A hardware mechanism to reduce the energy consumption of the register file of in-order architectures

José L. Ayala , Marisa López-Vallejo and Carlos A. López-Barrio

Department of Electronic Engineering,
Universidad Politécnica de Madrid, Spain

Alexander Veidenbaum

Center for Embedded Computer Systems,
University of California in Irvine, USA

Abstract: This paper introduces an efficient hardware approach to reduce the register file energy consumption by turning unused registers into a low power state. Bypassing the register fields of the fetch instruction to the decode stage allows the identification of registers required by the current instruction (instruction predecode) and allows the control logic to turn them back on. They are put into the low-power state after the instruction use. This technique achieves an 85% energy reduction with no performance penalty.

Keywords: register file; in-order; power reduction; predecode; hardware approach.

Reference to this paper should be made as follows: Ayala, J.L., López-Vallejo, M., López-Barrio, C.A. and Veidenbaum, A. (2008) 'A hardware mechanism to reduce the energy consumption of the register file of in-order architectures', *Int. J. Embeddded Systems*, Vol.

Biographical notes: José L. Ayala received the MS Degree in Telecommunication (Electrical) Engineering in 2001 from the Technical University of Madrid and the PhD Degree in Electrical Engineering in 2005 at the same university. His research interests include power estimation and power optimisation in processor-based systems, with particular emphasis on high-performance processors and multi-processor systems-on-chip.

Marisa López-Vallejo received the MS and PhD Degrees in Telecommunication (Electrical) Engineering from the Technical University of Madrid in 1992 and 1999, respectively. She joined the Faculty of the Technical University of Madrid in 1993 and is currently an Associate Professor in the Department of Electronic Engineering. Her research interests include VLSI design for communication systems and CAD tools for hardware-software codesign, with particular emphasis on power optimisation for embedded systems.

Carlos A. López-Barrio is a Professor of Electronics Technology at the Technical University of Madrid and Director of the Integrated Systems Laboratory (Department of Electronics Engineering). His research interests include VLSI design and test methodologies, EDA tools for the design and test of ASICs and systems, and digital architectures. He has been also Chairman of the Strategic Partners Committee of ES2 (European Silicon Structures), member of the Board of Directors of this company (1992–1995) and member of the Board of Directors of Venturini España (1997–2004).

Alexander Veidenbaum received a PhD from the University of Illinois at Urbana-Champaign in 1985. He was an Assistant Professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign from 1985 to 1994. From 1995 to 1998, he was an Associate Professor at the University of Illinois in Chicago and joined the UCI faculty in 1999. His research interests are hardware and systems, including building computer systems, computer architecture, and compiler design. He has made significant contributions in the areas of memory hierarchy and cache design, adaptive system architecture.

1 Introduction

Continuing advances in semiconductor technology have allowed dramatic performance gains for general-purpose microprocessors and embedded systems. These improvements are due both to increasing clock rates as well as to advanced support for exploiting instruction-level parallelism and memory locality using the additional transistors available in each process generation. However, as a negative consequence, this causes a significant increase in power dissipation, due to the fact that dynamic power is proportional to both clock frequency and to switching capacitance (which increases as more devices and on-chip components are included). Thus, despite continuous attempts to reduce voltages and to design lower power circuits, power dissipation levels have steadily increased with each new microprocessor generation (Gunther et al., 2001). Moreover, a new problem arises because the power savings achievable with low level techniques are reaching their theoretical maximum.

Improvements in integrated circuit fabrication technology have enabled to increase the number of transistors of every generation of microprocessors to more than double. At the same time, leakage current also increases with each technology generation. Thus, the energy consumption of memory structures (main memory unit, caches, register banks, etc.) will increase dramatically with future process technologies.

The register file consumes a sizable fraction of the total power in embedded processors and becomes a dominant source of energy dissipation when other power saving mechanisms have been applied. Register file power consumption in embedded systems depends very much on system configuration, mainly on the number of integrated registers, cache size and existence of a branch predictor table (i.e., depends on the relative size of other memory devices). In the Motorola's M.CORE architecture, the register file energy consumption could achieve 16% of the total processor power and 42% of the data path power (Gonzales, 1999). In out-of-order processors with a large number of physical registers which are implemented as part of the *Re-Order Buffer* (in Pentium III, for example), this structure dissipates as much as 27% of total energy according to Folegnani and González (2001).

