References

[1] C. Moura, "SuperDLX A Generic SuperScalar Simulator," ACAPS Technical Memo 64, School of Computer Science, McGill University, May 1993.

[2] C. Young, N. Gloy, and M. D. Smith, "A Comparative Analysis of Schemes for Correlated Branch Prediction," the 22nd International Symposium on Computer Architecture, June 1995. pp. 276-286.

[3] T. A. Diep, C. Nelson, J. P. Shen, "Performance Evaluation of the PowerPC 620 Microarchitecture," the 22nd International Symposium on Computer Architecture, June 1995. pp. 163-174.

[4] M. Tremblay, D. Greenley, and K. Normoyle, "The Design of the Microarchitecture of UltraSPARC-1," Proceedings of the IEEE, VOL. 83, NO. 12, December 1995, pp.1653-1663.

[5] B. Calder, D. Grunwald, "Next Cache Line and Set Prediction," the 22nd International Symposium on Computer Architecture, June 1995. pp. 287-296.

[6] D. Lee, J.-L. Baer, B. Calder, and D. Grunwald, "Instruction Cache Fetch Policies for Speculative Execution," the 22nd International Symposium on Computer Architecture, June 1995. pp. 357-367.

[7] T. M. Conte, K. N. Menezes, P. M. Mills, and B. A. Patel, "Optimization of Instruction Fetch Mechanisms for High Issue Rates," the 22nd International Symposium on Computer Architecture, June 1995. pp. 333-344.

[8] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," IEEE Micro, April 1996, pp.28-40.

[9] J. H. Edmondson, P. Rubinfeld, R. Preston, and V. Rajagopalan, "Superscalar Instruction Execution in the 21164 Alpha Microprocessor," IEEE Micro, April 1995, pp.33-43.

Table 1: Baseline Simulator Features.

Fetch Rate	User given.
I-Cache	KNL model.
Branch Prediction	User-specified direct-mapped BHT with 2-bit damping counter.
In-Order Issue	Out-of-order execution model.
Out-of-Order Issue	Out-of-order execution model .
Functional Units	User given.
Functional Units EXE latency	User given.
D-Cache	KNL model, write-back or write-through, write-allocate.

- Configurable functional units.
- A fully configurable KNL non-blocking cache structure incorporated in the simulator. (K: the number of ways. N: the number of cache lines in a way. L: line size). Split versus unified option and $c_m + \beta_m \frac{L}{D}$ memory latency model.
- More debugging functions that allow the interruption of the simulation, reconfiguration, restart in a cycle-by-cycle fashion, or run to the end.
- NT/Win95 platform ready.

Table 1 details the baseline features. The simulator supports both in-order and out-of-order issue models. Data are queued in the store buffer before writing into the memory and may be forwarded to a latter load if the address is known.

3 Summary

We have developed a DLX-based superscalar simulator that runs more like a real processor. First, it is instruction-driven and uses a realistic memory interface model. It models the external memory design including bus width, memory access time, and per-bus transfer time. The added on-chip cache structure is fully user-programmable. Modern out-of-order execution model is provided as the baseline simulator structure. A user has a very high degree of freedom in determining how the simulator is configured. The simulator provides richer debugging functions and allows the interruption of the simulation, reconfiguration, restart in a cycle-by-cycle fashion, or run to the end.

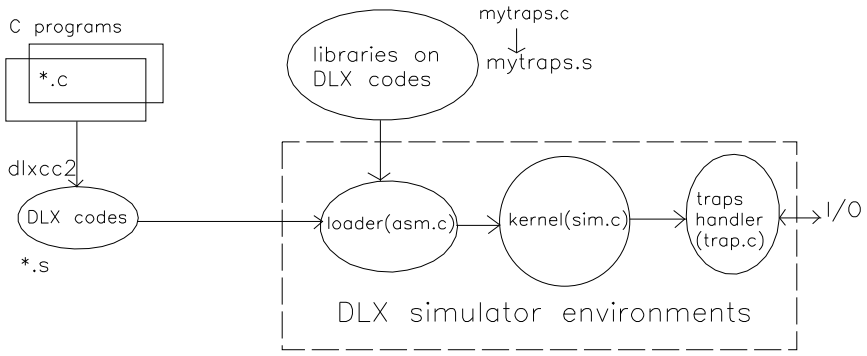

Figure 2: The Simulation Environment.

2.4 System Modification

In order for the simulator to run significant programs like those in SPEC92, we build the required run-time libraries and add trap handlers into the simulator. Figure 2 illustrates the simulation environment. For system-dependent or operating system dependent functions, we add them via traps. We add the functions needed to run the SPEC programs, such as mktemp(), getpid(), getenv(), .. etc. All these traps can be found in traps.h/trap.c in our package. Also some routines such as fread(), fwrite(), realloc() are provided in mytraps.c. After adding sufficient C-functions into the system, the original SPEC92 programs need some small modifications. This includes the following major issues: 1) Make sure that the main() does not use 'argv' and 'argc' to access command line parameters and pass them via static string copy instead. 2) Open a stream file 'inf' for 'stdin' and rename all 'stdin' in programs into 'inf'. 3) Open a stream file 'outf' for 'stdout' and 'stderr' and rename them in programs. 4) Rename all macro-defined functions in the original program to what we define. 5) Make sure that there are not any strongly system dependent functions like fork(), signal(), kill(), etc. If found, alter the program flow to avoid using them.

