Stall Power Reduction in Pipelined Architecture Processors

Pejman Lotfi-Kamran, Amir-Mohammad Rahmani, Ali-Asghar Salehpour,
Ali Afzali-Kusha, and Zainalabedin Navabi
Nanoelectronics Center of Excellence, School of Electrical and Computer Engineering, University of Tehran
plotfi@computer.org, am.rahmani@ece.ut.ac.ir, a.salehpour@ece.ut.ac.ir, afzali@ut.ac.ir, and navabi@ece.neu.edu

Abstract

This paper proposes a technique for dynamic power reduction of pipelined processors. Pipelined processors frequently insert NOP instruction to the pipe for generating delay or resolving dependency. Our study shows that the percentage of power consumed by NOP instructions in a pipelined processor is significant. This article studies the detail behavior of NOP instruction and proposes a technique for eliminating unnecessary transitions that are generated during execution of NOP instructions. Initial results demonstrate up to 10% reduction in power consumption for some benchmarks at a cost of negligible performance (almost zero) and area overhead (below 0.1%).

1. Introduction

Computer scientists have always tried to improve the performance of processors. Today's processors are much faster and far more versatile than their predecessors [1]. These chips are still somewhat below the power and power density limits afforded by the package/cooling solution of choice in server markets targeted by such processors. In designing future processors, however, energy efficiency is known to have become one of the primary design constraints [2] [3].

There are many microprocessor applications, typically battery-powered embedded applications, where energy consumption is the most critical design constraint. In these applications, where performance is less of a concern, relatively simple RISC like pipelines are often used [4] [5]. In the current CMOS technology, the most energy consumption occurs when transistor switching or memory access activity takes place [6] [7]. Among the instructions that a pipelined processor executes, the NOP instruction is one that does not contribute to any useful work. Therefore, the power consumed for its execution is wasted.

In the pipelined architectures, the NOP instruction is inserted for hazard elimination in addition to delay generation. There are three types of hazards; structural, data and control. The structural hazard may occur when there are not enough hardware resources for execution of combination of instructions [8] [9].

A data hazard occurs when an instruction needs the result of a prior instruction that is still in the pipeline and there is not enough latency between these instructions. Two instructions are data dependent when the second instruction requires the result of the first one to begin its execution. A technique for preventing data hazard is to use a forwarding unit. The forwarding unit detects dependencies and forwards the required data from the running instruction to the dependent instructions. In some cases, it is impossible to forward the result because it may not be ready. In these situations, using a NOP instruction is inevitable [8] [9].

The last type of hazard is control hazard that occurs when a branch prediction is mistaken or in general, when the system has no mechanism for branch prediction. There are two mechanisms for handling the miss-prediction. The first mechanism is flushing the pipe after the miss-prediction. Generally, flush mechanisms are not cost effective. A better solution is to fill the pipe after the jump instruction with specific number of NOPs [8] [9].

NOP insertion eliminates hazards but also degrades the performance of the processor. Many solutions are presented for stall reduction (e.g., Forwarding [8], Branch Prediction [8], Speculative Execution [8], etc.) but a significant number of stalls still remain.

The aim of this paper is to optimize dynamic power consumption of a pipelined processor by eliminating useless transitions that are generated in the pipeline when a stall happens. This article shows that in pipelined architectures a number of useless transitions

1063-9667/08 $25.00 © 2008 IEEE
DOI 10.1109/VLSI.2008.34

is generated when a NOP passes through pipe stages. We slightly modify the architecture of RISC processors to reduce the useless transitions generated when a stall happens. Our experimental results show that, with a negligible hardware overhead, a dynamic power reduction of up to 10 percent is achievable.

The rest of paper is organized as follows. The next section underlines related works and their properties. Section 3 presents a simple example that illustrates inserted NOP instructions in pipelined architectures contribute to unnecessary transitions. In Section 4 our proposed technique for reducing the unnecessary transitions is presented. Experimental results are presented in Section 5 and conclusions come in the last section.

