Teaching Computer Architecture/Organisation using simulators

Herbert Grünbacher
Vienna University of Technology
Treitlstrasse 3/182-2, A-1040 Vienna / Austria
E-mail hg@vlsivie.tuwien.ac.at

Abstract

Experience shows, that many students, especially those with little hardware background, encounter difficulties in understanding the consequences and even concepts of conventional instruction pipelining; superscalar instruction processing is even more complicated and harder to understand. It is particularly difficult to statically teach the concept of a pipeline. Therefore we developed software to simulate and dynamically visualize the processing of instructions by pipelined (superscalar) processors. Three simulators have been developed:

- *WinDLX is based on Hennessy/Pattersons DLX architecture and is modeled at the architecture level, therefore very little processor-internal information is given.*
- *MIPSim is based on Patterson/Hennessy's MIPS processor book and is modeled at the computer organization level, functional units like register file, pipeline registers, multiplexers are visible and MIPSim displays content and dynamic behavior of such units.*
- *M10kSim is based on the MIPS R10000 architecture and models the instruction decode and dispatch unit, the branch unit, the instruction queues and the functional units (address calculation, both ALUs, floating-point adder, floating-point multiply/divide/square-root unit). Concepts like register renaming, branch history table, branch resume buffer, out of order execution can be explained easily using the simulator.*

Teaching cache organization is an easier task, nevertheless visualising cache activities helps understanding the dynamics of a cache memory, Xcache is a simulator which displays the interactions between instruction memory and instruction cache, data memory and data cache, respectively.

The simulator are available for free downloading from http://www.vlsivie.tuwien.ac.at/CompArch

Introduction

Teaching the dynamics of pipelines and caches is rather difficult if done on a paper and pencil basis. In our experience students find it difficult to understand the principles and complications of pipelines and to a lesser extend of caches. To support teaching and give students an environment to experiment, we developed several pipeline simulators and a cache simulator.

My experience is that students appreciate using simulators and by using them get easily introduced to the subject. Based on the knowledge gained from using the simulators they are motivated to further study the subject using books.

Almost all of our students have their private PCs and most of them run Windows95/NT. This was the main reason why we develped the simulators to run under MS Windows. It turned out that students particuarly like to work at home and they are usually well prepared to ask questions in class.

WinDLX

WinDLX is a MS-Windows (16 bit) based pipeline simulator for the DLX processor as described in [1]. DLX is modeled at the architecture level, very little about the underlying computer organization is know at that level.

After loading a symbolic DLX assembler code, most of the information relevant to the CPU (pipeline, registers, I/O, memory, …) can be viewed and modified while executing the code step-by-step or continuously. WinDLX offers statistics about pipeline behavior in time.

WinDLX works with several configurations: Structure (number of floating point functional units) and latency of the floating point can be changed. Forwarding can be enabled/disabled and memory size can be modified. There is extensive online help available to explain the simulator and the internals of DLX.

"Register", "Code", "Pipeline", "Clock Cycle Diagram", "Statistics" and "Breakpoints" windows show internals of the pipeline. Further explanation is given below.