So far, this instruction driven simulator is able to run non-toy programs to the end. This includes 1) FP benchmark: Alvinn, Ear, Clinpack, Whetstone, FFT, Flops, and 2) Integer benchmark: Compress, Decompress, Eqntott, Espresso, Xlisp and etc.

2.5 Other Features

In addition to the built-in Tomasulo-based mechanism in the simulator, the following options and functions are provided by the simulator.

- Central window versus distributed reservation stations.

time c_m, per-bus transfer time β_m, and bus width D. A small c_m and β_m correspond to the use of a high-speed L2 cache with the processor.

2.2 Branch Prediction

For instruction accesses, two issues must be addressed: one is the prediction of the direction of conditional branches, and the second is the instruction fetching strategy. To improve instruction fetching bandwidth, branch target buffers or next cache line prediction have been used as the mechanism for branch prediction so that the fetch unit can fetch instructions speculatively [2]. The prediction mechanisms can be built with a branch target cache or a branch history table (BHT) such as those used in PowerPC 620 [3]. Alternatively, the prediction mechanisms are incorporated within the instruction caches [4, 5].

In addition to branch prediction, mechanisms for instruction extraction from the predicted paths and fetching policies for instruction cache misses also constrain the performance of instruction accesses [6, 7]. For instance, the fetch unit can always fetch from the speculative path or it may wait until the branches are resolved. Many of the current processors have been able to speculate more than one branch [3, 4, 8, 9]. As an example, the PowerPC 620 is able to fetch instructions speculatively up to 4 branches [3] while the UltraSPARC-1 can speculate up to five branches [4]. The simulator implements the BHT and the branch target buffer approach, and allows the users to specify the number of BHT entries and the reservation station entries of the branch unit.

2.3 Non-Blocking Design

The use of out-of-order execution model complicates the design tradeoffs on the memory systems because portion of the memory latency is hidden from the CPU execution time. For instance, a non-blocking cache allows multiple caches misses and hit accesses while a miss is on progress. With the use of reservation stations and load/store buffers, non-blocking memory accesses naturally become part of the processor execution model. Many of the current processors allow a very high degree of multiple L1 cache misses to tolerate memory latency. Load buffers are used to support these non-blocking misses. Essentially, the maximum number of cache misses that can be posted equals to the number of the load buffers associated with the load/store unit. On the other hand, the data cache ports are usually limited to the number of 1 or 2. This limitation is mitigated by the use of load/store buffers or store buffers associated with the load/store unit. The writes that are queued in the store buffer are not committed to the data cache until the store instructions can be retired from the reorder buffer. The simulator allows the users to specify the degree of non-blocking and the number of store buffers used.

