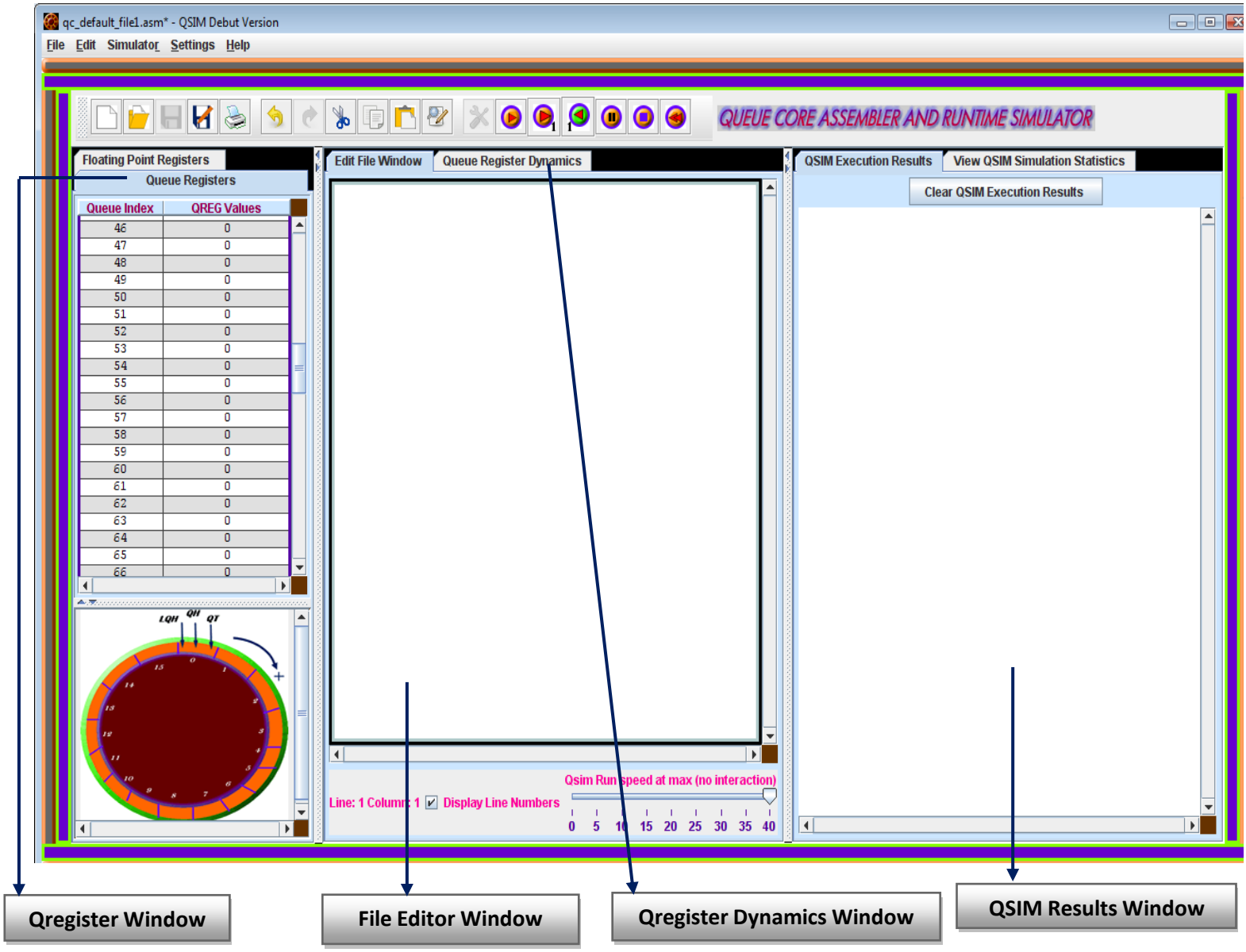


FIG 29: QSIM USER INTERFACE

4.2.3 TOOLS AND MENU BAR COMPONENTS

These components contain the common Menu and Tool items of the QSIM application. It include: File, Edit, New, Simulator, Settings, Copy, Save, Tools etc

More importantly, the Simulator Menu contains the following submenus:

RUN: for executing the source program after successful assemble

STEP: for stepping through the program execution in a stepwise fashion

BACKSTEP: for stepping through the program execution backward in a stepwise fashion
PAUSE: for pausing/ halting the program execution

STOP; for stopping the execution of current program.

RESET: for resetting the current program execution.

These submenus are denoted with symbols as shown below:

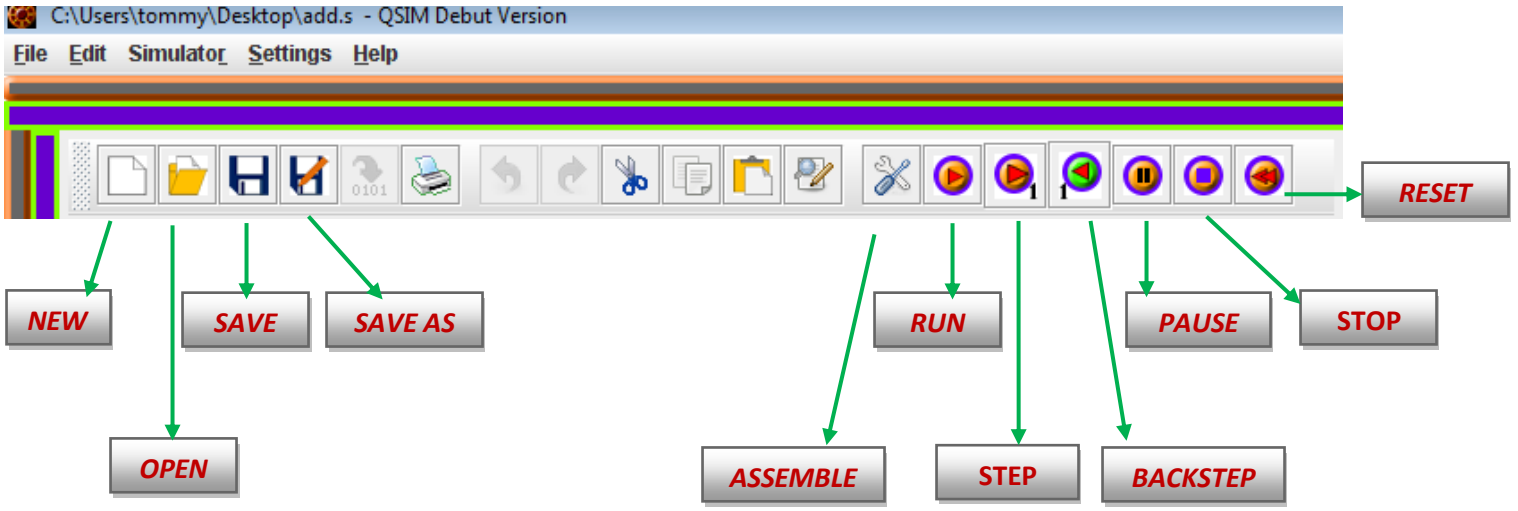


FIG 30: QSIM TOOLS AND MENU

4.2.4 REGISTERS WINDOW

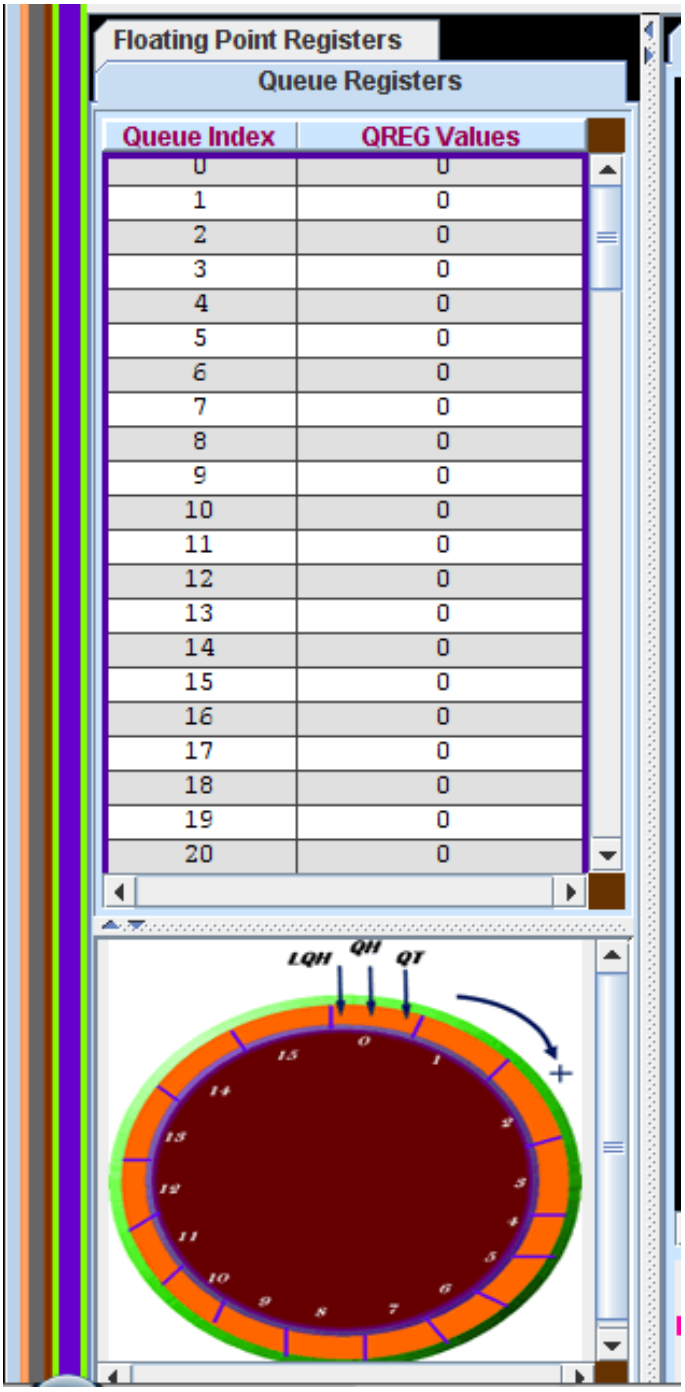
This window consists of two sub-windows which are QREG window and Floating Point window.

QREG WINDOW

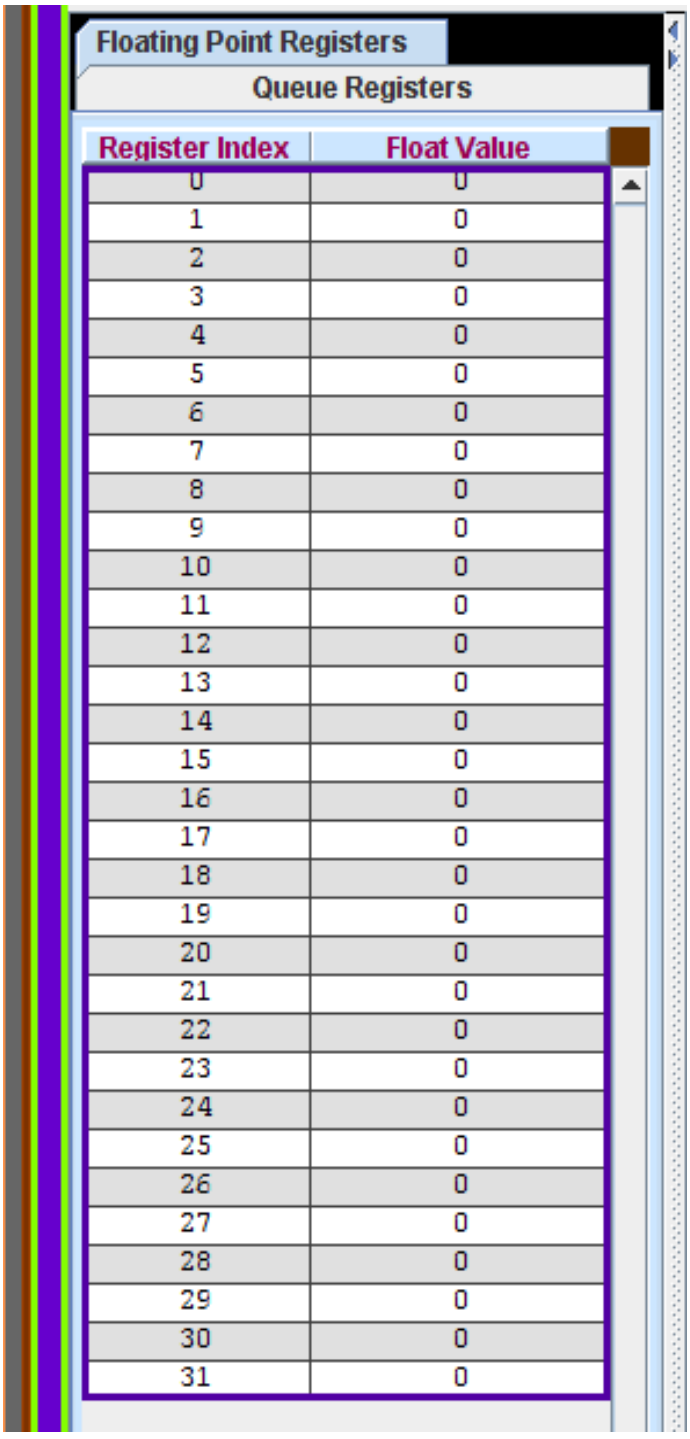
This window consists of Queue Registers that will store the intermediate values of QSIM computations. There are 128 Queue Registers.
The Qregister window displays the initial values of the Queue Register before any operation is performed. Prior to any QSIM operation, the Qregister contents are zeros (0).

The Floating Point Window contains 32 floating point registers. Currently, the content of the floating point registers is zero (0). The floating point register module has not been implemented. For now, we only show the content of the registers hence QSIM cannot perform floating point computations.

Below is the register window showing the Qregister and the Floating Point Registers.



QUEUE REGISTER



FLOATING POINT REGISTERS

FIG 31: QSIM REGISTERS WINDOW

4.2.5 FILE EDIT WINDOW

This component allows the user to load and edit new file containing the QC source code. It also presents a platform for the user to write assembly programs. A default file name (qc_Unsave_file_N.asm) is assigned to newly written unsaved user source program until a user chooses to save with a specified name. ***'N'*** denotes the number of newly created unsaved files.

The File Edit Window is shown in figure 34 below.

4.3. QSIM COMPUTATION PROCEDURE

The procedure below outlines how a user can use our Simulator for Queue computation.

1. Launch the QSIM application
2. Load Queue Core Assembler Language program by clicking on '***OPEN'*** in the File Menu or Tool bar. A user can alternatively write his own Queue Core Assembler language program by clicking on ***'NEW'*** from File Menu or Tool bar. After writing new program, it must be saved before running it.
3. Click on the ***RUN*** button on the Simulator Menu or Tool bar.

NOTE: QSIM accepts three file extensions**: .asm, .s and .qc.** Files loaded or saved without any of these file extensions will not be recognized by QSIM.

4.4 QSIM PROGRAM EXECUTION

4.4.1. QC-2 ASSEMBLY PROGRAM

Interestingly, Queue programs are generated from Level Order Traversal of an Expression Parse Tree. This task is leverage on the compiler generating the Queue Assembly Language program.

The Level Order Traversal has immense implication on the parallelism since it allows instruction in the same level to be executed in parallel.

Consider the expression $x = (a+b) / c-d$,

The Expression Parse Tree for the above expression is show below

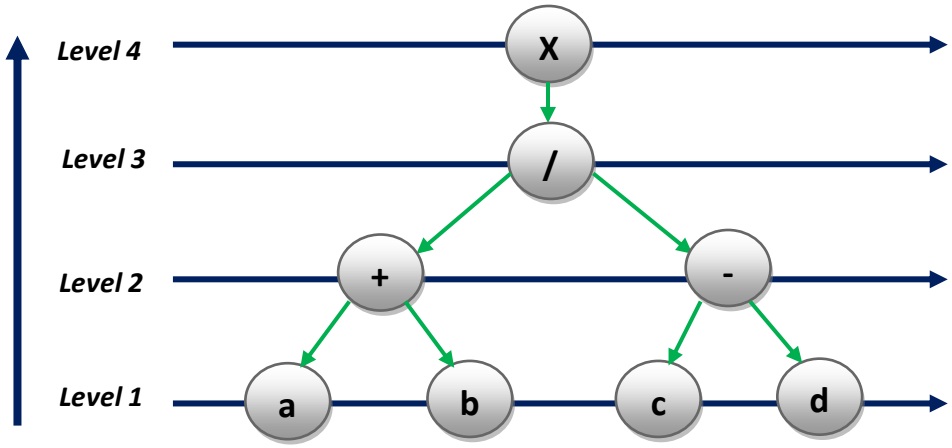


FIG 32: QUEUE ASSEMBLY PROGRAM PARSE TREE

The Queue Core Assembly language program is generated from the parse tree above in a level order manner (starting from level 1). The generated Queue program is shown below:

OPCODE	OFFSETS	COMMENTS
ld	**a**	# load the value of ***a*** into Queue Register
ld	**b**	# load the value of ***b*** into Queue Register
ld	**c**	# load the value of ***c*** into Queue Register
ld	**d**	# load the value of ***d*** into Queue Register
add	**1**	# perform addition operation
sub	**1**	# perform subtraction operation
div	**1**	# perform division operation
st	**x**	# store the results of computation into ***x***

FIG 33: QUEUE CORE ASSEMBLY PROGRAM

4.4.2. Assembly Program Execution

To execute the above generated Queue Core Assembly language program, we need to assign values to the variables a, b, c, d and x. These assigned values will be taken by QSIM as either value for computation or offset indices of the Queue Register depending on the instruction type.

