Automatic selection of instruction op-codes of low-power core processors

L. Benini, G. De Micheli, A. Macii, E. Macii and M. Poncino

Abstract: A methodology is presented for automatically determining an assignment of instruction op-codes that guarantees the minimisation of bit transitions occurring inside the registers of the pipeline stages involved in instruction fetching and decoding. The assignment of the binary patterns to the op-codes is driven by the statistics concerning instruction adjacency collected through instruction-level simulation of typical software applications. Therefore the technique is best exploited when applied to encode the instruction set of core processors and microcontrollers, since components of these types are commonly used to execute fixed portions of machine code within embedded systems. The effectiveness of the methodology is illustrated through experimental data obtained on a realistic case study, namely, the MIPS R4000 RISC microprocessor.

1 Introduction

Power consumption is becoming one of the most relevant constraints that must be taken into account during the design of modern digital systems, owing to the increased costs of packaging and cooling devices of these types. Obviously, the need of low-power electronic products is even more stringent for portable/wireless applications, such as laptop computers, mobile phones, and personal digital assistants. Usually, these applications carry onboard devices such as general-purpose microprocessors, dedicated core processors, microcontrollers, and DSPs that, besides reduced power dissipation, must ensure reasonably high performance, and thus run at fairly high clock frequencies [1].

Besides resorting to well-known technological choices (e.g. fabrication process, implementation style, clock frequency, supply voltage), there are several architectural solutions that can be adopted to limit the power required by microprocessor-based systems. These include modifications of the processor's organisation [2, 3], careful design of the memory and input/output subsystems [4–7], proper choice of data representation [8], adoption of dynamic power-management strategies [9–11], and exploitation of bus encoding/decoding techniques [12–17].

In this paper we focus attention on the dynamic (or switching) power dissipated internally by a CMOS microprocessor. More precisely, we target the minimisation of the power consumed by the instruction fetching and decoding circuitry. We move from the simple observation that during program execution, sequences of op-codes are continuously read from memory, decoded and stored into CPU registers. Sending an N-bit binary pattern over an on-chip bus and loading it into a register is usually a power consuming operation, since it may require to charge or discharge the considerably high capacitances associated with the bus lines and cells of the register. Up to N-bit transitions can occur on the bus wires and inside the register any time a new value is transferred and stored; properly choosing the binary patterns assigned to the instruction op-codes is then the key to reducing the number of transitions and thus the switching component of the power dissipated by the fetching and decoding logic.

We propose a method for automatically determining a low-power op-codes assignment that consists of two phases. First, instruction-level simulation is performed on traces of typical programs, and the information about the number of adjacencies between pairs of instructions is collected and recorded. Then $\lceil \log_2 K \rceil$-bit binary codes are assigned to the various op-codes using an algorithm which targets the minimisation of the Hamming distance between the codes which are very likely to be adjacent in typical machine instruction streams (K represents the number of op-codes in the microprocessor's instruction set).

We show the effectiveness of our methodology through a realistic case study. In particular, we have taken the MIPS R4000 [18] as the target microprocessor and collected the information on instruction adjacency when some general-purpose programs are executed. Then we have used the low-power encoding schemes of [19, 20] to re-encode the binary op-codes implemented in reality. Finally, we have simulated the execution of the re-encoded instruction streams, and calculated the obtained reduction in switching activity occurring in the op-code bits. Savings are between 15% and 42%, depending on the encoding algorithm and the benchmark program.

Obviously, the choice of an existing microprocessor (MIPS R4000) has been made only for the sake of illustration of the goodness and usefulness of our approach, which is applicable to the early phases of the design flow of new processors. In particular, it is self-evident that the output of the op-code assignment procedure is heavily influenced by the specific software used to compute the frequency of instruction adjacency. Therefore our technique is best suited for dedicated processors, since

IEE Proceedings online no. 19990419
DOI: 10.1049/ip-cdt:19990419

Paper first received 29th May 1998, and in revised form 3rd March 1999

L. Benini and G. De Micheli are with the Computer Systems Laboratory, Stanford University, Stanford CA 94305

A. Macii, E. Macii and M. Poncino are with the Politecnico di Torino, Dip. di Automatica e Informatica, Torino, Italy 10129

they are commonly used within embedded systems to execute the same portion of code over and over. The use of such components, commonly identified as intellectual proprietary (IP) cores, as basic blocks for the development of special-purpose digital systems is becoming a well-established design strategy in the microelectronics industry. This is the reason we believe that the proposed power optimisation strategy could be of interest for IP core engineers and vendors.

