PAPER Special Section on Advanced Processors Based on Novel Concepts in Computation

An Energy Efficient Instruction Window for Scalable Processor Architecture

Min CHOI[†a)], ***Student Member and*** **Seungryoul MAENG**[†], ***Nonmember***

SUMMARY Modern microprocessors achieve high application performance at the acceptable level of power dissipation. In terms of power to performance trade-off, the instruction window is particularly important. This is because enlarging the window size achieves high performance but naive scaling of the conventional instruction window can severely increase the complexity and power consumption. In this paper, we propose low-power instruction window techniques for contemporary microprocessors. First, the small reorder buffer (SROB) reduces power dissipation by deferred allocation and early release. The deferred allocation delays the SROB allocation of instructions until their all data dependencies are resolved. Then, the instructions are executed in program order and they are released faster from the SROB. This results in higher resource utilization and low power consumption. Second, we replace a conventional issue queue by a direct lookup table (DLT) with an efficient tag translation technique. The translation scheme resolves the instruction dependency, especially for the case of one producer to multiple consumers. The efficiency of the translation scheme stems from the fact that the vast majority of instruction dependency exists within a basic block. Experimental results show that our proposed design reduces the power consumption significantly for SPEC2000 benchmarks.

key words: *instruction window, superscalar, low-power microarchitecture, reorder buffer, issue queue*

1. Introduction

Enlarging the size of instruction windows can lead to performance improvement. However, naive scaling of the conventional instruction window severely affects the complexity and power consumption. In fact, Folegnani and Gonzalez [11] showed that the reorder buffer and the issue queue, constituting the instruction window, is the most complex and power-dense parts in dynamically scheduled processors. Thus, much research has been conducted to increase the size of the instruction window without negatively impacting power consumption. In this context, we propose two techniques for reducing power dissipation of the reorder buffer (ROB) and the issue queue, respectively. The proposed techniques are orthogonal at the microarchitecture level, and thus they are also complementary and work together to improve each other.

The ROB keeps a copy of all in-flight instructions and speculative results until they retire, and thus the ROB becomes a complex multi-ported structure. The K-instruction processor, the early load retirement, the cherry, and the runahead execution have been proposed by Cristal [3], Kirman [4], Martinez [5], and Mutlu [8], respectively. The K-instruction processor [3] proposed the concept of early releasing ROB resources via checkpointing. For the case of a long-latency operation, the processor takes a checkpoint and releases the ROB resource early. This achieves high resource utilization without significantly increasing energy consumption, but it closely depends on multicheckpointing. The early load retirement [4] mechanism combines register checkpointing and early release of the load instruction. This allows instructions dependent on the long-latency load to execute sooner. However, this scheme is still based on checkpointing and it requires the checkpointing overhead. The cherry [5] is developed to recycle physical registers and load/store queue entries aggressively using combination of ROB and periodic checkpointing. The runahead execution [8] proposed to realize large instruction windows by reducing the penalties from long-latency memory instructions. When the instruction window is blocked by the long-latency instruction, the architectural state of the load instruction is checkpointed and its dependent instruction are nullified. Then, the instructions following the blocking operation are executed quickly. This scheme gives increased performance of larger instruction windows. However, these four approaches commonly make use of checkpointing and a large amount of storage spaces. Our proposed method, called small reorder buffer (SROB), is distinct from these above approaches in that we achieve early release without depending on any checkpointing. This feature gives us relatively good performance with a low power dissipation.

The issue queue is examined every cycle to choose ready instructions for simultaneous execution. The wake-up logic in the issue queue consists of a large number of comparators and they are used to compare each broadcast tag with every source operand tag. Therefore, the power and performance optimization of issue queue in dynamically scheduled processors requires a careful balance between reducing the power consumption and improving on its performance. To this end, the direct tag search (DTS), the N-use, and the IQ pointer — tag associative buffer (IQP-TAB) scheme have been proposed by Weiss [20], Canal [19], and Weinraub [10], respectively. The DTS [20] is the first proposal to limit power dissipation by avoiding associative tag search at instruction wake-up logic. It uses a RAM-based structure instead of a CAM for issue queue. The structure is indexed by result operand tag to perform the result forwarding. The drawback of the DTS approach is that only

Manuscript received January 18, 2008.
Manuscript revised April 14, 2008.

[†]The authors are with the Department of EECS, Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea.
a) E-mail: min@kaist.ac.kr, Corresponding author.

DOI: 10.1093/ietele/e91–c.9.1427

one instruction can be referenced from the direct tag search table. The N-use [19] scheme is the retrospection of the DTS. It improves the limitation of the DTS by using set-associative memory which is possible to store up to N instructions in N-use table. However, this architecture still make use of associative storage for N-use table and it requires the CAM buffer to resolve resource conflicts occurring from the RAM-based issue queue. This is because only less than N instructions can be stored in one issue queue slot. In N-use scheme, the aspect of power consumption is not investigated because the target of the approach is to reduce the complexity of the issue logic. The IQP-TAB [10] scheme is recently proposed to focus on power saving in issue window. It also replaces the CAM-based issue queue with a RAM, and uses two additional structures — a table of addresses to the issue queue and a small, fully associative buffer. The dependency relation up to one or two consumers for each producer instruction can be represented using RAM-based IQ and the IQ pointer (IQP), but the others still have to be processed in power-consuming CAM buffer. And it is necessary to make the size of the CAM buffer large enough to support those overflowed instructions. Moreover, each row in the CAM buffer contains only one consumer for a producer instruction. This means that several rows may be used for representing one dependency relations if a producer has a lot of consumer instructions. As a result, the front-end pipeline stalls due to the buffer full in issue queue.

