Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
UAB/B/SHI/ Jenny LIU				
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
		1999-05-20	PA1	

VISUALIZATION EXECUTION PIPELINE

Jenny Yun Liu

M. S. C. S.

KUNGL TEKNISKA HÖGSKOLAN

Department of Teleinformatics

Royal Institute of Technology

Stockholm

Sweden

1998

Supervisor

Pär Asplund (B/SHI)

Ericsson Utvecklings AB

Ericsson AXE Research and Development

Examiner

Björn Lisper

Department of Teleinformatics

Royal Institute of Technology (KTH)

Supervisor

Sven Lämmermann

Department of Teleinformatics

Royal Institute of Technology (KTH)

Abstract

This project is based on two units that are function blocks in the subsystem of the control system for the Ericsson AXE switch. The execution in the units is super-scalar and pipelined in six steps. Each step operates in parallel with the other steps. Those six steps are discussed in detail in the report.

In order to examine the flow of execution instructions, a graphical tool was developed. The work includes two major parts: one is to find the information on every step, and the other is to develop a graphical tool to show the information. The design specification is also given in the report.

The result of this project is Visualization Execution Pipeline, an applet written in Java that shows the flow of executed instructions.

Abbreviations

Abbreviation	Description
ACQ	Address Calculation Queue
ALU	Arithmetic Logic Unit
a-parameter	ASA parameter used to access data from DS via RS, 12 bits
ASA	Assembler language of IPU
ASA Word	16 bits word from PRS containing ASA instructions
ATIDEC	Instruction Decoder in ATU
ATMFH	Micro Program Flow Handler in ATU
ATPARTDH	Partition buffer in ATU
ATDELIVERY	Delivery buffer in ATU
ATU	Assembler to microcode Translation Unit
CM	Control Memory
CPI	Clock cycles Per Instruction
CPS	Central Processor Subsystem
DSH	Data Store Handler
DSU	Data Store Unit
EP	ASA instruction "end of program"
HRU	Hardware Registers Unit
ICM	IDEC CM. The small Control Memory located in IDEC
IPU	Instruction Processor Unit

Abbreviation	Description
IQC	Instruction Queue Controller
JAMU	Jump Address Memory Unit
LMU	Load and Measurement Unit
MIP	Micro Program built up as a sequence of MIP words
MIP Word	Word from internal Control Memory CM, ICM controlling hardware execution of IPU
Multi MIP Word ASA instruction	ASA instruction which is built up by more than one MIP word
n-parameter	16 bits jump address within a program block
PRS	Program and Reference storage
PRSU	Program and Reference Storage Unit
PS	Program Storage
PS Word	128 bits word from PRS containing ASA instructions
RMU	Register Memory Unit
ROB	Re-order Buffer
SN	Sequence Number
ssp-parameter	Signal Sending Pointer Parameter is used by related ASA instructions to find the location in PS of the ASA code for the signal, 12 bits
TRAM	Trace Address Memory
UMU	Update and Match Unit
vIn	Sub variable information

Uppgjord (även faktaansvarig om annan) - *Prepared (also subject responsible if other)*		Nr - *No.*		
Dokansv/Godkänd - *Doc respons/Approved*	Kontr - *Checked*	Datum - *Date*	Rev	File
		1999-05-20	PA1	

Contents

Page

1 PREFACE

1.1 BACKGROUND

The central processors for the AXE system are developed at Ericsson AXE Research and Development. The processor is based on custom designed CPU circuits. A very high performance is achieved by using state of the art processor design techniques together with an architecture tailored for the application. Measurements and simulations are used for testing new ideas for future designs.

1.2 SPECIFICATION OF THE PROJECT

IPU, the Instruction Processor Unit, is a function block in the subsystem, CPS, of the control system APZ 212 30. ATU, the Assembler to micro code Translator Unit, is included as a function unit in the IPU. IQC, the Instruction Queue Controller, is included as a function unit in the IPU, too.

The IQC executes the decoded micro-instructions delivered from ATU. Two decoded micro-instructions can be delivered and executed. The results are extracted and fed back at each clock cycle which makes this machine two-way super-scalar.

IQC can run under the IPU simulation environment, which provides a framework for performing functional verification of almost all functions in the IPU. The output from HW simulators(verilog, VCS and speedSim) is usually displayed in waveforms. The aim of this master thesis work was to visualize the flow of instructions in the execution pipeline to enhance the understanding of a processor's internal functions.

1.3 THE ORGANIZATION OF THE REPORT

Chapter 2 gives the definition of the project. It includes the background of the thesis work, what ATU and IQC are and how they work. Then it describes the assignment.

Chapter 3, Related Work, includes all the studies I did before and during the project's process, which focuses on 3 parts, pipeline, Java and Java Script.

Chapter 4 describes the six steps of execution pipeline in detail. It explains how those steps work in parallel with the other steps.

Chapter 5 describes what kind of data is picked out and how the output file is formalized.

Chapter 6 is a design specification for the project.

In Chapter 7 there is a summary of the thesis work and some words about the further work.

1.4 ACKNOWLEDGMENTS

Here I would like to express my appreciation to the following persons who helped me for my project.

I am very thankful to my supervisor Pär Asplund at Ericsson AXE Research and Development. Without his help I would not have had my project done at Ericsson.

I am also thankful to my examiner Björn Lisper and my supervisor Sven Lämmermann at Department of Teleinformatics in the Royal Institute of Technology for supporting and examining this project.

Special thanks go to Lars Winberg at B/SHI of Ericsson AXE Research and Development for his endless patience, contribution and encouragement in helping me.

Thanks also to Per Holmberg, Michael Rosendahl, Lars-Erik Lundström, Sven-Åke Andersson and all staff at the department where I did the project.

I would also like to express my gratitude to Dr. Jonas Skeppstedt at the Centre of Computer Systems Architecture of Halmstad University for advising me.

2 PROBLEM DEFINITION

2.1 BACKGROUND

The AXE system is the most widely used switching system in the world. It is used in public telephony-oriented applications of every type, including traditional fixed network applications in local, transit, international and combined networks. AXE is also used for all major mobile standards – analogue and digital. AXE is very strong in intelligent networks and other real-time database applications.[17]

The central processor for the AXE system is developed at Ericsson AXE Research and Development. The processor APZ is based on custom designed CPU circuitry. A very high performance is achieved by using state of the art processor design techniques together with an architecture tailored for the application. The software is written in high level language PLEX and compiled to assemble ASA. PLEX and ASA are Ericsson's own languages. APZ-ODEN is a project that develops a new central processor, APZ 212 30. It has the real-time capacity of four times compared with the old processor, APZ 212 20.

The PRSU administrates the program and reference storage, PRS. That includes read, write and error correction and detection.

ATU controls the instruction fetch. ATU receives the ASA instructions from PRSU and translates them into MIP instructions. The instruction fields are extracted and distributed to the other blocks within the IPU. For example, variable information is delivered to PRSU, register parameters and constants are delivered to IQC.

IQC processes the MIP instructions delivered by ATU. First the sources of the instruction are fetched, then the result of the instruction is calculated by ALU and in the final step the result is stored.

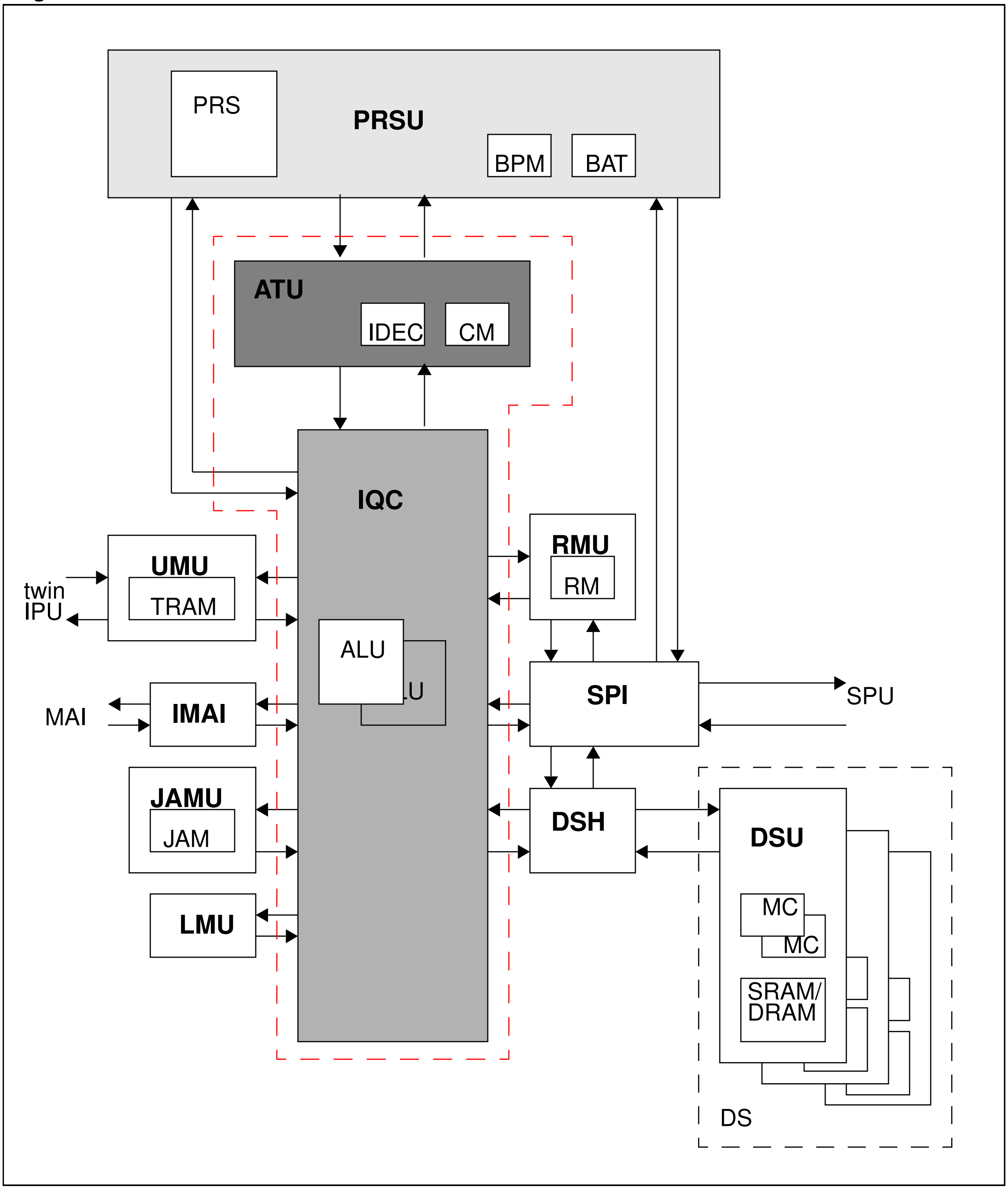
2.2 THE ASSIGNMENT

As we see in Figure 1, there are several units in IPU. ATU and IQC are two of them. They work as super-scalar and are pipelined in six steps. Those six steps are: PS-fetch, Partition, Decoder, Dispatch/OP-fetch, ALU-operation and Commit/ROB. The processor uses branch prediction and speculative execution. The simulation runs under the IPU simulatiuon enviroment. The output results from HW simulators(verilog, VCS and speedSim) are usually displayed in waveforms.

The aim of this thesis work was to develop a graphical tool to show the parallel actions. The output of each step will be shown as digital numbers in the graphical tool. The format of output is hexadecimal digits.

This graphical tool, Execution Pipeline Visualization, is an applet written in Java. The applet uses the output from HW simulators as the input. The applet shows the flow of instructions in the execution pipeline, i. e. the output data from each step will be shown in the applet.

Figure 1 Structure of IPU



3 RELATED WORK

3.1 GENERAL

The Literature Study is a part of the master thesis work. The goals of this study are:

- get used to the researcher's way of working by studying what others have done in the problem domain before attacking the problem.
- practice searching relevant literature.
- get used to reading research literature.
- do a better work by having a good background knowledge of the problem domain.

3.2 INSTRUCTION SETS

The instruction set, including the data types that it describes, is an important basis for the implementation of any machine. Efficient implementations of the instruction set take into account the objects and actions described by the instructions. More about instruction set and data types are described in detail in *Computer Architecture, Pipelined and Parallel Processor Design* by Micheal J. Flynn.Chapter [4]

An instruction consists of format, operation, addresses, and sequence control information. This information may be either represented implicitly or explicitly encoded into the instruction, creating a myriad of possible instruction set combinations. For a variety of reasons, most modern machines are based upon general-purpose register sets that contain many of the data arguments used by the instructions. For purposes of assessing code density, cache effectiveness, and general timing properties of an instruction set, we partition most modern instruction sets into one of three classes:

1. The L/S, or load-store class, which includes most of the RISC machines.
2. The R/M, or register-memory machines, which include the IBM System/390 and, more loosely, the Intel x86 series of instruction sets.
3. The R+M, or register-plus-memory architecture, which includes the DEC VAX and, more loosely, the Motorola 680x0 class of machines.

While the partitioning is only approximate, it is important to recognize that fundamental differences in instruction set formats, as represented by these classes, lead to significantly different levels of code density and implementation tradeoffs.

In dealing with addressing and memory, three levels of address mapping can be distinguished:

1. The processor, or user level, which is the address space visible to the user.
2 The system address space, which is the address space visible to the operating system.
3 The memory space, which is the address space visible to the hardware.

These three levels of mapping involve different tradeoffs and different considerations that must be carefully integrated for an efficient overall mapping mechanism.

Given a well-defined instruction set and the basic timing parameters of a processor (internal cycle time, definition of the cycle, and memory timing considerations), it is generally possible to predict instruction timing. There is an expected sequence of actions associated with the interpretation of each instruction. When an implementation executes the expected sequence, we describe that implementation as a well-mapped design.

3.3 PIPELINING

3.3.1 Historical Development of Pipelining

At the beginning of the 1980s, pipelining was used almost exclusively by mainframes and supercomputers. By the end of the decade, pipelining was used by every new machine.Chapter [5]

Pipelining improves the throughput of a machine without changing the basic cycle time. Thus, pipelining is an especially useful technique when the gate count available is growing faster than gate speed. Of course, improvements in the gate speed can be independently incorporated into changes in the cycle time.

In the early 1980s, most machines had little or no pipelining and the average instruction took 5 to 10 cycles(that is, the cycles per instruction was 5 to 10). By the end of the decade, several RISC machines achieved CPI values close to 1 for integer code. For the basic pipeline shown in Chapter Figure 2, a machine with a CPI of 1 essentially represents in ideal value.

Pipelining increases performance by exploiting instruction-level parallelism. instruction-level parallelism is available when instructions in a sequence are independent and thus can be executed in parallel by overlapping. In the ideal case, a pipelined machine can complete an instruction every cycle. One of the requirements to achieve this is an amount of instruction-level parallelism that is sufficient to keep the pipeline full of independent instructions.

3.3.2 What Is Pipelining

Pipelining is an implementation technique whereby multiple instructions are overlapped in execution. Today, pipelining is the key implementation technique used to make fast CPUs.Chapter [1]

A pipeline is like an assembly line. Figure 2 shows a simple machine pipeline. In an automobile assembly line, there are many steps, each contributing something to the constructions of the car. Each step operates in parallel with the other steps, though on a different car. In a computer pipeline, each step in the pipeline completes a part of an instruction. Like the assembly line, different steps are completing different parts of different instructions in parallel. Each of these steps is called a pipe stage or a pipe segment. The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end, just as cars would in an assembly line.

Figure 2 A Simple Machine Pipeline

Instruction Fetch	**Register Fetch**	**Execution**	**Data Memory Access**	**Write Register Result**		
	Instruction Fetch	**Register Fetch**	**Execution**	**Data Memory Access**	**Write Register Result**	
		Instruction Fetch	**Register Fetch**	**Execution**	**Data Memory Access**	**Write Register Result**

Pipelining yields a reduction in the average execution time per instruction. Depending on what you consider as the base line, the reduction can be viewed as decreasing the number of clock cycles per instruction(CPI), as decreasing the multiple clock cycles time, or as a combination. If the starting point is a machine that takes multiple clock cycles per instruction, then pipelining is usually viewed as reducing the CPI. This is the primary view we will take. If the starting point is a machine that takes one(long) clock cycle per instruction, then pipelining decreases the clock cycle time.

Pipelining is an implementation technique that exploits parallelism among the instruction in a sequential instruction stream. Using the DLX example, we will look at the problems that pipelining introduces and performance attainable under typical situations.

