ESCAPE: Environment for the Simulation of Computer Architectures for the Purpose of Education

Peter Verplaetse Jan Van Campenhout Henk Neefs
Department of Electronics and Information Systems
University of Ghent, Belgium
{pvrplaet|neefs|jvc}@elis.rug.ac.be

Abstract— **We have developed *ESCAPE*, an easy-to-use, highly interactive portable PC-based simulation environment aimed at the support of computer architecture education. The environment can simulate both a microprogrammed architecture and a pipelined architecture with single pipeline. Both architectures are custom-made, with a certain amount of configurability. Other tools, such as a memory monitor, assembler/disassembler and analysis tools, such as on-the-fly generation of pipeline activity and usage diagrams, are integrated with the environment.**
Based upon our limited experience with the material so far, we can state that the results are excellent. Students invariantly respond very positively, and the evaluations indicate a far deeper understanding than was previously attainable by using only the traditional textbook-and-paper-problems approach.

I. Introduction

THE complexity of computer architectures has increased significantly over the past decades. It is our experience that many students fail to understand even the basic concepts, such as microprogrammed architectures or pipelined execution with simple pipeline, making it impossible to fully understand the operation of a contemporary processor—typically superscalar with out-of-order execution, branch prediction and possible speculative execution.

One way to clarify these simple concepts is by the use of simulation tools. There are many simulators for computer architectures available, but most of them are unsuitable for inexperienced users. Most simulators are designed with accurate modeling as a main feature, as a result these simulators have a complexity similar to today's processors.

This paper describes *ESCAPE*, a computer architecture simulation environment used extensively in an undergraduate level course on computer architecture at the University of Ghent. This environment was created to increase the effectiveness of the course, i.e. to increase the level of insight in and understanding of computer architectures achieved by the students.

The paper is organized as follows. We briefly describe the architectural aspects of both machines. Details of the simulation software and possible uses for the environment are described in the next sections. We then briefly evaluate the preliminary results obtained, and conclude with an outlook on future work.

P. Verplaetse is a Research Assistant of the Fund for Scientific Research – Flanders (Belgium)(F.W.O.)

II. Architectural details

The *ESCAPE* environment consists of two simulators. The instruction set architectures of both machines are essentially identical, even though the microarchitectural aspects are very different (a microprogrammed processor versus a pipelined processor with simple pipeline). The instruction set architecture is inspired by Hennessy and Patterson's DLX [1]. Contrary to the DLX architecture the size of the bitfields in the instruction encoding is not fixed, but depends on the maximum number of instructions and the size of the register file. R-type instructions can have up to 6 formals, which can be useful for implementing more advanced operations in the microprogrammed architecture, a popular homework assignment.

A. Microprogrammed architecture

The architecture consists of a control unit and a datapath. A screenshot of the architecture as it appears in the simulator is shown in figure 1. The datapath consists of a

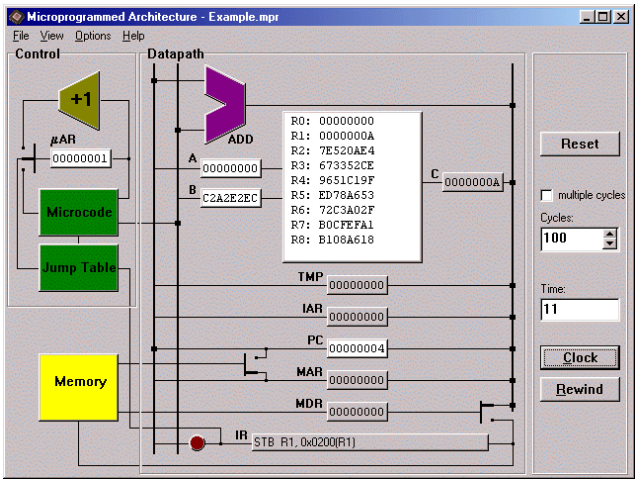

Fig. 1. Screenshot of the microprogrammed architecture.

register file, a set of organizational registers and an ALU which can perform a number of basic operations in a single cycle. A built-in comparator does zero and sign detection on the result.