Assuming we make the following arbitrary assignments to the variables above as shown below:

a=10, b=2, c=5, d=3 and x=10

We shall write the above assembly program in a text file and name it say ***Example1.s***

Following the Queue computation procedure above, we shall load the file name ***Example1.s***

into the Edit file window as shown below:

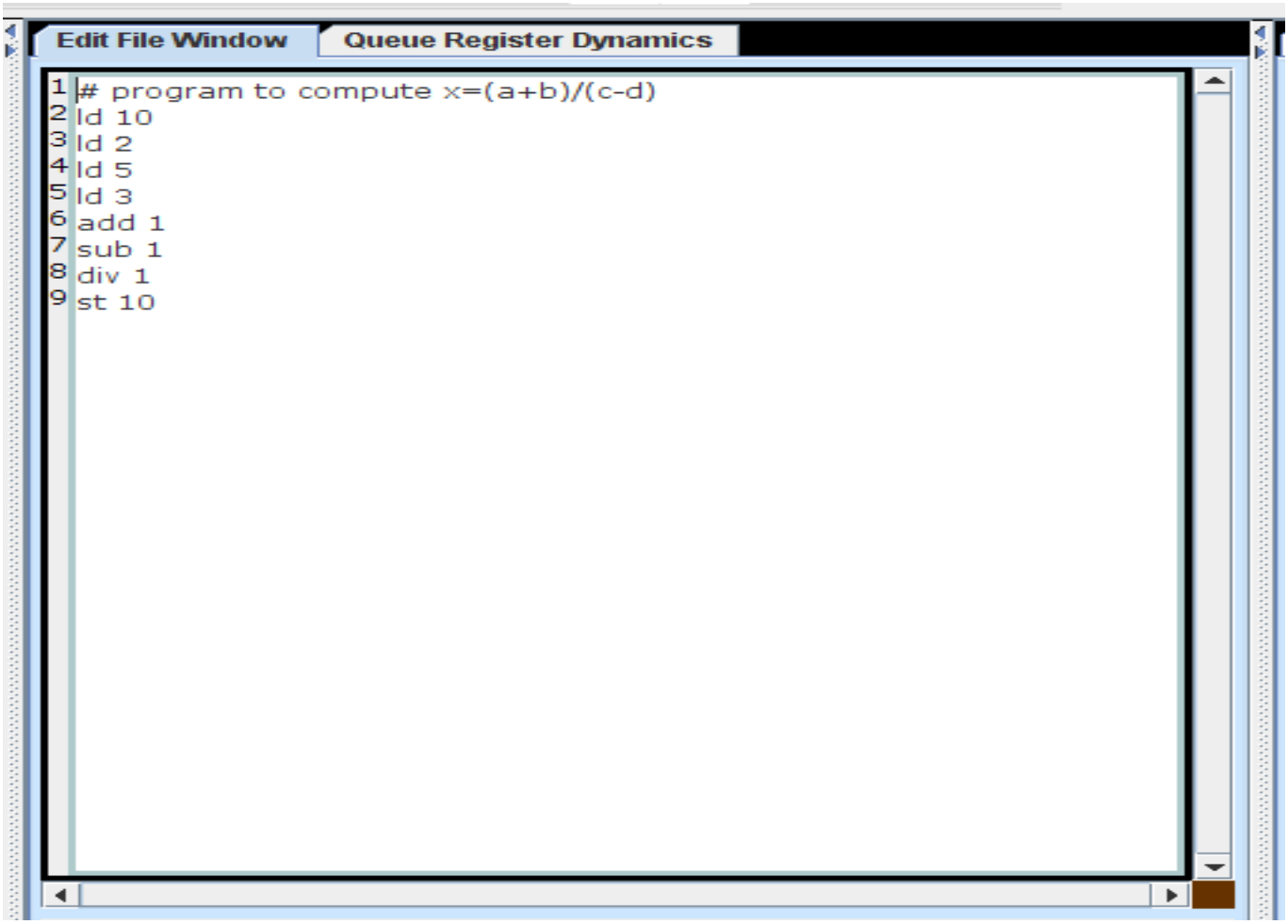


Fig 34: ***Loaded Program into Qsim Edit File Window***

After loading the ***Example1.s*** into the QSIM Edit File window, the Load Instruction (***ld***) stores all values following the load into the Qregister. This is done before any other operation is performed. The content of the Qregister after the load operation is shown in the Queue Register Dynamics window (see: Queue Register Content After load sub-window) as seen below:

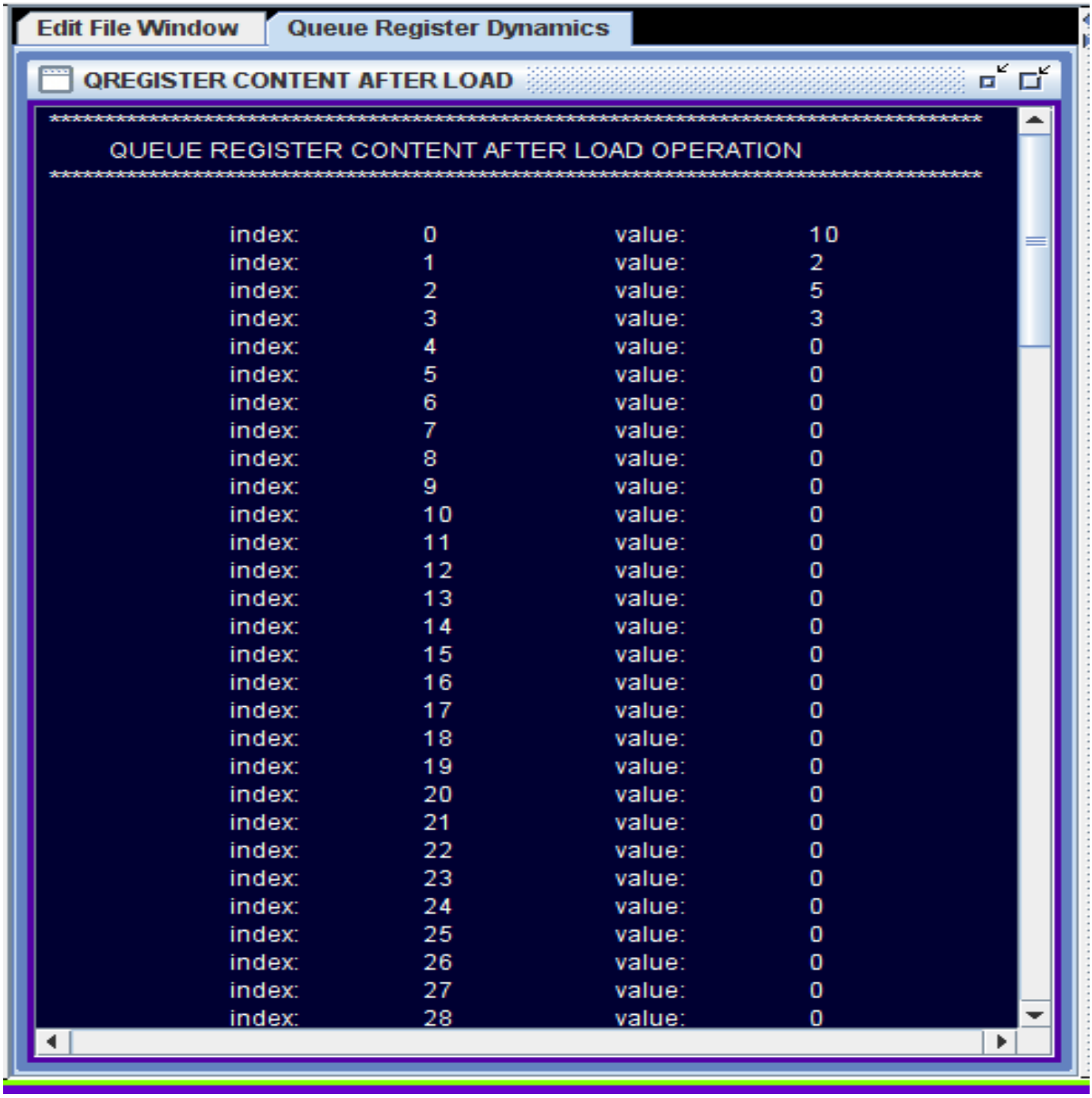


Fig 35: Loaded Values in the Qregister

The loaded values occupied only the first 4 indices out of the 128 Qregisters.

Following the Queue Computation procedure, we are good to execute the loaded Queue Assembly language program. To achieve this, we click on ***RUN*** on the Simulation menu or on the Tool bar.

The results of the QSIM computation are displayed on the QSIM Execution Results Window as shown below

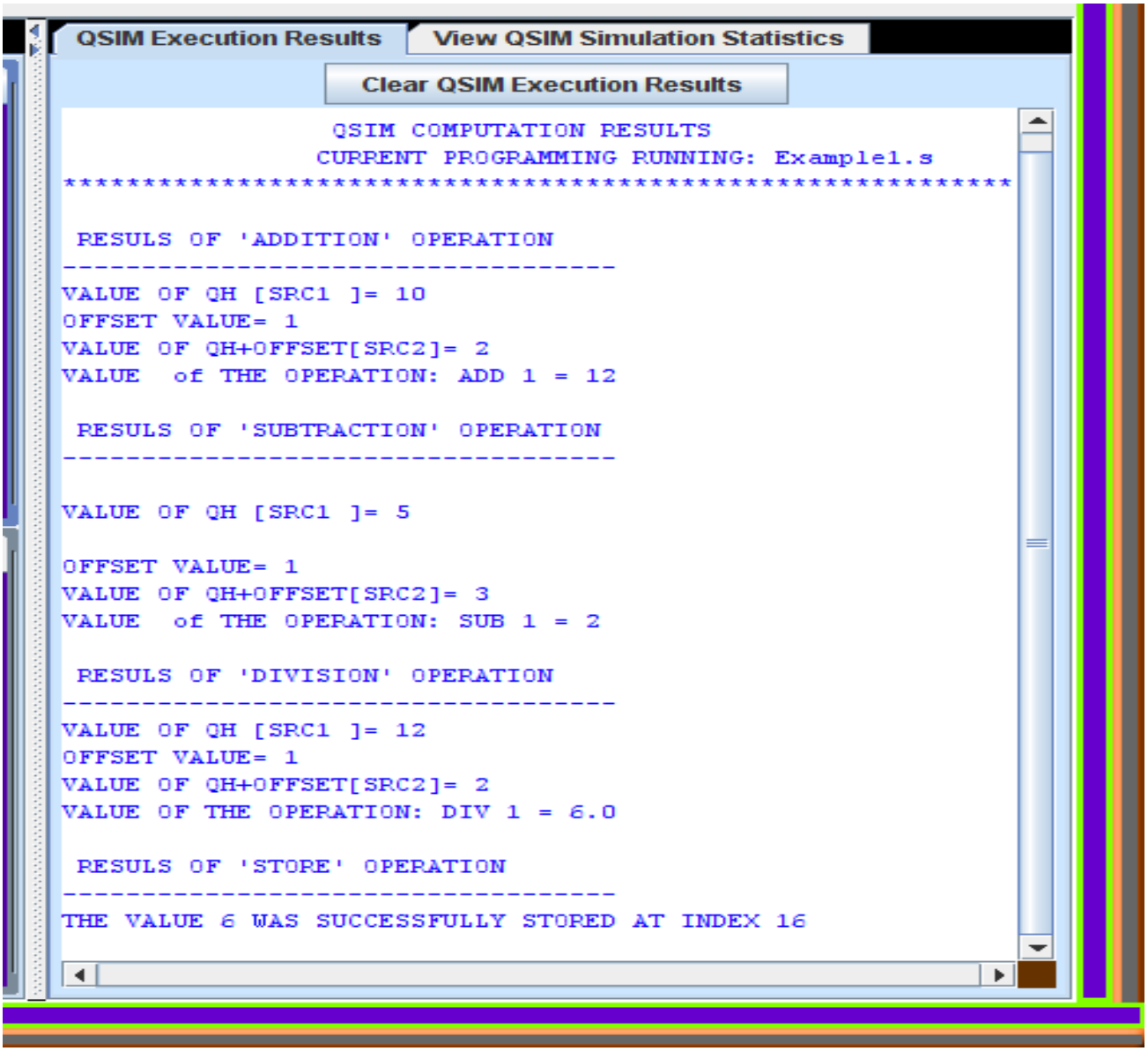


Fig 36: QSIM Execution Results

From the source program above, the first instruction after the load is add 1. The add operation takes its first operand (***src1***) from the Queue Head (QH) and the second operand (***src2***) from an offset reference with respect to the QH.

The result of addition operation from above indicates that the first operand (value of QH)=10.

The offset value =1. Hence to obtain the second operand we add the offset value to the QH index: 0+1. The second operand is therefore taken from index 1 in the Qregister (value of index 1=2).

The results of the add operation is therefore given by 10+2= 12 as shown above. This value is stored at the Queue Tail (QT) of the Qregister.

The next instruction is sub 1. The current QH index is 2 and the QH value =5. The offset value is 1. Hence second operand is taken from index (2+1) =3. The value at index 3=3.

The result of the subtraction operation is 5-2=2 as shown in the results above. This value is stored in the QT.

The next instruction after the ***sub*** instruction is ***div 1***. The current QH index is 4 and the QH value=12 (enqueued results of ***add*** operation). The offset value is 1 hence second operand = 2 (enqueued result of the ***sub*** operation). The result of the div operation is 12/2=6 as shown in the results above.

The last instruction is ***st 10***. This stores the current value of the QH at offset reference 10 with respect to the current QH index. The current QH index is 6 and coincidentally the value of the QH=6. The offset value of the ***st*** operation is 10 hence the effective location is: current QH index +10= 6+10=16. The value 6 is stored at the index 16 of the Qregister as shown in the results above.

The content of the Qregister after the QSIM computation is displayed on the Queue Register Dynamics window (see: Qregister Content after computation sub-window).

Qregister content after the QSIM computation is shown below:

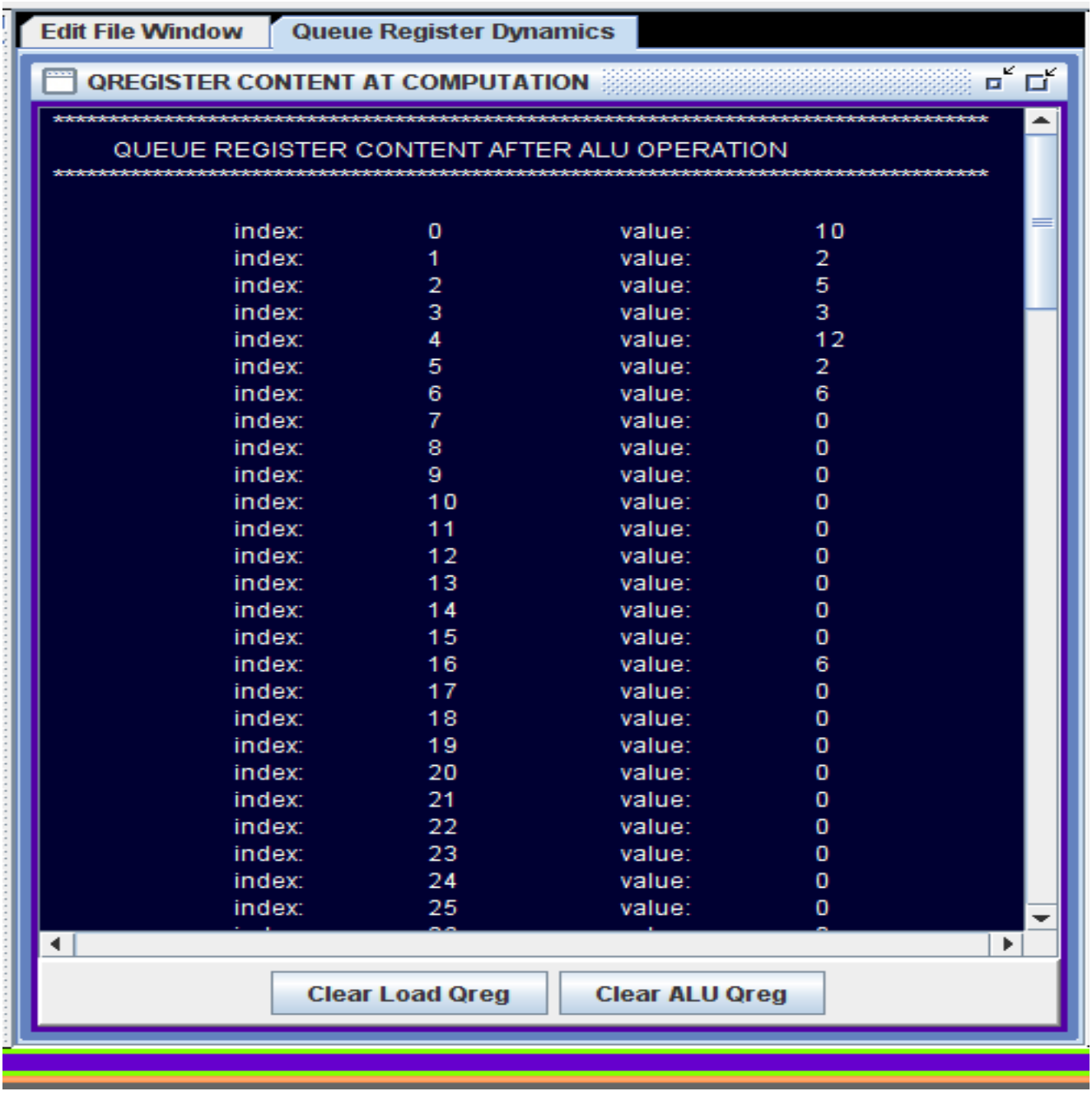


Fig 37: Queue Register Content after QSIM computation

4.4.3 QSIM EXECUTION STATISTICS

Interestingly, our Simulator provides a detailed statistics of the QSIM computation. This includes the total number of executed instruction, number of load instruction, store instructions and branch instructions and number of ALU instructions, coupled with their respective percentages.

It also determines the code size in bytes of the executed Queue Assembly language program.

The simulation time as well as the current program being executed is also determined.

The above statistics are displayed in the QSIM Simulation Statistics Window as shown below:

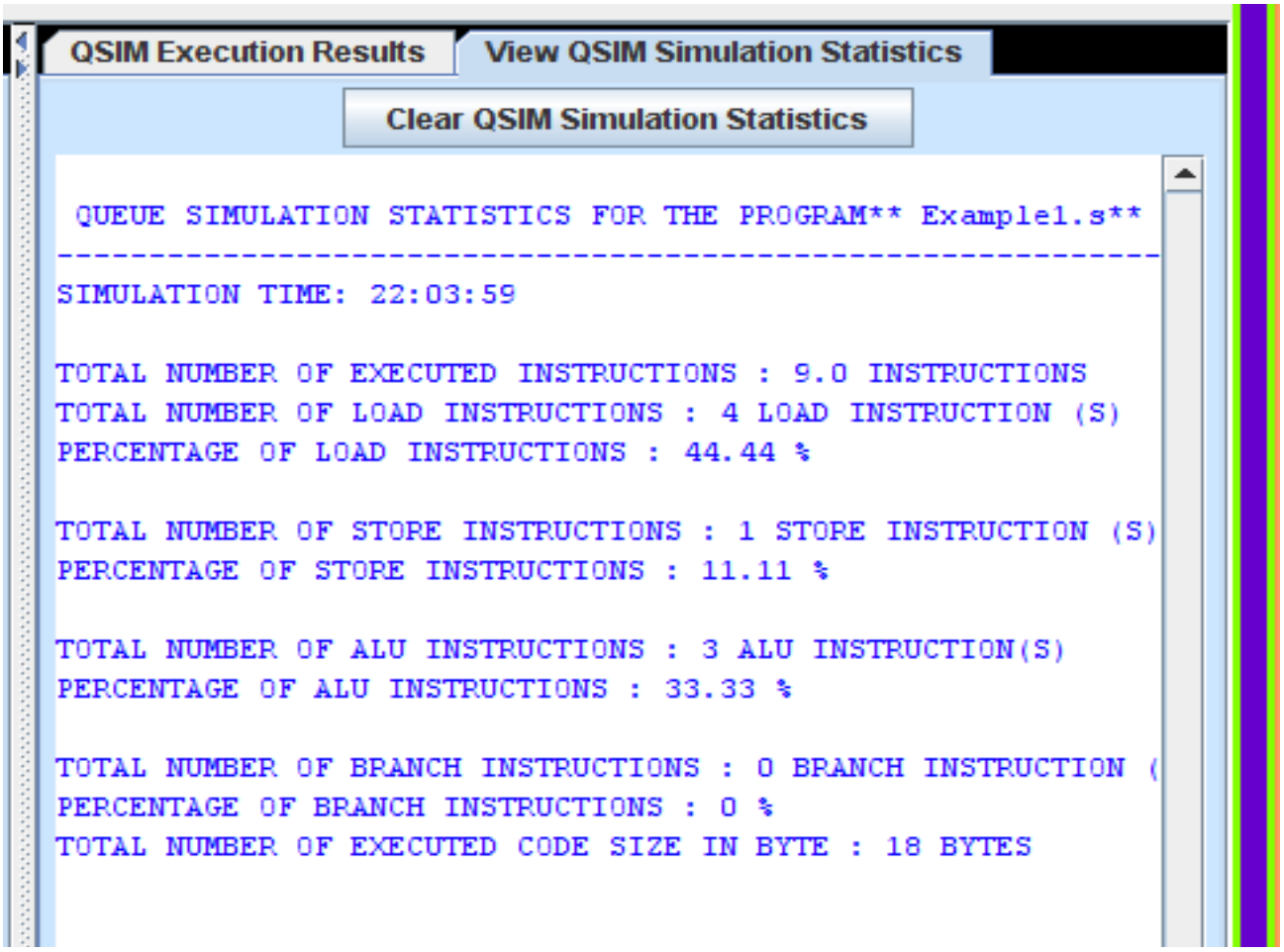


Fig 38: QSIM Computation Statistics

The results above show that from the executed Queue program, the name of the current program executed is ***Example 1.s***. The simulation time is displayed.

The total number of executed instructions is 9. Load instruction dominated with a percentage of 44.44%.

This is followed by ALU instruction with a percentage of 33.33%. There is only 1 store instruction, constituting 11.11%.

No branch instruction was encountered in the program hence 0%.

Finally, the code size in byte for the executed program is 18 byte. This is because each instruction is 16 bits (2 bytes).

The Execution statistics is an important part of the QSIM results because it allows us to determine the efficiency of the code being executed. It helps us to measure the performance of the compiler generating Queue Assembly language programs.

CHAPTER FIVE (5)

5.0 CONCLUSION AND FUTURE WORKS

5.1 CONCLUSION

We have designed QSIM for the Queue Core Processor. It is our strong believe that the advents of QSIM will contribute immensely to the Queue Core Processor and for that matter Queue computation.

It will serve as a tool for more research and evaluation of Queue Processor. Not limiting its usage, QSIM will play a major role as an educational simulator in teaching computer architecture course.

It is easily extensible so needed features and modules could be added whenever required

5.2 FUTURE WORKS ON QSIM

In QSIM, one of our major tasks was to design an in-built Assembler that will convert all Queue Assembly program into machine code before storing them in memory. We could not accomplish this due to time constraints. However, there already exists developed Queue Core Assembler for the Queue Core Processor. This was developed in C-programming language. Integrating this with our QSIM was quite uneasy due to difference in language platforms. Future works could possibly look into this integration.

