Journal of Circuits, Systems, and Computers
Vol. 19, No. 8 (2010) 1689–1709

DOI: 10.1142/S0218126610006992

World Scientific
www.worldscientific.com

REDUCING THE ENERGY DISSIPATION OF THE ISSUE QUEUE BY EXPLOITING NARROW IMMEDIATE OPERANDS*

İLKNUR CANSU KAYNAK[†,§] and YUSUF ONUR KOÇBERBER[‡,§]

School of Computer and Communication Sciences,
École Polytechnique Fédérale de Lausanne,
EPFL 1015, Lausanne, Switzerland
[†]*ilknurcansu.kaynak@epfl.ch*
[‡]*yusufonur.kocberber@epfl.ch*

OĞUZ ERGİN

Department of Computer Engineering,
TOBB University of Economics and Technology,
Söğütözü Cad. No. 43 Söğütözü, Ankara, Turkey
oergin@etu.edu.tr

Received 17 July 2009
Accepted 29 June 2010

In contemporary superscalar microprocessors, issue queue is a considerable energy dissipating component due its complex scheduling logic. In addition to the energy dissipated for scheduling activities, read and write lines of the issue queue entries are also high energy consuming pieces of the issue queue. When these lines are used for reading and writing unnecessary information bits, such as the immediate operand part of an instruction that does not use the immediate field or the insignificant higher order bits of an immediate operand that are in fact not needed, significant amount of energy is wasted. In this paper, we propose two techniques to reduce the energy dissipation of the issue queue by exploiting the immediate operand files of the stored instructions: firstly by storing immediate operands in separate immediate operand files rather than storing them inside the issue queue entries and secondly by issue queue partitioning based on widths of immediate operands of instructions. We present our performance results and energy savings using a cycle accurate simulator and testing the design with SPEC2K benchmarks and 90 nm CMOS (UMC) technology.

Keywords: Issue queue; immediate operands; encoding; energy consumption; low power.

*This paper was recommended by Regional Editor Piero Malcovati.
§This work was done when the authors were at the Department of Computer Engineering, TOBB University of Economics and Technology.

1. Introduction

Issue queue (or dispatch buffer) of modern superscalar microprocessors is the indispensable backbone of out of order execution, which is the method that exploits instruction level parallelism. The purpose of contemporary microprocessors is to achieve maximum execution parallelism of instructions in order to maximize the performance. In superscalar microprocessors, multiple instructions are fetched from the instruction cache and they are dispatched to the issue queue after decoding and renaming stages in program execution order. Each instruction that is dispatched to the issue queue is assigned to an entry that stores the immediate operand value and other related information of an instruction such as the functional unit code and tags for source and destination registers. Instructions wait inside the issue queue for being issued to a functional unit until their source operands are available and an execution unit becomes available for execution of the necessary function. This way, the issue queue provides mechanisms for instructions to execute simultaneously which are not dependent to any preceding instructions.

Issue queue is a component that allows out-of-order execution and improves the performance of the processor. Some studies claim that the issue queue may consume up to 25% of the total chip power.[8] Issue queue contains a wake up logic to identify ready operands of waiting instructions and selection logic to choose an appropriate instruction to be issued to a functional unit.[4] In addition to these main energy consumption factors of issue queue, writing the decoded information of each instruction into the fields of the issue queue entries in dispatch stage and reading the contents of these entries when instructions are issued to functional units are other sources of dynamic energy dissipation. It is not a surprising inference that, an increase in the number of issue queue entries results in an increase in the energy consumption of issue queue. Likewise, as the number of bits in an issue queue entry increases, energy dissipation originating from the activities of read and write lines and leakage energy dissipation for each bit increases accordingly.

Many techniques exist in the literature, which try to minimize energy consumption and power dissipation of issue queue logic.[1–4] However, exploiting the different characteristics of the immediate operands of the instructions stored inside the issue queue has not been considered yet to reduce the energy dissipation. Immediate operands occupy a large fraction of the bits inside the issue queue entries. These bits are accessed at each cycle either for writing the information for an incoming instruction at the dispatch stage or reading the contents at the issue stage which result in significant energy dissipation. On the other hand, many instructions do not possess immediate operands and even if the instructions use the immediate field only a small fraction of the bits contain necessary information. These values, that contain a large number of consecutive zeros and ones, are called narrow values as also defined in Ref. 13. Although the immediate operands that come together with the instructions are generally narrow, their upper order bits are written to, stored in and read

from the issue queue entries unnecessarily and existence of these unnecessary bits contribute to static and dynamic energy inefficiency. The importance of CMOS static energy dissipation according to the gate scaling trends briefly explained in Ref. 19.

In this paper, we propose two new techniques to reduce the energy dissipation of the immediate value holding part of the issue queue. The first technique is based on splitting the immediate operand files from issue queue entries into an immediate operand file that is pointed by issue queue entries, in case of an immediate operand need. We aim to lower both static and dynamic energy dissipation of the issue queue, by using smaller number of entries in the separate immediate operand file than the number of entries in the original issue queue and taking the narrow widths of the immediate operands into account. The second technique is based on modifying the issue queue by making use of instructions which do not use their immediate operand files and instructions with narrow immediate operands. The technique of partitioning the issue queue based on existence of immediate operands and widths of immediate operands, where unnecessary bit writes, storage of bits and bit reads are omitted, leads to an energy-efficient issue queue design. The rest of the paper is organized as follows: In Sec. 2, we summarize the background about code compression techniques and their usage in microprocessors. Section 3 describes the motivation for modifying the issue queue according to immediate operands. Section 4 presents the separate immediate operand file technique followed by the explanation of the issue queue partitioning technique in Sec. 5. The simulation methodology is explained in Sec. 6, performance and energy dissipation results are stated in Sec. 7. Finally, we offer our concluding remarks in Sec. 8.

2. Background

A large number of instruction and operand compression techniques are proposed in order to save area and power in instruction storage devices. In Ref. 5, Lin and Chung introduced a technique called operand files remapping in which frequently used operand codes and operands are stored in a dictionary matched with code words and their occurrences in a program are replaced with smaller size code words to reduce the instruction buffer area. Okuma *et al.* proposed several techniques for encoding immediate fields of instruction to reduce the size of instruction memory in embedded systems in Ref. 6, observing that the immediate field is the longest field of instructions in many processors. This approach considers the area consumption of large immediate values but prefers a challenging way of compression which includes time and energy consuming encoding and decoding processes.

A less challenging way of compression is using significance of operand values. Narrowness or zero bytes of values are exploited to reduce the power consumption when the values are transferred, written, read, stored and executed. Villa *et al.* proposed to compress all zero bytes in the data cache into a single bit by adding some additional hardware into RAM to detect and compress zero bytes in memory in

Ref. 7. In Ref. 8 Ponomarev *et al.* considered the narrowness of operands during dispatch, issuing and forwarding phases of issue queue and proposed adding extra zero byte indicator bit for each byte and avoid transferring and manipulating zero bytes of operands.

