An Investigation of the Performance of Various Instruction-Issue Buffer Topologies

Stéphan Jourdan, Pascal Sainrat, and Daniel Litaize

Institut de Recherche en Informatique de Toulouse
Université Toulouse III, 118 route de Narbonne, 31062 Toulouse, FRANCE
{jourdan, sainrat, litaize}@irit.fr

Abstract

In out-of-order issue superscalar microprocessors, instructions must be buffered before they are issued. This buffer can be either unified (one buffer linked to all functional units) as in the P6, distributed among the units as in the PowerPC 620, or semi-unified (a few buffers each shared by several units) as in the MIPS R10000. Of course, the size of this buffer also plays a leading role in the performance of the processor. Intensive trace-driven simulations on the SPEC92 suite have been made in order to determine the best designs and relevant choices are pointed out according to the dispatch width of the processor.

1 Introduction

There are many trends and ways of implementing out-of-order superscalar architectures and except for Mike Johnson's book [8] and [2], few results have been published on the most cost-effective execution configurations, especially on out-of-order issue superscalar microprocessors. In such processors, a buffer is required to hold instructions once they are dispatched and not yet issued. We call it the *instruction-issue buffer* throughout the paper. Different designs of this buffer have been implemented in five of the most recently manufactured microprocessors, the P6 [3], PA-8000 [4], R10000 [10], Sparc64 [11], and PowerPC 620 [7]. Intel implemented in its P6 a unified buffer. Hewlett-Packard in its Pa-8000, split it into two parts, one being used only for loads and stores. MIPS, in its R10000, divided it into three parts, one for integer, one for floating-point and the last for load/store instructions. The HaL Sparc64 has a similar split of the buffer except for two additional parts devoted to branches and address calculations. Finally, IBM and Motorola distributed the buffer among each of the functional units of the PowerPC 620.

In this paper, we have examined or quantified the widely held beliefs which are:

- distributing the buffer tends to ease the complexity of the arbitration among issuable instructions,
- unifying the buffer results in less buffer space required for the same performance, assuming constant cycle time.

We based these experiments on the results reported in [9] where we evaluated a large set of configurations of functional units according to the dispatch width of the processor (4, 6, and 8). That previous study was performed assuming an ideal instruction-fetch mechanism, no cache miss, but a unified instruction-issue buffer. These assumptions were made in order to define ideal configurations while considering only data dependencies. We therefore ended up with configurations which might be reduced if other restrictions in the processor were taken into account, but should not restrict results of further studies. We proceed with this methodology in this paper but vary the instruction-issue buffer.

The outline of this paper is as follows. Section 2 details the execution model. Section 3 introduces the simulation process. The benchmarks used are also briefly described in this section. Section 4 summarizes results previously reported in [9] which deal with the configurations of functional units and the impact of the size of the lookahead window on performance

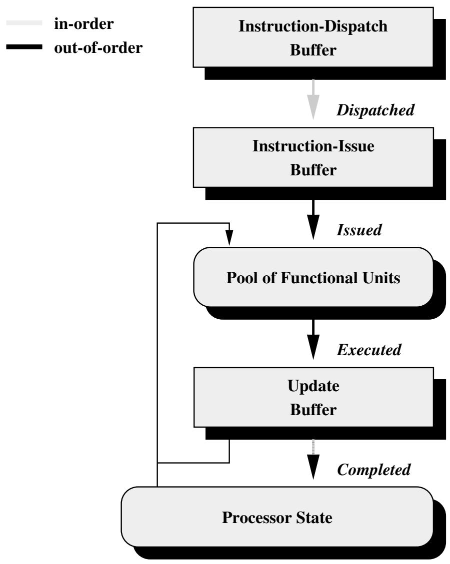

Figure 1: Model of Execution

with regard to the dispatch width. Section 5 presents various topologies of the instruction-issue buffer and reports on results of our simulations. Finally, section 6 compares the results and discusses complementary issues such as arbitration processes. We conclude in section 7.

2 Model of Execution

The modeled architecture implements an out-of-order execution-policy and speculative execution in order to best exploit *instruction-level parallelism* (ILP).

In such an architecture, after their *fetching* and their *decoding*, instructions are *dispatched* in order from the instruction-dispatch buffer to the instruction-issue buffer. The upper bound of the number of instructions dispatched each cycle defines the dispatch width. Register are usually renamed during the dispatch process. Instructions in the issue buffer may be issued out-of-order when all their operands are available, provided no functional-unit conflict occurs.

