Energy Efficient Register Renaming[1]

Gurhan Kucuk, Oguz Ergin, Dmitry Ponomarev, Kanad Ghose

Department of Computer Science,
State University of New York, Binghamton, NY 13902–6000
{gurhan, oguz, dima, ghose}@cs.binghamton.edu
http://www.cs.binghamton.edu/~lowpower

Abstract. Modern microprocessor designs implement register renaming using register alias tables (RATs), which maintain the mapping between architectural and physical registers. Because of the non–trivial power that is dissipated in a disproportionately small area, the power density in the RAT is significantly higher than in some other datapath components. In this paper, we propose mechanisms to reduce the RAT power and the power density by exploiting the fundamental observation that most of the generated register values are used by the instructions in close proximity to the instruction producing a value. Our first technique disables the RAT lookup for a source register if that register is a destination of an earlier instruction dispatched in the same cycle. The second technique eliminates some of the remaining RAT read accesses even if the source register value is produced by an instruction dispatched in an earlier cycle. This is done by buffering a small number of recent register address translations in a set of external latches and satisfying some RAT lookup requests from these latches. The net result of applying both techniques is a 30% reduction in the RAT energy with no performance penalty, little additional complexity and no cycle time degradation.

1 Introduction

Dynamically scheduled superscalar processors use aggressive out–of–order execution mechanisms to maximize performance by harvesting available parallelism in the sequential programs. Each successive generation of superscalars increases the number of instructions that are issued in a cycle and also uses larger instruction windows in order to consider more instructions for scheduling. The inevitable consequence of such an approach is a dramatic increase in the overall datapath complexity, power consumption and also power density, especially in high–frequency implementations. While in the past power was a consideration mainly in the domain of embedded systems, today it is an important design constraint for high–performance microprocessors. Unless power dissipation is controlled through technology–independent techniques, the areal power density will soon become comparable to that of nuclear reactors [11] leading to intermittent and permanent failures on the die and also creating serious challenges for the cooling facilities. Furthermore, the areal power density distribution across a typical chip is highly skewed, being lower over the on–chip caches and significantly higher elsewhere, resulting in the presence of the localized hot spots on the chip. The non–uniform thermal stresses that result are problematic.

One on–chip structure with a high power density is the Register Alias Table (RAT). RAT maintains the register address translations needed for handling the true data

[1] supported in part by DARPA through contract number FC 306020020525 under the PAC–C program, the NSF through award no. MIP 9504767 & EIA 9911099 and IEEC at SUNY Binghamton.

dependencies. True data dependencies are handled by assigning a new physical register for every new result that is produced into a register. The RAT maintains information to locate the most recent instance of an architectural register. Register renaming is a technique used in all modern superscalar processors to cope with false data dependencies by assigning a new physical register to each produced result. The mappings between the logical and the physical registers are maintained in the RAT, so that each instruction can identify its source *physical* registers by performing the RAT lookups indexed by the addresses of the source *logical* (architectural) registers. The read and write accesses to the RAT, as well as the actions needed for checkpointing and the state restoration, result in a significant amount of power dissipated in the RAT. For example, about 4% of the overall processor's power is dissipated in the rename unit of the Pentium Pro [8]. Even higher percentage of the overall power – 14% – is attributed to the RAT in the global power analysis performed in [4]. When coupled with the relatively small area occupied by the RAT, this creates a hot spot, where the power density is significantly higher than in some other datapath components, such as the on–chip caches.

In this paper, we introduce mechanisms to reduce the RAT power and the power density by exploiting the fundamental observation that most of the generated register values are used by the instructions that are in close proximity to the instruction producing a value. Specifically, we propose two methods to reduce the RAT power dissipation. Our first technique disables the RAT lookup for a source register if that register is a destination of an earlier instruction dispatched in the same cycle. The second technique eliminates some of the remaining RAT read accesses even if the source register value is produced by an instruction dispatched in an earlier cycle.

The rest of the paper is organized as follows. To motivate our work, we describe the RAT complexities and the sources of associated energy dissipations in Section 2. We present the details of our energy reduction techniques in Sections 3 and 4. Our simulation methodology is described in Section 5, and we present and discuss the simulation results in Section 6. Section 7 reviews the related work and we offer our concluding remarks in Section 8.