Benini *et al.* have proposed using code compression to reduce power consumption and storage area requirements with three new schemes based on static and dynamic compression in embedded microprocessors.[29] In Ref. 9 Canal *et al.* introduces pipelining techniques taking the advantage of significance compression for every phase of a pipeline and reducing the power consumption with a little CPI (cycle per instruction) increase. Canal *et al.* developed a new compiler based technique in Ref. 21, by modifying the instruction set architecture, to take the advantage of narrow values. Another compiler based technique is proposed in Ref. 23, where the actual bit widths of variables, independent from the data types of variables defined by the programmer in source code, are determined at compile time. Cao *et al.* defined a new approach to minimize power consumption by exploiting redundant bits, by lowering datapath width of the system.[22] Likewise, it is proposed to place narrow execution units (register files, functional units, caches) for all narrow operands in Ref. 27.

In Ref. 10 Brooks and Martonosi propose two techniques to take the advantage of narrowness of operands in execution units by disabling unnecessary bits of a functional unit those correspond to insignificant bits of an operand and packing multiple narrow significant parts of operands in a single functional unit. In Ref. 20, it is proposed to place new ports to the register file that only reads narrow values to save from dynamic energy consumption. Multiple narrow register values are packed in one register in Ref. 24 and in Refs. 25 and 26 multiple copies of a narrow register value are stored in one register for error detection and correction. Likewise, multiple narrow operands can be executed in one functional unit which results in higher performance speed.[28] Wang *et al.* proposed a banked register file design where upper order bits of the registers are removed in some of register banks to achieve energy efficiency in Ref. 16. Although the same design can be applied to the issue queue, there is a performance penalty due to result value mispredictions. Osmanlioglu *et al.* offers a technique for data holding components of the microprocessor which includes partitioning the issue queue in Ref. 18.

A large number of techniques are proposed in order to reduce the power consumption of the issue queue logic. Some techniques focus on source operands of issue queue entries and their comparisons. In Ref. 1, Ernst and Austin proposed to reduce the number of tag comparators assuming that most of the instructions are dispatched to the issue queue with at least one already available operand. Similarly, in Ref. 2 Kim and Lipasti proposed the techniques of sequential wake up and sequential register file access for instructions with two source operands stating that a high proportion of entries in issue queue do not possess two source operands, thus they need only one wake up logic bus and one register file access port. Folegnani and

Gonzalez, introduced the technique of disabling wake up logic of some partitions of issue queue according to the need for the wakeup logic to reduce power consumption in Ref. 3, observing that empty entries and ready operands of entries in issue queue do not need wake up logic. This technique is an example of portioning the issue queue. Another technique to partition the issue queue is proposed by Palacharla *et al.* in Ref. 4. They simplify the issue queue logic by grouping the dependent instructions into separate in-order execution FIFOs where instructions are issued to execution units parallel from different queues implementing the wake up and selection logic only for the head entries of FIFOs.

Although all of these techniques consider the narrowness of instructions, operands and operation codes in order to reduce the power consumption of a processor in different components of a processor, they pay little or no attention to the storage of large width immediate values in issue queue and their static and dynamic power consumptions when they are being transferred and executed.

3. Motivation

3.1. *Distribution of instructions based on the existence of the immediate operand*

In modern superscalar microprocessors, a large number of instruction formats exist that differs from each other mainly by their lengths, occurrence of prefixes, lengths of operation codes, number of source and destination registers and especially by the existence of immediate operand values and their lengths. For instance, in Intel 64 instruction set architecture, an immediate operand is optional depending on the operation of the instruction.[11] Figure 1 shows the distribution of instructions in

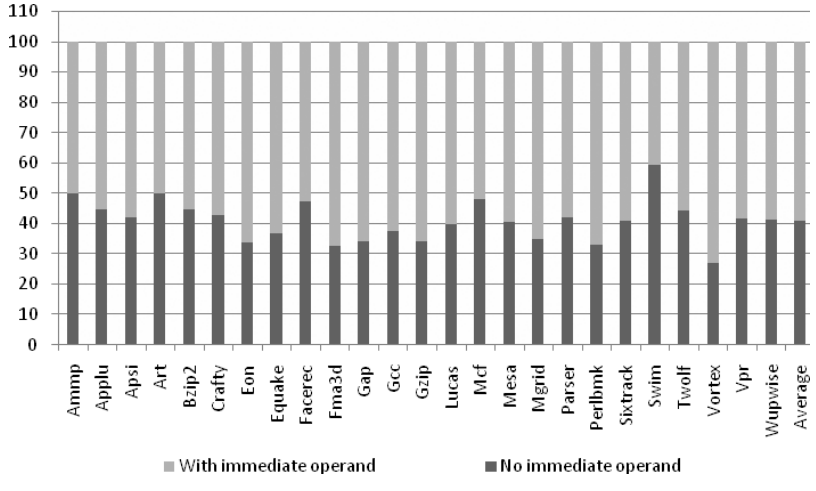

Fig. 1. Distribution of instructions based on their structure.

SPEC CPU2K benchmarks, which are dispatched to issue queue entries, based on their possession of immediate operand values. On the average, across all SPEC CPU2K benchmarks, 40% of the instructions do not possess an immediate operand value. Furthermore, among the instructions that possess immediate operands, width of an immediate operand may vary between 1 and 4 bytes. This means that, an immediate operand can be represented with less the 4 bytes in a 64 bit processor.

Bit width distribution of immediate operands stored in issue queue entries is shown in Fig. 2 and explained in Sec. 3.2. In 64 bit processors, immediate operand values are firstly sign extended to 32 bits before being dispatched to an issue queue entry, if they have a smaller width. Immediate operands occupy a storage space of 32 bits in their issue queue entries. These immediate operands are sign extended to 64 bits prior to their usage,[11] during the issue phase.

Most of the dispatched instructions do not contain immediate operand values or contain immediate operands possessing narrow significant bit widths. When these instructions are written to their reserved issue queue entries at dispatch phase and read at issue phase, the whole area reserved for these immediate operands is processed and also the corresponding bitcells cause static energy dissipation, even if they do not contain any valid data or if they contain insignificant bits only because of their physical existence in issue queue entries. Each of these operations results in energy dissipation. As observed in Ref. 8, in a dispatch bound issue queue where the contents of source registers are stored in issue queue entries, 65% of energy dissipation of issue queue is caused by the operations at dispatch and issuing phases. Immediate operand values contribute to an important proportion of energy dissipation in issue queue, because of their large bit widths: 32 bits in a 64 bit microprocessor.

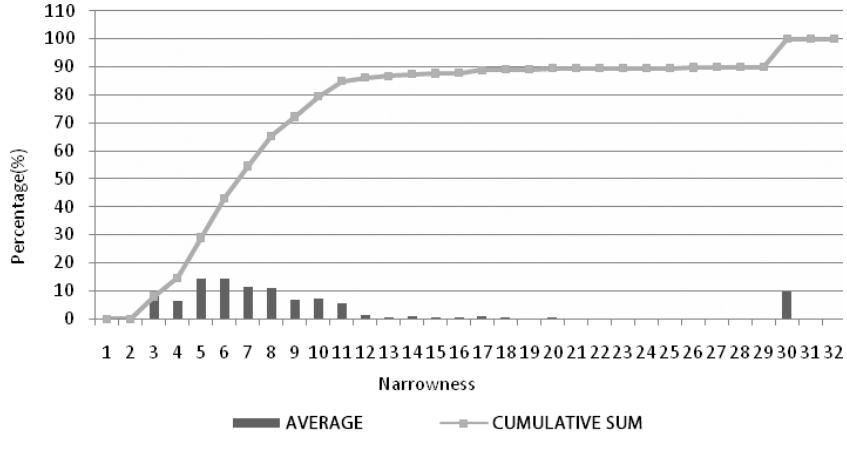

Fig. 2. Bit width distribution of immediate operands stored in issue queue entries.

