Visualizing the MMIX Superscalar Pipeline — Not Only for Teaching Purposes

Axel Böttcher

Munich University of Applied Sciences, Department of Computer Science/Mathematics
ab@cs.fhm.edu

Abstract

In this paper, we introduce an environment to visualize the internal activities of superscalar processors. The visualization environment is dedicated to the MMIX-processor. It uses its pipeline simulator but is implemented as a Plug-In to the Eclipse platform.

This environment helps to teach Computer Architecture on a less abstract level. It will be seen that a lot about hardware can be learnt by using a piece of software.

Additionally, this visualization environment would introduce a state of the art IDE and familiarize the students with it. This is a nice and important side effect for general educational purposes.

1 Introduction

Understanding the behavior of modern superscalar processors is becoming more and more difficult as their complexity increases. The ability of processors to simultaneously fetch several instructions and dispatch them to many parallel execution units, as well as elaborated branch prediction schemes and multilevel caches have an almost incomprehensible impact on the execution. Although the execution is nevertheless deterministic. This makes it somewhat difficult to teach these topics without being too abstract.

In this paper we present a visualization environment for the pipeline simulator of Donald Knuth's MMIX-processor. This virtual processor has mainly been designed for educational purposes and will be used as a reference in forthcoming issues of "The Art of Computer Programming". It is a state-of-the-art RISC machine providing many features implemented modern processors. During tha last few years, this processor has proven to be very useful for teaching undergraduate courses on computers and IT-Systems.

Donald Knuth provides a complete environment to write and simulate programs for this machine [1]. The summit is a simulator that simulates completely pipelined versions of the processor, clock cycle by clock cycle. This simulator is highly configurable and thus can be used to simulate realistic models of existing machines. Due to its unlimited configurability it is called Meta-MMIX, short mmmix. The simulator is a very valuable high complex, although well debugged program. Its output, however, is limited to a lot of textual information that can be generated to describe the machine's state during each clock cycle.

We developed a visualization environment to generate graphical representations based that textual information. This tool is used to demonstrate internal behavior of superscalar processors. It is not limited to demonstration purposes but can be used to support experiments and also for other courses than computer architecture. For example it could be used in a course on compilers to study the effects of register coloring or register spilling on program behavior and performance.

We will present this visualization environment and then briefly discuss some case studies of how it can be used and what can be expected to gain from it.

2 The Visualization Environment

During execution of a program, the pipeline simulator can display a large amount of information as text output. Depending on the information requested, this can amount to many kilobytes per clock cycle. So there is a need for a post-processor to get the most use out of that information. We used the open source Eclipse platform [2] to implement a visualization environment. Eclipse is a lean development platform written in Java. It can be extended by own contributions in an extremely flexible manner. A side effect of using this tool is that the students are familiarized with a state-of-the-art IDE.