2 The RAT Complexity

For this study, we used a RISC–type ISA, where the instructions may have at most two source registers and one destination register. We further assumed that the RAT is implemented as a multi–ported register file, where the number of entries is equal to the number of architectural general–purpose registers in the ISA. The RAT is indexed by the architectural register address to permit a direct lookup. The width of each RAT entry is equal to the number of bits in a physical register address. An alternative design is to have the number of entries in the RAT equal to the number of physical registers, such that each RAT entry stores the logical register corresponding to a given physical register and a single bit to indicate if the entry corresponds to the most recent instance of the architectural register. In this scheme, as implemented in the Alpha 21264 [7], the RAT lookup is performed by doing the associative search using the logical register address as the key. We did not consider this variation of the RAT in this paper, because it is inherently less energy–efficient than the direct–lookup implementation, due to the large dissipations that occur during frequent associative lookups. One way to address this problem is to use a recently proposed dissipate–on–match comparator [3] in the associative logic within the RAT, but such an evaluation is beyond the scope of this paper.

In a W–way superscalar machine, up to W instructions may undergo renaming in the same cycle. Thus, 2*W register address translations may have to be performed in a cycle to obtain the physical addresses of the source registers. In addition, up to W new physical registers may have to be allocated to hold the new results. The number of ports needed on the RAT in a W–way superscalar machine is quite significant. Specifically, 2*W read ports are needed to translate the source register addresses and W write ports are needed to update W RAT entries for the destinations of the co–dispatched instructions. In addition, before the destination register mapping is updated in the RAT, the old value has to be checkpointed in the reorder buffer for possible branch misprediction recovery. If the instruction that overwrites the entry is later discovered to be on the mispredicted path, the old mapping, saved within the reorder buffer, is used to restore the state of the RAT. W read ports, needed for such checkpointing, bring the total port requirements on the RAT to 3*W read ports and W write ports.

The energy dissipations take place in the RAT in the course of the following events:

1) Obtaining physical register addresses of the source operands.

2) Checkpointing the old mapping of the destination register.

3) Writing to the RAT for establishing the new mapping for the destination register.

3 Exploiting the Intra–Group Dependencies

In a superscalar machine, there may be sequential dependencies among the group of instructions that are co–dispatched within a cycle. To take care of such dependencies, register renaming must logically create the same effect as renaming the instructions individually and sequentially in program order. A sequential implementation of register renaming will be too expensive and will dictate the use of a slower clock. To avoid this, the accesses to the RAT and dependencies are handled as follows:

Step 1. The following substeps are performed in parallel:

(a) RAT reads for the sources of each of the co–dispatched instructions are performed in parallel, assuming that no dependencies exist among the instructions.

(b) New physical registers are allocated for the destination registers of all of the co–dispatched instructions.

(c) Data dependencies among the instructions are noted, using a set of comparators. The address of each destination register in a group of instructions is compared against the sources of all following instructions in the group and if a match occurs, the dependency is detected.

Step 2. If a data dependency is detected among a pair of instructions, the source physical register for the dependent instruction as read out from the RAT, is replaced with the allocated destination register address of the instruction producing the source to preserve the true dependencies. After resolving dependencies as described, the RAT is updated with the addresses of the new destinations.

The above steps, including the concurrent substeps within Step 1, avoid a bottleneck that would otherwise result from performing the RAT lookup and dependency handling in a strictly sequential manner among the co–dispatched instructions. The price paid in this approach is in the form of redundant accesses to the RAT for the mappings of source registers that are renamed because of dependencies within the group of co–dispatched instructions. If a dependency is not detected, the source mapping obtained from the RAT is used, otherwise it is tossed out.

