Tag Simplification: Achieving Power Efficiency through Reducing the Complexity of the Wakeup Logic

Mehmet Burak Aykenar, Muhammet Özgür, Vehbi Eşref Bayraktar, Oğuz Ergin
TOBB University of Economics and Technology, Ankara, Turkey
{mbaykenar, mozgur, vebayraktar, oergin}@etu.edu.tr

Abstract—**Contemporary microprocessor cores employ out-of-order execution in order to boost performance. One of the artifacts of the out-of-order execution is the Content Addressable Memory (CAM) cells that allow comparison of the incoming data with the stored value. Many components use these cells inside the processor such as the Issue Queue (IQ), which holds the instructions until their source operands are ready. These processor components receive lookup data each cycle and dissipate significant energy for the comparison operation.**

In this paper we propose a methodology to remove the capacitive load from the lookup buses and reduce the complexity of the comparison circuitry inside the CAM logic. For demonstration purposes we show the design of a new implementation of the IQ that allows the designers to transfer the complexity to the frontend stages of the processor. Our design reduces the dynamic energy dissipation of the CAM array inside the issue queue 15% with virtually no impact on performance.

Keywords-CAM logic; issue queue; wakeup logic; low-power

I. INTRODUCTION

Contemporary superscalar microprocessors employ aggressive techniques, such as dynamic scheduling and out of order execution, to improve performance. One of the components that make out of order execution possible is the instruction queue (also called the issue queue or dispatch buffer), which holds the instructions until they are moved to the execution units. This waiting queue inside the contemporary processors is written to and read from almost every cycle and it is one of the most power dissipating processor structures. The amount of energy dissipated inside the issue queue, when compared to total dissipated energy of the chip, was previously reported as 8% in Intel processors [22] and as high as 20-25% in Alpha 21264 [8][9].

IBM PowerPC processors use numerous queues which are responsible for issuing instructions to different functional units; whereas, some other architectures use one centralized queue for all instruction types [1]. In recent literature, researchers use the word "Issue Queue" for the monolithic hardware, where instructions are waiting for their operand(s) to become ready [2][3][4][5].

Every result producing instruction forwards its destination register tag to the issue queue to inform all of the possible consumers. Since all instructions need to compare their source operands with the incoming tag values, the tag areas of the issue queues are designed by using the Content Addressable Memory (CAM) cells that include comparator circuits.

Issue queues are designed in two parts: A CAM based partition that is used to store the source tags and include wakeup logic and an SRAM partition that is used to store the necessary information to perform operations once the instruction is selected for execution. Although there has been some research efforts to remove the CAM logic with different schemes like using a FIFO queue implementation [7] or dependence matrices [24], contemporary processors still use a CAM based design to avoid any performance degradation [20][25].

The issue queue is one of the most complex hardware modules within a superscalar CPU architecture. Its complexity brings power consumption problem with it. IQ is an important source of power dissipation due to charging and discharging of wire capacitance and the gate capacitance of the transistors that are used within the comparator structure.

In this paper, we propose a new architectural implementation of the IQ to reduce its complexity, by partitioning it into different segments each having a different comparator width. We exploit the fact that some of the operand tags can be represented with fewer bits as they hold a number of leading zero bits. The presence of these bits lead to unnecessary comparisons and provides an opportunity to simplify the comparator structure by using only one transistor to perform a bit level comparison. Our results show that partitioning of the issue queue with respect to comparison complexity can reduce the power consumption of the comparator circuitry used in the wakeup logic of the issue queue around 15% by moving complexity to the front-end.

The remainder of the paper is organized as follows. In Section 2 and 3, conventional superscalar datapath and dynamic scheduling logic are presented. In section 4, our proposed scheme which reduces the complexity of the wakeup logic is explained in detail. The simulation methodology is described in section 5. Section 6 includes the results of our simulations and discussions. Section 7 summarizes the related work. Finally, section 8 provides some clues to the future work and concludes the discussion.

II. SUPERSCALAR DATAPATH

Figure 1 shows a diagram of the superscalar pipeline. Instructions are fetched from the instruction cache and are decoded before their source and destination registers are renamed. In contemporary out of order microprocessors register renaming is employed in order to remove false data dependencies. In the implementation of the register renaming a new physical register is assigned to each result producing instruction and the mapping between the architectural and physical register is stored in the rename table so that any following consumer instruction can be informed about the location of the stored data. Each instruction includes some source tags and a destination register tag inside the issue queue.

