Energy-Efficient Register Renaming in High-Performance Processors[*]

José L. Ayala, Marisa López-Vallejo
Departamento de Ingeniería Electrónica
Universidad Politécnica de Madrid
Spain
{jayala,marisa}@die.upm.es

Alexander Veidenbaum
Center for Embedded Computer Systems
University of California, Irvine
USA
alexv@ics.uci.edu

ABSTRACT

Multiported register files are a critical component of high-performance superscalar processors. Conventional multiported structures consume significant power. This paper presents the hardware component of a hardware/compiler approach to manage the register file based on program behavior and achieve higher energy efficiency in out-of-order processors. The proposed approach outperforms previous schemes in both performance and power savings. With no performance penalty, the mechanism achieves more than 60% energy savings with simple and reduced extra logic. Additional savings can be obtained when this technique is applied with compiler support.

Categories and Subject Descriptors

C.5 [**Computer System Implementation**]: Microcomputers—*microprocessors*

General Terms

Register File management

Keywords

register file management, hardware mechanism, energy aware, out-of-order processor, compiler support

1. INTRODUCTION

Power dissipation has become a critical issue for both high performance and mobile processors. Dynamic power dissipation has been usually the dominant factor, but static power dissipation is becoming increasingly significant in upcoming processors. While dynamic power is directly related to the activity of the circuits, static power depends on the amount of power-on transistors and their physical characteristics. Current generation high-end processors like the dual-core IBM POWER4$^{\mathrm{TM}}$ [18] are performance-driven designs where overall power densities are reportedly still below acceptable limits [8], even though the net chip power is well over 100 watts [3]. Thus, it is worthwhile to devise microarchitectural techniques to reduce power and power density in the front-end, without sacrificing performance for high-end systems.

In high performance wide-issue microprocessors the register file often plays a critical role in determining the cycle time, directly through its access time and indirectly through its size. Furthermore it accounts for a significant fraction of the processor core's power consumption. These register files need to be large to support multiple in-flight instructions and multiported to avoid stalling the instruction issue. In the Alpha 21464, the register file design was several times larger than the 64 KB primary caches [9] and was split to reduce cycle time impact. Both large size and high number of ports result in slow access and high energy dissipation. Additionally, that structure is one of the processor hotspots. Thus, reducing power consumption in this power hungry structure is critical not only from the energy standpoint but also from the temperature standpoint [1].

In this paper, we propose an innovative microarchitecture that achieves significant energy savings in the register file without any performance loss. The approach is suitable for deeply pipelined superscalar out-of-order processors. Our proposal is based on a deterministic variation of the renaming algorithm which allows to turn-off many registers of this structure if they are not used, and thus, power is saved. The required registers are turned-back on in advance and no performance penalty is accomplished. For the functional implementation of this idea, a banked organization of the register file is used and the low-power state of the memory cells is accomplished by using *"drowsy"* memory cells [10]. The approach is also supported by several experimental facts that show how the register assignment is performed by the compiler: related registers are assigned first. The work presented here is a first attempt that supposses a general compiler without specific modifications. Future research work modifies this compiler to improve the energy savings obtained.

The rest of the paper is organized as follows: section 2 summarizes the previous work on this topic and section 3 presents the baseline architecture and fundamentals used

[*]This work was supported by the Spanish Ministry of Science and Technology under contract TIC2000-0583-C02-02

for our research work. Section 4 describes the technique and designed architecture, while the experimental results are shown in section 5. Finally, some conclusions will be drawn in section 6.

2. RELATED WORK

Many techniques have been proposed to reduce the area, energy, and delay of multiported register files. Some approaches split the microarchitecture into distributed clusters, each containing a subset of the register file and functional units [15, 21]. These schemes have the potential to scale to larger issue widths but require complex control logic to map instructions to clusters and to handle inter-cluster dependencies. Alternatively, other approaches retain a centralized microarchitecture, but divide the physical register file into interleaved banks, with fewer ports per bank [13]. Provided the number of simultaneous accesses to any bank is less than the number of ports on each bank, the structure can provide the aggregate bandwidth needs of a superscalar machine with significantly reduced area compared to a fully multiported register file. These banked schemes, however, require complex control logic with pipeline stalls that would likely limit the cycle time of a high-frequency design. Recent research on this area has reported significant energy savings (around 40%) but still the performance penalty could not be acceptable (IPC decreases around 5%) [12, 19]. Also, joint efforts on decreasing the register file and the issue-queue energy consumption by means of adaptive approaches, reduce the impact on performance (around 1.77% IPC loss) while still achieve considerable energy savings [2].

