Direct Instruction Wakeup for Out-Of-Order Processors

Marco A. Ramírez[1, 4], Adrian Cristal[1], Alexander V. Veidenbaum [2],Luis Villa [3], Mateo Valero [1]

[1] Computer Architecture Department, U.P.C., Barcelona Spain,
[2] dept. of Computer Science, University of California Irvine, USA,
[3] Mexican Petroleum Institute, Mexico. [4] National Polytechnic Institute, México.

e-mails: {mramirez, adrian, mateo}@ac.upc.es, alexv@ics.uci.edu, lvilla@imp.mx

Abstract— **Instruction queues consume a significant amount of power in high-performance processors, primarily due to instruction wakeup logic access to the queue structures. The wakeup logic delay is also a critical timing parameter. This paper proposes a new queue organization using a small number of successor pointers plus a small number of dynamically allocated full successor bit vectors for cases with a larger number of successors. The details of the new organization are described and it is shown to achieve the performance of CAM-based or full dependency matrix organizations using just one pointer per instruction plus eight full bit vectors. Only two full bit vectors are needed when two successor pointers are stored per instruction. Finally, a design and pre-layout of all critical structures in 70nm technology was performed for the proposed organization as well as for a CAM-based baseline. The new design is shown to use 1/2 to 1/5th of the baseline instruction queue power, depending on queue size. It is also shown to use significantly less power than the full dependency matrix based design.**

***Index Terms*— CAM, Direct Wakeup, Issue Queue, Low-Power, Out-of-Order Processors, Wakeup Instructions.**

I. INTRODUCTION

Out-of-order processors issue instructions even before their source operands are available. The instructions are typically entered into an Instruction Queue (IQ) where they wait for their operands. An instruction is dispatched to its execution unit when its operands are ready. This is accomplished using wakeup and select logic.

The wakeup logic is responsible for detecting when an instruction operand is ready. An instruction is considered for execution when all of its operands are ready.

This work was supported in part by the Ministry of Science and Technology of Spain under contract TIN2004-07739-C02-01 and HiPEAC, the European Network of Excellence on High Performance Architectures and Compilers, the program of scholarships ANUIES-SUPERA and COTEPABE-IPN from Mexico and by the National Science Foundation in the U.S.A under grant NSF CCR-0311738.

M. A. Ramírez S. and Luis Villa are with the Research Center for Computing from National Polytechnic Institute, Mexico, phone: +52 55-57296000 Ext.56519; e-mail: mars@cic.ipn.mx, mramirez@ac.upc.es).

The select logic chooses a subset of instructions flagged by the wakeup logic for execution. The operands may come from a register file or from a previously issued instruction via bypass logic. For instructions with a 1-cycle execution latency, wakeup and select logic have to have a latency of one cycle to avoid IPC loss.

Two types of instruction wakeup logic are typically discussed: a CAM-based IQ logic and a dependency-matrix based logic. The latter was described in an Intel patent [25] and [11].while the former was used in the MIPS R10K processor [21]. Neither one of these approaches is scalable with respect to instruction queues size, in particular for sizes being discussed for future processors. The approach proposed here aims to overcome this limitation and allow large IQ implementations while maintaining a 1-cycle wakeup-select cycle. In addition, it is also more energy efficient.

Consider a CAM-based wakeup logic implementation. An out-of-order processor inserts an instruction in the instruction queue after first decoding and renaming it. At that time it checks if the source register operands are ready and sets up CAM register tags and Ready flags for each source operand. Each completing (or selected) instruction broadcasts its destination register tag to the CAMs, which set Ready flags for all instructions that are waiting on this result. An Instruction Ready flag is set for an instruction when all of its source operands are ready. A vector of Instruction_Ready bits is the input to the select logic. The speed and scalability of this approach are limited by the CAM size, the number of ports, and the size of the register tag (number of registers).

The IQ CAM structure has to be multi-ported since multiple instructions complete each cycle. As the size and the number of ports of the IQ grow, this structure becomes unimplementable. For example, an 8-issue processor requires an instruction queue CAM with 16 comparison ports and 8 write ports for each source operand.

The Dependency-matrix approach uses a resource bit vector

for each instruction that specifies which of the processor resources need to be available for the instruction to become ready. Individual bits in the resource vector correspond to functional units and instruction(s) producing source operands for the instruction in this vector. An instruction is ready when all the resources it requires are available. The scalability of this type of wakeup logic is limited by the number of entries in the Dependency matrix, which is proportional to the number of instructions in the IQ.