3.2. *Exploration of narrow immediate operands*

Narrow values are values that contain many unnecessary bits used for sign extension. In 64 bit microprocessors, operands are sign extended to 32 bits to be stored in issue queue entries. The real bit width of a value is its significant part's bit width. Figure 2 shows the real bit width distribution of immediate operands in SPEC CPU2K benchmarks and the cumulative sum of each real bit width. For example, 8.1% of the immediate operands can be expressed with only 4 significant bits and 14.6% of the immediate operands can be expressed with 4 or less bits. As shown in Fig. 2, a high proportion of immediate operands can be expressed with less than 32 bits. When these narrow immediate values are stored in the issue queue entries, the bits occupying the immediate area, other than the significant part of the value, are insignificant sign extension bits. For instance, the number 5, which is equal to 0101 in two's complement notation, can be represented with only 4 bits, a sign bit and data bits, instead of 32 bits. It is unnecessary to write and read the other 28 bits of an immediate operand area which represent the extension of the sign of the value. In order to avoid power dissipation caused by write and read operations of unnecessary sign extension bits of immediate operands and static energy dissipation of these bitcells, their existence and thus unnecessary operations on these values may be omitted.

4. Separate Immediate Operand Files

In this section, we propose a technique in which, immediate operand files of issue queue entries are removed throughout the issue queue and immediate operands of instructions are stored in separate immediate operand files called narrow and wide immediate operand files. For this purpose, the 32 bits of immediate operand areas of issue queue entries are replaced with pointer bits as shown in Fig. 3. The number of pointer bits can change according to the number of entries in the immediate operand file. For example, if there are 16 immediate operand entries in the immediate operand file, each issue queue entry can point to one of these entries with 4 bits, if one needs an immediate operand storage area. For instructions with no immediate operands, the contents of these pointer bits are not important as the processor will not seek an immediate value at the issue stage. The immediate operand file has two partitions.

The first partition, narrow immediate operand file, is for narrow immediate operands, while the second partition, wide immediate operand file, is for wide immediate operands. Narrow and wide immediate operands are identified according to the narrowness factor. If an immediate operands' real bit width is less than or equal to the narrowness factor, it is identified as a narrow operand and it is placed to an immediate operand storage area in a narrow immediate operand file. Otherwise it is called a wide immediate operand and it is placed in a wide immediate operand file entry. The sizes of immediate operand files and the narrowness factor are set based

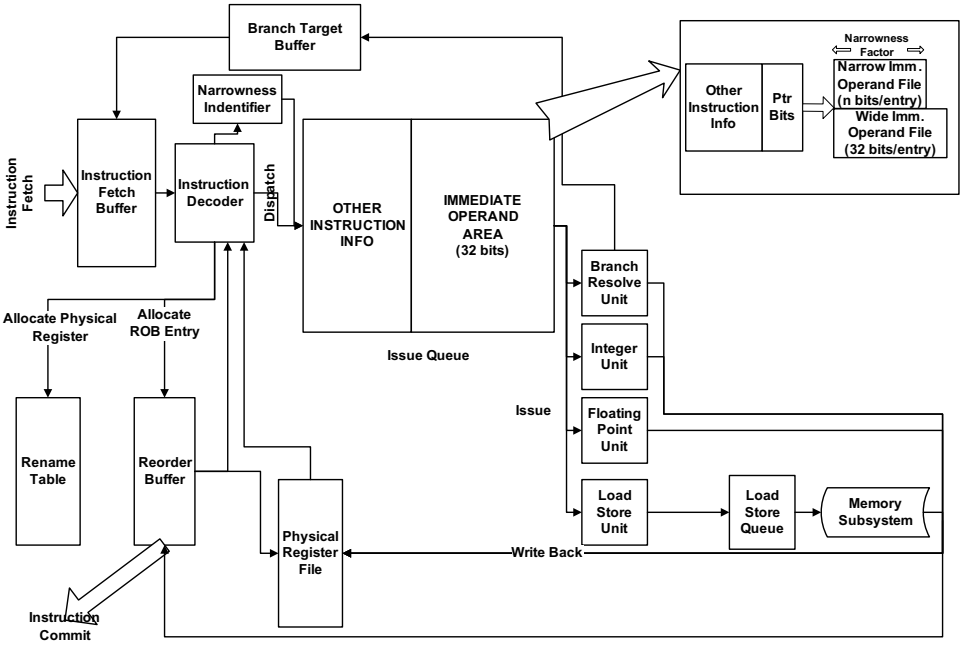

Fig. 3. Separate immediate operand files.

on Figs. 1 and 2, according to the ratio of instructions with immediate operands and the real bit width distribution respectively.

In this new configuration of a microprocessor, the architectural layout must be modified in order to determine the narrowness of the immediate operand of an instruction (whether it is narrower than the narrowness factor or not) by a special hardware component, narrow value identifier, parallel to one of the pipeline stages in order not to lengthen the critical path of the processor. The most suitable architectural phase, for the narrowness identifier of immediate operands, is the renaming phase which is considerably a time consuming complex operation. The narrowness identifier logic will operate on immediate operands of instructions while their registers are renamed as seen in Fig. 3.

Figure 4 shows a 24 bit wide two stage zero detector, constructed with 3 of consecutive zero detectors which are 8 bits wide. Consecutive zero detectors simply operate as OR function.[17,30] Outputs of detectors are inverted and input to the second stage of the detector which is 3 bits wide consecutive zero detector. For example, if the value is 8 bit narrow, most significant 24 bits of the 32 bit value are input to the narrow value identifier circuit in order to show that the 32 bit value can be represented with only 8 bits. If the value is narrow, precharged output remains at HIGH otherwise discharges and signals LOW. As expected, by implementing small sized inverters and accommodate small number of the transistors at the second stage, 3 nmos transistors in this example, TPHL and precharge delays are very low