978-1-4673-0465-8/11/$26.00 ©2011 IEEE

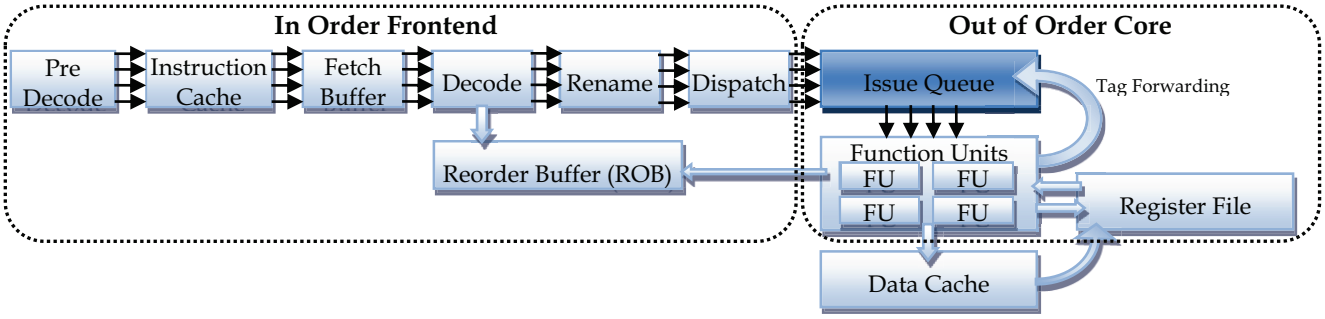

Figure 1. Superscalar Datapath

Each instruction is dispatched to the issue queue and waits for its source operands to be ready. The in-order stages coming before the dispatch stage are all called the front end. After the instructions pass through the frontend they enter the out of order core where they are executed out of the program order. After the execution, each result-producing instruction forwards its result tag to the issue queue in order to inform any dependent instructions.

III. Dynamic Scheduling Logic

In a superscalar processor, multiple instructions can be fetched from the instruction cache and can be issued to the function units. If the fetch width of the superscalar machine is determined to be N and the issue rate of the processor is also N (in fact in contemporary processors the issue rate is usually larger than the fetch rate [2]) then there are N forwarding buses that enter the issue queue. In each cycle, every instruction compares its source operand tag with the incoming tags to see if there is a match. If the stored and forwarded tags match, the valid bit of the corresponding source tag is set to indicate that operand is ready. When both of the valid bits are set, the ready bit of the instruction is set to 1 to indicate the instruction is ready to be issued to the function units. The selection logic selects ready instructions up to the available issue width according to some predefined priorities. The information that belongs to an instruction is read from the payload area only when the instruction is selected for execution.

The CAM array part of the issue queue is the source of continuous activity inside the issue queue and is targeted by most of the researchers that aims to reduce the energy dissipation of the issue queue as we'll discuss in the related work section. The payload area is accessed only when the instruction is written inside the issue queue and it is selected for execution.

Figure 2 shows the internal structure of the issue queue and Figure 3 shows the layout of the CAM array. We assume a pipeline where each instruction has two source operands. Therefore the CAM array contains two source tags which are compared against the tag values coming through forwarding buses.

The CAM cells used in the CAM array are in fact SRAM bitcells with embedded comparator circuitry. Figure 4 shows the circuitry that is used to compare the contents of the SRAM bitcell with the contents of the incoming forwarding bus. In a CAM cell, a version of this comparator is used where each cell contains a comparison block (one pull down path) and the cells forming the tag of the instruction share the precharge transistor and the match line that is connected to the output.

The comparator circuit employs dynamic logic for faster operation. The output node is first precharged to V_{dd} and it is later discharged to ground through the bit-wise comparator blocks in case of a mismatch, when the evaluation signal is high. The output node of the comparator is discharged even when only one input bit pair mismatches and the output stays at V_{dd} if all of the input bits match.

In a processor with 128 registers, the register tags used in the source operand fields of the issue queue are 7 bits wide. Therefore the comparator circuit shown in Figure 4 has 7-bit inputs in this case.