There have been several proposals for organizing the wakeup logic using pointers rather than using either of the approaches described above. This approach stores a pointer(s) to dependent instruction(s) with each instruction. The wakeup logic uses such a pointer to directly wakeup dependent instruction(s). This approach has it own difficulties and scalability limitations. For instance, the number of successors may be very large. Also, branch misprediction recovery becomes more complex since cancelled instructions are pointed to by valid instructions in the IQ. This requires a pointer cleanup.

The pointer-based approach proposed in this paper has two main advantages. Its design allows a simple pointer cleanup mechanism. In addition, it utilizes the knowledge of successor distribution to minimize storage required for pointers and to speedup multiple successor wakeup. The new mechanism combines the best parts of the pointer and full dependency matrix approaches. It provides space for one or two pointers in each instruction IQ entry. This is based on previously observed predominance of instructions with very few successors. For all other cases a small number of full dependency vectors is available. This almost completely eliminates stalls for instructions with more than 2 successors but requires a lot fewer resources than a full dependency matrix.

The new approach also provides a mechanism for fast pointer update on branch misprediction. The complexity of this update is one of the main difficulties encountered by previous approaches. This is accomplished by check-pointing a small amount of additional information on each conditional branch.

One of the main goals of the approach proposed in this paper is power minimization. Accurate power evaluation is difficult and requires hardware design and layout to be performed first. This paper presents the results of such an evaluation, where the design and pre-layout of the major IQ structures have been performed for a 70nm CMOS process. These structures include CAMs, RAMs, latches, etc as well as the wires connecting them. Based on these designs and their layout SPICE simulations were performed to evaluate the power consumption in the instructions queue.

Paper is organized as follows. First, the direct wakeup mechanism and microarchitecture to support it are described. Next, simulation results for a 4-issue processor modeled after COMPAQ Alpha 21264 processor are presented. The results of power consumption in the wakeup mechanism based on Spice simulations are presented next and compared with a CAM-based and full dependency matrix mechanism. Finally, related work is presented in the last section.

II. DIRECT INSTRUCTION WAKEUP

Direct instruction wakeup refers to a mechanism that uses successor pointer(s) for wakeup, where the successor pointer points to an IQ location. This has been discussed in the literature; however the existing proposals have difficulties dealing with multiple successors and branch misprediction recovery. The approach proposed here solves both of these problems. The key to the solution is the use of a physical register tag as a unique identifier of an instruction. This simplifies successor detection and handling as well as misprediction recovery.

The goal of this approach is to use RAM rather than CAM in the IQ design and to complete successor processing in one half of the clock cycle so that the Ready bit vector can be presented to the select logic. It is assumed that the Ready bits are latched at the end of first phase of the clock. As will be seen below, this requires a RAM access and a pointer decode to be completed. The RAM does not require an address decoder since it is driven by the select logic bit vector output. The overall design of the instruction queue using a Mapping Table (MT) and Multiple Wake-up Table (MWT) to keep track of successor information is shown in Figure 1.

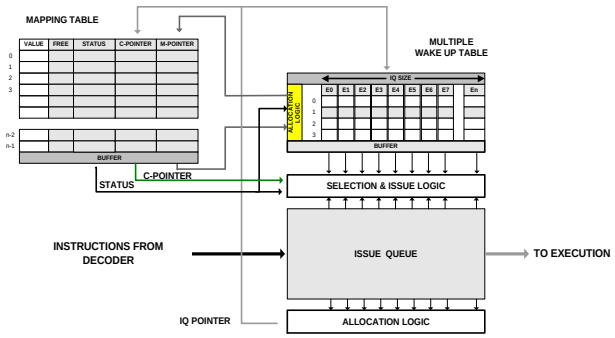

Figure 1. Direct Wakeup Mechanism (1C-Pointer , 4-Entry MWT)

A. *The Mapping and Multiple Wake-up Tables*

The mapping table (MT) contains dependent instruction information. This information is added to the MT as each new instruction is placed into the Instruction Queue (IQ) after decoding and renaming. MT entries for each instruction that produces source operands for the new instruction may need to be updated with successor information. The MT update consists of writing the IQ-Pointer of the new instruction in locations indexed by its two source operand register tags (src1 and src2).

It is assumed that an instruction is uniquely identified by its destination register, with the IQ size assumed to be less than or equal to the physical register file size. For simplicity the rest of the paper assumes the two are actually equal and the MT is of the same size as the physical register file. The MT has the following fields in each entry (assuming one dependent-instruction pointer):

Status:
00–No dependents,
01–One dependent,
10–Multiple dependents,
11–Completed (ready).
C-Pointer: A dependent instruction pointer.
M-Pointer: A pointer to a Multiple Wake-up Table entry identifying additional successors.