In this paper, we introduce energy-efficient techniques for instruction window. First, we focus on the fact that many instructions waste ROB resources without doing any useful work during data dependency resolution. To reduce such wasteful usage of resource and energy, we introduce the novel concept of a small reorder buffer (SROB). The SROB executes only dependent instructions in program order and releases the instructions faster. This results in higher resource utilization and low power consumption. The power reduction stems from deferred allocation and early release. The deferred allocation technique inserts instructions into the SROB only after fulfilling the data dependency. The SROB releases instructions earlier immediately after the execution completes, because precise exception is trivial under in-order execution. Second, in order to deal with the power problem on issue queue, we focus on a well known fact that the vast majority of instruction dependency exists within a basic block. In practice, a basic block is comprised of about 6 instructions on average [22]. Based on these characteristics, we propose a bit-vector based tag translation unit (TTU). It resolves collisions of instruction dependency when an issue queue slot is already occupied by another instruction. The TTU replaces the complex associative tag matching operation in conventional CAM buffer by simple bit-vector checking operation. In this structure, a row is associated with each issue queue slot and divided into two groups of six bits. Each bit corresponds to the relative displacement from producer to consumer in issue queue. The bit-vector based TTU executes in parallel and consumes less power than the conventional architecture because of structured nature. Exploiting the above technique, we reduce power consumption by 24.45% on average over IQP-TAB scheme.

The rest of this paper is organized as follows. Section 2 presents a brief review of the existing approaches related to instruction issue logic. Section 3 describes our modified reorder buffer architecture, the SROB, and the concept of deferred allocation and early release. Section 4 provides the explanation of the DLT issue queue is illustrated. We evaluate its performance and power consumption in Sect. 5. Finally, we conclude by summarizing our results in Sect. 6.

2. Related Work

A wealth of work has been undertaken to design a low power instruction window architecture. We begin with categorizing and summarizing a number of related works for power reduction of the ROB and the issue queue.

Bell [1] and cristal [2] propose the concept of out-of-order commit. Instructions from a few ROB slots in Bell's approach are allowed to retire out-of-order under certain conditions. However, instructions are still retired only when they meet six necessary commit conditions such that they have completed and are guaranteed to retire safely. Cristal's proposal describes the out-of-order commit processor which increases the capacity of future processors by augmenting the number of in-flight instructions. It exploits a new checkpointing mechanism that is capable of keeping thousands of in-flight instructions at a practically constant cost.

Recently, three works takes checkpointing approach to deal with the scalability problems of processor resources. Martinez [5] propose a scheme to recycle physical registers and load/store queue entries aggresively. It makes use of a combination of ROB and periodic checkpointing to realize precise exceptions. Akkary [6] and Cristal [3] propose ROB-less semi ROB-less micro-architectures based on a multicheckpointing scheme. Our approach is distinct from these approaches in that we achieve high resource utilization and precise exception by early release of the ROB entries without any checkpoints.

The concept for runahead execution was first proposed by Dundas [7] to improve the data cache performance on in-order execution. More recently Mutlu [8] extends the concept for out-of-order execution processors. Mutlu proposes a runahead execution to realize large instruction windows by reducing the penalties from long-latency memory instructions. When the instruction window is blocked by the long-latency instruction, the architectural register state is checkpointed and retires early. By this way, the instructions following the blocking operation are executed quickly. This work is close to our early release scheme in SROB, but it is different in that we eliminate the use of the checkpointing.

Ponomarev [15], Folegnani [11], Buyuktosunoglu [13], and Jones [12] proposed mechanisms to dynamically adjust the sizes of the instruction window based on periodic sampling of their occupancies. They try to achieve significant

power savings with minimal impact on performance. They present an adaptive issue queue design by dynamically shutting down and re-enabling blocks of the issue queue. By shutting down unused blocks of the issue queue, they are able to proportionately reduce the energy dissipated. However, extra overhead exists when activating the previously disabled partitions of issue queue. Moreover, additional hardware and control logics for monitoring and sampling the utilization of the IQ are necessary.

In a slightly different context, Ehrhart [16] and Ernst [17] completely eliminate the CAM structures by proposing wakeup free scheduling. These schemes commonly utilize prediction to obtain instruction arrival time by compile-time and run-time information. They support a selective replay mechanism to resolve the mis-predicted instructions that execute too early. However, the prediction mechanism is subject to varying degrees of uncertainty and is not reliable for non-deterministic events, such as a cache miss.

Weinraub [10], Canal [19], and Chen [27] limit associative tag search at instruction wakeup to reduce power dissipation. These approaches use an associative mapping table, instead of CAM, for issue queue. The associative mapping table commonly requires resolving resource conflicts since only one particular tag can be stored in one issue queue slot. Moreover, it is difficult to keep the size of the mapping table as small as that of CAM due to inherent instruction dependency in a program. As a result, the CAM buffer overflow still results in large power consumption.

3. The Small Reorder Buffer

In general, the function of the reorder buffer (ROB) is to put the instructions back into the original program order after the instructions have finished execution possibly out of order. The ROB maintains an ordered list of the instructions and takes into account recovery and precise exception. Conventionally, the instructions are inserted into both the issue queue and the ROB. The instructions stay in the ROB until the instruction commits. As soon as the dependency of an instruction is fulfilled, the processor executes the instruction sequentially in program order. In addition to the ordinary ROB, we propose the concept of the small ROB (SROB), as depicted in Fig. 1. Figure 1 shows the overall pipeline architecture in which the colored components represent the modified (or newly added) parts in this work. The processor decodes fetched instructions (FETCH) and assigns physical registers to hold their results. This register renaming process (RR) maps the architected registers into a larger set of physical registers. Decoded instructions (DECODE) are inserted in both instruction window and reorder buffer (ROB) at dispatch time in program order. For load and store instructions, they are assigned to entries in load-store queues (LSQ). Instructions leave the instruction queue when they are issued, and free their reorder buffer entries when they commit. Reorder buffer holds the result of an instruction between the time the operation associated with the instruction completes and the time the instruction commits. The fuctional units (FU) can execute an operation of a certain type. The system retrieves the operands from register file (RF), and stores the operands into the register file. The stand-alone rename registers (SARR) are split register file to implement the rename buffers. In Fig. 1, each entry in the SROB has the same structure as the ordinary ROB. However, the SROB manages only dependent instructions, while an ordinary ROB processes the rest of the instructions such as control instructions, independent instructions, and load/store instructions. The execution of dependent instructions is serialized inherently by the true dependency. Most of these instructions will wait for a long time to resolve their data dependencies, even if we put the dependent instructions into the general ROB. Figure 2 shows the example of instruction allocation in the ROB and the SROB. Consequently, the instructions waiting in the ROB do not any useful work and severely affect the power consumption and the instruction level parallelism (ILP). This is because the ROB is a complex multi-ported structure and represents a significant source of power dissipation. Moreover, if the dependent instructions are in a long dependency chain, power and performance problem gets worse.