2 Automatic selection of low-power op-codes

2.1 Methodology

The methodology for selecting the binary patterns to be assigned to the various instruction op-codes we propose is pictorially summarised in Fig. 1. It consists of two main phases. In the first, instruction-level simulation is executed on typical instruction streams to collect the information concerning the adjacency of pairs of op-codes. Assuming an instruction set consisting of a total of K distinct op-codes, we store the adjacency statistics into a $K \times K$ matrix, called A, whose entries $a_{i,j}$ represent the number of times op-code i immediately precedes op-code j. From matrix A we derive a weighted undirected graph $G_A(V, E, W)$ with $|V| = K$ vertices. We call graph G_A the *instruction adjacency graph*. Each vertex $v_i \in V$ represents op-code i, and each edge $e_{i,j} \in E$ is labelled $w_{i,j} = a_{i,j} + a_{j,i}$ and connects vertex v_i to vertex v_j. If $w_{i,j} \neq 0$, $\forall\ i \neq j$, $G_A(V, E, W)$ is completely connected, i.e. it is a clique. Given the instruction adjacency graph $G_A(V, E, W)$, the target is to assign to each vertex v_i a binary code of length $\lceil \log_2 |K| \rceil$ in such a way that the following cost function gets minimised:

$$C(G_A(V, E, W)) = \sum_{e_{i,j}} \omega_{i,j} \cdot H_{i,j} \qquad (1)$$

where $H_{i,j}$ is the Hamming distance between the two codes of vertices v_i and v_j. The ultimate objective is to assign closer (in the Hamming sense) codes to vertices joined by 'heavy' edges. In this way, pairs of op-codes that are likely to be adjacent in the instruction stream, when stored in the same register at two consecutive clock cycles, will require a small number of switchings within the register's bits, thus reducing the total switching power.

Example: Consider the instruction adjacency graph $G_A(V, E, W)$ of Fig. 2a. If we encode the vertices $v_0, \ldots, v_3$ as in Fig. 2b, the cost is 1710. On the other hand, with the encoding of Fig. 2c the value of the cost function is 1300.

The problem we are addressing (i.e. finding a minimum-length encoding for the vertices of graph $G_A(V, E, W)$ which minimises the cost function of eqn. 1) has been thoroughly investigated in the context of FSM encoding for

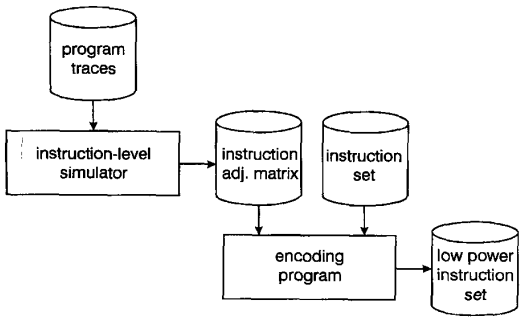

Fig. 1 *Low-power op-codes selection methodology*

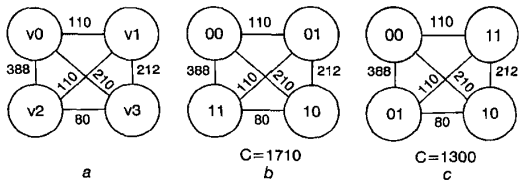

Fig. 2 *Example of $G_A(V,E,W)$ encoding*

low power and efficient solutions can be found in the recent literature ([21] is a comprehensive survey).

Exact encoding algorithms are only applicable to very small graphs (a few tens of vertices), since finding the optimal encoding is simply an instance of the *grph embedding* problem, which is provably NP-hard. This is the reason we propose to adopt two heuristic procedures. The first [19], more accurate, is based on the solution of an integer linear programming problem; it relies on explicit enumeration of all the vertices in the graph and can be used for midsized examples. The second procedure [20], on the other hand, is less accurate but it is fully based on implicit representations of boolean and pseudo boolean (i.e. real-valued) functions by means of BDDs [22] and ADDs [23], and solves the encoding problem as a maximum weighted matching problem. Therefore it is of interest for larger graphs (more than a few hundreds of vertices).