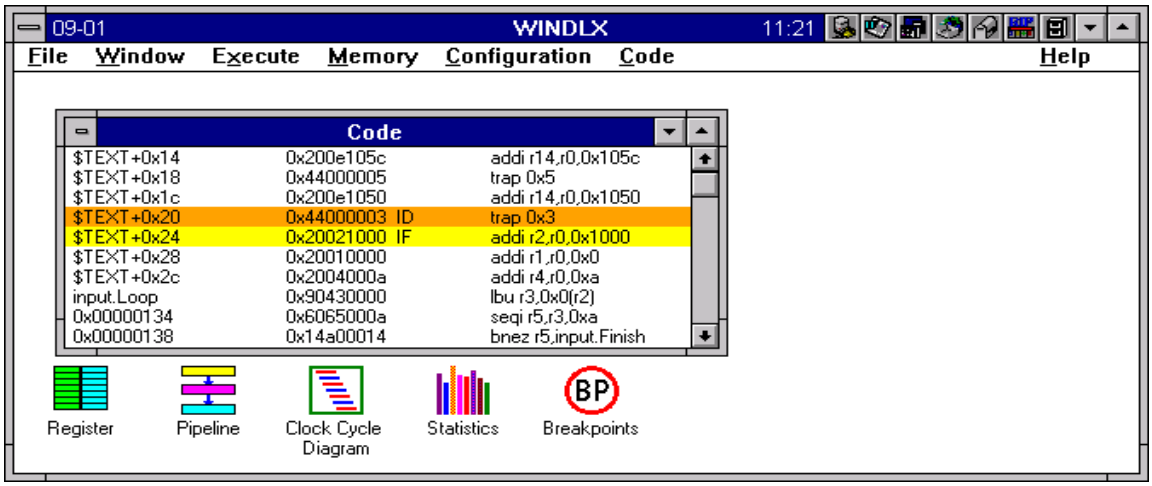

Figure 1 Main Window with open Code Window

Code Window

The code window displays a three column representation of the memory: address (symbolic or in hex), the machine code in hex and the assembler command. Figure 1 shows the main simulation window with a code segment in the open Code Window. Color coding in the different simulation windows is consistent, e.g. WB (Write Back) is colored in blue. Double-clicking on instructions in any of the simulation windows displays pipeline status information in text form giving details about internal registers, operations, stalling and forwarding status.

Pipeline Window

The pipeline window shows the inner structure of the DLX processor - the five pipeline stages of the DLX processor and the floating point units (addition / subtraction, multiplication and division).

Clock Cycle Diagram Window

Figure 2 - the cycle diagram window - shows the timing behavior of the pipeline. The simulation shown is in the 4th cycle, the first command is in the MEM stage, the second in intEX and the fourth in IF. The third command, however, is denoted as "aborted". This is because the second command, jal, is an unconditional branch. This is known after the 3rd cycle, when jal has been decoded. During this cycle the command movi2fp (following after jal) has already been fetched, but the next executed command will be at another address. Therefore the execution of movi2fp must be aborted, leaving a "bubble" in the pipeline.

The branch address of jal is named "InputUnsigned". By clicking Memory/Symbols in the main window, the correspondence between the used symbols and the actual addresses is shown.

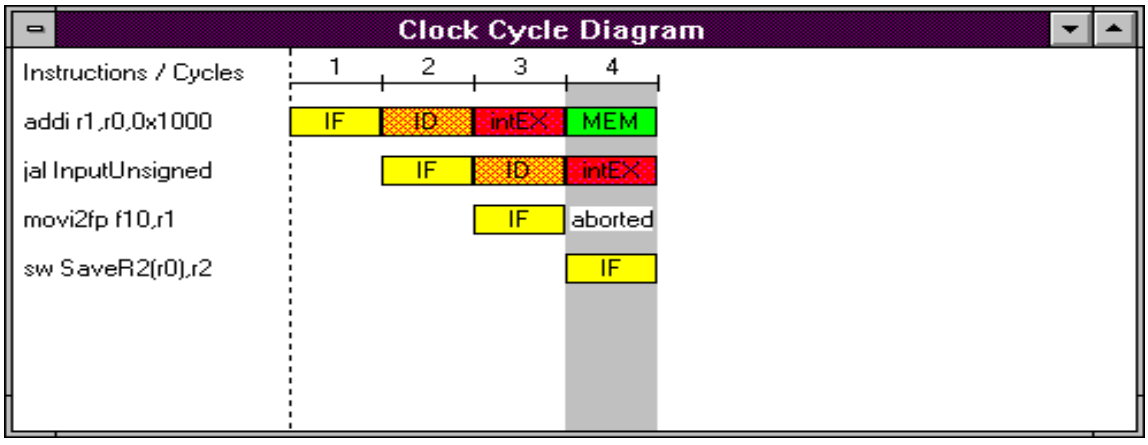

Figure 2. Clock Cycle Diagram

Breakpoint, Register and Statistics Window

Setting breakpoints stops the simulation at user defined points.