3.4 A PIPELINED DATAPATH MODEL

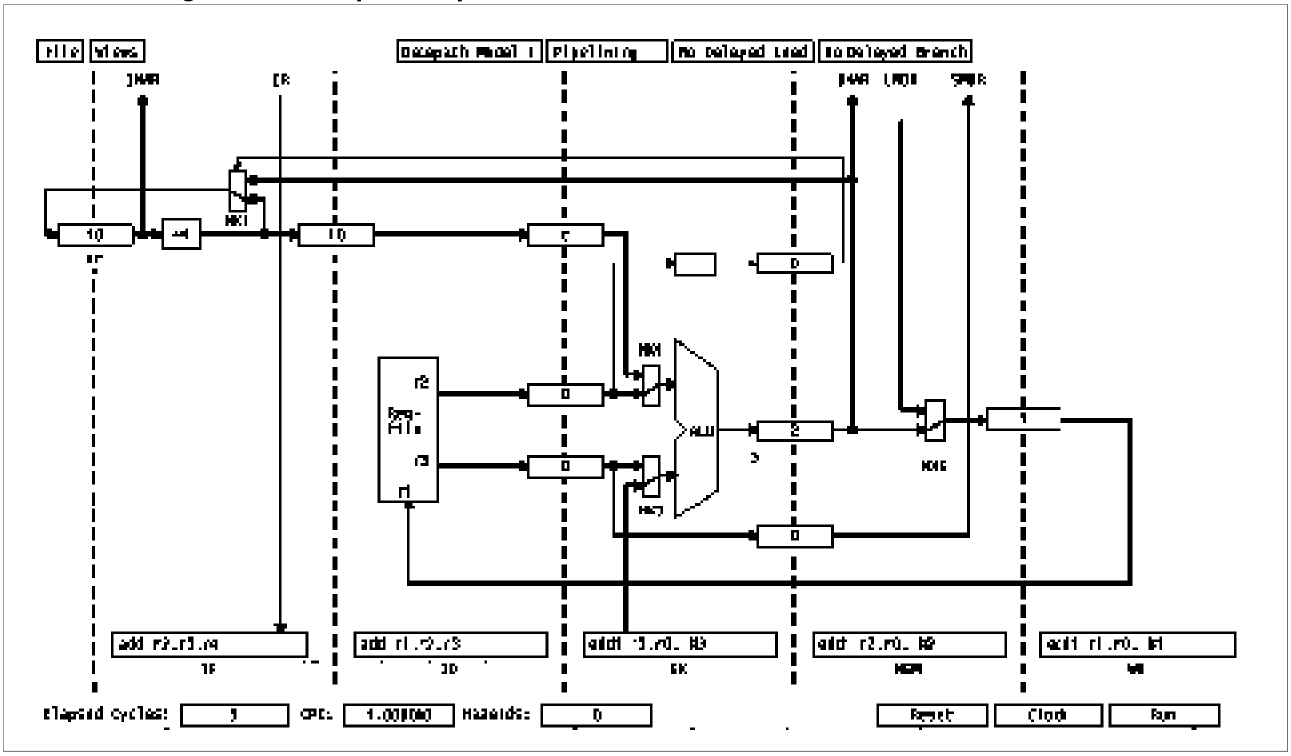
3.4.1 DLX -- A Simple Datapath Model

In the landmark book *Computer Architecture -- A Quantitative Approach* Hennessy and Patterson develop a hypothetical instruction set model DLX which is shown in Figure 3.

DLX emphasizes

- A simple load-store instruction set
- Design for pipelining efficiency, including a fixed instruction set encoding
- Efficiency as a compiler target

Figure 3 A simple datapath model for DLX



This DLX model consists of five pipeline stages denoted IF(Instruction Fetch), ID(Instruction Decode), EX(Execute), MEM(Memory Access), and WB(Write Back to Registerfile). All functional units of the simple datapath model are shown as boxes between two vertical lines. The following functional units are the most important ones: the registerfile, the ALU, and a number of multiplexers denoted MX1, MX4, MX5 and MX6. All thick arrows denote 32-bit buses and their directions. For instance, since there are two read ports in the registerfile, there are two 32-bit buses. The line in each multiplexer visualizes how it is controlled. For example, MX4 connects the registerfile with the ALU, whereas MX5 connects the immediate in the instruction with the ALU.Chapter [3]

3.4.2 Implementation of DLX

Based on this model, Per Stenström, Håkan Nilsson and Jonas Skeppstedt from Lund University have developed a graphical tool that makes it possible to study the parallel actions involved in a pipelined datapath and control unit. They have also successfully developed a laboratory based on the tool.Chapter [3]

The source code of the graphical tool can be reached at: *ftp://ftp.it.lth.se/pub/dlx_pipe_sim.*

At course Computer Architecture Fundamentals which gives by Department of Teleinformatics on KTH, I used this tool by doing laboratory 1 - pipeline.

The laboratory aims at understanding that the functionality of the datapath is sufficient to execute all DLX instructions by picking one instruction from each of the instruction groups: LOAD, STORE, ALU and BRANCH. This is used by allowing only a single instruction in the pipeline at a time, and carefully examining the dataflow in the datapath caused by the instruction. The tool has the capability to disable pipelining to allow this.

Another goal of this laboratory is to show the potential performance improvement of instruction pipelining.

In order to visualize the parallelism associated with pipelining, there is a box beneath each pipeline stage which keeps the mnemonic of the instruction that is currently executed in this stage. For example, as shown in Chapter Figure 3 the instruction add r1,r2,r3 is in the ID-stage whereas the instruction addi r3,r0,#3 is in the EX-stage. By clicking the Clock with the mouse, instructions are forwarded one step in the pipeline.

Finally we can say that instruction pipelining provides a potential speedup of five times as compare to the non-pipelined datapath.

3.5 ABOUT JAVA

Why is Java so interesting? Sun described Java as follows:

Java: A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language.[21]

There are several reasons for why I use Java as the implementation language, and some of them are: simple and powerful, object oriented, and architecture neutral [28].

The flexibility of Java is enhanced by its foundation on an object-oriented paradigm. Java supplies all the advantages of object-oriented programming by supporting such features as encapsulation and inheritance. One will be forced to write program in a object-oriented way if he uses Java. Once the basis concepts of object-oriented programming are understood, any one can learn to program in Java very quickly.

Java also supports dynamic downloading of application components in a distributed system. As much, Java overcomes many of the important technical challenges posted by the diversity of underlying hardware and software technologies.

Applets are in fact small applications, which are supported by Internet servers, they can be transported over the net, automatically installed, and run as a part of a web document. Applets can even send messages to other applets, working in groups. At this moment, the most popular browsers which support Java Applets are Netscape Communicator, Microsoft Internet Explore and HotJava.

It's fun to write programs in Java, with the packages Java provides, one can develop very quickly and get satisfying results without a lot of pain. The standard compiler "javac" always gives accurate, sensible error reports to help you detect the bugs and eventually correct them[29].

But the most important strength of Java language might very well be the ability it has to be totally hardware independent. Today, operating systems, processor and core systems are upgraded so quickly, one cannot even be sure that today's program can be run tomorrow, even on the very same machine. But Java's "write once, run anywhere anytime, forever" philosophy makes sure that you will not have to face this situation, with other words, these applications can run anywhere there a Java Virtual Machine has been set up.

4 EXECUTION PIPELINE

4.1 STRUCTURE

The six steps of Execution Pipeline are shown in Figure 4: PS-fetch, Partition, Decoder, Dispatch/OP-fetch, ALU-operation and Commit/Write-back. Each step takes one clock period.

Figure 4 Execution Pipeline -- six steps

Execution Pipeline

PS-Fetch

Partition

Decode

Dispatch/OP-Fetch

ALU-operation

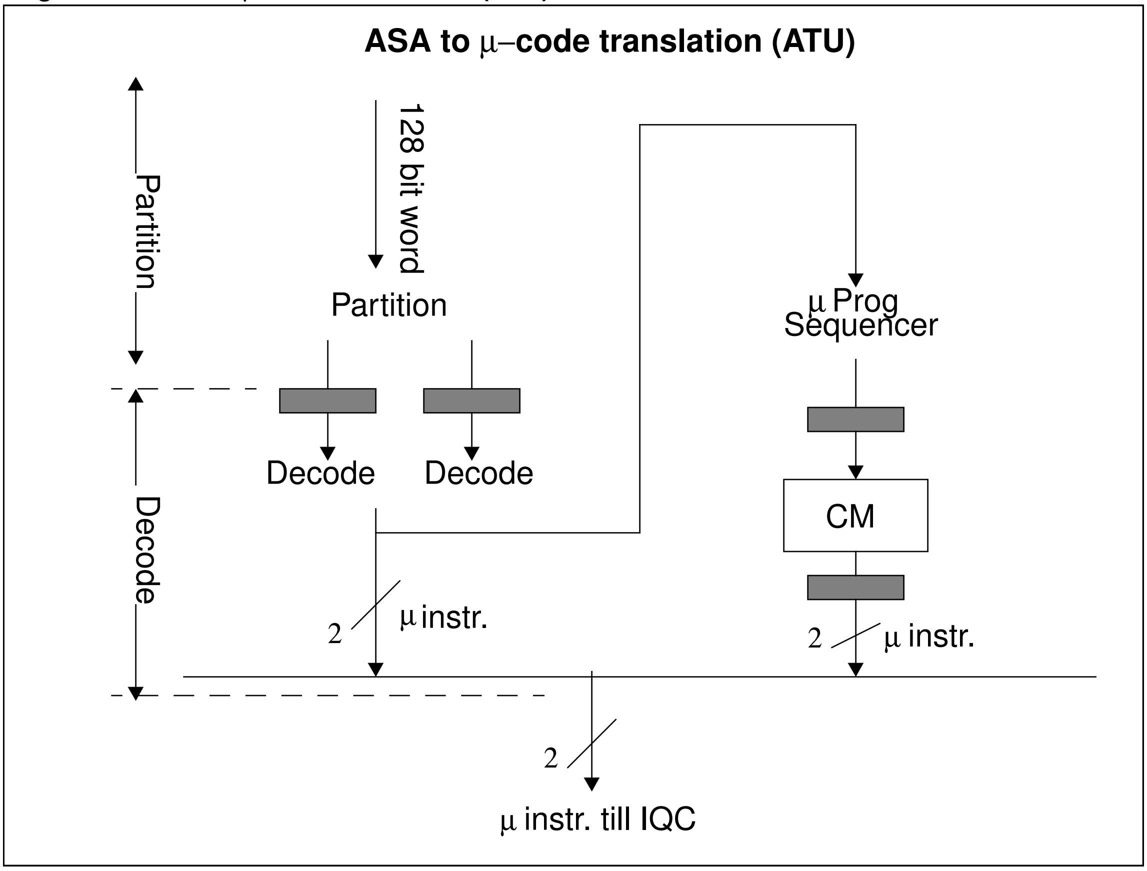
Commit/Write-back

As in Figure 5, ATU translates assembler code into micro code. ATU has a pipelined throughput of two ASA instructions in every cycle. The ATU receives the words from PRSU and sends them to Partition. It will partition the words into ASA instructions and deliver them to Decode. Instruction Decoder decodes the ASA instructions. It means that it looks up the first MIP word of the ASA instruction, looks up the CM address of multi MIP word ASA instructions and extracts the parameters of the ASA instructions.

The Micro Program Flow Handler(ATMFH) handles the CM and is only involved in multi MIP words instructions and different exception cases. If an ASA instruction is built up of more than one MIP word, the first MIP word is delivered from ATIDEC and the rest of the MIP words are delivered by ATMFH.

ATDELIVERY selects the source of the MIP word instructions. These instructions are then distributed to IQC and insert parameters, including trace flags, according to the MIP words into the source- and destination fields. If a multi MIP words instruction is detected, ATMFH is selected. The original ASA parameters in a multi MIP word instruction are available in ATDELIVERY during the whole instruction. ATDELIVERY inserts the correct parameters into the specified source or destination fields of the micro code before delivering it to IQC.

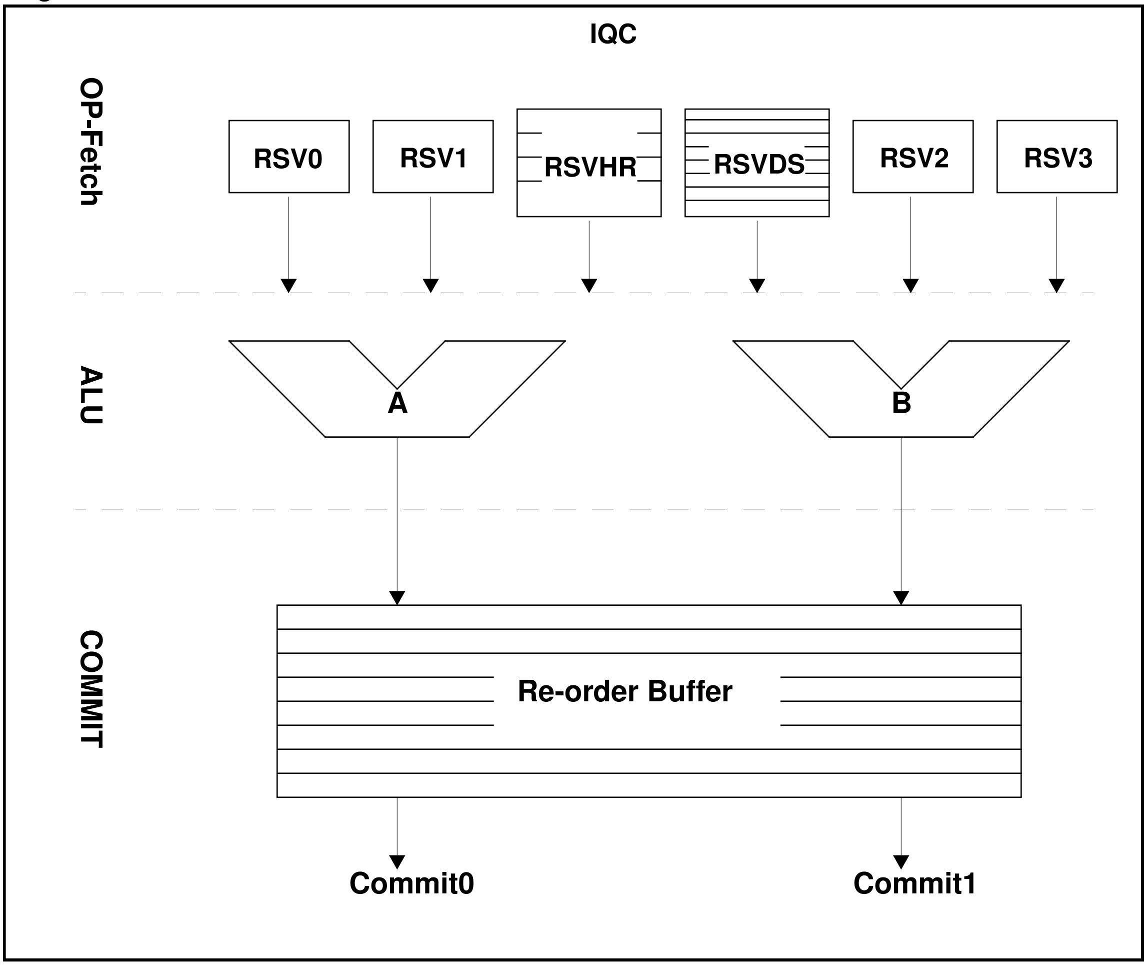
Figure 5 ASA to μ-code translation (ATU)



It is possible to partition two ASA instructions in each clock period, decode two ASA instructions and look-up two MIP words in each path. At the same time it is possible to look-up two MIP words in CM and distribute two instructions to IQC.

The IQC executes the decoded micro-instructions delivered from ATU. Two decoded micro-instructions can be delivered, executed and the result is written back in each clock cycle that makes this machine as two-way superscalar. The pipeline shows in Figure 6.

Figure 6 IQC



The IQC is pipelined. It means that new decoded microinstructions can be fetched before the previous fetched microinstructions are finished. The IQC controls three pipeline steps. For each decoded microinstruction delivered by the ATU, the IQC will fetch constants from the decoded microinstruction, source data from the data storage, data from the register memory unit or data from one of the hardware control and status registers. This is done in the first pipeline step. The source data is executed in one of two 32-bit arithmetic logic units, ALU in the second step and written back to the destination, which is DS, RMU or HR in the third pipeline step.

The result from the instructions that are executed is called destination data and is stored in the re-order buffer. Destination data which can not immediately be committed are the out-of-order and speculative executed instructions. Up to two results in the re-order buffer can be written back in each clock cycle.

Dispatch/Reservation Stations: in each clock cycle the OP-fetch receives two, one or zero instructions from Decode. The number of instructions that can be received depends on two circumstances. The first is if the Decode is able to deliver instruction and second if the RS is ready to receive instructions. In this case, it is assumed that there are always instructions that are able to be delivered from Decode. The Dispatch only receives one instruction at a time. And all reservation stations must be empty before a new instruction can be received.

The clock selects block controls -- ALU MUX which passes data from a reservation station where the data is fetched. The reservation stations send ready signals to the ALU select block when the data is ready.

During the processing of an instruction, the main part of the instruction context is always kept in the Reorder Buffer. The reorder buffer has space for up to 8 instructions. It can receive two, one or no new instructions in each cycle. It can also receive two results of out-of-order from ALU and one address calculation indication from ACQ. Up to two instructions can be committed in each cycle.