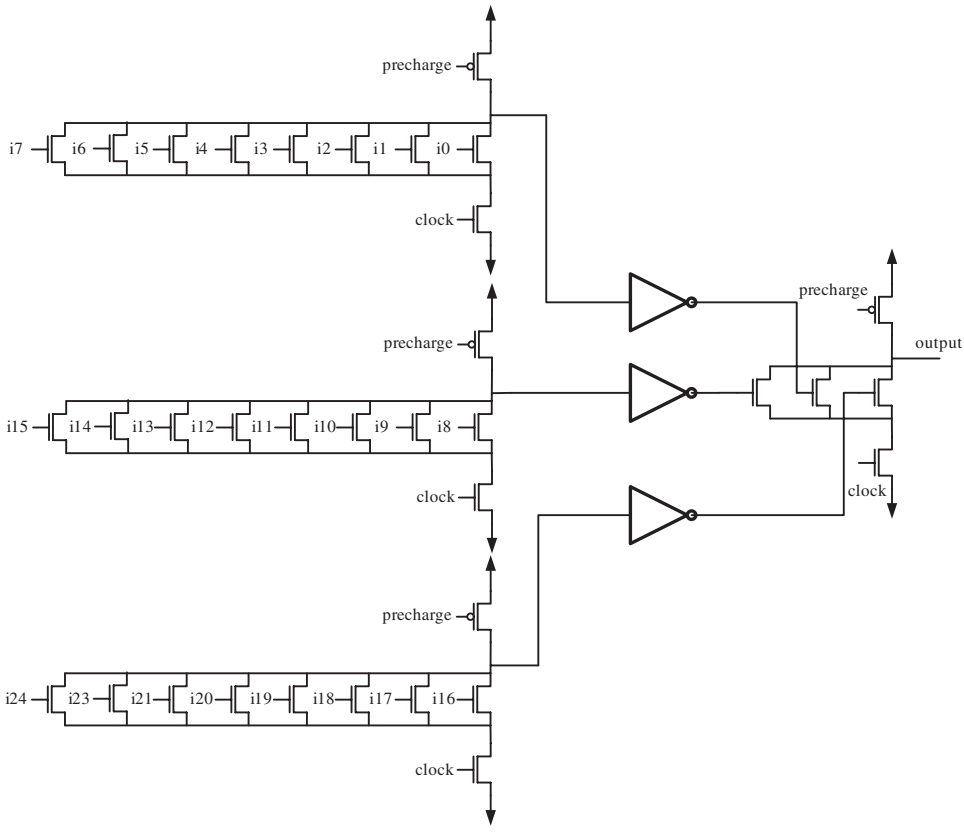

Fig. 4. 24 bit wide 2 stage consecutive zero detector.

compared to the one period of the clock cycle. To detect a narrow value which consists of only ones, the values are first inverted then input to the zero detector circuit. Results of the zero detector and one detector are ORed to form a narrow value identifier circuit.

Whenever an instruction does not possess an immediate operand, it will continue in the pipeline as in the normal case without needing an immediate operand entry from the immediate operand files and also, no bitcell will be written and read for immediate operands of this kind of instructions. If an instruction has an immediate operand with a real bit width larger than the narrowness factor it will be assigned to an issue queue entry and also an immediate operand entry from the wide immediate operand file. Whenever an instruction with a narrow immediate operand is dispatched, an issue queue entry and an entry from the narrow immediate operand file must be assigned to this instruction. When an available entry from the narrow immediate operand file does not exist, an entry from the wide immediate operand file can be assigned to this instruction. Unless one of the required conditions are met, the

instruction will stall at dispatch stage, until the necessary entries become available which may result in a small performance loss that will be discussed in Sec. 7.

In issue bound processors, instructions read their operand values from the register file during the issue phase. Contemporary microprocessors, such as the Intel's Pentium 4, follow this scheme and use an adjusted pipeline to achieve back to back execution.[14] Reading of the immediate value from the immediate file can be overlapped with the register read stage as the processor already spends the time to fetch the operands from the register file. Therefore adding a new index to the issue queue and accessing the immediate file at the issue stage relatively neither affects the cycle time of the processor or the issue process nor adds an additional cycle to the issue stage.

5. Issue Queue Partitioning

The main idea of the issue queue partitioning technique is to reduce the energy dissipation of issue queue by not writing the unnecessary sign extension bits in issue queue entries and not reading them to functional units. For this purpose, all or some of the bit cell area reserved for immediate operand bits will be removed from some parts of issue queue. Consequently, the storage area and leakage energy dissipation of the immediate operand bits will also be reduced. Energy savings are achieved by not driving insignificant sign bits during a write operation and removing the sense amplifiers and prechargers on a read port of issue queue immediate operand bit cells.

In order to reduce the energy consumption, the issue queue is partitioned into three components. The first component is dedicated to instructions without any immediate operand and is called the no-immediate partition, the second component is for instructions with narrow immediate operands (called the narrow partition) and the third component is reserved for instructions with wide immediate operands (in our case 32 bits), called the wide partition, as shown in Fig. 5. For the first issue queue partition, bitcells, read and write logic for immediate operand bits are eliminated as entries from this partition are reserved for instructions which do not possess immediate operands. For the narrow partition only significant bits of immediate operands are driven and the repeating sign bits are omitted. The wide partition allows the storage of full-width immediate operands.

The narrowness factor (which is defined as the number of bits provided for the narrow immediate of operands) of the narrow partition is determined based on the bit width distribution of immediate operands shown in Fig. 2 and the number of entries for each issue queue partition are determined by the ratio of instructions that do not possess any immediate operands as shown in Fig. 1. The designer should use the information depicted in Figs. 1 and 2 to determine the sizes of the issue queue partitions and the value-width of the narrow immediate operand partition.

Before the dispatch of an instruction, it must be detected whether the immediate operand of the instruction is narrow or wide. For detection of narrowness, a special

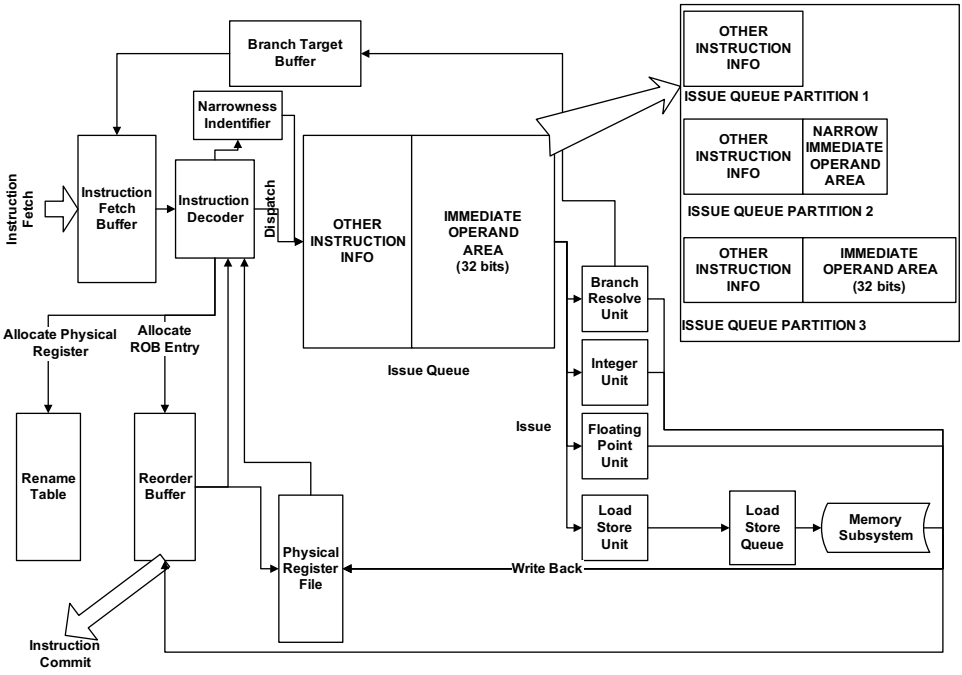

Fig. 5. Issue queue partitioning technique.

hardware component detecting the insignificant 0 and 1 bits must be located before the issue queue as seen in Fig. 4. This operation does not necessarily occur at dispatch stage in order not to lengthen the critical path of an instruction in the pipeline. The narrowness of an immediate operand can be detected at any of frontend stages, possibly in parallel with other operations such as the register renaming as shown in Fig. 5. Although fast leading zero/one detectors can be designed using the dynamic logic,[13] they still have a circuit delay and should be placed in a pipeline stage that tolerates the extra latency. This way, a decrease in processor frequency can be avoided. On the other hand, it is better to detect the narrowness of an immediate operand as close to the dispatch stage as possible, in order to minimize the extra energy dissipation for transferring required control signals between the pipeline stages.