The C-Pointer is used for the first detected dependent instruction. The M-pointer is used when the Status field value is equal or greater than 10 indicating more than one dependent.

The MT hardware configuration necessary for a 4-way processor is a RAM with eight write ports and four read ports. The status fields are independent state-machines with eight parallel inputs that can change the state on each access to a physical register. The Multiple Wake-up Table has eight 1-bit write ports and four *n*-bit read ports, where *n* is the number of entries in the queue. The number of entries in the MWT is less than or equal to *n*. Each entry is *n* bits long, one bit for each instruction in the queue.

Note that while the initial processing of each instruction may be complex it does not need to be performed in one cycle. The front-end takes multiple cycles to enter an instruction in the instruction window and the MT/MWT processing can be performed during this time.

III. Mechanism Operation

A. The Instruction Queue

Each instruction queue RAM entry contains a 2-bit Ready counter used in wakeup to indicate when the operands became available. The counter is appropriately initialized for single-operand, ready at issue operands (source register status of 11), and other special cases. The counter inputs come from the dependent pointers in the MT and MWT tables.

The IQ also contains information necessary to read the register file and issue the instruction to the functional units: Opcode, Operand1 register tag, Operand2 Register tag, Destination register tag, etc. These are used after the wakeup-select cycle.

B. Direct Instruction Wakeup

The wakeup mechanism proposed in this paper operates as follows. After an instruction is issued, both its IQ Ready counter and its MT entries are setup as described above. The MT entry and possibly an allocated MWT entry are updated as its dependent instructions are added to the IQ. Instruction issue stalls if an MWT entry is needed but is unavailable.

Consider the following code segment (with physical register numbers):

```
0  ADD   $2,$7, $8        ; $2 <= $7   + $8
1  SUB   $4,$12, $2       ; $4 <= $12  -  $2
2  OR    $6,$9, $2        ; $6 <= $9   V $2
```

Let us track the changes to the MT entry for the ADD. Assume that the ADD is just added to the IQ. At this time its Status field 00-no dependents. When SUB instruction is added to the IQ, the status field of MT entries are changed for instructions producing registers $2 and $12 need to be changed. The Status field in the MT entry for the ADD is changed to 01(one dependent) and the C-pointer field is updated to point to SUB's IQ entry. Next, when OR is added to the IQ, the status field in the MT entry for the ADD is changed again, to 10 (multiple dependents). The C-pointer field in the ADD's entry was already allocated to the SUB, so for the 2nd successor (the OR) an MWT entry is allocated. The location of the MWT entry is stored in the M-pointer. One bit of the MWT entry which corresponds to the OR's position in the IQ, is set to 1.

When an instruction is selected to issue its successors are processed by the wakeup logic. The C-Pointer is decoded and ORed with an MWT entry if the latter is present. The resulting bit for each IQ position is an input to the Ready counter. The counter decrements and the result is available at the start of 2nd phase of the clock. A counter with a value of zero indicates to the select logic that the instruction is ready for issue. (The counter is used only to simplify this explanation; a faster logic can be designed to perform this function but is beyond the scope of this paper).

C. Branch Misprediction Recovery

On a branch misprediction all instructions allocated after the branch are cancelled in the queue. To recover the successor information the Free and status bits of the mapping table MT are check-pointed on each conditional branch, similarly to the register map. Misprediction recovery restores status bits. The new status determines if the C- and M-Pointers are valid. The C-Pointer is completely recovered at this point. The MWT entry pointed to by the M-Pointer still needs fixing if the status is 10.

The MWT entry may contain successors that have been canceled. The new free bits vector of the instruction queue indicates which instruction queue entries are valid at this point. The Free bit vector of the queue is ANDed into every MWT entry invalidating any cancelled successors. It is possible that an entry in the MT change its status from 10 to 01, in this case

the MWTentry pointed by M-Pointer field will have a value of zero after the recovery process. Such an entry in the MWT is marked as free.

IV. Results and Analysis

The performance of the direct wakeup mechanism was evaluated via simulation. The power consumption was evaluated with Spice after a design and pre-layout of the main structures was performed for a 70nm process. The microarchitecture of the Direct Wakeup scheme was evaluated using a modified SimpleScalar 3.0 Tools.

The simulator was modified to model separate integer, floating point and load/store queues, and was configured to emulate the Alpha 21264 organization (albeit with different parameters). The processor configuration is shown in Table 1.