2.2 Applicability and impact

In this Section we discuss the scope of applicability of our methodology and its implications on microprocessor (microcontroller) design. Furthermore, we analyse where and how power is saved by applying op-code re-encoding. As seen in the previous Section, our technique assigns op-codes to instructions in a power-conscious fashion. If we consider processor cores with proprietary instruction sets, the processor architect has complete freedom in designing the instruction set architecture (ISA) and is free to choose instruction op-codes as well. However, it is often the case that microprocessor (or microcontroller) cores must be compatible with a given mainstream instruction set architecture (such as the well-known Intel x86 ISA). In this case, changing the instruction op-codes implies that legacy code is not binary-compatible with the new machine.

Fortunately, the upgrade of legacy code needed to make it compatible with the new op-code encoding is straightforward and can be performed directly on program binaries. It is sufficient to pass legacy binaries through a filter that replaces the op-codes of original instructions with the new, low-power ones. This operation requires a single pass through the binaries. No recompilation is necessary, nor complex binary translation schemes. The only limitation of this procedure is that it cannot be applied to self-modifying codes. The case of migration of legacy code and the absence of any special requirements for external or internal hardware support are two strong points in favour of op-code re-encoding.

To assess the usefulness of op-code re-encoding it is also useful to estimate the power savings that can be obtained. The new op-codes reduce the number of bit transitions between back-to-back instructions. Hence, power will be saved wherever instruction streams are processed. In a microprocessor, instructions are usually read from first-level cache (or, less frequently, from memory), then processed by the instruction pipeline. The flow of the instruction stream is depicted in Fig. 3. For the sake of explanation, we assume a microprocessor architecture with

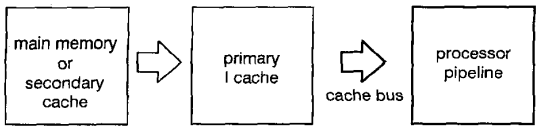

Fig. 3 *Flow of instruction stream in microprocessor*

split first-level instruction cache (I-cache) and data cache (D-cache).

Second-level cache or main memory are accessed only on cache misses (cache misses are generally rare relatively to hits), while instructions are read from I-caches in every clock cycle. Moreover, the sequencing of instruction streams is often destroyed when accessing second-level cache. Hence, op-code re-encoding is particularly useful to reduce transitions, and therefore power, on the bus that connects the I-cache with the microprocessor pipeline [24] and within the pipeline itself [25] (the blocks drawn with bold lines in Fig. 3).

It may be argued that the power consumed in this portion of a microprocessor is only a fraction of the total power dissipation, and op-code re-encoding impacts only a small portion of the total power. Furthermore, in some microprocessors, instruction decoding logic may be on the critical path. In this case, even op-code selection may be dictated by performance constraints; thus, there may be limited degrees of freedom in op-code selection. For example, low-power assignment could be restricted to some specific op-codes. Nevertheless, the power savings provided by our technique, when applicable, can be obtained with little design effort because optimal op-codes are automatically computed.

As a last note, we observe that instruction encoding techniques have been applied successfully in commercial low-power microprocessors. The ARM family [26] provides two instruction sets: a complete set, whose instructions are encoded in 32 bits, and a reduced set with 16-bit instructions. In contrast to op-code re-encoding, supporting two instruction sets imposes significant hardware overhead in the microprocessor. ARM's dual instruction set is claimed to be power efficient [26] because it reduces memory usage for storing programs, thereby reducing the number of cache reads and the power dissipated in the communication between cache and microprocessor. The implementation of instruction encoding techniques in real-life microprocessors confirms that such techniques are indeed useful and may have a nonnegligible impact on overall system power dissipation, even if they target only a fraction of total chip power.

3 Case study

3.1 MIPS R4000 architecture

The MIPS R4000 is a 64-bit microprocessor which provides a 64-bit on-chip floating-point unit, a 64-bit integer arithmetic logic unit, 64-bit integer registers, a 64-bit virtual address space, and a 64-bit system bus. Fig. 4 shows a block diagram of the R4000 microprocessor.