After detecting whether the immediate operand of an instruction is narrow, it will be placed in one of the three issue queue partitions: no-immediate, narrow or wide partition. Whenever an instruction does not possess an immediate operand, an entry from the no-immediate partition of issue queue is allocated for that instruction. Instructions with no immediate operand are not allowed to be stored in other two partitions, because necessary number of issue queue entries is reserved for them in their partition. Allowing them to be stored in other partitions will lead to the stall of instructions with an immediate operand at dispatch stage. If the instruction includes

an immediate operand, it will be placed in one of the other two partitions according to the real bit width of its immediate operand. If an instruction with an immediate operand width smaller or equal to the narrowness factor, it is said to be a narrow instruction and an issue queue entry is allocated to this narrow instruction from the narrow (second) partition. When an instruction is narrow, if there are no free entries in the narrow partition, an entry from the wide (third) partition is allocated for this instruction. If an instruction's immediate operand width is larger than the narrowness factor, the instruction will be placed in the third partition. For example, when an immediate operand is determined to be narrow, but the narrow issue queue partition does not have an available entry, the incoming instruction can be sent to the wide issue queue partition. Unless these conditions are satisfied, the instruction will stall at dispatch stage until an appropriate issue queue entry for this instruction is available. When this happens, some energy efficiency is sacrificed for reducing the performance impact of not finding an available entry and stalling the pipeline. In order to guarantee forward operation, at least one entry should be present with a full width immediate field. Otherwise the pipeline may get stuck as there would not be any available issue queue entry when an instruction with a wide immediate operand arrives.

After instructions wait in the issue queue for their execution conditions to be met, they are issued to functional units. In issue stage the immediate operands of instructions, which are placed in narrow partition, must be sign extended before they are executed in order to meet the input width of the functional units.

A 32 row 32 column baseline immediate operand partition of issue queue layout is shown in Fig. 6. Partitioning the issue queue into three pieces according to the immediate operand files of the instructions reduces both the dynamic and static

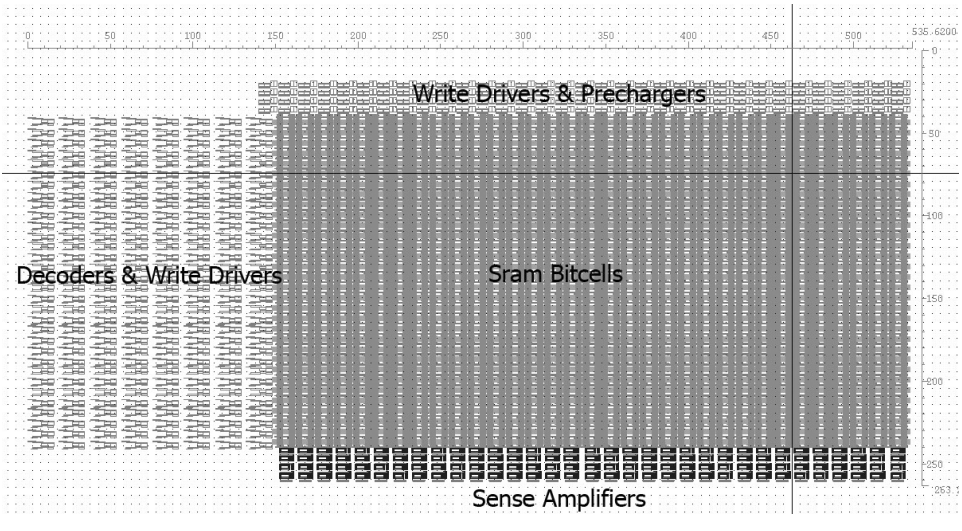

Fig. 6. Layout of the baseline immediate operand area of issue queue.

energy dissipation of the issue queue. Dynamic energy dissipation is reduced because of the bit lines that are not driven for the upper order bits of the narrow immediate values. Word select driver lines are also shorter for narrow storage components and require less energy to be activated. Since the number of entries for the storage that holds the immediate values is reduced, the capacitance on the bit lines is also reduced which results in a reduction in the energy dissipation of the write drivers and pre-chargers. If a partition holds only a few entries, the sense amplifiers, which are connected to the bit lines for faster read operation, can also be removed for further energy efficiency.

Static energy dissipation due to the leakage current is also reduced through the proposed technique. Since the unused bitcells are removed from the processor instead of not driving the bit lines and word lines for unused immediate operands, the number of paths between the voltage source and the ground is decreased. Consequently, the static energy dissipation of the immediate field of the issue queue decreases proportionally with the percentage of bits removed from the storage space.

In the baseline, with a unified issue queue, the processor stalls at the dispatch stage when there are no available entries inside the issue queue. In the proposed technique this constraint is a little tighter as the possibility of a stall is increased due to the reduced number of entries for each instruction type. Especially for the instructions with full-width immediate values, the number of entries that the instruction can be placed is limited to the capacity of the third partition. Therefore the proposed scheme may result in performance degradation if the sizes of the partitions are not determined appropriately.

6. Evaluation Methodology

The cycle accurate out-of-order x86 microprocessor simulator PTLsim[12] is used to observe the real bit width distributions of immediate operands in the issue queue. To test our methodology, we modified the simulator's issue queue structure, rename and dispatch phases. The baseline processor performance is determined with one 32-entry issue queue in one cluster. The significant bit widths of immediate operands of instructions are determined at the rename stage, the instructions are dispatched to issue queue according to their real bit widths and the number of writes and reads are determined at dispatch and issue stages of instructions. The other configuration components of the simulator are listed in Table 1.

Spec CPU2K benchmarks are run on various simulator configurations to observe the performance variation and to observe the distribution of all instructions to issue queue partitions and immediate operand files to calculate the power and energy consumptions of different configurations. 1.5 billions of instructions are committed for each benchmark. In PTLsim, there is no need to warm up the simulator, because PTLsim directly executes the programs on the host CPU until it switches to cycle accurate simulation mode, so there is no way to count instructions in this manner.[12]

Table 1. Configuration of the simulated processor.

Parameter	Configuration
Machine width	4-wide fetch, 4-wide issue, 4-wide commit
Window size	80 entry load/store queue, 128-entry ROB
Number of clusters	1 cluster
Function units	2 Arithmetic Logic, 2 Floating Point, 2 Load, 2 Store
L1 I-cache	32 KB, 4-way set-associative, 64 byte line, 1 cycles hit time
L1 D-cache	32 KB, 4-way set-associative, 64 byte line, 2 cycles hit time
L2 Cache unified	256 KB, 16-way set-associative, 64 byte line, 6 cycles hit time

For estimating energy dissipations at dispatch and issue phases, event counts from the simulator were combined with the energy dissipations measured from actual full custom CMOS layouts of the processor components as proposed in Ref. 15. A 90 nm CMOS (UMC) technology with a V_{dd} of 1 Volt was used in order to measure the energy dissipation.