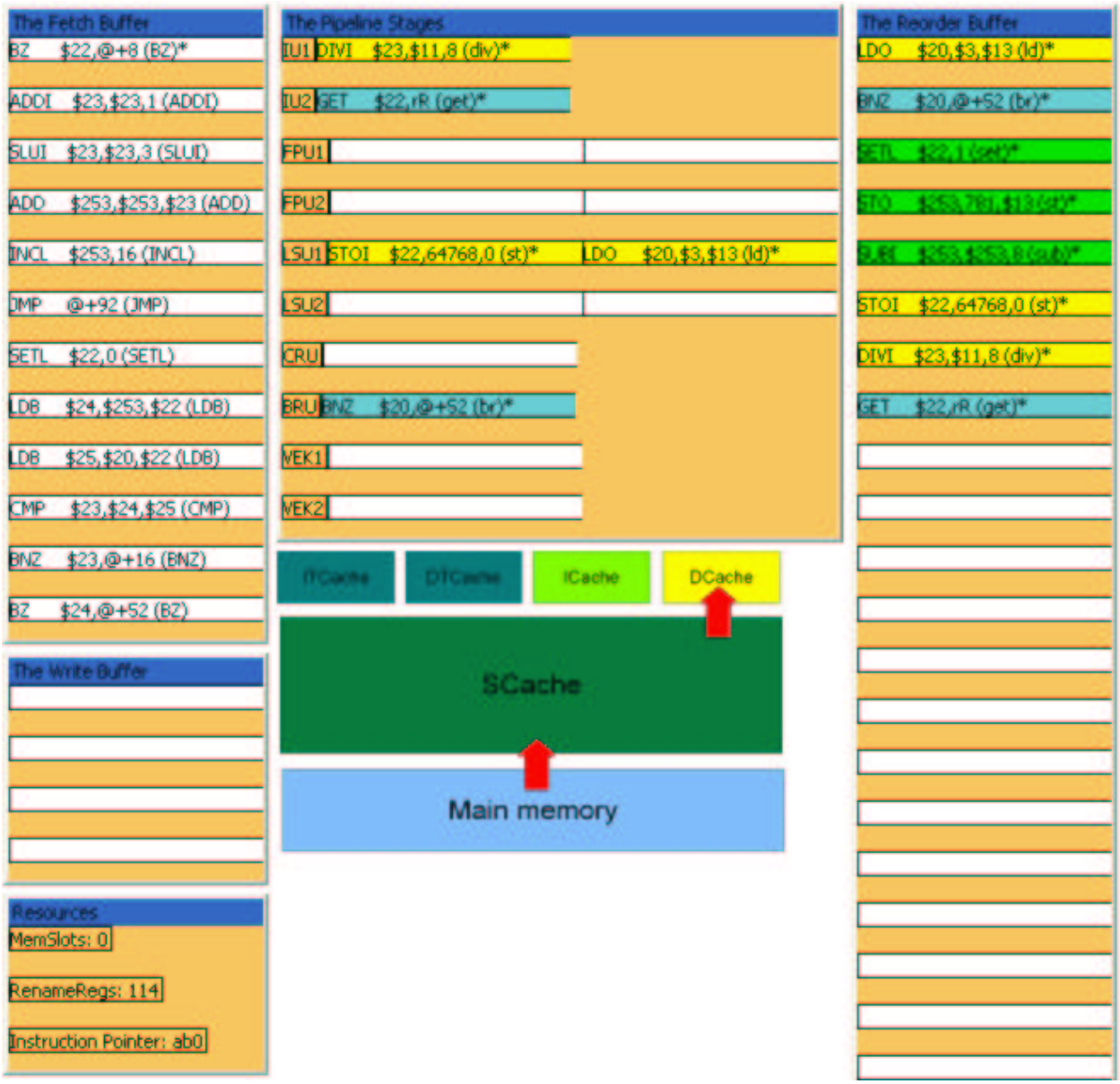


Figure 1: Visualization of the processor configuration during one single clock cycle. Instructions will proceed from left to right.

At the moment the pipeline visualization consists of two main views: First, an *overview* of the configuration – see Figure 1 – showing in detail from left to right:

- The fetch buffer content, i.e. those instructions that have already been fetched (loaded) but are not yet being executed.
- The execution units (titled *The Pipeline Stages* in Figure 1) with all instructions being currently executed. Instructions are taken from the fetch buffer in strict program order and scheduled (dispatched) to the units. However, the next instruction can only be scheduled when there is a unit available that is able to execute this type of instruction. Execution itself will start as soon as all the operands are available.

  Those units for loading/storing (LSU) and for floating point operations (FPU) are themselves pipelined [1]. Thus a new instruction can be scheduled as soon as the previous one has entered the second stage.
- The reorder buffer (ROB) containing all instructions that are waiting for operands, being executed, or having finished execution but have not yet been committed. Execution of instructions can end out of order, because – once on a unit – some may stall due to long execution times, or unavailability of operands e.g. during memory accesses – a so called Read-After-Write hazards.

  In Figure 1 we see three completed instructions in the ROB (number 3 to 5; those with green background), three in execution (the second, sixth and seventh; yellow background), and two instructions that are stalled (second and last; blue background). Instructions will be committed and thus are leaving the ROB in strict program order.
- The state of important resources like number of available rename registers, write buffer entries, or the program counter.
- Overview of memory activities indicated by arrows between the components of memory hierarchy. In the example we see that the D-Cache is filled from main memory through the S-cache.

Furthermore an *activity view* window gives an overview of the reorder buffer's content and memory activity versus time. Each pixel in the diagram of Figure 2 corresponds to one clock cycle. So the figure shows about 650 cycles (corresponding to 650ns when we assume a clock speed of 1GHz!). The bar graph in the upper half just represents the number of instructions in the ROB. Coloring is as above: blue for stalled instructions, green for finished instructions, and yellow for still executing instructions. Sometimes we observe some finished instructions in red color that will be discarded due to mispredicted branches.

The involved execution units can not yet be determined from this view; a double-click on the required cycle shows the details in the other window. The lower part of the activity view shows activity of the memory interface which is the main reason for stalls in the pipeline. We omit the details here concerning by which line can be concluded for what reason the memory interface is busy (e.g. filling of data/code/secondary caches or write back).