TABLE I
PROCESSOR CONFIGURATION

Element	Configuration
Reorder Buffer	256 entries
Load/Store Queue	64 entries
Integer Queue	32-64 entries
Floating Point queue	32-64 entries
Fetch/decode/commit width	4/4/4
Functional units	4 integer/address ALU, 2 integer mult/div, 4 fp adders, 2 mult/div and 2 memory ports.
Branch predictor	16 K-entry GShare
Branch penalty	8 cycles
L1 Data cache	32 KB, 4 way, 32 byte/line, 1 cycles
L1 Instruction cache	32 KB, 4 way, 32 byte/line, 1 cycles
L2 Unified cache	512 KB, 4 way, 64 byte/line, 10 cycles
TLB	64 entries, 4 way, 8KB page, 30 cycles
Memory	100 cycles
Integer Register file	128 Physical Registers
FP Register file	128 Physical Registers

The workload used was the SPEC2000 benchmark suite compiled for the Alpha. A dynamic sequence of instructions representing its behavior was chosen for each benchmark and 200M committed instructions were simulated, with statistics gathered after a 100M-instruction "warm-up" period.

The number of dependent-instruction pointers and the size of MWT determine required hardware resources. First, let us study the distribution of successor instructions in the selected benchmarks.

A. Number of successors

The multiple-wake-up table (MWT) is a memory of M rows by E columns, where E is the size of the instruction queue and M≤E. The required size of the MWT can be estimated by studying the distribution of the number of instruction's successors.

As can be seen in Figures 2 and 3 below, on average nearly 18% of instructions in SPEC2000 have two or more successors. In some cases the average is as high as 38% for SPECint and 60% for SPECfp benchmarks, respectively.

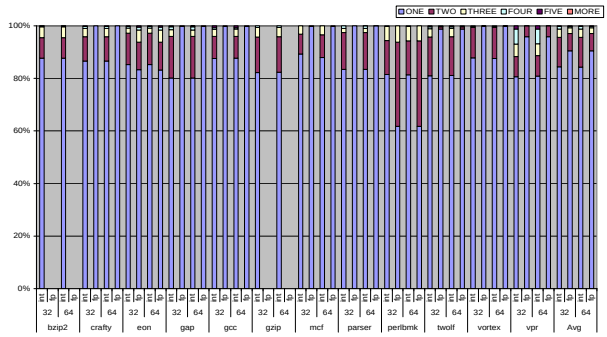

Figure 2: Distribution of instruction successors for SPECint

One integer and one f.p. benchmark have more than twice the average: 40% and 60%, respectively. Even more interestingly, a large number of benchmarks has a small percentage of instructions with 3, 4, and more successors.

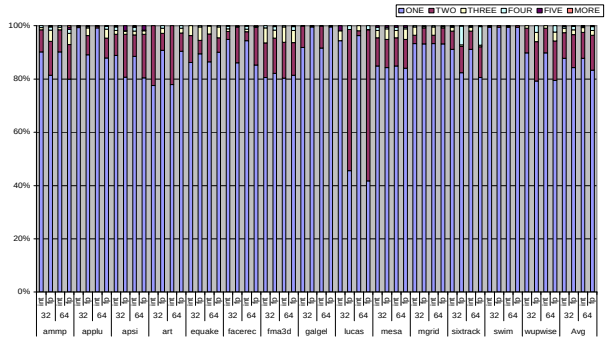

Figure 3: Distribution of instruction successors for SPECfp

These results indicate that one or two C-pointers should be sufficient for the direct wakeup implementation. They also indicate that a small number of MWT entries should suffice and that limiting the number of MWT entries is unlikely to significantly affect performance.

B. Performance

The study of successor instruction distribution above showed that one or two C-Pointers should be sufficient to achieve near-optimal performance. These two configuration parameters will be used in the remainder of this study. The other critical parameter is the size of the MWT table.

This section presents the evaluation of the effect of the MWT size on performance using IPC (instructions per cycle) as the evaluation metric. The results in Figures 4-7 show the impact of these parameters on the average performance. They are shown for 4- and 8-issue processors with 32- and 64-entry IQs and with one or two C-Pointers.

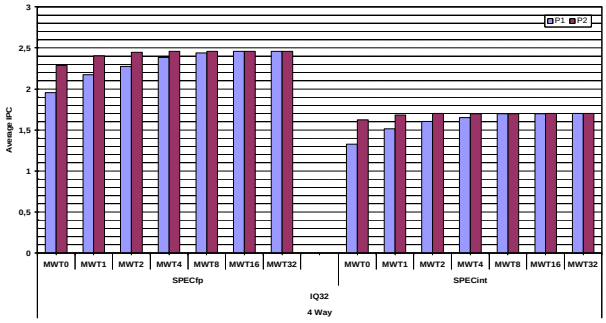

Fig. 4. Average IPC for different MWT sizes.
4-way processor, 32 entry IQ, 1and 2 C-pointers.