The R4000 microprocessor realises instruction parallelism by using an eight-stage superpipeline; each stage takes one *P-cycle* (*P-clock* operates at twice the frequency of the *master-clock*). The execution of each instruction takes at least eight *P-cycles*. Normally, two instructions are issued at each *master-clock* cycle. Once the pipeline has been filled, eight instructions are executed simultaneously.

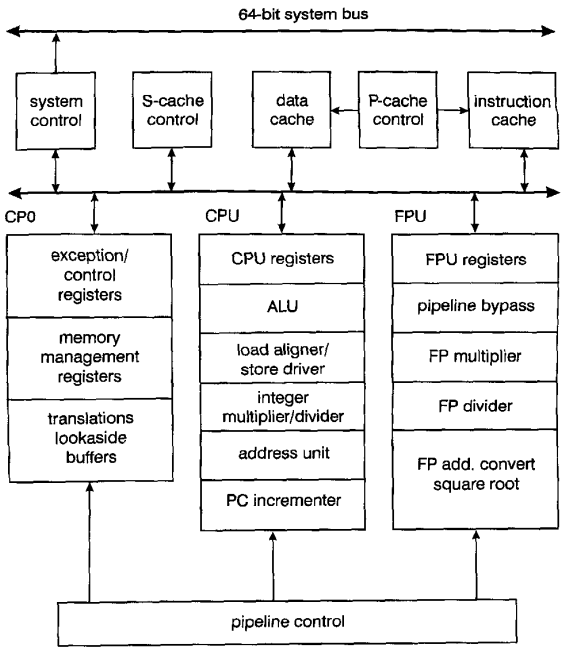

Fig. 4 *R4000 processor internal block diagram*

The microprocessor achieves high throughput by pipelining cache accesses, reducing register access times and allowing the latency of functional units to span more than one pipeline cycle. Fig. 5 shows the eight stages of the instruction pipeline.

3.2 Instruction set summary

Each CPU instruction is 32 bits long. There are three instruction formats, shown in Fig. 6:

- *Immediate* (I-type)
- *Jump* (J-type)
- *Register* (R-type)

The instruction decoding phase is greatly simplified by limiting the number of formats to these three. This limitation means that the more complicated and less frequently used operations and addressing modes can be synthesised

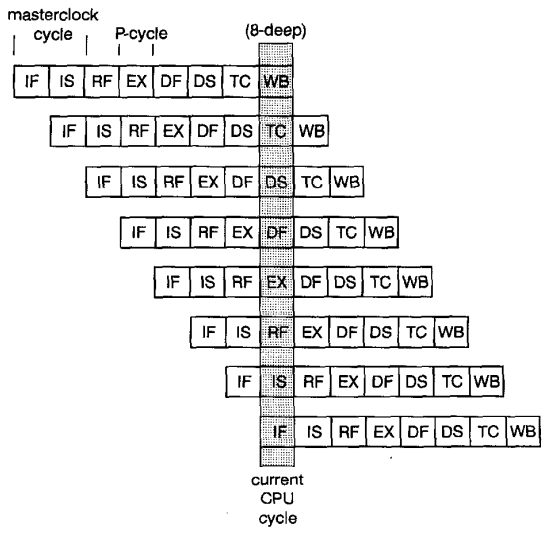

Fig. 5 *Instruction pipeline stages*

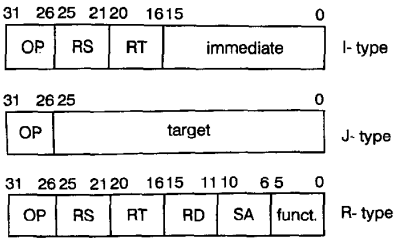

Fig. 6 *CPU instruction formats*
I = immediate type; J = jump type; R = register type

by the compiler, using sequences of these same simple instructions.

The instruction op-codes are six-bits long; their binary encodings are shown in Table 1 (rows are labelled with bits [31..29], and columns are labelled with bits [28..26]).

The SPECIAL op-code (corresponding to the 000000 binary string) actually identifies a total of 52 *R-type* instructions; such instructions are characterized by a six-bit *extended op-code* that is placed in the right-most bits of the binary word. Table 2 shows the actual encoding of such extended op-codes (rows are labelled with bits [5..3], and columns are labelled with bits ([2..0]).

