An Opcode Encoding Method for Low-Power Instruction Fetch

Sunghwan Kim

Department of Computer Science
Seoul National University
Seoul, Korea 151-742
Tel: +82-2-880-5378
Fax: +82-2-871-4912
e-mail: shift@davinci.snu.ac.kr

Jihong Kim

Department of Computer Science
Seoul National University
Seoul, Korea 151-742
Tel: +82-2-880-8792
Fax: +82-2-871-4912
e-mail: jihong@davinci.snu.ac.kr

Abstract— In designing today's mobile embedded systems such as cellular phones and PDAs, power consumption is an important design constraint. In a CMOS circuit, switching activity accounts for over 90% of total power dissipation. In this paper, we describe a method of encoding opcodes for low-power instruction fetch by reducing the switching activity from the instruction fetch logic. To reduce the switching activity from the instruction-fetch logic, our method encodes opcodes so that more frequently consecutive instruction pairs have a smaller Hamming distance between their opcodes. Our experiment shows that the switching activity reduction of 36.4% to 66.7% is achievable over a naive encoding method.

I. Introduction

Power consumption has become a dominant design constraint for mobile embedded systems such as cellular phones and PDAs. In digital CMOS circuits (that use well-designed logic gates), switching activity accounts for over 90% of total power consumption [1]. Therefore, many techniques have been proposed and developed to reduce the amount of switching activity in multiple levels of design abstraction [2].

One papular approach widely used in reducing the switching activity is to encode digital values so that the number of bit changes by the values are reduced. For example, bus-invert coding tries to minimize the power dissipated in system bus by dynamically inverting the bus lines if the inversion reduces the number of bits switched on system bus [3]. Gray code addressing takes advantage of temporal redundancy in the instruction access patterns during program execution by using Gray-coded addresses as instruction addresses [4]. Register relabeling modifies the register number assignments so that more frequently consecutive register pairs have a smaller Hamming distance, reducing the switching activity in the register fields during instruction fetches and decodes [5].

In this paper, we describe a method of encoding opcodes for low-power instrucion fetches by reducing the switching activity from the instruction fetch logic. Many redundant bit changes between consecutive instructions can be removed by encoding opcodes so that more frequently consecutive instruction pairs have a smaller Hamming distance between their opcodes. In principle, our method is similar to Gray code addressing [4] and register relabeling [5], in that digital values are statically encoded to minimize the number of bit changes by the values. However, we believe that this is the first attempt applying the low-power encoding scheme for the opcode encoding. For benchmark programs we tested, we were able to reduce the switching activity by 36.4% to 66.7% over a naive encoding method. We explain the opcode encoding method in Section II and report the experimental results in Section III.

II. Low-Power Opcode Encoding

A. Basic Idea

When a new instruction is fetched from the instruction cache into the instruction register, many bit positions of the current instruction cache bus and instruction register are switched to the opposite state. The switching activity during the instruction fetch phase is directly proportional to the number of bits switched in the instruction cache bus and instruction register between the successively fetched instructions. In order to reduce the switching activity from the instruction fetch logic, therefore, it is necessary to encode each field of an instruction so that more frequently consecutive field values have a smaller Hamming distance between their values. Our method of encoding opcodes is based on the observation that the distribution of instruction transitions is not uniform, but highly skewed. By assigning opcodes with a smaller Hamming distance to more frequently consecutive instruction pairs, the switching activity from the opcode field can be decreased.