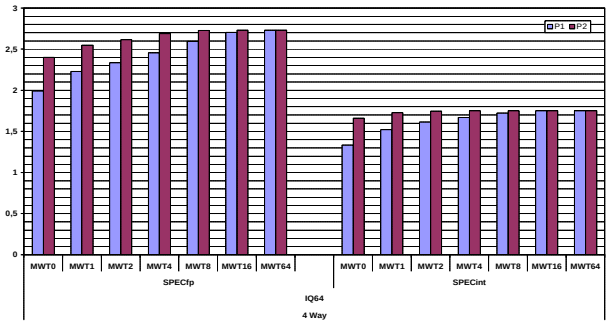

Fig. 5. Average IPC for different MWT size
. 4-way processor, 64 entries IQ, 1 and 2 C-pointers.

The results are presented separately for integer and floating-point benchmarks.

Two extreme cases can be distinguished in the results. First, the case of MWT0 corresponds to not having the MWT and only using one or two direct pointers. In this case the processor stalls when all pointers in the MT entry have been allocated.

The other extreme is the case of MWT equal to the IQ size, i.e. the equivalent of the full dependency matrix (plus one pointer).

The latter case corresponds to the highest performance possible (in terms of successor handling) since there can be no stall due to lack of pointer space. It can be seen in Fig. 4 that for a 4-issue processor with a a 32-entry IQ, one C-pointer, and eight MWT entries the performance reaches its peak. With two C-pointers only two MWT entries are sufficient. However, even a smaller MWT results in near-optimal performance: a 4-entry MWT results in approximately 2% IPC loss. It can even be argued that performance loss with a 2-entry MWT is small. Furthermore, with two C-pointers a single-entry MWT is near the maximum performance.

Similarly, the near-optimal results are achieved with an 8-entry MWT for an 8-issue processor (Figure 6) with a 32-entry IQ and one C-pointer.

The absolute performance loss for going to 4-entry MWT is slightly higher in this case. The performance loss for using a 1-entry MWT is almost negligible when two C-Pointers are used.

For larger instruction queue with 64-entries the performance loss increases when small MWT is used. For instance, the performance loss with a 1-entry MWT is almost double compared to the same configuration but a 32-entry IQ. The average performance with two C-pointers is again near-optimal with just four MWT entries for f.p. codes. For integer codes two MWT entries suffice.

Another optimization may be possible given that the architecture evaluated here uses separate integer and f.p. instruction queues, and thus separate MWT tables. Thus an integer MWT and the f.p. MWT can be of different size.

C. Power Evaluation

The energy and timing of CAM and RAM structures used in the instruction queues were evaluated using Spice3 tools for the 70nm technology. CMOS transistor and interconnect technology parameters used in the evaluation were obtained from ***BPTM*** provided by the Device Group at UC Berkeley. A design and pre-layout of all major structures were performed for different wakeup mechanisms, taking into account wires.

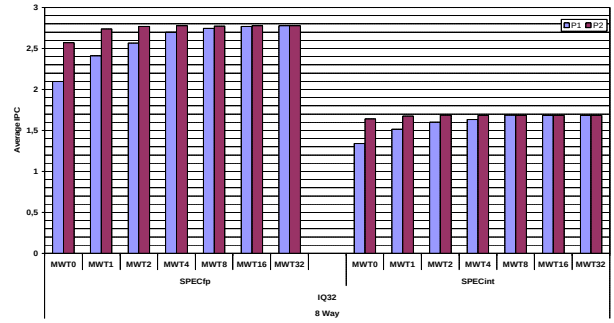

Fig. 6. Average IPC for different MWT sizes. 8-way processor, 32 entries IQ, 1 and 2 C-pointers.

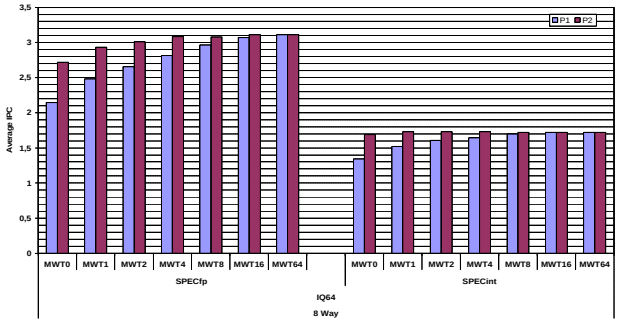

Fig. 7. Average IPC for different MWT size. 8-way processor, 64 entries IQ, 1 and 2 C-pointers.