4.2 MAIN FUNCTIONS

4.2.1 Reset

At reset all state machines are set to their initial state. All buffers are emptied.

4.2.2 PS-fetch

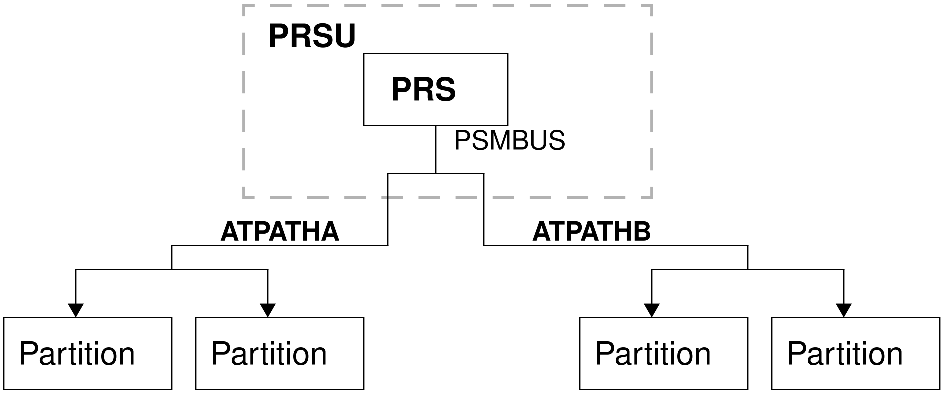
PS-Fetch fetches 128 bits word, one per cycle. Incoming words from PRSU to ATU come on bus PSMBUS[127:0].

There are two prefetch paths, ATPATHA and ATPATHB, which fetch instructions from PS.

4.2.3 Partition

There are two partitions proceed concurrently. They receive words from ATPATHA and ATPATHB, as in Figure 7. ATPATHA partitions the word into ASA instructions and delivers them to ATIDEC(Instruction Decoder).

Figure 7 PS-Fetch to Partition



ATPARTDH gets Program Store words of 128 bits from PRSU, one per cycle, extracts ASA instructions and delivers them to ATIDEC. Two instructions can be delivered per cycle.

4.2.4 Delivery

The ATIDEC block receives ASA code from ATPARTDH. Two instructions can be decoded in parallel.

ATIDEC translates the OP-code of each ASA instruction to the ICM address of the instruction and performs a look-up of the first MIP word in Internal Control Memory(ICM). At the same time the instruction is decoded.

In one cycle parameters that may change the program flow, n-and ssp-parameters, and parameters used for data store access, a- and vln-parameters, are decoded.

Decoded is also if the n-parameter comes from a conditional or unconditional jump instruction and if the ssp-parameter comes from a buttered or unbuffered signal sending instruction.

A special decode is done if the a-parameter comes from a new type of dynamic buffer instruction introduced in 212 30.

Information if an instruction changes the link register stack, LRS, is decoded as well as if the instruction is EP or not.

Table jumps and other instructions causing a halt in the program flow are also decoded.

Also during the clock cycle the operation code which can be from 4 to 12 bits is translated to a 9 bits address in IDEC Control Memory, ICM, where the first MIP word for every instruction is stored. Instructions with operation codes shorter than 9 bits will use these bits as address with the most significant bits set to zero. Operation codes longer than 9 bits are re decoded to 9 bits and for instructions with 9 bits operation code this code is the same as the address.

For supporting ASA instructions that are translated into many microinstructions the IPU includes a larger 16 k * 60 bits large Control Memory (CM) and a microprogram sequences for supporting microprogram jump, subroutines etc. The Control Memory holds microprogram code for Initial Test (ITEST), all MIP for instruction execution and MIP for maintenance during operation. The ITEST program will be temporary loaded during initialization if it does not fit into CM.

Before the MIP words are sent to IQC the parameters from the ASA code are inserted in the source and destination fields, i.e. ATDELIVERY.

4.2.5 Dispatch/OP-Fetch

There are six reservation stations, RSV0, RSV1, RSV2, RSV3, RSV HW and RSV DS. Four of them receives RM data and one receives HR data and one DS data. Two of the four that receives RM data receives MIP0 and the other two receives MIP1. The reservation stations for HR and DS receives both MIP0 and MIP1. It is the dispatch block which determines which reservation station every instruction goes to. The reservation station picks the operand data from RM, HR or DS. If the dispatch or the re-order buffer detects a data dependency the operand data can also be picked from the ALU or from the re-order buffer. The reservation station also decodes if the operand is a constant and in that case load the constant from the MIP word.

4.2.6 ALU

When data is ready in any of the six reservation stations an ALU operation takes place. Two ALU operations can take place in parallel. ALU-A can take operand data from MIP0 RM instructions and HR instruction. ALU-B can take the operands from MIP1 RM and from DS.

4.2.7 ROB and Commit

All results from the ALUs are written to the re-order buffer.

When the least recently received instruction has its data ready it is written back to its destination. The destination can be RM, HR or DS. Up to two instructions can be committed in parallel on the two commit buses DEST0 and DEST1. There is a logical order between them DEST0 is the first and DEST1 is the second instruction result. If there is two simultaneous commit to the same register the data on DEST1 should be used. If the destination address is a FIFO DEST0 should be put before DEST1 in the FIFO. DEST0 is connected to all the destinations. DEST1 is connected to RM and SPI which is a part of HR.

5 SIMULATION

5.1 GENERAL

Execution Pipeline Visualization is an application that shows the flow of instructions in the execution pipeline. The execution pipeline has six steps: PS-fetch, Partition, Decoder, Dispatch/OP-fetch, ALU-operation and Commit/Write-back. The first three steps are executed in ATU, which sends instructions to IQC. So in IQC, Dispatch/OP-fetch, ALU-operation and Commit/Write-back are executed.

5.2 SCOPE

Based on the Design Specifications “Instruction Queue Controller, IQC" and "Assembler to micro code Translator Unit, ATU", this application shows the flow of instruction in the execution pipeline through six steps. In these six steps, the information is picked up from PS-Fetch, Partition, Decode, CM, ICM, Delivery, RS, ALU, ROB and Commit. Information from DS, PRS, Unrolling and SPU is not interested in this project. It shows a super-scalar execution, double partitioning and double decoding.

5.3 THE OUTPUT FILE

Blocks ATU and IQC are designed in VisualHDL system for Verilog that provides system designers, ASIC designers and simulation model develops with a complete design environment [19].

The simulation runs under the IPU simulation environment which provides a framework for performing functional verification of almost all functions in the IPU. A script controls which modules, libraries and support code that will be included. A special designed file writes the output of each step to a log file. This file is named by "IPU_supervision", which is shown in Appendix E.

File "IPU_supervision" is written in Verilog. This file creates a log_file, picks up the information from simulation and writes them to the log_file. This log_file is named as "Execution_data". "Execution_data" is a standard output file. Figure 8 shows a part of the log_file, data from simulators at clock cycle 70.

Figure 8 The output data

Clock N	**70**
Block F (fetch)	**PSMBUS: 0000000021c31d831943150321c31d83**
Block P (partition)	**PATHA-ASAX: 150310c30c83** **PATHA-ASAY: 1943150310c3** **PATHB-ASAX: 000000000000** **PATHB-ASAY: 000000000000**
Block D (decode/delivery)	**DECODE-AX: 0c8308430403** **DECODE-AY: 10c30c830843** **DECODE-BX: 000000000000** **DECODE-BY: 000000000000** **CM: 00000000000000** **ICM: 000000000000000000** **DELIVERY0: 400f** **DELIVERY1: 400f**
Block R (reservation stations)	**RSV0: 00000000** **SN0: 00** **RSV1: 00000000** **SN1: 00** **RSV2: 00000000** **SN2: 00** **RSV3: 00000000** **SN3: 00** **Op0: 00000000 SN 00** **Op1: 00000000 SN 00** **Op2: 00000000 SN 00** **Op3: 00000000 SN 00** **Op4: 00000000 SN 00** **Op5: 00000000 SN 00** **Op6: 00000000 SN 00** **Op7: 00000000 SN 00** **HRdata0: 00000000 SN 00** **HRdata1: 00000000 SN 00** **HRdata2: 00000000 SN 00** **HRdata3: 00000000 SN 00**
Block A (alu)	**FromALUA: 00000000** **SN: 00** **FromALUB: 00000000** **SN: 00**
Block C (commit)	**ROBPOS1: 00** **ROBPOS2: 00** **CommitA: 0000** **IQV0=0** **CommitB: 0000** **IQV1=0**

In Figure 8, clock N is a clock counter. It shows the clock cycle to current data. The data is divided into six blocks as six steps. First block contains the data from PS-fetch. Second block is block P that shows data from Partition. Block D gets the data from Decode, CM/ICM and Delivery. Block R shows the data of reservation stations. Block A shows the data from ALU. Block C has the Commit data. Each block takes one clock cycle.

Each data line contains two parts: one is the name of the data, the other is the data itself.

Since this execution is pipelined, each step executes in parallel. Figure 9 is a data flow chart. For example:

Step 1: PS-fetch fetches 128 bit word from PRSU. It shows in the block P at clock 69. At the same time block P shows the partitioned data result from the block P at clock 68, and so on.

Step 2: Partition partitions the word into ASA instructions which are shown in the block P at clock 70.

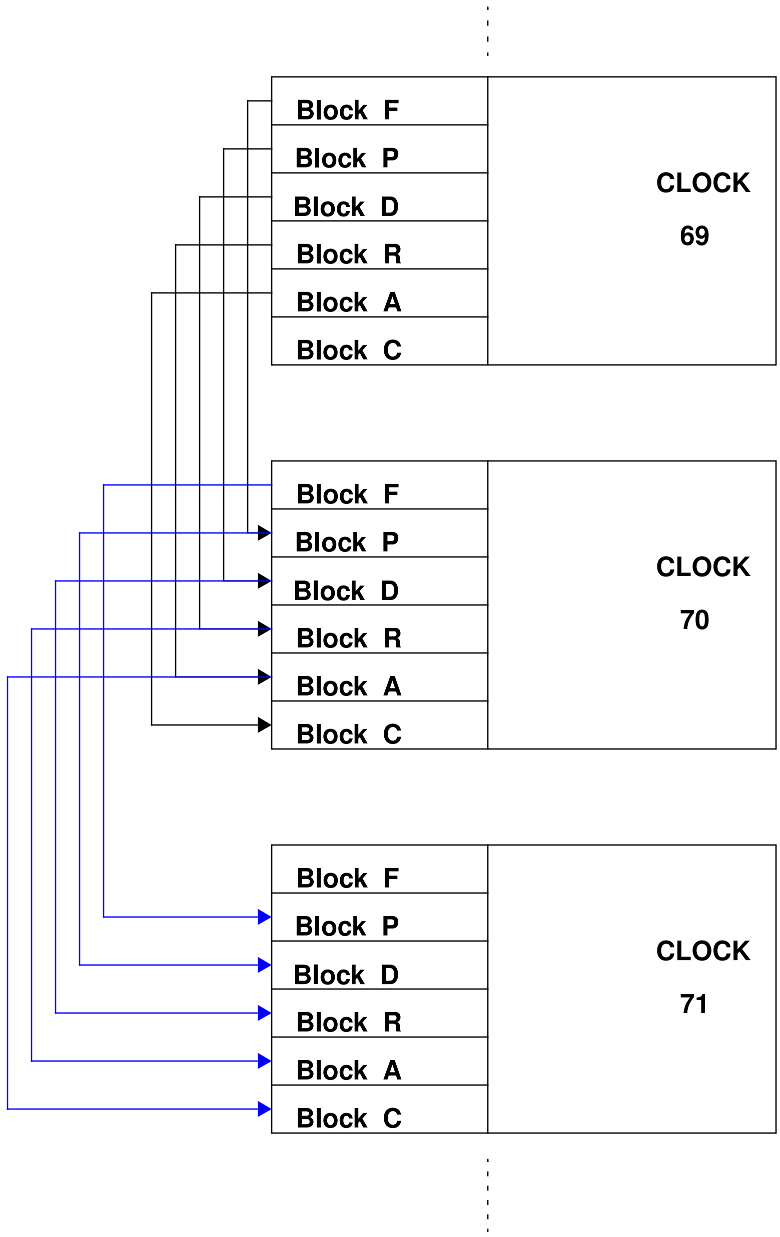
Step 3: Decoded instructions are shown in the block D at clock 71.

Step 4: MIP words are delivered by ATU. The block R shows the information of reservation stations at clock 72.

Step 5: The block A shows the executed instructions at clock 73.

Step 6: The results of execution are shown in the block C at clock 74.

Figure 9 Data Flow Chart



To get the data from each step, the bus needs to be known where the data come from. Following are the buses that IPU_supervision gets information from. The names of the buses are the data that come from the simulation. As I mentioned before, IPU_supervision creates a log_file and writes the data to the log_file.

PS-Fetch fetches 128 bit word from *IPC.PRSU.PRSUCORE.PRMH.PSMBUS*

Partition has two paths. ASA instructions which are partitioned on path A are:

IPC.ATU.ATUCORE.ATPATHA.PATH_A.ASAX
IPC.ATU.ATUCORE.ATPATHA.PATH_A.ASAY

ASA instructions which are partitioned on path B are:
IPC.ATU.ATUCORE.ATPATHB.PATH_B.ASAX
IPC.ATU.ATUCORE.ATPATHB.PATH_B.ASAY

Instruction Decoder decodes the instructions from path A:
IPC.ATU.ATUCORE.ATPATHA.PATH_A.ATIDEC.ATIDECREGX.ASAD1X
IPC.ATU.ATUCORE.ATPATHA.PATH_A.ATIDEC.ATIDECREGY.ASAD1Y

For path B, instructions are decoded:
IPC.ATU.ATUCORE.ATPATHB.PATH_B.ATIDEC.ATIDECREGX.ASAD1X
IPC.ATU.ATUCORE.ATPATHB.PATH_B.ATIDEC.ATIDECREGY.ASAD1Y

CM and ICM show the MIP word on:
IPC.ATU.ATUCORE.ATMFH.ATCMH2.CMWD
IPC.ATU.ATICMBLOCK.ICMRDAB

The final instructions which will delivery to IQC are: *IPC.ATCMAB0* and *IPC.ATCMAB1.*

In IQC, first step OP-fetch has six reservation stations. RS zero to four receive RM data, included sequence numbers:
IPC.IQC.IQRSV.IQRSVRM.IQR0.RSOPA_REG0
IPC.IQC.IQRSV.IQRSVRM.IQR0.RSSN0
IPC.IQC.IQRSV.IQRSVRM.IQR1.RSOPA_REG1
IPC.IQC.IQRSV.IQRSVRM.IQR1.RSSN1
IPC.IQC.IQRSV.IQRSVRM.IQR2.RSOPA_REG2
IPC.IQC.IQRSV.IQRSVRM.IQR2.RSSN2
IPC.IQC.IQRSV.IQRSVRM.IQR3.RSOPA_REG3
IPC.IQC.IQRSV.IQRSVRM.IQR3.RSSN3

RS DS receives DS data, 8 operands:
IPC.IQC.IQRSV.IQR5.OPA0, IPC.IQC.IQRSV.IQR5.SN0
IPC.IQC.IQRSV.IQR5.OPA1, IPC.IQC.IQRSV.IQR5.SN1
IPC.IQC.IQRSV.IQR5.OPA2, IPC.IQC.IQRSV.IQR5.SN2
IPC.IQC.IQRSV.IQR5.OPA3, IPC.IQC.IQRSV.IQR5.SN3
IPC.IQC.IQRSV.IQR5.OPA4, IPC.IQC.IQRSV.IQR5.SN4
IPC.IQC.IQRSV.IQR5.OPA5, IPC.IQC.IQRSV.IQR5.SN5
IPC.IQC.IQRSV.IQR5.OPA6, IPC.IQC.IQRSV.IQR5.SN6
IPC.IQC.IQRSV.IQR5.OPA7, IPC.IQC.IQRSV.IQR5.SN6

RS HR receives HR data:
IPC.IQC.IQRSV.IQR4.OPA_DATA0, IPC.IQC.IQRSV.IQR4.SN_DATA0
IPC.IQC.IQRSV.IQR4.OPA_DATA1, IPC.IQC.IQRSV.IQR4.SN_DATA1
IPC.IQC.IQRSV.IQR4.OPA_DATA2, IPC.IQC.IQRSV.IQR4.SN_DATA2
IPC.IQC.IQRSV.IQR4.OPA_DATA3, IPC.IQC.IQRSV.IQR4.SN_DATA3

Here are ALUs operations and sequence numbers:
IPC.IQC.IQAL_ALUS.ALRESA, IPC.IQC.IQAL_ALUS.ALASN
IPC.IQC.IQAL_ALUS.ALRESB, IPC.IQC.IQAL_ALUS.ALBSN

The results are shown in ROB:
IPC.IQC.IQROB.ROBSN0 and IPC.IQC.IQROB.ROBSN1.
The destination data on Commit:
IPC.IQCMAB0_A and IPC.IQCMAB1_A.