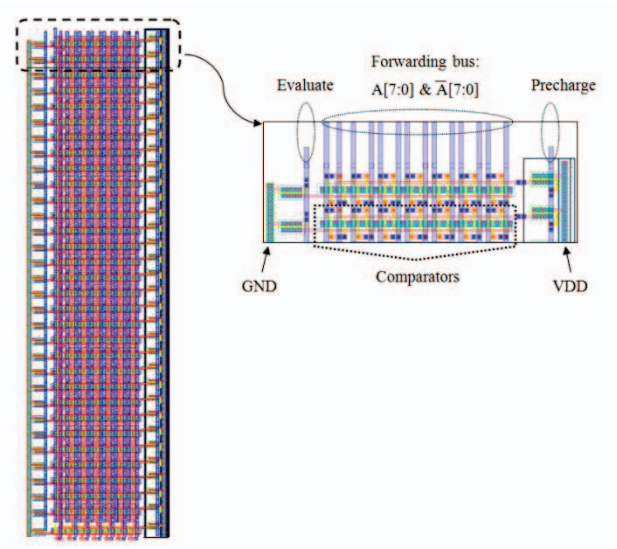

Figure 3. Layout of the CAM array

IV. Tag Simplification: Reducing The Complexity of the Wakeup Logic

A. Comparator Simplification

All of the bit level comparison blocks in Figure 3 are actually performing bit level XOR operation (as shown in Equation 1) and match line that connects these bit level comparisons is actually performing a NOR operation.

$$C = \bar{A}B + A\bar{B} \quad (1)$$

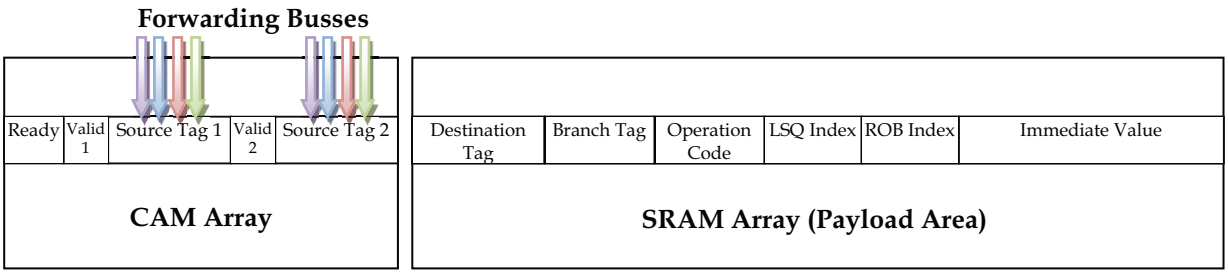

Figure 2. Issue queue organization

Equation 1 mandates the use of 4 NMOS transistors in the pull-down comparison block. However, if it is known for one bit pair that one of the inputs is zero this pull down path can be represented by using only a single transistor.

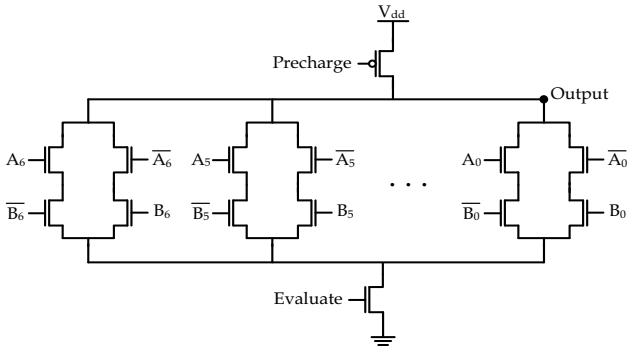

Figure 4.Traditional comparators in an issue queue line.

For a one bit comparator structure in Figure 4, the B input of the bit-wise comparator block is set to 0. In this case, the NMOS transistor whose input is connected to B will be in the cutoff mode and there will be no current passing through that side of the circuit. As B's complement on the left path will take the value of 1, the corresponding NMOS transistor will always be in ON mode. Therefore the circuit can be represented with a single transistor that is connected to A.

B. Issue Queue Partitioning with Tag Simplification

If the upper order bits of the tag stored inside the issue queue is determined to be composed of 0 bits, the comparator circuitry can be simplified. This can be detected in the front-end of the processor where there is plenty of room to fit in a comparison circuitry.

The motivation of tag simplification is to decrease the transistor count in the tag comparators, so that moving the complexity from the dynamic scheduling logic to the front-end of the CPU. The delay and the power consumption of the wakeup logic can be reduced, if the capacitance of the signal bus, driving the forwarded tag, is decreased. When a stored tag value contains leading zeros, the comparator circuit is simplified. For example, if the tag value is 5 (101), it contains 4 zeros (0000) in its upper order bits and the four pull-down blocks are simplified as shown in Figure 5.