An MWT with eight entries and a single pointer was laid out for a 4-way processor. This required 8 write ports and 6 read ports for MT. The status fields were designed as independent state-machines with eight parallel inputs that can change the state (OR their inputs) on each access to each physical register. The MWT, needed eight 1-bit write ports, and four, 32-bit (or 64-bit) read ports.

The average access power was computed taking into account all three components of IQ operation: allocation, issue, and wakeup. The power dissipation was measured using an equivalent circuit with capacitance obtained from Spice and analyzing the supply current flow using the transient analysis [19]. As seen in the results below, it is necessary to consider all three components of the total power to have accurate estimates. Only considering the wakeup energy, for example, significantly underestimates the total. The results in Table 2 show the average power for all three types of IQ organizations.

TABLE II
POWER EVALUATION

Issue Queue power for 70nm technology				
IQ entries	**CAM Wakeup**			
	Allocation *(mW)*	Issue *(mW)*	Wakeup *(mW)*	Total Avg Power$_T$
32	43.74	11.50	94.30	149.54
64	120.85	13.75	190.54	325.14
	Dependency Matrix Wakeup			
32	90.03	11.50	8.90	110.70
64	182.40	13.75	22.00	218.15
	Direct Wakeup			
32	16.44	11.50	21.38*	49.32
64	23.87	13.75	27.64*	65.26

The results show that the direct wakeup is the most energy efficient mechanism. This is the result of a much more efficient allocation than in the other two schemes. Dependency matrix has a somewhat lower *wakeup*-only power for a 64-entry queue and a much lower wakeup power for a 32-entry queue, but not the total power. But the total power is lowest for the direct-wakeup case.

V. Related Work

The energy consumption of a modern dynamically scheduled superscalar processor is between 50 and 100 Watts. At the micro-architecture level, the issue logic is one of the main consumers of energy responsible for approximately 25% of the total energy consumption [3]. Many approaches to designing the wakeup logic have been proposed, both to reduce delay and to reduce energy consumption. [4] proposed a pointer-based solution, where each instruction has a pointer to its dependent instruction for direct wakeup. This was done for a singe-issue, non-speculative processor, however.

[5] extended the above solution to modern processors with wide issue and speculation. The effect of one, two, or three successor pointer entries per instruction was evaluated. Three approaches to deal with the case of an instruction with more successors than pointer entries were proposed. The first one stalled the instruction issue. The second one stopped recording successors and instead woke the instruction up when it reached the top of the instruction window. Both of these approaches lead to a loss of IPC. The third approach added a scoreboard to avoid stalls, an expensive approach to say the least. Finally, successor pointers were saved on speculated branches, which is quite expensive. Overall, the solution does not require the use of CAM and thus significantly reduces both the delay and the energy consumption of the wakeup logic.

[6] proposed a circuit design to adaptively resize an instruction queue partitioned into fixed size blocks (32 entries and 4 blocks were studied). The resizing was based on IPC monitoring. The use of self-timed circuits allowed delay reduction for smaller queue size. [7] further improved this design using voltage scaling. The supply voltage was scaled down when only a single queue block was enabled.

[8] used a segmented bit line in the IQ RAM/CAM design. Only selected segments are used in access and comparison. In addition, a form of bit compression was used and a special comparator design to reduce energy consumption on partial matches.

[3] proposed a design, which divided the IQ into blocks (16 blocks of 8 entries). Blocks which did not contribute to the IPC were dynamically disabled using a monitoring mechanism based on the IPC contribution of the last active bank in the queue. In addition, their design dynamically disabled the wake up function for empty entries and ready operands[1].

[9] split the IQ into 0-, 1-, and 2-tag queues based on operand availability at the time an instruction enters the queue. This was combined with a predictor for the 2-operand queue that predicted which of the two operands would arrive last. The wakeup logic only examined the operand predicted to arrive last. This approach reduces the IPC while saving energy. First, an appropriate queue with 0-, 1-, or 2-tags must be available at issue, otherwise a stall occurs. Second, the last-operand prediction may be incorrect, requiring a flush.