3.1 Deferred Allocation and Early Release

In order to resolve the power and performance problems, we prevent dependent instructions from moving through the ROB at dispatch time. The instructions wait for issue on the instruction queue, not on the ROB. After the instruc-

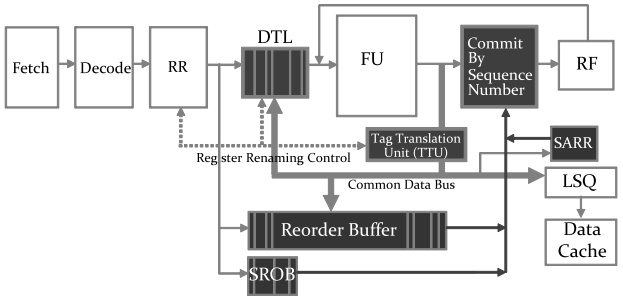

Fig. 1 DLT pipeline architecture.

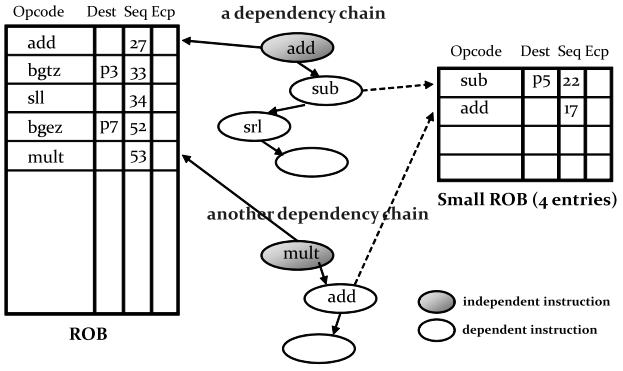

Fig. 2 Instruction allocation in reorder buffer.

tion dependency is fulfilled, the instructions can go to the SROB. As a result, one instruction of a dependency chain executes in the SROB at a time naturally as shown in Fig. 2. We call this the deferred allocation of the SROB. Moreover, the instructions in the SROB are released earlier and the result of the instruction is written into rename buffers immediately after the execution completes. Then, the result values in the rename buffer are written into the architectural register file at the commit state. Since the instructions in the SROB are executed in program order, we need not maintain the order of instructions and thus we have to take the results only. For implementation of the deferred allocation, we need to check whether an instruction is in a certain dependency chain or not. However, facilitating such a hardware checker causes complexity at the front-end. So, we take a straightforward approach to realize a simple instruction classification at the decoding stage. Our classifier checks only the operand availability of each instruction. If operands are available, the instruction is independent. Otherwise, the instruction is dependent and it thus goes to the SROB. This classification mechanism is very simple, yet able to capture in a uniform way all the dependency chains through a given microarchitectural execution of a program.

3.2 Mechanism for In-Order Commit

In conventional reorder buffer architecture, a processor only looks at the bottom of the centralized reorder buffer and it commits the instruction reaching the bottom of the reorder buffer to the architectural register. We have a slightly different commit mechanism based on sequence numbers which is similar to a timestamp scheme. The commit mechanism must take into account the following three different buffers: (1) stand-alone rename registers (SARR); (2) small reorder buffer (SROB); and (3) reorder buffer (ROB). The SARR is a split register file to implement the rename buffer. Although we also use the ROB in our architecture, the ROB only keeps track of a subset of instructions such as control instructions or independent instructions. Therefore, we need to separate the rename buffers from the ROB and facilitate the SARR to cover all types of instructions. Nevertheless, this is not severe limitation because recent processors except the Intel Pentium typically incorporate the stand-alone rename register scheme [28]. In our commit mechanism, the fetch unit tags each instruction with a sequence number that is unique for a processor. The number is increased by one for each instruction. The sequence number ensures that the instructions retire in order. This is necessary because the instructions are waiting for retirement from three different distributed storages in our architecture. Looking for an eligible instruction to commit is achieved by the min_seq scheduling logic. The min_seq scheduler determines the candidate instruction by comparing the sequence numbers among the three buffers. As a result, an instruction retires at commit stage only if it meets the condition that all lower numbered instructions are already committed. This strategy serializes the commit sequence among all in-flight instructions. In addition, the comparator that determines the minimum sequence number has compensation logic. This is because the sequence number register becomes zero when it overflows; we provide a compensation mechanism to handle the relative order of in-flight instructions correctly.

3.3 Enforcing Precise Exceptions

The SROB also provides precise exception. The precise exception occurs when an exception is raised and the processor state looks exactly as if the instructions were executed sequentially in strict program order. Since the ROB in our configuration manages only subsets of all instructions, providing precise exception only in the ROB is not enough. In order to deal with precise exception for all instructions, we use a straightforward strategy to enforce precise exception in the SROB. If an exception is detected, the SROB just holds the instruction and the status flags until all uncompleted instructions prior to the instruction are executed and retired. Then, the exception is processed and the operations are re-executed where necessary to process the exception. Due to the true data dependency, the instructions in the SROB are executed in order. The precise exception during in-order execution is trivial.

However, this straightforward approach is only proper to sequentially executed instructions such as in the SROB. Providing precise exception in conventional ROB is still necessary because the SROB in our configuration manages only subsets of all instructions. When an exception occurs in the ROB, retiring all instructions prior to the excepting instruction is the state of the art solution. After handling from the exception, the processor resumes executing instructions in the correct state.