In more detail, Figure 2 shows the following main steps:

1. Although the pipeline is stalled due to a load operation, the memory interface is busy loading new instructions. This has just been started before the load was issued.
2. The requested data are supplied from the cache. The instructions just loaded are scheduled, some are committed and again new instructions have to be fetched from main memory.
3. A few new instructions arrive but the fetch of further instructions immediately stalls the pipeline again.
4. Some new instructions have been fetched. But same as above: new instructions are being fetched just before the issue of a load operation.

The visualization environment is implemented as a Plug-In to the Open Source development platform eclipse [2]. The programming language for this exercise is Java. This has several advantages: All facilities Eclipse is offering can be used, e.g. handling of dirty editors, project management, or progress indication.

The `mmmix`-simulator is run in a separate process and just its output is parsed, so there is no need to modify that program. An adapter class wraps the process and thus allows integration into a Model-View-Controller architecture.

3 Usage in Class

The most obvious possibility for usage of the environment are demonstrations done by the teacher. The students may speculate which effects changes

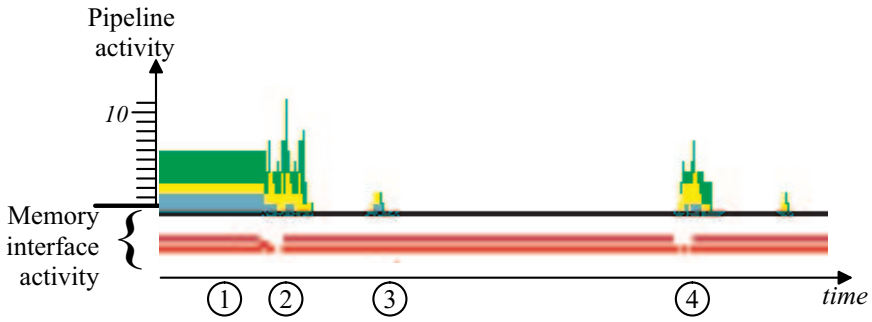


Figure 2: Overview of the activity on the execution units and on the memory interface. The memory interface shows (top to bottom) three types of activity in this example: filling of instruction cache, filling of secondary cache, and filling of data cache.

to the configuration or changes to the program will have during execution (most of the time also educated guesses come out to be wrong).

Secondly there are infinite possibilities for experiments that can be done by the students themselves: Trying to find all parameters of a real existing processor and letting them simulate `MMIX` using this set of parameters.

Furthermore the students can be asked to extend the visualization itself. Eclipse is a highly modular Java based system. We tried to follow the Model-View-Controller paradigm with the implementation of the Plug-In. As a framework we present interfaces where relevant information can be retrieved and thus own views to `MMIX`'s pipeline interna can be programmed by students. So we also have a basis for complex software engineering student projects.

Finally, scientific investigations considering the interworking of architectural parameters and software can be done [3]. The visualisation helps to get a quick overview of the system behavior.

4 Some Case Studies

4.1 Test configuration

We have configured the simulator to behave as far as possible like the PowerPC 970 processor [4]. Since there are no timing characteristics for access to main memory available, we assumed 20 cycles to address memory and to read/write data. Details can be found in table 1; please note that each of the parameters can be changed nearly arbitrarily.

parameter	value
ROB entries	120
max. ops dispatched/cycle	8
max. ops committed/cycle	5
Execution units	9 (see Fig. 1)
data-cache (L1)	64KB
instruction cache (L1)	64 KB
L2-Cache	512 KB
memory address time	20
memory read time	20
memory write time	20

Table 1: Used configuration values to adopt PowerPC970 [4].

4.2 A very simple example

As a very first and quite silly example we will show a drastic effect of a mispredicted branch which, however, we constructed artificially. The following code snippet shows a long running operation `FDIV` (floating point division – takes 40 clock cycles) and a branch depending on its outcome.

```
1          LOC    #100
2 Main     FDIV   $2,$1,$0
3          BNZ    $2,1F
4          ADD    $2,$2,1
5          ADD    $3,$3,1
6          ADD    $4,$3,1
7          ADD    $5,$3,1
8          ADD    $6,$3,1
...        ...    ..
```

The subsequent fast additions will overtake (except that one in line 4) So the instruction window (reorder buffer) gets filled with finished but uncommitted instructions. Finally the branch will execute and turns

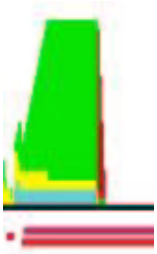


Figure 3: Effect of a mispredicted branch: many finished instructions have to be discarded in this particular case (red bars).

out to have been mispredicted. So all speculatively executed instructions have to be discarded from the reorder buffer. The activity view for this situation can be seen in Figure 3. In practical applications up to now we have not observed this extreme behavior.