A considerable amount of energy is expended within the multi–ported register file that implements the RAT because of the concurrent accesses to it within a single cycle. It is thus useful to consider techniques for reducing the energy dissipation within the RAT. Our first proposed solution disables parts of the RAT read accesses if the intra–group data dependency is noted, as described by Step 1(c). The outputs of the comparators, corresponding to a given source, are NOR–ed and the output of the NOR gate is used as the enabling signal for the sensing logic on the bitlines used for reading the physical address mapping of this source from the RAT. For example, consider three instructions – I1, I2 and I3 (in program order) that are renamed in the same cycle. A source for the instruction I3 is compared for possible intra–group dependencies against the destination of I1 and the destination of I2. If one of these comparators indicates a match (the output of the comparator stays precharged at the logical "1"), the output of the NOR–gate becomes zero and the sense amps used for reading the bitlines of a source of the instruction I3 are not activated, thus avoiding the energy dissipation in the course of sensing. Measurable power savings can be realized, because sense amps contribute to a large fraction of the overall read energy. In the rest of the paper we abbreviate this technique as the *CSense* (Conditional Sensing).

Figure 1 shows the percentage of the RAT lookups for the source registers that can be aborted if the *CSense* is used for a 4–way and a 6–way processor. Results are presented for the simulated execution of a subset of the SPEC 2000 benchmarks, including both integer and floating point codes. Detailed processor configurations are described in Section 5. On the average across all simulated benchmarks, around 31% of the RAT read accesses can be aborted for a 4–way processor and around 41% for a 6–way processor. The latter number is higher, because more instructions are dispatched in the same cycle, thus increasing the possibility that the most recent definition of a source register is within the same instruction group. Notice, finally, that our technique does not prolong the cycle time, because the output of the NOR gate is driven to the sense amp before the wordline driver completes the driving of the word line. Our simulations of the actual RAT layouts in a 0.18 micron 6–metal layer TSMC process (Section 5) show that the decoder delay is about 150 ps, the delay of the wordline driver is about 100 ps, the bitline delay to create the small voltage difference across the bitline pair is 60 ps and at that point the sense amp is activated. The comparator delay is about 120 ps and the delay of the three–input NOR–gate is 60 ps. Consequently, the signal that controls the activation of the sense amp is available 180 ps after the beginning of the cycle, while the sense amp is normally activated after 310 ps are elapsed since the beginning of the cycle. The sense amp control signal is thus available well in advance of when it needs to be used, leaving enough time to route the signal to the sense amps, if need be. These delays were obtained using highly optimized handcrafted layouts of the RAT assuming 32 architectural registers and 4–way wide dispatch/renaming. Therefore, the *CSense* can be applied without any increase in the cycle time.

4 Buffering Recent Address Translations

Our simulations of the SPEC 2000 benchmarks show that the dependent instructions are usually very close in proximity to each other. If the register needed as a source is not defined by an earlier instruction dispatched in the same cycle, then it is likely defined by an instruction dispatched one or at most a few cycles earlier. We exploit this behavior by caching recently updated RAT entries in a small number of associatively–addressed

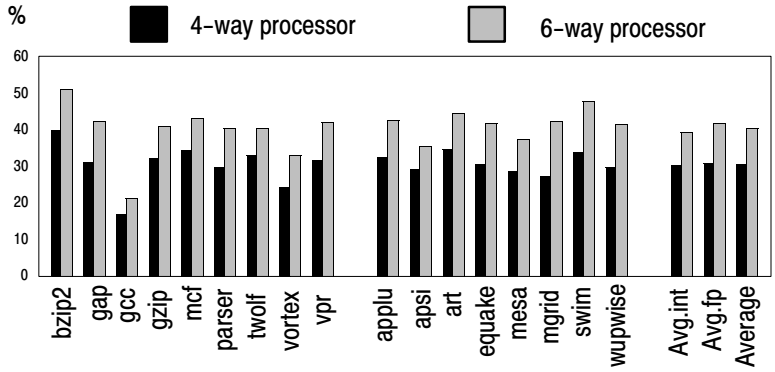

Fig. 1. The percentage of the source operands that are produced by the instructions co–dispatched in the same cycle

latches (ELs), external to the RAT. The basic idea of avoiding a RAT access using ELs was inspired by the work of [5], where multiple line buffers were used to reduce the overall cache energy dissipations. The RAT access for a source register now proceeds as follows:

(a) Start accessing the RAT and at the same time address the ELs to see if the desired entry is located in one of the ELs.