Many recent compiler optimisation techniques increase the register pressure, and there is a current trend towards implementing larger register files. Large register files present several advantages: decreased power consumption in memory hierarchy (cache and main memory) by eliminating accesses, improved performance by decreasing path length and memory traffic by removing load and store operations. Previous works showed that the existing compiler technology could not make effective use of a large number of registers (Mahlke et al., 1992), and the industry followed this statement implementing most processors with 32 or fewer registers. However, current research in this area (Postiff et al., 2000) and the efforts on optimising spill code indicate the need of more registers. Sophisticated optimisations can increase the number of variables and the register pressure (Llosa, 1996); furthermore, global variable register allocation can increase the number of required registers. The number of registers in recent architectures has grown considerably. The number of registers, the register pressure and the aggressive compiler optimisations (by allocating global variables, promoting aliased variables and inlining) lead to increase the energy consumption.

Table 1 shows the register file size for several modern processors clearly showing a significant increment in size. This motivates the need of efficient techniques to reduce register file energy consumption in embedded systems. This paper introduces a new technique which is characterised by a lack of performance penalty. It is based on the observation that a register is only used when an instruction reads from it or writes to it, the register is 'idle' at all other times. By keeping the idle registers in a low power (or 'drowsy') state a significant amount of energy can be saved. Most registers are idle at any given cycle given that no more than three registers are accessed by an issued instruction. The architectural modifications we propose in this paper allow the drowsy registers to be turned back to the 'active' state when the instruction accesses them.

Table 1 Register file size

Processor	*Integer RF*	*Floating RF*
ARM	32	8
Power PC	32	32
Xscale	32	32
Transmeta crusoe	64	64

The rest of this paper is organised as follows. Next section summarises previous research on this and other related topics. Section 3 is devoted to the explanation of the proposed approach and introduces the modified register file and the energy saving technique. Section 4 presents the experimental setup and shows the resulting energy savings, including a comparison of the approach with a compiler-based technique. Finally, some conclusions are drawn in Section 5.

2 Related work

Low power techniques can be applied at different abstraction levels: from technology and circuit levels to system and algorithm levels. Low level power optimisation techniques are achieving their theoretical maximum, thus making necessary to look into approaches at higher abstraction levels. Energy optimisation and estimation at the system level is an area of very active research. At the system level, the sources of energy dissipation can be grouped under three broad categories:

- processing units
- memories
- interconnects and communication (Benini and Micheli, 2000).

Different techniques address more than one category, while others are more narrowly focused. The register file can be considered as a component of the memory hierarchy and, therefore, energy reduction techniques for memories can be applied with some modifications.

Several approaches focus on creatively exploiting caching to reduce energy consumption. They use a new structure (often called 'cache buffer' or L0 cache) (Kin et al., 1997) or apply compilation and data type optimisations to reduce memory accesses (Takamura et al., 2002). Alternatively, architectural modifications to the memory cell and both address decoders and bitline drivers have been previously proposed by some authors (Tseng and Asanovic, 2000). All of these approaches try to exploit simple data locality, which may not be enough for appreciable energy reduction. Moreover, the extra hardware required is not justified by the energy savings obtained.

Other techniques target more general memory and register file hierarchy, where caches as well as various types of memories (SRAMs, DRAMs) are available. In these architectures, data transfer and placement are tightly controlled to minimise power consumption per memory access (Inoue, 2001). Also, hardware reconfiguration policies, based on power aware compiler support, have been studied in Ayala et al. (2003a). These approaches do not always obtain high energy savings due to the coarse granularity level at which the compiler has to work and, even worse, may suffer a performance penalty.

Solutions based on code versioning and selection by the compiler which use heuristics and profile data (Azevedo et al., 2001) have also been proposed. Finally, there are several software approaches based on code profiling and code annotation (Azevedo et al., 2002), operating system management (Savransky et al., 2002), and circuit level optimisations on the register file (Zalamea et al., 2004). None of these techniques have been able to obtain appreciable energy savings in the register file without a time penalty or an increased cache pressure.

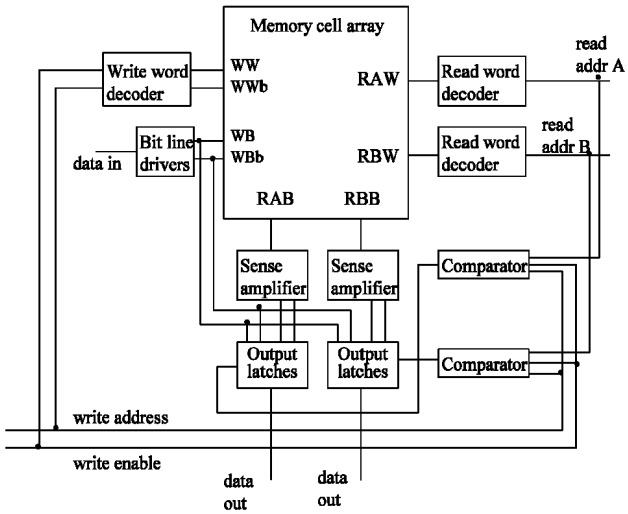
Finally, some works using a banked implementation of the register file have been recently proposed for high-performance out-of-order processors (Ayala et al., 2003b; Ayala et al., 2004).