4. The DLT Issue Window

In this section, we describe the design and the structure of our DLT issue window (DLT IW)[†]. To realize the DLT issue window, we use the Alpha 21264 architecture as the base platform. Figure 1 also shows the proposed architecture for the DLT-based microarchitecture. The fetch stage includes the I-cache, branch prediction and the instruction fetch queue. Instructions go through register rename unit before entering the issue window. Then, the instructions are inserted into a DLT IW slot at which the one of the source operand pointed. The instructions wait at the DLT IW until their source operands are ready. On each cycle, the result operand tags of instruction are broadcasted through the common data bus (CDB) to wake up consumer instructions. At that time, the issue window matches the tags against their unready source operands. The DLT IW wakes up the dependent instructions. They proceed with the register read, execution, and memory/writeback stages. The TTU snoops result operand tags passing by and translates them. In Fig. 3,

[†]The terms 'issue queue' and 'issue window' are used interchangably in this paper.

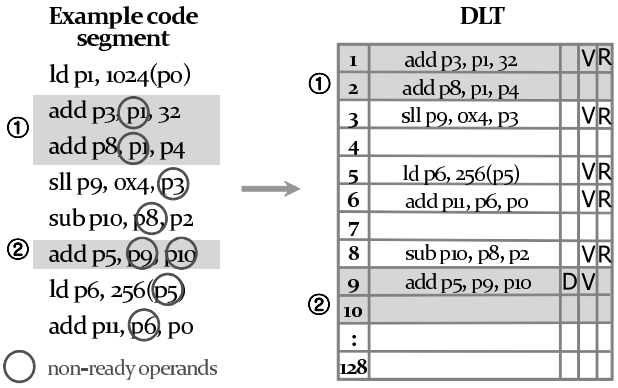

Fig. 3 Example code segment and its allocation into DLT issue window.

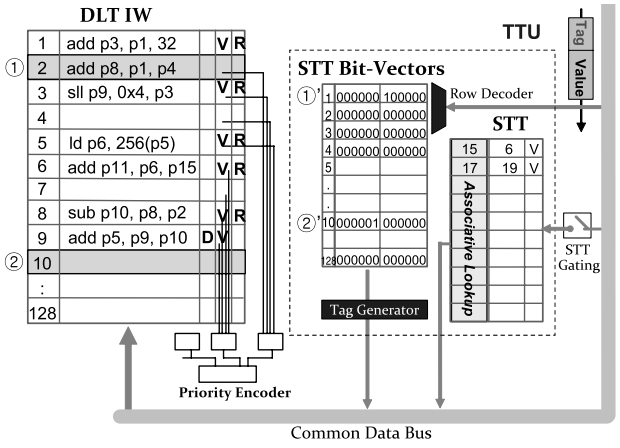

Fig. 4 DLT IW allocation.

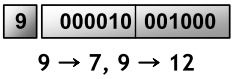

Fig. 5 A bit-vector structure.

we see an example of the instruction allocation in DLT IW for a sample code segment. In this example code segment, the red circled numbers represent the non-ready operands. We assume that the physical registers, p0, p4, and p2 are already available at the begining. When an instruction is dispatched, an entry is allocated in DLT IW depending on the non-ready source operand. For example, the second instruction in Fig. 3 goes to the first DLT slot because the operand p1 is a non-ready source operand. The fifth instruction is located to the eighth slot since the operand p8 is not available at the time. If the DLT IW slot is already occupied by another instruction, the DLT allocation scheme attempts to place the instruction for an available entry as near from the producer instruction as possible. For instance, the third instruction goes to the second DLT slot because another instruction is already allocated in the first slot. Specifically, we leverage the instruction to be placed, if possible, within six issue window slots of the desired location and the TTU maintains the location mapping. If placement is not available within six DLT IW slots, we put the instruction in any blank DLT slot and the TTU maintains the mapping. The TTU resolves collisions on DLT IW slot allocation by tag translation using the location mapping. Later, the result of the producer instruction comes from the functional unit and the result operand number is used directly as an index to look for the consumer instruction. The ① and ② in Fig. 3 show the cases when both operands are not available and when two consecutive instructions need the same operands. Since only one particular tag and its corresponding instruction can be stored in one DLT IW slot, we call this resource conflict or overflow in issue window. The handling of the resource conflict involves two specialized hardware structures and they are shown in Fig. 4. Figure 4 describes the organization of DLT IW and TTU. The DLT issue window (IW) makes use of three different bit flags to represent status information for each DLT entry: valid, ready, and double bits. A valid bit maintained for each instruction queue entry indicates whether the entry contains valid information or not. A ready bit represents whether all of source registers are available or not. If a ready bit is set, the instruction can be issued for the execution. A double bit indicates that an instruction needs two source operands. If both operands of an instruction are not ready, our DLT scheme does not replicate the same instruction into both queue entries (refer to case ② in Fig. 4). Instead, our DLT scheme just puts the information into only one of them, and sets the double bit. From then on, TTU takes over the burden of tag translation. For instance, tag 10 goes to slot 9 in addition to slot 10 when tag 10 appears in the CDB.

4.1 STT and STT Bit-Vectors

The TTU consists of snoopy tag translator (STT) bit-vectors. The STT consists of a small number of associatively-addressed retention latches for snooping the result on the CDB. It is designed to attract a certain flow from the CDB and look up the tag from the table. If a tag match occurs, the STT generates another result tag corresponding to the operand. For instance, when an instruction is followed by another instruction having the same source operand already dispatched in DLT, rename logic places the second instruction at any available entry in DLT as case in case ① in Fig. 4. Meanwhile, the STT records the mapping between the source and the true allocated index. With this tag translation mechanism, the complex associative tag matching operation is replaced by simple direct table lookup operation.