The register window shows all registers, not just the register file, and their content in hex.

This statistics window provides information about general aspects (e.g. number of simulation cycles), the hardware configuration used in the simulation, stalls and their causes, conditional branches, load-/store-instructions, floating point stage instructions and traps. Usually, absolute count of events and percentage are given, e.g. "RAW stalls: 17 (7.91 % of all cycles)".

The statistics window is very useful to compare the effects of changes in the pipeline configuration.

MIPSim

MIPSim is a pipeline simulator for the MIPS processor as described in [2]. MIPS is modeled at the computer organization level. Functional units like register files, pipeline registers, ALU, multiplexers, data and control flow are visible.

The user can write small programs (currently there is only a subset of the MIPS instruction set implemented) and watch the pipeline doing its work, modify the program and the content of data memory and register file 'on the fly' and go on simulating to see the effects.

At present MIPSim models a rather simple pipeline without hazard detection and forwarding units.

Assembler Program / Instruction Memory Content

In the very left window in Figure 3 the program code is shown. The program can be executed in single step or running mode. By setting the pointer (in essence the program counter) to a particular address, manual jumps in the program can be accomplished. By double clicking on the Instr. box a window opens in which modifications of the instruction memory content (the program) can be done.

Data Memory Content

By double clicking on the Data box a window opens. Modifications (overwriting) of the data memory content can be done interactively.
Modifying the content of instruction/data memory is very valuable for experimenting with the pipeline, e.g. to show data hazards.

Control / Data Flow Signals

After executing the program code data path and control signal can be displayed by clicking on them.
The instruction content of the different pipeline stages is displayed on top of each stage.
Extensive help as well as a introductory tutorial is available online.

M10KSim

The R10000 is a dynamic superscalar microprocessor which implements the 64-bit Mips Instruction Set Architecture [3], [4]. It fetches and decodes four instructions per cycle and dynamically issues them to five fully-pipelined low-latency execution units. Instructions can be fetched and executed speculatively beyond branches. Instructions graduate in order upon completion. Although execution is aggressively out-of-order, the processor still provides sequential memory consistency and precise exception handling.

Model of the R10000

Our R1000k model concentrates on the most important issues of a superscalar architecture and we wanted to have an easy to learn not to complex user-interface. The following parts of the processor are modelled:

Instruction decode and dispatch unit, responsible for instruction fetching, instruction decoding, register renaming and finally dispatching the instruction to the appropriate queues. The dispatcher works together with the *branch unit* when predicting the outcome of conditional branches. During this process they need to access the *branch history table* and the *branch resume buffer*, which therefore are also simulated. As soon as instructions are being dispatched to the queues they are also given an entry in the *active list*, which also is part of our simulation.
All of the R10000's *instruction queues*, namely an address queue, an integer queue and a floating-point queue are included in the simulation. To be able to determine, which operand results are ready, they access the also simulated *busy table*.

The remaining parts of the simulation are the five *functional execution units*, the address calculation unit, both ALUs, the floating-point adder unit and the floating-point multiply/divide/square-root unit.
Data is read from and written to *memory*, which can be viewed and modified during the simulation.

The memory is simplified and it is assumed to be accessible without any delay. Exception handling is not implemented. The functional units simulate latencies and repeat rate correctly, but the internal pipeline structure is not visible as in MIPSim. Only a reduced set of instructions is implemented.