3 A new approach

3.1 Register file design overview

In a typical configuration, the register file is an array of N words by M bits. Any of the N words can be simultaneously accessed by two read ports and a single write port. The block diagram of the register file that appears in Figure 1 shows that the register file contains seven distinct types of functional blocks (Steidl, 2001). These are the memory cell array, the read address decoders and word line drivers, the write address decoder, the bit line drivers, the sense amplifiers, the output latches and the comparators.

Figure 1 Register file block diagram



The memory cell array stores the bits of data. It is arranged in a grid of N rows by M columns of memory cells. When any of the ports accesses the memory cell array, the read or write operation is performed simultaneously on every memory cell in the selected row. Each read address decoder is responsible for decoding a log_2N-bit address to determine which row is selected in every read operation. The read word line drivers are responsible for driving the read word lines accordingly. The write address decoder selects the row to be written. The write word line drivers drive the write word lines while the bit line drivers provide input data to the memory cells. The independent read and write address decoders allow parallel access to up to three register operands of an instruction. The rest of the register file components and operation are not described in this paper due to space limitations. Readers interested in a deeper knowledge on the register file structure are referred to Sima et al. (1997).

3.2 Background on energy reduction

As was explained before, current processors have a large register file. Nevertheless, this device is underutilised most of the time. Embedded processors are rapidly becoming wide issue and speculative. 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.

The $2i + i$ registers have to be on to provide the source operands and store the results at a given cycle. But the remaining 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, if possible, should be turned into a low-power state.

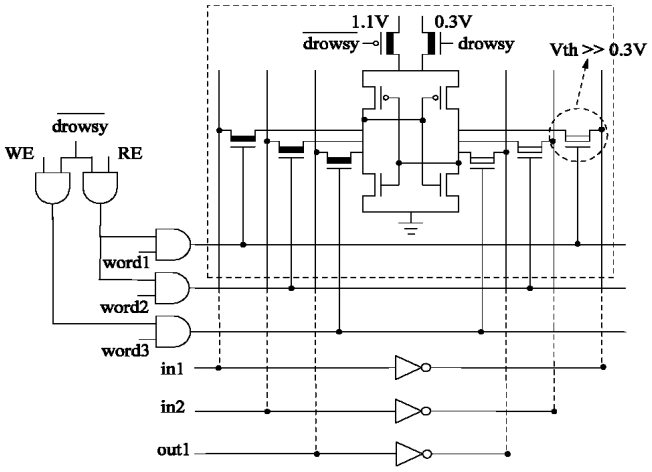
Turning memory devices into a low-power state is not a brand new idea. Previous research has focused on turning off unused memory banks (or other resources) by gating the power source. When the objective is a memory device, the cost of recovering the lost information could hide any power saving or, at least, represent a very significant time penalty.[1] 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. Turning these unused registers into a low-power state (*drowsy* state) the power consumption can be reduced to a minimum without data lost (Hanson et al., 2001).

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 circuit technique for implementing drowsy memory devices is adaptive body-biasing with multi-threshold CMOS (ABB-MTCMOS), where the threshold voltage is increased dynamically to yield reduction in leakage energy. This leakage reduction technique requires that the voltages of the N-well and of the power and ground supply rails are changed whenever the circuit enters or exits the drowsy mode. Since the N-well capacitance of the PMOS devices is quite significant, this increases the energy required to switch the memory cell to high-power mode and can also significantly increase the time needed to transition to/from drowsy mode. A more efficient approach to achieve the drowsy state is proposed by Flautner et al. (2002), 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 dissipation.

3.3 System design

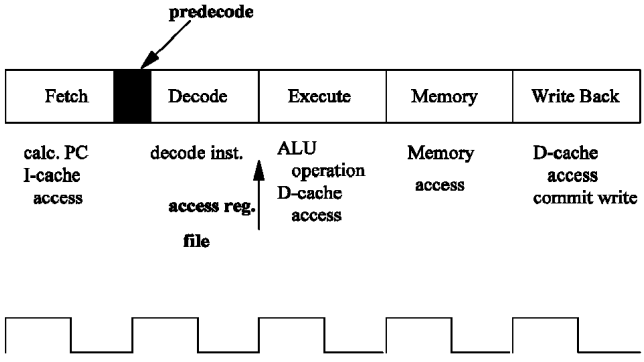
Figure 2 shows the modified register file cell used to support the drowsy state. As can be observed, the dual power supply is switched to low V_{DD} when the cell is in drowsy state. It is necessary to use *high-*V_{th} devices as pass transistors because the voltage on bit lines could destroy the cell contents. Before a register cell can be accessed, the power supply has to be switched to high V_{DD} to restore the 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.