We could not also design the Memory module for QSIM due to the absence of the Assembler for QSIM. It was expected that the designed Memory will store all assembled instructions before program execution. Future development of QSIM Assembler will therefore necessitate the design of the QSIM Memory module.

More importantly, QSIM will require Data and Text segments that will show the content of the Data and Text Segments of the QSIM Memory. Visualization in the QSIM GUI will be required to show this to the user.

In QSIM, we have designed Registers Components to store integer and floating point intermediate values for Queue Computation. For now, only the Queue Registers have been implemented to handle integer computation. Future works should look into the implementation of the floating point registers for the QSIM.

Some components were added to QSIM but were not implemented due to the absence of QSIM assembler and Memory module. This includes the Speed panel that the user could use to specify how fast or slow execution should be done. The ***Step Forward***, ***Back Step***, ***Stop*** and ***Reset*** are menu and tool bar components designed for future use when the Memory and Assembler components are incorporated. Only the ***Run*** button on the menu or tool bar item has been implemented to execute QSIM programs.

QSIM execute only one instruction at a time. For maximum parallelism, pipeline stages of the Queue Core Processor need to be implemented. Naturally, Queue Core Processor executes four (4) instructions per cycle. Future works should seek this implementation.

REFERENCES

[1]. M. Sowa, B. Abderazek, T. Yoshinaga, Parallel Queue Processor Architecture Based on Produced order Computation Model, The Journal of Supercomputing, Volume 32, Issue 3, 2005, pp. 217-229

[2]. B. Abderazek, S. Kawata, M. Sowa, Design and Architecture for an Embedded 32-bit QueueCore, Journal of Embedded Computing,Vol. 2, No. 2, pp 191-205, 2006

[3]. A. Ben Abdallah, A. Canedo, T. Yoshinaga, and M. Sowa: The QC-2 Parallel Queue Processor Architecture, Journal of Parallel and Distributed Computing, Vol. 68, No. 2, pp.235-245, 2008.

[4]. A. Canedo, B. Abderazek, and M. Sowa: A GCC based Compiler for the Queue Register Processor, Proceedings of International Workshop on Modern Science and Technology, pp. 250-255, May 2006.

[5]. Ben A. Abderazek, Masashi Masuda, Arquimedes Canedo, Natural Instruction Level Parallelism-aware Compiler for High-Performance QueueCore Processor Architecture Adaptive Systems Laboratory, University of Aizu, Japan

[6]. H. Hoshino, A. Ben Abdallah and K. Kuroda: Advanced Optimization and Design Issues of a 32- bit Embedded Processor Based on Produced Order Queue Computation Model, IEEE/IFIP International Conference on Embedded and Ubiquitous Computing EUC2008, pp. 16-22, Dec. 2008.

[7]. Canedo, A., Code Generation Algorithms for Consumed and Produced Order Queue Machines. Master's thesis, University of Electro-Communications, Tokyo, Japan (September 2006).

[8]. S. Okamoto. Design of a Superscalar Processor Based on Queue Machine Computation Model, In IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pages 151-154, 1999.

[9]. A. Canedo, B. Abderazek, M. Sowa, A new code generation algorithm for 2-offset producer order queue computation model, Graduate School of Information Systems, University of Electro-Communications, Japan

[10]. A. Canedo, B. Abderazek, M. Sowa, An Efficient Code Generation Algorithm for Code Size Reduction using 1-offset P-Code Queue Computation Model, Graduate School of Information Systems, University of Electro-Communications, Japan

[11]. Canedo, Ben Abderazek, and Masahiro Sowa, Compiler Support for Code Size Reduction using a Queue-based Processor Arquimedes Graduate School of Information Systems, University of Electro-Communications, Japan

[12]. Ben A. Abderazek, QC1 Processing Stages Algorithms *Technical Report, TRQC1PSA03,* October 3rd, 2003

[13]. ASL-Ben Abdallah Group, Qasm Technical Report, Ref. TR12010, University of Aizu

[14]. Hiroki Hoshino, QC-2 Data Path Version 2.0, Adaptive Systems Laboratory, University of Aizu, October 2009

[15]. Abderazek Ben Abdallah, Technical report QC2MY07, University of Electro-communications Graduate School of Information Systems, May 2007

[16]. Ben A. Abderazek; Queue Core Instruction set architecture, January 2003

[17]. ASL-BENABDALLAH Group, Queue Machines, University of Aizu, August 2010

[18]. http://en-wikipedia.org/wiki/simulation, September 17, 2010

[19]. http://www.wisegeek.com, September 17, 2010

[20]. http://www.answers.com/topic/simulation , September 17, 2010

[21]. http://www.oxfordadvancedlearnersdictionary.com/dictionary/simulation

[22]. http://encyclopedia2.thefreedictionary.com/simulation, September 18, 2010

[23]. http://www.sparxsystems.com/resources/uml2_tutorial, September 19, 2010

[24]. http://www.nada.kth.se/~karlm/prutt05/lectures/prutt05_lec6.pdf, September 19, 2010

[25]. R.Max Wideman, The Role of Project Life Cycle in Project Management, updated February, 2004

[26]. Walt Scacchi, Process Models in Software Engineering, Institute of Software Research, University of California, Irvine, February 2001.

[27]. http://www.softpanorama.org/SE/software_life_cycle_models.shtml , September 21, 2010

[28]. Bernd Bruegge and Allen H. Dutoit, Object-Oriented Software Engineering: using UML, Patterns, and Java.

[29]. http://docstore.mik.ua/orelly/java-ent/jnut/cho1_02.htm

[30].http://www.streetdirectory.com/travel_guide/114362/programming/most_significant_advantages_of_java_language.html.

[31]. http://en.wikipedia.org/wiki/integrated_development_environment

[32]. B. Preiss and C. Hamacher, Data Flow on Queue Machines. In 12th Int. IEEE Symposium on computer Architecture, pages 342-351, 1987.

[33]. Gary S, Mikhail S. Edward S., Evaluating the use of Register Queues in Software Pipelined Loops, Advanced Computer Architecture Lab, University of Michigan

[34]. Schmitt H., Benjamin L., Benjamin, Y., Queue Machines, Hardware Compilation in Hardware, Carnegie Mellon University.

[35]. Fernandes, Marcio M, A Clustered VLIW Architecture Based on Queue Register Files, school of informatics, College of Engineering, University of Edinburgh, 1998

[36]. S. Okamoto, H. Suzuki, A. Maeda and M. Sowa, Design of a Superscalar Processor Based on Queue Machine Computation Model: IEEE PACRIM, 1999, pp. 22-24.

[37]. B. A. Abderazek, N. Kirilka, and M. Sowa: FARM queue Mode: On a Practical Queue Execution model, Proceedings of the Int. Conf. on Circuits and Systems, Computers and Communications, Tokushima, Japan, 2001, pp. 939-944.

[38]. H. Suzuki, O. Shusuke, A. Maeda and M. Sowa, Implementation and evaluation of a Superscalar, Processor Based on Queue Machine Computation Model, IPSJ SIG, 99(21), 1999, 91-96.

[39]. M. Sowa, "Queue Processor Instruction Set Design", the University of Electro Communications, Sowa laboratory, Technical Report SLL97301, 1997.

[40]. M. Sowa, B. A. Abderazek, S. Shigeta, K. Nikolova, and T. Yoshinaga, " Proposal and Design of a Parallel Queue Processor Architecture (PQP) ", 14th IASTED Int. Conf. on Parallel and Distributed Computing and System, Cambridge, USA, Nov. 2002, pp. 554-560.

[41]. B.A. Abderazek, M. Arsenji, S. Shigeta, T. Yoshinaga, M. Sowa, Queue Processor for novel queue computing paradigm based on produced order scheme, in: HPC2004, International Conference on High Performance Computing, Tokyo, July 2004, pp. 169–177.

[42]. B.A. Abderazek, Dynamic instructions issue algorithm and a queue execution model toward the design of hybrid Processor architecture, Ph.D. Thesis, Graduate School of Information Systems, the University of Electro-Communications, 2002.

[43] Object Management Group (OMG), Unified Modeling Language Specification, Version 2.1.1, (2007-02-07).

[44] Booch, G., Object Oriented Design with Applications, Benjamin/Cummings, Redwood, California, 1991. ISBN: 0-8053-0091-0, ISSN: 0896-8438

[45] Coad, P., and E. Youdon, Object-Oriented Analysis, Prentice Hall, Eaglewood Cliffs, New Jersey, 1991. ISBN: 0-13-630070-7

[46] http:// course.missouristate.edu/kenVollmar/MARS/index.htm

[47] http:// pages.cs.wisc.edu/~larus/spim.html

APPENDIX I

INSTRUCTION SET ARCHITECTURE FOR QC-2 PROCESSOR

The QC-2 Processor implements a produced order instruction set architecture. Here, each instruction can encode at most 2 operands that specify the location in the register file where to read the operands. The Processor determines the physical location of operands by adding the offset reference in the instruction to the current QH pointer.

All instructions of QC-2 Processor are 16-bit wide, allowing simple fetch and decode stages and facilitating pipelining of the Processor. In the case of insufficient bit to express large constants, memory offsets or offset references, the QueueCore Processor implements the QCaEXT which inserts a “covop” instruction. This special instruction is used to extend load and store instructions offsets and also extends immediate values when necessary. This consequently curbs the bit-width constraints in the QueueCore Processor.

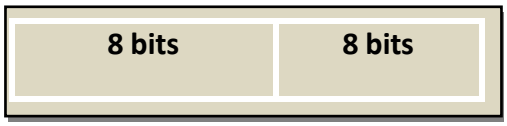
INSTRUCTION FORMAT AND ENCODING FOR THE QC-2 ARCHITECTURE

QC-2 has a fixed instruction length format where each instruction has a fixed length of 16-bit wide. This obviates the need for instruction length checking in the decode stage hence enhancing instruction processing performance in the QC-2 Processor.

INSTRUCTION FORMAT

QC-2 instruction set architecture uses mnemonics to denote operations. Each operation has a unique operation code (opcode). The instruction format for some of selected operations of the QC-2 instruction set is given below:

INSTRUCTION FORMAT FOR ***COVOP INSTRUCTION***



15	8	7	0
covop: 00000100		**Addr 1**	

Mnemonic	Action	QH	QT	Binary
Covop addr1	Convey address	0	0	**00000100**

Description: Convey an 8-bit address to a load or store instruction to extend the addressing bits from 8 to 16 bits. Here, QH = 0 & QT = 0

INSTRUCTION FORMAT FOR ***LOAD / STORE INSTRUCTION***

LOAD INSTRUCTION: ***Ld k: k=byte, string, word (k could be unsigned or signed)***

15	10	9	8	7	0
Load :opcode		**d**		**addr 0**	

Mnemonic	Action	QH	QT	Binary
Ld*k addr0(d)*	QTw ← m((d)+addr1.addr0)	0	1	***opcode***

Description: the above instruction load ***K*** (byte, string or word-signed/unsigned) to the operand queue pointed by the QT from memory address ((d)+addr0) or((d)+addr1.addr0).

STORE INSTRUCTION: ***st k: k=byte, string, word (k could be unsigned or signed)***

15 — 10	9 — 8	7 — 0
Load :opcode	**d**	**addr 0**

Mnemonic	Action	QH	QT	Binary
St *k addr0(d)*	QHw → m((d)+addr1.addr0)	1	0	***opcode***

Description: the above instruction store ***K*** (byte, string or word-signed/unsigned) to the operand queue pointed by the memory address ((d) +addr0) or ((d) +addr1.addr0) from the Queue Head (QH).

INSTRUCTION FORMAT FOR ***IMMEDIATE INSTRUCTION***

8 bits	8 bits

LOAD IMMEDIATE: ***Ldi k: k=value or address***

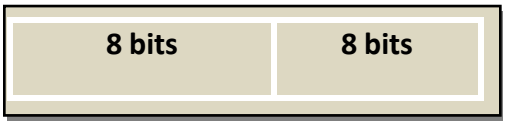
15 — 8	7 — 0
Load :opcode	**value**

Mnemonic	Action	QH	QT	Binary
Ldi k value	**QTw ← (value)/(address of inst+value)**	**1**	**0**	***opcode***

Description: the above instruction load ***K*** (immediate value or address) to the operand queue pointed by the QT from memory address ((d) +addr0) or ((d) +addr1.addr0). When k=address, PQPcfo and PQPcf+ are set to yes.

INSTRUCTION FORMAT FOR CONTROL INSTRUCTIONS

BRANCH INSTRUCTION



BRANCH TARGET/ BRANCH *K* TARGET: *b t/ b k t: k=eq, ne, lt, gt, le, ge*

15 8	7 0
b/b k :opcode	**t**

Mnemonic	Action	QH	QT	Binary
b t/ b k t	FC ← pc+2(addr1.t) , if cc= k or Ø	0	0	*opcode*

Description: This instruction to target the operand queue pointed by the program counter ((pc+t) or (pc+addr1.t)) to FC. CC refers to the condition code.

K=eq: branch equal (be)

K=ne: branch not equal (bne)

k=lt: branch less than (blt)

k=gt: branch greater than (bgt)

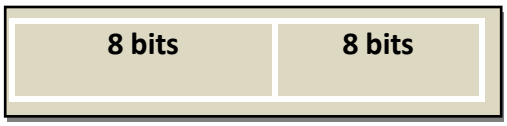
k=ge: branch greater or equal (bge)

k=le: branch less or equal (ble)

b t: branch target

QUEUE CONTROL INSTRUCTIONS FORMAT

STOP QH/LQH MOVE



STOP QH/LQH MOVE: stpqh /stplqh n

15 — 8	7 — 0
Stpqh/ stplqh :opcode	**n**

Mnemonic	Action	QH	QT	Binary
stpqh/stplqh n	**Stop QH/LQH moving**	**QH=9; LQH=0**	**0**	***opcode***

Description: This instruction stops QH/LQH move to target the operand queue pointed that the QH/LQH stop to move from n^{th} position.

FIXED QH AND LQH AUTOMATICALLY

AUTO FIXED QH/ LQH: autqh /autlqh n

15 — 8	7 — 0
autqh / autlqh :opcode	**n**

Mnemonic	Action	QH	QT	Binary
autqh/autlqh n	**Fixed QH/LQH automatically**	QH=9; LQH=0	0	***opcode***

Description: This instruction to target the operand queue pointed that the QH/LQH will be fixed in the nth position

INSTRUCTION FORMAT FOR ***JUMP/ CALL***

6 -bits	2 -bits	8-bits

JUMP/ CALL TARGET: jump/ call t(a)

15 10	9 8	7 0
Jump/call: opcode	**a**	**t**

Mnemonic	Action	QH	QT	Binary
Jump/call t(a)	FC ← (a)+2(addr1)	0	0	***opcode***

Description: This instruction to target the operand queue pointed by the memory address ((a) +t) or ((a) +2(addr1.t)) to FC. '(a)': refers to the content of register 'a'.

INSTRUCTION FORMAT FOR ***BARRIER / INTERRUPT***

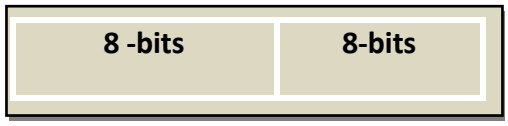
8 bits	0

The following instructions use the above instruction format:

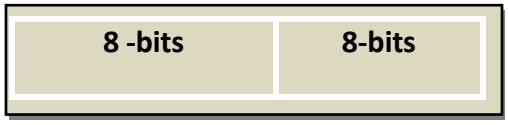
Return from call ***(rfc),*** return from interrupt ***(rfi),*** no operation ***(nop)***, stop ***(halt)***, barrier ***(bar)***

SHIFT INSTRUCTION FORMAT

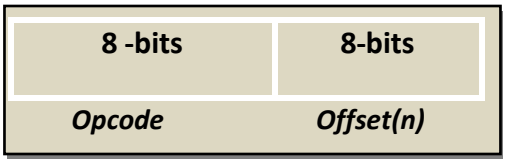
ROTATE INSTRUCTION FORMAT



DUPLICATE/ MOVE INSTRUCTION FORMAT



ALU INSTRUCTION FORMAT FOR SINGLE WORD



OPERATION n: operation (add, *addu, sub, subu, subo, subuo, mul, mulu, div, divu, divo, mod*)

15 — 8	7 — 0
Operation: opcode	n

Mnemonic	Action	QH	QT	Binary
Operation n	qtw0 ← qhw0 op (qh+n) w	1	1	*opcode*

Description: This instruction performs the operation on two single operands to the operand queue pointed by QT from (qhw0 ***op*** (qh+n) w). Here, 'op' refer to the specified operator (+, - , * , /etc).

ALU INSTRUCTION FORMAT FOR DOUBLE WORD

The instruction format for double is similar to that for single word except that each operation is double word instead of single. Each operation is appended with d (double)

OPERATION d n: operation (addd, *adddu, subd, subdu, subdo, subduo, muld, muldu, divd, divdu, divdo, modd*)

15 — 8	7 — 0
Operation: opcode	n

Mnemonic	Action	QH	QT	Binary
Operation n	qtd0 ← qhd0 op (qh+n) d	2	2	*opcode*

Description: These instructions perform operation on two double operands to the operand queue pointed by the QT from (qhd0 ***op*** (qh+n) d).

APPENDIX II

QUEUE CORE ASSEMBLER AND RUNTIME SIMULATOR SOURCE CODES

The selected source codes are provided according to packages.