The control unit is microcoded. Each cycle the microcode address is either incremented or replaced by an absolute value. This jump address resides either in the microcode or is read from a jump table (indexed by the opcode field). The latter is useful for instruction decoding.

The number of jump tables is adjustable from 1 to 4.

The memory interface can load and store bytes, halfwords (16 bit) or words (32 bit), with adjustable access time. Both instructions and data are stored in the same memory (von Neumann architecture).

The microprogrammed architecture (both control unit and datapath) have deliberately been kept simple. There is no microcode pipelining register, it only has basic single-cycled operations, and virtually no microcoding tricks have been used [4]. The datapath is very lean, and could be improved on several counts. This deliberate simplicity leaves ample room for the students to suggest improvements for the architecture.

B. *Pipelined architecture*

A screenshot of the pipelined architecture is shown in figure 2. Both the control unit and the datapath are pipelined

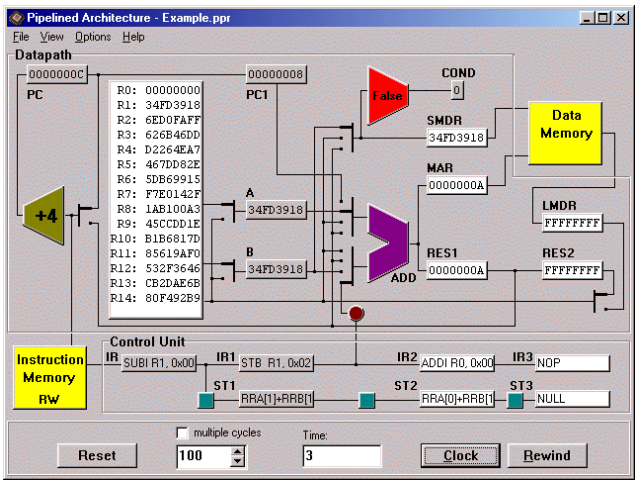

Fig. 2. Screenshot of the pipelined architecture.

into the five traditional stages: IF (instruction fetch), ID (instruction decode), EX (execute and effective address calculation), MEM (memory) and WB (write back). A forwarding mechanism is implemented to prevent the pipe from unnecessary stalling. The register file is read in the ID stage, but written during the WB stage. Write through is explicit by the use of multiplexers.

The EX stage consists of an ALU and a comparator. During the execution of a branch the comparator evaluates the branch condition while the ALU calculates the effective address. Depending on the settings of the simulator the two instructions following the branch can be executed (i.e., a *double* delay slot), nullified (no delay slot), or only the instruction in the IF stage is nullified (single delay slot).

There are two separate memory interfaces: one for instructions and one for data (Harvard architecture). Access to the data memory occurs during the MEM stage. The data memory access time is adjustable.

III. Features of the simulation environment

The *ESCAPE* environment has been implemented in Borland®'s Delphi®. Because the code is compatible with Delphi 1.0, we have both 16- and 32-bit versions, which makes the application run on every Windows®based operating system. A copy of the latest version, further documentation as well as sample exercises can be downloaded from the web:

`http://www.elis.rug.ac.be/~pvrplaet/escape.html`

After starting the simulator an architecture specific form appears. The layout of this form is based on the structural representation of the architecture, as shown in figures 1 and 2. Having all the key elements of the architecture on one single form makes it possible to understand the processor operation without having to swap back and forth between windows. The forms have been designed to be displayable with a 640 × 480 resolution, for classroom use.

A few other forms exist. The memory can be viewed and/or edited in two ways. The data form acts as a memory monitor/editor that allows you to examine or edit the memory content in groups of bytes, halfwords or words, and different number bases (unsigned hexadecimal and unsigned or signed decimal). The code form behaves as an assembler/disassembler that allows easy writing of assembly code.