Figure 2 Register file cell



In a typical pipeline organisation (Figure 3) the register read access occurs during the same clock cycle than the instruction decode. Therefore, the register power supply must be switched to high V_{DD} prior to this phase: the fetch cycle (we are assuming no renaming phase in an embedded in-order processor). In some pipelines, e.g., *ARM10*, there is a pre-fetch cycle which allows detecting branch instructions ahead of the fetch stage, predicting those branches that are likely to be taken or remove those branches that are not likely to be taken (ARM, 2001). However, this stage is only used when a new instruction has to be reloaded from memory instead of I-cache, and it may not be available in all architectures.

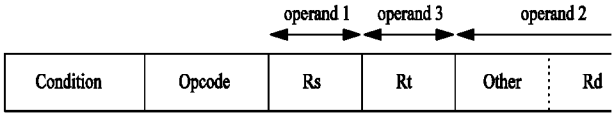
Figure 3 Pipele configuration (see online version for colours)



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 register file address decoders. This is possible due to the fixed instruction format used by RISC processors shown in Figure 4. The register designators are in a fixed position and can be easily extracted for the

current instruction. This intermediate step can be performed in the *issue* stage of the pipeline. If the issue stage is not present in the processor pipeline, the time required to complete this simple decoding can be overlapped with the delay between the availability of the cache data and the completion of the tag comparison.

Figure 4 General instruction format (see online version for colours)



In order to reduce the complexity of the additional logic and to assure the register identification in the shortest time (and before the starting of the next clock cycle), the register pre-decode cannot make use of any memory access to help the decoding. Therefore, in our approach only the address decoders present in the register file architecture are used. The delay of these decoders is small enough to identify the accessed registers before the end of a clock cycle and, therefore, to turn these on before the following access.

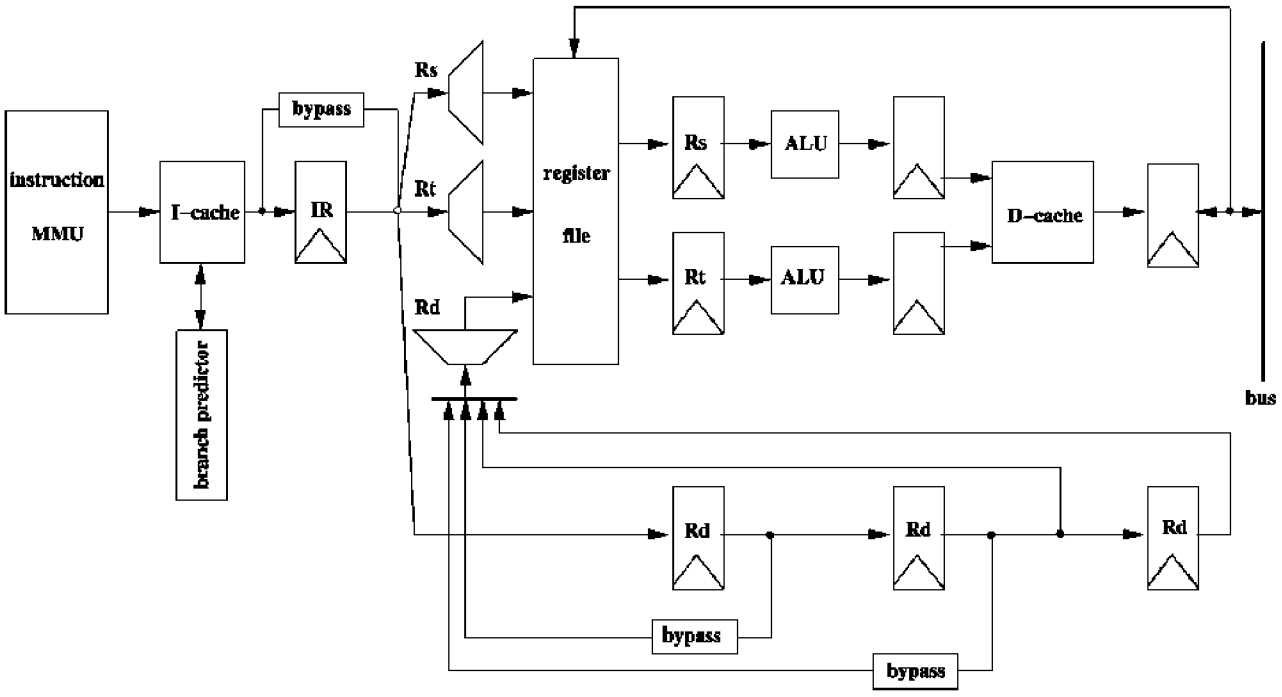
An interesting point to outline is that the current executed instruction may not correspond to the general instruction format presented in Figure 4. There are instructions that may not include the three register fields, or these fields may have a different meaning (memory address or immediate operand), but this is only known when the instruction decode has been accomplished (unless pre-decoded info is stored in the I-cache). Therefore, the pre-decode logic will be fitted to the general instruction format (three operand registers) and, when the instruction shows a different configuration the result will be the awakeness of registers that are not used. Consequently, several registers that could be kept in the low power state are activated and extra power is consumed. This effect will be analysed in detail in Section 4.1, let us say for now that the worst case involves waking up three registers when none is used. The overall energy savings from our approach are still more than significant.