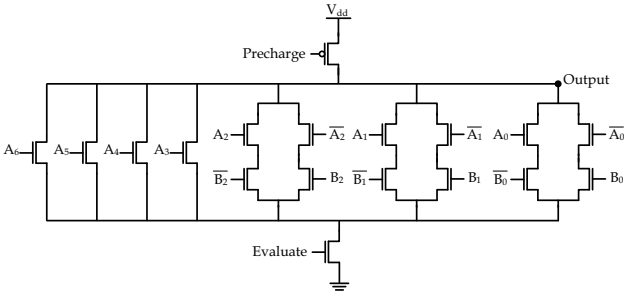

Figure 5. Scheme-1 type comparators

Tag simplification has several effects on the power dissipation in the CAM part of the issue queue:

- As the transistor count is reduced the capacitance on the output node is decreased, effectively decreasing the power dissipation of the comparator circuit.
- The inverted input line driven with the input A is no longer connected to the comparator which decreases the capacitance on the A bar line.
- Since some of the bitcells are not connected to the comparator circuit anymore, the load on the simplified bitcells is reduced.
- As the number of pull-down paths from V_{dd} to ground is decreased, the static energy dissipation of the comparator circuit will be slightly decreased as a bi-product.

We propose segmenting the issue queue so that some entries only hold simplified tags and some entries retain the capability to work with the full-width tags. There can be several different segments that can operate with different width comparators. Figure 6 shows a diagram of the segmented issue queue where each dot shows a regular connection and a line without a dot shows a simplified circuitry. The example is shown for a 24 entry issue queue and three segments where there are entries with 3, 5 and 7 bit comparison capability. As it is shown on the example, some of the forwarding lines now have very small amount of capacitance on them reducing significant energy consumption on the tag drive operation.

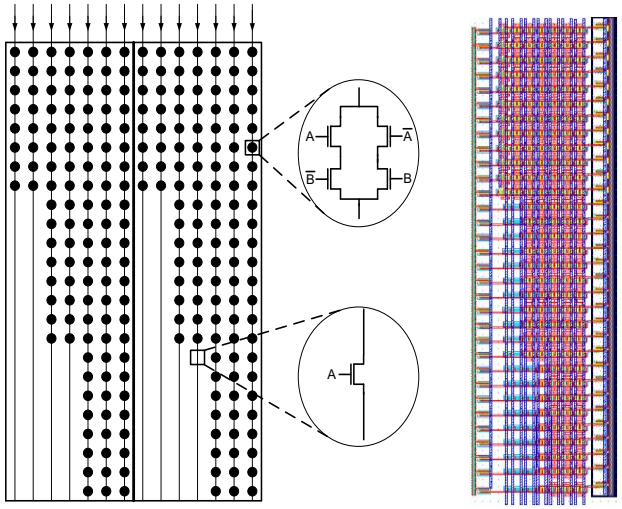

Figure 6. Segmented issue queue

Although it is possible to put a full width tag, without any zeros in the upper order bits, into a simplified tag area, this tag cannot be correctly compared any forwarded value (as it will be assumed by the circuitry that the upper order bits contain zeros). However the opposite is not true; an instruction with simplified register tags can reside in a wide-tag area by placing the zero bits in the upper order portion. Since a full-width tag cannot be written into a simplified entry, the dispatch stops if the processor cannot find a suitable location for the instruction. Therefore it is important to arrange the sizes of the partitions such that the performance degradation is minimal.

Figure 7 shows the distribution of the tag sizes of the instructions that enter the issue queue. If an instruction has two sources it will be dispatched to the segment that supports the wider tag. As it is expected, the tag widths are evenly distributed.

In order to increase the number of instruction that leverages the partition with simplified comparators, we also checked the availability of the operands of the instructions entering the issue queue. If the operands are both ready or if there are no operands associated with the instruction, any segment can be used to improve energy efficiency and to lower the performance

degradation. In this case we start from the segment with the narrowest comparators to dispatch such instructions and move to the segment with wider comparators in case the narrower segment does not have any available entry. When an instruction with a ready operand is placed into the issue queue the valid bit of the corresponding tag is set to 1 in a regular issue queue. Therefore placement of a wide ready tag inside en entry with simplified comparators does not create a problem.