QSIM .REGISTERS PACKAGE

Qregister.JAVA CLASS

```
package qsim.registers;
import java.util.Observable;

/**Class for the Queue Register Object */
public class QRegister extends Observable{
        private int index,resetValue;
        private String name;
        private volatile int value;

        /**constructor for new Queue register */
public QRegister(int index,int value){
        this.index=index;
        this.value=value;
        this.resetValue=value;
}

        /**constructor for new Floating point register */
public QRegister(String name, int value){
        this.name=name;
        this.value=value;
        this.resetValue=value;
}

        /*****************************************/
        /**getter methods for QRegister variables**/
        /*****************************************/
```

```
        /**get index of a QRegister*/
public int getIndex(){
        return index;
}

        /** Return the name of the Floating point Register*/
public String getName(){
        return name;
}

        /** Returns the value of the QRegister.  Observers are notified of the READ operation.
           @return value The value of the QRegister.
         */
 public synchronized int getValue(){
        notifyAnyObservers(AccessNotice.READ);
        return value;
}


        /**  Returns the value of the QRegister.  Observers are not notified
         *   @return value The value of the QRegister.*/
 public synchronized int getValueNoNotify(){
        return value;
}

        /**  Returns the reset value of the QRegister.
         *   @return The reset (initial) value of the QRegister.
public int getResetValue(){
        return resetValue;
}
```

```
        /**  Sets the value of the QRegister to the val passed to it.
         *   Observers are notified of the WRITE operation.
         *   @param val Value to set the QRegister to.
         *   @return previous value of Qregister*/
public synchronized int setValue(int val){
        int old = value;
        value = val;
        notifyAnyObservers(AccessNotice.WRITE);
        return old;
}

        /**  Resets the value of the QRegister to the value it was constructed with. Observers are
        not notified.*/
   public synchronized void resetValue(){
        value = resetValue;
}

        /**  Change the QRegister's reset value; the value to which it will be
         *  set when (resetValue()) is called */
   public synchronized void changeResetValue(int reset) {
        resetValue = reset;
}

        /** Method to notify any observers of register operation that has just occurred.*/
   private void notifyAnyObservers(int type) {
        if (this.countObservers() > 0){
                this.setChanged();
                this.notifyObservers(new QRegisterAccessNotice(type, this.index));
                }
        }
}
```

QregisterFile.JAVA CLASS

```
package qsim.registers;
import java.util.Observer;
import javax.swing.JOptionPane;
import qsim.Globals;
import qsim.gui.QRegisterSegmentForAlu;
import qsim.gui.QRegisterSegmentForLoad;
import qsim.gui.QsimResultsPane;


/** Class to create the Queue Registers */
public class QRegisterFile {
        public static final int STACK_POINTER_REGISTER = 0;
        private static int QH=0;
        private static int QT=0;
        private static int CN=0;
        private static int PN=0;
        private static int qSize=0;
        private int qMaxSize=256;
        QsimResultsPane messagePane=new QsimResultsPane ();

         /** creating the Queue register with initial values */
        private static QRegister []qReg={
                new QRegister(0,0),new QRegister(1,0),
                new QRegister(2,0),new QRegister(3,0),
                new QRegister(4,0),new QRegister(5,0),
                new QRegister(6,0),new QRegister(7,0),
                new QRegister(8,0),new QRegister(9,0),
                new QRegister(10,0),new QRegister(11,0),
                new QRegister(12,0),new QRegister(13,0),
                new QRegister(14,0),new QRegister(15,0),
```

```
new QRegister(16,0),new QRegister(17,0),
new QRegister(18,0),new QRegister(19,0),
new QRegister(20,0),new QRegister(21,0),
new QRegister(22,0),new QRegister(23,0),
new QRegister(24,0),new QRegister(25,0),
new QRegister(26,0),new QRegister(27,0),
new QRegister(28,0),new QRegister(29,0),
new QRegister(30,0),new QRegister(31,0),
new QRegister(32,0),new QRegister(33,0),
new QRegister(34,0),new QRegister(35,0),
new QRegister(36,0),new QRegister(37,0),
new QRegister(38,0),new QRegister(39,0),
new QRegister(40,0),new QRegister(41,0),
new QRegister(42,0),new QRegister(43,0),
new QRegister(44,0),new QRegister(45,0),
new QRegister(46,0),new QRegister(47,0),
new QRegister(48,0),new QRegister(49,0),
new QRegister(50,0),new QRegister(51,0),
new QRegister(52,0),new QRegister(53,0),
new QRegister(54,0),new QRegister(55,0),
new QRegister(56,0),new QRegister(57,0),
new QRegister(58,0),new QRegister(59,0),
new QRegister(60,0),new QRegister(61,0),
new QRegister(62,0),new QRegister(63,0),
new QRegister(64,0),new QRegister(65,0),
new QRegister(66,0),new QRegister(67,0),
new QRegister(68,0),new QRegister(69,0),
new QRegister(70,0),new QRegister(71,0),
new QRegister(72,0),new QRegister(73,0),
new QRegister(74,0),new QRegister(75,0),
new QRegister(76,0),new QRegister(77,0),
```

```
        new QRegister(78,0),new QRegister(79,0),
        new QRegister(80,0),new QRegister(81,0),
        new QRegister(82,0),new QRegister(83,0),
        new QRegister(84,0),new QRegister(85,0),
        new QRegister(86,0),new QRegister(87,0),
        new QRegister(88,0),new QRegister(89,0),
        new QRegister(90,0),new QRegister(91,0),
        new QRegister(92,0),new QRegister(93,0),
        new QRegister(94,0),new QRegister(95,0),
        new QRegister(96,0),new QRegister(97,0),
        new QRegister(98,0),new QRegister(99,0),
        new QRegister(100,0),new QRegister(101,0),
        new QRegister(102,0),new QRegister(103,0),
        new QRegister(104,0),new QRegister(105,0),
        new QRegister(106,0),new QRegister(107,0),
        new QRegister(108,0),new QRegister(109,0),
        new QRegister(110,0),new QRegister(111,0),
        new QRegister(112,0),new QRegister(113,0),
        new QRegister(114,0),new QRegister(115,0),
        new QRegister(116,0),new QRegister(117,0),
        new QRegister(118,0),new QRegister(119,0),
        new QRegister(120,0),new QRegister(121,0),
        new QRegister(122,0),new QRegister(123,0),
        new QRegister(124,0),new QRegister(125,0),
        new QRegister(126,0),new QRegister(127,0),
    };
```

```
        /** method to show the content of the QRegisters */
public static void showQReg(){
Globals.getGui().getMainPane().getQRegisterDynamicsPane().getaluSegmentWindow();
        QRegisterSegmentForAlu.getAluQreg().append("************************\n");
QRegisterSegmentForAlu.getAluQreg().append("QUEUE REGISTER CONTENT AFTER ALU
OPERATION \n");
QRegisterSegmentForAlu.getAluQreg().append("*************************\n\n");
        for(int i=0;i<qReg.length;i++){
        QRegisterSegmentForAlu.getAluQreg().append("index:\t"+qReg[i].getIndex()+"\t"+
"value: \t"+qReg[i].getValue()+"\n");
                }
        }

        /** method to show content of QRegister after load instruction */
public static void showLoadQReg(){
Globals.getGui().getMainPane().getQRegisterDynamicsPane().getaluSegmentWindow();

        QRegisterSegmentForLoad.getLoadQreg().append("************************\n");
                QRegisterSegmentForLoad.getLoadQreg().append(" QUEUE REGISTER
CONTENT AFTER LOAD OPERATION \n");

        QRegisterSegmentForLoad.getLoadQreg().append("*************************\n\n");
                for(int i=0;i<qReg.length;i++){

        QRegisterSegmentForLoad.getLoadQreg().append("index:\t"+qReg[i].getIndex()+"\t"+
"value: \t"+qReg[i].getValue()+"\n");
                }
        }
```

```
        /** method to return a value at particular index of the QRegister*/
public static int getIndexValue(int index){
        return qReg[index].getValue();
}

        /** check if queue is full */
public boolean isFull(){
        if(qMaxSize==size()){
                return true;
        }
        else
                return false;
}

        /**check for empty queue */
public boolean isEmpty(){
        if (qReg[QH]==null){
                return true;
        }
        else
                return false;
}

        /** enqueue a value at the QT*/
  public static void enqueue(int value){
        qReg[QT].setValue(value);
        QT=(QT+1)%qReg.length;
        qSize++;
}
```

```
        /** dequeue a value at QH*/

public static int dequeue(){
        int consume=getQHValue();
        QH=(QH+1)%qReg.length;
        return consume;
}

    /**method to store a value at a particular index */
public static void setStoreValue(int val){
        int index=(QH)+val;
        int regIndx;
        if(index<0){
                JOptionPane.showMessageDialog(null, "Offset index is out of range");
        }
        else{
                int head=dequeue();
                for(int i=0;i<qReg.length;i++){
                        regIndx=qReg[i].getIndex();
                        if(regIndx==index){
                                qReg[i].setValue(head);
                        }
                }
        }
}

        /** method to get offset index*/
public static int getOffsetIndex(int val){
        int index=(QH)+val;
        return index;
}
```

```
/** method for getting offset value*/
public static int getOffsetValue(int j){
        int offsetIndex=(QH-1)+j;
        if(offsetIndex<0){
                JOptionPane.showMessageDialog(null, "Offset index is out of range");
                return 0;
        }
        else{
                int offsetValue=getIndexValue(offsetIndex);
                QH=(QH+1)%qReg.length;
                return offsetValue;
        }
}

/** method to return the current size of the QRegister*/
public int size(){
        return (qMaxSize-QH+QT)%qMaxSize;
}

public int qLength(){
        return qMaxSize;
}

/** get number of consume data*/
 public static int getConsumeData(){
        return CN;
}
        /** get number of produce data*/
public static int getProduceData(){
        return PN;
}
```

```
/** method to return current Queue Tail Index*/
public static int getQT(){
        return QT;
}

/** Method to return current Queue Head Index*/
public int getQH(){
        return QH;
}

/** method to set QT value */
public void setQTValue(int val){
        qReg[QT].setValue(val);
}

/**Method to return the value of Queue Head*/
public static int getQHValue(){
        int value=qReg[QH].getValue();
        CN++;
        return  value;
}

/**Method to get current Queue Tail value */
public int getQTValue(){
        int val=qReg[QT-1].getValue();
        return val;
}
```

```
/**Method to reset the Queue Register values to zero (0) */
public static void clearReg(){
        for(int i=0;i<qReg.length;i++){
        qReg[i].setValue(0)    ;
        }
}

/**Method to update the QRegiste value to a given value */
public static void updateQRegister(int index, int val){
        qReg[index].setValue(val);
}

/**  Method For returning the set of QRegisters.*/
public static QRegister[]getQRegister(){
        return qReg;
}

/**  Method to reinitialize the values of the QRegisters**/
public static void resetQRegisters(){
        for(int i=0; i< qReg.length; i++){
                qReg[i].resetValue();
        }
}

/**  Each individual QRegister is a separate object and Observable.  This method
 *  adds the given Observer to each one */
public static void addRegistersObserver(Observer observer) {
        for (int i=0; i<qReg.length; i++) {
                qReg[i].addObserver(observer);
        }
}
```

```
/**  Each individual register is a separate object and Observable.  This handy method
 *  will delete the given Observer from each one.  Currently does not apply to Program
 * Counter */
public static void deleteRegistersObserver(Observer observer) {
                for (int i=0; i<qReg.length; i++) {
                        qReg[i].deleteObserver(observer);
                }
        }
}
```

QInstructionSet.JAVA CLASS

```
package qsim.registers;
import javax.swing.JOptionPane;
import qsim.*;

/** The list of Instruction objects, each of which represents a QC instruction */

public class QInstructionSet{

        /*****************************************************************/
        /** list of implemented Queue Core Instruction Set Architecture*/
        /*****************************************************************/

        /**********/
        /** ld **/
        /** st **/
        /** add **/
        /** sub **/
        /** mul **/
        /** div **/
        /** mod **/
        /** and **/
        /** or  **/
        /** xor **/
        /**********/

/**method for performing load operation */
public static void load(int val){
        QRegisterFile.enqueue(val);
}
```

```
/** method for performing store operation */
public static void store(int value){
        int index=      QRegisterFile.getOffsetIndex(value);
        int storeValue=         QRegisterFile.getQHValue();
        QRegisterFile.setStoreValue(value);
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
'STORE' OPERATION  " +"\n----------------------------------\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("THE VALUE
"+storeValue+" WAS SUCCESSFULLY STORED AT INDEX "+index +"\n" );
}

/** method for performing addition operation*/
public static void addition(int offset){
        int src1=QRegisterFile.dequeue();
        int src2=QRegisterFile.getOffsetValue(offset);
        int sum=src1+src2;
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
'ADDITION' OPERATION  " +"\n----------------------------------\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF QH
[SRC1 ]= "+src1+"\n" );
        Globals.getGui().getResultsPane().getResultsTextArea().append("OFFSET VALUE=
"+offset +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF
QH+OFFSET[SRC2]= "+src2 +"\n");
        QRegisterFile.enqueue(sum);
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE  of THE
OPERATION: ADD " +offset+" = "+sum+"\n");
 }

/** method for performing subtraction operation */
```

```
public static void subtract(int offset){
        int src1=QRegisterFile.dequeue();
        int src2=QRegisterFile.getOffsetValue(offset);
        int diff=src1-src2;
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
'SUBTRACTION' OPERATION  " +"\n---------------------------------\n\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF QH
[SRC1 ]= "+src1+"\n\n" );
        Globals.getGui().getResultsPane().getResultsTextArea().append("OFFSET VALUE=
"+offset +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF
QH+OFFSET[SRC2]= "+src2 +"\n");
        QRegisterFile.enqueue(diff);
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE  of THE
OPERATION: SUB " +offset+" = "+diff+"\n");
  }

  /** method for performing multiplication operation*/
  public static void multiply(int offset){
        int src1=QRegisterFile.dequeue();
        int src2=QRegisterFile.getOffsetValue(offset);
        int prod=src1*src2;
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
'MULTIPLICATION' OPERATION  " +"\n---------------------------------\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF QH
[SRC1 ]= "+src1+"\n" );
        Globals.getGui().getResultsPane().getResultsTextArea().append("OFFSET VALUE=
"+offset +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF
QH+OFFSET[SRC2]= "+src2 +"\n");
        QRegisterFile.enqueue(prod);
```

```
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE  of THE
OPERATION: MUL " +offset+" = "+prod+"\n");
  }

  /** method for performing modulo arithmetic operation*/
  public static void modulo(int offset){
        int src1=QRegisterFile.dequeue();
        int src2=QRegisterFile.getOffsetValue(offset);
        if(src2==0){
        JOptionPane.showMessageDialog(null, " CANNOT DIVIDE BY ZERO:
UNSUCCESSFUL MODULO OPERATION !!!");
        }
        else{
        int mod=src1%src2;
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
'MODULO' OPERATION  " +"\n---------------------------------\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF QH
[SRC1 ]= "+src1+"\n" );
        Globals.getGui().getResultsPane().getResultsTextArea().append("OFFSET VALUE=
"+offset +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF
QH+OFFSET[SRC2]= "+src2 +"\n");
        QRegisterFile.enqueue(mod);
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE  of THE
OPERATION: MOD " +offset+" = "+mod+"\n");
        }
 }


/** method for performing division operation*/
```

```
public static void division(int offset){
        int src1=QRegisterFile.dequeue();
        int src2=QRegisterFile.getOffsetValue(offset);
        if(src2==0){
        JOptionPane.showMessageDialog(null, "CANNOT DIVIDE BY ZERO:
UNSUCCESSFUL DIVISION OPERATION !!!");
  }
        else{
        double div=src1/src2;
        QRegisterFile.enqueue((int)div);
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
'DIVISION' OPERATION " +"\n---------------------------------\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF QH
[SRC1 ]= "+src1+"\n" );
        Globals.getGui().getResultsPane().getResultsTextArea().append("OFFSET VALUE=
"+offset +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF
QH+OFFSET[SRC2]= "+src2 +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF THE
OPERATION: DIV " +offset+" = "+div+"\n");
        }
 }

/** method for performing bitwise AND operation*/
public static void and(int offset){
        int src1=QRegisterFile.dequeue();
        int src2=QRegisterFile.getOffsetValue(offset);
        int and=src1&src2;
        QRegisterFile.enqueue(and);
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
BITWISE 'AND' OPERATION " +"\n---------------------------------\n");
```

```
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF QH
[SRC1 ]= "+src1+"\n" );
        Globals.getGui().getResultsPane().getResultsTextArea().append("OFFSET VALUE=
"+offset +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF
QH+OFFSET[SRC2]= "+src2 +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE  OF THE
OPERATION: AND " +offset+" = "+and+"\n");
                        }

 /** method for performing bitwise OR operation*/
 public static void or(int offset){
        int src1=QRegisterFile.dequeue();
        int src2=QRegisterFile.getOffsetValue(offset);
        int or=src1|src2;
        QRegisterFile.enqueue(or);
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
BITWISE 'OR' OPERATION " +"\n----------------------------------\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF QH
[SRC1 ]= "+src1+"\n" );
        Globals.getGui().getResultsPane().getResultsTextArea().append("OFFSET VALUE=
"+offset +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF
QH+OFFSET[SRC2]= "+src2 +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE  OF THE
OPERATION: OR " +offset+" = "+or+"\n");
   }
```

```
/** method for performing bitwise XOR operation*/
 public static void xor(int offset){
        int src1=QRegisterFile.dequeue();
        int src2=QRegisterFile.getOffsetValue(offset);
        int xor=src1^src2;
        QRegisterFile.enqueue(xor);
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
BITWISE 'XOR' OPERATION " +\n----------------------------------\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF QH
[SRC1 ]= "+src1+"\n" );
        Globals.getGui().getResultsPane().getResultsTextArea().append("OFFSET VALUE=
"+offset +"\n");
                Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE
OF QH+OFFSET[SRC2]= "+src2 +"\n");
                Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE
OF THE OPERATION: XOR " +offset+" = "+xor+"\n");
}

/** method for performing bitwise NOT operation*/
public static void not(int offset){
        int src1=QRegisterFile.dequeue();
        int src2=QRegisterFile.getOffsetValue(offset);
        int not=~src2;
        QRegisterFile.enqueue(not);
        Globals.getGui().getResultsPane().getResultsTextArea().append("\n RESULS OF
BITWISE 'NOT' OPERATION " +"\n----------------------------------\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF QH
[SRC1 ]= "+src1+"\n" );
        Globals.getGui().getResultsPane().getResultsTextArea().append("OFFSET VALUE=
"+offset +"\n");
```

```
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE OF
QH+OFFSET[SRC2]= "+src2 +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("VALUE  OF THE
OPERATION: NOT " +offset+" = "+not+"\n");
        }
}
```

QSIM PACKAGE

QCprogram.JAVA CLASS

```
package qsim;
import java.util.*;
import java.io.*;
import qsim.registers.*;

/** Internal representations of QC program */
public class QCprogram {
        private static String filename;
        private static ArrayList sourceList;
        private ArrayList tokenList;
        private ArrayList parsedList;
        private ArrayList machineList;
        private String inst;


        /** Generate list of source statements that comprise the program.
         * @return ArrayList of String. Each String is one line of QC source code.
         **/

        public static ArrayList getSourceList() {
                return sourceList;
        }

        /** Produces name of associated source code file.
         * @return File name as String.
         **/

        public static String getFilename() {
```

```
            return filename;
    }
    /**the actual name of the file from the entire filename */
    public static String getNameOfFIle(){
            File tempFile= new File(filename);
            String file=tempFile.getName();
            return file;
    }

    /** Produces specified line of QC source program.
     * @param n:Line number of QC source program to get. Line 1 is first line.
     * @return Returns specified line of QC source. If outside the line range,
     it returns null. Line 1 is first line.
     **/

    public String getSourceLine(int n) {
            if ((n >= 1) && (n <= sourceList.size()))
                    return (String) sourceList.get(n - 1);
            else
                    return null;
    }

    /** Reads QC source code from file into structure. Will always read from
     * file.
     * @param file: String containing name of QC source code file **/
    public void readSource(String file) throws Exception {
            QCprogram.filename = file;
            QCprogram.sourceList = new ArrayList();
            BufferedReader inputFile;
            String line;
            try {
```

```
                inputFile = new BufferedReader(new FileReader(file));
                line = inputFile.readLine();
                while (line != null) {
                        sourceList.add(line);
                        line = inputFile.readLine();
                }
        }
        catch (Exception e) {
        }
        for(int i=0;i<sourceList.size();i++){
                inst=(String)QCprogram.getSourceList().get(i);
                getLoadQreg(inst);
        }
        QRegisterFile.showLoadQReg();
        return;
    }

    /** Method to decode load instruction */
    public static void getLoadQreg(String str){
        String [] myList=str.split(" ");
        String opcode=myList[0];

        if(opcode.equalsIgnoreCase("ld")){
                String soperand=myList[1];
                int operand=Integer.parseInt(soperand);
                QInstructionSet.load(operand);
        }
    }
}
```

QsimLaunch.JAVA CLASS

```
package qsim;
import javax.swing.*;
import qsim.gui.*;

 /** Launch the Qsim application */
public class QsimLaunch{

        /** time in MS to show splash screen*/
        private static final int splashDuration = 5000;



        public QsimLaunch(String[] args) {
                boolean gui = (args.length == 0);
                Globals.initialize(gui);
                if (gui) {
                        launchIDE();
                }
        }

        /** launching  of the GUI-fronted integrated development environment*/
        private void launchIDE() {
                new QsimSplashScreen(splashDuration).showSplash();
                SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                                new UserGui("QSIM " + Globals.version);
                        }
                });
                return;
        }
}
```