7. Results and Discussions

Creating a separate immediate file and decoupling the immediate values from the issue queue offers an opportunity to reduce energy dissipation of the issue queue as described in Sec. 4. In this proposed technique, the instructions with no immediate operands can get any entry inside the issue queue just as it is in the baseline. The instructions that use the immediate operand are dispatched to the issue queue only if there is an available entry in the suitable immediate file partition. Different from the issue queue partitioning scheme, in our implementation of the immediate file scheme it is possible for an instruction without an immediate operand to proceed to the issue queue if there is an available entry. For the issue queue partitioning scheme we did not allow no-immediate instructions to occupy the entries reserved for instructions with narrow or wide immediate values.

Figure 7 shows the configurations for the proposed technique and energy and area reduction results for the varying immediate file configurations. In this proposed technique, there is an immediate operand file to store immediate operands of instructions in issue queue. The immediate file is divided into a narrow and a wide partition. Every immediate operand entry in the narrow partition has a number of bit cells equal to the narrowness factor. Immediate operand entries in wide partition have 32 bitcells. "File size" in Fig. 7 represents the number of immediate operand entries in each partition. Total file size can be defined as the sum of narrow file size and wide file size. As seen in Fig. 7, total file size is set to 16 (narrow file size + wide file size) in the first four configurations. To point to all of the immediate operand file entries from issue queue entries when they are needed by the instruction, every issue queue entry has an additional 4 pointer bits which will also cause some energy dissipation additionally. However, the 16 immediate operand entries are deficient when the results of Fig. 1 are considered. It is observed in Fig. 1 that 60% of the

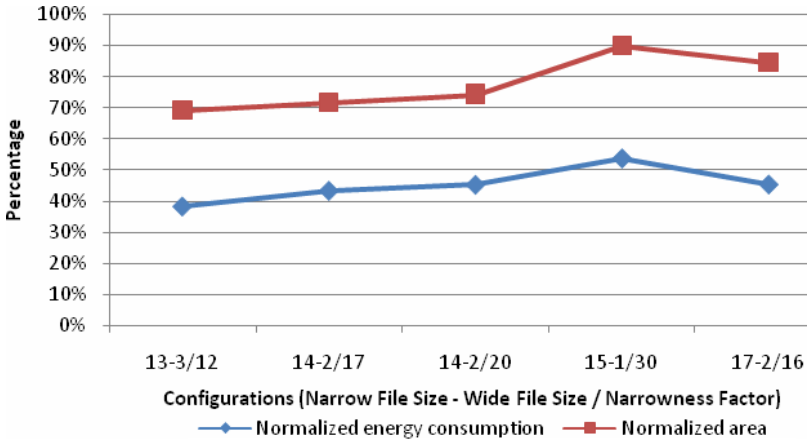

Fig. 7. Results of separate file for different configurations.

instructions possess an immediate operand. It means that, on the average, 60% (19/32) of the issue queue entries needs an immediate operand entry from the immediate operand file, but in this case 5 pointer bits for each issue queue entry is needed. This configuration is also experimented in the fifth configuration (17-2/16) in Fig. 7.

Four configurations with an immediate operand file size 16 and a configuration with immediate operand file size 19 is tried out. You can see the results of these configurations in Fig. 7. The narrowness factors of the configurations are determined according to the breaking points in Fig. 2. The immediate operand file partition sizes are also determined based on the narrowness factors' cumulative sum in Fig. 2. For the first four configurations, narrowness factor and proportionally, immediate operand file partition sizes are modified.

The energy and area reduction and performance changes are determined by comparing the energy dissipation and area of the proposed separate immediate file technique with an issue queue in a normal microprocessor which has 32 entries and each entry has 32 bitcells for immediate operands. This means that, the energy and area reductions listed in Fig. 7 are for only the operations on immediate operands in issue queue. All configurations consist of three energy components. First, if the operation is write, narrow value identification energy which was subjected in Sec. 4 should be considered. Second, read or write energy including of all the issue queue components used during the operation is calculated. Last, if the value to be read is narrow, extension operation energy which is implemented by transmission gate multiplexor is calculated.

In first four configurations, a small amount of performance loss is observed due to the small number of total immediate file sizes than in the base configuration. Although 60% of the instructions possess an immediate operand, only half of the issue queue is reserved for these instructions which resulted with approximately 0.25% performance degradation.

It is observed that, the smaller the narrowness factor, the higher the energy and area reduction. Performance (IPC-instruction per cycle) did not change noticeably, because the sizes of the immediate operand files are determined based proportional to the distribution of narrowness of immediate operands.

Power consumption distribution of immediate operand files is shown in Fig. 8. The aim is to minimize the usage of wide immediate file but its usage is compulsory for instructions with wide immediate operands, also to avoid stalls of instructions with narrow immediate operands. As observed in Fig. 8, energy dissipation proportion of wide immediate operand file partition becomes smaller as its size decreased and the narrowness factor increased, as expected. As the narrowness factor increases, much more number of the immediate operands are placed in narrow partition which results in energy saving.

As the results reveal, using a separate immediate file reduces the area and the energy dissipation of the immediate part of the issue queue. However this decrease is smaller than the benefits of issue queue partitioning scheme as the additional indexing bits are occupying area and dissipating additional energy.

For the technique described in Sec. 5, the original issue queue is partitioned into three components: For instructions without an immediate operand (IQ1), for instructions with a narrow immediate operand (IQ2) and for instructions with a wide immediate operand (IQ3). Our purpose is to have minimum power dissipation and area without losing performance by mostly using the no-immediate and narrow partitions, as the number of bits inside the immediate operands that are driven to read and write drivers are either zero or a smaller number than the datapath width. On the other hand, it is desirable to avoid performance degradation caused by the stalled instructions during the allocation of an available issue queue entry. Instructions can stop at the dispatch stage and cause performance degradation if an entry

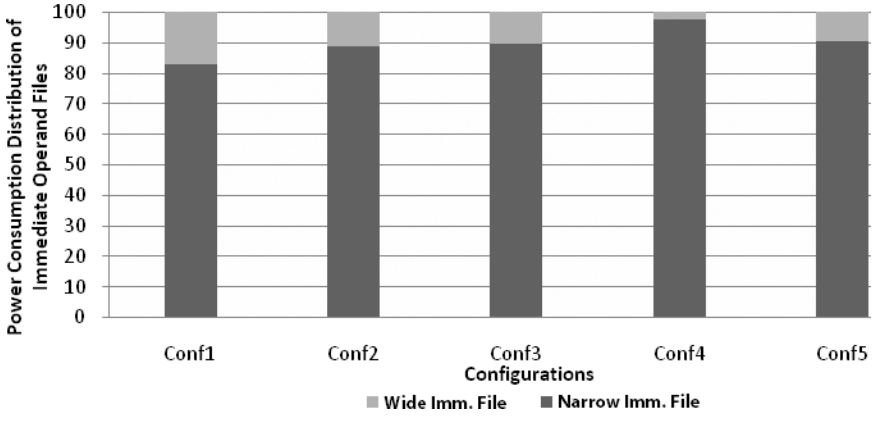

Fig. 8. Power consumption distribution of immediate files at Fig. 7.

from the required partition is not available. In order to decrease the number of stalls caused by waiting instructions at the dispatch phase, the entries of the issue queue must be distributed to partitions with a fair heuristic, mostly proportional to characteristics of the dispatched instructions shown in Figs. 1 and 2. It is observed that 40% of the instructions do not possess an immediate operand, so that 40% of the issue queue entries, 13 entries, are reserved for instructions without immediate operands. The rest of the issue queue entries, 19 entries, are shared between instructions with a narrow immediate operand and instructions with a wide immediate operand. Narrow immediate file size is determined based on the narrowness factor and narrowness factor is determined from the critical breaking points in Fig. 2 for the first seven configurations. Narrow immediate file size is set to the narrowness factor's cumulative sum percentage determined in Fig. 2 times 19, as there are 19 entries left for instructions with an immediate operand. For the eighth configuration, the number of entries for each issue queue partition is distributed equally.