So far we have only discussed register reads. The register write access happens after the *memory* stage, and at this time the write register designator is known in any case. Thus the register to be written can be turned on after the execute or memory stage (depending on the pipeline structure) and written in the write-back stage.

Figure 5 represents the general architecture of a processor supporting the proposed power aware register file. The bypasses required by our technique have been represented shadowed.

As can be noticed, the main advantages of this approach are the no-time penalty on the pre-decode stage, the simplicity of the extra logic (only a few modifications to the memory cell, the gating circuit and the bypass network) that means small energy and area overhead and, finally, the possibility of applying this approach to different microprocessor architectures present in many embedded systems. The downside of the approach is that the extra dynamic consumption (wake-up cost) has to be amortised over a small number of accesses. Flautner et al. (2002) have estimated this value to be around seven accesses. Taking into account the number of drowsy registers per instruction in the register file (all of them but the three accessed and the control registers), this cost has not a big impact on the power savings of our approach.

Figure 5 Schematic of the general architecture



4 Experimental results

The architecture described in the previous paragraphs has been validated with a simulator derived from the SimpleScalar tool suite (Austin et al., 2002). SimpleScalar is an execution-driven simulator that implements a derivative of the MIPS-IV instruction set. The version used in our experiments is 3.0 c. The compiler/architecture reserves four registers for special use. They are the stack pointer, the zero register, the return address register and the return value register. These registers are going to be left in the active state permanently.

The baseline system models a typical architecture of a high-performance embedded system. It has been modified to support the power aware register file. Table 2 presents the main characteristics of the baseline embedded architecture.

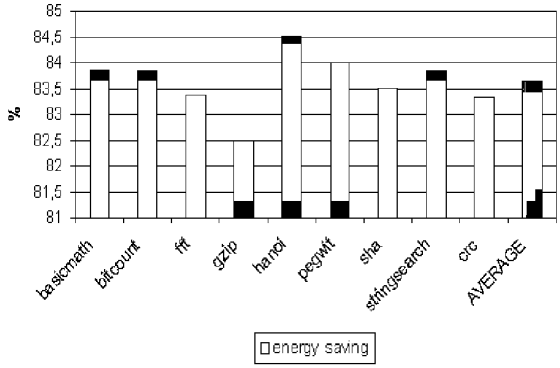
Table 2 Baseline processor configuration

Baseline architecture	
Pipeline	in-order
Clock	600 MHz
Data width	32 bits
Scheduling	in-order
Decode width	2 insts/cycle
Issue width	2 insts/cycle
Commit width	2 insts/cycle
Fetch width	2 insts/cycle
Functional units	1 integer ALU
	1 integer multiply/divide unit
	1 FP ALU
	1 FP multiply/divide unit
Register file	32 + 32 registers

Figure 6 shows the energy savings obtained in the register file of the modified architecture running programs from SPEC2000, MediaBench, and MiBench benchmark suites. All results are normalised with respect to the register file power dissipation in the baseline architecture. In these experiments it has been assumed that the energy consumption of a drowsy register is negligible. This assumption is based on the results published by Flautner et al. (2002) who show a 2% of energy consumption during the drowsy state for a cache memory cell. Also, area and energy overhead due to the extra logic (gating circuit) is completely negligible and has not been considered in the results.

As can be seen, the register file power consumption is reduced by nearly 84%, on average. The results are very similar for all benchmarks because the number of active registers per clock cycle is always the four reserved registers and up to three register operands. The slight differences between benchmarks are due to the variations in the number of register file accesses and instruction type distribution.