(b) If a matching entry is found, discontinue the access from the RAT.

As long as the overhead of accessing the ELs is less than the energy spent in accessing the RAT before the RAT accessing is aborted, this technique will result in an overall energy saving.

Figure 2 depicts a RAT with multiple ELs (4 in this case). The hardware augmentations to the basic register renaming structure are as follows. First, we need four latches to hold the addresses of each of the four most recently accessed architectural registers. Second, four comparators are used to compare the address of the architectural register, whose lookup in the RAT is being performed against the register addresses located in the four ELs.

Assuming a two–phase clock, the access steps for a RAT with multiple external latches are as follows:

Phase 1:

(a) Precharge the RAT for a read access.

(b) Start the decoding of the register address.

(c) Simultaneously, compare the register address with the register addresses stored in the latches.

Phase 2:

If a match occurs to an ELs (a latch hit), abort the readout from the RAT. Otherwise (on a latch miss), proceed with the regular RAT access.

Notice that on a read miss on the associatively–addressed ELs, the data is not brought from the RAT array into the latches. This is so for the following reason. A large percentage of the register values are consumed by just one instruction; this was observed in [2] and also noticed in our simulations. If this is the case, then bringing the data that was once read from the RAT into the external latches will only result in polluting the

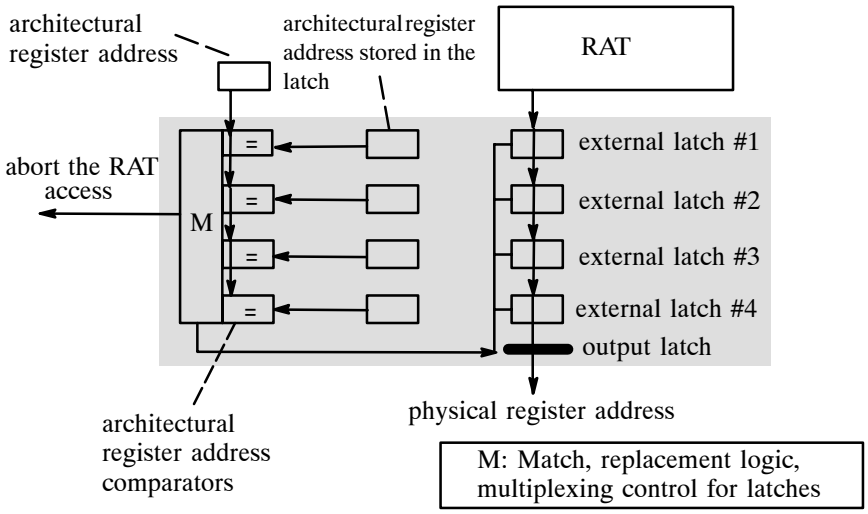

Fig. 2. Renaming Logic with Four External Latches (ELs)

latches with unusable data in most situations. In addition, extra energy dissipation occurs if such data movement is needed. Because of this, we only record the translation in the ELs when the physical register is allocated. In other words, the update of the RAT and the update of the external latches proceed in parallel. A victim EL is selected randomly for setting up the new entry. Notice, that as a consequence of this policy, there is no need to write the translation information back to the RAT once an entry is evicted from the ELs.

It is critical to limit the energy spent in associatively addressing the ELs. To accomplish this, we make use of comparators that dissipate energy on a match (for example, as introduced in [3]), instead of traditional CAM cells or comparators that dissipate energy on a mismatch. The use of these comparators actually helps in two ways. First, as at most one of the external latches will have the matching entry, energy is dissipated within at most one comparator during the associative lookup. Second, the comparators of [3] actually have a faster response time than the traditional comparators or CAM cell. (The difference in timing between the comparator of [3] and the traditional comparator is, however, very small so even the traditional pull–down comparator can be used in our scheme, albeit with higher power dissipated in the latches.) For the same reason as in the *CSense* scheme, the detection of a match in the external latches is completed well in advance of the normal sense amp activation. Therefore, the energy dissipation in the sense amps can be avoided without compromising the cycle time. These savings exceed that spent in locating a matching entry within a latch array consisting of four external latches, as shown in the result section. Also, the use of dissipate–on–match comparators within the intra–group dependency checking logic is not an attractive solution from the energy standpoint because of a higher percentage of match situations. Our detailed analysis, using microarchitectural data about the bit patterns of the comparands indicate that the use of traditional comparators is a more energy–efficient approach for the use in the intra–block dependency checking logic.