Other authors have investigated how to reduce the power and complexity of the register files. Cruz et al. [7] proposed a multilevel register file organization for low complexity and fast access time to registers. Zyuban and Kogge [20] studied the complexity of a centralized register file and proposed a scheme to distribute it.

Our approach does not work with the register file itself, but modifies the assignment of free registers during the register renaming process. Therefore, it is orthogonal to those previous works and can be easily combined to achieve higher energy savings.

Finally, our research group has also solved this problem for in-order embedded processors by proposing a hardware mechanism [5] and a compiler technique [6] to reduce power consumption in the register file. The ideas proposed in those previous works have been extended and improved in the current work to cover high-performance out-of-order processors.

3. BASELINE ARCHITECTURE

In this section we will describe the fundamentals of register renaming and the microarchitectural design of the baseline architecture used in the present approach.

3.1 Background on Register Renaming

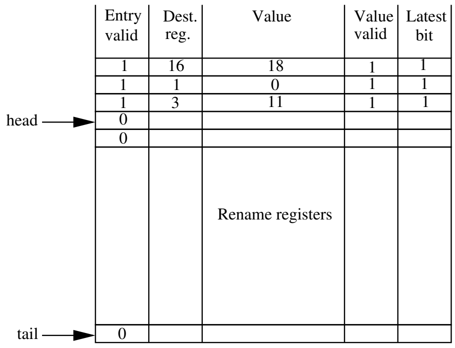

Figure 1: Allocation of a new rename buffer to the destination register

Register renaming is a standard technique for removing false data dependencies, that is, WAR (*write after read*) and WAW (*write after write*) dependencies, among register data. Register renaming may be implemented either statically or dynamically. In the case of static implementation, register renaming is carried out during compilation. This technique was first introduced in parallel compilers for pipelined processors, and later in superscalar processors. When register renaming is implemented dynamically, renaming takes place during execution. Obviously, this requires extra circuitry in terms of supplementary register space, additional data paths and logic [17]. Our research work is oriented to the power reduction in processors with dynamic (hardware) register renaming, as usually can be found in high-performance processors.

The layout of the rename buffers establishes the actual framework for renaming. Its three basic components are the type and the number of the rename buffers as well as the basic mechanism which is used for accessing rename buffers. The chosen type of rename buffers determines where the intermediate results of the instructions are to be written into or read from. Certainly, the most extended approaches in modern superscalar processors are the *separate register file* (when rename buffers are implemented separately from the architectural registers; i.e, rename register values are held in a separate register file), and the *reorder buffer* (when this structure is used to implement rename buffers). Alpha 21464 is an example of processor using the former approach, while Pentium IV is an example of the later.

The approach presented in this paper is developed for the separate implementation of the rename buffers. In our discussion about the implementation of register renaming alternatives we will consider only one aspect: the allocation of a free rename buffer to a destination register. Whenever an instruction referencing a destination register is issued, a new rename register needs to be allocated to the destination register concerned. This allocation is performed by looking for and reserving a free entry, and properly initializing its fields, in a FIFO of free registers (see Figure 1).

When a new entry is required, the entry indicated by the head pointer is selected for use and the pointer is stepped. If an entry is no longer needed, this entry is deallocated by updating the tail pointer. During allocation of a new rename register, the following updates have to be accomplished: setting the Entry valid and Latest entry bits[1], copying the destination register number into the corresponding field and resetting the Value bit.

The baseline architecture we have selected for the experimentation has 256 rename buffers (and, therefore, the same number of positions in the FIFO of free registers), and 64 logical registers assigned by the compiler. Half of these registers are Floating Point registers, while the other half are Fixed Point registers. This configuration represents a typical configuration for current-trend high-performance processors.