Figure 6 Register file energy savings (see online version for colours)

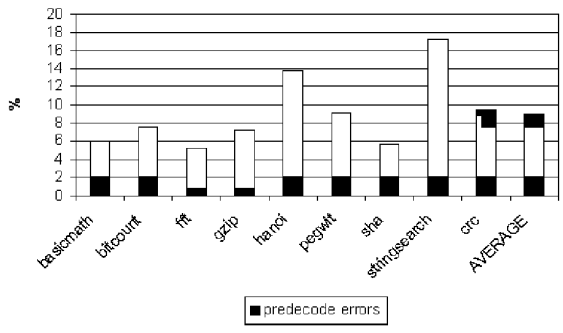


It should be noted that for out-of-order systems which have a large number of physical registers (80 in the case of Alpha 21264) even higher total energy savings are expected. In addition, the register file may be implemented as part of the Reorder Buffer (ROB) in this case the energy savings are even higher. In this case, turning physical registers into the drowsy state also allows the rest of the ROB to be more energy efficient by turning off unused memory cells and ports (Kucuk et al., 2002).

4.1 Accuracy of the approach

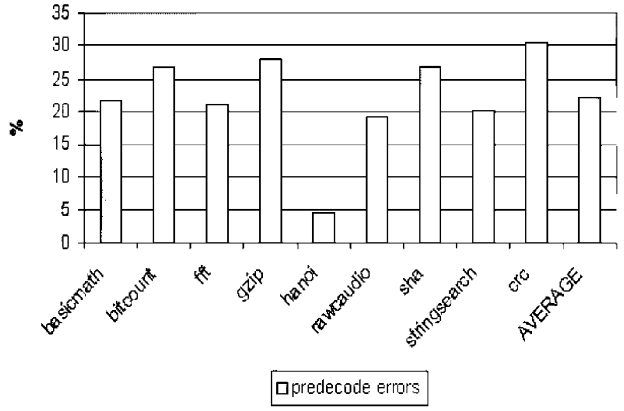
As was explained in Section 3.3, the pre-decode logic assumes that a three register operand instruction is always executed, what could not be true. In order to analyse how far our approximation is to the real execution, the following analysis has been performed. If no register operand is required by any instruction, the register file power savings could go up to 60/64 = 94% (four awake reserved registers per instruction). This is the theoretical maximum we have on power saving. Figure 7 shows the number of erroneous register awakeness normalised to the total number of required registers during program execution. A 100% value in this graph would mean that every instruction requires no register operand. However, on average only a 9% of registers are erroneously awaken, what means that the maximum power savings expected in the register file would be less than 88% on average, only 1% higher[2] than our approximation.

Figure 7 Erroneous register awakeness (MIPS-IV) (see online version for colours)



This behaviour will vary slightly depending on the processor and the corresponding *Instruction Set Architecture* (ISA), but it is expected to be similar for other machine configurations. Figure 8 shows comparable results for an Alpha 21264 processor. Even for the extended ISA of this processor our approach achieves results that are quite near to the MIPS-IV case. Now, collected stats show how a 20% of the awake registers could have been avoided, keeping these registers into the drowsy state and saving a little more extra energy than our approach (2% for 32 integer and 32 floating-point registers).

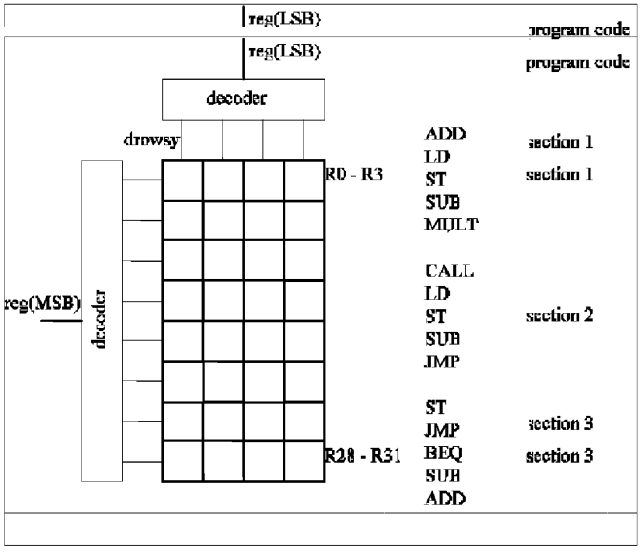
Figure 8 Erroneous register awakeness (Alpha 21264) (see online version for colours)



4.2 Compiler-based approach

The previously presented approach has been compared with other technique based on a power-aware compiler and where no architectural modifications are needed. This technique works as follows: those registers unused by a predefined section of code are turned into the drowsy state just before the section is reached. When code using the turned off registers is reached (the predefined section ends) they have to be turned back on (Figure 9).