2. Related Work

Hartstein and Pusak [10] explored the impact of pipeline length on both the power and performance of a microprocessor by theory and by simulation. Their results show that the more important power metric is, the shorter the optimum pipeline length that results.

In another article, authors present a bipartition dual-encoding architecture for low-power pipelined circuits [11]. They exploit the bipartition approach as well as encoding techniques to reduce power dissipation not only of combinational logic blocks but also of the pipeline registers. Based on Shannon expansion, they partition a given circuit into two sub-circuits such that the number of different outputs of both sub-circuits are reduced, and then encode the output of both sub-circuits to minimize the Hamming distance for transitions with a high switching probability.

The main goal of another article in low power design is to introduce a dynamic branch prediction scheme suitable for energy-aware VLIW (Very Long Instruction Word) processors. The proposed technique is based on a compiler hint mechanism to filter the accesses to the branch predictor blocks [12].

In the same way, another group proposes a low complexity and low power Re-Order Buffer (ROB) design [13] that exploits the fact that the bulk of the source operand values is obtained through data forwarding to the issue queue or through direct reads of the committed register values. Their ROB design uses an organization that completely eliminates the read ports needed to read out operand values for instruction issue.

It is widely known that branch prediction has enabled micro-processors to increase instruction level parallelism (ILP) by allowing programs to speculatively execute beyond control boundaries. In a related work, authors present an innovative method for power reduction which, unlike the previous work that sacrificed flexibility for performance, reduces power in high-performance microprocessors without impacting performance [14]. In particular; they introduce a hardware mechanism called pipeline gating to control rampant speculation in the pipeline. They present inexpensive mechanisms for determining when a branch is likely to mispredict, and for stopping wrong-path instructions from entering the pipeline.

There are many works that target power optimization of pipelined processor, but almost all of them neglect the redundant transitions of NOP instructions and their useless power consumption. In this article by studying the behavior of dynamic power consumed when a NOP instruction executes, an efficient mechanism for power reduction of this instruction is proposed.

3. A Simple Scenario

As discussed, NOP instructions are inserted into the pipeline by many pipelined processors to eliminate hazards. In this section, through a simple example, we demonstrate how these inserted NOP instructions contribute to the overall dynamic power of a pipelined processor by generating a number of unnecessary transitions. For the sake of simplicity and clarity, we discuss the scenario in the usual 5 stage MIPS pipeline [8], but the problem can easily be generalized to any pipelined architectures.

In pipelined architectures and in the DECODE stage, each instruction is analyzed and control signals for running that instruction are generated. In the later stages of pipeline, the generated control signals are used to control the flow of data. If the control unit determines that the current instruction depends on the former instructions and the forwarding cannot resolve the dependency, the control unit inserts a NOP instruction by deactivating some critical control signals to be used in the later stages of pipeline including control signals for writing to memory and register file. We demonstrate through a simple example, the inserted NOP contributes to unnecessary transitions.

```
LOAD $1, 100($2)
ADD  $3, S1, $3
```

Figure 1. A simple program

A simple program is shown in Figure 1. The first instruction is a load from memory and the second instruction is an ADD instruction that uses the loaded data. Because of the dependency between these two instructions, after load instruction, a NOP instruction should be inserted into the pipeline. During the