[10] also used a single dependent pointer in their design. However, a tag comparison is still always performed requiring a full CAM. In the case of multiple dependent instructions a complex mechanism using *Broadcast* and *Snoop* bits reduces the total number of comparisons. The *Broadcast* bit indicates a producer instruction with multiple dependents (set on the second dependent). Each such dependent is marked with a *Snoop* bit. Only instructions with a *Snoop* bit on perform a comparison when an instruction with a *Broadcast* bit on

completes. Pointer(s) to squashed dependent instructions may be left dangling on branch misprediction and cause unnecessary comparisons, but tag compare guarantees correctness.

[11] used a full bit matrix to indicate all successors of each instruction in the instruction queue. Optimizations to reduce the size and latency of the dependence matrix were considered. This solution does not require the use of CAM but does not scale well with the number of physical registers and the IQ size which keep increasing. [12] also described a design of the Alpha processor using a full register scoreboard.

[13] proposed a scalable IQ design, which divides the queue into a hierarchy of small, and thus fast, segments to reduce wakeup latency. The impact on energy consumption was not evaluated and is hard to estimate.
In addition, dynamic data compression techniques ([14], [15]) have been proposed as a way to reduce energy consumption in processor units. They are orthogonal to the design proposed here.

Several mechanisms were also proposed for speeding up the wakeup/select critical timing loop in high-performance processors. [21] described *speculative wakeup*, which pipelines the scheduling logic over two cycles while having only a minor impact on IPC. Speculative wakeup uses a dependency lookahead scheme to stretch the critical scheduling loop (wakeup + select) over two cycles while still allowing dependent instructions to schedule in consecutive cycles.

[22] proposed a pointer-like scheme for keeping track of the first use of each result. A first-use queue indexed by the destination register tag keeps track of such instructions. There is also a (completely) Ready queue from which instructions are scheduled.

[23] proposed tag elimination, a scheme combining specialized windows, each with a different number of tags required, and last-tag speculation to speed up the wakeup logic. [24] proposed a "half-price" architecture which simplified the wakeup logic by using only one CAM tag and serializing wakeup when both sources became ready simultaneously.

VI. Conclusions

This paper presented a new instruction queue organization using pointers for dependent instruction wakeup. The new organization also uses a number of "multiple successor" bit vectors, similar to the dependency matrix approach but with a much smaller number of entries. It was shown that using one or two successor pointers per instruction plus a small number of entries in the multiple wakeup table result in high processor performance (measured by IPC). The results show performance for both a limited number of multiple wakeup entries and a full dependency matrix, with the former achieving the performance of the latter with just 4 or 8 entries and a single direct pointer.

The proposed IQ checkpoints 3 bits per instruction on each branch: the IQ valid bit and two status bits. The latter two bits store the number of successors at the time of the branch. This allows a simplified branch misprediction recovery mechanism to be used, in which the successor information is easily restored. The "multiple successor" bit vector is ANDed with the IQ valid bits in the case of multiple successors, eliminating any dangling pointers.

In addition to performance the power consumption of IQ access was evaluated for three IQ organizations: CAM-based, full dependency matrix, and the organization proposed in this paper. All major hardware elements of the three organizations were designed and laid out for a 70nm process. The new organization is shown to use 1/3rd of the total power of the CAM-based mechanism and 1/2 the power of the dependency matrix based mechanism for 32-entry IQ on a 4-issue processor. The power used becomes 1/5th and 1/3rd, respectively, for 64-entry IQs.

Acknowledgment

This work was supported by the Ministry of Science and Technology of Spain, under contract TIN2004-07739-C02-01 and HiPEAC, the European Network of Excellence on High Performance Architectures and Compilers, the program of scholarships SUPERA-ANUIES and COTEPABE-IPN from Mexico, and by the National Science Foundation in the U.S.A under grant NSF CCR-0311738.