3.2 Dynamic Voltage Scaling

It has been shown that current processors present high energy consumption in the register file. Nevertheless, this device is underutilized most of the time. The register file is one of the units which limits the clock frequency in wide issue processors. With i instructions or micro-operations issued per cycle each consuming (up to) two operands and producing (up to) one result, the register file needs to support $2i$ reads and i writes per cycle. High clock frequencies require deeper pipelines combined with wide instruction issue and increased depth of speculation call for an even larger number of physical registers. Unfortunately the silicon area, the energy consumption and the access time of the register file are proportional to the number of write and read ports and the total number of physical registers.

These $2i+i$ registers have to be on to provide the source operands and store the results in a given cycle. But the rest of the N registers do not need to be on and are unused but energy-hungry resources. They are not being read or written by any instruction and should be turned into a low-power state.

Turning memory devices into a low-power state is not a brand new idea. Previous research has focused on turning off unused memory banks (or other resources) by gating the power source. When the objective of this technique is a memory device, the cost of recovering the lost information could hide any power saving or, at least, represent a very significant time penalty[2]. When working with the register file, there is no way to recover data from memory without extra accesses to the cache, and something has to be done in the unused registers to prevent the information from being lost. By turning these unused registers into a low-power state (*drowsy* state) the power consumption can be reduced to a minimum without data lost [11].

The information in a memory cell is preserved while it is in the drowsy state. However, the data line must be restored to a high-energy mode before its contents can be accessed. One way to implement drowsy memory devices was proposed by Flautner et al. in [10], where a dynamic voltage scaling (DVS) technique is exploited to reduce static power consumption. Due to short-channel effects in deep-submicron processes, leakage current is significantly reduced with voltage scaling. The combined effect of reduced leakage current and voltage yields a dramatic reduction in leakage power. This is the solution used in our approach to reduce energy consumption.

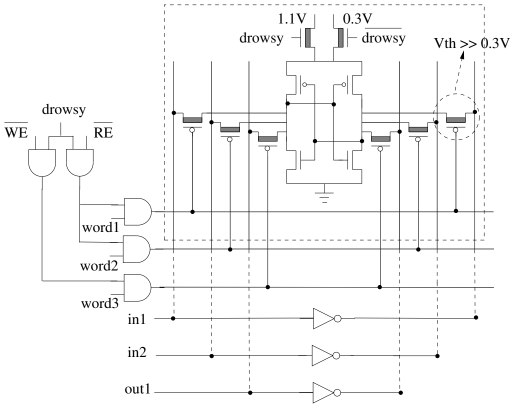

Figure 2: Register file cell

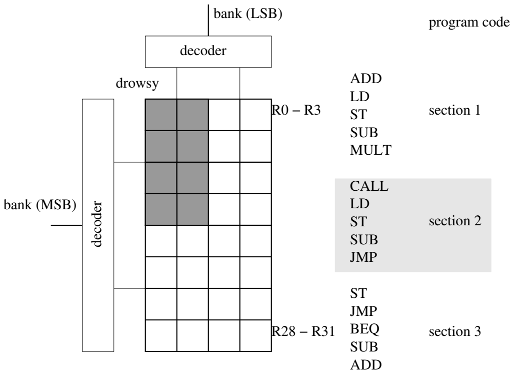

Figure 3: Bank of registers

Figure 2 shows the modified register file cell used to support the drowsy state. As can be observed, the dual power supply is switched to low V_{DD} when the cell is in drowsy state. It is necessary to use $high - V_{th}$ devices as pass transistors because the voltage on bit lines could destroy the cell contents. Before a register cell can be accessed, the power supply has to be switched to high V_{DD} to restore its contents and allow the access. An extra read_enable/write_enable gating circuit assures the memory cell is not accessed (read/written) while being in drowsy state.