4.3 Performance of Quicksort

To give a further rough overview of the possibilities offered, we take a closer look at the execution of a reference implementation of the quicksort algorithm. Figure 4 shows the pipeline activity for 500 cycles, each taken from three different sections. In part a.) the execution during a partionioning step of the array is shown with memory accesses to a cold data cache. Long stalls due to pending load instructions can be seen. Part b.) shows partinioning with memory access to warm cache. Thirdly, part c.) shows execution during the end game with insertion sort and also a warm cache. Each part has its own characteristics. Parts b.) and c.) show a steadily running pipeline, whereas in part a.) we have lots of instructions in the pipeline, most of them stalled. Here we see a starting point for further program optimization.

5 Limitations

Finally we have to comment on the limitations of the pipeline simulator:

- The number of pipeline stages is fixed. For better comparability with existing processors it would sometimes be helpful, to insert some additional stages (e.g. for register renaming).
- The memory/chipset interface is relatively inflexible. The only parameters to configure are a memory address time and read/write times.
- There are no reservation stations. Thus issue (schedule) of instructions to execution units is always in order and stalls as soon as no unit for the next instruction is available.

6 Conclusions and Further Work

Simple examples of `MMIX`-programs for a superscalar configuration have been demonstrated. Understanding the details of the execution on a clock cycle basis is quite a tricky task and all but straightforward. The visualization environment greatly supports the understanding and analysis. Trouble spots are easier to detect giving greater focus to investigations.

The visualization environment shall be extended to cover cache details and to display more information in the activity diagrams (e.g. types of instructions that are currently being executed) as well as the state of more resources.

From `mmix-plugin.sourceforge.net`, the plugin to visualize the `MMIX`-pipeline for the eclipse platform can be downloaded. Eclipse itself is located at `eclipse.org`.

7 Acknowledgements

The author wishes to thank Donald E. Knuth for the nice `MMIX`-processor and for having motivated this work, and Martin Ruckert for many helpful discussions

References

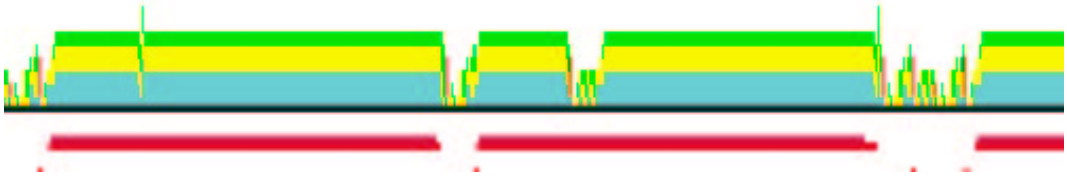
[1] D. E. Knuth, *MMIXware: A RISC Computer for the Third Millennium.* Berlin, Heidelberg: Springer-Verlag, 1 ed., 1999.

[2] E. Gamma and K. Beck, *Contributing to eclipse – Principles, Patterns, and Plug-Ins.* Addison-Wesley, 1 ed., 2004.

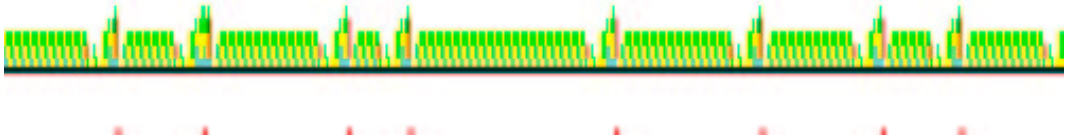
[3] A. Böttcher, "A visualization environment for superscalar machines," *Facta Universitatis (to appear)*, vol. 16, 2004.

[4] P. Sandon, "Powerpc 970: First in a new family of 64-bit high performance powerpc processors," *IBM technical note*, 2002.

a)



b)



c)

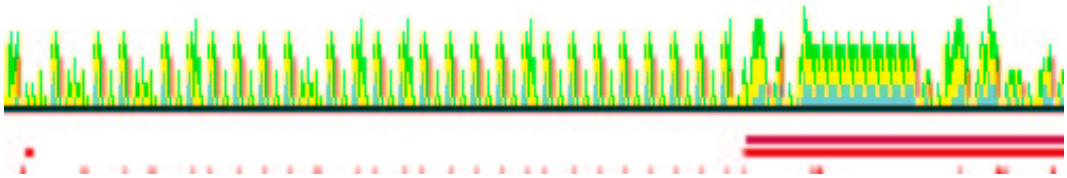


Figure 4: Experiments with quicksort: a) cold cache, b) warm cache c) insertion sort.