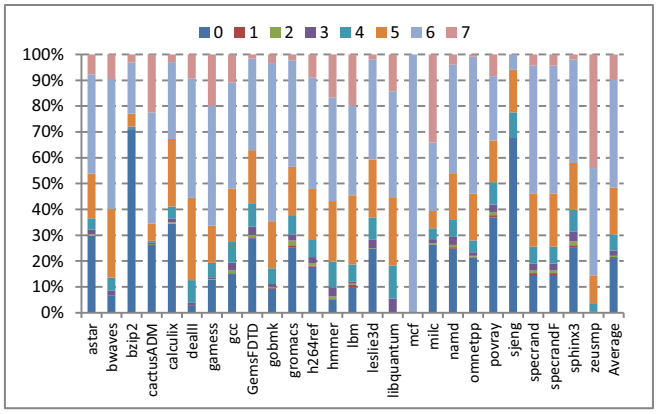

Figure 7. Percentage of tag sizes

Many instructions contain a single operand or enter the issue queue with at least 1 operand ready [11][14]. This property of instructions improves the efficiency of our tag simplification technique as most of the instructions can be placed inside the partitions with simplified comparators.

C. Moving the Complexity to the Frontend

The proposed scheme divides the issue queue into several segments; each segment having a different tag comparator structure. There can be different segmentation configurations with varying number of segments and different number of entries for each segment. For this paper we chose the number of segments as 3 and tried different configurations to observe the energy and performance outcomes of the proposed schemes.

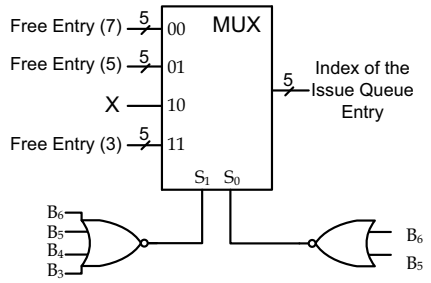

Figure 8. IQ Segment decision in the front end

V. Simulation Methodology

As there are 3 segments inside the issue queue, the only new operation that needs to be done before dispatch is to determine the segment (or issue queue entry id) that the instruction will be written to. For this purpose there needs to be a selection structure in the front-end of the microprocessor to decide which segment will be used for the coming instruction. Since there are 32 entries in the baseline issue queue of our experimental setup, there is a logic that finds an available entry and give the instruction a 5-bit entry id. This logic is already available in a superscalar CPU; what is needed additionally is two more of the same logic that selects empty entries from different parts of the original issue queue. The proposed selection structure shown in Figure 8 allows the selection of an empty entry easily without much performance overhead. In order to detect the 0s in the upper order bit of the tag, the use of only 2 NOR gates is enough as shown in Figure 8.

In order to get an accurate estimation of the energy savings obtained by using the proposed scheme, issue queue layouts are designed and simulated for the delay and power consumption calculations with UMC 90 nm 9-metal layer process in the Cadence software environment and for the determination of the issue queue segment sizes we have simulated our programs with cycle-accurate MSIM simulator [6] with SPEC2006 benchmarks. We skipped the first 100 million instructions and simulated the following 100 million committed instructions. Table 1 presents the configuration of the baseline simulated processor.

Table 1. Configuration parameters for MSIM simulations

Parameter	Configuration
Machine width	4-wide fetch, 4-wide issue, 4-wide commit
Window size	32 entry Issue queue,48 entry load/store queue,128 entry ROB
Function Units	1 Multiplier/Divider (Both FP & INT), 4 ALU (Both FP& INT)
Physical Registers	128 Registers (FP & INT)
L1 D-cache	256 sets,64 byte per block,4-way, 1 cycle hit time
L1 I-cache	512 sets,64 byte per block,2-way,1 cycle hit time
L2 Cache unified	512 sets,64 byte per block,16 way,10 cycle hit time
Branch Predictor	Bimod,2048 table size
BTB	512 sets, 4 –way
I-TLB	16 sets,4096 bytes per block, 4 way, 30 cycle miss latency
D-TLB	32 sets,4096 bytes per block,4-way, 30 cycle miss latency

VI. Results & Discussions

Table 2 shows the different segmentation configurations of the IQ in tag simplification. In the table, there are 10 configurations on each entry of the table, where there are 3 columns that show how many entries belong to that segment. The Seg-7 column shows how many entries include the full width comparators, Seg-5 shows how many entries allow up to 5 bit comparisons and Seg-3 shows the number of entries that allows only 3 bit comparisons. We kept the total number of entries at 32 as it was the case in our baseline.