The proposed solution deals with the "scalable" register file as follows. Instead of working with the whole register file, only the registers belonging to one (or more) of the predefined banks will be kept on. Once the register file bank that contains the required registers is known, the other registers are turned into a low power state by keeping the corresponding banks in the drowsy state. In this way, the power consumption is reduced to a minimum and the clock distribution network is gated as well (see Figure 3).

[1] Latest bit is necessary because a particular destination register may be renamed repeatedly.

[2] For example, to load the lost data from main memory.

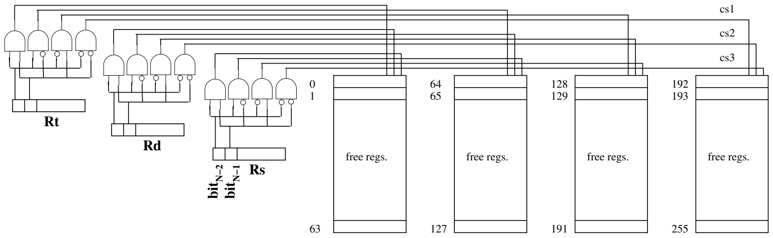

Figure 4: Modified allocation of free registers

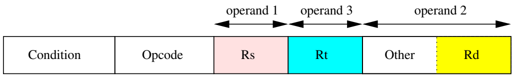

Figure 5: General instruction format

fetch	decode	issue	execute	memory	write back
– Forwarding of labels – Selection of FIFOs – Regs. OFF	– Reading of FIFOs – Prediction errors are corrected	– Regs. ON	– Selection of FIFO for output reg.	– Output reg. ON	

Figure 6: Out-of-order timing

The amount of power that can be saved will be determined by the number of banks kept in the low-power mode. Several experiments we carried out show that the compiler assigns correlative registers and, consequently, only one of the register file banks has to be on most of the time, increasing the power savings. A power-aware compiler designed to maximize the locality (proximity) of assigned registers would improve the power saving results.

As a result, the product $V_{dd} \times I_{dd}$ is reduced for all the register banks kept in the drowsy state due to the decrease in the voltage power supply, and thus the total static power consumption is also reduced.

The independent control of every register in the register file would allow higher energy savings (in this case, only 3 registers per instruction would be on, the 2 sources required by the current instruction and the destiny required by the former instruction in the pipeline). However, the complexity of the required logic needed to perform this approach, as well as the penalty on energy and area due to the extra logic and routing, advises against this.

Due to the necessity of restoring the register contents previously to the access, the accessed registers must be known at least one cycle before the access happens. However, the allocation of a free register is a non deterministic task and it is not possible to know in advance the rename buffer assigned to a destination register. The main goal of this work is to reduce this indetermination so that the required rename buffer can be known in advance to be turned on.

4. DESIGNED ARCHITECTURE

As has been previously explained, the objective of this approach is to keep the unused registers in a low-power state, and switch these back on before they are required. In order to accomplish this goal, the allocation of free registers has to be modified to reduce the indetermination related to this task. Therefore, attending to the logical registers coded in the instruction word, the set of physical registers possibly assigned during the renaming phase will be known in advance. The banks of registers will be kept on, while the other banks of registers will be turned into a low-power state to save power.

There are two main changes in the processor architecture that have to be performed by the proposed approach: firstly, to modify the FIFO of free registers found in the register assignment module; and secondly, to deal with the new power aware policy in the pipeline execution. Next two sections will describe this.

4.1 New FIFO structure

The first step is to split the 256-position FIFO of free registers into four 64-position FIFOs. The size of these FIFOs has to be large enough to provide rename buffers to the destination registers but avoiding to become empty. [3] In this way, the problem of indetermination has been mostly solved. Instead of assigning free rename buffers from the same FIFO to the whole set of logical registers, the first set of 16-logical registers is assigned to the first FIFO, the second set to the second FIFO, the third to the third one and the last set of 16-logical registers is assigned to the last 256-position FIFO (Figure 4).