3.3 Low-power op-codes selection

To show the effectiveness of the low-power op-codes selection methodology described in Section 2, we present the results obtained by applying it to the case of the MIPS R4000 microprocessor.

We selected a total of eight software applications, including DBMSs, word processors, data compression and logic synthesis tools. For each application i, we generated the corresponding instruction trace by executing it on a DEC-Station 5000/200 with the MIPS R4000 microprocessor and running the Ultrix operating system configured in single-user/single-task mode. Then we built matrix A_i and the associated instruction adjacency graph G_{Ai}. Finally, we constructed the global graph $G_A{}^G = \Sigma_i G_{Ai}$ that encompasses the instruction adjacency information for all the considered benchmark programs, and we have run on such graph the explicit and the implicit encoding algorithms of [19, 20]. Tables 3 and 4 show the new binary patterns for all the main op-codes. By inspection, the application of the explicit encoding algorithm has modified almost all the op-codes; on the other hand, a lower number of changes has been introduced by the implicit algorithm. This behaviour was expected, since the explicit algorithm introduces fewer approximations in the computation of the near-optimal codes, and therefore it more heavily modifies the initial encoding. The simulation results of the following Section demonstrate that the op-codes of Table 3 are substantially better than the ones of Table 4. On the other hand, as mentioned, the algorithm of [19] may not be used in the case of op-codes longer than six bits because of computational complexity.

An additional step of re-encoding has been applied to the extended op-codes of the SPECIAL instructions (i.e. the *R-type* instructions characterised by an 'all-zero' main op-code). In this case, the re-encoding problem has been formulated in a more articulated way, since additional constraints on the selection of the binary patterns to be assigned to the extended op-codes do exist. In fact, SPECIAL instructions are often adjacent to non-SPECIAL ones, for which the six right-most bits cannot be modified. We have solved the op-codes assignment problem by applying a simple genetic local search algorithm similar to the *Galops* procedure proposed by Olson and Kang [27].

Table 5 reports the new binary patterns that were assigned to the extended op-codes of the SPECIAL

Table 1: Original op-codes

	[28..26]							
[31..29]	000	001	010	011	100	101	110	111
000	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
010	COP0	COP1	COP2	RESRVD1	BEQL	BNEL	BLEZL	BGTZL
011	DADDI	DADDIU	LDL	LDR	RESRVD2	RESRVD3	RESERVD4	RESRVD5
100	LB	LH	LWL	LW	LBU	LHU	LWR	LWU
101	SB	SH	SWL	SW	SDL	SDR	SWR	CACHE
110	LL	LWC1	LWC2	RESRVD6	LLD	LDC1	LDC2	LD
111	SC	SWC1	SWC2	RESRVD7	SCD	SDC1	SDC2	SD

Table 2: Original extended op-codes

	[2..0]							
[5..3]	000	001	010	011	100	101	110	111
000	SLL	RESRVD1	SRL	SRA	SLLV	RESRVD2	SRLV	SRAV
001	JR	JALR	RESRVD3	RESRVD4	SYSCALL	BREAK	RESRVD5	SYNC
010	MFHI	MTHI	MFLO	MTLO	DSLLV	RESRVD6	DSRLV	DSRAV
011	MULT	MULTU	DIV	DIVU	DMULT	DMULTU	DDIV	DDIVU
100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
101	RESRVD7	RESRVD8	SLT	SLTU	DADD	DADDU	DSUB	DSUBU
110	TGE	TGEU	TLT	TLTU	TEQ	RESRVD9	TNE	RESRVD10
111	DSLL	RESRVD11	DSRL	DSRA	DSLL32	RESRVD12	DSRL32	DSRA32

Table 3: Low-power op-codes (explicit algorithm)

	[28..26]							
[31..29]	000	001	010	011	100	101	110	111
000	LW	SB	J	XORI	REGIMM	LDR	LL	BLEZL
001	LUI	LWU	SWL	SDR	LWR	RESRVD3	BGTZL	BEQL
010	SW	RESRVD6	ADDI	SDC1	SLTIU	LDC1	RESRVD1	SCD
011	LWC1	SDC2	COP2	RESRVD7	LWL	RESRVD5	LDC2	LDL
100	SPECIAL	ANDI	JAL	ORI	BNE	BGTZ	LH	LWC2
101	ADDIU	SH	BLEZ	SC	LBU	SDL	DADDIU	RESRVD2
110	BEQ	SLTI	LB	LD	LHU	COP0	SWC2	CACHE
111	COP1	SD	SWR	RESRVD4	SWC1	LLD	BNEL	DADDI