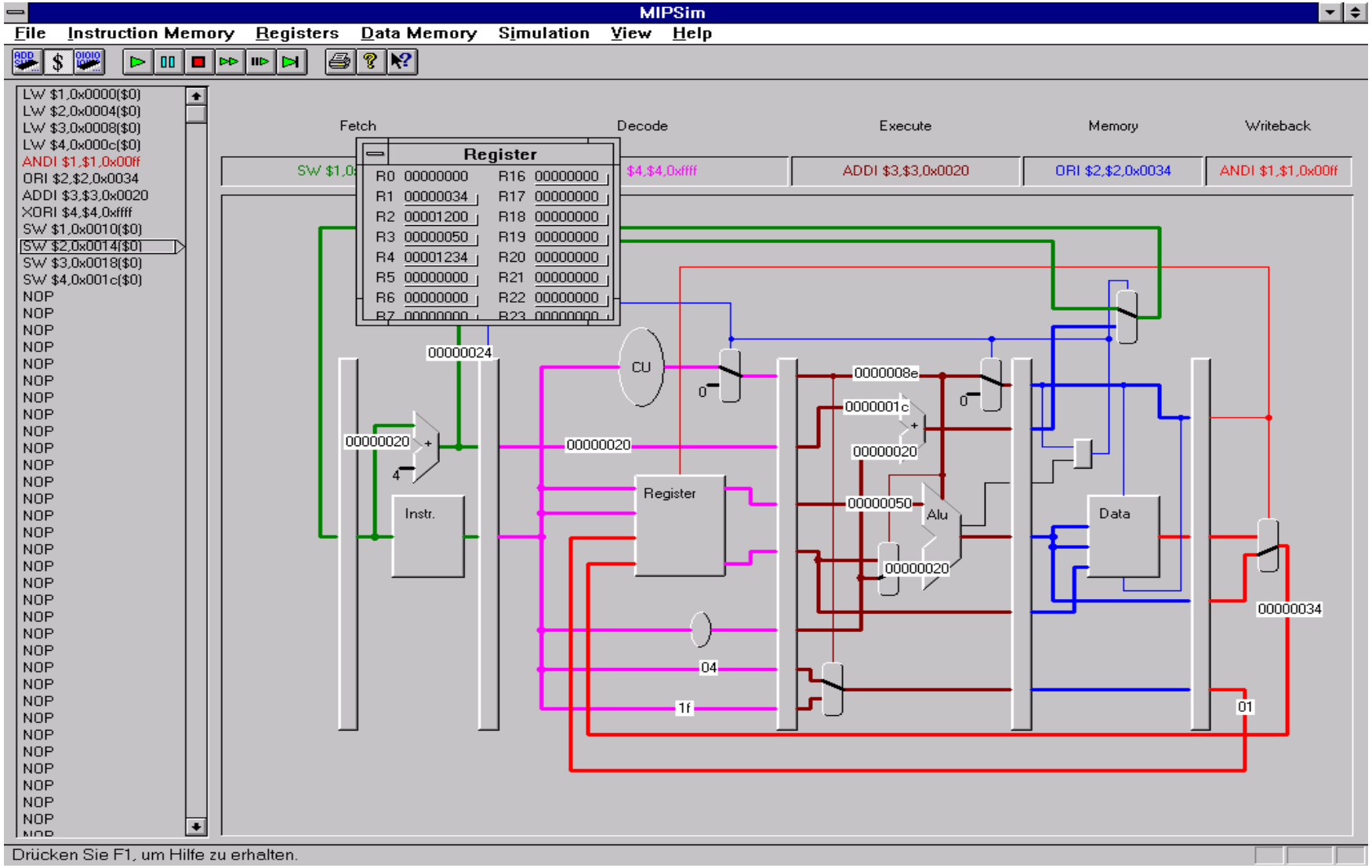

Figure 3. MIPSim Window with Data and Control Signals

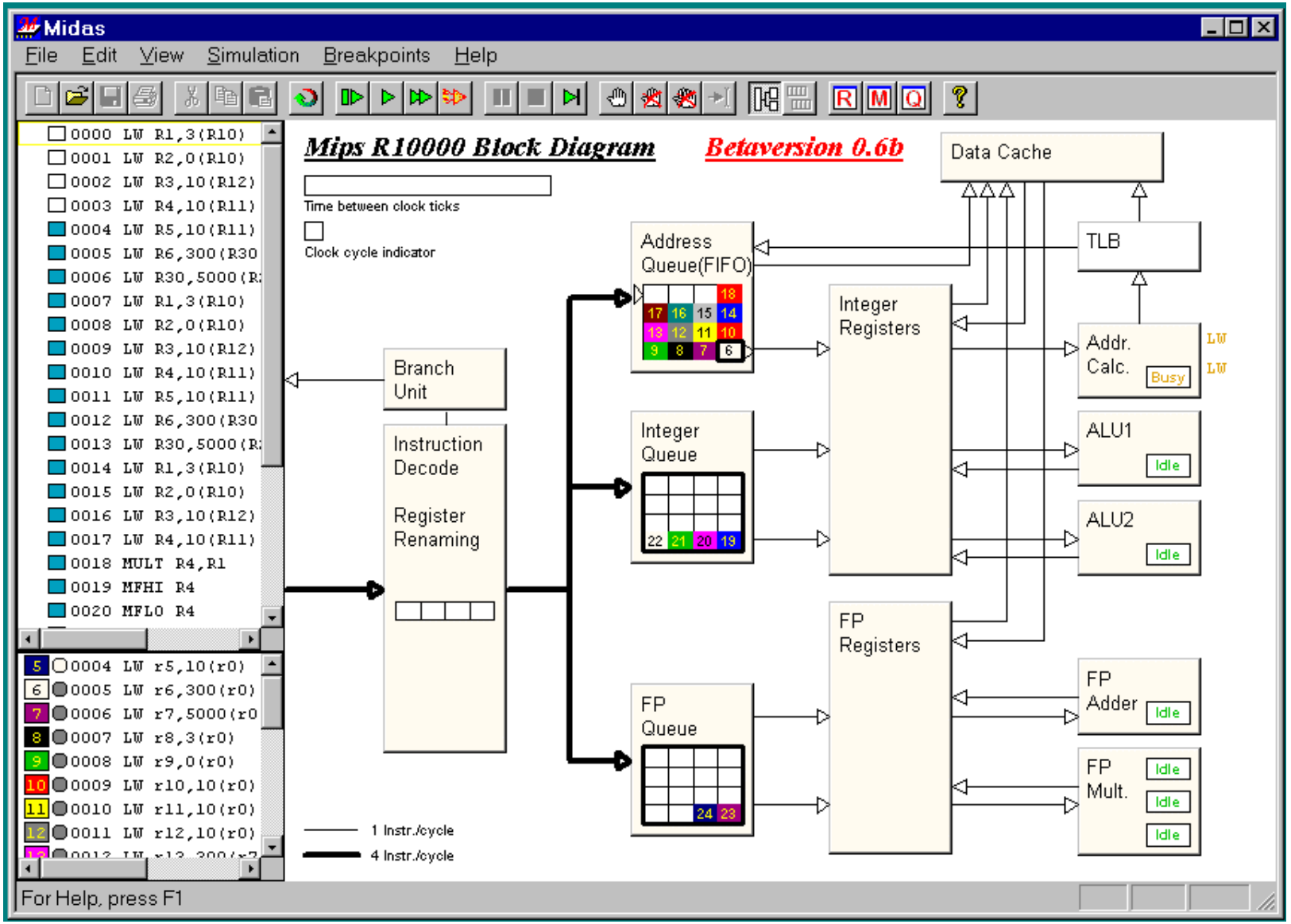

Figure 4. M10kSim Main Windowl

The M10kSimulator

The left windows in Figure 4 shows the assembler window and the active list window. The block diagram on the right side of the windows main screen shows the main components of the simulator.

During a simulation run, the instructions, represented as small balls in different colours, "wander" along the connections between these elements, thus demonstrating their flow through the superscalar instruction pipeline. When an instruction reaches a processor's unit, such as a queue or a functional unit, its "ball" representation disappears and the unit takes over the display of the instruction.

Clicking on the Queue, Register and Data Cache boxes displays the content ot the respective functional units.

Extensive help and an introduction how to use the simulator is available online.