QsimSplashScreen.JAVA CLASS

```
package qsim;
import java.awt.*;
import javax.swing.*;
import javax.swing.border.BevelBorder;

/** Produces QSIM splash screen */
public class QsimSplashScreen extends JWindow {
	private static final long serialVersionUID = 1L;
	private int duration;

	public QsimSplashScreen(int time) {
		duration = time;
	}

	/** The method to show a small window in the center of the screen
	 * for the amount of time given in the constructor
	 */
	public void showSplash() {
		Color bgColor = Color.black;
		Color fgColor = Color.YELLOW;
		JPanel content;
		JPanel mainpan,N_panel,S_panel,E_panel,W_panel;
		JLabel N_label,S_label;

		/** Set the window's bounds, centering the window*/
		int width = 500;
		int height = 400;
		Toolkit tk = Toolkit.getDefaultToolkit();
		Dimension screen = tk.getScreenSize();
		int x = (screen.width - width) / 2;
```

```
int y = (screen.height - height) / 2;
setBounds(x, y, width, height);

/** defining external borders for window*/
mainpan=new JPanel();
mainpan.setLayout(new BorderLayout());
mainpan.setBackground(bgColor);
content = (JPanel) getContentPane();
content.setLayout(new BorderLayout());
content.setBackground(bgColor);

N_panel=new JPanel();
N_panel.setBackground(bgColor);
S_panel=new JPanel();
S_panel.setBackground(bgColor);
E_panel=new JPanel();
E_panel.setBackground(bgColor);
W_panel=new JPanel();
W_panel.setBackground(bgColor);

E_panel.setBorder(new BevelBorder(BevelBorder.RAISED));
W_panel.setBorder(new BevelBorder(BevelBorder.RAISED));

content.add(N_panel, BorderLayout.NORTH);
content.add(S_panel, BorderLayout.SOUTH);
content.add(E_panel, BorderLayout.EAST);
content.add(W_panel, BorderLayout.WEST);

JPanel titles = new JPanel(new GridLayout(2, 1));
JPanel copyrights = new JPanel(new GridLayout(3, 1));
```

```
JLabel picture = new JLabel(new ImageIcon(tk.getImage(this.getClass()
                .getResource(Globals.imagesPath +QSIMThumbnail_Ani.gif"))));
JLabel title = new JLabel("QSIM: QUEUE-CORE RUNTIME SIMULATOR",
                JLabel.CENTER);
JLabel copyrt1 = new JLabel("Version " + Globals.version
                + " Copyright (c) " + Globals.copyrightYears, JLabel.CENTER);
JLabel copyrt2 = new JLabel(Globals.copyrightHolders, JLabel.CENTER);

N_label= new JLabel(new ImageIcon(tk.getImage(this.getClass()
                .getResource(Globals.imagesPath + "N_label_splash.gif"))));
S_label= new JLabel(new ImageIcon(tk.getImage(this.getClass()
                .getResource(Globals.imagesPath + "S_label_splash.gif"))));

N_panel.add(N_label);
S_panel.add(S_label);

copyrt1.setFont(new Font("Sans-Serif", Font.BOLD, 12));
copyrt2.setFont(new Font("Sans-Serif", Font.BOLD, 12));
copyrt1.setForeground(fgColor);
copyrt2.setForeground(fgColor);
title.setFont(new Font("Sans-Serif", Font.BOLD, 16));
title.setForeground(fgColor);
titles.add(new JLabel(" "));
titles.add(title);
titles.setBackground(bgColor);
mainpan.add(titles, BorderLayout.NORTH);
mainpan.add(picture, BorderLayout.CENTER);

copyrights.setBackground(bgColor);
copyrights.add(copyrt1);
copyrights.add(copyrt2);
```

```
            copyrights.add(new JLabel(" "));
            mainpan.add(copyrights, BorderLayout.SOUTH);
            content.add(mainpan,BorderLayout.CENTER);
            Color oraRed = new Color(255, 55,100);
            content.setBorder(BorderFactory.createLineBorder(oraRed, 7));
            setVisible(true);

            /** Wait a little while, maybe while loading resources*/
            try {
                  Thread.sleep(duration);
            } catch (Exception e) {
            }
            setVisible(false);
      }
}
```

SimulateQCprogram.JAVA CLASS

```
package qsim;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import javax.swing.JOptionPane;
import qsim.registers.*;

/*************************************************************************/
/** class to fetch, decode, compute operand, and write into QRegister*/
/*************************************************************************/

public class SimulateQCprogram {
        static String code;
        static double aluNumber=0;
        static double loadNumber=0;
        static double storeNumber=0;
        static double branchNumber=0;
        static double totatInstNumber=0;
        static double codeSizeInByte=0;
        static double aluPercent=0.0;
        static double loadPercent=0.0;
        static double storePercent=0.0;
        static double branchPercent=0.0;
        static DecimalFormat fm=new DecimalFormat("#.##");

        /**Method to fetch Queue Core instructions for decode */
        public static  void simulateprogram(){
                displayHeader();
                for(int i=0;i<QCprogram.getSourceList().size();i++){
                        code=(String)QCprogram.getSourceList().get(i);
```

```
                executeProgram(code);
            }
        }

        /**Method to decode the fetched instructions  */
        public static  void executeProgram(String str){
        String [] myList=str.split(" ");
        String opcode=myList[0];

        if(opcode.equalsIgnoreCase("add")){
                aluNumber++;
                int operand=Integer.parseInt(myList[1]);
                QInstructionSet.addition(operand);
        }
        else if(opcode.equalsIgnoreCase("sub")){
                aluNumber++;
                int operand=Integer.parseInt(myList[1]);
                QInstructionSet.subtract(operand);
        }
        else if(opcode.equalsIgnoreCase("mul")){
                aluNumber++;
                int operand=Integer.parseInt(myList[1]);
                QInstructionSet.multiply(operand);
        }

        else if(opcode.equalsIgnoreCase("mod")){
                aluNumber++;
                int operand=Integer.parseInt(myList[1]);
                QInstructionSet.modulo(operand);
        }
```

```
else if(opcode.equalsIgnoreCase("div")){
        aluNumber++;
        int operand=Integer.parseInt(myList[1]);
        QInstructionSet.division(operand);
}
else if(opcode.equalsIgnoreCase("ld")){
        loadNumber++;
}

else if(opcode.equalsIgnoreCase("st")){
        storeNumber++;
        int operand=Integer.parseInt(myList[1]);
        QInstructionSet.store(operand);
}
else if(opcode.equalsIgnoreCase("and")){
        aluNumber++;
        int operand=Integer.parseInt(myList[1]);
        QInstructionSet.and(operand);
}

else if(opcode.equalsIgnoreCase("or")){
        aluNumber++;
        int operand=Integer.parseInt(myList[1]);
        QInstructionSet.or(operand);
}

else if(opcode.equalsIgnoreCase("xor")){
        aluNumber++;
        int operand=Integer.parseInt(myList[1]);
        QInstructionSet.xor(operand);
}
```

```
        else if(opcode.equalsIgnoreCase("not")){
                aluNumber++;
                int operand=Integer.parseInt(myList[1]);
                QInstructionSet.not(operand);
        }

        else if(opcode.equalsIgnoreCase("b")){
                JOptionPane.showMessageDialog(null, "BRANCH INSTRUCTION IS NOT
IMPLEMENTED IN THE INSTRUCTION SET");
        }

        else if(opcode.equalsIgnoreCase("jump")){
                JOptionPane.showMessageDialog(null, "JUMP INSTRUCTION IS NOT
IMPLEMENTED IN THE INSTRUCTION SET");
        }
        else{
                JOptionPane.showMessageDialog(null, "SOURCE PROGRAM ( "+code+ " )
NOT RECOGNIZE BY QSIM INSTRUCTION SET" );
        }
        }

        /** getter methods for Number of ALU instructions */
        public static double getAluNumber(){
                return aluNumber;
        }

        /** getter methods for computing percentage of ALU instructions */
        public static double getAluPercent(){
                double aluPercent=(getAluNumber()/getTotalInstNumber())*100;
                return aluPercent;
```

```
}

/** getter method for Number of Load instructions*/
public static double getLoadNumber(){
        return loadNumber;
}

/** getter methods for computing percentage of LOAD instructions */
public static double getLoadPercent(){
        double loadPercent=(getLoadNumber()/getTotalInstNumber())*100;
        return loadPercent;
}


/** getter method for Number of Store instructions*/
public static double getStoreNumber(){
        return storeNumber;
}

/** getter methods for computing percentage of STORE instructions */
public static double getStorePercent(){
        double storePercent=(getStoreNumber()/getTotalInstNumber())*100;
        return storePercent;
}

/** getter method for Number of Branch instructions*/
public static double getBranchNumber(){
        return branchNumber;
}
```

```
        /** getter methods for computing percentage of BRANCH instructions */
        public static double getBranchPercent(){
                double branchPercent=(getBranchNumber()/getTotalInstNumber())*100;
                return branchPercent;
        }

        /**  method computing the Total Number QSIM Executed instructions*/
        public static double getTotalInstNumber(){
                totatInstNumber=QCprogram.getSourceList().size();
                return totatInstNumber;
        }

        /** method computing total code size in byte for the Total Number QSIM Executed
instructions*/
        public static double getCodeSizeInByte(){
                codeSizeInByte=getTotalInstNumber()*2;
                return codeSizeInByte;
        }

        /** method to find simulation time*/
        public static final String DATE_FORMAT = "HH:mm:ss"; //declaring time format
        public static String getSimTime() {
                Calendar cal = Calendar.getInstance(); //creating instance of calendar class
                SimpleDateFormat DF = new SimpleDateFormat(DATE_FORMAT);
         String mytime=DF.format(cal.getTime());    //extract time from calendar class
                return mytime;
        }
```

```
/** method to display header to simulation results */
public static void displayHeader(){
        Globals.getGui().getResultsPane().getResultsTextArea().append("****************\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("QSIM
COMPUTATION RESULTS \n ");
        Globals.getGui().getResultsPane().getResultsTextArea().append("CURRENT
PROGRAMMING RUNNING: "+QCprogram.getNameOfFIle() +"\n");
        Globals.getGui().getResultsPane().getResultsTextArea().append("************\n");
}

/** method to display simulation statistics*/
public static void simStatistics(){
        Globals.getGui().getResultsPane().getStatsTextArea().append("\n QUEUE
SIMULATION STATISTICS FOR THE PROGRAM** "+QCprogram.getNameOfFIle()+
"**\n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("------------------ \n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("SIMULATION TIME:
"+getSimTime()+"  \n\n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("TOTAL NUMBER OF
EXECUTED INSTRUCTIONS : "+getTotalInstNumber()+" INSTRUCTIONS \n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("TOTAL NUMBER OF
LOAD INSTRUCTIONS : "+(int)getLoadNumber()+" LOAD INSTRUCTION (S) \n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("PERCENTAGE OF
LOAD INSTRUCTIONS : "+fm.format(getLoadPercent())+" %\n\n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("TOTAL NUMBER OF
STORE INSTRUCTIONS : "+(int)getStoreNumber()+" STORE INSTRUCTION (S) \n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("PERCENTAGE OF
STORE INSTRUCTIONS : "+fm.format(getStorePercent())+" %\n\n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("TOTAL NUMBER OF
ALU INSTRUCTIONS : "+(int)getAluNumber()+" ALU INSTRUCTION(S) \n");
```

```
        Globals.getGui().getResultsPane().getStatsTextArea().append("PERCENTAGE OF ALU
INSTRUCTIONS : "+fm.format(getAluPercent())+" % \n\n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("TOTAL NUMBER OF
BRANCH INSTRUCTIONS : "+(int)getBranchNumber()+" BRANCH INSTRUCTION (S)
\n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("PERCENTAGE OF
BRANCH INSTRUCTIONS : "+fm.format(getBranchPercent())+" %\n");
        Globals.getGui().getResultsPane().getStatsTextArea().append("TOTAL NUMBER OF
EXECUTED CODE SIZE IN BYTE : "+(int)getCodeSizeInByte()+" BYTES\n");
        }
}
```