Table 4: Low-power op-codes (implicit algorithm)

	[28..26]							
[31..29]	000	001	010	011	100	101	110	111
000	BLEZ	BGTZ	REGIMM	SLTIU	BNE	SLTI	BEQ	ANDI
001	ORI	XORI	JAL	LUI	SPECIAL	LW	ADDIU	SW
010	ADDI	LB	LWL	LWR	SWL	SWR	LH	CACHE
011	J	SB	LHU	SH	COP1	LWC1	SC	SWC1
100	DADDI	DADDIU	LDL	LDR	RESRVD2	RESRVD3	RESRVD4	RESRVD5
101	BEQL	BNEL	BLELZ	BGTZL	LBU	LWU	SDL	SDR
110	COP0	LL	LWC2	RESRVD6	LLD	LDC1	LDC2	LD
111	COP2	RESRVD1	SWC2	RESRVD7	SCD	SDC1	SDC2	SD

Table 5: Low-power extended op-codes

	[2..0]							
[5..3]	000	001	010	011	100	101	110	111
000	SLL	RESRVD1	XOR	SLT	OR	RESRVD2	SRAV	SRLV
001	SLTU	AND	RESRVD3	RESRVD4	SUBU	BREAK	RESRVD5	SYNC
010	MFHI	MTHI	SYSCALL	MTLO	DSLLV	RESRVD6	DSRLV	DSRAV
011	MULT	MULTU	SLLV	MFLO	DMULT	DMULTU	DDIV	DDIVU
100	ADD	ADDU	SUB	SRA	SRL	DIV	DIVU	NOR
101	RESRVD7	RESRVD8	JR	JALR	DADD	DADDU	DSUB	DSUBU
110	TGE	TGEU	TLT	TLTU	TEQ	RESRVD9	TNE	RESRVD10
111	DSLL	RESRVD11	DSRL	DSRA	DSLL32	RESRVD12	DSRL32	DSRA32

instructions when the re-encoding phase was driven by the data collected on the software applications mentioned. By comparing Tables 2 and 5 it can be seen that only a few extended op-codes have been modified. This is essentially due to the fact that the constraints posed to our genetic local search re-encoding procedure by the non-SPECIAL instructions present in the trace excessively reduce the degrees of freedom that can be exploited to determine advantageous binary assignments of the extended op-codes.

3.4 Results for real software applications

The end result we expect from the application of our technique is a reduction of the switching activity in the op-code bits of some registers of the pipeline stages when sequences of machine instructions are executed. To make sure that this is actually what happens, we have taken the machine code of eight different programs and monitored the average switching activity of each op-code bit. Then, for each application, we have determined the low-power op-codes using both the explicit and the implicit algorithms, re-encoded the original instruction stream using the new op-codes, and calculated the new average switching activity of each op-code bit. Table 6 shows the results of the comparison. Savings are considerably high: Between 30 and 42% for the explicit encoding algorithm, and between 15 and 33% for the implicit one. Notice that the data in the table only refer to the main op-codes (i.e. the six left-most bits of each instruction); improvements in the switching activity of the six right-most bits obtained by optimising the extended op-codes of the SPECIAL instructions have been quite limited (around 5% on average).

As mentioned, the results are obtained using *ad-hoc* instruction encodings. Therefore they clearly show the usefulness of the proposed approach as a tool for helping in determining the most suitable encoding for a special-purpose machine on which a well-established piece of embedded code will be repeatedly executed. However,