execution of the simple program of Figure 1, when the LOAD instruction is in the DECODE stage, the control signals and the required data corresponding to this instruction are generated/extracted. On the rising edge of the clock the generated/extracted control/data are latched into the DE/EXE pipeline register. In the next clock cycle, the ADD instruction is in the DECODE stage and the control unit determines that a NOP instruction should be inserted into the pipeline. Therefore, critical control signals are deactivated and these deactivated control signals along with the other control signals and the required data of the ADD instruction (current instruction in the DECODE stage) are latched on the rising edge of the clock. Generally, the data parts of the current and previous instructions are different. It means that data part of NOP is different form the former instruction (i.e., LOAD). Therefore, passing the NOP instruction in the pipe generates a number of transitions. In the third clock cycle, the ADD instruction should be passed to the pipeline. Therefore, control signals corresponding to ADD are generated and are getting latched along with its required data. The expectation is that the data and some control signals of NOP and ADD are the same, so the number of transitions of passing ADD in the pipeline stages is negligible, but it is not the case. At least parts of the data of the ADD instruction are different from those of NOP because those data were not available when they were extracted in the DECODE stage (in fact the lack of availability of those data is the reason of NOP insertion). It means that a number of useless transitions is generated because of the change of data part of the ADD instruction relative to that of the NOP instruction. These unnecessary transitions contribute to the overall dynamic power but do not contribute to any useful work. The aim of this paper is to eliminate (minimize) these transitions, thus optimizing the dynamic power consumption of a pipelined processor.

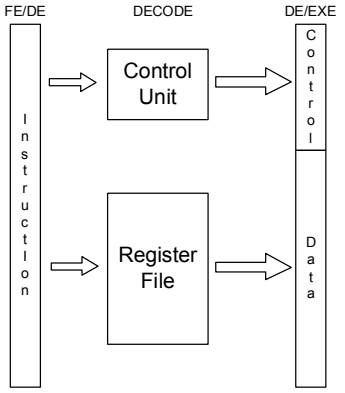

Figure 2. Decode stage of a simple processor

4. Our Proposed Solutions

As discussed, the data part of an inserted NOP instruction is not the same as that of its preceding or subsequent instruction. It means that passing a NOP instruction in the pipe generates a number of transitions. In addition, passing the pending instruction after NOP generates still a number of other transitions. A NOP instruction does not perform any useful work; therefore, the component of dynamic power used for running it is wasted. The technique that is proposed here tries to minimize this component of dynamic power dissipation.

For the NOP instruction to generate as few transitions as possible, its data part should be the same as that of its preceding or subsequent instruction. As discussed, because of the unavailability of certain data of the pending instruction (the instruction passing the pipe after NOP), the data part of the NOP instruction cannot be the same as its subsequent instruction. Therefore the best choice for the power reduction is to use the data part of the instruction preceding NOP as data part of the NOP instruction. In this way, as a NOP instruction passes through a pipe, relative to the previous cycle, the same operations are performed on the same data in all stages of the pipeline; therefore only a small number of transitions is generated as a result of the NOP insertion and propagation.

For this to be implemented, it is sufficient to add a load enable signal to the data and non-critical control parts of DE/EXE pipe register (i.e., only critical control signals [e.g., write to memory and register file signals] that should be loaded in each clock cycle are not controlled by the added load enable). When a NOP is decided to be inserted into the pipe, the controller should deactivate the load enable signal. This way, the content of data and non-critical control part of the inserted NOP instruction are not changed relative to those of its preceding instruction.

4.1. Propagation Boundary Limitation

The technique proposed in the previous section decreases the number of unnecessary transitions generated when a NOP is inserted into the pipe. When data part of the instruction preceding NOP is valid when it is extracted in the DECODE stage, the proposed technique guarantees no useless transitions is generated as a NOP instruction passes the pipe. However, if parts of the data of the instruction preceding NOP are not valid when they are extracted in the DECODE stage, for the correct execution, valid data are prepared by the forwarding unit. In order to minimize the number of transitions generated during execution of NOP, the same data should be prepared

for the NOP instruction. If valid data of the instruction preceding NOP are still in some pipe registers when the NOP instruction needs them, the forwarding unit prepares the data for the NOP as well. In this case, a few number of transitions is generated during execution of the NOP instruction. On the other hand, if the valid data are not available in any pipe register when the NOP instruction needs them (because the instruction that generates those data is finished and goes out of the pipe), different data are loaded into some operators, therefore a number of useless transitions is generated. This causes the inputs of some other operators to change and this cycle continues until transitions reach to the last stage of the pipeline. The technique that we propose here limits the propagation boundary of transitions to a single stage, i.e., the transitions do not propagate to all stages of the pipeline.