Splitting the FIFOs can represent a small penalty in terms of energy and area due to the need of an extra port. This extra port allows to assign a free register for every one of the three possible operands in an instruction word, while the register assigned to the former write instruction in the pipeline is freed. To check these possible side effects we have performed a comparative study between our approach and the conventional implementation of the register renaming. Table 1 shows the parameters obtained with CACTI [16] for both FIFO configurations and for a $0.18\mu m$ technology. As can be seen, the area and energy penalties are negligible compared with the total area and power consumption of the register file and renaming logic. On the other hand, the access time is decreased for the smaller FIFO, assuring in this way the delay is under the clock period threshold.

[3] The correctness of the 64-position choice has been validated by simulation, as will be lately shown.

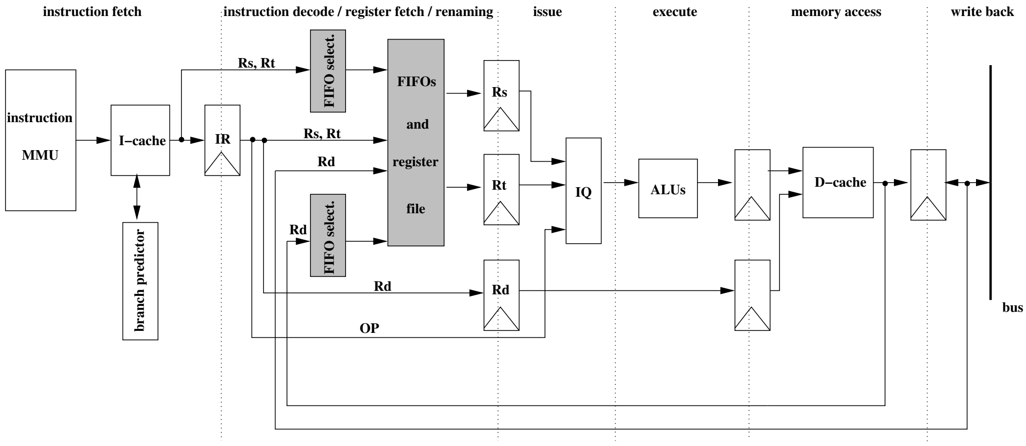

Figure 7: Schematic of the general architecture

	256-position	64-position
Access time (ns)	2.08988	1.54804
Total area (cm^2)	0.013766	0.008798
Total power (nJ)	0.595781	0.275265

Table 1: Delay, area and energy for the two FIFO configurations

Whenever a rename buffer has to be assigned to a destination register, the source FIFO providing the rename buffer is selected by a simple logic. This simple logic (a reduced number of gates) selects the corresponding FIFO by means of the 2-most significant bits of the logical registers coded in the instruction word (R_s, R_d and R_t). This simple logic has to be replicated for both source operands and for the destination registers, allowing in this way to turn the registers on for the read (decode stage) and write (write-back stage) of the operands. Also, those split FIFOs are designed with an extra port (4-port FIFOs) to allow concurrent access to the three possible operands of the executed instruction and the write back access of the former instruction in the pipeline.

In this way, for every single destination register, the set of permitted rename buffers are known in advance, allowing to keep the other registers off and this set on. The best execution case is when all the operands coded in the instruction word belong to the same FIFO (consecutive assignment) and the energy savings are the highest expected. The worst execution case happens when every operand coded in the instruction word belongs to a different FIFO (spread assignment) and the expected savings are the lowest achievable. An intermediate behavior is expected during normal execution.

4.2 Pipeline sequence

During the fetch stage the instruction is loaded from I-cache into the instruction register. Our approach takes advantage of this fact by forwarding the operand fields of the instruction to the FIFO selection logic. This is possible due to the fixed instruction format used by RISC processors shown in Figure 5. The register designators are in a fixed position and can be easily extracted for the current instruction.

Figure 6 shows the typical pipeline for the out-of-order processor considered in this work. This figure also includes the suitable timing for the main tasks involved in the process, and that is explained in the following paragraphs.

Fetch stage. The three operand labels coded in the instruction word are forwarded to the FIFO selection logic. The FIFOs associated to the destination registers are selected and the unselected rename buffers are kept off.

Decode stage. The rename buffers are assigned to the destination registers. If any instruction label was incorrectly considered as an operand (i.e. the instruction requires less than three operands but this case cannot be known before the decode phase happens), this is discarded and the associated rename buffers kept off (unless the former instruction in the pipeline is requiring them).