Issuing policies are the algorithms which arbitrate issuable entries of the instruction-issue buffer. This buffer can be unified (all entries are linked to all functional units), split (all entries are linked to only one unit), or semi-unified. A similar mechanism is called the *node table* by Butler and Patt in [1]. The most natural issuing policy is *oldest first* where the entry which holds the oldest dispatched instruction has priority over the others. Nonetheless, [8] showed that using this priority is unnecessary: cheaper policies such as a fixed priority give roughly the same performance. But more recently, Butler and Patt in [1] showed that these results do not apply to floating-point programs such as *tomcatv* since the performance loss is 20 %. Thus, we avoid most of the possible degradation of the issuing performance by choosing the *oldest first* priority since we are not investigating issuing policies.

When an instruction finishes execution, its result is recorded in the update buffer. Examples of the update buffer are the reorder buffer [12] and the backup files of the checkpoint repair mechanism [5]. To enforce precise interrupt management, an instruction is *completed* only when all prior instructions cannot produce an interrupt anymore. Instructions capable of generating interrupts are conditional branches, divides, and memory accesses. In the latter instruction class, subsequent instructions may update the state as soon as the address is processed. Each cycle, multiple out-of-order completions can be made.

To sum up, as shown in figure 1, the different states followed by an instruction are *fetched*, *decoded*, *dispatched*, *issued*, *executed*, and *completed*.

Concerning data-cache accesses, such an instruction can bypass a store if, and only if, both addresses are known (the value to be stored is forwarded to the load when adresses match). Furthermore, the memory update of stores starts only when all previously dispatched instructions cannot produce interrupts anymore, in order to preserve memory coherency.

Finally, the lookahead window refers to the upper bound of the number of instructions being processed. An instruction enters the window when it is dispatched, and exits when it is completed.

Since we want to point out in this study relevant topologies of the instruction-issue buffer to exploit instruction-level parallelism disregarding any other parameters which can lead to performance degradation, the model assumes:

- an ideal instruction-fetch mechanism: no instruction-cache miss, no branch misprediction and a dispatch process only limited by the fullness of the lookahead window or the instruction-issue buffer, and the dispatch width,
- no data-cache miss.

3 Simulation Process

As previously outlined, the simulator models an ideal instruction-fetch mechanism. Thus, it only takes into account the dispatch and completion widths of the processor, the size of the lookahead window, the memory and register data dependencies, the configuration of functional units, and the topology and the size of the instruction-issue buffer we want to evaluate. The latencies are those of the PowerPC 604 [6], listed in

instructions	latency	instructions	latency
integer	1	load/store	2/3
shift	1	fp add/conv	3
int. multiply	3	fp multiply	3
int. divide	20	fp divide	18s/31d
s stands for single precision and d for double precision			

Table 1: Latencies

table 1. All functional units but divide units, are fully pipelined and mutually independent.

We used the PIXIE profiler [13] in order to produce instruction traces from a real processing of the SPEC benchmarks. From all the data reported by this software, we picked out only the opcode, the operands, and the dynamic memory address (in the case of a memory access) for each instruction. These traces are read by our simulator which performs a *cycle-by-cycle* simulation and gathers the mean number of *instructions completed per cycle* (IPC).

The results presented in this paper are for programs from the SPEC92 suite [14]. Two subsets are defined: CINT92 (*integer*) and CFP92 (*floating-point*). In order to make our evaluations, we use all programs from both sets. They were compiled on a R4600-based SGI workstation using the standard makefiles provided with the suite (with all optimizations turned on). Except for *backprop*, *dnasa7*, *wave5*, and *spice2g6*, all the programs have been run to completion: the smallest input files or slightly modified versions have been used. In all, more than 600 million instructions have been captured (all *NOPs* being removed from the traces).

4 Previous Results

We have defined in an extended version of [9] cost-effective configurations of functional units according to the dispatch width, assuming a unified instruction-issue buffer the size of which was that of the lookahead window. Hence, the features of the instruction-issue buffer should not impact any results.

The type of functional units we have defined are:

- S: integers and Shifts.
- I: Integers.
- M: integers, Multiplies and divides.
- B: Branches.
- A: Address calculations.
- D: Data cache accesses.