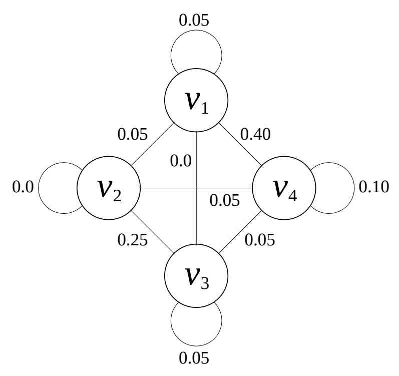

Fig.1. An Example Instruction Transition Graph G

Instruction	Opcode
v_1	00
v_2	01
v_3	10
v_4	11

a. assignment function f_1
$asa(G, f_1) = 1.45$

Instruction	Opcode
v_1	00
v_2	01
v_3	11
v_4	10

b. assignment function f_2
$asa(G, f_2) = 0.85$

Fig.2. Two Opcode Assignment Functions for the Instruction Transition Graph G shown in Fig. 1

B. Instruction Transition Graph

Once an instruction set architecture of a microprocessor is determined and its compiler is available, using the benchmark programs of the target application domains of the microprocessor, we can measure the instruction transition frequencies for all the pairs of the instructions. The instruction transition frequencies can be represented by an *instruction transition graph (ITG)* $G = (V, E, w)$ where V is a set of instructions, E is a set of the undirected edges between all the elements in V, and w is a probability density function that maps each edge $e = (v_1, v_2)$ in E to a real number between 0 and 1. $w(e)$ indicates the relative frequency of the instruction transitions between v_1 and v_2. Fig. 1 shows an example instruction transition graph G with four instructions. Each edge represents the relative transition frequency between two instructions connected by the edge. For example, the transition frequency between instructions v_2 and v_3 is 0.25.

An instruction transition graph of a benchmark program can be constructed by counting the number of transitions between all the pairs of the instructions. Once an instruction transition graph of an individual benchmark program is obtained, a global instruction transition graph can be built by merging the individual instruction transition graphs using an appropriate weighting scheme (e.g., equal-time weighting).

C. Optimal Opcode Encoding

Given a global instruction transition graph $G = (V, E, w)$, our goal is to find an opcode assignment that minimizes the number of bit changes on the opcode field. If an instruction set architecture has M instructions, we need at least $\lceil \log_2 M \rceil$-bit codes to uniquely identify the M instructions. An opcode assignment can be represented by an opcode assignment function $f : V \to S$ where S is a set of binary codes of length $\lceil \log_2 |V| \rceil$. The quality of the opcode assignment function is determined by our power metric $asa(G, f)$, the average switching activity per instruction in G under f, which is defined as follows:

$$asa(G, f) = \sum_{e=(v_1,v_2)\in E} w(e) \times h(f(v_1), f(v_2))$$

where h is a function that returns the Hamming distance between two binary codes. As an example, consider the instruction transition graph G shown in Fig. 1. For two opcode assignment functions f_1 and f_2 shown in Figs. 2a and 2b, we can see that $asa(G, f_1)$ is 1.45 and $asa(G, f_2)$ is 0.85. That is, the switching activity in the opcode field is reduced by 41.4% by changing the opcode assignment function from f_1 to f_2.

Finding an optimal opcode assignment function from an instruction transition graph is an NP-hard problem [6]. For an approximate solution, we have used three heuristics, *2-opt* [7], *simulated annealing* [8], and *slack-based heuristic* [6]. The 2-opt heuristic repeatedly swaps the opcodes of randomly selected two instructions if the swap results in the switching activity reduction. When a locally optimal solution is found, the 2-opt heuristic restarts another local search from a random solution. Simulated annealing is similar to the 2-opt heuristic, but it may swap the opcodes even if the swap increases the switching activity. The slack-based heuristic first sorts all the instruction pairs in the decreasing order based on their contributions to the total switching activity, then changes the opcodes starting from the first instruction pair in the sorted list to reduce switching activity.

D. Decoding Restriction

In practice, for a simpler decoding logic implementation, we often do not have the complete freedom in assigning opcodes to the instructions. For example, the instructions of similar types may have the same bit pattern for some of their opcodes. In order to reflect this restriction during the optimization process, the opcode field is divided into two subfields, the decode-restricted subfield and decode-free subfield. The decode-restricted subfield represents the portion of the opcode whose encoding is limited for a simpler decoder implementation. The decode-free subfield, on the other hand, can be assigned to any code possible.