Figure 9 Activation of the drowsy state (see online version for colours)



This register file management technique requires a code exploration and code generation phase that marks the code sections where the register file reconfiguration should be performed. This phase has to detect those frequently executed code sections with reduced number of required registers and mark the beginning and end of such portion of code. Readers interested on details about this approach are referred to the published material by Ayala et al. (2003).

4.2.1 Description of the modified compiler

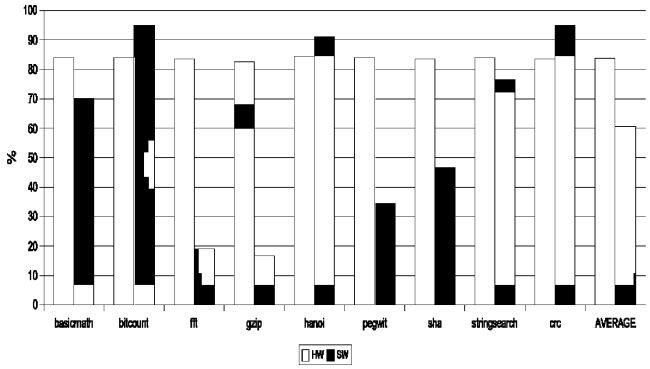
The register file management policy that has been outlined above requires a single ISA modification. It needs a signalling mechanism to mark the beginning and the end of a code section. This signaling is performed by incorporating a new instruction to the ISA that determines when the registers have to be turned on and off. After register allocation, the compiler knows exactly which registers are used by all instructions, and can specify the register requirements per function (or a loop inside the function if a lower granularity is used) in a new instruction that turns the unused registers into the drowsy state. After this portion of code, the same instruction turns on the drowsy registers.

Profiling information can be used to improve the selection of the most time and power consuming function.

4.2.2 Experimental results

Figure 10 shows the comparison of the energy savings when the software (compiler) and hardware (pre-decode) approaches are used.

Figure 10 Comparison of the energy savings (see online version for colours)



As can be noticed the average energy savings obtained with the hardware approach are higher than the ones obtained with the compilation approach (83.7% compared to 60.5%). Moreover, there are some cases where the compilation approach presents a much worse behaviour than the pre-decode technique (i.e., fft, gzip, pegwit and sha). For those benchmarks, the granularity used for the power-aware compiler (the two most time consuming functions and the most time consuming loop inside these functions) is not able to obtain appreciable energy savings. The hardware approach is not limited to specific portions of code and can save energy during the whole run of the benchmark.

There are also some cases where the power-aware compiler is able to save a little more energy than the pre-decode technique (i.e., bitcount, hanoi and crc). Those benchmarks present little register usage and one or two most time consuming functions. Therefore, the compiler is able to turn into the low power state most registers during the execution.

On the other hand, the power reduction technique based on the power-aware compiler presents some benefits like the simplicity of the approach where no hardware modifications are needed, the lack of performance penalties, and the automatic extension to different target architectures, but the behaviour is quite irregular with much more energy savings for some benchmarks. Our current research work tries to increase the global energy savings by combining the large scope of the hardware mechanism with the local savings obtained by the compilation technique.

5 Conclusions

The register file is a power hungry device in modern embedded processors. At the same time, current research on compiler technology and computer architecture encourages the implementation of an increasing number of registers. Thus, the register file power consumption becomes a significant factor. The number of registers actually used per instruction is very small: less than or equal to three. This fact is exploited by the technique proposed in this paper to save power by turning the unused registers into a low-power state. They are then awakened before being accessed by an instruction.

An efficient hardware mechanism to turn the unused registers into a low power state has been described in this paper. This is performed at the micro-architecture level for each instruction. A DVS technique is used to keep the information stored in the registers while reducing the power consumption to a minimum. The required extra logic has been simplified to reduce the register specified pre-decode time.

The proposed technique reduces the register file power consumption by 84%, on average, when running benchmarks on embedded processors. Assuming all instructions use three register operands results in 9% of extra register wake-ups over the theoretical minimum. Therefore, the optimal case where only the required registers would be awakened would only save an additional 1% over the proposed approach. The presented technique has no performance penalty and the area/energy overhead is negligible.

This technique has been compared with an approach based on a power-aware compiler, which reaches a 65% of energy savings in the register file without any performance overhead.

The simplicity of the approaches and the minimal additional logic required make them an effective low-power technique for embedded processors. Future work will combine both approaches to increase the expected energy savings.

References

ARM (2001) ARM10022E: Technical Reference Manual.

Austin, T., Larson, E. and Ernst, D. (2002) 'SimpleScalar: an infrastructure for computer system modeling', *Computer*, Vol. 35, No. 2, February, pp.59–67.