QSIM.GUI PACKAGE

UserGui.JAVA CLASS

```
package qsim.gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import qsim.*;
import java.net.*;

/**  Top level container for UserGUI **/
public class UserGui extends JFrame{
        private static final long serialVersionUID = 1L;
        UserGui qsimGui;
        JMenuBar menu;
        JToolBar toolbar;
        QsimMainWindow qsimMainWindow;
        QRegisterPane qRegisterPane;
        QRegistersWindow registersTab;
        FloatingPointWindow floatingPointTab;
        QsimResultsPane qsimResultsPane;
        JSplitPane splitter, horizonSplitter;
        JPanel north;
        JLabel Qlabel,NS_label,SN_label,SE_label,ES_label;
        JPanel main_N,main_S,main_E,main_W;
        JPanel main_N1,main_S1,main_E1,main_W1;

        private static int menuState = FileStatus.NO_FILE;
        private static boolean reset= true;
        private static boolean started = false;
        Editor editor;
```

```
/** components of the menu bar*/
private JMenu file, simulator, help, edit, settings;
private JMenuItem fileNew, fileOpen, fileClose, fileSave, fileSaveAs, filePrint, fileExit;
private JMenuItem editUndo, editRedo, editCut, editCopy, editPaste, editFindReplace;
private JMenuItem simulatorRun, simulatorStep, simulatorBackstep, simulatorReset,
simulatorStop, simulatorPause;
private JMenuItem  settingsEditor;
private JMenuItem  helpAbout;

/** components of the tool bar */
private JButton Undo, Redo, Cut, Copy, Paste, FindReplace;
private JButton New, Open, Save, SaveAs, Print;
private JButton Run, Reset, Step, Backstep, Stop, Pause;
private Action fileNewAction, fileOpenAction, fileCloseAction, fileSaveAction;
private Action fileSaveAsAction, filePrintAction, fileExitAction;
EditUndoAction editUndoAction;
EditRedoAction editRedoAction;
private Action editCutAction, editCopyAction, editPasteAction, editFindReplaceAction;
private Action simulatorRunAction, simulatorStepAction, simulatorBackstepAction,
simulatorResetAction,
simulatorStopAction, simulatorPauseAction;
private Action          settingsEditorAction, settingsHighlightingAction;
private Action helpAboutAction;
```

```
        /** Constructor for the Class. Sets up a window object for the UserGui
                @param name Name of the window to be created.
         **/

        public UserGui(String name) {
                super(name);
                qsimGui = this;
                Globals.setGui(this);
                double screenWidth  = Toolkit.getDefaultToolkit().getScreenSize().getWidth();
                double screenHeight = Toolkit.getDefaultToolkit().getScreenSize().getHeight();

                double registersWidthPct = (screenWidth<1000.0)? 0.16 : 0.20;
                double registersHeightPct = (screenWidth<1000.0)? 0.70 : 0.78;

                Dimension registersPanePreferredSize = new
Dimension((int)(screenWidth*registersWidthPct),(int)(screenHeight*registersHeightPct));
                Globals.initialize(true);

                /**  image designed by Aziz (August, 2010.) */
                URL im = this.getClass().getResource(Globals.imagesPath+"QSIM16.gif");
                if (im == null) {
                        System.out.println("Internal Error: images folder or file not found");
                        System.exit(0);
                }
                Image qsim = Toolkit.getDefaultToolkit().getImage(im);
                this.setIconImage(qsim);
                registersTab = new QRegistersWindow();
                floatingPointTab = new FloatingPointWindow();
                Toolkit tk = Toolkit.getDefaultToolkit();
      Qlabel= new JLabel(new ImageIcon(tk.getImage(this.getClass()
                .getResource(Globals.imagesPath + "label.gif"))));
```

```
NS_label= new JLabel(new ImageIcon(tk.getImage(this.getClass()
            .getResource(Globals.imagesPath + "NS_lable.gif"))));

SN_label= new JLabel(new ImageIcon(tk.getImage(this.getClass()
                        .getResource(Globals.imagesPath + "NS_lable.gif"))));

SE_label= new JLabel(new ImageIcon(tk.getImage(this.getClass()
                        .getResource(Globals.imagesPath + "SE_label.gif"))));

ES_label= new JLabel(new ImageIcon(tk.getImage(this.getClass()
                        .getResource(Globals.imagesPath + "SE_label.gif"))));

qRegisterPane = new QRegisterPane(qsimGui, registersTab,floatingPointTab);
qRegisterPane.setPreferredSize(registersPanePreferredSize);
qsimMainWindow= new QsimMainWindow(qsimGui, registersTab, floatingPointTab);
qsimResultsPane= new QsimResultsPane();
splitter= new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, qsimMainWindow,
qsimResultsPane);
splitter.setOneTouchExpandable(true);
splitter.resetToPreferredSizes();
splitter.setBackground(Color.BLACK);
horizonSplitter = newJSplitPane(JSplitPane.HORIZONTAL_SPLIT,qRegisterPane,splitter );
horizonSplitter.setBackground(Color.BLACK);
horizonSplitter.setOneTouchExpandable(true);
horizonSplitter.resetToPreferredSizes();

 /** set up menu/tool bar */
this.createActionObjects();
menu= this.setUpMenuBar();
this.setJMenuBar(menu);
```

```
toolbar= this.setUpToolBar();
JPanel northpan = new JPanel(new FlowLayout(FlowLayout.LEFT));
northpan.add(toolbar);
northpan.add(Qlabel);
JPanel center= new JPanel(new BorderLayout());
center.add(northpan, BorderLayout.NORTH);
center.add(horizonSplitter);
Color myColor=new Color(102,0,205);
Color color=new Color(132,255,0);
main_N=new JPanel();
main_S=new JPanel();
main_E=new JPanel();
main_W=new JPanel();

main_N.setLayout(new BorderLayout());
main_S.setLayout(new BorderLayout());
main_E.setLayout(new BorderLayout());
main_W.setLayout(new BorderLayout());

main_N1=new JPanel();
main_S1=new JPanel();
main_E1=new JPanel();
main_W1=new JPanel();

int dim=3;
main_N1.setBorder(BorderFactory.createBevelBorder(5));      // set border for table
main_N1.setBorder(BorderFactory.createLineBorder(color, dim)); // set line color for
border
main_S1.setBorder(BorderFactory.createBevelBorder(5));              // set border for table
main_S1.setBorder(BorderFactory.createLineBorder(color, dim)); // set line color for
border
```

```
        main_E1.setBorder(BorderFactory.createBevelBorder(5));              // set border for table
        main_E1.setBorder(BorderFactory.createLineBorder(color, dim)); // set line color for
border
        main_W1.setBorder(BorderFactory.createBevelBorder(5));              // set border for table
        main_W1.setBorder(BorderFactory.createLineBorder(color, dim)); // set line color for
border

        main_N.setBackground(Color.BLACK);
        main_S.setBackground(Color.BLACK);
        main_E.setBackground(Color.BLACK);
        main_W.setBackground(Color.BLACK);

        main_N1.setBackground(myColor);
        main_S1.setBackground(myColor);
        main_E1.setBackground(myColor);
        main_W1.setBackground(myColor);

        main_N.add(NS_label, BorderLayout.NORTH);
        main_N.add(main_N1, BorderLayout.SOUTH);
        main_S.add(SN_label, BorderLayout.SOUTH);
        main_S.add(main_S1, BorderLayout.NORTH);
        main_E.add(SE_label, BorderLayout.EAST);
        main_E.add(main_E1, BorderLayout.WEST);
        main_W.add(ES_label, BorderLayout.WEST);
        main_W.add(main_W1, BorderLayout.EAST);

        this.getContentPane().add(center);
        this.getContentPane().add(main_N,BorderLayout.NORTH);
        this.getContentPane().add(main_S,BorderLayout.SOUTH);
        this.getContentPane().add(main_E,BorderLayout.EAST);
        this.getContentPane().add(main_W,BorderLayout.WEST);
```

```
FileStatus.reset();
FileStatus.set(FileStatus.NO_FILE);

/**  It will set the application to appear at full screen size */
this.addWindowListener(
        new WindowAdapter() {
            public void windowOpened(WindowEvent e) {
                qsimGui.setExtendedState(JFrame.MAXIMIZED_BOTH);
            }
        });

/** called when  application is close through the X icon.
   it checks for unsaved edits before exiting
 */
this.addWindowListener(
        new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                if (qsimGui.editor.editsSavedOrAbandoned()) {
                    System.exit(0);
                }
            }
        });

/** This handles windowClosing event in the
   situation where user Cancels out of "save edits?" dialog.
 */
this.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
this.pack();
this.setVisible(true);
   }
```

```
    /**Action objects are used instead of action listeners because one
     * can be easily shared between a menu item and a toolbar button */
        private void createActionObjects() {
                Toolkit tk = Toolkit.getDefaultToolkit();
                Class cs = this.getClass();
                try {
                        fileNewAction = new FileNewAction("New",
                                        new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"New22.png"))),
                                        "Create a new QC file for editing", new
Integer(KeyEvent.VK_N),
                                        KeyStroke.getKeyStroke( KeyEvent.VK_N,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
                                        qsimGui);
                        fileOpenAction = new FileOpenAction("Open ...",
                                        new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Open22.png"))),
                                        "Open a QC file for editing", new
Integer(KeyEvent.VK_O),
                                        KeyStroke.getKeyStroke( KeyEvent.VK_O,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
                                        qsimGui);
                        fileCloseAction = new FileCloseAction("Close", null,
                                        "Close the current QC file", new Integer(KeyEvent.VK_C),
                                        KeyStroke.getKeyStroke( KeyEvent.VK_W,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
                                        qsimGui);
                        fileSaveAction = new FileSaveAction("Save",
                                        new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"save22.png"))),
```

```
                                        "Save the current QC file", new Integer(KeyEvent.VK_S),
                                        KeyStroke.getKeyStroke( KeyEvent.VK_S,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
                                        qsimGui);
                        fileSaveAsAction = new FileSaveAsAction("Save As ...",
                                        new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"saveAs22.png"))),
                                        "Save current QC file with different name", new
Integer(KeyEvent.VK_A),
                                        null, qsimGui);
                        filePrintAction = new FilePrintAction("Print ...",
                                        new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Print22.gif"))),
                                        "Print current QC file", new Integer(KeyEvent.VK_P),
                                        null, qsimGui);
                        fileExitAction = new FileExitAction("Exit", null,
                                        "Exit Qsim", new Integer(KeyEvent.VK_X),
                                        null, qsimGui);
                        editUndoAction = new EditUndoAction("Undo",
                                        new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Undo22.png"))),
                                        "Undo last edit", new Integer(KeyEvent.VK_U),
                                        KeyStroke.getKeyStroke( KeyEvent.VK_Z,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
                                        qsimGui);
                        editRedoAction = new EditRedoAction("Redo",
                                        new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Redo22.png"))),
                                        "Redo last edit", new Integer(KeyEvent.VK_R),
                                        KeyStroke.getKeyStroke( KeyEvent.VK_Y,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
```

```
                                        qsimGui);
                        editCutAction = new EditCutAction("Cut",
                                        new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Cut22.gif"))),
                                        "Cut", new Integer(KeyEvent.VK_C),
                                        KeyStroke.getKeyStroke( KeyEvent.VK_X,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
                                        qsimGui);
                        editCopyAction = new EditCopyAction("Copy",
                                        new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Copy22.png"))),
                                        "Copy", new Integer(KeyEvent.VK_O),
                                        KeyStroke.getKeyStroke( KeyEvent.VK_C,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
                                        qsimGui);
                editPasteAction = new EditPasteAction("Paste",
                                new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Paste22.png"))),
                                "Paste", new Integer(KeyEvent.VK_P),
                                KeyStroke.getKeyStroke( KeyEvent.VK_V,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
                                qsimGui);
                editFindReplaceAction = new EditFindReplaceAction("Find/Replace",
                                new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Find22.png"))),
                                "Find/Replace", new Integer(KeyEvent.VK_F),
                                KeyStroke.getKeyStroke( KeyEvent.VK_F,
Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()),
                                qsimGui);
                simulatorRunAction = new SimulatorRunAction("Run",
```

```
                    new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Play22.png"))),
                    "Run the current Qsim program", new Integer(KeyEvent.VK_G),
                    KeyStroke.getKeyStroke( KeyEvent.VK_F5, 0),
                    qsimGui);
        simulatorStepAction = new SimulatorStepAction("StepForward",
                    new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"StepForward22.png"))),
                    "Run Program one step at a time", new Integer(KeyEvent.VK_T),
                    KeyStroke.getKeyStroke( KeyEvent.VK_F7, 0),
                    qsimGui);
        simulatorBackstepAction = new SimulatorBackstepAction("Backstep",
                    new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"StepBack22.png"))),
                    "Undo the last step", new Integer(KeyEvent.VK_B),
                    KeyStroke.getKeyStroke( KeyEvent.VK_F8, 0),
                    qsimGui);
        simulatorPauseAction = new SimulatorPauseAction("Pause",
                    new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Pause22.png"))),
                    "Pause the currently simulatorning program", new
Integer(KeyEvent.VK_P),
                    KeyStroke.getKeyStroke( KeyEvent.VK_F9, 0),
                    qsimGui);
        simulatorStopAction = new SimulatorStopAction("Stop",
                    new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Stop22.png"))),
                    "Stop the currently running program", new Integer(KeyEvent.VK_S),
                    KeyStroke.getKeyStroke( KeyEvent.VK_F11, 0),
                    qsimGui);
        simulatorResetAction = new SimulatorResetAction("Reset",
```

```
                    new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Reset22.png"))),
                    "Reset QC qsimMemory and registers", new Integer(KeyEvent.VK_R),
                    KeyStroke.getKeyStroke( KeyEvent.VK_F12,0),
                    qsimGui);
        settingsEditorAction        = new SettingsEditorAction("Editor...",
                    null,
                    "View and modify text editor settings.",
                    null,null,
                    qsimGui);
        helpAboutAction = new QSIMHelpAction("About QSIM ...",null,
                    "Information about QSIM", null,null, qsimGui);
            }
            catch (NullPointerException e) {
                    System.out.println("Internal Error: images folder not found, or other null
pointer exception while creating Action objects");
                    e.printStackTrace();
                    System.exit(0);
            }
        }


    /** build the menus and connect them to action objects */

        private JMenuBar setUpMenuBar() {
            Toolkit tk = Toolkit.getDefaultToolkit();
            Class cs = this.getClass();
            JMenuBar menuBar = new JMenuBar();
            file=new JMenu("File");
            file.setMnemonic(KeyEvent.VK_F);
            edit = new JMenu("Edit");
```

```
                edit.setMnemonic(KeyEvent.VK_E);
                simulator=new JMenu("Simulator");
                simulator.setMnemonic(KeyEvent.VK_R);
                settings = new JMenu("Settings");
                settings.setMnemonic(KeyEvent.VK_S);
                help = new JMenu("Help");
                help.setMnemonic(KeyEvent.VK_H);

                fileNew = new JMenuItem(fileNewAction);
                fileNew.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"New16.png"))));
                fileOpen = new JMenuItem(fileOpenAction);
                fileOpen.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Open16.png"))));
                fileClose = new JMenuItem(fileCloseAction);
                fileClose.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"MyBlank16.gif"))));
                fileSave = new JMenuItem(fileSaveAction);
                fileSave.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"save16.png"))));
                fileSaveAs = new JMenuItem(fileSaveAsAction);
                fileSaveAs.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"saveAs16.png"))));
                filePrint = new JMenuItem(filePrintAction);
                filePrint.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Print16.gif"))));
                fileExit = new JMenuItem(fileExitAction);
                fileExit.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"MyBlank16.gif"))));
                file.add(fileNew);
                file.add(fileOpen);
```

```
                file.add(fileClose);
                file.addSeparator();
                file.addSeparator();
                file.add(fileSave);
                file.add(fileSaveAs);

                file.addSeparator();
                file.addSeparator();
                file.add(filePrint);
                file.addSeparator();
                file.addSeparator();
                file.add(fileExit);

                editUndo = new JMenuItem(editUndoAction);
                editUndo.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Undo16.png"))));//"Undo16.gif"))
));
                editRedo = new JMenuItem(editRedoAction);
                editRedo.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Redo16.png"))));//"Redo16.gif")))
);
                editCut = new JMenuItem(editCutAction);
                editCut.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Cut16.gif"))));
                editCopy = new JMenuItem(editCopyAction);
                editCopy.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Copy16.png"))));//"Copy16.gif")))
);
                editPaste = new JMenuItem(editPasteAction);
```

```
                editPaste.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Paste16.png"))));//"Paste16.gif")))
);
                editFindReplace = new JMenuItem(editFindReplaceAction);
                editFindReplace.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Find16.png"))));//"Paste16.gif"))))
;
                edit.add(editUndo);
                edit.add(editRedo);
                edit.addSeparator();
                edit.addSeparator();
                edit.add(editCut);
                edit.add(editCopy);
                edit.add(editPaste);
                edit.addSeparator();
                edit.addSeparator();
                edit.add(editFindReplace);

                simulatorRun = new JMenuItem(simulatorRunAction);
                simulatorRun.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Play16.png"))));//"Play16.gif"))));
                simulatorStep = new JMenuItem(simulatorStepAction);
                simulatorStep.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"StepForward16.png"))));//"MyStepForward16.gif"))));
                simulatorBackstep = new JMenuItem(simulatorBackstepAction);
                simulatorBackstep.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"StepBack16.png"))));//"MyStepBack16.gif"))));
                simulatorReset = new JMenuItem(simulatorResetAction);
```

```
                simulatorReset.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Reset16.png"))));//"MyReset16.gi
f"))));
                simulatorStop = new JMenuItem(simulatorStopAction);
                simulatorStop.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Stop16.png"))));//"Stop16.gif"))));
                simulatorPause = new JMenuItem(simulatorPauseAction);
                simulatorPause.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"Pause16.png"))));//"Pause16.gif"))
));
                simulator.addSeparator();
                simulator.add(simulatorRun);
                simulator.add(simulatorStop);
                simulator.add(simulatorPause);
                simulator.add(simulatorStep);
                simulator.add(simulatorBackstep);
                simulator.add(simulatorReset);
                simulator.addSeparator();
                simulator.addSeparator();

                settingsEditor = new JMenuItem(settingsEditorAction);
                new JMenuItem(settingsHighlightingAction);
                settings.addSeparator();
                settings.add(settingsEditor);
                settings.addSeparator();
                helpAbout = new JMenuItem(helpAboutAction);
                helpAbout.setIcon(new
ImageIcon(tk.getImage(cs.getResource(Globals.imagesPath+"MyBlank16.gif"))));
                help.add(helpAbout);

                /** add menu items to menu bar */
```

```
            menuBar.add(file);
            menuBar.add(edit);
            menuBar.add(simulator);
            menuBar.add(settings);
            menuBar.add(help);
            return menuBar;
        }


    /** build the toolbar and connect items to action objects */

        JToolBar setUpToolBar() {
            JToolBar toolBar = new JToolBar();
            New = new JButton(fileNewAction);
            New.setText("");
            Open = new JButton(fileOpenAction);
            Open.setText("");
            Save = new JButton(fileSaveAction);
            Save.setText("");
            SaveAs = new JButton(fileSaveAsAction);
            SaveAs.setText("");
            Print= new JButton(filePrintAction);
            Print.setText("");
            Undo = new JButton(editUndoAction);
            Undo.setText("");
            Redo = new JButton(editRedoAction);
            Redo.setText("");
            Cut= new JButton(editCutAction);
            Cut.setText("");
            Copy= new JButton(editCopyAction);
            Copy.setText("");
```

```
Paste= new JButton(editPasteAction);
Paste.setText("");
FindReplace = new JButton(editFindReplaceAction);
FindReplace.setText("");
Run = new JButton(simulatorRunAction);
Run.setText("");
Step = new JButton(simulatorStepAction);
Step.setText("");
Backstep = new JButton(simulatorBackstepAction);
Backstep.setText("");
Reset = new JButton(simulatorResetAction);
Reset.setText("");
Stop = new JButton(simulatorStopAction);
Stop.setText("");
Pause = new JButton(simulatorPauseAction);
Pause.setText("");

/**add components to toolbar */
toolBar.add(New);
toolBar.add(Open);
toolBar.add(Save);
toolBar.add(SaveAs);
toolBar.add(Print);
toolBar.add(new JToolBar.Separator());
toolBar.add(Undo);
toolBar.add(Redo);
toolBar.add(Cut);
toolBar.add(Copy);
toolBar.add(Paste);
toolBar.add(FindReplace);
toolBar.add(new JToolBar.Separator());
```

```
            toolBar.add(Run);
            toolBar.add(Step);
            toolBar.add(Backstep);
            toolBar.add(Pause);
            toolBar.add(Stop);
            toolBar.add(Reset);
            toolBar.add(new JToolBar.Separator());
            toolBar.add(new JToolBar.Separator());
            return toolBar;
        }


    /** Determine the menu state from FileStatus (enabled/disabled).

     *Current states are:
     * setMenuStateInitial: set upon startup and after File->Close
     * setMenuStateEditingNew: set upon File->New
     * setMenuStateEditing: set upon File->Open or File->Save or erroneous Run->Assemble
     * setMenuStateRunning: set upon Simulator->Run
     */
      void setMenuState(int status) {
        menuState = status;
        switch (status) {
          case FileStatus.NO_FILE:
            setMenuStateInitial();
            break;
          case FileStatus.NEW_NOT_EDITED:
            setMenuStateEditingNew();
            break;
          case FileStatus.NEW_EDITED:
            setMenuStateEditingNew();
```

```
            break;
        case FileStatus.NOT_EDITED:
            setMenuStateEditing();
            break;
        case FileStatus.EDITED:
            setMenuStateEditing();
            break;
        case FileStatus.RUNNING:
            setMenuStateRunning();
            break;
        default:
            System.out.println("Invalid File Status: "+status);
            break;
    }
}


void setMenuStateInitial() {                    // set initial state of the menu
        fileNewAction.setEnabled(true);
        fileOpenAction.setEnabled(true);
        fileCloseAction.setEnabled(false);
        fileSaveAction.setEnabled(false);
        fileSaveAsAction.setEnabled(false);
        filePrintAction.setEnabled(false);
        fileExitAction.setEnabled(true);
        editUndoAction.setEnabled(false);
        editRedoAction.setEnabled(false);
        editCutAction.setEnabled(false);
        editCopyAction.setEnabled(false);
        editPasteAction.setEnabled(false);
        editFindReplaceAction.setEnabled(false);
```

```
        simulatorRunAction.setEnabled(false);
        simulatorStepAction.setEnabled(false);
        simulatorBackstepAction.setEnabled(false);
        simulatorResetAction.setEnabled(false);
        simulatorStopAction.setEnabled(false);
        simulatorPauseAction.setEnabled(false);
        helpAboutAction.setEnabled(true);
        editUndoAction.updateUndoState();
        editRedoAction.updateRedoState();
    }

    void setMenuStateEditing() {                                    // file open action
        /** undo and redo are handled separately by the undo manager*/
        fileNewAction.setEnabled(true);
        fileOpenAction.setEnabled(true);
        fileCloseAction.setEnabled(true);
        fileSaveAction.setEnabled(true);
        fileSaveAsAction.setEnabled(true);
        filePrintAction.setEnabled(true);
        fileExitAction.setEnabled(true);
        editCutAction.setEnabled(true);
        editCopyAction.setEnabled(true);
        editPasteAction.setEnabled(true);
        editFindReplaceAction.setEnabled(true);
        simulatorRunAction.setEnabled(true);
        simulatorStepAction.setEnabled(true);
        simulatorBackstepAction.setEnabled(true);
        simulatorResetAction.setEnabled(false);
        simulatorStopAction.setEnabled(true);
        simulatorPauseAction.setEnabled(true);
        helpAboutAction.setEnabled(true);
```

```
        editUndoAction.updateUndoState();
        editRedoAction.updateRedoState();
    }

    /** Use this when "File -> New" is used, to force user into Save As */

    void setMenuStateEditingNew() {
        fileNewAction.setEnabled(true);
        fileOpenAction.setEnabled(true);
        fileCloseAction.setEnabled(true);
        fileSaveAction.setEnabled(false);
        fileSaveAsAction.setEnabled(true);
        filePrintAction.setEnabled(true);
        fileExitAction.setEnabled(true);
        editCutAction.setEnabled(true);
        editCopyAction.setEnabled(true);
        editPasteAction.setEnabled(true);
        editFindReplaceAction.setEnabled(true);
        simulatorRunAction.setEnabled(true);
        simulatorStepAction.setEnabled(true);
        simulatorBackstepAction.setEnabled(true);
        simulatorResetAction.setEnabled(true);
        simulatorStopAction.setEnabled(true);
        simulatorPauseAction.setEnabled(true);
        helpAboutAction.setEnabled(true);
        editUndoAction.updateUndoState();
        editRedoAction.updateRedoState();
    }
```

```
/**Menu state when  program is running*/
void setMenuStateRunning() {
        fileNewAction.setEnabled(false);
        fileOpenAction.setEnabled(false);
        fileCloseAction.setEnabled(false);
        fileSaveAction.setEnabled(false);
        fileSaveAsAction.setEnabled(false);
        filePrintAction.setEnabled(false);
        fileExitAction.setEnabled(false);
        editCutAction.setEnabled(false);
        editCopyAction.setEnabled(false);
        editPasteAction.setEnabled(false);
        editFindReplaceAction.setEnabled(false);
        simulatorRunAction.setEnabled(false);
        simulatorStepAction.setEnabled(false);
        simulatorBackstepAction.setEnabled(false);
        simulatorResetAction.setEnabled(false);
        simulatorStopAction.setEnabled(true);
        simulatorPauseAction.setEnabled(true);
        helpAboutAction.setEnabled(true);
        editUndoAction.setEnabled(false);
        editRedoAction.setEnabled(false);
}
```

```
 /** Get current menu state.
 * @return current menu state.
 **/

 public static int getMenuState() {
        return menuState;
 }

 /* *  Set whether QRegister values are reset.
  *   @param bool Boolean true if the QRegister values have been reset.
  **/

 public static void setReset(boolean bool){
   reset=bool;
 }

 /* *  Set whether QC program execution has started.
  *   @param bool true if the QC program execution has started **/
public static void setStarted(boolean bool){
        started=bool;
 }

 /**
  *  determine whether the QRegister values are reset.
  *   @return Boolean true if the QRegister values have been reset.
  **/
 public static boolean getReset(){
        return reset;
 }
```

```
    /* *  determine whether QC program is currently executing.
     *   @return  true if QC program is currently executing.
     **/
    public static boolean getStarted(){
        return started;
    }


    /* *  Get reference to main pane associated with this GUI.
     *   @return qsimMainPane object associated with the GUI **/
    public QsimMainWindow getMainPane() {
        return qsimMainWindow;
    }


    /* *  Get reference to Results pane associated with this GUI.
     *   @return QsimResultsPane object associated with the GUI.
     **/
    public QsimResultsPane getResultsPane() {
        return qsimResultsPane;
    }

    /**
     *  Get reference to registers pane associated with this GUI.
     *   @return QRegisterPane object associated with the GUI.
     **/

    public QRegisterPane getRegistersPane() {
        return qRegisterPane;
    }
}
```

QsimMainWindow.JAVA CLASS

```
package qsim.gui;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import qsim.*;

/**Creates the tabbed areas and windows in the UserGui */
public class QsimMainWindow extends JTabbedPane{
        private static final long serialVersionUID = 1L;
        EditorWindow editTab;
        QRegisterDynamicsPane qRegTab;

        /**  Constructor for the QsimMainWindow class*/
        public QsimMainWindow(UserGui appFrame,QRegistersWindow
regs,FloatingPointWindow floatReg){
                super();
                  editTab = new EditorWindow(appFrame);
                  qRegTab = new QRegisterDynamicsPane(appFrame, regs, floatReg);
                  this.addTab("Edit File Window", editTab);
                  this.addTab("Queue Register Dynamics", qRegTab);
                  this.setToolTipTextAt(0,"Simple text editor for QSIM source file");
                  this.setToolTipTextAt(1,"View Changes in Queue Register at Execution.");

                  /** setWindowsBounds() method set the bound of the internal frame
                   * when QRegisterDynamics Tab is selected
                   */
                  this.addChangeListener(
            new ChangeListener() {
              public void stateChanged(ChangeEvent ce) {
                JTabbedPane tabbedPane = (JTabbedPane) ce.getSource();
```

```
                int index = tabbedPane.getSelectedIndex();
                Component c = tabbedPane.getComponentAt(index);
                QRegisterDynamicsPane qRegisterDynamicsPane =
Globals.getGui().getMainPane().getQRegisterDynamicsPane();
                if (c == qRegisterDynamicsPane) {
                  qRegisterDynamicsPane.setWindowBounds();
                  Globals.getGui().getMainPane().removeChangeListener(this);
                }
              }
            });
        }

        /** returns component containing text editor
         * @return the editor pane*/
        public EditorWindow getEditPane() {
                return editTab;
        }

        /** returns component containing execution-time display
         * @return the execute pane
         */
        public QRegisterDynamicsPane getQRegisterDynamicsPane() {
                return qRegTab;
        }
}
```