W	L	S	I	M	B	A	D	F1	F2	F	Ci	Cf
4	32	2	0	1	2	2	2	1	1	0	3.5	2.1
6	64	2	1	1	2	3	3	0	0	2	5.1	2.8
8	96	2	2	1	3	4	4	0	0	2	6.3	3.3

Table 2: Configurations of Functional Units

- F1: Floating-point adds, converts, and divides.
- F2: Floating-point multiplies and divides.
- F: all Floating-Point instructions.

Table 2 summarizes the cost-effective configurations of functional units we ended up with according to the dispatch width (W). It also reports the size of the lookahead window (L). Simulations have shown that the completion width must be equal to the dispatch width. Ci and Cf are the geometric means of the IPCs obtained with this model for respectively CINT and CFP programs. These results are our references for all *performance ratios* reported in this paper.

5 Topologies of the Instruction-Issue Buffer

In this section, different architectures of the instruction-issue buffer and their performances are reported. As previously said, we distinguish three basic topologies of the instruction-issue buffer: unified, semi-unified and split. In the semi-unified topology, different cases are discussed.

5.1 Unified Instruction-Issue Buffer

In the unified scheme, instructions are dispatched from the instruction-dispatch buffer into a central buffer that holds all instructions waiting for their issue, regardless of their classes or of the functional units they must be issued to. Such a centralized scheme was pioneered in the CDC 6600 supercomputer [15] and is currently implemented in the Intel P6. Figure 2 plots the IPCs gathered while varying the number of entries of a unified instruction-issue buffer on CINT92 and CFP92 programs. Whatever the degree is, one can notice in this figure that the behaviour of the curves is homogeneous.

We have defined relevant sizes which are listed in table 3 according to the dispatch width (W). The size of the lookahead window (L) is also recalled.

One should note the difference between the sizes of the instruction-issue buffer and the lookahead window. As a matter of fact, the lookahead window represents all the instructions in the issue buffer as well as

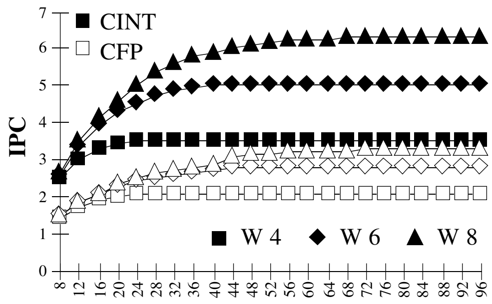

Figure 2: Impact of the Number of Entries in a Unified Topology

W	L	IIB	Ratio (Cint)	Ratio (Cfp)
4	32	28	100 %	99.9 %
6	64	48	99.9 %	99.9 %
8	96	72	99.8 %	99.8 %

Table 3: Relevant Sizes of the Unified Issue Buffer

those issued but not yet completed: executed instructions cannot complete if they were dispatched after an instruction that can still generate an interrupt. As the dispatch is wider, the lookahead window features more entries and thus the execution is more speculative: the number of instructions stalling the completion is higher. Hence, the difference between the sizes of the instruction-issue buffer and the lookahead window widens with the dispatch width.

Moreover, whatever the topology is, the values reported for the size of the instruction-issue buffer are minima for an equivalent level of performance. As a matter of fact, a unified issue buffer resolves the short-term demands on any functional units.

5.2 Semi-Unified Instruction-Issue Buffer

The instruction-issue buffer may be divided into several parts, each part being devoted to a set of functional units. We distinguish three ways of splitting this buffer corresponding to three recent microprocessors. Firstly, as in the PA-8000, the buffer features two parts. Integer, branch, and floating-point instructions are dispatched to the same part (IBFP) while memory accesses are dispatched to the other one (LSP) since their issue procedure is quite different: they must be issued two times, first for the address calculation, and second for the data-cache access. Secondly, integer, branch, and floating-point instructions can be decoupled using one part for integer and branch (IBP) and one for floating-point instructions (FP). This alternative matches the features of the R10000. Finally, branches can be separated from integer instructions and an additional part can be implemented for the process of the adresses while another one is devoted to data-cache accesses, thus symplifying the issue procedure. Consequently, the issue buffer is divided into 5 parts as in the Sparc64.