Another important structure in the TTU is the STT bit-vectors (STT-BV). The STT-BV execute in parallel and consume less power than STT because of their structured nature. When a result operand tag shows upon CDB, STT-BV is checked. If it is set, another result operand tag is generated by TTU. A Bit-Vector is 12 bits wide. It is associated with each DLT slot which is divided into two groups of 6 bits as shown in Fig. 5. The first six bits represent instruction dependency in reverse order. The second six bits represent it in forward order. The STT-BV width is determined to be

6 because the vast majority of instruction dependency exists within a basic block. In practice, a basic block is comprised of an average of 6 instructions [22]. Each bit corresponds to the relative displacement of the dependency. For instance, the first entry in STT-BV (refer to ①' in Fig. 4) has 000000:100000. This indicates the first instruction also needs the value of tag number 1. In case of ②', it means that the first previous instruction also requires the same result, tag number 10. Assuming the number of physical register tags is 32, the STT-BV can handle dependencies for up to 384 (32 X 12) entries. Unlike this case, the STT is able to handle only 32 entries. When allocating the DLT entry for an incoming consumer instruction, the DLT scheme attempts to place the instruction for an available entry as close to the producer instruction as possible. Specifically, the instruction is placed within six DLT slots on either side. This is because a higher hit ratio on STT-BV is better than more hits on STT in terms on power. If a location is not available within six DLT IW slots, we put the instruction in any blank slot and insert mapping into the STT instead.

To support tag translation, we need additional structures: a tag generator, a row decoder, and a STT-gating logic. The tag generator accesses STT-BV to check a set bit and generates an additional result operand tag. The average instruction per cycle (IPC) on 4-way issue/commit configuration is about 2.07 in our experiment and more or less in the other experiments. This means that the CDB bandwidth is under-utilized for a significant portion of time, unless there is a high fetch/decode/issue rate. Thus, the tag generator can generate additional two or more tags on average per cycle. Figure 6 illustrates that the tag generator reads a tag passing through the CDB and generates a new tag in the available lane of the CDB. The row decoder selects one line of the STT-BV table depending on the result operand tag value. The STT-gating logic is attached to control the data flow between the CDB and the STT. The STT and the STT-BV maintain the exclusion property for their stored entries. Thus, tag search matches only one of them. Since the STT consumes more power than the STT-BV, the STT-gating logic limits access to the STT unless a match will actually occur. To do so, the logic has a set of select bits corresponding to each physical register tag. When a tag mapping entry is added, the select bit is set to allow tag search. When the tag mapping is used, the corresponding select bit is cleared. Therefore, the tag search is limited by the existence of a tag mapping entry in STT.

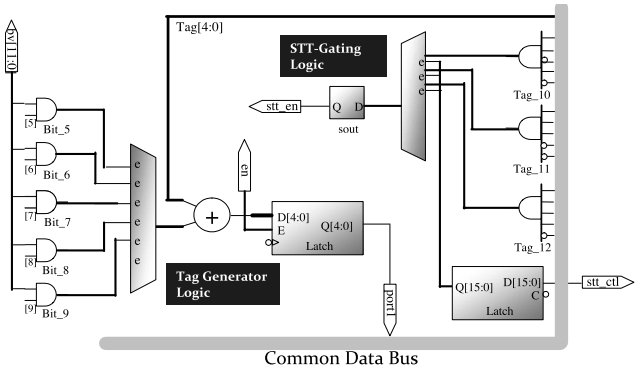

Fig. 6 Detailed structure of tag generator and STT-gating logic.

5. Experimental Result

All tests and evaluations were performed with programs from the SPEC2000 CPU benchmark suite on Sim-Panalyzer [6]. The Sim-Panalyzer is a cycle accurate and architecture level power simulator which is built on the SimpleScalar simulator. The total power dissipation in a CMOS circuit can be expressed as the sum of three main components.

$$Total\ Power = VI_{leak} + CV^2 f + ACV^2 f$$

In order to calculate power using the above equation, the Sim-Panalyzer make use of both analytical and experimental models. We need to count how much switching has happened for each module during simulation. Then, we multiply the unit power for a micro-architectural block by the number of accesses. The Sim-Panalyzer lumps the issue queue, the reorder buffer, and the physical register file into a register update unit (RUU). In order to better model the power consumption of contemporary microprocessor architecture, we split the RUU into the reorder buffer and the issue queues. To evaluate the performance of the DLT architecture, we use the Alpha 21264 architecture as the baseline platform. The Alpha is an out-of-order-issue microprocessor that can fetch and execute up to four instructions per cycle. It also features dynamic scheduling and speculative execution to maximize performance. The Alpha pipeline contains four integer execution units. The two of the integer execution units can perform memory address calculations for load and store operations. The 21264 pipeline also contains two floating-point execution units to perform add, divide, square root, and multiply functions. The 21264 pipeline has 7 stages which consist of instruction fetch, branch prediction, register renaming, instruction issue, register access, execution and writeback. The architecture parameters used

Table 1 Simulated architecture parameters.

	SROB	DLT	CAM-based IQ	IQP-TAB
fetch,decode, issue,commit	4, 4, 4, 4			
rn:size	32			
iq:size	32			
stt:size	-	16	-	-
iqp:size	-	-	-	32
tab:size	-	-	-	16
srob:size	4	-	-	16
ialu, imult, fpalu, fpmult	8, 2, 2, 2			
Branch Prediction				
bpredictor	2lev. Bimodal, 2K,			
btb	128, 4-way			
return address stack	16 entry			
mispred penalty	2 cycles			
Memory Hierarchy				
L1 D-cache, L1 I-cache	1024 sets, 32B blocks, 2-way, LRU			
L2 cache	2048, 64B blocks, 4-way, LRU			
memory latency	20 cycles			

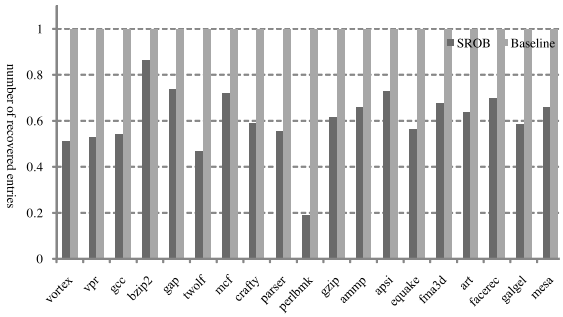

Fig. 7 The number of recovered entries in SROB and Baseline.