For the microprogrammed architecture a form similar to the code form exists to edit the microinstructions and jump tables. This is the so-called microcode form. Another important form is the configuration form, that allows one to configure the two architectures.

Key features of the simulation environment are:

- easy-to-use interface;
- partially configurable, easy-to-understand custom-made architectures;
- cycle-per-cycle simulation, or multi-cycle simulation with breakpoints;
- clock rewind, can be disabled to increase simulation speed;
- memory monitor and assembler/disassembler;
- microcode editor;
- on-the-fly trace generation;
- on-the-fly generation of pipeline activity and pipeline usage diagrams;
- all files are in ASCII format, which allows them to be altered with external editors.

IV. Possible uses of *ESCAPE*

Studying the microprogrammed architecture will allow the student to

- become acquainted with the basic synchronous operation at the register transfer level of a datapath and its microprogrammed control;
- learn the basics of microprogramming;
- learn about microprogram optimization techniques, which is quite relevant with respect to contemporary VLIW architectures;
- obtain quantitative data on speed and code size, the influence of memory speed on microcode efficiency, etc.

By using *ESCAPE* for simulating the pipelined architecture, the student can

- become acquainted with pipelined operation, hazards and their solutions;

• experiment with traditional code optimization techniques such as code motion, register renaming, loop unfolding, software pipelining, etc;
• perform small-scale quantitative measurements on benchmark programs, which will result in better insight on the power and main limitations of pipelined execution.

V. Results

A provisional version of the *ESCAPE* environment has been used for two different offerings of the course: once in a post-graduate version, addressed to students that graduated 5–10 years ago (class size: 15), and once in the regular engineering curriculum of the University of Ghent (class size: 120). From the experience gathered from homework assignments, significant improvement has been observed both in the understanding of the architectural issues, as in the ability to effectively deploy such architectures. The comparison is based on the performance on similar assignments during previous years.

VI. Conclusions and future work

In this paper, an interactive graphical simulation environment has been presented, aimed at the support of computer architecture education. The environment allows simulation of simple custom-made microprogrammed or pipelined architectures. First experiments have revealed significant improvements of the teaching effectiveness. Students invariantly respond very positively, and the evaluations indicate a far deeper understanding than was previously attainable by using only the traditional textbook-and-paper-problems approach.

At this point the environment simulates either a microprogrammed or a pipelined machine with limited configurability. Several extensions and additions are being formulated. We plan to extend the simulation model with caches (which will result in variable instruction memory access), out-of-order write back, multiple cycle ALU operations, multiple execution units, and possibly superscalar pipelines with scoreboarding and branch prediction. This will allow the use of a single environment for teaching a wide range, from basic concepts to more advanced topics in contemporary computer architecture.

References

[1] Hennessy, J.L, Patterson, D.A, *"Computer Architecture A Quantitative Approach"*, Morgan Kaufmann Publishers, 1990 & 1996.
[2] Patterson, D.A, Hennesy, J.L, *"Computer Organization & Design. The Hardware/Software Interface"*, Morgan Kaufmann Publishers, 1998.
[3] Flynn, M.J, *"Computer Architecture Pipelined and parallel processor design"*, Jones and Barlett Publishers, 1995.
[4] Rauscher, T.G., Adams, P.M., *"Microprogramming: A Tutorial and Survey of Recent Developments"*, IEEE transactions on Computers, Vol. C-29, No. 1, pp 2–20, 1980.
[5] Sima, D., Fountain, T., and Kacsuk, P, *"Advanced Computer Architectures A Design Space Approach"*, Allison Wesley Longman, 1997.
[6] Heuring, V.P., and Jordan, H.F, *"Computer Systems Design and Architecture"*, Allison Wesley Longman, 1997.