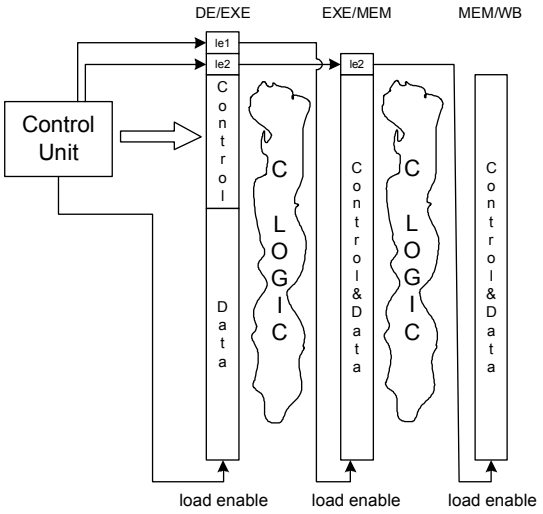

Figure 3. Load enable propagation in a pipeline

The ultimate goal is that the NOP instruction produces the same results as those of its preceding instruction in all pipe stages. In this condition, the value that is loaded in each pipe register when NOP and its preceding instruction execute are the same except for a few critical control signals (e.g., write to memory or register file). Therefore, loading to pipe registers can almost be deactivated during execution of NOP instructions. For this purpose, a load enable is added to each pipe register. This control signal is only applied to data and non-critical control parts of the pipe registers. When this signal is activated, the pipe register performs its usual operation. When this signal is deactivated, only critical control signals are loaded into the pipe register and the value of data and non-critical control signals do not change. By deactivating a pipe register's load enable when NOP results are written to it, only critical control signals of that pipe register are changed and its other parts remain unchanged. If data of a NOP instruction are not valid (i.e., NOP data defer from those of instruction preceding NOP), in some pipe stages a number of transitions is generated. These transitions are propagated until they reach a pipe register. They do not propagate any further.

These load enables are generated by the controller in the DECODE stage and are propagated through pipe registers like other control signals to the desired destination (i.e., specific pipe register). Figure 3 illustrates the mechanism of propagating load enable control signals in the pipe registers.

5. Experimental Results

In this section, we analyze and report the power reduction, area overhead, and timing penalty of our proposed power reduction technique. The described techniques have been implemented in three general processors: MIPS [8], DLX [8], and PAYEH [9].

MIPS is a 5 stage pipelined processor and its architecture is RISC with fixed-width 32-bit instructions.

DLX is a text book example of a RISC processor with a 5 stage pipeline using forwarding to avoid data hazards.

PAYEH is a pipelined version of SAYEH [15] with a similar instruction set and has five pipe stages. SAYEH is a multi-cycle RISC processor with 16-bit data and 16-bit address buses. PAYEH architecture uses a forwarding unit. This forwarding unit can resolve all dependencies by forwarding the required data from the next pipe stages to the previous ones.

In the first series of experiments, we evaluated the effectiveness of our proposed technique when an ASIC is targeted. The modified and original MIPS, DLX, and PAYEH processors are synthesized using the CUB library and the area usage and the maximum clock cycle of each of them are extracted. Table 1 compares the area usage and maximum clock cycle of the original and modified processors. As Table 1 indicates, the performance penalty of the modified processors is approximately 0%. Also the area overhead of the proposed approach is about 0.1% of the original processors.

Four benchmark programs are used to evaluate the effectiveness of our proposed technique. The Factorial benchmark reads a number and calculates its factorial. Fibonacci reads a number and calculates the Fibonacci series. Power reads two numbers, a and b, and calculates a to power b (i.e., a^b), and Vector Addition reads two vectors and calculates their addition element by element.