TABLE I
Top 10 Instruction Pairs with the Highest Transition Frequencies

SPEC CPU95 Benchmark		UTDSP Benchmark	
Inst. Pair	Frequency	Inst. Pair	Frequency
sll, addu	6.6%	addiu, sw	4.8%
mul.d, l.d	3.9%	addiu, lw	4.4%
lw, lw	3.5%	lw, lw	3.2%
sw, sw	3.3%	sw, sw	3.1%
addu, l.d	2.9%	addiu, addiu	2.9%
add.d, l.d	2.8%	addu, sw	2.8%
addu. lw	2.8%	lw, bne	2.2%
addiu, addiu	2.4%	addiu, addu	2.2%
addiu, sw	2.3%	sw, lw	2.2%
addiu, lw	2.1%	addiu, beq	2.1%
total	32.6%	total	29.9%

To find an optimal opcode encoding under the decoding restriction, we use a two-phase optimization method. In the first phase, the encoding of decode-restricted subfield is determined. Instructions that have the same value for the decode-restricted subfield form an instruction group. To minimize the switching activity from the decode-restricted subfield, information on the transition frequencies between the instruction groups is necessary. The transition frequencies between the instruction groups can be extracted from the global instruction transition graph and represented by a *group transition graph (GTG)*. A group transition graph can be constructed in a similar fashion as an instruction transition graph except that a vertex represents an instruction group, not an instruction. Given a group transition graph, we can find the encoding for the decode-restricted subfield with the same manner as the optimal opcode encoding from an instruction transition graph is found. In the second phase, with the decode-restricted subfield fixed, the encoding of decode-free subfield is decided using an appropriate heuristic.

III. Experimental Results

In order to compare the switching activity reduction over a naive opcode encoding method, we have performed experiments using the SimpleScalar tool set [9]. The SimpleScalar architecture is a derivative of MIPS architecture and has 119 instructions with the 7-bit opcode field. Two benchmark suites, SPEC CPU95 benchmark [10] and UTDSP benchmark [11], were used for the experiments to evaluate the applicability of the proposed method in the different application domains. Two ITGs, G_{SPEC} and G_{UTDSP}, were built from the instruction transition information collected from the modified SimpleScalar simulator. As expected, the probability density functions w_{SPEC} and w_{UTDSP} of G_{SPEC} and G_{UTDSP}, respectively, were highly skewed: for example, about 1% of the total instruction pairs accounted for about 90% of the total instruction transitions. Table I lists the top 10 instruction pairs that have the highest transition frequencies. The top 10 instruction pairs (out of the total 7140 pairs) account for 32.6% and 29.9% of the total instruction transitions for SPEC CPU95 and UTDSP, respectively. From the ITGs, the optimized opcode assignment functions f_{SPEC} and f_{UTDSP} are obtained using the three heuristics described in Section II.C. Tables II and III summarize the switching activity reduction results for the selected benchmark programs over the original opcode assignment used for SimpleScalar. As shown in Tables II and III, three heuristics performed equally well. With the 2-opt heuristic, on an average, switching activities were reduced by 49.1% for SPEC CPU95 and 45.2% for UTDSP. In order to consider the decoding restriction, we have performed the experiments with the upper three bits (out of the 7-bit opcode field) set as the decode-restricted subfield for a simpler decoding. Tables IV and V show the switching activity reduction results under the decoding restriction using the 2-opt heuristic. Although the improvements are smaller than ones reported in Tables II and III, on an average, the switching activity reduction of 38.2% and 36.6% is achieved for SPEC CPU95 and UTDSP, respectively.