Table 2. Different Configurations for Tag Simplification

	Number of comparison paths in each segment		
Schemes	***Seg - 7***	***Seg - 5***	***Seg - 3***
1	20	9	3
2	16	10	6
3	13	13	6
4	13	10	9
5	13	6	13
6	12	11	9
7	9	10	13
8	7	13	12
9	6	16	10
10	5	10	17

Partitioning the IQ into three segments slightly increases complexity of the frontend stages. We have just added one 2-input and one 4-input NOR gate and a multiplexer to detect the width of the register tags. These extra hardware modules add a small amount of (43.1 fJ) energy dissipation to the front-end in each cycle of the processor. Figure 9 shows the energy consumption in one clock cycle (2GHz) for all schemes (front-end overhead is included in computations of the energy). Energy savings percentage rises up to 17.5% in Scheme-10.

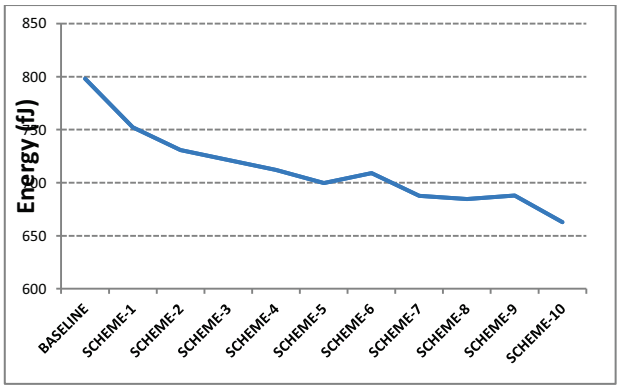

Figure 9. Energy consumption for a clock cycle

Figure 10 shows the Instructions per Cycle (IPC) values achieved by using the proposed technique for each SPEC 2006 benchmarks and on the average. Considerable IPC decrease is started to seen in Scheme-7 for most of the specs. Considering Figure 9 and Figure 10 together it is possible to achieve around 15% energy efficiency in the CAM array part of the issue queue with virtually no impact on the performance. On the other hand it is possible to achieve more energy savings by increasing the number of entries with narrow comparison capability and sacrificing more performance.

VII. Related Work

Unless the wakeup and selection is described as an atomic operation, dependent instructions cannot issue in consecutive cycles as is the case with back-to-back execution [7]. Although there has been some research effort to pipeline the dynamic scheduling logic through speculative wakeup [10], the atomicity of the process is necessary in reality to avoid any performance loss. This atomicity property of the wakeup logic makes it one of the critical paths of the processor.

In [24] and [25] a RAM based wakeup scheme named "Matrix Scheduler" is utilized to move the associative complexity in dynamic scheduling logic to the front-end, however, they only give quantitative results on delays of the wakeup logic. Energy considerations of this scheme is neither tested nor mentioned.

Many works exist in the literature that tries to reduce the energy dissipation of the issue queue. Researchers proposed different schemes to reduce the power consumption of the energy dissipating CAM part of the issue queue. Dissipate-on-match comparators were proposed to reduce the energy dissipated by the traditional mismatch-based comparator circuits [12]. Number of ports in the issue queue was reduced to achieve latency and power reduction in [13]. [16] proposed grouping several instructions and treating them as a single instruction in the wakeup logic to reduce the total power dissipation. In [17] it is observed that the vast majority of the register values generated by a program are read at most once and the use of a table indexed by the register identifier is suggested, instead of an associative search through the comparators in wakeup logic. In [18], disabling the wakeup function for instructions with no operands or for the instruction whose operands are already available is proposed with the dynamically resizing of the issue queue is proposed for energy efficiency.

Most of the instructions enter the issue queue with at least one operand ready. This observation was exploited by Ernst and Austin in [11] for statically partitioning the issue queue into three banks that has entries with 0, 1 and 2 slots for the source tags; effectively lowering the number of comparators and forwarding buses. This way unnecessary use of comparators was avoided and energy efficiency was achieved during the bus drive. In this work we also exploit the same observation to reduce the performance impact introduced by our schemes. With the same observation, Sharkey et al. proposed sharing the same issue queue entry between two instructions and effectively reducing the size of the CAM logic in [14].

Avoiding the driving of the forwarding bus bits if the upper order bits of the forwarded value were the same as the previous cycle was proposed in [15]. This scheme exploits a similar idea to our proposal but checks only two different values on the same forwarding tag and tries to find similarity in two consecutive cycles.

With the observation that many result producing instructions has at most one consumer, Huang et al. proposed reducing the number of comparisons through making producers remember the locations of their consumers in the issue queue [21]. Other schemes try to measure the activity in the issue queue and control the size of the structure [4][23]. In a recent publication [19], a new technique is suggested to reduce the power dissipation of the issue queue by placing the immediate operands in a different table rather than storing them in the issue queue or partitioning the issue queue according to immediate value width.