5 Simulation Methodology

To evaluate the energy impact of the proposed techniques, we designed and used the AccuPower toolsuite [10]. The widely–used Simplescalar simulator [1] was significantly modified to implement *true hardware level, cycle–by–cycle* simulation models for realistic superscalar processor. The main difference from the original Simplescalar code is that we split the Register Update Unit into the issue queue, the reorder buffer and the physical register file. It is important, because in real processors the number of entries in all these structures, as well as the number of ports to these are quite disparate. The configuration of a simulated 4–way superscalar processor is shown in Table 1. For simulating a 6–way machine, we increased the window size and the cache dimensions proportionately.

We simulated the execution of 9 integer *(bzip2*, *gap*, *gcc*, *gzip*, *mcf*, *parser*, *twolf*, *vortex* and *vpr*) and 8 floating point (*applu*, *apsi*, *art*, *equake*, *mesa*, *mgrid*, *swim* and *wupwise*) benchmarks from SPEC 2000 suite. Benchmarks were compiled using the Simplescalar gcc compiler that generates code in the portable ISA (PISA) format. Reference inputs were used for all the simulated benchmarks. The results from the simulation of the first 2 billion instructions were discarded and the results from the execution of the following 200 million instructions were used for all of the benchmarks.

For estimating the energy/power dissipations for the key datapath components, the event counts gleaned from the simulator were used, along with the energy dissipations measured from the actual VLSI layouts using SPICE. Hand–crafted CMOS layouts for the RAT in a 0.18 micron 6 metal layer CMOS process (TSMC) were used to get an accurate idea of the energy dissipations for each type of transition. A 2 GHz clock and a V_{dd} of 1.8 volts were assumed for all the measurements.

Table 1. Architectural configuration of a simulated 4–way superscalar processor

Parameter	Configuration
Machine width	4–wide fetch, 4–wide issue, 4–wide commit
Window size	32 entry issue queue, 96 entry ROB (integrating physical register file), 32 entry load/store queue
Function Units and Latency (total/issue)	4 Int Add (1/1), 1 Int Mult (3/1) / Div (20/19), 2 Load/Store (2/1), 4 FP Add (2), 1FP Mult (4/1) / Div (12/12) / Sqrt (24/24)
L1 I–cache	32 KB, 2–way set–associative, 32 byte line, 2 cycles hit time
L1 D–cache	32 KB, 4–way set–associative, 32 byte line, 2 cycles hit time
L2 Cache combined	512 KB, 4–way set–associative, 128 byte line, 4 cycles hit time
BTB	4096 entry, 4–way set–associative
Branch Predictor	Combined with 1K entry Gshare, 10 bit global history, 4K entry bimodal, 1K entry selector
Memory	128 bit wide, 60 cycles first chunk, 2 cycles interchunk
TLB	64 entry (I), 128 entry (D), fully associative, 30 cycles miss latency

6 Results and Discussions

Figure 3 shows the hit ratio to the ELs – that is, the percentage of the RAT accesses that can be satisfied from the ELs. Separate results are presented for integer and floating point registers, with the use of 4 and 8 ELs. With the use of 4 ELs, the average hit ratio is about 44% for the integer ELs, and about 30% for the floating–point ELs. Adding four more entries to the ELs increases the hit ratios only slightly (about 53% for integer, and about 38% for floating–point ELs) because, again, the use of most register values is very close in proximity to the definitions of those registers. On the other hand, a higher number of ELs increases the complexity and power.