Issue stage. The rename buffers contended in the remaining selected FIFOs are turned on one cycle before the reading access.

Execute stage. The previously decoded output operand is sent to the FIFO selection logic.

Memory stage. The registers included in the selected FIFO are turned on one cycle before the writing access.

Write-Back stage. The register to be written has been already turned on and, therefore, no other action is required.

Figure 7 represents the general architecture of the processor supporting the proposed power aware register file. The shadowed modules are those modified or included in the presented approach to perform the energy-aware register renaming implementation. As can be seen, the FIFO selection logic has to be replicated to perform this task also during the write-back stage and, therefore, to turn the required register bank before the write access.

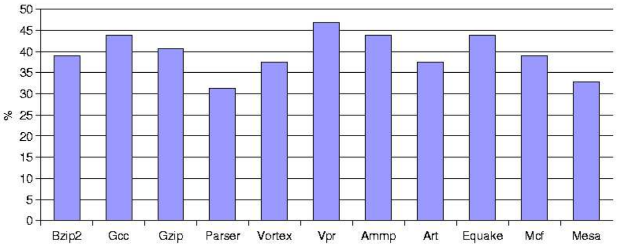

Figure 8: Maximal occupation of 64-position FIFOs

5. EXPERIMENTAL RESULTS

The architecture described in previous paragraphs has been validated with a simulator derived from the SimpleScalar tool suite 4.0 [4]. The baseline architecture has 256 rename buffers and 64 logical registers (32 floating point and 32 fixed point registers). The compiler/architecture reserves four registers for special use. These are the stack pointer, the zero register, the return address register and the return value register. These registers will be left in the active state permanently.

The benchmarks used for the experiments belong to a precompiled set of the SPEC2000 benchmark suite [14]. The benchmarks were run up to completion (about 100 million instructions), skipping the initialization part.

Figure 8 shows the maximal occupation in the 64-position FIFOs of free registers. As can be seen, the maximal occupancy of these FIFOs is lower than 50% for the whole set of benchmarks. Therefore, the 64-position choice can be considered a right design choice because it is not expected to be full-filled. In case these FIFOs were smaller than needed, the processor had to spill some registers to memory and free their assigned rename buffers.

The theoretical maximum for the energy savings would be achieved if all the operands per instruction belong to only one of the FIFOs. Figure 9 shows the occupation for the four FIFOs of free registers. As can be seen, the average occupation is close to the theoretical maximum (only one FIFO used per instruction), and consequently slightly lower energy savings will be obtained due to this difference. The analysis of this figure reflects how most of the instructions require only one FIFO (1.3 on average for the whole running).

Figure 10 shows the energy savings in the register file with the proposed approach. As can be seen, the achieved energy savings are above 60% on average. Differences between benchmarks and with the theoretical maximum[4] correspond to instructions that access spread assigned destination registers. Our new research on compiler technology, optimizing the assignment performed by the compiler, can improve these values.

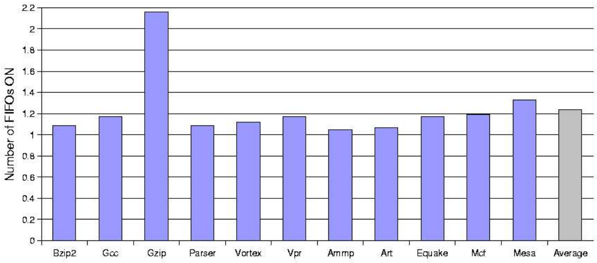

Figure 9: Occupation of the 4 FIFOs

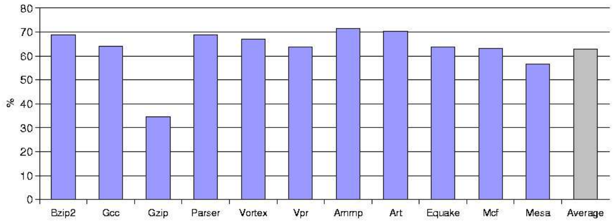

Figure 10: Energy savings in the register file

6. CONCLUSIONS

We have presented a novel scheme that dynamically reconfigures the register file in order to save power consumption in out-of-order processors. Attending to the facts that only three registers are accessed per instruction, and that the compiler usually assigns registers consecutively, we rearrange the register file in banks to keep the unused registers in a low-power state. The proposed approach modifies the allocation of free registers, accomplished during the renaming stage, to turn the drowsy registers on before they are required. In this way, no performance penalty is produced.