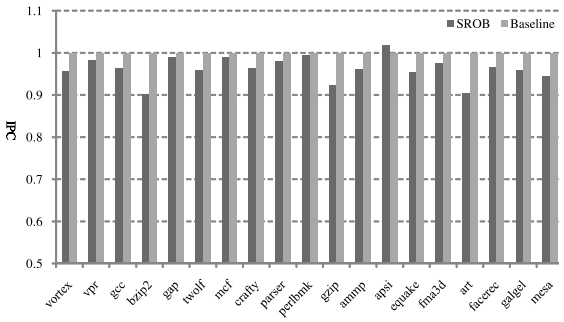

Fig. 8 IPC achieved with SROB and Baseline.

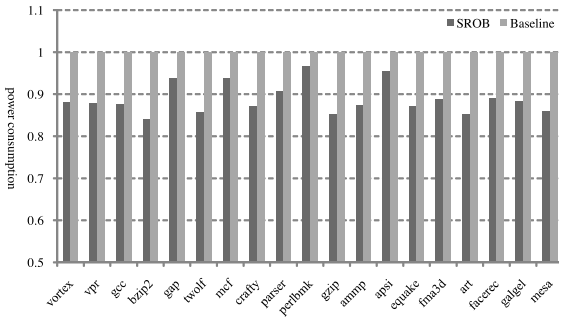

Fig. 9 The power dissipation in SROB and Baseline.

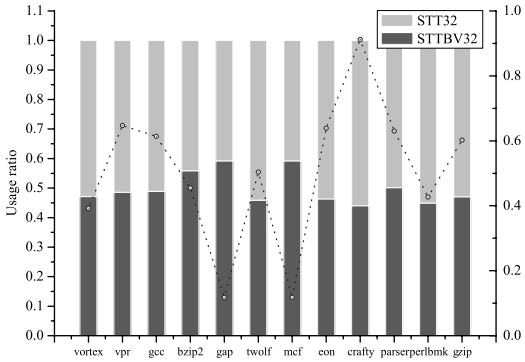

Fig. 10 STT and STT-BV utilization and the total number of conflicts on IW size 32.

in our Sim-Panalyzer simulations are listed in Table 1. The parameters such as iw:size, stt:size, and rn:size in Table 1 are only available in DLT architecture. The iw:size means the entry size of issue queue. The stt:size is the queue entry size in STT. If the stt:size is large, more power is consumed. If the size is too small, a full buffer frequently occurs, resulting in performance degradation. In this experiment, we maintain the size of STT as half the issue window (IW) size: 16 entries for STT for a DLT of size 32, and 32 entries for STT for a DLT of size 64. The rn:size is for adjusting the range of register renaming. It indicates how many physical registers are mapped to logical register names. Without register renaming, running a binary executable compiled for 32 registers on 64 register machine will repetitively make use of first 32 registers only. This is because the renamed register tag is used as an index to lookup the IW in DLT architecture. This technique avoids recompilation overhead when a binary executes on different architecture in terms of pyhsical register size. The srob:size configures the size of the SROB buffer. The reason we set this parameter as 4 is to make the SROB size equal to the issue/commit bandwidth. If the size is more or less than the bandwidth, it may result in performance bottleneck or resource waste. The iqp:size and tab:size are only applicable in the IQP-TAB scheme.

Figure 7 represents the number of entries to be recovered when a mis-prediction or an exception occurs. Our early release technique reduces 39.4% of the wrong-path instructions which are remaining in reorder buffer. The reduced entries contribute to power efficiency because recovery consumes power to squash the unnecessary entries for a wrong path instruction. However, the results of bzip, gap, apsi, and perlbmk have minimal impact to power dissipation because the absolute differences are not significant in contrast to the proportional differences shown in Fig. 7.

Figure 8 shows an average of IPC attained by SpecFP and SpecInt applications in simulations. The results are normalized to the baseline values. The performance degradation is due to the SROB contention. The exception is that apsi delivers even better performance while maintaining an effective power consumption level (4.9% less than the baseline power).

Figure 9 represents the evaluated power dissipation. The SROB method achieved power reduction to 11.2% of baseline power. The power reduction stems from deferred allocation and early release in the SROB. The power savings come with a performance penalty of only 3.7% on average. We note that power saving of the 11.2% is not total system savings, but a portion of the total system savings. The savings only applies to the power saving in the ROB unit. However, the overall power savings in the perspective of total system are not negligible. This is because the ROB consumes the most significant amount of energy among all structures. In fact, it takes 27.1% of total system power dissipation. At the same time, we achieved power reduction of the ROB unit to 11.2%. Therefore, the overall power savings in the perspective of total system are 3.04%.

Figures 10 and 11 show the STT-BV/STT occupancy ratio and the total number of conflicts in DLT IW. The occupancy ratio is cumulatively defined as the proportion of the

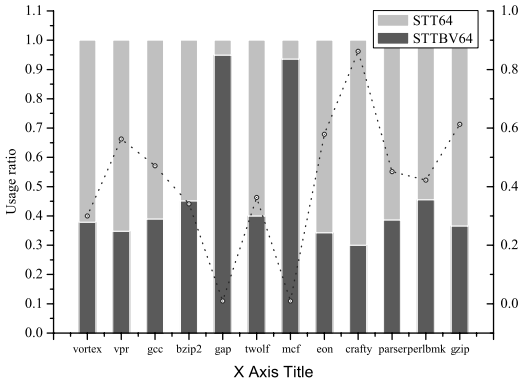

Fig. 11 STT and STT-BV utilization and the total number of conflicts on IW size 64.

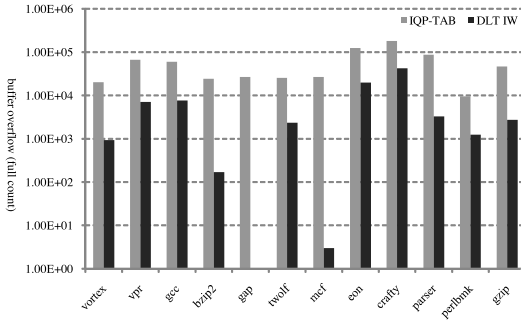

Fig. 12 Buffer overflow count on IW size 32.