Table 6: Average switching activity reduction

Program	Average Switching activity per main op-code bit				
		explicit algorithm		implicit algorithm	
	before	after	savings	after	savings
espresso	0.3025	0.1752	42.06%	0.2045	32.39%
gs	0.2995	0.2017	32.64%	0.2449	18.22%
gunzip	0.2989	0.1941	35.07%	0.2337	21.80%
gzip	0.3206	0.2062	35.67%	0.2582	19.45%
jedi	0.2861	0.1779	37.80%	0.1898	33.65%
latex	0.3036	0.2097	30.91%	0.2576	15.14%
matlab	0.3340	0.2062	38.25%	0.2676	22.86%
oracle	0.3443	0.2117	36.83%	0.2684	22.05%
Global	0.3012	0.1950	35.23%	0.2091	30.57%

the proposed methodology can be beneficial also to designers of general-purpose microprocessors. For devices of this type, the goal would be to determine the *best average* encoding, that is, the one which minimises the power for most of the applications whose execution on the processor is the most likely to happen. The approach to be followed is then that of collecting the statistics on instruction adjacency for all such applications, and then use this information to determine the new encoding. To show the applicability of our technique also to the case of general-purpose machines, we have re-encoded the instruction stream of each program using the op-codes of Tables 3 and 4, and determined the average switching activity per op-code bit before and after re-encoding. The last row of Table 6, named Global, reports the average of these values taken over the eight programs we have considered. Savings are larger than 30%.

4 Conclusions

Microprocessors of the latest generations, including application-specific products (e.g. embedded cores, microcontrollers, and DSP processors), are performance-critical devices, since they tend to run at very high clock frequencies; consequently, they normally consume a considerable amount of dynamic power. Designers are thus constrained to resort to optimisation techniques to keep the available power budget under control. We have directed our attention to the power dissipated by the fetching and decoding logic of a processor. We have demonstrated that the choice of the instruction binary codes plays a key role in the minimisation of the power consumed by these portions of the digital system. We have therefore presented a methodology that can be fruitfully exploited by processor engineers to automatically determine a near-optimal, low-power assignment of the op-codes for special-purpose machines, and have supported our claims concerning the viability and the effectiveness of the proposed technique through experimental results collected on a real-life microprocessor, namely, the MIPS R4000.

5 Acknowledgment

This work is supported, in part, by a grant from SGS-Thomson Microelectronics.