Different configurations based on different narrowness factors are figured out in Fig. 9 with their energy and area change results. Seven configurations of the simulated processor with different number of entries in the issue queue components and narrowness factor for each partition of issue queue are experimented. The parameters of different issue queue configurations used in our experiments are indicated in Fig. 9. For each configuration we chose to have a total of 32 entries inside the issue queue in order to avoid a large performance drop and to have a fair comparison with our baseline that has a single 32-entry issue queue.

The energy and areas comparisons are done with a 32-entry issue queue where each entry has a 32-bit immediate operand area. The energy and area reduction results depicted in Fig. 9 are for only read and write operations on immediate

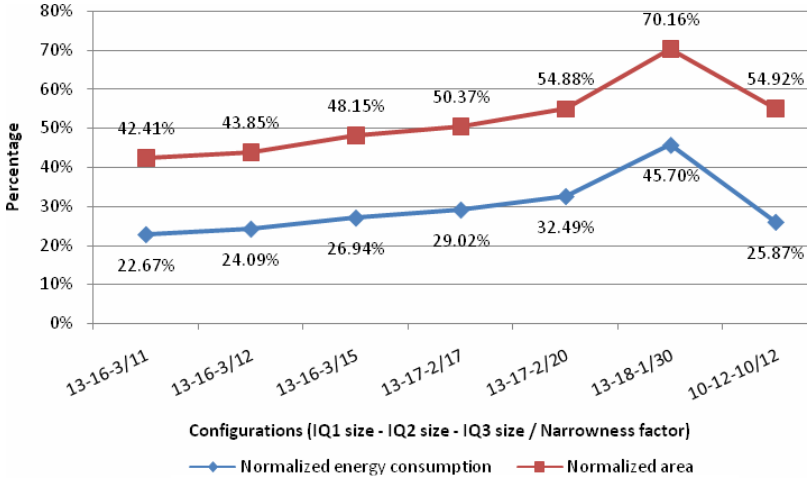

Fig. 9. Results of issue queue partitioning for different configurations.

operands in the issue queue. The energy dissipation to identify narrow value, read, write and extend the immediate operands in issue queue in the proposed technique is compared to the energy dissipation that is caused by reading and writing immediate operands in a normal issue queue. The results show that, removing some part or the entire immediate field from some entries of the issue queue significantly reduces the energy dissipation on the immediate part of the issue queue. The smaller the narrowness factor, the higher the energy and area reduction, as observed in first six configurations. As the narrowness factor gets smaller, the number of bitcells decreased and the energy to process these bitcells also decreases. The area of the immediate part of the issue queue is also drastically reduced as the narrowness factor is reduced. Area reduction can be observed by comparing the issue queue layout pictures depicted in Figs. 6 and 10.

No performance loss is observed in first six configurations, because issue queue partition sizes are determined based on the distribution of instruction types. In the seventh configuration, 2.86% performance loss is due to the unnecessary high number issue queue entries that are reserved for instructions with wide immediate operands. The results in Fig. 9 show that 32-bit wide immediate operands are rarely used and create energy inefficiency in the issue queue. As these values are usually not used, when the space reserved for the wide operands are removed, the performance of the processor is not affected. Increasing the number of entries for the wide immediate operands, results in the most performance degradation as the instructions without an immediate operand or with narrow immediate operand fail to find an available entry at dispatch time.

Although there are cases in Figs. 7 and 9 that offer more energy reduction at the expense of some performance degradation, our simulation results should be taken as the proof of concept rather than exact figures. A processor designer should analyze the workloads that will be run on the processor and has to decide on the power/performance tradeoffs according to the design budget.

Figure 11 shows the power consumption distribution for the issue queue partitions of configurations in Fig. 9. Since there are no bitcells reserved for the immediate

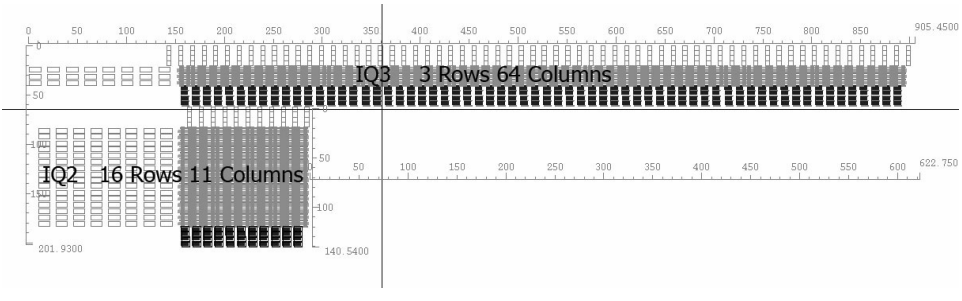

Fig. 10. Layout of the immediate operand area of partitioned issue queue for Conf. 1(13-16-3/11) in Fig. 9.

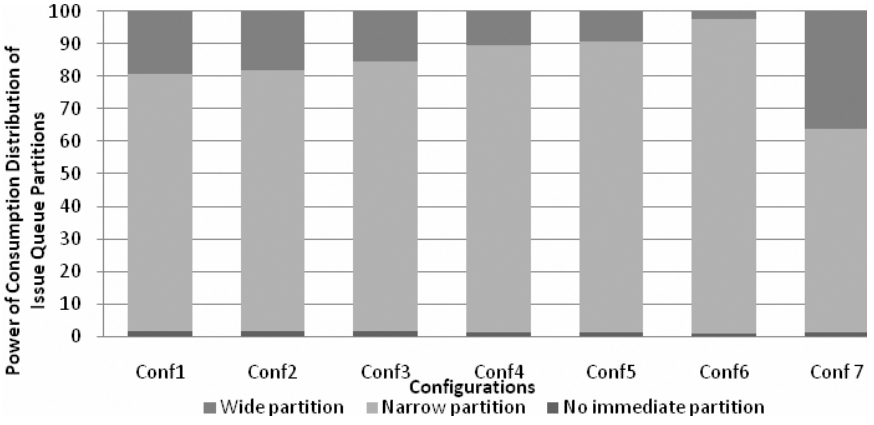

Fig. 11. Distribution of power consumption of issue partitions at Fig. 9.

values in the no-immediate partition, dynamic and static energy consumptions for immediate operands do not exist in this partition. Only decoder energy is calculated in order to make a realistic comparison to the baseline issue queue consumption. Because a big proportion of the instructions are stored in narrow partition, it is the most energy consuming partition. Although the purpose is to minimize the usage of wide partition, it is used for instructions with wide immediate operands and also to avoid stalls of instructions with narrow immediate operands. Most of the energy is consumed by the narrow partition and power consumptions of no-immediate and wide partitions are minimized.