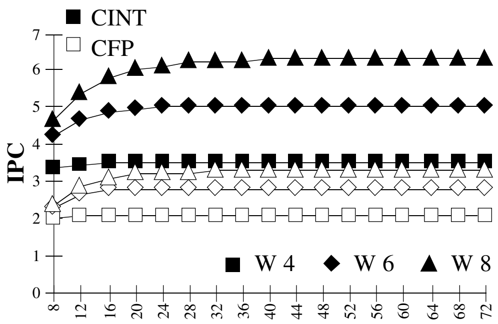

Figure 3: Impact of the Number of Entries of the LSP

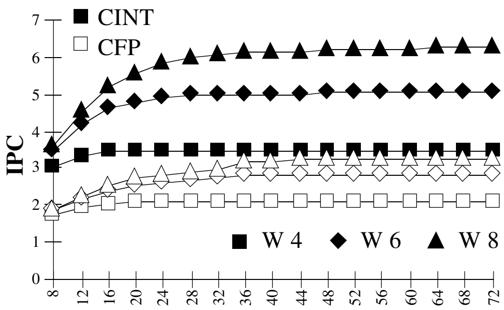

Figure 4: Impact of the Number of Entries of the IBFP

Firstly, as load/stores are always processed separately, we first evaluate the number of entries of the LSP while the size of the IBFP is large enough to avoid influencing any results. Figure 3 brings to light the relevant sizes of the LSP according to the dispatch width. These sizes are 12, 24, and 36 when dispatch widths are 4, 6, and 8 respectively.

Once this number fixed, we evaluate the number of entries of the IBFP. Figure 4 relates the performance when varying these numbers. Cost-effective configurations are summarized in table 4 with their performance ratios. One should know that such LSP sizes are required to obtain a performance ratio close to 100 % on CINT92 programs.

In the second alternative where integer and floating-point instructions have their own part (IBP and FP), the IBP is the same size as the IBFP derived from CINT92 since no floating-point instructions occur in these programs. One should note that the sizes

W	IBFP	LSP	Ratio (Cint)	Ratio (Cfp)
4	24	12	99.5 %	99.3 %
6	36	24	99.3 %	99.4 %
8	52	36	99.1 %	99.9 %

Table 4: Relevant Sizes of the Semi-Unified Issue Buffer Split up into one IBFP and one LSP

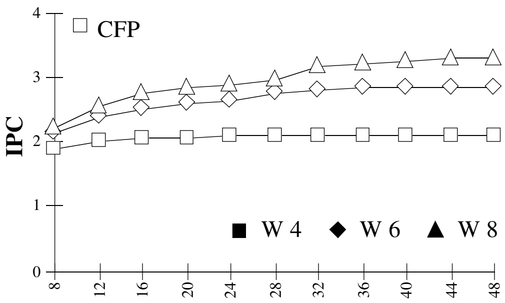

Figure 5: Impact of the Number of Entries of the FP

of the IBP and IBFP are equal except for a dispatch width of 4 where 20 entries in the IBFP are enough to obtain a ratio close to 100 % on CINT92. Figure 5 plots the performance obtained when varying only the size of the FP. Table 5 lists the best configurations for this second alternative.

Finally, branches may be dispatched to an additional part (BP). The integer part (IP) then features less entries than the IBP as pointed out by the choices reported in table 6. Moreover, the load/store part may be split into two parts, one devoted to address calculations (AP) and the other one to data-cache accesses (LSP). Thus, instructions are dispatched simultaneously to the two buffers: the number of entries of the LSP is still the same. However, the number of entries of the address buffer is lower since an entry is freed as soon as the address is processed. Table 6 reports on the number of entries of the integer buffer, the branch buffer, and the address buffer of such a topology.

W	IBP	FP	LSP	Ratio (Cint)	Ratio (Cfp)
4	20	24	12	99.2 %	99.3 %
6	36	32	24	99.3 %	99.3 %
8	52	44	36	99.1 %	99.6 %

Table 5: Relevant Sizes of the Semi-Unified Issue Buffer Split up into one IBP, one FP, and one LSP

W	IP	BP	FP	AP	LSP	Ratio (Cint)	Ratio (Cfp)
4	16	8	24	8	12	98.8 %	99.3 %
6	32	16	32	16	24	98.9 %	99.6 %
8	48	24	44	24	36	98.9 %	99.6 %

Table 6: Relevant Sizes of the Semi-Unified Buffer Featuring one IP, one BP, one FP, one AP, and one LSP

Note that a semi-unified topology can be used to easily implement an out-of-order issue policy. The issue buffer is then divided according only to the type of data processed, that is integer and floating-point. The issuing in each part is in-order and the registers are read only at issue time. This scheme overcomes the need for global operand forwarding. Providing there is enough buffering in each part, there are opportunities for out-of-order execution. Such a scheme is implemented in the POWER1 and POWER2 [16].