6 References

1 CHANDRAKASAN, A.P., SHENG, S., and BRODERSEN, R. W.: 'Low-power CMOS digital design', *IEEE J. Solid-State Circuits*, 1992, **27**, (4), pp. 473–484
2 GARY, S.: 'Low-power microprocessor design', *in*: RABAEY, J.M., PEDRAM,, M., (Eds), 'Low power design methodologies' (Kluwer, Norwell, MA, 1996) Chap. 9
3 DOBBERPUHL, D.: 'The design of a high performance low-power microprocessor', Proceedings, of ACM/IEEE international symposium on *Low-power Electronics and Design*, ISLPED'96, August 1996, Monterey, CA, pp. 11–16
4 WUYTACK, S., CATTHOOR, F., NACHTERGAELE, L., and DE MAN, H.: 'Global communication and memory optimizing transformations for low power design', Proceedings of ACM/IEEE international workshop on *Low Power Design*, IWLPD-94, April 1994, Napa Valley, CA, pp. 203–205
5 PANDA, P.R., and DUTT, N.D.: 'Reducing address bus transitions for low power memory mapping', Proceedings of IEEE European conference on *Design and Test*, EDTC-96, March1996, Paris, France, pp. 63–67
6 WUYTACK, S., CATTHOOR, F., NACHTERGAELE, L., and DE MAN, H.: 'Power exploration for data dominated video applications', Proceedings of ACM/IEEE international symposium on *Low Power Electronics and Design*, ISLPED-96, August 1996, Monterey, CA, pp. 359–364
7 DIGUET, J.P., WUYTACK, S., CATTHOOR, F., and DE MAN, H.: 'Formalized methodology for data reuse exploration in hierarchical memory mappings', Proceedings of ACM/IEEE international symposium on *Low Power Electronics and Design*, ISLPED-97, August 1997, Monterey, CA, pp. 30–35
8 CHANDRAKASAN, S., and BRODERSEN, R.W.: 'Minimizing power consumption in digital CMOS circuits.' *Proc. IEEE*, 1995 **83**, (4), pp. 498–523
9 ALIDINA, M., MONTEIRO, J., DEVADAS, S., GHOSH, A., and PAPAEFTHYMIOU, M.: 'Precomputation-based sequential logic optimization for low power', *IEEE Trans.*, 1994 **VLSI-2**, (4), pp. 426–436
10 BENINI, L., SIEGEL, P., and DE MICHELI, G.: 'Automatic synthesis of gated clocks for power reduction in sequential circuits', *IEEE Design. Test Comput.*, 1994 **11**, (4), pp. 32–40
11 SRIVASTAVA, M.B., CHANDRAKASAN, A., and BRODERSEN, R.W.: 'Predictive system shutdown and other architectural techniques for energy efficient programmable computation', *IEEE Trans.*, 1996 **VLSI-4**, (1), pp. 42–55
12 SU, C.L., TSUI, C.Y., and DESPAIN, A. M.: 'Saving power in the control path of embedded processors', *IEEE Design Test Comput.*, 1994 **11**, (4), pp. 24–30
13 STAN, M.R., and BURLESON, W.P.: 'Bus-invert coding for low-power I/O', *IEEE Trans.*, 1995 **VLSI-3**, (1), pp. 49–58
14 BENINI, L., DE MICHELI, G., MACII, E., SCIUTO, D., and SILVANO, C.: 'Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems', Proceedings of IEEE 7th Great Lakes symposium on *VLSI, GLS-VLSI-97*, March 1997, Urbana, IL, pp. 77–82
15 BENINI, L., DE MICHELI, G., MACII, E., PONCINO, M., and QUER, S.: 'Reducing power consumption of core-based systems by address bus encoding', *IEEE Trans.*, 1998 **VLSI-6**, (4), pp. 554–562
16 MUSOLL, E., LANG, T., and CORTADELLA, J.: 'Working-zone encoding for reducing the energy in microprocessor address buses', *IEEE Trans.*, 1998 **VLSI-6**, (4), pp. 568–572
17 KALAMBUR, A., and IRWIN, M.J.: 'An extended addressing mode for low power', Proceedings of ACM/IEEE international symposium on *Low Power Electronics and Design*, ISLPED-97, August 1997, Monterey, CA, pp. 208–213
18 HEINRICH, J.: 'MIPS R4000 microprocessor user's manual' (MIPS Technologies, Mountain View, CA, 1994), 2nd edn.
19 BENINI, L., and DE MICHELI, G.: 'State assignment for low power dissipation', *IEEE J. Solid State Circuits*, 1995, **30**, (3), pp. 258–268
20 HACHTEL, G.D., HERMIDA, M., PARDO, A., PONCINO, M., SOMENZI, F.: 'Re-encoding sequential circuits to reduce power dissipation', Proceedings of ACM/IEEE international conference on *Computer-Aided Design*, ICCAD-94, November 1994, San Jose, CA, pp. 70–73
21 MACII, E.: 'Sequential synthesis and optimization for low power', *in*: MERMET, J., NEBEL, S.W., (Eds), 'Low power design in deep subraicron electronics' (Kluwer, Dordrechs, The Netherlands, 1997) Chap. 5.3
22 BRYANT, R.: 'Graph-based algorithms for Boolean function manipulation', *IEEE Trans.*, 1986 **C-35**, (8), pp. 79–85
23 BAHAR, R.I., FROHM, E., GAONA, C., HACHTEL, G.D., MACII, E., PARDO, A., and SOMENZI, F.: 'Algebraic decision diagrams and their applications', *Formal Methods Syst. Des.*, 1997 **10**, pp. 171–206
24 BENINI, L., DE MICHELI, G., MACII, E., SCIUTO, D., and SILVANO, C.: 'Address bus encoding techniques for system-level power optimization', Proceedings of IEEE Conference on *Design Automation and Test in Europe*, DATE-98, February 1998, Paris, France, pp. 861–866
25 CHANG, J. M., and PEDRAM, M.: 'Low power register allocation and binding', Proceedings of ACM/IEEE conference on *Design Automation.*, DAC-32, June 1995, San Francisco, CA, pp. 29–35
26 FURBER, S.: 'ARM system architecture' (Addison-Wesley, Reading, MA, 1997)
27 OLSON, E., and KANG, S. M.: 'Low-power state assignment for finite state machines', Proceeding, of ACM/IEEE International workshop on *Low Power Design*. IWLPD-94, April 1994 , Napa Valley, CA, pp. 63–68