TABLE II
Switching Activity Reduction Results from SPEC CPU95 Benchmark

	2-opt Heuristic	Simulated Annealing	Slack-based Heuristic
applu	62.1%	62.1%	59.9%
compress	44.3%	45.8%	46.2%
fpppp	50.5%	51.2%	51.0%
gcc	39.6%	38.9%	39.3%
m88ksim	45.2%	47.1%	49.2%
perl	40.3%	38.9%	40.7%
tomcatv	45.9%	44.6%	44.1%
wave	53.9%	51.3%	47.1%
average	49.1%	49.5%	48.0%

TABLE III
Switching Activity Reduction Results from UTDSP Benchmark

	2-opt Heuristic	Simulated Annealing	Slack-based Heuristic
V32	41.7%	38.5%	40.3%
adpcm	43.0%	46.6%	44.5%
edge_detect	37.5%	36.4%	37.0%
histogram	43.9%	40.9%	41.8%
jpeg	45.0%	48.9%	45.9%
lpc	37.4%	40.8%	38.7%
spectral	66.7%	66.7%	66.7%
trellis	46.3%	50.8%	50.6%
average	45.2%	46.3%	45.7%

TABLE IV
Switching Activity Reduction Results from SPEC CPU95 Benchmark under the Decoding Restriction

	No Restriction	Decoding Restriction
applu	62.1%	57.8%
compress	44.3%	32.0%
fpppp	50.5%	39.2%
gcc	39.6%	26.0%
m88ksim	45.2%	30.0%
perl	40.3%	30.7%
tomcatv	45.9%	27.1%
wave	53.9%	48.0%
average	49.1%	38.2%

TABLE V
Switching Activity Reduction Results from UTDSP Benchmark under the Decoding Restriction

	No Restriction	Decoding Restriction
V32	41.7%	32.7%
adpcm	43.0%	34.6%
edge_detect	37.5%	33.0%
histogram	43.9%	33.9%
jpeg	45.0%	37.5%
lpc	37.4%	27.8%
spectral	66.7%	50.0%
trellis	46.3%	42.0%
average	45.2%	36.6%

IV. Conclusion

A method of encoding opcodes for low-power instruction fetch is presented. It is based on the observation that the distribution of the instruction transitions is highly skewed. Our method exploits this observation in the opcode encoding so that more frequently consecutive instruction pairs are encoded to have a smaller Hamming distance between their opcodes. Experimental results show that we can reduce the switching activity from the instruction fetch logic by 36.4% to 66.7% over a naive opcode encoding method. Considering the decoding restriction for a simpler decoding, the encoding method is still effective, reducing the switching activity by 26.0% to 57.8% over a naive encoding method.

Acknowledgement

This research was supported in part by S.N.U. Posco Research Fund (98-09-2093).

References

[1] A. Chandrakasan, T. Shung, and R. W. Brodersen, "Low power CMOS digital design," *Journal of Solid State Circuits*, Vol. 27, No. 4, pp. 473-484, 1992.

[2] S. Devadas and S. Malik, "A Survey of optimization techniques targeting low power VLSI circuits," *Proceedings of the 32nd Design Automation Conference*, pp. 242-247, 1995.

[3] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Transactions on VLSI Systems*, Vol. 3, No. 1, pp. 49-58, 1995.

[4] L. Su, C. Y. Tsui, and A. M. Despain, "Low power architecture design and compilation techniques for high-performance processors," *Proceedings of COMPCON'94*, pp. 489-498, 1994.

[5] H. Mehta, R. M. Owens, M. J. Irwin, R. Chen, and D. Ghosh, "Techniques for low energy software," *Proceedings of the 1997 International Symposium on Low Power Electronics and Design*, pp. 72-75, 1997.

[6] V. Veeramachaneni, A. Tyagi, and S. Rajgopal, "Re-encoding for low power state assignment of FSMs," *Proceedings of the International Symposium on Low Power Design*, pp. 173-178, 1995.

[7] E. Aart and J. K. Lenstra, *Local search in combinatorial optimization*, John Wiley & Sons Ltd., Baffins Lane, Chichester, 1997.

[8] L. Davis and M. Steenstrup, *Handbook of Genetic Algorithms,* L. Davis Ed., PHG Van Nostrand Reinhold, 1991.

[9] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," *http://www.cs.wisc.edu/~mscalar/simplescalar.html*, 1997.

[10] SPEC, "SPEC CPU95 benchmarks," *http://www.spec.org/org/cpu95* , 1995.

[11] M. Stoodley and C. Lee, "UTDSP benchmark suite," *http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.html*, 1997.