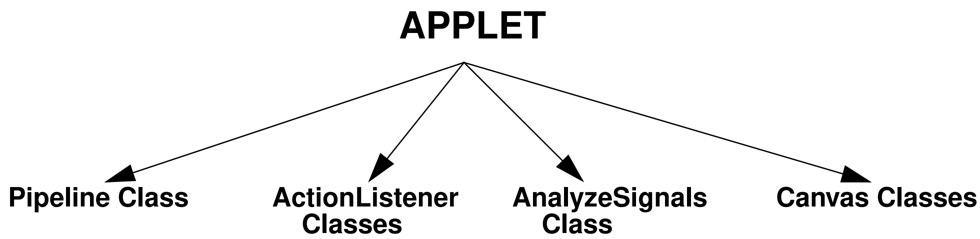
6 VISUALIZATION

6.1 STRUCTURE

The applet class Pipeline is the main class. The whole layout and events handling are defined in the main class.

Class AnalyzeSignals handles the log_file. It reads log_file and puts them to the TextFields. Several Canvas classes draw lines and arrows between panels. Figure 10 explains the whole structure of the applet classes.

Figure 10 Structure of the applet classes



All Java code is listed in Appendix B.

6.2 GUI LAYOUT

GUI design for Visualization Execution Pipeline is quite simple. A usual attribute is represented with Label, TextField/TextArea and Button. It has two parts.

First part is a Menu Bar. It contains four buttons and one TextField which is editable. For buttons, the background colour is set to yellow and foreground to red.

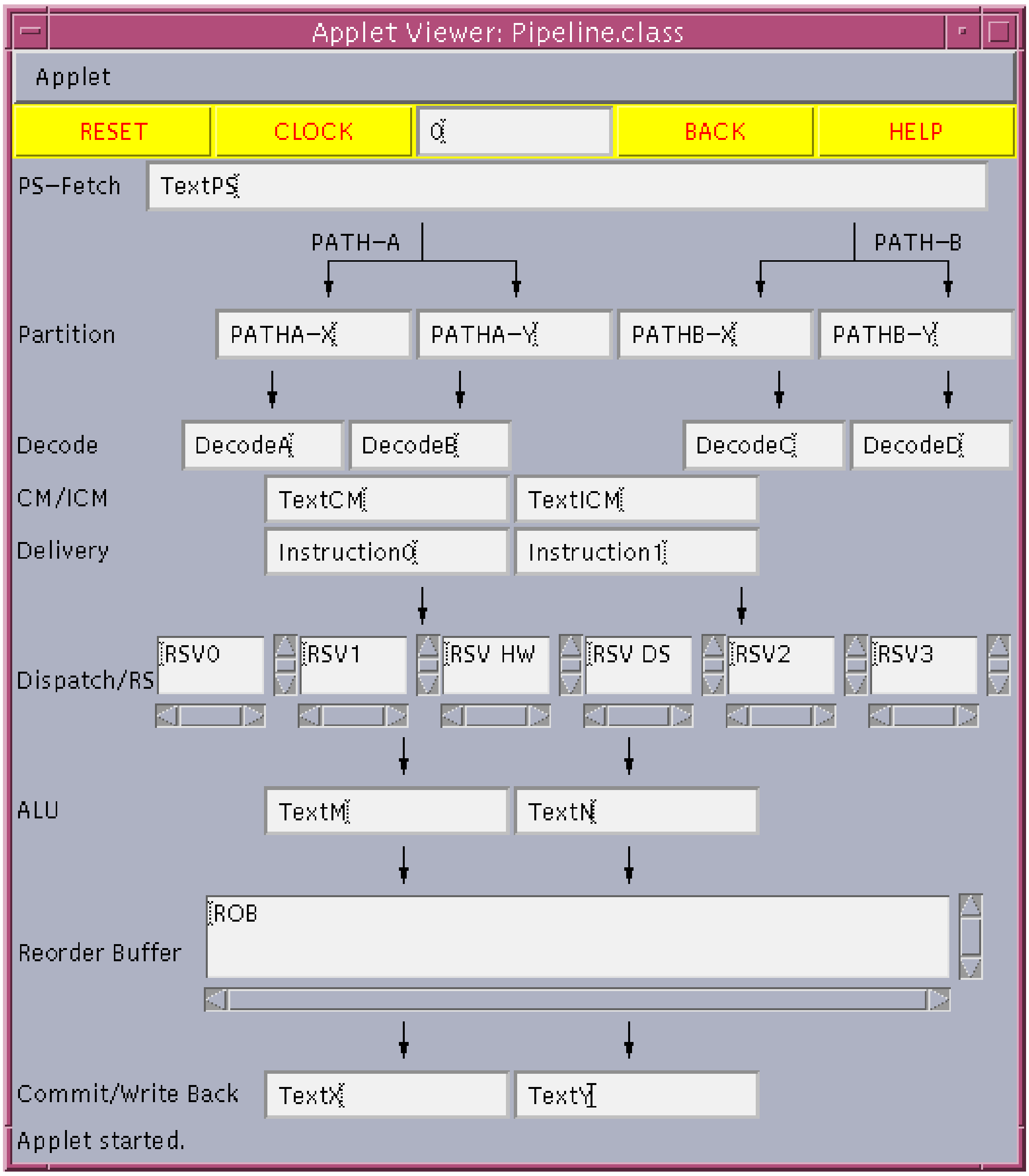
Second part is Labels and Textfields. Those TextFields are non-editable, just for read-only. On Dispatch/RS, TextAreas is used instead of TextFields, since more than one data are needed. TextFields is used at beginning, while the text in TextField can only be put in one line. Such that different data can be displayed on different lines in the TextAreas.

Between steps, lines and arrows are drawn to show the pipelining.

In the whole view of user interface, GridBagLayout is used as Layout Manager to constrain the panels. And for each panel, GridLayout is used to constrain every component to certain position.

Figure 11 shows the GUI Layout of the applet.

Figure 11 GUI Layout of Applet. This is the default layout.



6.3 EVENT HANDLING

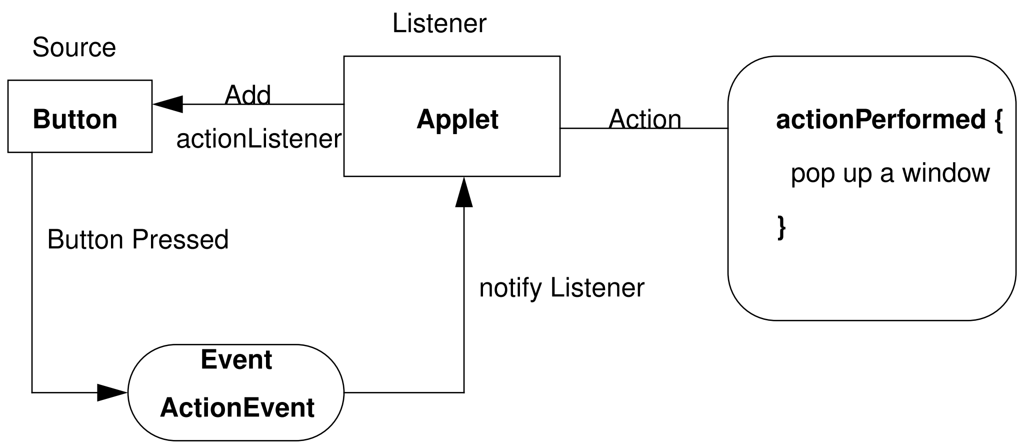
There are two kinds of events in the applet. One is for Buttons and one is for TextFields.

At MENU Bar, there is a TextField. This TextField is editable. When one write a cycle N and then press RETURN, ActionEvent is started and actionListener will react with actionPerformed method. The applet shows the data at cycle N.

When Button is presses, ActionEvent is fired and Button’s actionListener will react with actionPerformed method.

Figure 12 is the example of Help button. When Help button is pressed, it will pop up a window.

Figure 12 Event handling for Help button



6.4 MAIN FUNCTIONS

6.4.1 Reset

The Reset button sets the applet to the default state, i.e. all text fields are emptied and the TextCLOCK will set to 0.

6.4.2 Clock, TextCLOCK and Back

When the Clock button is pressed, the applet reads the data from file "execution_data", and puts them to the text fields. Meanwhile it counts the cycles and displays at TextCLOCK. The Back button will show data when the cycle is clock - 1. If one wants to see the data under a special cycle n, he just writes the number N into the text field TextCLOCK, which will show the data at cycle N.

6.4.3 Help

The Help button will pop up a window of text which explains the functions of the buttons.

6.5 READ DATA

As we see Figure 9, "Execution_data" is a text file. Each line has only one data information: the name of data and data itself. Thus in Java programming, I can easily use "readLine" for reading the data. I use StringTokenize to split the line to two parts and put the second one into the TextField.

The following is the Java code:

```
line = in.readLine();
st = new StringTokenize(line);
token1 = st.nextToken();
token2 = st.nextToken();
pipeline.textPATHX.setText(token2);
```

First "line" reads a line from log_file. Then object "st" has all the information of one line. Method "nextToken" splits the "st" to two tokens, token1 is the name of data and token2 is the data. Finally data -- token2 is put to the text field. Here, pipeline is an applet , textPATHX is an object of textField, setText is a method of TextField.

6.6 RUN APPLET

PS-fetch fetches the assemble code and visualizes in the text field. It takes one cycle.

Partition has two paths from PS-Fetch. Path A sends the partitioned data to both PATHA-X and PATHA-Y while Path B sends to PATHB-X and PATHB-Y. It takes one cycle.

It takes also one cycle for showing on Decode, CM/ICM and Delivery. All the data are stand by untill the IQC finishs execution. Delivery shows the final information which will be sent to IQC.

Reservation stations show micro instruction from ATU and operands HR, DS. It takes one cycle.

ALU executes instructions and operands from Dispatch. It takes one cycle.

All the results send thereafter to ROB which will be read by Commit/Write back. It takes one cycle.

Applet has a SnapShot to crab blocks from "execution_data". The size of the SnapShot is 6 blocks. Figure 13 describes how the applet shows data by using SnapShot. At clock 1, SnapShot crabs only one block of data and shows it in the block F. At clock 2, SnapShot crabs two blocks of data and puts the first block of data in the block F and the second in the block P. The data of block P is the partitioned data of the block F at clock 1. At clock 3 SnapShot crabs three blocks of data and puts them in the blocks F, P and D. At clock 4 SnapShot crabs four blocks of data and puts them in the blocks F, P, D and R. At clock 5 SnapShot crabs five blocks of data and puts them in the blocks F, P, D, R and A. At clock 6 SnapShot crabs six blocks of data and puts them in the blocks F, P, D, R, A, and C. So for one execution line, here is the flow of execution pipeline:

F(Clock 1) -> P(Clock 2) -> D(Clock 3) -> R(Clock 4) -> A(Clock 5) -> C(Clock 6)

Figure 13 Data Flow

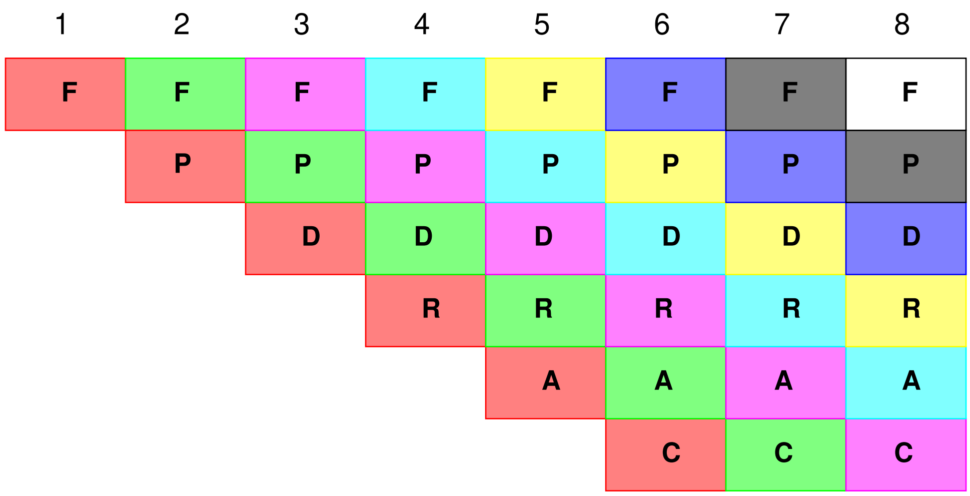
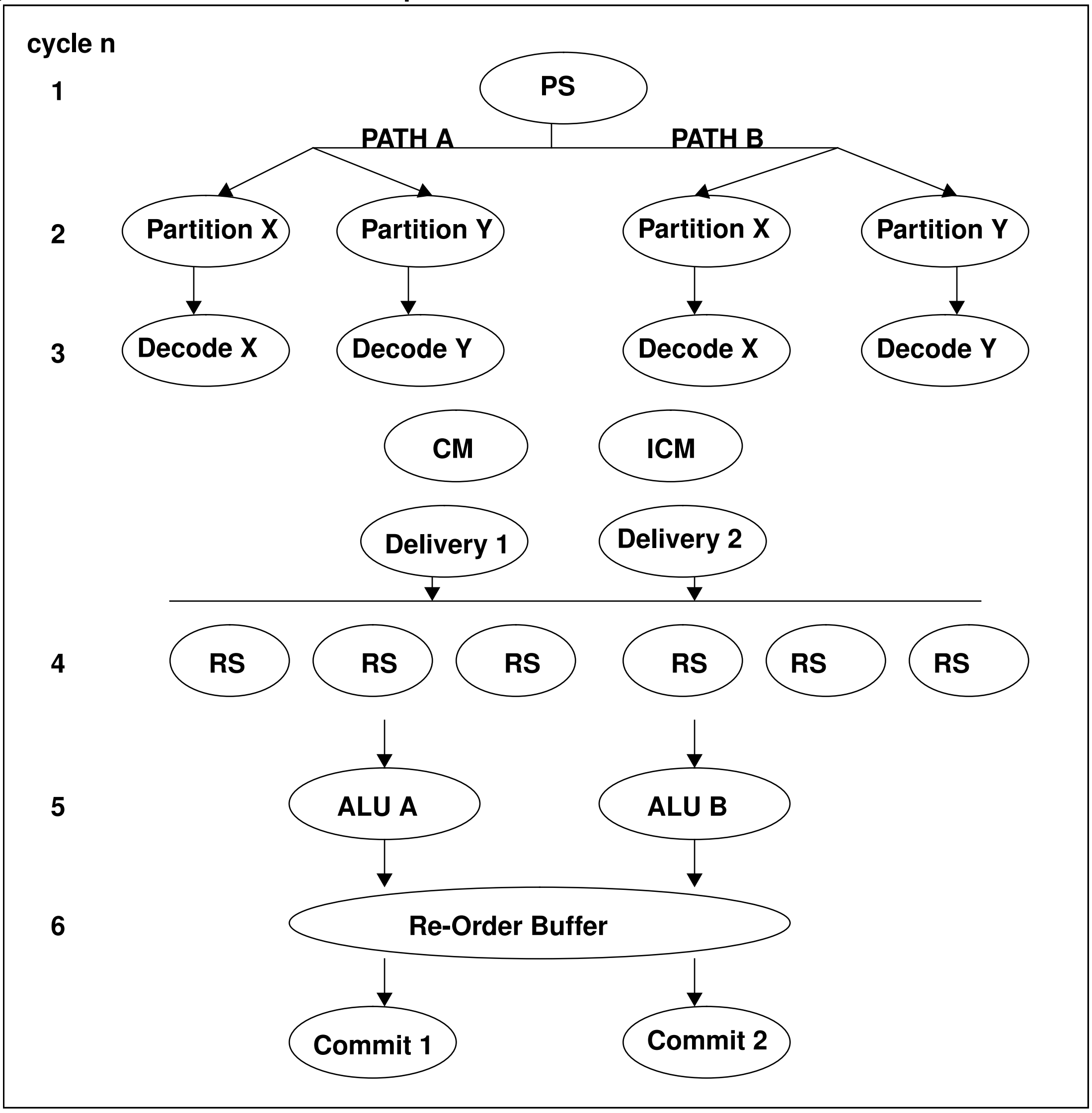


Figure 14 describes the flow of execution pipeline. It has tree structure. At clock 1, it shows only on first circle; at clock 2, it shows data on second level, and so on. It takes six clocks to fill all circles.