Table 1. Area and frequency characteristics of original and modified processors

Processor	Area Characteristic			Frequency Characteristic		
	Original Area (mil^2)	Modified Area (mil^2)	Overhead (%)	Original Frequency (MHz)	Modified Frequency (MHz)	Overhead (%)
MIPS	13457	13470	0.097	21.2	21.2	≈ 0
DLX	8400	8411	0.13	39.5	39.5	≈ 0
PAYEH	5974	5981	0.12	46.7	46.7	≈ 0

Table 2. Dynamic power characteristics of original and modified MIPS processor (ASIC)

Benchmark	Power Characteristic		
	# Transitions (Original)	# Transitions (Modified)	Improvement (%)
Factorial	1381280	1247710	9.67
Fibonacci	1317690	1203841	8.64
Power	1385450	1298028	6.31
Vector Addition	1057050	993309	6.03

Table 3. Dynamic power characteristics of original and modified DLX processor (ASIC)

Benchmark	Power Characteristic		
	# Transitions (Original)	# Transitions (Modified)	Improvement (%)
Factorial	1763100	1653430	6.22
Fibonacci	1660220	1560440	6.01
Power	1864770	1760530	5.59
Vector Addition	1507060	1432460	4.95

Table 4. Dynamic power characteristics of original and modified PAYEH processor (ASIC)

Benchmark	Power Characteristic		
	# Transitions (Original)	# Transitions (Modified)	Improvement (%)
Factorial	2763960	2519070	8.86
Fibonacci	2565480	2352030	8.32
Power	2663630	2456400	7.78
Vector Addition	2202560	2043540	7.22

These benchmark programs are applied to the original and modified synthesized processors and dynamic power consumption of each processor is estimated by counting the number of transitions that are generated when the benchmark is running. Tables 2, 3, and 4 show the results obtained for MIPS, DLX, and PAYEH respectively.

As Table 2 indicates, for the MIPS processor, a maximum dynamic power reduction of 9.67% is achieved. The average power reduction of the proposed approach is about 7.66% for this processor. Almost the same results are achieved for DLX and PAYEH processors. For the DLX processor, a maximum and average power reduction of 6.22% and 5.69% are achieved. For PAYEH, 8.86%, and 8.04% are the percentage of maximum and average amount of power saving that is achieved by the proposed technique. The results of Table 1 to 4 indicate that with a 0.13% area overhead, an average dynamic power reduction of 5 to 8 percent is realizable.

In the second series of experiments, we evaluated the effectiveness of our proposed technique when the synthesis target is an FPGA. The modified and original MIPS, DLX, and PAYEH are synthesized. Four benchmark programs are applied to the original and modified synthesized processors and dynamic power consumption of each processor for execution of the benchmarks are estimated. Tables 5, 6, and 7 show the results obtained for MIPS, DLX, and PAYEH respectively.

Table 5. Dynamic power characteristics of original and modified MIPS processor (FPGA)

Benchmark	Power Characteristic		
	Power (Original)	Power (Modified)	Improvement (%)
Factorial	191.06 mw	176.07 mw	7.85
Fibonacci	182.26 mw	169.51 mw	7
Power	191.63 mw	181.56 mw	5.25
Vector Addition	146.21 mw	138.89 mw	5

As Table 5 indicates, for the MIPS processor, a maximum dynamic power reduction of 7.85% is achieved. The average power reduction of proposed approach is about 6.28% for this processor. For DLX, a

maximum and average power reduction of 5.5%, and 5.23% is achieved. For the PAYEH processor, 8.7% and 7.82% are the percentage of maximum and average amount of power savings respectively. The results of Table 5 to 7 indicate that an average power reduction of 5 to 8 percent is realizable by using the proposed approach.