Ayala, J.L. and López-Vallejo, M. (2004) 'Low-power register assignment for out-of-order Architectures', *IEEE Workshop on Innovative Architectures*, USA, pp.42–49.

Ayala, J.L., López-Vallejo, M. and Veidenbaum, A. (2003b) 'Energy-efficient register renaming in high-performance processors', *IEEE Workshop on Application Specific Processors*, USA, pp.321–326.

Ayala, J.L., Veidenbaum, A. and López-Vallejo, M. (2003a) 'Power-aware compilation for register file energy reduction', *International Journal of Parallel Programming* Kluwer Academic Publishers, Vol. 31, No. 6, December, pp.449–465.

Azevedo, A., Issenin, I., Cornea, R., Gupta, R., Dutt, N., Veidenbaum, A. and Nicolau, A. (2001) 'Architectural and compiler strategies for dynamic power management in the COPPER project', *International Workshop on Innovative Architecture*, USA, pp.35–40.

Azevedo, A., Issenin, I., Cornea, R., Gupta, R., Dutt, N., Veidenbaum, A. and Nicolau, A. (2002) 'Profile-based dynamic voltage scheduling using program checkpoints in the COPPER framework', *Design and Test in Europe Conference*, France, pp.168–175.

Benini, L. and Micheli, G.D. (2000) 'System-level power optimization: techniques and tools', *Design and Test in Europe*.

Flautner, K., Kim, N.S., Martin, S., Blaauw, D.D. and Mudge, T. (2002) 'Drowsy caches: simple techniques for reducing leakage power', *International Symposium on Computer Architecture*, USA, pp.148–157.

Folegnani, D. and González, A. (2001) 'Energy-effective issue logic', *International Symposium on Computer Architecture*, Sweden, pp.230–239.

Gonzales, D.R. (1999) 'Micro-RISC architecture for the wireless market', *IEEE Micro*, Vol. 19, No. 4, July–August, pp.30–37.

Gunther, S.H., Binns, F., Carmean, D.M. and Hall, J.C. (2001) 'Managing the impact of increasing microprocessor power consumption', *Intel Technology Journal*, Vol. 5, No. 1, pp.1–9.

Hanson, H., Hrishikesh, M., Agarwal, V., Keckler, S.W. and Burger, D. (2001) 'Static energy reduction techniques for microprocessor caches', *International Conference on Computer Design*, USA, pp.276–281.

Inoue, K. (2001) *High-Performance Low-Power Cache Memory Architectures*, PhD Dissertation, Kyushu University.

Kin, J., Gupta, M. and Mangione-Smith, W.H. (1997) 'The filter cache: an energy efficient memory structure', *International Symposium on Microarchitecture*, USA, pp.184–193.

Kucuk, G., Ponomarev, D. and Ghose, K. (2002) 'Low-complexity reorder buffer architecture', *International Conference on Supercomputing*, USA, pp.57–66.

Llosa, J. (1996) *Reducing the Impact of Register Pressure on Software Pipelined Loops*, PhD Dissertation, Universidad Politécnica de Cataluña, Spain.

Mahlke, S.A., Chen, W.Y., Chang, P.P. and Hwu, W. (1992) 'Scalar program performance on multiple-instruction issue processors with a limited number of registers', *Hawaii International Conference on System Sciences*, pp.34–44.

Postiff, M., Greene, D. and Mudge, T. (2000) *The Need for Large Register File in Integer Codes*, Electrical Engineering and Computer Science Department, The University of Michigan (USA), Tech. Rep. CSE-TR-434-00.

Savransky, G., Ronen, R. and Gonzalez, A. (2002) 'Lazy retirement: a power aware register management mechanism', *Workshop on Complexity Effective Designs*.

Sima, D., Fountain, T. and Kacsuk, P. (1997) *Advanced Computer Architectures: A Design Space Approach*, Addison-Wesley, UK.

Steidl, S.A. (2001) *A 32-Word by 32-Bit Three-Port Bipolar Register File Implemented Using a SiGe HBT BiCMOS Technology*, PhD Dissertation, Rensselaer Polytechnic Institute, USA.

Takamura, H., Inoue, K. and Moshnyaga, V.G. (2002) 'Reducing power consumption of register files through operand reuse', *SIGNotes Computer ARChitecture*, No. 149, August.

Tseng, J.H. and Asanovic, K. (2000) 'Energy-efficient register access', *Symposium on Integrated Circuits and System Design*, Brazil, pp.377–382.

Zalamea, J., Llosa, J., Ayguadé, E. and Valero, M. (2004) 'Software and hardware techniques to optimize register file utilization in VLIW architectures', *International Journal of Parallel Programming*, Vol. 32, No. 6, December, pp.447–474.