Figure 14 The Flow of Execution Pipeline



6.7 SCRIPT

The Common Gateway Interface is a method that lets Web pages communicate with programs on an HTTP server. Prior to the advent of Java, this was the only way to get interactive Web pages: An HTML page sent limited information back to the server, which generated a new HTML page and returned it to the client to display. Java replaces some uses of CGI by allowing truly dynamic Web pages, doing the processing on the client instead of the server.[25]

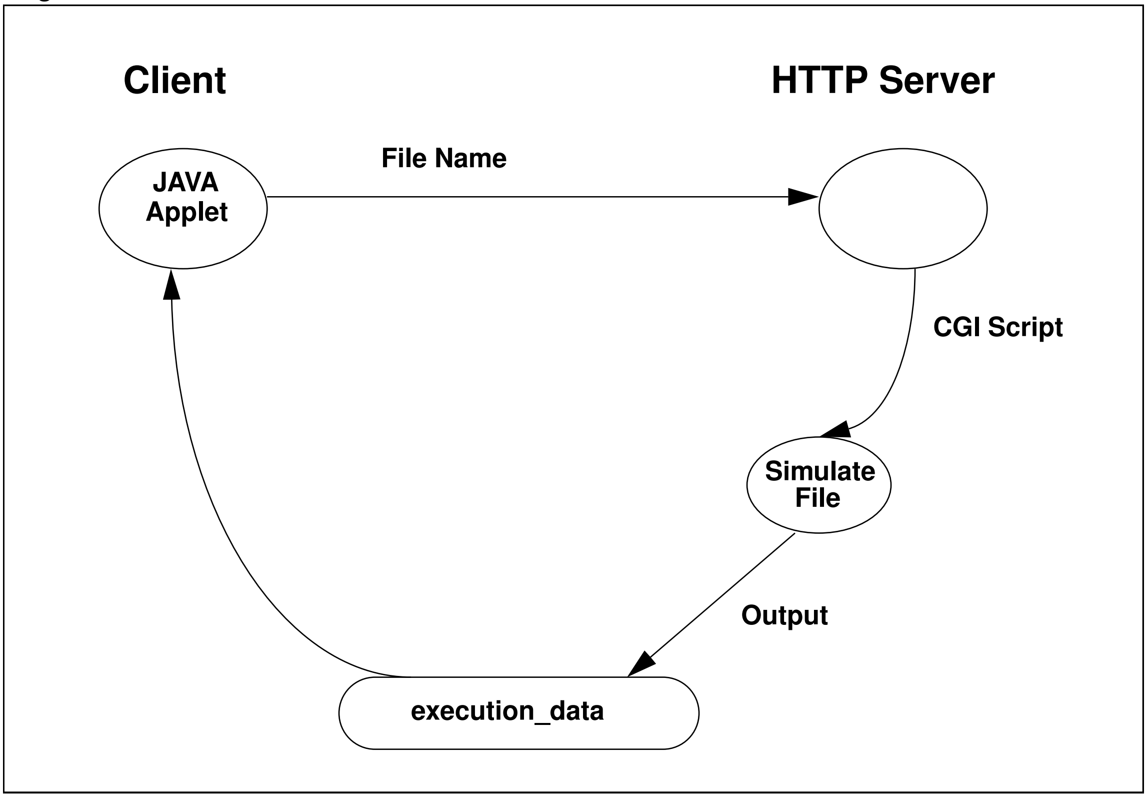
Appendix D. is a CGI script in Perl. This file is trying to start simulator VCS to simulate a file which is given by user.

Figure 15 shows how a client and a server are related. When a client wants to simulate, it connects to server. The server starts a CGI script. The script executes the simulation model and writes the results to file "execution_data". Then client can read from the output file and show the results on applet.

The benefits of using an applet and a web sever are:

1 The applet can choose any simulator, doesn't matter how many and how different.
2 Anybody can change the simulator then execute again.
3 It supports that more than one person can run the applet at same time.

Figure 15 Client and Server



7 CONCLUSION AND FURTHER WORK

7.1 CONCLUSION

The aim of this project is to visualize the flow of instructions in the execution pipeline, i.e. to provide a graphical tool to visualize how a pipelined datapath and control unit work. The graphical user interface shows what happens to an instruction at each stage of the pipe.

As in Figure 11, the applet makes it possible to examine the dataflow in the datapath caused by the instruction. By clicking the Clock button with the mouse, instructions are forwarded one step in the pipeline.

There are several things I have learned during the mapping process. First of all, before I dive into implementation, it is always a nice and necessary thing to have a sketch over the objects, (or components) I am going to use. Chapter [30]

Second, one should try to discover and use the common parts of the components as much as possible, to try to deal with each single component in a different way is a waste of both time and resource.

In my project, I created a log_file to put all output into it. When I created this log_file, I had to concern that this file would be read by an applet which is written in Java. That's why I put the output line by line.

In the applet there are plenty of components. How to manage them in a nice manner was my major problem at the beginning. After studying another examples and Java books, I found a good way to put them all together. I used GridBagLayout which is a powerful and confusing Layout Manager[21].

Since the applet reads , writes and simulates, I had to create a CGI Script as well. This script file connects the client and server. It is possible that several persons can start the applet and simulate different models at the same time.

7.2 FURTHER WORK

My thesis work is only concerned six steps pipelining within two units in IPU. As in Figure 1, we can see that IPU includes more units. The applet can let user to examine the dataflow step by step. In the future, I believe that we can develop even more advanced graphical tool which lets a user to examine each stage in the pipeline of every unit.

Appendix A. References and Bibliography

[1] David A. Patterson and John L. Hennessy. [1996] *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, INC. USA. ISBN 1 - 55860 - 329 - 8.

[2] D. W. Anderson, F. J. Sparacio and R.M. Tomasulo. [1967] T*he IBM System/360 Model 91: Machine Philosophy and Instruction-handling.* IBM Journal, 11:8 - 24.

[3] Per Stenström, Håkan Nilsson and Jonas Skeppstedt. [1993] *Using Graphics and Animation to Visualize instruction Pipelining and its Hazards.* Proceedings of ICSEE'93, January.

[4] Michael J. Flynn. [1995] *Computer Architecture: Pipelined and Parallel Processor Design*. Jones and Bartlett Publishers. USA. ISBN 0 - 86720 - 204 - 1.

[5] John L. Hennessy and Norman P. Jouppi. [1991] *Computer Technology and Architecture: An Evolving Interaction*. IEEE Computer, 24(9):18 - 29.

[6] John L. Hennessy and David A. Patterson. [1994] *Computer Organization and Design: the hardware/software interface.* Morgan Kaufmann Publishers, INC. USA. ISBN 1 - 55860 - 281 - X.

[7] Kenny Ranerup and Lars Philipson. *Optimization of finite state machines using subroutines*. Department of Computer Engineering, Lund University, Sweden.

[8] Kenny Ranerup. *Control Architecture Selection From State Graph Characteristics. Department of Computer Engineering, Lund University, Sweden.*

[9] Lars Philipson, Anders Ardö and Kenny Ranerup. *Design of a Microprocessor and its Software as a Seven Week Class Project.* Department of Computer Engineering, Lund University, Sweden.

[10] Johan Wennlund. [1997] *Något om RISC -- Reduced Instruction Set Computer*. Department of Teleinformatics, Royal Institute of Technology, Sweden.

[11] Norman P. Jouppi and David W. Wall. *Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines.* Digital Equipment Corporation, Western Research Lab. USA.

[12] L-E Thorelli. [1996] *The DLX Architecture*. Department of Teleinformatics, Royal Institute of Technology, Sweden. Based on: Anthony P. Reeves. [June 1997] *The DLX Architecture*. EE Department, Cornell University, USA. http://www.ee.cornell.edu/~reeves/ee308

[13] Lars Winberg. [1998] *Instruction Queue Controller, IQC.* Internal document, B/SHI, Ericsson AXE Research and Development. Sweden. 102 60-COA 101 1143.

[14] Johan Parin. [1996] *IPU Simulation Environment.* Internal document, B/SHI, Ericsson AXE Research and Development. Sweden. 8/10264-CNZ 211 390.

[15] XKKDANH. [1998] *Assembler to micro code Translation Unit, ATU.* Internal document, B/SHI, Ericsson AXE Research and Development. Sweden. 02 62-COA 101 1123.

[16] XKKDANH. [1997] *Micro format for IPU in APZ 212 30.* Internal document, B/SHI, Ericsson AXE Research and Development. Sweden. 1/102 62-CNZ.

[17] Terje Egeland. [1995] *APZ 21220 - The New High-end Processor for AXE 10.* Ericsson Review N01 1995.

[18] Per Holmberg. [1996] *Instruction Processor Unit, IPU.* Internal document, B/SHI, Ericsson AXE Research and Development. Sweden. 102 62-CNZ 211 390.

[19] Summit Design, INC. [1994]. *Visual HDL for Verilog on UNIX, User's Guide. Edition 3.0 for UNIX*. Summit Design, INC.USA.

[20] Cadence Design, INC. [1991]. *Verilog-XL Reference Manual, Version 1.6.* Cadence Design, INC.USA.

[21] David Flanagan. [1997]. *JAVA In a Nutshell: a desktop quick reference.* O'Reilly & Associates, INC. USA. ISBN 1-56592-262-X.

[22] Ken Arnold and James Gosling. [1998]. *The JAVA Programming Language Second Edition.* An Imprint of Addison Wesley Longman, INC. USA. ISBN 0-201-31006-6.

[23] Bruce Eckel. [1998]. *Thinking in JAVA.* Prentice-Hall INC. USA. ISBN 0-13-659723-8.

[24] About APACHE. http://www.apache.org

[25] Marty Hall. [1998]. *Core Web Programming.* Prentice Hall INC. USA. ISBN 0-13-625666-x.

[26] Daniel Griscom. *Code Signing for Java Applets.* http://www.suitable.com/Doc_CodeSigning.shtml.

[27] Peter van der Linden. *Java Programmer FAQ.* http://www.afu.com/javafaq.html.

[28] http://www.java.sun.com

[29] Sun Microsystems. *The Java Tutorial Object-Oriented Programming for the Internet.* http:// java.sun.com/docs/books/tutorial/index.html

[30] *IDL/Java code mapping.* http:// splash.javasoft.com/JavaIDL-alpha2.0/pages/release-notes-win-x86.html