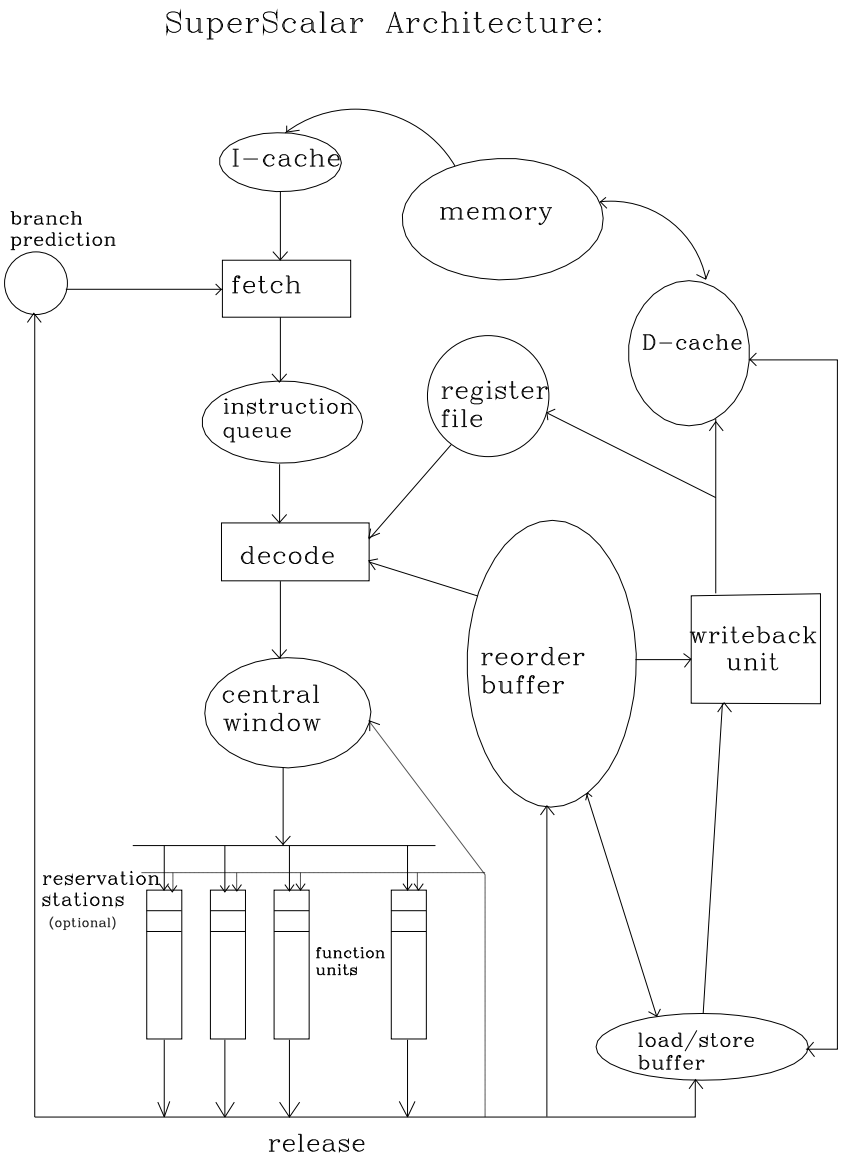

Figure 1: Block Diagram of a DLX-Based Superscalar Simulator.

1 Introduction

Modern microprocessors employ out-of-order execution model, aggressive branch prediction, distributed reservation stations, dynamic renaming for register files, and support in-order completion and precise interrupts. The processor prepares the instructions by loading them into the internal queue usually associated with the fetch unit, thus allowing multiple instructions to be decoded. Instruction fetching must be able to look ahead of the control instructions which alter the execution sequence. Branch prediction is the mechanism used to guide the fetch unit for fetching instructions across a basic block. Multiple instructions may be decoded and dispatched to the reservation stations associated with a functional unit for execution. Instructions that have obtained their source operands can be issued for execution. A Tomasulo-based algorithm is used to support out-of-order execution of the instructions. To support precise interrupt, results of the functional units are written into a reorder buffer and later committed to the processor register file in program order.

We have developed a superscalar simulator that supports the functions and features of modern microprocessors mentioned above. This simulator is based on the DLX scalar processor. We add many functions into the simulation system so that this instruction-driven simulator is able to run many of the SPEC92 programs. The information about this simulator can be found at http://com.el.yuntech.edu.tw. In the following, we address the added features of this simulation system.

2 A DLX-Based Simulator

This instruction-driven simulator provides many more functions than its predecessors developed elsewhere [1]. One of the most important enhancement is that it can run large programs such as the SPEC92 benchmarks. We have added the trap functions and required C-libraries in the system. Figure 1 shows a block diagram of this DLX superscalar simulator. We describe the simulation system starting from its memory system interface.

2.1 Memory Interface Model

The external memory design determines how soon a cache line may be moved into the cache from the outside memory system. This part of memory design relates to the processor interface control. A general form of memory cycles reading a line of size L bytes in a burst transfer memory of bus width D bytes can be represented by $c_m + \beta_m(\frac{L}{D})$ where c_m is a constant time and β_m is the number of clocks of per-bus transfer. Constant time c_m is the clock cycles for the presentation of address to the memory system and memory access latency. A processor may request a word on the line which is being filled during the $c_m + \beta_m(\frac{L}{D})$ cycles. The simulator allows the users to specify the constant

An Enhanced DLX-Based Superscalar System Simulator

Chung-Ho Chen and Akida Wu

Institute of Information and Electronic Engineering
and Department of Electronic Engineering
National Yunlin Institute of Technology
Touliu, R.O.C. on Taiwan
{chen,akida}@el.yuntech.edu.tw

Abstract

We have designed a DLX-based superscalar processor simulator. This simulator provides many more functions than its predecessors developed elsewhere. We have added trap handlers and required C functions in the system so that most of the SPEC92 programs now run on the simulator. In addition, this simulator is fully configurable and re-configurable. Specifically, the following options and functions are provided by the simulator.

- Central window versus distributed reservation stations.
- Branch prediction mechanisms using static or dynamic schemes. The later provides branch target buffer and branch history table.
- Configurable functional units.
- A fully configurable KNL non-blocking cache structure incorporated in the simulator. (K: the number of ways. N: the number of cache lines in a way. L: line size). Split versus unified option and $c_m + \beta_m \frac{L}{D}$ memory latency model.
- Better debugging functions allowing the interruption of the simulation, reconfiguration, restart in a cycle-by-cycle fashion, or run to the end.
- NT/Win95 platform ready.

This DLX-based superscalar simulator is instruction-driven, which offers richer educational features than most of the trace-driven simulators. For information about this simulator, please refer to http://com.el.yuntech.edu.tw.