The approach has been simulated and validated with benchmarks from the SPEC2000 suite, showing power savings around a 60% of the total power consumption for the register file. Higher energy savings are expected by modifying the register assignment performed by the compiler. This is the hardware part, enabling compiler support. The compiler can and will assign registers to maximize bank reuse. It is currently under development and appears very promissing.

7. REFERENCES

[1] J. Abella and A. González. On reducing register file pressure and energy in multiple-banked register files. In *International Conference on Computer Design*, 2003.

[2] J. Abella and A. González. Power-aware adaptive issue queue and register file. In *International Conference on High Performance Computing*, 2003.

[3] C. J. Anderson, J. Petrovich, J. Keaty, and G. Nusbaum. Physical design of a fourth-generation POWER GHz microprocessor. In *International Solid-State Circuits Conference*, 2001.

[4] All the registers coded in the instruction word can be found in the same FIFO

[4] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *Computer*, 35(2):59–67, February 2002.

[5] J. L. Ayala, M. López-Vallejo, A. Veidenbaum, and C. A. López. Energy aware register file implementation through instruction predecode. In *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 2003.

[6] J. L. Ayala and A. Veidenbaum. Reducing register file energy consumption using compiler support. In *IEEE Workshop on Application Specific Processors*, 2002.

[7] J. L. Cruz, A. González, and M. Valero. Multiple-banked register file architectures. In *International Symposium on Computer Architecture*, 2000.

[8] P. B. et al. Early-stage definition of LPX: A low power issue-execute processor. In *Workshop on Power-Aware Computing Systems*, 2002.

[9] R. P. P. et al. Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading. In *International Solid-State Circuits Conference*, 2002.

[10] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and M. T. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture*, 2002.

[11] H. Hanson, M. Hrishikesh, V. Agarwal, S. W. Keckler, and D. Burger. Static Energy Reduction Techniques for Microprocessor Caches. In *International Conference on Computer Design*, 2001.

[12] N. S. Kim and T. Mudge. Reducing register ports using delayed write-back queues and operand pre-fetch. In *International Conference on Supercomputing*, 2003.

[13] I. Park, M. D. Powell, and T. N. Vijaykumar. Reducing register ports for higher speed and lower energy. In *MICRO*, 2002.

[14] M. Postiff, D. Greene, C. Lefurgy, D. Helder, and T. Mudge. The MIRV SimpleScalar/PISA Compiler. Technical Report CSE-TR-421-00, University of Michigan EECS Department, April 2000.

[15] A. Seznec, E. Toullec, and O. Rochecouste. Register write specialization register read specialization: A path to complexity-effective wide-issue superscalar processors. In *MICRO*, 2002.

[16] P. Shivakumar and N. P. Jouppi. CACTI 3.0: An integrated cache timing, power, and area model. Technical report, Western Research Laboratory, August 2001.

[17] D. Sima, T. Fountain, and P. Kacsuk. *Advanced computer architectures: a design space approach.* Addison-Wesley Longman, 1997.

[18] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy. POWER4 systems microarchitecture. *IBM Journal Research and Development*, 46(1):5–27, January 2002.

[19] J. H. Tseng and K. Asanović. Banked multiported register files for high-frequency superscalar processors. In *International Symposium on Computer Architecture*, 2003.

[20] V. V. Zyuban and P. M. Kogge. The energy complexity of register files. In *International Symposium on Low Power Electronics and Design*, 1998.

[21] V. V. Zyuban and P. M. Kogge. Inherently lower-power highe-performance superscalar architectures. *IEEE Transactions on Computers*, 50(3):268–285, March 2001.