QsimResultsPane.JAVA CLASS

```
package qsim.gui;
import javax.swing.*;
import javax.swing.border.BevelBorder;
import qsim.*;
import qsim.registers.QRegisterFile;
import java.awt.*;
import java.awt.event.*;

/**  Creates QSIM Results window */
public class QsimResultsPane extends JTabbedPane{
        private static final long serialVersionUID = 1L;
        JTextArea results, stats;
        JPanel resultsTab, statsTab;
        JScrollPane resultsScroll,statsScroll;
        Color scrollColor;
         SimulateQCprogram sim;

        /** value to keep scrolled contents of the message areas from becoming too large.*/

        public static final int MAXIMUM_SCROLLED_CHARACTERS =
        Globals.maximumMessageCharacters;
        public static final int NUMBER_OF_CHARACTERS_TO_CUT =
        Globals.maximumMessageCharacters/10 ;

/** Constructor for the class, sets up two fresh tabbed text areas for program feedback */
        public QsimResultsPane() {
                super();
                this.setMinimumSize(new Dimension(0,0));
                sim=new  SimulateQCprogram();
                results= new JTextArea();               // Results text area
```

```
            stats= new JTextArea();                    // Statistics text area
            scrollColor=new Color(204,230,255);
            results.setEditable(false);
            stats.setEditable(false);

            Font monoFont = new Font(Font.MONOSPACED, Font.PLAIN, 12);
            results.setFont(monoFont);
            stats.setFont(monoFont);
            results.setBackground(Color.WHITE);
            results.setForeground(Color.BLUE);
            stats.setBackground(Color.WHITE);
            stats.setForeground(Color.BLUE);
            Color myColor=new Color(204,230,255);
            Color color=new Color(102,50,0);

    JButton resultsTabClearButton = new JButton("Clear QSIM Execution Results ");
    resultsTabClearButton.setToolTipText("Clear the QSIM Results Area");
    resultsTabClearButton.addActionListener(
          new ActionListener() {
            public void actionPerformed(ActionEvent e){
              results.setText("");
              QRegisterFile.clearReg();
            }
          });
    resultsTab = new JPanel(new BorderLayout());
    resultsTab.setBackground(myColor);
    resultsTab.setBorder(new BevelBorder(BevelBorder.RAISED));
    resultsTab.setBorder(BorderFactory.createLineBorder(scrollColor, 2));
    resultsTab.add(createBoxForButton(resultsTabClearButton),BorderLayout.NORTH);
   resultsScroll=new JScrollPane(results,
   ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
```

```
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
resultsScroll.setBackground(color);
resultsScroll.setBorder(BorderFactory.createLineBorder(scrollColor,2));
resultsTab.add(resultsScroll, BorderLayout.CENTER);

/** implemented as execute Queue program */
JButton statsTabClearButton = new JButton("Clear QSIM Simulation Statistics");
statsTabClearButton.setToolTipText("Clear the QSIM Execution Error area");
statsTabClearButton.addActionListener(
      new ActionListener() {
        public void actionPerformed(ActionEvent e){
              stats.setText("");
        }
      });
statsTab = new JPanel(new BorderLayout());
statsTab.setBackground(myColor);
statsTab.setBorder(new BevelBorder(BevelBorder.RAISED));
statsTab.setBorder(BorderFactory.createLineBorder(scrollColor, 2));
statsTab.add(createBoxForButton(statsTabClearButton),BorderLayout.NORTH);
statsScroll=new JScrollPane(stats,
ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
statsScroll.setBackground(color);
statsScroll.setBorder(BorderFactory.createLineBorder(scrollColor,3));
statsTab.add(statsScroll, BorderLayout.CENTER);
this.addTab("QSIM Execution Results", resultsTab);
this.addTab(" View QSIM Simulation Statistics", statsTab);
this.setToolTipTextAt(0,"Results produced by QSIM Program Execution ");
this.setToolTipTextAt(1,"Instuction Statistic of QSIM program Execution");
}
```

```
    /** Center given button in a box, centered vertically and 6 pixels on left and right*/
     private Box createBoxForButton(JButton button) {
       Box buttonRow = Box.createHorizontalBox();
       buttonRow.add(Box.createHorizontalStrut(6));
       buttonRow.add(button);
       buttonRow.add(Box.createHorizontalStrut(6));
       Box buttonBox = Box.createVerticalBox();
       buttonBox.add(Box.createVerticalGlue());
       buttonBox.add(buttonRow);
       buttonBox.add(Box.createVerticalGlue());
       return buttonBox;
     }
    /**Method to return the Results text area */
     public JTextArea getResultsTextArea() {
            return results;
     }
     /**Method to return the Computation Statistics Text area */
     public JTextArea getStatsTextArea() {
            return stats;
     }

     /** Make the Results tab current (up front) */
     public void selectQsimMessageTab() {
            setSelectedComponent(resultsTab);
     }

     /** Make the statistics tab current (up front) */
     public void selectStatsMessageTab() {
            setSelectedComponent(statsTab);
     }
}
```

QregistersWindow.JAVA CLASS

```
package qsim.gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.table.*;
import javax.swing.border.BevelBorder;
import javax.swing.event.*;
import qsim.*;
import qsim.registers.*;


/**  Sets up a window to display QRegisters in the Gui **/

public class QRegistersWindow extends JPanel implements Observer {
        private static final long serialVersionUID = 1L;
        private static JTable table;
        private static Object[][] tableData;
        private static boolean highlighting;
        private static int highlightRow;
        private JScrollPane regScrollPane,qLabelScrollPane;
        private JSplitPane regSplitter;
        private static final int INDEX_COLUMN = 0;
        private static final int VALUE_COLUMN = 1;
        private static Settings settings;
        JTableHeader header;
        JLabel qLabel;
        Font boldFont;
        Color myColor;
        Color color;
```

```
        Color headBg;
        Color headFg;

/**  Constructor which sets up a fresh window with a table that contains the QRegister values */
        public QRegistersWindow(){
                settings = Globals.getSettings();
                QRegistersWindow.highlighting = false;
                Color myColor=new Color(102,50,0);          //scroll pane color
                table = new MyTippedJTable(new RegTableModel(setupWindow()));
                myColor=new Color(102,50,0);                //scroll pane color
                Toolkit tk = Toolkit.getDefaultToolkit();

                double screenWidth  = Toolkit.getDefaultToolkit().getScreenSize().getWidth();
                double screenHeight = Toolkit.getDefaultToolkit().getScreenSize().getHeight();
                double registersWidthPct = (screenWidth<1000.0)? 0.16 : 0.20;
                double registersHeightPct = (screenWidth<1000.0)? 0.70 : 0.78;
                Dimension qRegDim=new
Dimension((int)(screenWidth*registersWidthPct),(int)(screenHeight*registersHeightPct*0.6));
                Dimension qLabelDim=new
Dimension((int)(screenWidth*registersWidthPct),(int)(screenHeight*registersHeightPct*0.3));

        /** get the QRegister label image*/
                qLabel=new JLabel(new ImageIcon(tk.getImage(this.getClass()
                                .getResource(Globals.imagesPath + "qLabel.gif"))));

                /** set preferred size for index and value columns*/
                table.getColumnModel().getColumn(INDEX_COLUMN).setPreferredWidth(25);
                table.getColumnModel().getColumn(VALUE_COLUMN).setPreferredWidth(70);
                customiseReg();

                /** Display QRegister values (String-field) right-justified in mono font*/
```

```
        table.getColumnModel().getColumn(INDEX_COLUMN).setCellRenderer(new
RegisterCellRenderer(CenterCellRenderer.MONOSPACED_PLAIN_12POINT,
SwingConstants.CENTER));
        table.getColumnModel().getColumn(VALUE_COLUMN).setCellRenderer(new
RegisterCellRenderer(CenterCellRenderer.MONOSPACED_PLAIN_12POINT,
SwingConstants.CENTER));
        table.setPreferredScrollableViewportSize(new Dimension(200,700));
        table.setGridColor(Color.DARK_GRAY);      // set the grid lines color to dark gray
        this.setLayout(new BorderLayout());
                regScrollPane= new JScrollPane(table,
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        qLabelScrollPane=new JScrollPane(qLabel,
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        qLabelScrollPane.setPreferredSize(qLabelDim);
        regScrollPane.setPreferredSize(qRegDim);
        regScrollPane.setBackground(myColor);
        qLabelScrollPane.setBackground(myColor);
        regScrollPane.setBorder(BorderFactory.createLineBorder(headBg,3));
                regSplitter=new
JSplitPane(JSplitPane.VERTICAL_SPLIT,regScrollPane,qLabelScrollPane);
        regSplitter.resetToPreferredSizes();
        regSplitter.setOneTouchExpandable(true);
        this.add(regSplitter);
}
```

```
/** Method to customize the JTable for QRegisters */
public void customiseReg(){
        color=new Color(82,0,163);                  //Border line color
        headBg=new Color(204,230,255);              // header background color
        headFg=new Color(153,0,77);                 // header foreground color
        header=table.getTableHeader();
        header.setBorder(BorderFactory.createBevelBorder(3));
        header.setBackground(headBg);
        header.setForeground(headFg);
        boldFont= header.getFont().deriveFont(Font.BOLD);
        header.setBorder(new BevelBorder(BevelBorder.RAISED));
        table.getTableHeader().setFont(boldFont);
        table.setBorder(new BevelBorder(BevelBorder.RAISED));
        table.setBorder(BorderFactory.createLineBorder(color,3));
}

/**  Sets up the data for the window **/
public static Object[][] setupWindow(){
        tableData = new Object[128][2];
        for(int i=0; i< QRegisterFile.getQRegister().length; i++){
                tableData[i][0]= (QRegisterFile.getQRegister()[i].getIndex());
                tableData[i][1]= QRegisterFile.getQRegister()[i].getValue();
        }
        return tableData;
}

/**  clear and redisplay registers*/
public static void clearWindow() {
        QRegistersWindow.clearHighlighting();
        QRegisterFile.resetQRegisters();
}
```

```
/** Clear highlight background color from any cell currently highlighted */
public static void clearHighlighting() {
        highlighting=false;
        if (table != null) {
                table.tableChanged(new TableModelEvent(table.getModel()));
        }
        highlightRow = -1;
}

/** Refresh the table, triggering re-rendering */
public static void refresh() {
        if (table != null) {
                table.tableChanged(new TableModelEvent(table.getModel()));
        }
}

/**  Highlight the row corresponding to the given register.
 *  @param register Register object corresponding to row to be selected.
 */
void highlightCellForRegister(QRegister register) {
        QRegistersWindow.highlightRow = register.getIndex();
        table.tableChanged(new TableModelEvent(table.getModel()));
}

/** Cell renderer for displaying register entries */
private class RegisterCellRenderer extends DefaultTableCellRenderer {
        private static final long serialVersionUID = 1L;
        private Font font;
        private int alignment;
```

```
public RegisterCellRenderer(Font font, int alignment) {
        super();
        this.font = font;
        this.alignment = alignment;
}

public Component getTableCellRendererComponent(JTable table, Object value,
                boolean isSelected, boolean hasFocus, int row, int column) {

JLabel cell = (JLabel) super.getTableCellRendererComponent(table, value,
        isSelected, hasFocus, row, column);
        cell.setFont(font);                             // set font for the cell
        cell.setHorizontalAlignment(alignment);      // set cell alignment
        if (settings.getRegistersHighlighting() && highlighting &&
       row==highlightRow) {
                cell.setBackground(
settings.getColorSettingByPosition(Settings.REGISTER_HIGHLIGHT_BACKGR
OUND) );//set background color for the cell
                cell.setForeground(
settings.getColorSettingByPosition(Settings.REGISTER_HIGHLIGHT_FOREGR
OUND) );//set foreground color for the cell
                cell.setFont(
settings.getFontByPosition(Settings.REGISTER_HIGHLIGHT_FONT) );
                // set font for the cell
        }
        else if (row%2==0) {          // for even cells
                cell.setBackground(
settings.getColorSettingByPosition(Settings.EVEN_ROW_BACKGROUND) );
                cell.setForeground(
settings.getColorSettingByPosition(Settings.EVEN_ROW_FOREGROUND) );
```

```
                    cell.setFont(
            settings.getFontByPosition(Settings.EVEN_ROW_FONT) );
                }
                else {   // for odd cells
                    cell.setBackground(
            settings.getColorSettingByPosition(Settings.ODD_ROW_BACKGROUND) );
                    cell.setForeground(
            settings.getColorSettingByPosition(Settings.ODD_ROW_FOREGROUND) );

                    cell.setFont(
            settings.getFontByPosition(Settings.ODD_ROW_FONT) );
                }
                return cell;
            }
        }

        class RegTableModel extends AbstractTableModel {
            private static final long serialVersionUID = 1L;
            final String[] columnNames =  {"Queue Index", "QREG Values"};
            Object[][] data;

            public RegTableModel(Object[][] d){
                data=d;
            }

            public int getColumnCount() {
                return columnNames.length;
            }

            public int getRowCount() {
                return data.length;
```

```
        }

        public String getColumnName(int col) {
            return columnNames[col];
        }

        public Object getValueAt(int row, int col) {
            return data[row][col];
        }


    /** JTable uses this method to determine the default renderer editor for each cell */
        public Class getColumnClass(int c) {
            return getValueAt(0, c).getClass();
        }
    }

    private class MyTippedJTable extends JTable {
        private static final long serialVersionUID = 1L;
        MyTippedJTable(RegTableModel m) {
            super(m);
            this.setRowSelectionAllowed(true);
            this.setSelectionBackground(Color.GREEN);
        }
```

```
private String[] regToolTips = {
        "QREG INDEX    0", "QREG INDEX    1", "QREG INDEX    2",
        "QREG INDEX    3","QREG INDEX    4","QREG INDEX    5",
        "QREG INDEX    6","QREG INDEX    7","QREG INDEX    8",
        "QREG INDEX    9","QREG INDEX    10","QREG INDEX    11",
        "QREG INDEX    12","QREG INDEX    13","QREG INDEX    14",
        "LAST QREG     15","QREG INDEX    16", "QREG INDEX   17",
        "QREG INDEX    18","QREG INDEX    19","QREG INDEX    20",
        "QREG INDEX    21","QREG INDEX    22","QREG INDEX    23",
        "QREG INDEX    24","QREG INDEX    25","QREG INDEX    26",
        "QREG INDEX    27","QREG INDEX    28","QREG INDEX    29",
        "QREG INDEX    30","LAST QREG     31","QREG INDEX    32",
        "QREG INDEX    33","QREG INDEX    34","QREG INDEX    35",
        "QREG INDEX    36","QREG INDEX    37","QREG INDEX    38",
        "QREG INDEX    39","QREG INDEX    40","QREG INDEX    41",
        "QREG INDEX    42","QREG INDEX    43","QREG INDEX    44",
        "QREG INDEX    45","QREG INDEX    46","LAST QREG     47",
        "QREG INDEX    48", "QREG INDEX   49","QREG INDEX    50",
        "QREG INDEX    51","QREG INDEX    52","QREG INDEX    53",
        "QREG INDEX    54","QREG INDEX    55","QREG INDEX    56",
        "QREG INDEX    57","QREG INDEX    58","QREG INDEX    59",
        "QREG INDEX    60","QREG INDEX    61","QREG INDEX    62",
        "LAST QREG     63","LAST QREG     64","QREG INDEX    65",
        "QREG INDEX    66","QREG INDEX    67","QREG INDEX    68",
        "QREG INDEX    69","QREG INDEX    70","QREG INDEX    71",
        "QREG INDEX    72","QREG INDEX    73","QREG INDEX    74",
        "QREG INDEX    75","QREG INDEX    76","QREG INDEX    77",
        "QREG INDEX    78","QREG INDEX    79","LAST QREG     80",
        "LAST QREG     81","QREG INDEX    82", "QREG INDEX   83",
        "QREG INDEX    84","QREG INDEX    85","QREG INDEX    86",
        "QREG INDEX    87","QREG INDEX    88","QREG INDEX    89",
```

```
        "QREG INDEX    90","QREG INDEX   91","QREG INDEX   92",
        "QREG INDEX    93","QREG INDEX   94","QREG INDEX   95",
        "QREG INDEX    96","LAST QREG    97","QREG INDEX   98",
"QREG INDEX    99","QREG INDEX  100","QREG INDEX    101",
"QREG INDEX   102","QREG INDEX  103","QREG INDEX    104",
"QREG INDEX   105","QREG INDEX 106","QREG INDEX    107",
"QREG INDEX   108","QREG INDEX 109","QREG INDEX    110",
"LAST QREG    111","QREG INDEX  112","QREG INDEX    113",
"QREG INDEX   114","QREG INDEX 115","QREG INDEX    116",
"QREG INDEX   117","QREG INDEX  118","QREG INDEX    119",
"QREG INDEX   120","QREG INDEX 121","QREG INDEX    122",
"QREG INDEX   123","QREG INDEX  124","LAST QREG   125",
"QREG INDEX   126","QREG INDEX 127",
};

/**Implement table cell tool tips.*/
public String getToolTipText(MouseEvent e) {
        String tip = null;
        java.awt.Point p = e.getPoint();
        int rowIndex = rowAtPoint(p);
        int colIndex = columnAtPoint(p);
        int realColumnIndex = convertColumnIndexToModel(colIndex);
        if (realColumnIndex ==INDEX_COLUMN  ) { //Register index column
                tip = regToolTips[rowIndex];
        }
        else {
                tip = super.getToolTipText(e);
        }
        return tip;
}
```

```
        private String[] columnToolTips = {
                /* Index */ "Corresponding Register Index",
                /* value */  "Current 16 bit QREG value"
        };

        /**table header tool tips. */
        protected JTableHeader createDefaultTableHeader() {
            return
            new JTableHeader(columnModel) {
                private static final long serialVersionUID = 1L;
                public String getToolTipText(MouseEvent e) {
                    String tip = null;
                    java.awt.Point p = e.getPoint();
                    int index = columnModel.getColumnIndexAtX(p.x);
                    int realIndex =
        columnModel.getColumn(index).getModelIndex();
                    return columnToolTips[realIndex];
                }
            };
        }
    }

    public void update(Observable arg0, Object arg1) {
    }
}
```

QregistersPane.JAVA CLASS

```
package qsim.gui;
import javax.swing.*;

/**  Contains tabbed areas in the UerGui to display QSIM register contents **/

public class QRegisterPane extends JTabbedPane{
        private static final long serialVersionUID = 1L;
        QRegistersWindow regsTab;
        FloatingPointWindow floatTab;

        /**  Constructor for the QRegisterPane class */

        public QRegisterPane(UserGui appFrame, QRegistersWindow regs,
FloatingPointWindow floatP){
                super();
                regsTab = regs;
                floatTab = floatP;
                regsTab.setVisible(true);
                floatTab.setVisible(true);
                this.addTab("Queue Registers", regsTab);
                this.addTab("Floating Point Registers  ", floatTab);
                this.setToolTipTextAt(0,"Queue Registers");
                this.setToolTipTextAt(1,"floating point registers");
        }

        /** Return Queue Register component */
        public QRegistersWindow getQRegistersWindow() {
                return regsTab;
        }
```

```
        /** Return  floating point register set */
        public FloatingPointWindow getFloatingPointWindow() {
                return floatTab;
        }
}
```

QregisterSegmentForAlu.JAVA CLASS

```
package qsim.gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.border.*;
import qsim.*;

/**  Represents the QRegister Content window after Queue computation **/
public class QRegisterSegmentForAlu extends JInternalFrame implements Observer {
        private static final long serialVersionUID = 1L;
        private static JTextArea aluQreg;
        private JScrollPane aluScroller;
        private JButton clearAluQreg,clearLoadQreg;
        private Container contentPane;
        private JPanel aluPan;


        /**  Constructor for the QRegister Content window */

        Color myColor=new Color(0,0,50);                    //scroll pane color
        Color color=new Color(82,0,163);                    //Border line color
        Color scrollColor=new Color(204,230,255);
        public QRegisterSegmentForAlu (){
                super("QREGISTER CONTENT AT COMPUTATION", true, false, true, true);
                aluQreg=new JTextArea();
                clearAluQreg=new JButton("Clear Load Qreg");
                clearLoadQreg=new JButton("Clear ALU Qreg");
                contentPane = this.getContentPane();
                aluPan=new JPanel();
```

```
aluQreg.setBackground(myColor);
aluQreg.setForeground(color.WHITE);
aluQreg.setEditable(false);
aluPan.setLayout(new BorderLayout());
JPanel features = new JPanel();
features.setBorder(new BevelBorder(BevelBorder.RAISED));
features.add(clearAluQreg);
features.add(clearLoadQreg);

contentPane.add(features, BorderLayout.SOUTH);
((JComponent) contentPane).setBorder(BorderFactory.createBevelBorder(3));
        // set border for table in data segment
((JComponent) contentPane).setBorder(BorderFactory.createLineBorder(color,
3));           // set line color for border in data segment
aluScroller=new
JScrollPane(aluQreg,ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);

aluPan.add(aluScroller,BorderLayout.CENTER);
contentPane.add(aluPan);

/** Action listener to clear Qregister content after computation */
clearAluQreg.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e){
                aluQreg.setText("");
        }
});

/** Action Listener to clear QRegister content after load operation */
clearLoadQreg.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e){
```

```
        Globals.getGui().getMainPane().getQRegisterDynamicsPane().getaluSegmentWindow();
                                QRegisterSegmentForLoad.clearLoad();
                        }
                });
        }


        /** method to return the QRegister content text area */
        public static JTextArea getAluQreg(){
                return aluQreg;
        }

        public void update(Observable arg0, Object arg1) {
        }
}
```