Appendix B. JAVA Program Code

Pipeline.java

```
/*
 * Name:     Execution Pipeline
 * Designer: Jenny Liu
 * Version:  1.0
 * Date:     Nov. 1998
 * File:     /home/uabjenn/java-prog/Pipeline/
 */
/*
 * Pipeline.java creates an applet which
 * shows the execution pipeline
 */

import java.applet.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.lang.*;
import java.util.*;

import PSCanvas;
import PARTCanvas;
import DisCanvas;
import AnalyzeSignals;

/******  An applet, interface  ******/
public class Pipeline extends Applet  {

    PSCanvas    canvPS;
    PARTCanvas  canPART;
    LineCanvas  canv;
    DisCanvas   canDis;

    Button resetB, clockB, backB, helpB;          // buttons in menu bar
    Panel panel;
    TextField textCLOCK;                          // text fields for cycle counting
    TextField textPS;                             // text fields for PS-Fetch
    TextField textPATHAX, textPATHAY, textPATHBX, textPATHBY; // textfields for partition
    TextField textDeA, textDeB, textDeC, textDeD;  // text fields for decode
    TextField textCM, textICM;                    // text fields for CM and ICM
    TextField textInstr0, textInstr1;             // text fields for delivery
    TextArea  textJ, textK, textO, textP;         // text areas for reservation stations
    TextField textM, textN;                       // text fields for ALU
    TextField textX, textY;                       // text fields for commit
    TextArea  rob;                                // text area for reorder buffer
    TextArea  rsHW, rsDS;                         // text areas for reservation stations
    Label psL, partitionL, decodeL, decodeHL;
    Label dispatchL, aluL, commitL, psHL, deliL, deliHL;
    Label decodeHL1, decodeHL2, decodeHL3, cmL, icmL;
    Label aluHL1, commitHL1, commitHL2, commitHL3;

    public Frame frame = null;                    // Get Frame

    public void init() {
                    setSize(600, 600);

                    /***** the applet uses GridBagLayout  *****/
                    /** to handle the buttons and textfileds **/
                    GridBagLayout      gridBag = new GridBagLayout();
                    GridBagConstraints cons    = new GridBagConstraints();
                    setLayout(gridBag);

                    /****** MENU Bar ******/
                    resetB     = new Button("RESET");
                    clockB     = new Button("CLOCK");
                    backB      = new Button("BACK");
                    helpB      = new Button("HELP");
                    textCLOCK  = new TextField("0");

                    resetB.addActionListener(new ResetAction(this));
                    clockB.addActionListener(new ClockAction(this));
                    backB.addActionListener(new BackAction(this));
                    helpB.addActionListener(new HelpAction(this));
                    textCLOCK.addActionListener(new TextAction(this));

                    cons.fill = GridBagConstraints.BOTH;
                    cons.anchor =  GridBagConstraints.CENTER;
                    cons.gridheight = 1;
                    cons.gridwidth = GridBagConstraints.REMAINDER;
                    cons.weightx = 1.0;
                    cons.weighty = 0.0;

                    panel = new Panel(new GridLayout(1, 6));
                    gridBag.setConstraints(panel, cons);

                    panel.setBackground(Color.yellow);
                    panel.setForeground(Color.red);
                    panel.add(resetB);
                    panel.add(clockB);
                    panel.add(textCLOCK);
                    panel.add(backB);
                    panel.add(helpB);
                    add(panel);

                    /****** PS-Fetch ******/
                    textPS = new TextField("TextPS");
                    psL    = new Label("PS-Fetch");
                    psHL   = new Label("");
```

```
cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1; ;
cons.weightx = 1.0;
cons.weighty = 0.0;
panel = new Panel(new BorderLayout());
gridBag.setConstraints(panel, cons);
panel.add(psL, "West");
panel.add(textPS, "Center");
panel.add(psHL, "East");
add(panel);

/******* Add a canvas with lines ******/
canvPS = new PSCanvas();
cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1; ;
cons.weightx = 1.0;
cons.weighty = 0.0;
panel = new Panel();
gridBag.setConstraints(panel, cons);
panel.add(canvPS);
add(panel);

/****** Partition ******/
textPATHAX = new TextField("PATHA-X");
textPATHAY = new TextField("PATHA-Y");
textPATHBX = new TextField("PATHB-X");
textPATHBY = new TextField("PATHB-Y");
partitionL = new Label("Partition");

cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridheight = 1;
cons.gridwidth = GridBagConstraints.REMAINDER;
cons.weightx = 1.0;
cons.weighty = 0.0;
panel = new Panel(new GridLayout(1,5));
gridBag.setConstraints(panel, cons);
panel.add(partitionL);
panel.add(textPATHAX);
panel.add(textPATHAY);
panel.add(textPATHBX);
panel.add(textPATHBY);
add(panel);

/**** Add a canvas with lines ****/
canPART = new PARTCanvas();
cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1; ;
cons.weightx = 1.0;
cons.weighty = 0.0;
panel = new Panel();
gridBag.setConstraints(panel, cons);
panel.add(canPART);
add(panel);

/***** Decode *****/
textDeA  = new TextField("DecodeA");
textDeB  = new TextField("DecodeB");
textDeC  = new TextField("DecodeC");
textDeD  = new TextField("DecodeD");
decodeL  = new Label("Decode");
decodeHL = new Label("");

cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridheight = 1;
cons.gridwidth = GridBagConstraints.REMAINDER;
cons.weightx = 1.0;
cons.weighty = 0.0;
panel = new Panel(new GridLayout(1,6));
gridBag.setConstraints(panel, cons);
panel.add(decodeL);
panel.add(textDeA);
panel.add(textDeB);
panel.add(decodeHL);
panel.add(textDeC);
panel.add(textDeD);
add(panel);

/****** CM/ICM ******/
textCM  = new TextField("TextCM");
textICM = new TextField("TextICM");
cmL     = new Label("CM/ICM");
icmL    = new Label("");

cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1;
cons.weightx = 1.0;
cons.weighty = 1.0;
panel = new Panel(new GridLayout(1,4));
gridBag.setConstraints(panel, cons);
panel.add(cmL);
panel.add(textCM);
panel.add(textICM);
panel.add(icmL);
add(panel);

/****** Delivery ******/
textInstr0  = new TextField("Instruction0");
```

```
textInstr1  = new TextField("Instruction1");
deliL       = new Label("Delivery");
deliHL      = new Label(" ");

cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1;
cons.weightx = 1.0;
cons.weighty = 1.0;
panel = new Panel(new GridLayout(1,4));
gridBag.setConstraints(panel, cons);
panel.add(deliL);
panel.add(textInstr0);
panel.add(textInstr1);
panel.add(deliHL);
add(panel);

/******* Add a canvas with lines ******/
canDis = new DisCanvas();
cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1;
cons.weightx = 1.0;
cons.weighty = 0.0;
panel = new Panel();
gridBag.setConstraints(panel, cons);
panel.add(canDis);
add(panel);

/****** Dispatch/RSV  ******/
textJ     = new TextArea("RSV0",2,6);
textK     = new TextArea("RSV1",2,6);
textO     = new TextArea("RSV2",2,6);
textP     = new TextArea("RSV3",2,6);
rsHW      = new TextArea("RSV HW",2,6);
rsDS      = new TextArea("RSV DS",2,6);
dispatchL = new Label("Dispatch/RSV");

cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1;
cons.weightx = 1.0;
cons.weighty = 1.0;
panel = new Panel(new GridLayout(1, 7));
gridBag.setConstraints(panel, cons);
panel.add(dispatchL);
panel.add(textJ);
panel.add(textK);
panel.add(rsHW);
panel.add(rsDS);
panel.add(textO);
panel.add(textP);
add(panel);

/******* Add a canvas with lines ******/
canv = new LineCanvas();
cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1; ;
cons.weightx = 1.0;
cons.weighty = 0.0;
panel = new Panel();
gridBag.setConstraints(panel, cons);
panel.add(canv);
add(panel);

/******  ALU  ******/
textM  = new TextField("TextM");
textN  = new TextField("TextN");
aluL   = new Label("ALU");
aluHL1 = new Label(" ");

cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1;
cons.weightx = 1.0;
cons.weighty = 1.0;
panel = new Panel(new GridLayout(1,4));
gridBag.setConstraints(panel, cons);
panel.add(aluL);
panel.add(textM);
panel.add(textN);
panel.add(aluHL1);
add(panel);

/******* Add a canvas with lines ******/
canv = new LineCanvas();
cons.fill = GridBagConstraints.BOTH;
cons.anchor =  GridBagConstraints.CENTER;
cons.gridwidth =  GridBagConstraints.REMAINDER;
cons.gridheight = 1; ;
cons.weightx = 1.0;
cons.weighty = 0.0;
panel = new Panel();
gridBag.setConstraints(panel, cons);
panel.add(canv);
add(panel);

/****** ROB ******/
textX     = new TextField("TextX");
textY     = new TextField("TextY");
rob       = new TextArea("ROB",3,50);
```

```
                commitL   = new Label("Reorder Buffer");
                commitHL1 = new Label(" ");
                commitHL2 = new Label(" ");
                commitHL3 = new Label(" ");

                cons.fill = GridBagConstraints.BOTH;
                cons.anchor =  GridBagConstraints.CENTER;
                cons.gridwidth =  GridBagConstraints.REMAINDER;
                cons.gridheight = 1;
                cons.weightx = 1.0;
                cons.weighty = 1.0;
                panel = new Panel(new BorderLayout());
                gridBag.setConstraints(panel, cons);
                panel.add(commitL, "West");
                panel.add(rob,"Center");
                panel.add(commitHL1, "East");
                add(panel);

                /******* Add a canvas with lines ******/
                canv = new LineCanvas();
                cons.fill = GridBagConstraints.BOTH;
                cons.anchor =  GridBagConstraints.CENTER;
                cons.gridwidth =  GridBagConstraints.REMAINDER;
                cons.gridheight = 1; ;
                cons.weightx = 1.0;
                cons.weighty = 0.0;
                panel = new Panel();
                gridBag.setConstraints(panel, cons);
                panel.add(canv);
                add(panel);

                /***** Commit/Write Back *****/
                commitHL2 = new Label("Commit/Write Back");
                cons.fill = GridBagConstraints.BOTH;
                cons.anchor =  GridBagConstraints.CENTER;
                cons.gridwidth =  GridBagConstraints.REMAINDER;
                cons.gridheight = 1;
                cons.weightx = 1.0;
                cons.weighty = 1.0;
                panel = new Panel(new GridLayout(1,4));
                gridBag.setConstraints(panel, cons);
                panel.add(commitHL2);
                panel.add(textX);
                panel.add(textY);
                panel.add(commitHL3);
                add(panel);
    }

    /***** Get Frame *****/
    public Frame getFrame() {
                if (frame == null) {
                    Container containingWin = getParent();
                    while ( containingWin != null) {
                                if (containingWin instanceof Frame) {
                                    frame = (Frame) containingWin;
                                    break;
                                } else
                                    containingWin = containingWin.getParent();
                    }}
                return(frame);
    }
} // End Pipeline

/*
 * class LineCanvas draw
 * lines between steps
 */
class LineCanvas extends Canvas {
    LineCanvas() {
                setSize(600, 20);}
    public void paint(Graphics g) {
                g.drawLine(240, 0, 240, 20);
                g.drawLine(360, 0, 360, 20);
                g.fillArc(230,10,20,20,75,30);
                g.fillArc(350,10,20,20,75,30);}
} // End LineCanvas

/*
 * class ResetAction set all the textfields to
 * default state, i.e. all textfields are empties,
 * clock cycle will be zero
 */
class ResetAction implements ActionListener{
    Pipeline pipeline;
    SetBlank reset;
    public ResetAction (Pipeline pipe){
                pipeline = pipe;}
    public void actionPerformed(ActionEvent e){
                Object source = e.getActionCommand();
                if("RESET".equals(source)) {
                    reset = new SetBlank();
                    reset.setBlank(pipeline);
                    pipeline.textCLOCK.setText("0");}
    }
} // End ResetAction

class ClockAction implements ActionListener{
    Pipeline        pipeline;
    AnalyzeSignals   signals;
    SetBlank        reset;
    int             i;

    public ClockAction (Pipeline pipe) {
                pipeline = pipe;}
    public void actionPerformed(ActionEvent e){
                i = Integer.parseInt(pipeline.textCLOCK.getText());
                Object source = e.getActionCommand();
```

```
                        if("CLOCK".equals(source)) {
                          i++;
                          pipeline.textCLOCK.setText(Integer.toString(i));

                          reset = new SetBlank();
                          reset.setBlank(pipeline);
                          signals = new AnalyzeSignals();
                          signals.analyze(i,pipeline);
                        }
  }
} // End ClcokAction

class BackAction implements ActionListener{
  Pipeline       pipeline;
  AnalyzeSignals  signals;
  SetBlank       reset;
  int i;

  public BackAction (Pipeline pipe) {
                        pipeline = pipe;}
  public void actionPerformed(ActionEvent e){
                        i = Integer.parseInt(pipeline.textCLOCK.getText());
                        Object source = e.getActionCommand();
                        if("BACK".equals(source)) {
                          i--;
                          if (i >= 0){
                                      pipeline.textCLOCK.setText(Integer.toString(i));

                                      reset = new SetBlank();
                                      reset.setBlank(pipeline);
                                      signals = new AnalyzeSignals();
                                      signals.analyze(i,pipeline);
                          }
                          else
                                      System.out.println("No more cyckles");
                        }
  }
} // End BackAction

class SetBlank{
  Pipeline pipeline;

  public void setBlank(Pipeline pipeline){
                        pipeline.textPS.setText(" ");
                        pipeline.textPATHAX.setText(" ");
                        pipeline.textPATHAY.setText(" ");
                        pipeline.textPATHBX.setText(" ");
                        pipeline.textPATHBY.setText(" ");
                        pipeline.textDeA.setText(" ");
                        pipeline.textDeB.setText(" ");
                        pipeline.textDeC.setText(" ");
                        pipeline.textDeD.setText(" ");
                        pipeline.textCM.setText(" ");
                        pipeline.textICM.setText(" ");
                        pipeline.textJ.setText(" ");
                        pipeline.textK.setText(" ");
                        pipeline.textM.setText(" ");
                        pipeline.textN.setText(" ");
                        pipeline.textO.setText(" ");
                        pipeline.textP.setText(" ");
                        pipeline.textInstr0.setText(" ");
                        pipeline.textInstr1.setText(" ");
                        pipeline.rsHW.setText(" ");
                        pipeline.rsDS.setText(" ");
                        pipeline.rob.setText(" ");
                        pipeline.textX.setText(" ");
                        pipeline.textY.setText(" ");
                        System.out.println("ResetNew");
  }
} // End reset

class TextAction implements ActionListener{
  Pipeline       pipeline;
  AnalyzeSignals  signals;
  SetBlank       reset;
  int i;

  public TextAction (Pipeline pipe) {
                        pipeline = pipe;}
  public void actionPerformed(ActionEvent e){
                        i = Integer.parseInt(pipeline.textCLOCK.getText());
                        reset = new SetBlank();
                        reset.setBlank(pipeline);
                        signals = new AnalyzeSignals();
                        signals.analyze(i,pipeline);
  }
} // End TextAction

class HelpAction implements ActionListener{
  Pipeline  pipeline;
  PopHelp   popHelp;

  public HelpAction (Pipeline pipe){
    pipeline = pipe;}

  public void actionPerformed(ActionEvent e){
                        Object source = e.getActionCommand();
                        if("HELP".equals(source)) {
                          /***** Pop up a "Running" window *****/
                          popHelp = new PopHelp(pipeline.getFrame());
                        }
  }
} // End HelpAction

class PopHelp extends Dialog implements ActionListener {
  Button button;
  String string1 = "This Execution Pipeline was made by Jenny Liu fall 1998. ";
```

```
String string2 = "It shows the execution pipelining in six steps. ";
String string3 = "Each steps executes under one cycle.";
String string4 = "There are five buttons and one text fields on the menu bar: ";
String string5 = "1. Button Reset will set all text fields empties and clock to 0.";
String string6 = "2. button Clock read the data and shows them in textfileds. ";
String string7 = "3. Button Back shows the data on cycle-1. ";
String string8 = "4. Button Help shows messages";

public PopHelp(Frame parent) {

                    super(parent, "HELP", true);
                    setSize(600,250);

                    TextArea textArea  = new TextArea("");
  button = new Button("OK");

                    textArea.setText(string1 + "\n" + string2 + "\n" + string3);
                    textArea.append("\n" + string4 + "\n" + string5);
                    textArea.append("\n" + string6 + "\n" + string7);
                    textArea.append("\n" + string8);

                    button.addActionListener(this);
                    setLayout(new BorderLayout());
                    add("North", textArea);
  add("South", button);
                    show();
}

public void actionPerformed(ActionEvent ex) {
  String action = ex.getActionCommand();
  if ("OK".equals(action)) {
                     dispose();}}
} // END PopHelp
```

AnalyzeSignals.java

```
/*
 * AnalyzeSignals read data file
 * and put them to the text fields
 */
import java.applet.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.lang.*;
import java.util.*;
import java.net.*;

class AnalyzeSignals {
   int             clock;
   Pipeline        pipeline;
   String          signals, token, base;
   URL             url;
   BufferedReader   in;
   URLConnection    connection;
   StringTokenizer  st;
   FirstClock       firstClock;
   SecondClock      secondClock;
   ThirdClock       thirdClock;
   FourthClock      fourthClock;
   FifthClock       fifthClock;
   SixthClock       sixthClock;

   public void analyze(int i, Pipeline pipeline){

                    /****** read from file execution_data *****/
                    try {
                       base = "execution_data";
                       url = new URL(pipeline.getDocumentBase(), base);
                       connection = url.openConnection();
                       connection.setDoOutput(true);

                       /***** sends POST query *****/
                       in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
                       for(clock=1; clock<4100; clock=clock+1){
                                    signals = in.readLine();
                                    st = new StringTokenizer(signals);
                                    token = st.nextToken();

                                    if (i==1 && (token.equals(Integer.toString(i)))){
                                       firstClock = new FirstClock();
                                       firstClock.first(in,pipeline);
                                       break;
                                    }
                                    if (i==2 && (token.equals(Integer.toString(i)))){
                                       secondClock = new SecondClock();
                                       secondClock.second(in,pipeline);
                                       break;
                                    }
                                    if (i==3 && (token.equals(Integer.toString(i)))){
                                       thirdClock = new ThirdClock();
                                       thirdClock.third(in,pipeline);
                                       break;
                                    }
                                    if (i==4 && (token.equals(Integer.toString(i)))){
                                       fourthClock = new FourthClock();
                                       fourthClock.fourth(in,pipeline);
                                       break;
                                    }
                                    if (i==5 && (token.equals(Integer.toString(i)))){
                                       fifthClock = new FifthClock();
                                       fifthClock.fifth(in,pipeline);
                                       break;
                                    }
                          if (i>=6 && (token.equals(Integer.toString(i)))){
```

```
                                        sixthClock = new SixthClock();
                                        sixthClock.sixth(in,pipeline);
                                        break;
                                    }
                }
                in.close();
            } catch(MalformedURLException ex){
                System.out.println("URLException:"+ex);
            } catch(IOException ex){
                System.out.println("IOException:"+ex);;
            }
    }
}// End AnalyzeSignals

class FirstClock{
    Pipeline        pipeline;
    String          line;
    String          token1,token2;
    BufferedReader in;
    StringTokenizer  st;

    public void first(BufferedReader in, Pipeline pipeline){
            try {
                /*** read first data PS-Fetch ***/
                line = in.readLine();
                st = new StringTokenizer(line);

                token1 = st.nextToken();    // take away first parameter
                token2 = st.nextToken();    // this is data
                pipeline.textPS.setText(token2);
            } catch(Exception ex){
                ex.printStackTrace();
            }
    }
}// End FirstClock

class SecondClock{
    Pipeline        pipeline;
    String          line;
    String          token1,token2;
    BufferedReader   in;
    StringTokenizer  st;
    FirstClock      firstClock;

    public void second(BufferedReader in, Pipeline pipeline){

            try {
                firstClock = new FirstClock();
                firstClock.first(in,pipeline);

                /*** read data Partitions ***/
                line   = in.readLine();     // PathA-X
                st     = new StringTokenizer(line);
                token1 = st.nextToken();
                token2 = st.nextToken();
                pipeline.textPATHAX.setText(token2);

                line   = in.readLine();     // PathA-Y
                st     = new StringTokenizer(line);
                token1 = st.nextToken();
                token2 = st.nextToken();
                pipeline.textPATHAY.setText(token2);

                line   = in.readLine();     // PathB-X
                st     = new StringTokenizer(line);
                token1 = st.nextToken();
                token2 = st.nextToken();
                pipeline.textPATHBX.setText(token2);

                line   = in.readLine();     // PathB-Y
                st     = new StringTokenizer(line);
                token1 = st.nextToken();
                token2 = st.nextToken();
                pipeline.textPATHBY.setText(token2);
            } catch(Exception ex){
                ex.printStackTrace();
            }
    }
}// End SecondClock

class ThirdClock{
    Pipeline        pipeline;
    String          line;
    String          token1,token2;
    BufferedReader   in;
    StringTokenizer  st;
    SecondClock     secondClock;

    public void third(BufferedReader in, Pipeline pipeline){

            try {
                secondClock = new SecondClock();
                secondClock.second(in,pipeline);

                /*** read data Decode, CM, ICM and Delivery ***/
                line   = in.readLine();       // decode from pathA-X
                st     = new StringTokenizer(line);
                token1 = st.nextToken();
                token2 = st.nextToken();
                pipeline.textDeA.setText(token2);

                line   = in.readLine();       // decode from PathA-Y
                st     = new StringTokenizer(line);
                token1 = st.nextToken();
                token2 = st.nextToken();
                pipeline.textDeB.setText(token2);
```

```
                    line   = in.readLine();        // decode from PathB-X
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textDeC.setText(token2);

                    line   = in.readLine();        // decode from PathB-Y
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textDeD.setText(token2);

                    line   = in.readLine();        // CM
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textCM.setText(token2);

                    line   = in.readLine();        // ICM
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textICM.setText(token2);

                    line   = in.readLine();        // Instruction0
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textInstr0.setText(token2);

                    line   = in.readLine();        // Instruction1
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textInstr1.setText(token2);

                } catch(Exception ex){
                    ex.printStackTrace();
                }
    }
}// End ThirdClock

class FourthClock{
    Pipeline        pipeline;
    String          line;
    String          token1,token2;
    BufferedReader  in;
    StringTokenizer st;
    ThirdClock      thirdClock;

    public void fourth(BufferedReader in, Pipeline pipeline){

                try {
                    thirdClock = new ThirdClock();
                    thirdClock.third(in,pipeline);

                    /*** read data Reservation Stations ***/
                    line   = in.readLine();        // RSV0
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textJ.setText(token2);
                    line   = in.readLine();
                    pipeline.textJ.append("\n" + line);

                    line   = in.readLine();        // RSV1
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textK.setText(token2);
                    line   = in.readLine();
                    pipeline.textK.append("\n" + line);

                    line   = in.readLine();        // RSV2
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textO.setText(token2);
                    line   = in.readLine();
                    pipeline.textO.append("\n" + line);

                    line   = in.readLine();        // RSV3
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.textP.setText(token2);
                    line   = in.readLine();
                    pipeline.textP.append("\n" + line);

                    line   = in.readLine();        // RSV DS
                    pipeline.rsDS.setText(line);
                    line   = in.readLine();
                    pipeline.rsDS.append("\n" + line);
                    line   = in.readLine();
                    pipeline.rsDS.append("\n" + line);
                    line   = in.readLine();
                    pipeline.rsDS.append("\n" + line);
                    line   = in.readLine();
                    pipeline.rsDS.append("\n" + line);
                    line   = in.readLine();
                    pipeline.rsDS.append("\n" + line);
                    line   = in.readLine();
                    pipeline.rsDS.append("\n" + line);
                    line   = in.readLine();
                    pipeline.rsDS.append("\n" + line);

                    line   = in.readLine();        // RSV HW
```

```
                    pipeline.rsHW.setText(line);
                    line   = in.readLine();
                    pipeline.rsHW.append("\n" + line);
                    line   = in.readLine();
                    pipeline.rsHW.append("\n" + line);
                    line   = in.readLine();
                    pipeline.rsHW.append("\n" + line);
                } catch(Exception ex){
                    ex.printStackTrace();
                }
    }
}// End FourthClock

class FifthClock{
    Pipeline        pipeline;
    String          line;
    String          token1,token2;
    BufferedReader  in;
    StringTokenizer st;
    FourthClock     fourthClock;

    public void fifth(BufferedReader in, Pipeline pipeline){

                try {
                    fourthClock = new FourthClock();
                    fourthClock.fourth(in,pipeline);

                    /*** read data ALU ***/
                    line   = in.readLine();           // From ALU A
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    line   = in.readLine();
                    pipeline.textM.setText(token2 + " " + line);

                    line   = in.readLine();           // From ALU B
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    line   = in.readLine();
                    pipeline.textN.setText(token2 + " " + line);
                } catch(Exception ex){
                    ex.printStackTrace();
                }
    }
}// End FifthClock

class SixthClock{
    Pipeline        pipeline;
    String          line;
    String          token1,token2,token3,token4;
    String          token5,token6,token7;
    BufferedReader  in;
    StringTokenizer st;
    FifthClock      fifthClock;

    public void sixth(BufferedReader in, Pipeline pipeline){

                try {
                    fifthClock = new FifthClock();
                    fifthClock.fifth(in,pipeline);

                    /*** read data to ROB ***/
                    line   = in.readLine();
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    pipeline.rob.setText(token2);

                    line   = in.readLine();
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token3 = st.nextToken();
                    pipeline.rob.append("\n" +token3);

                    /*** read data to Commit ***/
                    line   = in.readLine();
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    line   = in.readLine();
                    pipeline.textX.setText(token2 + " "+ line);

                    line   = in.readLine();
                    st     = new StringTokenizer(line);
                    token1 = st.nextToken();
                    token2 = st.nextToken();
                    line   = in.readLine();
                    pipeline.textY.setText(token2 + " " + line);
                } catch(Exception ex){
                    ex.printStackTrace();
                }
    }
}// End SixthClock
```