The technique provided in this paper can be used in conjunction with all of these power-saving schemes to achieve higher energy efficiency in the issue queue. It should be also noted that, contrary to some of these previous techniques that use speculative approaches and increase the complexity of the issue logic, our proposal removes complexity from the issue queue and does not involve any speculation.

VIII. Conclusion

The main goal of the tag simplification is to reduce the power consumption of the wakeup logic without decreasing the IPC. We

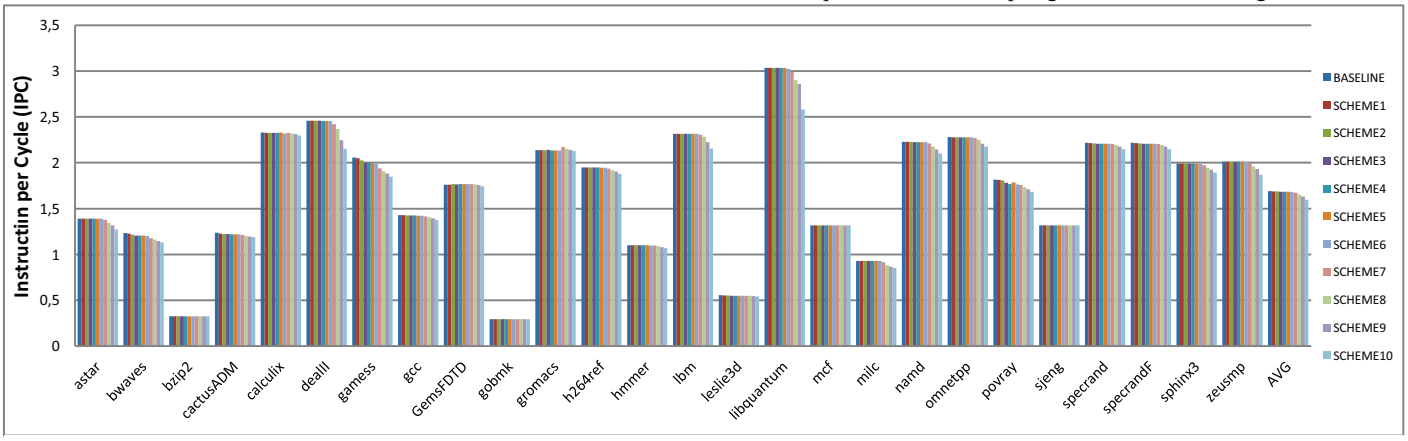

Figure 10. IPC values of SPEC 2006 benchmarks for all schemes and baseline model

show that by segmenting the issue queue into some partitions where the widths of the comparators vary and transferring some complexity from the issue queue to the frontend of the processor, it is possible to reduce the energy dissipation of the issue queue's CAM logic by 20% on the average across all spec 2006 benchmarks. We show that this comes at the expense of some IPC degradation but it is possible to achieve around 15% energy savings with virtually no impact on the performance.

Transferring the activity from the issue queue to the frontend is also beneficial in terms of the thermal design point of the processor. As the heat is spread over the different stages of the CPU, tag simplification allows the dissipation of the generated heat more easily.

In the future, we plan to employ the underutilized forwarding buses and remove the transistors on the upper order bits of the comparators completely. This will physically partition the issue queue into three parts where there will be more opportunities of energy savings. We also plan to find the optimum configuration of the issue queue segmentation in the issue queue by implementing a genetic algorithm based optimum solution finder program.

ACKNOWLEDGMENT

This work was partially supported by the Scientific and Technological Research Council of Turkey (TUBITAK) through the research grant 109E043.

REFERENCES

[1] D. Levitan, T. Thomas, P. Tu, "The PowerPC 620 microprocessor: a high performance superscalar RISC microprocessor," compcon, pp.285, 40th IEEE Computer Society International Conference (COMPCON'95), 1995

[2] G. Hinton et al., "A 0.18-um CMOS IA-32 processor with a 4-GHz integer execution unit" IEEE Journal of Solid-State Circuits, vol. 36, no. 11, November 2001

[3] R. Kalla,B. Sinheroy, J. M. Tendler, "IBM Power5 chip: a dual-core multithreaded processor," IEEE Micro, vol. 24, issue 2, March/April 2004

[4] A. Buyuktosunoglu et al., "Adaptive issue queue for reduced power at high Performance," IBM Research Report RC 21874, Yorktown Heights, New York, Nov. 2000.