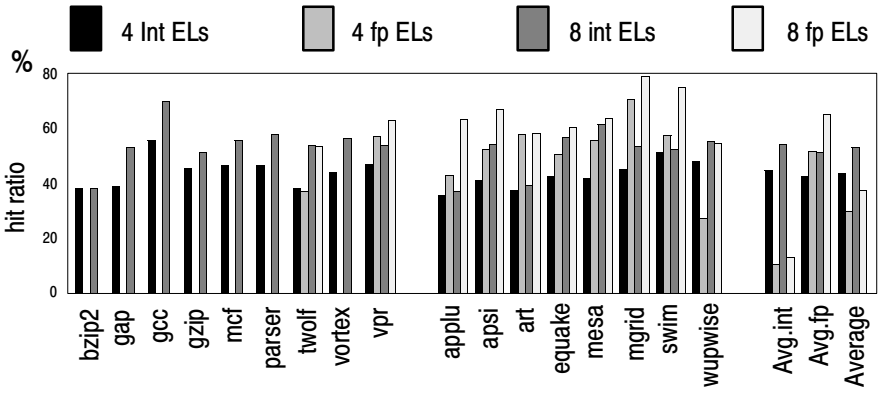

Fig. 3. The hit ratio to integer and floating–point External Latches (ELs)

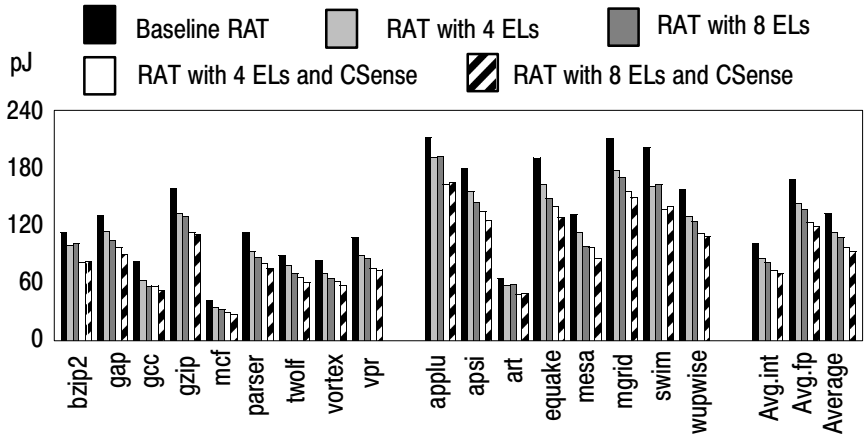

Fig. 4. Energy of the baseline and proposed RAT designs

The percentages shown in Figure 3 do not account for the matches within the co–dispatched group, as those matches are detected by the intra–group dependency checking logic. Combined, the percentages shown in Figure 1 and Figure 3 represent the total percentage of cases when the sense amps can be disabled if the two techniques are used in conjunction.

Figure 4 shows the energy reduction achievable by applying the proposed techniques. The first bar shows the energy dissipation of the baseline RAT. The second and third bars show the energy impact of adding four and eight ELs, respectively. The last two bars show the energy reduction in the RAT if ELs are used in conjunction with the *CSense*. The average energy savings with the use of four and eight ELs are around 15% and 19%, respectively. The combination of both techniques results in about 27% energy savings on the average for four ELs and 30% energy reduction for eight ELs. Our analysis also indicate that the use of eight ELs is the optimal configuration, because the overall energy increases if the number of ELs goes beyond eight, due to the additional complexity and power of managing the ELs. At the same time, the percentage of EL hits stays nearly unchanged, as described earlier.