8. Concluding Remarks

Many instructions do not use the immediate values in contemporary microprocessors. However all of the entries of the issue queue structure, which hold the instructions before they proceed to the execution stage, are designed to hold immediate values for each instruction as all instructions can be placed in any entry. The storage space allocated for the immediate operands are written to and read from unnecessarily when an instruction without an immediate operand enters the issue queue. Moreover, many of the immediate values that are actually used by the instructions do not need the full bit width of the storage space and can be expressed with less number of bits.

In this paper we proposed a scheme to design an issue queue that can exploit the immediate value characteristics of the incoming instructions for energy efficiency. The first proposed technique is to create an immediate file separate from the issue queue, that stores immediate operands of issue queue entries if they exist. In the second proposed technique, we show that significant energy savings are achieved by partitioning the issue queue into three parts, where there is a part for instructions without any immediate operands, narrow immediate operands and regular immediate

operands. Our simulation results show that on the average across all Spec CPU2K benchmarks, it is possible to achieve up to nearly 61% power and 30% area reduction for the immediate operand fields of the issue queue with a very small amount of performance degradation with the separate immediate file technique. With the issue queue partitioning technique, 74% power reduction and 45% area reduction is achieved, for immediate operands stored in issue queue, with no performance loss.

References

1. D. Ernst and T. Austin, Efficient dynamic scheduling through Tag Elimination, *Proc. 29th Int. Symp. Computer Architecture (ISCA'02)* (2002), pp. 37–46.
2. I. Kim and M. H. Lipasti, Half-price architecture, *Proc. 30th Int. Symp. Computer Architecture (ISCA)* (2003), pp. 28–38.
3. D. Folegnani and A. Gonzalez, Energy-effective issue logic, *Proc. 28th Int. Symp. Computer Architecture (ISCA'01)* (2001), pp. 230–239.
4. S. Palacharla, N. P. Jouppi and J. E. Smith, Complexity-effective superscalar processors, *Proc. 24th Int. Symp. Computer Architecture (ISCA'97)* (1997), pp. 206–218.
5. K. Lin and C. P. Chung, Code Compression techniques using operand field remapping, *Computers and Digital Techniques IEE Proc.* **149** (2002) 25–31.
6. T. Okuma, H. Tomiyama, A. Inoue, E. Fajar and H. Yasuura, Instruction encoding techniques for area minimization of instruction ROM, *Proc. 11th Int. Symp. System Synthesis* (1998), pp. 125–130.
7. L. Villa, M. Zhang and K. Asanovic, Dynamic zero compression for cache energy reduction, *Proc. 33rd Ann. ACM/IEEE Int. Symp. Microarchitecture (MICRO'00)* (2000), pp. 214–220.
8. D. V. Ponomarev, G. Kucuk, O. Ergin, K. Ghose and P. M. Kogge, Energy efficient issue queue design, *IEEE Trans. Very Large Scale Integration (VLSI) Syst.* **11** (2003) 789–800.
9. R. Canal, A. Gonzalez and J. Smith, Very low power pipelines using significance compression, *Proc. 33rd Ann. ACM/IEEE Int. Symp. Microarchitecture (MICRO'00)* (2000), pp. 181–190.
10. D. Brooks and M. Martonosi, Dynamically exploiting narrow width operands to improve processor power and performance, *Proc. 5th Int. Symp. High Performance Computer Architecture (ISCA'99)* (1999), p. 13.
11. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M, 2007.
12. M. T. Yourst, Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator, *ISPASS* (2007).
13. O. Ergin, Exploiting narrow values for energy efficiency in the register files of superscalar microprocessors, *16th Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'06)* (2006), pp. 477–485.
14. G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker and P. Roussel, The microarchitecture of the Pentium 4 processor, *Intel Technol. J.* Q_1 (2001).
15. D. Ponomarev, G. Kucuk and K. Ghose, AccuPower: An accurate power estimation tool for superscalar microprocessors, *Proc. Conf. Design, Automation and Test in Europe (DATE'02)* (2002), pp. 124–129.

16. S. Wang, H. Yang, J. Hu and S. G. Ziavras, Asymmetrically banked value-aware register files for low energy and high performance, *Microprocessors and Microsystems* **32** (2008) 171−182.
17. Y. Osmanlioglu, Y. O. Kocberber and O. Ergin, Reducing parity generation latency through input value aware circuits, *Proc. 19th Great Lakes VLSI Symp. (GLSVLSI'09)* (2009).
18. Y. Osmanlioglu, Y. S. Hanay and O. Ergin, Modifying the data-holding components of the microprocessors for energy efficiency, *J. Circuits Syst. Comput.* **18** (2009) 1093−1117.
19. G. Sery, S. Borkar and V. De, Life is CMOS: Why chase the life after? *Proc. 39th Ann. Design Automation Conf.* (2002).
20. A. Aggarwal and M. Franklin, Energy efficient asymmetrically ported register files, *Proc. 21st Int. Conf. Computer Design (ICCD'03)* (2003).
21. R. Canal, A. Gonzalez and J. Smith, Software-controlled operand gating, *Proc. Int. Symp. Code Generation and Optimization* (2004), p. 125.
22. Y. Cao and H. Yasuura, A system-level energy minimization approach using datapath width optimization, *Int. Symp. Low Power Electronics and Design (ISLPED'01)* (2001), pp. 231−236.
23. M. Stephenson, J. Babb and S. Amarasinghe, Bitwidth analysis with application to silicon compilation, *Proc. ACM SIGPLAN 2000 Conf. Programming Language Design and Implementation* (2000), pp. 108−120.
24. O. Ergin, D. Balkan, K. Ghose and D. Ponomarev, Register packing: Exploiting narrow-width operands for reducing register file pressure, *Proc. 37th Ann. IEEE/ACM Int. Symp. Microarchitecture (MICRO'04)* (2004), pp. 304−315.
25. O. Ergin, O. Unsal, X. Vera and A. González, Exploiting narrow values for soft error tolerance, *IEEE Computer Architecture Letters (CAL)* **5** (2006).
26. J. Hu, S. Wang and S. G. Ziavras, In-register duplication: Exploiting narrow-width value for improving register file reliability, *Proc. Int. Conf. Dependable Systems and Networks (DSN'06)* (2006), pp. 281−290.
27. S. Kumar, P. Pujara and A. Aggarwal, Bit-sliced datapath for energy-efficient high performance microprocessors, *Proc. 4th Workshop on Power Aware Computer Systems (PACS'04)*, Portland, Oregon, USA, December 2004.
28. G. H. Loh, Exploiting data-width locality to increase superscalar execution bandwidth, *Proc. 35th Ann. ACM/IEEE Int. Symp. Microarchitecture (MICRO'02)* (2002), pp. 395−405.
29. L. Benini, F. Menichelli and M. Olivieri, An adaptive data compression scheme for memory traffic minimization in processor-based systems, *IEEE Int. Symp. Circuits and Systems (ISCAS'02)*, 4: IV−866−IV−869 (2002).
30. Y. O. Koçberber, Y. Osmanlıoğlu and O. Ergin, Exploiting narrow values for faster parity generation, *Microelectronics International* **26** (2009).