Table 6. Dynamic power characteristics of original and modified DLX processor (FPGA)

Benchmark	Power Characteristic		
	Power (Original)	Power (Modified)	Improvement (%)
Factorial	243.87 mw	230.45 mw	5.5
Fibonacci	229.51 mw	217.34 mw	5.3
Power	256.43 mw	243.27 mw	5.13
Vector Addition	207.44 mw	197.13 mw	4.97

Table 7. Dynamic power characteristics of original and modified PAYEH processor (FPGA)

Benchmark	Power Characteristic		
	Power (Original)	Power (Modified)	Improvement (%)
Factorial	350.4 mw	319.9 mw	8.7
Fibonacci	343.2 mw	315.3 mw	8.11
Power	344.5 mw	319.5 mw	7.24
Vector Addition	305.2 mw	283.1 mw	7.23

6. Conclusions

In this paper, a technique is proposed for eliminating unnecessary transitions that are generated when a NOP instruction is inserted into the pipe of a pipeline processor. The proposed technique is applicable to many pipelined architectures. While hardware overhead and timing penalty of the proposed approach is negligible, the dynamic power reduction of up to 10% on some pipelined processor and benchmark programs is achieved.

7. References

[1] Vasanth, Venkatachalam, and M Franz, "Power Reduction Techniques for Microprocessor Systems", ACM Computing Surveys, Vol. 37, No. 3, September 2005, pp. 195–237.

[2] D. Brooks et al., "Power-aware Microarchitecture: Design and Modeling Challenges for the next-generation microprocessors", *IEEE Micro*, Nov./Dec. 2000, pp. 26-44.

[3] M. J. Flynn, P. Hung, and K. Rudd, "Deep-Submicron Microprocessor Design Issues", *IEEE Micro*, July/Aug. 1999, pp. 11-22.

[5] J. Montanaro and et al. "A 160-MHz, 32-b, 0.5 W CMOS RISC Microprocessor", Digital Tech. J'rnal, Vol. 9, Dec. 1997, pp. 49 - 62.

[5] "PowerPC 405CR User Manual", IBM/Motorola, 6/2000.

[6] G. Cai and C. H. Lim, "Architectural Level Power/ Performance Optimization and Dynamic Power Estimation", Cool Chips tutorial in conjunction with MICRO 32, November 1999, Vol. 17, Issue 11, pp. 1061-1079.

[7] Ramon Canal, Antonio González and James E. Smith, "Very Low Power Pipelines using Significance Compression", 33rd Annual IEEE/ACM International Symposium on Microarchitecture, 2000, pp. 181-190.

[8] D. A. Patterson, and J. L. Hennessy, 2003. *Computer Architecture: A Quantitative Approach, 3rd Edition*. Morgan-Kaufmann, San Francisco, CA.

[9] S. Shamshiri, H. Esmaeilzadeh, and Z. Navabi, "Instruction-Level Test Methodology for CPU Core Self-Testing", ACM Transactions on Design Automation of Electronic Systems, Vol. 10, No. 4, October 2005, pp. 673–689.

[10] A. Hartstein, and T. R. Puzak, "The Optimum Pipeline Depth Considering Both Power and Performance", ACM Transactions on Architecture and Code Optimization, Vol. 1, No. 4, December 2004, pp. 369-388.

[11] S.-J. Ruan, K.-L. Tsai, E. Naroska, and F. Lai, "Bipartitioning and Encoding in Low-Power Pipelined Circuits", ACM Transactions on Design Automation of Electronic Systems, Vol. 10, No. 1, January 2005, pp. 24–32.

[12] M. Monchiero, G. Palermo, M. Sami, C. Silvano, V. Zaccaria, R. Zafalon, "Power-Aware Branch Prediction Techniques: A Compiler-Hints Based Approach for VLIW Processors", *GLSVLSI'04,* April 26–28, 2004, Boston, Massachusetts, USA, pp. 440-443.

[13] G. Kucuk, D. Ponomarev, K. Ghose, "Low–Complexity Reorder Buffer Architecture", *ICS'02*, June 22–26, 2002, New York, USA, pp. 57 - 66.

[14] S. Manne, A. Klauser, D. Grunwald, "Pipeline Gating: Speculation Control for Energy Reduction", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, Issue 11, Nov 1998, pp. 1061-1079.

[15] Z. Navabi, 2004, *Digital Design and Implementation with Field Programmable Devices*. Kluwer Academic Publisher.