5.3 Split Instruction-Issue Buffer

An alternative design is where the instruction-issue buffer is split up into the well-known *reservation stations*, each of them being devoted to one functional unit. Hence, instructions may be dynamically dispatched to different parts of the issue buffer. This induces a performance loss since a functional unit may be waiting for issuable instruction while another of the same type may be requested by several issuable instructions. This loss compared to a non-split scheme is reported in table 7 where all parts are assumed to be infinite, and when using a round-robin dispatch policy over units of the same type. One should note that this loss might be to some extent due to the long time spent by stores in the issue buffer. This might induce an irregular fullness of the load/store parts, which results in a lower issue rate since subsequent instructions are dispatched to the emptiest parts. These performance losses may render this topology unattractive, even more when restricting the sizes of the buffers. Table 8 reports on relevant sizes of the parts of the issue buffer. As regards the total number of entries, the split buffer matches the Sparc64-like buffer except for the load/store parts (LSP). This must come from the reason stated a few lines above.

6 Complementary Issues

In this section, we discuss the dispatch and the issue procedures which are closely related to the topology of the instruction-issue buffer.

Whatever the topology is, each dispatch bus must be linked to all entries of the issue buffer. As shown in

W	Ratio (Cint)	Ratio (Cfp)
4	97.9 %	96.4 %
6	95.4 %	95.2 %
8	93.6 %	95.2 %

Table 7: Fundamental Losses of the Split Buffer

W	S	I & M	B	A	D	F	Ratio (Cint)	Ratio (Cfp)
4	6	4	4	4	10	12	97.2 %	94 %
6	8	8	8	6	12	16	94.9 %	92.8 %
8	10	10	8	6	14	22	93.1 %	92.9 %

Table 8: Relevant Sizes of the Split Issue Buffer

the previous section, the number of entries decreases as the topology becomes more unified. Hence, the dispatch procedure is simpler and faster in a unified topology. However, in order to reduce the complexity and the delay of the dispatch procedure in a non-unified topology, a limit can be applied to the number of instructions which may be dispatched to each part of the issue buffer. Table 9 reports on performance losses where the limit of each part is fixed to its number of issues (always equal to one in a split buffer). These results are reported for a split issue buffer and a semi-unified Sparc64-like topology. This alternative does not seem to be attractive for a split issue-buffer since the reported losses are high. On the contrary, in a Sparc64-like topology, the performance losses may seem to afford some interest with regard to the savings achieved.

Moreover, in a split topology, a policy (round-robin in our case) must be chosen in order to determine the part to which an instruction is dispatched. The main complexity does not rely on the round-robin algorithm but on the fact that the arbiters take into account the fullness of the parts of the issue buffer. However, the arbitration time should be partly overlapped with the

Topology	Programs	W=4	W=6	W=8
Split	Cint92	-6.6 %	-5.6 %	-3.5 %
	Cfp92	-7.1 %	-8.1 %	-9 %
Sparc64-like	Cint92	-2.2 %	-2.8 %	-1.7 %
	Cfp92	-2.6 %	-3.4 %	-3.9 %

Table 9: Losses due to the Dispatch Alternative

register renaming process.

Concerning the issue procedure, in a split buffer, the arbitrations for functional unit accesses are quite simple mainly because they select only one entry. But, as soon two or more units share several entries, the arbitrations become more complex and time-consuming. Indeed, concurrent arbitration processing in two logic layers requires a number of gates which evolves in $O(n^3)$, n being the number of entries to be arbitrated. In our case, n is the overall number of entries of the instruction-issue buffer. Moreover, some arbitrations cannot be processed in parallel since their results are mutually dependent. Several units of the same type are implemented and therefore, same entries may be selected. Consequently, it is necessary to implement arbiters selecting more than one request. In the case of an arbiter selecting m requests, the number of layers rises with m. Finally, the interconnections between the unified buffer and the functional units are quite complex since each entry must be linked to all functional units. Nevertheless, the issue width can be lower than the number of functional units (11, 14, and 18 for a dispatch width of 4, 6, and 8 respectively). Simulations have pointed out that, in order to give almost the same performance, the issue widths must be equal to 8, 10, and 12 instructions per cycle for the same configurations. But as these common busses are shared by all units, their access must be arbitrated and this further complicates the issue logic of the buffer and should counterbalance the previous gain.