Xcache

XCache simulates and visualizes the behaviour of a cache on a step-by-step basis rather than performing statistical evaluations.

However, to enable advanced cache performance analysis, an interface to Mark D. Hill's cache simulator DINERO was incorporated.

XCache used the same format for the memory reference pattersn DINERO. The user may specify cache parameters like associativity, size, etc., then load an input stream and watch the cache at work. Alternatively, it is possible to define the command-line parameters for DINERO, run a simulation with it and view the results.

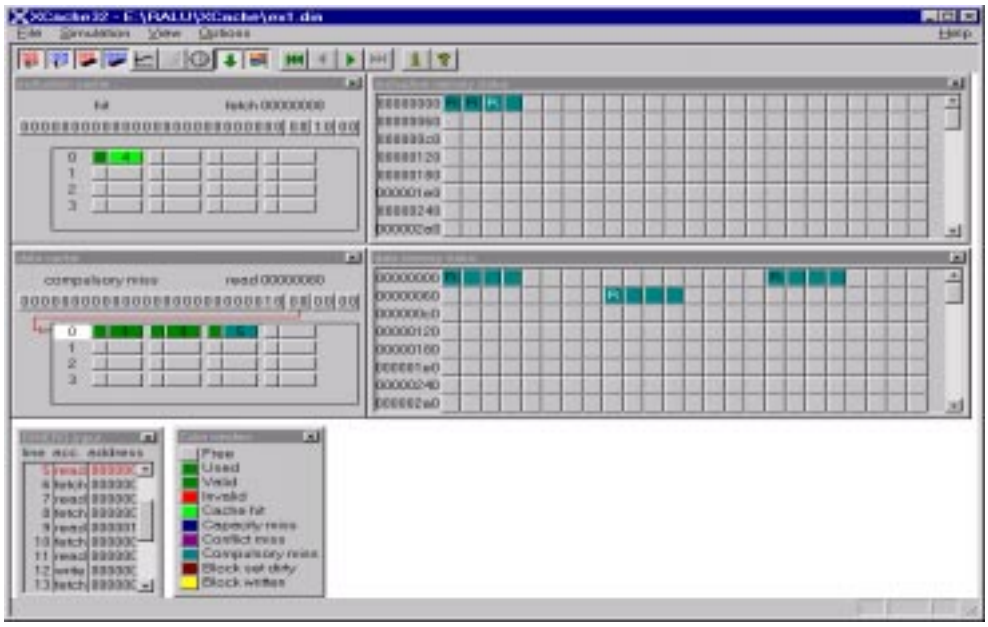

Figure 5 Xcache Main Window

Acknowledment

The simualtors have been developed as part of diploma thesis at our department. Special thanks go to G. Raidl, M. Frigeri, G. Gridling, Ch. Fuss and J. Silhan.

Implementation

The simulators have been written in C++. We make the source code available provided that we get the modified sources in return and that the executables are made available for free to the public domain.

Summary

We have no quantitative measures how much our simulators improved teaching computer architecture. But we do know that students spend more time getting familiar with pipelining than they spent using the paper and pencil approach. We also know that students come well prepared to the lab which accompanies the course and even better come with proposals how to improve the pipeline. Another point which is attractive to students is that they can work at home and this is also helpful to the University as is reduces the load on our computers.

References

[1] D.A. Patterson / J.L. Hennessy: Computer Architecture - A Quantitative Approach, Morgan Kaufmann Publishers, San Mateo, California, 1990

[2] J.L. Hennessy / D.A. Patterson: Computer Organization & Design The Hardware/Software Interface, Morgan Kaufmann Publishers, San Mateo, California, 1994

[3] Presentation of the R10000, Hot Chips, August 17, 1995

[4] J. Heinrich et al.: MIPS R10000 Microprocessor User's Manual, Version 2.0, October 1996, MIPS Technologies, Mountain View, CA

[5] WinDLX, MIPSIM, M10kSim, Xcache at http://www.vlsivie.tuwien.ac.at/CompArch