[5] D. V. Ponomarev, G. Kucuk, O. Ergin, K. Ghose, P.M. Kogge "Energy-efficient issue queue design," IEEE Transactions on VLSI systems, vol. 11, issue 5, November 2003

[6] J.J. Sharkey, D. V. Ponomarev, K. Ghose, "M-SIM: A flexible, multithreaded architectural simulation environment," Tech Report CS-TR-05-DP01, Dept. of C.S., State Univ of New York at Binghamton, Oct 2005. http://www.cs.binghamton.edu/~jsharke/m-sim/

[7] S. Palacharla, N. P. Jouppi, J. E. Smith, "Complexity-Effective supersclarar processors," ISCA 97 Proceedings of the 24th annual international symposium on computer architecture, vol. 25, issue 2, May 1997

[8] M.K. Gowan, L.L. Biro, D.B. Jackson, "Power considerations in the design of the Alpha 21264 microprocessor," Design Automation Conference, pp. 726 – 731, June 1998

[9] K. Wilcox, S. Manne "Alpha processors: A history of power issues and a look to the future", Proceedings of the CooIChips tutorial. An Industrial Perspective on Low Power Processor Design in conjunction MICRO-33, 1999

[10] J. Stark, M.D. Brown, Y.P. Patt, "On pipelining dynamic scheduling logic," MICRO 33 proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture, ACM New York , NY, USA 2000

[11] D. Ernst, T. Austin, "Efficient dynamic scheduling through tag elimination," In Proc. of the 29th Int'l Symp. on Computer Architecture, pp. 37–46, May 2002.

[12] O. Ergin, K. Ghose, G. Kucuk, and D. Ponomarev, "A circuit-level implementation of fast, energy-efficient comparators for high-performance microprocessors," in Proc. Int. Conf. Computer Design (ICCD), 2002, pp. 118–121.

[13] I. Kim and M. H. Lipasti, "Half-price architecture", in Proc. Of 30th International Symposium on Computer Architecture, 2003.

[14] J.J. Sharkey, D.V. Ponomarev, K. Ghose, and O. Ergin, "Instruction packing: reducing power and delay of the dynamic scheduling logic," in ISLPED, pp. 30-35, 2005

[15] J.J. Sharkey, D. Ponomarev, K. Ghose, O. Ergin, "Reducing delay and power consumption of the wakeup logic through instruction packing and tag memoization," in Proc. of the 4th Workshop on Power-Aware Computer Systems, 2004.

[16] H. Sasaki, M. Kondo, H. Nakamura, "Energy-efficient dynamic instruction scheduling logic through instruction grouping," In Proc. ISLPED, pp. 43–48, Oct. 2006.

[17] R. Canal, A. Gonzalez, "Reducing the complexity of the issue logic," proceedings of the 14th international conference on supercomputing, pp. 327 – 335, May 2000.

[18] D. Folegnani, A. Gonzalez, "Energy-effective issue logic," In Proceedings of the 28th Annual International Symposium on Computer Architecture, pp. 230 - 239, July 2001.

[19] I.C. Kaynak, Y.O. Kocberber, O. Ergin, "Reducing the energy dissipation of the issue queue by exploiting narrow immediate operands," Journal of Circuits, Systems and Computers (JCSC), vol. 19, issue 8, pp. 1689 – 1709, December 2010.

[20] J. Leenstraet. al., "A 1.8 GHz instruction window buffer for an out-of-order microprocessor core," IEEE Journal of Solid-State Circuits, vol. 36, issue 11, pp. 1628-1635, August 2002.

[21] M. Huang, J. Renau and J. Torrellas, "Energy-Efficient hybrid wakeup logic," in proceedings of Intl. Symposium on Low-Power Electronics Design, 2002.

[22] D. Brooks, V. Tiwari, M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations", Proceedings of the 27th annual international symposium on Computer architecture, p.83-94, June 2000, Vancouver, British Columbia

[23] D. V. Ponomarev, G. Kucuk, K. Ghose, "Dynamic resizing of superscalar datapath components for energy efficiency", IEEE Trans. Computers 55(2): 199-213 (2006)

[24] M. Goshima et al., "A high-speed dynamic instruction scheduling scheme for superscalar processors," micro, pp.225, 34th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'01), 2001

[25] P. G. Sassone, J. Rupley, E. Brekelbaum, G. H. Loh, B. Black, "Matrix scheduler reloaded," proceeding in ISCA '07