In brief, the unified topology is well-suited since it is storage-efficient, but it requires time-consuming arbitrations that cannot be overlapped. On the contrary, the split topology requires a bit less than thrice more entries and gives a poorer performance but with a possible shorter cycle time. The semi-unified topologies may be a good trade-off between cycle time and hardware costs.

7 Conclusion

Throughout this paper, we have studied the impact on performance of the size and the topology of the instruction-issue buffer in out-of-order processors featuring a dispatch width of 4, 6, and 8. The results are based on trace-driven simulations performed on the SPEC92 suite. We investigated various topologies and we have defined for each of them the size of the various parts of the issue buffer giving levels of performance close to those obtained with an ideal buffer. These sizes are recalled in table 10. From this table, we can note that as the instruction-issue buffer is further divided, the number of entries is higher for a poorer

W	Unified	PA8000-like	R10k-like	Sparc64-like	Split
4	28	36	56	68	76
6	48	60	92	120	134
8	72	88	132	176	198

Table 10: Total Sizes of the Issue Buffers

performance. A split buffer requires a bit less than thrice more entries than a unified buffer. Conversely, a unified buffer requires longer arbitration times for the issuing and this may increase the cycle time of the processor: the two independent arbiters implemented for each part in the PA-8000 confirms that such arbitrations are time-consuming. This may also explain the difference between the size of the parts featured in the PA-8000 and our results. Another more probable reason is that the instruction-issue buffer plays also the role of the reorder buffer.

To conclude, choosing the topology of the instruction-issue buffer in an out-of-order processor results from architectural and technological trade-offs. The various topologies adopted by manufacturers emphasizes this conclusion.

Acknowledgements

We would like to thank T.M. Conte for providing us some of his papers, the referees for their useful comments, and Glenn Farrall for his careful review.

References

[1] M. Butler and Y. N. Patt, "An Investigation of the Performance of Various Dynamic Scheduling Techniques," Proceedings of the 25th Annual International Symposium on Microarchitecture, December 1992.

[2] T.M. Conte, K.N.P. Menezes, and S.W. Sathaye, "A Technique to Determine Power-Efficient, High-Performance Superscalar Processors," Proceedings of the 28th Annual Hawaii International Conference on System Sciences, January 1995.

[3] L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," Microprocessor Report, Vol. 9 Num. 2, 1995.

[4] L. Gwennap, "Pa-8000 Combines Complexity and Speed," Microprocessor Report, Vol. 8 Num. 15, 1994.

[5] W.W. Hwu, Y.N. Patt, "Checkpoint Repair for Out-of-Order Execution Machines," Proceedings of the 14th Annual International Symposium on Computer Architecture, Pittsburgh, June 1987.

[6] IBM and Motorola, "PowerPC 604 RISC Microprocessor Technical Summary," Advance Information, MPR604TSU-1, 1994.

[7] IBM and Motorola, "PowerPC 620 RISC Microprocessor Technical Summary," Advance Information, MPR620TSU-1, 1994.

[8] M. Johnson, *Superscalar Microprocessor Design,* Prentice-Hall, 1991.

[9] S. Jourdan, P. Sainrat, and D. Litaize, "Exploring Configurations of Functional Units in an Out-of-Order Superscalar Processor," Proceedings of the 22th Annual International Symposium on Computer Architecture, June 1995.

[10] Mips technologies Incorporated, "R10000 Microprocessor Product Overview," Technical Report, October 1994.

[11] S. Simone et al., "Implementation Trade-offs in Using a Restricted Data Flow Architecture in a High Performance RISC Microprocessor," Proceedings of the 22th Annual International Symposium on Computer Architecture, June 1995.

[12] J.E. Smith and A. R. Pleszkun, "Implementation of Precise Interrupts in Pipelined Processors," Proceedings of the 12th Annual International Symposium on Computer Architecture, June 1985.

[13] M.D. Smith, "Tracing with Pixie," Technical report, Stanford University, April 1991.

[14] "SPEC 92", Technical report, December 1992.

[15] J.E. Thornton, *Design of a Computer: The Control Data 6600,* Glenview, Foresman, 1970.

[16] S. Weiss and J.E. Smith, *POWER and PowerPC: Principles, Architecture and Implementation,* Morgan Kaufmann Publishers Inc., 1994.