QregisterSegmentForLoad.JAVA

```
package qsim.gui;
import javax.swing.*;
import java.awt.*;

/**  Represents the QRegister Content window after Load Operation **/
public class QRegisterSegmentForLoad extends JInternalFrame {
        private static final long serialVersionUID = 1L;
        private static JTextArea loadQreg;
        private JScrollPane loadScroller;
        private  Container contentPane;
        Color myColor=new Color(0,0,50);                    // color for scroll pane
        Color color=new Color(82,0,163);                    //  border line color
        Color scrollColor=new Color(204,230,255);

        /**  Constructor, sets up a new JInternalFrame */
        public QRegisterSegmentForLoad(){
                super("QREGISTER CONTENT AFTER LOAD", true, false, true, true);
                contentPane = this.getContentPane();
                ((JComponent) contentPane).setBorder(BorderFactory.createBevelBorder(3));
                // set border for table
                ((JComponent) contentPane).setBorder(BorderFactory.createLineBorder(color,
3));     // set line color for border
                loadQreg=new JTextArea();
                loadQreg.setBackground(myColor);
                loadQreg.setForeground(Color.WHITE);
                loadQreg.setEditable(false);
                 loadScroller=new
                JScrollPane(loadQreg,ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
                                ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
                contentPane.add(loadScroller);
```

```
        }

        /**Method to return text area object of this class */
        public static JTextArea getLoadQreg(){
                return loadQreg;
        }

        /**Method to clear the clear the text area */
        public static void clearLoad(){
                loadQreg.setText("");
        }
}
```

QregisterDynamicsPane.JAVA

```
package qsim.gui;
import javax.swing.*;
import java.awt.*;

/** Container for the QSIM Register Dynamics windows */
public class QRegisterDynamicsPane extends JDesktopPane {
        private static final long serialVersionUID = 1L;
        private QRegistersWindow registerValues;
        private FloatingPointWindow floatingValues;
        private QRegisterSegmentForAlu aluSegment;
        private QRegisterSegmentForLoad  loadSegment;

/** initialize the QRegisterDynamics pane with major components*/
public QRegisterDynamicsPane(UserGui qsimGui, QRegistersWindow regs,
FloatingPointWindow floatRegs) {
                registerValues = regs;
                floatingValues= floatRegs;
                loadSegment = new QRegisterSegmentForLoad();
                aluSegment = new QRegisterSegmentForAlu();
                loadSegment.setVisible(true);
                aluSegment.setVisible(true);
                this.add(loadSegment);
                this.add(aluSegment);
        }
        /** This method will set the bounds of this JDesktopPane's internal windows
         * relative to the current size of this JDesktopPane.
         */
        public void setWindowBounds() {
                int fullWidth = this.getSize().width - this.getInsets().left - this.getInsets().right;
                int fullHeight = this.getSize().height - this.getInsets().top -
```

```
                this.getInsets().bottom;
                int halfHeight = fullHeight/2;
                Dimension textDim = new Dimension((int)(fullWidth),halfHeight);
                // set load Register segment window dim
                Dimension dataDim = new Dimension((int)(fullWidth),halfHeight);
                // set alu Register segment window dim
                aluSegment.setBounds(0,textDim.height+1, dataDim.width, dataDim.height);
                loadSegment.setBounds(0, 0, textDim.width, textDim.height);
        }

        /** Access the QRegisterForLoad  segment window */
        public QRegisterSegmentForLoad getloadSegmentWindow() {
                return loadSegment;
        }

        /** Access the QRegisterForAlu segment window */
    public QRegisterSegmentForAlu getaluSegmentWindow() {
      return aluSegment;
    }

      /** Access the QRegister values window */
    public QRegistersWindow getQRegistersWindow() {
          return registerValues;
    }

    /** Access floating point register values window */
    public FloatingPointWindow getFloatingPointWindow() {
          return floatingValues;
    }
}
```

EditorWindow.JAVA CLASS

```
package qsim.gui;
import javax.swing.*;
import javax.swing.border.BevelBorder;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.undo.*;
import qsim.*;
import java.util.*;
import java.io.*;


/** Creates the tabbed areas and internal windows in the UserGui */

public class EditorWindow extends JPanel implements Observer {
        private static final long serialVersionUID = 1L;
        private static JTextArea sourceCode;
        private UserGui qsimGui;
        private UndoManager undo;
        private String currentDirectoryPath;
        private JLabel caretPositionLabel;
        private JCheckBox showLineNumbers;
        private JLabel lineNumbers;
        private static final char newline = '\n';
        private boolean isCompoundEdit = false;
        private CompoundEdit compoundEdit;
        Color scrollColor;
```

```
/**  Constructor for the EditorWindow class */
 public EditorWindow(UserGui appFrame){
                super(new BorderLayout());
                this.qsimGui = appFrame;
                currentDirectoryPath = System.getProperty("user.dir","c:\\");
                undo = new UndoManager();
                qsimGui.editor = new Editor(qsimGui);
                Color myColor=new Color(175,202,202);
                Globals.getSettings().addObserver(this);
                EditorWindow.sourceCode = new JTextArea();
                sourceCode.setBorder(BorderFactory.createBevelBorder(5));
                sourceCode.setBorder(BorderFactory.createLineBorder(myColor, 3));
                sourceCode.setBackground(myColor);
                EditorWindow.sourceCode.setFont(Globals.getSettings().getEditorFont());
                EditorWindow.sourceCode.setTabSize(4);
                EditorWindow.sourceCode.setMargin(new Insets(0,3,3,3));

                /**set flag to trigger/request file save If source code is modified */
                sourceCode.getDocument().addDocumentListener(
                                new DocumentListener() {
                                        public void insertUpdate(DocumentEvent evt) {
                                                FileStatus.setEdited(true);
                                                switch (FileStatus.get()) {
                                                case FileStatus.NEW_NOT_EDITED:
                                                        FileStatus.set(FileStatus.NEW_EDITED);
                                                        break;
                                                case FileStatus.NEW_EDITED:
                                                        break;
                                                default:
                                                        FileStatus.set(FileStatus.EDITED);
                                                }
```

```
                                qsimGui.editor.setFrameTitle();
                                if (showingLineNumbers()) {

        lineNumbers.setText(getLineNumbersList());//Set line numbers label
                                }
                        }
                public void removeUpdate(DocumentEvent evt) {
                        this.insertUpdate(evt);
                }
                public void changedUpdate(DocumentEvent evt) {
                        this.insertUpdate(evt);
                        }
                });

    /** support undo/redo capability */
            sourceCode.getDocument().addUndoableEditListener(
                    new UndoableEditListener() {
                            public void undoableEditHappened(UndoableEditEvent e) {
                                    if (isCompoundEdit) {
                                    compoundEdit.addEdit(e.getEdit());
                                    }
                                    else {
                                            undo.addEdit(e.getEdit());
                                    qsimGui.editUndoAction.updateUndoState();
                                            qsimGui.editRedoAction.updateRedoState();
                                    }
                            }
                    });

            sourceCode.getCaret().addChangeListener(
                            new ChangeListener() {
```

```
                    public void stateChanged(ChangeEvent e) {
        displayCaretPosition(convertStreamPositionToLineColumn(sourceCode.getCaretPosition
()));
                        }
                    });
            this.setSourceCode("",false);
            lineNumbers = new JLabel();
            lineNumbers.setFont(getLineNumberFont(sourceCode.getFont()));
            lineNumbers.setForeground(Color.BLACK);
            lineNumbers.setBackground(Color.WHITE);
            lineNumbers.setVerticalAlignment(JLabel.TOP);
            lineNumbers.setText("");
            lineNumbers.setVisible(false);
            showLineNumbers = new JCheckBox(" Display Line Numbers");
            showLineNumbers.setToolTipText(" If checked, will display line number for
            each line of text.");
            showLineNumbers.setForeground(new Color(255,0,153));
            showLineNumbers.setEnabled(false);

            /** Show line numbers by default */
        showLineNumbers.setSelected(Globals.getSettings().getEditorLineNumbersDisplayed());
            lineNumbers.setVisible(true);

        /** Listener fires when "Display Line Numbers" check box is clicked */
            showLineNumbers.addItemListener(
                    new ItemListener() {
                        public void itemStateChanged(ItemEvent e) {
                            if (showLineNumbers.isSelected()) {
                        lineNumbers.setText(getLineNumbersList());
                                lineNumbers.setVisible(true);
                            }
```

```
                                        else {
                                                lineNumbers.setVisible(false);
                                        }

Globals.getSettings().setEditorLineNumbersDisplayed(showLineNumbers.isSelected());

                                        setCursorVisible(true); // caret to reappear
                                        sourceCode.requestFocusInWindow();
                                }
                        });
                Color color=new Color(102,51,0);     // border line color
                scrollColor=new Color(204,230,255);
                JPanel source = new JPanel(new BorderLayout());   // panel for the edit pane
                source.setBorder(new BevelBorder(BevelBorder.RAISED));
                source.setBorder(BorderFactory.createLineBorder(Color.BLACK, 3));
                source.add(lineNumbers,BorderLayout.WEST);// add line number label at west
                source.add(sourceCode,BorderLayout.CENTER);// add editor window at center
                JScrollPane editAreaScrollPane = new JScrollPane(source,
                                ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
                                ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
                editAreaScrollPane.setBackground(color);
                editAreaScrollPane.getVerticalScrollBar().setUnitIncrement(

sourceCode.getFontMetrics(EditorWindow.sourceCode.getFont()).getHeight());
                editAreaScrollPane.setBorder(BorderFactory.createLineBorder(scrollColor,3));
                // set color of edit text area scroll pane
                this.add(editAreaScrollPane, BorderLayout.CENTER);
                JPanel editInfo = new JPanel(new FlowLayout(1,1,1));
                editInfo.setBorder(new BevelBorder(BevelBorder.RAISED));
                        // set line check box panel raised
```

```
            editInfo.setBorder(BorderFactory.createLineBorder(scrollColor, 3));
            // set line color for border

            caretPositionLabel = new JLabel();
            caretPositionLabel.setForeground(new Color(255,0,153));
            caretPositionLabel.setToolTipText("Tracks the current position of the text editing
            cursor.");
            displayCaretPosition(new Point());
            editInfo.add(caretPositionLabel);
            editInfo.add(showLineNumbers);
            editInfo.add(SimulatorSpeedPanel.getInstance());
            this.add(editInfo,BorderLayout.SOUTH);
    }


    /**  For initializing the source code when opening an ASM file
     *   @param s String set to null
     *   @param editable set true if code is editable else false
     **/

    public void setSourceCode(String s, boolean editable){
            EditorWindow.sourceCode.setText(s);
                                        // set text area to null
            EditorWindow.sourceCode.setBackground( (editable)? Color.WHITE :
            scrollColor);   //if text area is editable set color to white else blue
            EditorWindow.sourceCode.setEditable(editable);
            EditorWindow.sourceCode.setEnabled(editable);
            EditorWindow.sourceCode.getCaret().setVisible(editable);
            EditorWindow.sourceCode.setCaretPosition(0);
            if (editable) this.sourceCode.requestFocusInWindow();
}
```

```
        /** Form string with source code line numbers.
         * Resulting string is HTML,
         * The line number list is a JLabel with one line number per line.
         */
        public String getLineNumbersList() {
                StringBuffer lineNumberList = new StringBuffer("<html>");
                int lineCount = this.getSourceLineCount();   // get number of lines in source code
                for (int i=1; i<=lineCount;i++) {
                        lineNumberList.append(""+i+"<br>");
                }
                lineNumberList.append("</html>");
                return lineNumberList.toString();
        }

        /** Calculate and return number of lines in source code text.
         *  Do this by counting newline characters then adding one if last line does
         *  not end with newline character.
         */
        public int getSourceLineCount() {
                BufferedReader bufStringReader = new BufferedReader(new
StringReader(sourceCode.getText()));
                int lineNums = 0;
                try {
                        while (bufStringReader.readLine() != null) {
                                lineNums++;
                        }
                }
                catch (IOException e) {
                }
```

```
        return lineNums;
}

/**  Adds the source code line by line.
 *   @param str A line of source code.
 **/

public void append(String str){
        this.sourceCode.append(str);
        this.sourceCode.setCaretPosition(0);
}


/** Get source code text
 * @return String containing source code
 */
public String getSource(){
        return sourceCode.getText();
}

public static JTextArea getTextarea(){
        return sourceCode;
}

/**  Get the current directory path
 * @return String containing current directory pathname
 */
public String getCurrentDirectoryPath() {
        return currentDirectoryPath;

/**  Set the current directory pathname
```

```
     * @param path the desired directory pathname
     */
    public void setCurrentDirectoryPath(String path) {
            currentDirectoryPath = path;
    }

    /**  get the manager in charge of UnDo operations
     *  @return the UnDo manager
     */
    public UndoManager getUndoManager() {
            return undo;
    }

    /** copy currently-selected text into clipboard */
 public void copyText() {
   sourceCode.copy();
   setCursorVisible(true);
   sourceCode.getCaret().setSelectionVisible(true);
 }

 /** cut currently-selected text into clipboard */
 public void cutText() {
   sourceCode.cut();
   setCursorVisible(true);
 }

 /** paste clipboard contents at cursor position */
 public void pasteText() {
       sourceCode.paste();
       setCursorVisible(true);
 }
```

```
/** Control cursor visibility
 * @param vis true to display cursor, false to hide it
 */
public void setCursorVisible(boolean vis) {
        sourceCode.getCaret().setVisible(vis);
}

/** get editor's line number display status
 *  @return true if editor is current displaying line numbers, false otherwise.
 */
public boolean showingLineNumbers() {
        return showLineNumbers.isSelected();
}

/**  enable or disable checkbox that controls display of line numbers
 *  @param enable True to enable box, false to disable.
 */
public void setShowLineNumbersEnabled(boolean enabled) {
        showLineNumbers.setEnabled(enabled);
}

/**  Display cursor coordinates
 *  @param p  Point object with x-y (column, line number) coordinates of cursor
              */
public void displayCaretPosition(Point p) {
        caretPositionLabel.setText("Line: "+ p.y + " Column: "+ p.x);
}
```

```
/**  Methods to support Find/Replace feature */
public static final int TEXT_NOT_FOUND = 0;
public static final int TEXT_FOUND = 1;
public static final int TEXT_REPLACED_FOUND_NEXT = 2;
public static final int TEXT_REPLACED_NOT_FOUND_NEXT = 3;

/** Finds next occurrence of text in a forward search of a string. Search begins
 * at the current cursor location, and wraps around when the end of the string
 * is reached.
 * @param find the text to locate in the string
 * @param caseSensitive true if search is to be case-sensitive, false otherwise
 * @return TEXT_FOUND or TEXT_NOT_FOUND, depending on the result.
 */
public int doFindText(String find, boolean caseSensitive) {/ find is the string searching for
      int findPosn = sourceCode.getCaretPosition();      // current cursor position
      int nextPosn = 0;
      nextPosn = nextIndex( sourceCode.getText(), find, findPosn, caseSensitive );
      if ( nextPosn >= 0 ) {
            sourceCode.setSelectionStart( nextPosn );  // position cursor at word start
            sourceCode.setSelectionEnd( nextPosn + find.length() );
            return TEXT_FOUND;
      }
      else {
            return TEXT_NOT_FOUND;
      }
}
```

```
/** Returns next posn of word in text - forward search.  If end of string is
 *  reached during the search, will wrap around to the beginning one time.
 * @return next indexed position of found text or -1 if not found
 * @param input the string to search
 * @param find the string to find
 * @param start the character position to start the search
 * @param caseSensitive true for case sensitive. false to ignore case
 */
public int nextIndex(String input, String find, int start, boolean caseSensitive ) {
	int textPosn = -1;
	if ( input != null && find != null && start < input.length() ) {
		if ( caseSensitive ) { // indexOf() returns -1 if not found
			textPosn = input.indexOf( find, start );
			// If not found from non-starting cursor position, wrap around
			if (start > 0 && textPosn < 0) {
				textPosn = input.indexOf( find );
			}
		}
		else {
			String lowerCaseText = input.toLowerCase();
			textPosn = lowerCaseText.indexOf( find.toLowerCase(), start );
			// If not found from non-starting cursor position, wrap around
			if (start > 0 && textPosn < 0) {
				textPosn = lowerCaseText.indexOf( find.toLowerCase() );
			}
		}
	}
	return textPosn;
}
```

```
 /** Finds and replaces next occurrence of text in a string in a forward search.
  * @param find the text to locate in the string
  * @param replace the text to replace the find text with - if the find text exists
  * @param caseSensitive true for case sensitive. false to ignore case
  * @return Returns TEXT_FOUND if not initially at end of selected match and matching
  * occurrence is found.  Returns TEXT_NOT_FOUND if the text is not matched.
  * Returns TEXT_REPLACED_NOT_FOUND_NEXT if replacement is successful but there
are
  * no additional matches.  Returns TEXT_REPLACED_FOUND_NEXT if replacement is
  * successful and there is at least one additional match.
  */
    public int doReplace(String find, String replace, boolean caseSensitive) {
        int nextPosn = 0;
        if (find==null || !find.equals(sourceCode.getSelectedText()) ||
                        sourceCode.getSelectionEnd()!=sourceCode.getCaretPosition()) {
            return doFindText(find, caseSensitive);
        }
        // We are positioned at end of selected "find".  replace and find next.
        nextPosn = sourceCode.getSelectionStart();
        sourceCode.grabFocus();
          sourceCode.setSelectionStart( nextPosn );                // posn cursor at word start
          sourceCode.setSelectionEnd( nextPosn + find.length() );        //select found text
          isCompoundEdit = true;
          compoundEdit = new CompoundEdit();
          sourceCode.replaceSelection(replace);
          compoundEdit.end();
          undo.addEdit( compoundEdit );
          qsimGui.editUndoAction.updateUndoState();
          qsimGui.editRedoAction.updateRedoState();
          isCompoundEdit = false;
```

```
            sourceCode.setCaretPosition(nextPosn + replace.length());
            if (doFindText(find, caseSensitive) == TEXT_NOT_FOUND) {
                return TEXT_REPLACED_NOT_FOUND_NEXT;
            }
            else {
                return TEXT_REPLACED_FOUND_NEXT;
            }
    }

    /** Finds and replaces ALL occurrences of text in a string in a forward search.
     *  All replacements are bundled into one CompoundEdit, so one Undo operation will
     *  undo all of them.
     * @param find the text to locate in the string
     * @param replace the text to replace the find text with - if the find text exists
     * @param caseSensitive true for case sensitive. false to ignore case
     * @return the number of occurrences that were matched and replaced.
     */
    public int doReplaceAll(String find, String replace, boolean caseSensitive) {
        int nextPosn = 0;
        int findPosn = 0;                   // begin at start of text
        int replaceCount = 0;
        compoundEdit = null;                // new one will be created upon first replacement
        isCompoundEdit = true;              // undo manager's action listener needs this
        while (nextPosn >= 0) {
      nextPosn = nextIndex( sourceCode.getText(), find, findPosn, caseSensitive );
      if ( nextPosn >= 0 ) {
        if (nextPosn < findPosn) {
          break;
        }
        sourceCode.grabFocus();
        sourceCode.setSelectionStart( nextPosn );                   // posn cursor at word start
```

```
        sourceCode.setSelectionEnd( nextPosn + find.length() );  //select found text
        if (compoundEdit == null) {
          compoundEdit = new CompoundEdit();
        }
        sourceCode.replaceSelection(replace);
        findPosn = nextPosn + replace.length();                    // set for next search
        replaceCount++;
      }
    }
        isCompoundEdit = false;       // Will be true if any replacements were performed
        if (compoundEdit != null) {
              compoundEdit.end();
              undo.addEdit( compoundEdit );
              qsimGui.editUndoAction.updateUndoState();
              qsimGui.editRedoAction.updateRedoState();
        }
        return replaceCount;
}

/** Given byte stream position in text being edited, calculate its column and line
 * number coordinates.
 * @param stream position of character
 * @return position Its column and line number coordinate as a Point
 */
public Point convertStreamPositionToLineColumn(int position) {
        String textStream = sourceCode.getText();
        int line = 1;
        int column = 1;
        for (int i=0; i<position; i++) {
                if (textStream.charAt(i) == newline) {
                        line++;
```

```
                    column=1;
                }
                else {
                    column++;
                }
            }
            return new Point(column,line);
        }


        /**  Update, if source code is visible, when Font setting changes.
         *  This method is specified by the Observer interface.
         */
        public void update(Observable fontChanger, Object arg) {
            sourceCode.setFont(Globals.getSettings().getEditorFont());
            sourceCode.revalidate();
            lineNumbers.setFont(getLineNumberFont(sourceCode.getFont()));
            lineNumbers.revalidate();
        }


        /** Method to Determine font to use for editor line number display, given current
         * font for source code.
         */
        private Font getLineNumberFont(Font sourceFont) {
            return  (sourceCode.getFont().getStyle() == Font.PLAIN)
            ? sourceFont
                    : new Font(sourceFont.getFamily(), Font.PLAIN, sourceFont.getSize());
        }
    }
```