PSCanvas.java

```
/*
 * PSCanvas.java draws lines between
 * PS-fetch and Partition
 */

import java.awt.event.*;
```

```
import java.awt.*;

class PSCanvas extends Canvas {
   PSCanvas() {
                              setSize(600, 40);
   }
   public void paint(Graphics g) {
                              g.drawString("PATH-A", 190, 15);
                              g.drawString("PATH-B", 490, 15);

                              /***** Path A *****/
                              g.drawLine(250, 0, 250, 20);
                              g.drawLine(250, 20,200, 20);
                              g.drawLine(250, 20,300, 20);
                              g.drawLine(200, 20,200, 40);
                              g.drawLine(300, 20,300, 40);

                              g.fillArc(190,30,20,20,75,30);
                              g.fillArc(290,30,20,20,75,30);

                              /***** Path B *****/
                              g.drawLine(480, 0, 480, 20);
                              g.drawLine(480, 20,430, 20);
                              g.drawLine(480, 20,530, 20);
                              g.drawLine(430, 20,430, 40);
                              g.drawLine(530, 20,530, 40);

                              g.fillArc(420,30,20,20,75,30);
                              g.fillArc(520,30,20,20,75,30);
   }
}
```

PSCanvas.java

```
/*
 * PARTCanvas.java draws lines between
 * Decode and Partition
 */

import java.awt.event.*;
import java.awt.*;

class PARTCanvas extends Canvas {
   PARTCanvas() {
                              setSize(600, 20);
   }
   public void paint(Graphics g) {

                              g.drawLine(170, 0, 170, 20);
                              g.drawLine(270, 0, 270, 20);
                              g.drawLine(440, 0, 440, 20);
                              g.drawLine(530, 0, 530, 20);

                              g.fillArc(160,10,20,20,75,30);
                              g.fillArc(260,10,20,20,75,30);
                              g.fillArc(430,10,20,20,75,30);
                              g.fillArc(520,10,20,20,75,30);
   }
}
```

DisCanvas.java

```
/*
 * DisCanvas.java draws lines between
 * Delivery and Dispatch
 */
import java.awt.event.*;
import java.awt.*;

class DisCanvas extends Canvas {
   DisCanvas() {
                              setSize(600, 20);
   }
   public void paint(Graphics g) {
                              g.drawLine(250, 0, 250, 20);
                              g.drawLine(420, 0, 420, 20);
                              g.fillArc(240,10,20,20,75,30);
                              g.fillArc(410,10,20,20,75,30);
   }
}
```

Appendix C. HTML Files

pipe.html

```
<HTML>
<HEAD>
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
  <META NAME="GENERATOR" CONTENT="Mozilla/4.06 [en] (X11; I; SunOS 5.5.1 sun4m)
[Netscape]">
   <TITLE>Execution pipeline</TITLE>
</HEAD>
<BODY BACKGROUND="bg.gif">

<CENTER>EXECUTION PIPELINE</CENTER>

<P>This is a demo which visualizes the flow of instructions in the
execution pipeline. You can start simulation by clicking here <A
HREF="http://uabx04c214.uab.ericsson.se:8080/tst.html" target="top">Start
Simulation.</A> The results of the simulation shows in the applet below.
<P><APPLET code="Pipeline.class" width="600" height="600"></APPLET>
</BODY>
</HTML>
```

tst.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
  <META NAME="GENERATOR" CONTENT="Mozilla/4.06 [en] (X11; I; SunOS 5.5.1 sun4m)
[Netscape]">
   <TITLE>Simulation</TITLE>
</HEAD>
<BODY BACKGROUND="whtmarb.jpg">

<CENTER>
<H1>
<FONT SIZE=+4>Start Simulation</FONT></H1></CENTER>
Here you can choose a file which you want to simulate. Enter the filename
in the text field below, it will start simulate. It will tell you when
the simulation is finished.
<BR>For example, if you enter file name "IP_LCC.tc", after simulation it will
type "IP_LCC.tc has been simulated" on the web.
<BR><BLINK>Attention! </BLINK>This simulation takes a very long time.
<BR><FORM method=POST action="http://uabx04c214.uab.ericsson.se:8080/cgi-
bin/simu.pl"><FONT COLOR="#000099">Give
file:</FONT><INPUT type=text name="filename" size=40></FORM>
</BODY>
</HTML>
```

Appendix D. simu.pl

```
#!/usr/local/bin/perl5

print "Content-type: text/html","\n\n";

#
# Parse input data
#
&parse_input_data (*input_array);

$file=$input_array{'filename'};
#print $file;

# Simulation SIMU
open (TCFILE,"> /home/uabjenn/IPU_SIM/jennies.tl");
print TCFILE $file;

system "/home/uabjenn/IPU_SIM/simu vcsrun config.tcl";
system "cp /home/uabjenn/IPU_SIM/execution_data /home/uabjenn/weServer/apache_1.3.3/share/htdocs/execution_data";

sub print_error_n_exit
# help sub routine for error handling
{
  local($title,$header,$text, $version) =@_;
  print <<End_of_Error;

        <title>$title</title>
                                <h1>$header</h1>
                                <hr>$text<hr>
                                $version
End_of_Error
  exit(1);
}

sub return_error
# help sub routine for error handling
# see reference in subroutine sub_parse_data.
{
  local($status,$keyword, $message) =@_;
  print "Content-type: text/html","\n";
  print "Status: ", $status, " ", $keyword, "\n\n";
  print <<End_of_Error;

                                <title>CGI Program - Unexpected Error</title>
                                <h1>$keyword</h1>
                                <hr>$message</hr>
                                Please contact your webmaster for more information.

End_of_Error
  exit(1);
}

sub parse_input_data
# help subroutine for parsing input data.
# see S. Gundavaram:    CGI Programming on the World Wide Web
#                       pp. 63-67.
#                       March 1996, O'Reilly & Associates Inc.
#                       ISBN 1-56592-168-2
#
{
  local(*INPUT_DATA) = @_;
  local ( $request_method, $query_string, @key_value_pairs, $key_value,
                                                        $key,$value);
# Find request method
  $request_method = $ENV{'REQUEST_METHOD'};

# If GET get info from env. var. QUERY_STRING
# If not ( request method = POST ) get info from stdin

  if ($request_method eq "GET") {
    $form_info = $ENV{'QUERY_STRING'}
  } elsif ( $request_method eq "POST") {
    $size_of_form_information = $ENV{'CONTENT_LENGTH'};
    read (STDIN, $form_info,$size_of_form_information);
  } else {
    &return_error(500,"Server Error","Unsupported REQUEST_METHOD");
  }
  @key_value_pairs = split (/&/,$form_info);

  foreach $key_value (@key_value_pairs) {
    ($key,$value) = split(/=/,$key_value);
    $value=~ tr/+/ /;
    $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack ("C",hex ($1))/eg;
    if (defined($INPUT_DATA{$key})) {
      $INPUT_DATA{$key} = join("\0", $INPUT_DATA{$key},$value);
    } else {
      $INPUT_DATA{$key} = $value;
    }
  }
}

print $file;
print " has been simulated.";
```

Appendix E. IPU_supervision

```
//*******************************************************
// This is a part of my thesis work
// All changes I do are under "Jenny's changing"
// File:                /home/uabjenn/IPU_SIM/IPU_supervision
// Author:              uabjenn
// Date:                1998/11/4
//*********************************************************

`include                "/home/uablkl/apz_oden/IPU_verification/SIMULATION/std/icm_entry"
// new or old RMU
//`define levelmem `IPC.RMU.RMUCORE1.RM1.RMU_RMBLOCK
//`define levelmem `IPC.RMU.RMUCORE1.RM1.RMU_VER_RMBLOCK.rm.rmcore_0
`define levelmem `IPC.RMU.C3.RM1.RMU_RMBLOCK
reg                     [15:0] instr;
reg                     [15:0] cycler;

integer log_file;

initial
        begin
          log_file=$fopen("execution_data");
        end

/******** Jenny's changing *****************/
integer countClock;
initial
  begin
                        countClock = 1;
  end

always @(posedge `IPC.CLK)
begin
        $fwrite(log_file,"%d \n",countClock);

        //***** ATU *****
        $fwrite(log_file,"PSMBUS:  %x \n",`IPC.PRSU.PRSUCORE.PSMH.PSMBUS);

        $fwrite(log_file,"PATHA-ASAX:  %x \n",`IPC.ATU.ATUCORE.ATPATHA.PATH_A.ASAX);
        $fwrite(log_file,"PATHA-ASAY:  %x \n",`IPC.ATU.ATUCORE.ATPATHA.PATH_A.ASAY);
        $fwrite(log_file,"PATHB-ASAX:  %x \n",`IPC.ATU.ATUCORE.ATPATHB.PATH_B.ASAX);
        $fwrite(log_file,"PATHB-ASAY:  %x \n",`IPC.ATU.ATUCORE.ATPATHB.PATH_B.ASAY);
        $fwrite(log_file,"DECODE-AX:  %x \n",
                'IPC.ATU.ATUCORE.ATPATHA.PATH_A.ATIDEC.ATIDECREGX.ASAD1X);
        $fwrite(log_file,"DECODE-AY:  %x \n",`IPC.ATU.ATUCORE.ATPATHA.PATH_A.ATIDEC.ATIDECREGY.ASAD1Y);
        $fwrite(log_file,"DECODE-BX:  %x \n",
                'IPC.ATU.ATUCORE.ATPATHB.PATH_B.ATIDEC.ATIDECREGX.ASAD1X);
        $fwrite(log_file,"DECODE-BY:  %x \n",`IPC.ATU.ATUCORE.ATPATHB.PATH_B.ATIDEC.ATIDECREGY.ASAD1Y);

        $fwrite(log_file,"CM:  %x \n",`IPC.ATU.ATUCORE.ATMFH.ATCMH2.CMWD);
        $fwrite(log_file,"ICM:  %x \n",`IPC.ATU.ATICMBLOCK.ICMRDAB);

        $fwrite(log_file,"DELIVERY0:  %x \n",`IPC.ATCMAB0);
        $fwrite(log_file,"DELIVERY1:  %x \n",`IPC.ATCMAB1);


        // ***** IQC *****

        $fwrite(log_file,"RSV0:  %x \n",`IPC.IQC.IQRSV.IQRSVRM.IQR0.RSOPA_REG0);
        $fwrite(log_file,"SN0:  %x\n",`IPC.IQC.IQRSV.IQRSVRM.IQR0.RSSN0);
        $fwrite(log_file,"RSV1:  %x \n",`IPC.IQC.IQRSV.IQRSVRM.IQR1.RSOPA_REG1);
        $fwrite(log_file,"SN1:  %x\n",`IPC.IQC.IQRSV.IQRSVRM.IQR1.RSSN1);
        $fwrite(log_file,"RSV2:  %x \n",`IPC.IQC.IQRSV.IQRSVRM.IQR2.RSOPA_REG2);
        $fwrite(log_file,"SN2:  %x\n",`IPC.IQC.IQRSV.IQRSVRM.IQR2.RSSN2);
        $fwrite(log_file,"RSV3:  %x \n",`IPC.IQC.IQRSV.IQRSVRM.IQR3.RSOPA_REG3);
        $fwrite(log_file,"SN3:  %x\n",`IPC.IQC.IQRSV.IQRSVRM.IQR3.RSSN3);

        $fwrite(log_file,"Op0:  %x SN %x \n",`IPC.IQC.IQRSV.IQR5.OPA0,`IPC.IQC.IQRSV.IQR5.SN0);
        $fwrite(log_file,"Op1:  %x SN %x \n",`IPC.IQC.IQRSV.IQR5.OPA1,`IPC.IQC.IQRSV.IQR5.SN1);
        $fwrite(log_file,"Op2:  %x SN %x \n",`IPC.IQC.IQRSV.IQR5.OPA2,`IPC.IQC.IQRSV.IQR5.SN2);
        $fwrite(log_file,"Op3:  %x SN %x \n",`IPC.IQC.IQRSV.IQR5.OPA3,`IPC.IQC.IQRSV.IQR5.SN3);
        $fwrite(log_file,"Op4:  %x SN %x \n",`IPC.IQC.IQRSV.IQR5.OPA4,`IPC.IQC.IQRSV.IQR5.SN4);
        $fwrite(log_file,"Op5:  %x SN %x \n",`IPC.IQC.IQRSV.IQR5.OPA5,`IPC.IQC.IQRSV.IQR5.SN5);
        $fwrite(log_file,"Op6:  %x SN %x \n",`IPC.IQC.IQRSV.IQR5.OPA6,`IPC.IQC.IQRSV.IQR5.SN6);
        $fwrite(log_file,"Op7:  %x SN %x \n",`IPC.IQC.IQRSV.IQR5.OPA7,`IPC.IQC.IQRSV.IQR5.SN7);

        $fwrite(log_file,"HRdata0:  %x SN %x \n",
                        `IPC.IQC.IQRSV.IQR4.OPA_DATA0,`IPC.IQC.IQRSV.IQR4.SN_DATA0);
        $fwrite(log_file,"HRdata1:  %x SN %x \n",
                        `IPC.IQC.IQRSV.IQR4.OPA_DATA1,`IPC.IQC.IQRSV.IQR4.SN_DATA1);
        $fwrite(log_file,"HRdata2:  %x SN %x \n",
                        `IPC.IQC.IQRSV.IQR4.OPA_DATA2,`IPC.IQC.IQRSV.IQR4.SN_DATA2);
        $fwrite(log_file,"HRdata3:  %x SN %x \n",
                        `IPC.IQC.IQRSV.IQR4.OPA_DATA3,`IPC.IQC.IQRSV.IQR4.SN_DATA3);

        $fwrite(log_file,"FromALUA: %x \n",`IPC.IQC.IQAL_ALUS.ALRESA);
        $fwrite(log_file,"SN: %x \n",`IPC.IQC.IQAL_ALUS.ALASN);
        $fwrite(log_file,"FromALUB: %x \n",`IPC.IQC.IQAL_ALUS.ALRESB);
        $fwrite(log_file,"SN: %x \n",`IPC.IQC.IQAL_ALUS.ALBSN);

        $fwrite(log_file,"ROBPOS0:                      %x \n",`IPC.IQC.IQROB.ROBSN0);
        $fwrite(log_file,"ROBPOS1:      %x \n",`IPC.IQC.IQROB.ROBSN1);

        $fwrite(log_file,"CommitA: %x \n",`IPC.IQCMAB0_A);
        $fwrite(log_file,"IQV0=%x\n",`IPC.IQC.IQV0_A);
        $fwrite(log_file,"CommitB: %x \n",`IPC.IQCMAB1_A);
        $fwrite(log_file,"IQV1=%x\n",`IPC.IQC.IQV1_A);
        countClock = countClock + 1;
end

always @(posedge `IPC.CLK)
begin
/**************ATU-IQC interface******************/
```

```
if(`IPC.GRSZ)
   cycler = cycler + 1;
else
   begin
     cycler = 0;
     instr = 0;
   end
end

always @(posedge `IPC.CLK)
begin
/**************ATU-IQC interface******************/
if(`IPC.ATPATH0)
                        $write("ATU-B: ");
else
                        $write("ATU-A: ");