7 Related Work

Moshovos proposed to reduce the power consumption of the register alias table in [9]. The proposed optimizations reduce power of the renaming unit in two ways. First, the number of read and write ports needed on the register alias table is reduced. This is done by exploiting the fact that most instructions do not use the maximum number of source and destination register operands. Additionally, the intra–block dependence detection logic is used to avoid accessing the register alias table for those operands that have a RAW or a WAW dependence with a preceding, simultaneously decoded instruction. The technique of [8] only steers the sources that actually need to be translated through the RAT lookup to the RAT ports. The source addresses that are not used (for example, some instructions have only one source) or those produced by the instruction in the same block do not have to be translated through the RAT array. All sources that need to be translated using the RAT access are first identified and then they are steered to the available RAT ports. If a port is not available, the renaming blocks. This incurs an inherent performance penalty in terms of IPCs and, in addition, stretches the cycle time, because the filtering of source addresses and the RAT accesses are done serially. Simulation results show that for an aggressive 8–way superscalar machine it is possible to reduce the number of read ports from 24 to 12 and the number of write ports from 8 to 6 with a performance penalty of only 0.5% on the average across the SPEC 2000 benchmarks. The second optimization reduces the number of checkpoints that are needed to implement aggressive control speculation and rapid recovery from the branch mispredictions. This is done by allowing out–of–order control flow resolution as an alternative to conventional in–order resolution, where the checkpoint corresponding to a branch can not be discarded till this branch itself as well as all preceding branches are resolved.

In [6], Liu and Lu suggested using the hierarchical RAT. A small, first–level RAT is used to hold the mappings of the most recent renamed registers. Instructions access this small RAT first and only on a miss access the large full–blown second–level RAT. Because the process is serialized, the performance degradation is unavoidable as at least one extra cycle is needed in the front–end of the pipeline, thus increasing the branch misprediction penalty.

In contrast to these techniques, our proposed mechanisms do not have any performance penalty, nor do they increase the cycle time of the processor.

8 Concluding Remarks

We proposed two complementary techniques to reduce the energy dissipation within the register alias tables of modern superscalar microprocessors. The first technique uses the intra–group dependency checking logic already in place to disable the activation of the sense amps within the RAT when the register address to be read is redefined by an earlier co–dispatched instruction. The second technique extends this approach one step further by placing a small number of associatively–addressed latches in front of the RAT to cache a few most recent translations. Again, if the register translation is found in these latches, the activation of the sense amps within the RAT is aborted. Combining the two proposed techniques results in a 30% reduction in the power dissipation of the RAT. The power savings comes with no performance penalty, little additional complexity and no increase in the processor's cycle time.

References

1. Burger, D. and Austin, T. M., "The SimpleScalar tool set: Version 2.0", Tech. Report, Dept. of CS, Univ. of Wisconsin–Madison, June 1997 and documentation for all Simplescalar releases.
2. Cruz, J–L., Gonzalez, A., Valero, M. et. al., "Multiple–Banked Register File Architecture", *in Proceedings 27th International Symposium on Computer Architecture*, 2000, pp. 316–325.
3. Ergin, O., et.al., "A Circuit–Level Implementation of Fast, Energy–Efficient CMOS Comparators for High–Performance Microprocessors", *in Proceedings of ICCD*, 2002.
4. Folegnani, D., Gonzalez, A., "Energy–Effective Issue Logic", *in Proceedings of International Symposium on Computer Architecture*, July 2001.
5. Ghose, K. and Kamble, M., "Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit–Line Segmentation", *in Proceedings of International Symposium on Low Power Electronics and Design (ISLPED'99)*, August 1999, pp.70–75.
6. Liu, T., Lu, S., "Performance Improvement with Circuit Level Speculation", *in Proceedings of the 33rd International Symposium on Microarchitecture*, 2000.
7. Kessler, R.E., "The Alpha 21264 Microprocessor", IEEE Micro, 19(2) (March 1999), pp. 24–36.
8. Manne, S., Klauser, A., Grunwald, D., "Pipeline Gating: Speculation Control for Energy Reduction", *in Proceedings of the 25th International Symposium on Computer Architecture (ISCA)*, 1998, pp. 132–141.
9. Moshovos, A., "Power–Aware Register Renaming", Technical Report, University of Toronto, August, 2002.
10. Ponomarev, D., Kucuk, G., and Ghose, K., "AccuPower: an Accurate Power Estimation Tool for Superscalar Microprocessors", *in Proceedings of 5th Design, Automation and Test in Europe Conference (DATE–02)*, March, 2002.
11. Pollack, F., "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies", Keynote Presentation, *32nd International Symposium on Microarchitecture,* November 1999.