References

[1] S. Palacharla, "*Complexity effective Superscalar processors*" PhD Thesis, University of Wisconsin, Madison 1998.
[2] Alper Buyoktusunoglu, Stanley E. Shuster. David Brooks, Pradid Bose, Peter W. Cook, and David H. Albonesi, "*An Adaptive Issue Queue for Reduced Power at High Performance*", Workshop on Power Aware Computer Systems, in conjunction with ASPLOS-IX, November 2000.
[3] Daniel Folegnani and Antonio González, "*Energy Effective Issue Logic*", Proceedings of 28th Annual of International Symposium on Computer Architecture, 2001. Page(s): 230-239, Göteborg Sweden.
[4] Shlomo Weiss, James E. Smith, "*Instruction Issue Logic for Pipelined Supercomputers*", Proceedings of 11th Annual International Symposium on Computer Architecture, 1984 Page(s): 110-118.
[5] Toshinori Sato, Yusuke Nakamura, Itsujiro Arita, "*Revisiting Direct Tag Search Algorithm on Superscalar Processors*", Workshop Complexity-Effective Design, ISCA 2001.
[6] Alper Buyoktusunoglu, Stanley E. Shuster. David Brooks, Pradid Bose, Peter W. Cook, and David H. Albonesi, "*An Adaptive Issue Queue for Reduced Power at High Performance*", Workshop on Power Aware Computer Systems, in conjunction with ASPLOS-IX, November 2000.
[7] Vasily G. Moshnyaga, "Reducing Energy Dissipation of Complexity Adaptive Issue Queue by Dual Voltage Supply", Workshop on Complexity Effective Design, June 2001.
[8] Vasily G. Moshnyaga, "Reducing Energy Dissipation of Complexity Adaptive Issue Queue by Dual Voltage Supply", Workshop on Complexity Effective Design, June 2001.

[9] Dan Ernst, Todd Austin, "*Efficient Dynamic Scheduling Through Tag Elimination*", Proceedings of 29th Annual of International Symposium on Computer Architecture, 2002.

[10] Michael Huang, Jose Renau and Josep Torrellas, "*Energy-Efficient Hybrid Wakeup Logic*", Proceedings of ISLPED August 2002 Page(s): 196-201, Monterrey California, USA.

[11] Masahiro Goshima, Kengo Nishino, Yasuhiko Nakashima, Shin-ichiro Mori, Toshiaki Kitamura, Shinji Tomita, "*A high-Speed Dynamic Instructions Scheduling Scheme for Superscalar Processors*" Proceedings of 34th Annual International Symposium on Microarchitecture, 2001.

[12] James A. Farrell and Timothy C. Fisher, "*Issue Logic for a 600-Mhz Out-of-Order Execution Microprocessors*" IEEE Journal of Solid State Circuits Vol. 33, No. 5 , May 1998. Page(s): 707-712.

[13] Steven E. Raasch, Nathan L. Binkert and Steven K. Reinhardt, "*A Scalable Instruction Queue Design Using Dependence Chains*", Proceedings of 29th Annual of International Symposium on Computer Architecture, 2002 Page(s): 318-329.

[14] L. Villa, M. Zhang M. and K. Asanovic, "*Dynamic Zero Compression for Cache Energy Reduction*", Micro-33, Dec. 2000.

[15] Vasily G. Moshnyaga, "*Energy Reduction in Queues and Stacks by adaptive Bit-width Compression*",. Proceedings of International Symposium on Low Power Electronics and Design, August 2001 Page(s): 22-27 Huntington Beach California, USA.

[16] Manoj Franklin, Gurindar S. Sohi "*The Expandable Split Window Paradigm for Exploiting Fine Grain Parallelism*", Proceedings of 19th Annual of International Symposium on Computer Architecture, 1992 Page(s): 58-67.

[17] D. Brooks, V. Tiwari, and M. Martonosi. "*Wattch: A framework for architectural-level power analysis and optimizations*. Proceedings of 27th Annual International Symposium on Computer Architecture, June 2000.

[18] Marco A. Ramírez, Adrian Cristal, Alexander V. Veidenbaum ,Luis Villa, Mateo Valero. "*A Simple Low-Energy Instruction Wakeup Mechanism*", Proceedings Intl. Symposium on High-Performance Computing (ISHPC-IV), Oct. 2003

[19] Sun Mo Kang, "*Accurate Simulation of PowerDisipation in VLSI Circuits*", IEEE Journal of Solid-State Circuits, vol. SC-21, No. 5, October 1996.

[20] Gregory J. Fisher, "*An Enhanced Power Meter for Spice2 Circuit Simulation*", IEEE Transaction on Computer-Aided Design. Vol.7 No. 5 May 1988.

[21] Kenneth C. Yeager, "The MIPS R10000 Superscalar Processor," IEEE Micro, April 1996.

[22] R. Canal, A. Gonzalez, "A Low-Complexity Issue Logic," Intl. Conference on Supercomputing, May 2000.

[23] D. Ernst,T. M. Austin, *Efficient dynamic scheduling through tag elimination*, Intl. Symposium on Computer Architecture, 2002.

[24] lhyun Kim, Mikko H. Lipasti, "*Half-price Architecture*," Intl. Symposium on Computer Architecture, June 2003.

[25] Alexander Henstrom US patent number 6557095 "Scheduling operations using a dependency matrix" December 27, 1999. Intel Co.