/*** MIP 0 ****/
if(`IPC.IQSTEP[0]) begin
                        $write("%s ", sn2a(`IPC.IQC.DPSN0 ,0));
                        $write("%h ", `IPC.ATIA0);
                        if(`IPC.ATCMAB0[14]) begin
                                                $write("%s ",Inst_name(`IPC.ATCMAB0[8:0]));
                        end
                        else begin
                                                $write("%h   ",`IPC.ATCMAB0[14:0]);
                        end
end
else
                        $write(". .... ...... ");

/*** MIP 1 ****/

if(`IPC.IQSTEP[1]) begin
                        $write("%s ", sn2a(`IPC.IQC.DPSN1 ,0));
                        $write("%h ", `IPC.ATIA1);
                        if(`IPC.ATCMAB1[14]) begin
                                                $write("%s ",Inst_name(`IPC.ATCMAB1[8:0]));
                        end
                        else begin
                                                $write("%h   ",`IPC.ATCMAB1[14:0]);
                        end
end
else
                        $write(". .... ...... ");


/**********JUMP-REPORT*********************/
                        $write("J:");
                        if(`IPC.IQCJE_A) begin
                                                if(`IPC.IQCJT_A)
                                                                        $write("C ");
                                                else

                                                                        $write("c ");
                        end
                        else                    if(`IPC.IQMIPJMP[1]) begin
                                                  if(`IPC.IQMIPJMP[0])
                                                                        $write("M ");
                                                  else

                                                                        $write("m ");
                                                end
                                                else
                                                                        $write(". ");
/**********DS ADDRESS CALCULATION*********************/
                        $write("A:");
                        if(`IPC.IQWRRQ) begin
                                                                        $write("W");
                                                                        $write("%s ", sn2a(`IPC.IQRQTAGB,0));
                                                                        end

                        else
                        if(`IPC.IQRDRQ) begin
                                                                        $write("R");
                                                                        $write("%s ", sn2a(`IPC.IQRQTAGB,0));
                                                                        end

                        else

                                                                        $write(".. ");


/**********DS ORDER AND READ DATA *********************/
                        $write("DS: ");
                        if(`IPC.DSOB==4'h0)                             $write("... ");
                        else if(`IPC.DSOB==4'h1)    $write("REF ");
                        else if(`IPC.DSOB==4'h2)    $write("W32 ");
                        else if(`IPC.DSOB==4'h3)    $write("R32 ");
                        else if(`IPC.DSOB==4'h4)    $write("WLO ");
                        else if(`IPC.DSOB==4'h5)    $write("WHI ");
                        else if(`IPC.DSOB==4'h6)    $write("WRL ");
                        else if(`IPC.DSOB==4'h7)    $write("WRH ");
                        else if(`IPC.DSOB==4'h8)    $write("DWD ");
                        else if(`IPC.DSOB==4'h9)    $write("DWC ");
                        else if(`IPC.DSOB==4'ha)    $write("DRD ");
                        else if(`IPC.DSOB==4'hb)    $write("DRC ");
                        else if(`IPC.DSOB==4'hc)    $write("DW0 ");
                        else if(`IPC.DSOB==4'hd)    $write("DW1 ");
                        else if(`IPC.DSOB==4'he)    $write("DR0 ");
                        else                                                                                            $write("DR1 ");

                        if(`IPC.DHRDV)
                                                $write("%s ", sn2a(`IPC.DHRDTAGB,0));
                        else
                                                $write(". ");
```

```
/**************IQC-COMMITT******************/
                        $write("IQC: ");
                        if(`IPC.IQV0_A) begin
                              if(`IPC.IQCMAB0_A[14] == 1)
                                                      begin
                                                        $write("%s ",Inst_name(`IPC.IQCMAB0_A[8:0]));
                                                        $write("%h ",`IPC.IQIA0_A);
                                 instr = instr +1;
                                                      end
                              else
                                                      begin
                                                        $write("%h   ",`IPC.IQCMAB0_A);
                                                        $write(".... ");
                                                      end
                                                     $write("%h ",`IPC.IQDE0AB_A);
                                                     $write("%h ", `IPC.IQDE0DB_A);
                        end
                        else begin
                                                     $write("...... ");
                                                     $write(".... ");
                                                     $write("... ");
                                                     $write("........ ");
                        end
                        if(`IPC.IQV1_A) begin
                              if(`IPC.IQCMAB1_A[14] == 1)
                                                      begin
                                                        $write("%s ",Inst_name(`IPC.IQCMAB1_A[8:0]));
                                                        $write("%h ",`IPC.IQIA1);
                                 instr = instr +1;
                                                      end
                              else
                                                      begin
                                                        $write("%h   ",`IPC.IQCMAB1_A);
                                                        $write(".... ");
                                                      end
                                                     $write("%h ", `IPC.IQDE1AB_A);
                                                     $write("%h ", `IPC.IQDE1DB_A);
                        end
                        else begin
                                                     $write("...... ");
                                                     $write(".... ");
                                                     $write("... ");
                                                     $write("........ ");
                        end
                        $write("%d %d",cycler,instr);
$display();

end                     // always @(posedge `IPC.CLK)

/********************************************/

function [8:1] sn2a;
  input [4:0] sn;
  input upper_case;
  begin
    if (sn>=5'd10)
      sn2a = sn + (upper_case?"A":"a") - 5'd10;
    else
      sn2a = sn + "0";
  end
endfunction
/********************************************/
function [8:1] h2a;
  input [3:0] hex;
  input upper_case;
  begin
    if (hex >= 4'ha)
      h2a = hex + (upper_case?"A":"a") - 4'ha;
    else
      h2a = hex + "0";
  end
endfunction
/********************************************/
function [6*8:1] Inst_name;
input                   [8:0]                    addr;

begin
   casex (addr)
            `MFR:     Inst_name = "MFR   " ;        `RSA:     Inst_name = "RSA   " ;
            `WSA:     Inst_name = "WSA   " ;        `LCC:     Inst_name = "LCC   " ; //6 bits opcode
            `ACC:     Inst_name = "ACC   " ;        `SHL:     Inst_name = "SHL   " ;
            `SCC:     Inst_name = "SCC   " ;        `SHR:     Inst_name = "SHR   " ;
            `SSPD:    Inst_name = "SSPD  " ;        //8 bits opcode
            `SSPL:    Inst_name = "SSPL  " ;        `SSPB:    Inst_name = "SSPB  " ;
            `SSPBD:   Inst_name = "SSPBD " ;        `XSTQ:    Inst_name = "XSTQ  " ;
            `XSTQD:   Inst_name = "XSTQD " ;        `EP_L:    Inst_name = "EP-L  " ;
            `EP_LT:   Inst_name = "EP-LT " ;        `EP_G:    Inst_name = "EP-G  " ;
            `EP_GT:   Inst_name = "EP-GT " ;        `EP_J:    Inst_name = "EP-J  " ;
            `EP_JT:   Inst_name = "EP-JT " ;        `XCID:    Inst_name = "XCID  " ;
            `XDID:    Inst_name = "XDID  " ;        `XIID:    Inst_name = "XIID  " ;
            `XRBN:    Inst_name = "XRBN  " ;        `XSRP:    Inst_name = "XSRP  " ;
            `XSRPE:   Inst_name = "XSRPE " ;        `XJCA:    Inst_name = "XJCA  " ;
            `XJCAE:   Inst_name = "XJCAE " ;        `XNOP:    Inst_name = "XNOP  " ;
            `XSRPH:   Inst_name = "XSRPH " ;        `XMP:     Inst_name = "XMP   " ;
            `SHL0:    Inst_name = "SHL0  " ;        //SHL with z = 0
            `SHR0:    Inst_name = "SHR0  " ;        //SHR with z = 0
            `WZU:     Inst_name = "WZU   " ;        `WOU:     Inst_name = "WOU   " ;
            `XLPSU:   Inst_name = "XLPSU " ;        `XSPSU:   Inst_name = "XSPSU " ;
            `XLTSU:   Inst_name = "XLTSU " ;        `XSTSU:   Inst_name = "XSTSU " ;
            `OBSOL1:  Inst_name = "OBSOL1" ;        `OBSOL2:  Inst_name = "OBSOL2" ;
            `OBSOL3:  Inst_name = "OBSOL3" ;        `JTR:     Inst_name = "JTR   " ; //6 bits opcode
            `JTS:     Inst_name = "JTS   " ;        `FESRU:   Inst_name = "FESRU " ;
            `FCZSU:   Inst_name = "FCZSU " ;        `BLO:     Inst_name = "BLO   " ; //BLO with c <> 0
            `JEC:     Inst_name = "JEC   " ;
            `JUC:     Inst_name = "JUC   " ;        `LWCD:    Inst_name = "LWCD  " ;//8 bits opcod
            `RSSU:    Inst_name = "RSSU  " ;        `WSSU:    Inst_name = "WSSU  " ;
            `SSRPU:   Inst_name = "SSRPU " ;        `SSRPEU:  Inst_name = "SSRPEU" ;
```

```
        `SSB:       Inst_name = "SSB   " ;       `SSBD:      Inst_name = "SSBD  " ;
        `SSIB:      Inst_name = "SSIB  " ;       `SSIBD:     Inst_name = "SSIBD " ;
        `AWCD:      Inst_name = "AWCD  " ;       //10 bits opcode
        `SWC:       Inst_name = "SWC   " ;       `SWCD:      Inst_name = "SWCD  " ;
        `NWC:       Inst_name = "NWC   " ;       `AWC:       Inst_name = "AWC   " ;
        `XRST:      Inst_name = "XRST  " ;       `RSII:      Inst_name = "RSII  " ;  //12 bits opcode
        `RDB:       Inst_name = "RDB   " ;       `WSII:      Inst_name = "WSII  " ;
        `WDB:       Inst_name = "WDB   " ;       `LHC:       Inst_name = "LHC   " ;
        `LHCE:      Inst_name = "LHCE  " ;       `LAT:       Inst_name = "LAT   " ;
        `MFRE:      Inst_name = "MFRE  " ;       `MTR:       Inst_name = "MTR   " ;
        `TNB:       Inst_name = "TNB   " ;       `CFID:      Inst_name = "CFID  " ;
        `AR:        Inst_name = "AR    " ;       `ARD:       Inst_name = "ARD   " ;
        `SR:        Inst_name = "SR    " ;       `SRD:       Inst_name = "SRD   " ;
        `MR:        Inst_name = "MR    " ;       `SHLD:      Inst_name = "SHLD  " ;//SHLD with x <> 0
        `SHRD:      Inst_name = "SHRD  " ;       //SHRD with x <> 0
        `ROL:       Inst_name = "ROL   " ;       //ROL with z <> 0
        `DR:        Inst_name = "DR    " ;
        `ER:        Inst_name = "ER    " ;       //12 bits opcode
        `ROR:       Inst_name = "ROR   " ;       //ROR with z <> 0
        `RORD:      Inst_name = "RORD  " ;       //RORD with x <> 0
        `OR:        Inst_name = "OR    " ;       `NR:        Inst_name = "NR    " ;
        `ROLD:      Inst_name = "ROLD  " ;       //ROLD with x <> 0
        `SRT0:      Inst_name = "SRT0  " ;       //SRT with w = 0
        `SCBS:      Inst_name = "SCBS  " ;       `RCBS:      Inst_name = "RCBS  " ;
        `SRT:       Inst_name = "SRT   " ;       //SRT with w <> 0
        `XRBA:      Inst_name = "XRBA  " ;       `XRDS:      Inst_name = "XRDS  " ;
        `XRPS:      Inst_name = "XRPS  " ;       `XRRS:      Inst_name = "XRRS  " ;
        `XRS:       Inst_name = "XRS   " ;       `XRSS:      Inst_name = "XRSS  " ;
        `XWBA:      Inst_name = "XWBA  " ;       `XWDS:      Inst_name = "XWDS  " ;
        `XWPS:      Inst_name = "XWPS  " ;       `XWRS:      Inst_name = "XWRS  " ;
        `XWS:       Inst_name = "XWS   " ;       `XWSS:      Inst_name = "XWSS  " ;
        `XFTD:      Inst_name = "XFTD  " ;       `XFTP:      Inst_name = "XFTP  " ;
        `XFTR:      Inst_name = "XFTR  " ;       `XRIM:      Inst_name = "XRIM  " ;
        `XSIM:      Inst_name = "XSIM  " ;       `XRSV:      Inst_name = "XRSV  " ;
        `XWAM:      Inst_name = "XWAM  " ;       //12 bits opcode
        `XSTP:      Inst_name = "XSTP  " ;       `XRAM:      Inst_name = "XRAM  " ;
        `XRRM:      Inst_name = "XRRM  " ;       `XWRM:      Inst_name = "XWRM  " ;
        `XSJTS:     Inst_name = "XSJTS " ;       `XRJBUM:    Inst_name = "XRJBUM" ;
        `XWJBUM:    Inst_name = "XWJBUM" ;       `XRREFM:    Inst_name = "XRREFM" ;
        `XWREFM:    Inst_name = "XWREFM" ;       `XRIS:      Inst_name = "XRIS  " ;
        `XSIS:      Inst_name = "XSIS  " ;       `XMPI:      Inst_name = "XMPI  " ;
        `SHLD0:     Inst_name = "SHLD0 " ;       //SHLD with x = 0
        `SHRD0:     Inst_name = "SHRD0 " ;       //SHRD with x = 0
        `ROL0:      Inst_name = "ROL0  " ;       //ROL with z = 0
        `ROLD0:     Inst_name = "ROLD0 " ;       //ROLD with x = 0
        `ROR0:      Inst_name = "ROR0  " ;       //ROR with z = 0
        `RORD0:     Inst_name = "RORD0 " ;       //RORD with x = 0
        `XMPS:      Inst_name = "XMPS  " ;
        `RSE:       Inst_name = "RSE   " ;       //10 bits opcode
        `RDP:       Inst_name = "RDP   " ;       `RFID:      Inst_name = "RFID  " ;
        `WS:        Inst_name = "WS    " ;       `WSE:       Inst_name = "WSE   " ;
        `WHC:       Inst_name = "WHC   " ;       `WZL:       Inst_name = "WZL   " ;
        `WFID:      Inst_name = "WFID  " ;       //10 bits opcode
        `AHCS:      Inst_name = "AHCS  " ;       `SHCS:      Inst_name = "SHCS  " ;
        `RS:        Inst_name = "RS    " ;       `WOL:       Inst_name = "WOL   " ;
        `TFID:      Inst_name = "TFID  " ;       //12 bits opcode
        `XLPSL:     Inst_name = "XLPSL " ;       `XSPSL:     Inst_name = "XSPSL " ;
        `XLTSL:     Inst_name = "XLTSL " ;       `XSTSL:     Inst_name = "XSTSL " ;
        `SSIL:      Inst_name = "SSIL  " ;       //10 bits opcode
        `SSN:       Inst_name = "SSN   " ;       `SSL:       Inst_name = "SSL   " ;
        `JLL:       Inst_name = "JLL   " ;       `JLN:       Inst_name = "JLN   " ;
        `SSIN:      Inst_name = "SSIN  " ;       `JZR:       Inst_name = "JZR   " ; //10 bits opcode
        `JER:       Inst_name = "JER   " ;       `JUR:       Inst_name = "JUR   " ;
        `JGT:       Inst_name = "JGT   " ;       `JLT:       Inst_name = "JLT   " ;
        `BLO0:      Inst_name = "BLO0  " ;       //BLO with c = 0
        `JOR:       Inst_name = "JOR   " ;       `BLOD:      Inst_name = "BLOD  " ;//BLOD with c <> 0
        `BLOD0:     Inst_name = "BLOD0 " ;       //BLOD with c = 0
        `FESRL:     Inst_name = "FESRL " ;       `FCZSL:     Inst_name = "FCZSL " ;
        `RDBI:      Inst_name = "RDBI  " ;       //8 bits opcode
        `WDBI:      Inst_name = "WDBI  " ;       `XMOV:      Inst_name = "XMOV  " ;
        `XRJT:      Inst_name = "XRJT  " ;       `XWJT:      Inst_name = "XWJT  " ;
        `XRCMI:     Inst_name = "XRCMI " ;       `XRCMS:     Inst_name = "XRCMS " ;
        `RSI:       Inst_name = "RSI   " ;       `RSSL:      Inst_name = "RSSL  " ;
        `WSI:       Inst_name = "WSI   " ;       `WSSL:      Inst_name = "WSSL  " ;
        `SSRPL:     Inst_name = "SSRPL " ;       //9 bits opcode
        `SSRPEL:    Inst_name = "SSRPEL" ;       `XSJAM:     Inst_name = "XSJAM " ;
        `WRCI:      Inst_name = "WRCI  " ;       `RCB:       Inst_name = "RCB   " ;
        `RCBI:      Inst_name = "RCBI  " ;       `WCB:       Inst_name = "WCB   " ;
        `WCBI:      Inst_name = "WCBI  " ;       ‘ CS:       Inst_name = "CS    " ;
        default:    Inst_name = "??????" ;
   endcase
end
endfunction
```