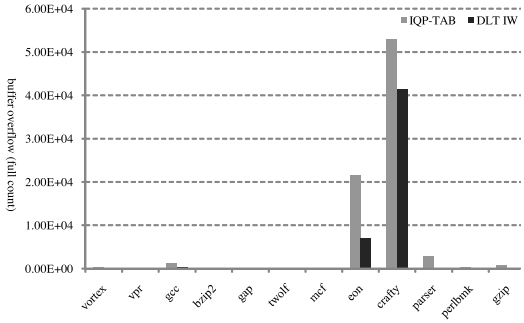

Fig. 13 Buffer overflow count on IW size 64.

sum of 2STT-BV Occupancy over the sum of STT Occupancy. The conflict in IW directly affects power consumption. This is because it causes the IW allocation scheme to resolve the conflicts through TTU and the resolution involves a power consuming operation such as STT access. In our approach, many IW slot conflicts are processed in STT-BV rather than STT; 49.77% on average for DLT size 32, and 47.53% on average for DLT size 64. A high hit ratio on STT-BV is better than high hit ratio on STT because of it gives rise to a power reduction opportunity.

Figures 12 and 13 show the effectiveness of using STT-BV. The STT overflow count is total number of event that the STT is full. This factor directly affects IPC, resulting in application performance degradation. This is because when

Table 2 Power dissipation and access delay of additional logic circuits.

		STT-gating	Row Decoder	Tag Generator	STT-BV line
Power Dissipation	Dynamic	1.0560mW	366.3912mW	5.2750mW	1.2551mW
	Leakage	4.9051nW	590.3000nW	36.5359nW	5.2789nW
Access delay		0.71ns	0.53ns	1.61ns	0.12ns
Area		911.43362	262.7856	4487.3137	974.6352

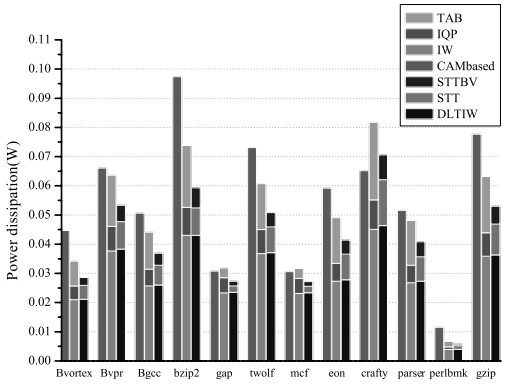

Fig. 14 Power consumption on IW size 32.

the STT is full, the front-end† in a microprocessor stops executing until it is resolved.

We implemented the additional logic structures above using Verilog HDL. These designs have been synthesized with the Synopsys Design Compiler [7] targeted towards a 0.18 micron TSMC library. For the sake of simplicity, logic structures are designed on the assumption of single port I/O. For multi-port I/O, we designed another version of the tag generator to support for up to 4 ports. It consumes 9.86 mW of dynamic power and 36.16 nW of leakage power. Similarly, the extension to multi-port design for the other structures can be achieved within a reasonable range of power consumption. As shown in Table 2, we measured power dissipation, area estimation, and access delay. The result shows that the STT-BV consumes 76.1% less power than the STT. This is because the STT is made up of content addressable memory (CAM), whereas the STT-BV is composed of RAM. For the STT-BV, the total power consumption are estimated as follows: 1.0560 mW (STT-gating) + 366.3912 mW (Row Decoder) + 5.2750 mW (Tag Generator) + 1.2551 mW (STT-BV line). For the STT, the STT has to lookup all entries for every operation due to its inherent structure. Thus, the total power dissipation are at least 1.2551 mW *32, since the STT must access all 32 STT-BV cells at every time. This gives under-estimated results in terms of the STT power consumption because the real CAM structure is more complex than the STT-BV structure.

Figures 14 and 15 show the evaluated power dissipation

†Front-end denotes the mechanisms responsible for supplying instructions to the execution units (back-end). The front-end includes fetch unit, register renaming unit, branch predictor, and other support structures.

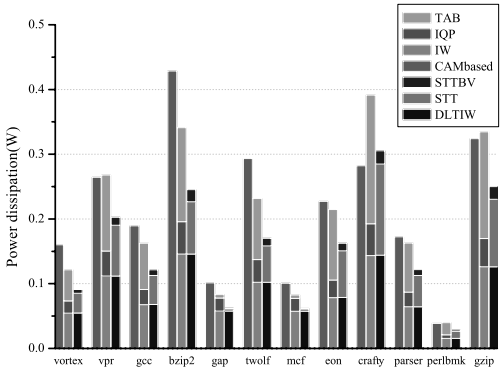

Fig. 15 Power consumption on IW size 64.

on IW size 32 and 64, respectively. In the graph, our scheme is 24.45% more effective than IQP-TAB. This is because our approach considerably reduces the amount of buffer overflow. The power consumption is primarily affected by the number of buffer overflow. However, in our approach, the majority of tag translation is processed on STT-BV, instead of STT. Thus, the amount of buffer overflow is reduced and power dissipation is low. This trend continues as the number of DLT entries increases. Actually, the same experiment on IW size 64 saved even more power (39.5%).

6. Conclusion

At the microarchitecture level, we proposed two orthogonal techniques for reducing the power dissipation on instruction window. The small reorder buffer (SROB) reduces power dissipation by deferred allocation and early release. These two techniques result in higher resource utilization and low power consumption. Therefore, up to 3.04% of power saving comes with an average of only 3.7% performance penalty. In current version of implementation, we limited the role of the SROB to process only dependent instructions. The power saving will be much increased if the SROB approach is extended to all types of instructions as future work. Even though there is a little performance penalty, our SROB technique for reducing the power dissipation is still meaningful, especially on the embedded computing. In the embedded environment, the energy saving is the most critical due to the limited battery capacity.

In addition, we proposed a power-efficient technique for resolving the dependency relation of one producer to multiple consumers. The tag translation scheme is for resolving the instruction dependency by bit-vector based overflow handling. This structure significantly increases total capacity to handle the data dependency. In fact, the STT-BV was set to handle the dependency up to 384 (32 X 12) entries in our environment. Thus, the bit-vector structure is physically small, but has very large capacity to represent dependency relation. Consequently, it decreases a large amount of power dissipation by utilizing the small and power efficient bit-vector structure. Experimental results achieved using our proposed design reduced power consumption by an average of 24.45%.

Acknowledgements

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute for Information Technology Advancement) (IITA-2008-C1090-0801-0045).

References

[1] G. Bell and M. Lipasti, "Deconstructing commit," Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2004.
[2] A. Cristal, D. Ortega, J. Llosa, and M. Valero, "Out-of-order commit processors," Proc. IEEE International Symposium on High-Performance Computer Architecture (HPCA), Feb. 2004.
[3] A. Cristal, O. Santana, F. Cazorla, M. Galluzzi, T. Ramirez, M. Pericas, and M. Valero, "Kilo-instruction processors: Overcoming the memory wall," IEEE Micro, vol.25, no.3, pp.48–57, 2005.
[4] N. Kirman, M. Kirman, M. Chaudhuri, and J. Martinez, "Checkpointed early load retirement," Proc. International Symposium on High-Performance Computer Architecture (HPCA), 2005.
[5] J. Martinez, J. Renau, M. Huang, M. Prvulovic, and J. Torrellas, "Cherry: Checkpointed early resource recycling in out-of-order microprocessors," Proc. IEEE International Symposium on Microarchitecture (MICRO), 2002.
[6] H. Akkary, R. Rajwar, and S. Srinivasan, "Checkpoint processing and recovery: Towards scalable large instruction window processors," Proc. IEEE International Symposium on Microarchitecture, Dec. 2003.
[7] J. Dundas and T. Mudge, "Improving data cache performance by pre-executing instructions under a cache miss," Proc. ACM International Conference on Supercomputing (ICS), July 1997.
[8] O. Mutlu, J. Stark, C. Wilkerson, and Y.N. Patt, "Runahead execution: An alternative to very large instruction windows for out-of-order processors," Proc. IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp.129–140, Feb. 2003.
[9] R. Kessler, "The alpha 21264 microprocessor," IEEE Micro, vol.19, no.2, pp.24–36, 1999.
[10] Y. Weinraub and S. Weiss, "Power-aware out-of-order issue logic in high-performance microprocessors," Microprocessors and Microsystems, Elsevier Science, vol.30, no.7, pp.457–467, 2006.
[11] D. Folegnani and A. Gonzalez, "Energy-effective issue logic," Proc. IEEE International Symposium on Computer Architecture (ISCA), 2001.
[12] T. Jones, M. O'Boyle, J. Abella, and A. Gonzalez, "Software directed issue queue power reduction," Proc. International Symposium on High-Performance Computer Architecture (HPCA), 2005.
[13] A. Buyuktosunoglu, D. Albonesi, S. Schuster, D. Brooks, P. Bose, and P. Cook, "A circuit level implementation of an adaptive issue queue for power-aware microprocessors," Proc. ACM Greate Lakes Symposium on VLSI (GLSVLSI), 2001.
[14] T. Huang, L. Wills, R. Melton, and C. Alford, "Predicting communication protocol performance on superscalar architectures using instruction dependency," Perform. Eval., Elsevier Science, vol.63, pp.939–955, 2006.
[15] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources," Proc. IEEE International Symposium on Microarchitecture (MICRO), Dec. 2001.
[16] T. Ehrhart, "Reducing the scheduling critical cycle using wakeup prediction," Proc. IEEE International Symposium on High-

Performance Computer Architecture (HPCA), 2004.
[17] D. Ernst, "Cyclone: A low-complexity broadcast-free dynamic instruction scheduler," Proc. IEEE International Symposium on Computer Architecture (ISCA), 2003.
[18] P. Michaud and A. Seznec, "Data-flow prescheduling for large instruction windows in out-of-order processors," Proc. 7th International Symposium of High-Performance Computer Architecture (HPCA), Jan. 2001.
[19] R. Canal, "Reducing the complexity of the issue logic," Proc. ACM International Conference of Supercomputing (ICS), 2001.
[20] S. Weiss and J. Smith, "Instruction issue logic in pipelined supercomputers," IEEE Trans. Comput., vol.39, no.3, pp.110–118, 1990.
[21] T. Sato, Y. Nakamura, and I. Arita, "Revisiting direct tag search algorithm on superscalar processors," Proc. Workshop Complexity-Effective Design, ISCA, 2001.
[22] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, 4th ed., Morgan Kauffman, San Francisco, CA, 2006.
[23] SimpleScalar v2.0.2, http://www.eecs.umich.edu/panalyzer/
[24] SPEC CPU2000, http://www.spec.org/cpu/
[25] M. Powell, "Gated-Vdd," Proc. IEEE International Symposium of Low Power Electronic Devices (ISLPED), 2000.
[26] Synopsys Design Compiler 2007, http://www.synopsys.com/partners/tapin/sdc.html
[27] C. Chen and K. Hsiao, "Scalable dynamic instruction scheduler through wake-up spatial locality," IEEE Trans. Comput., vol.56, no.11, pp.1534–1548, Nov. 2007.
[28] D. Sima, "The design space of register renaming techniques," IEEE Micro, vol.20, no.5, pp.70–83, 2000.

Min Choi received the B.S. degree in Computer Science from Kwangwoon University, Korea, in 2001, and the M.S. degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 2003. He has been a Ph.D. student in KAIST since 2003. His current research areas include distributed/parallel system, computer architecture, and embedded systems.

Seungryoul Maeng received the B.S. degree in Electronics Engineering from Seoul National University (SNU), Korea, in 1977, and the M.S. and Ph.D. degrees in Computer Science from the Korea Advanced Institute of Science and Technology (KAIST), in 1979 and 1984, respectively. Since 1984 he has been a faculty member of Department of Computer Science of the Korea Advanced Institute of Science and Technology. From 1988 to 1989, he was with the University of Pennsylvania as a visiting scholar. His research interests include microarchitecture, parallel computer architecture, cluster computing, and embedded systems.