Increasing Design Space of the Instruction Queue with Tag Coding

Junwei Zhou

Department of ECE, Michigan State University
East Lansing, MI 48824
zhoujunw@egr.msu.edu

Andrew Mason

Department of ECE, Michigan State University
East Lansing, MI 48824
mason@egr.msu.edu

ABSTRACT

The instruction queue is a critical component and performance bottleneck in superscalar microprocessors. Conventional designs use physical register identifiers to wake up instructions. This paper proposes decoupling the tags for instruction wakeup from the tags for physical register access, thus increasing the design space of the instruction queue by encoding its operand tags. Two coding methods have been developed. One uses a linear code to increase the Hamming distance between tags, reducing the tag match delay by more than 50% and achieving 12% improvement in the total wakeup/select delay for TSMC 0.18 μ m technology at 1.8v. The second method uses one-hot code to encode the operand tag, removing the tag OR and tag read operations from the wakeup/select loop. For a 32-entry instruction queue, 15% reduction in the wakeup/select loop has been achieved. Furthermore, one-hot code also removes the dissipation-on-mismatch in the wakeup logic, significantly reducing the dynamic power consumption of the instruction queue.

Categories and Subject Descriptors

C.0 Computer System Organization: System Architectures.
C.5.3 Microcomputers: Microprocessors

General Terms: Performance, Design, Experimentation

Keywords: Instruction Queue, Wakeup/select Loop

1. INTRODUCTION

In the superscalar microarchitecture, operand registers of fetched instructions are mapped to physical registers at the renaming stage to remove false data dependence. The renamed instructions are then inserted into the instruction queue and their logical source operands are replaced with physical register identifiers (tags). Instructions wait at the instruction queue until their source operands are ready. Each cycle, the destination operand tags of issued instructions are broadcast to each entry of the instruction queue, waking up the dependent instructions. The ready instructions compete for issue slots, and the winners put their destination tags on the broadcast bus, forming a closed loop. To achieve high IPC performance, this wakeup/select loop at the instruction queue has to be finished in one clock cycle so that the dependent instructions can be executed consecutively.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GLSVLSI'05, April 17–19, 2005, Chicago, Illinois, USA.

Copyright ACM 1-59593-057-4/05/0004...\$5.00.

In the conventional design, the operand tag corresponding to a physical register is the binary code of the register's identifier. The tag is used for both physical register access and instruction wakeup. In fact, the tags in the wakeup/select loop keep track of data dependencies between instructions waiting in the instruction queue. They can be decoupled from the tags used to access data storage registers, thus permitting the use of tag coding techniques and increasing the design space of the instruction queue.

The conventional instruction queue and its delay sources are described in this paper along with the potential benefits of decoupling operand tags. Two operand coding methods that reduce the total wakeup/select loop delay are discussed and analyzed, and simulations results of these coding methods are presented.

2. INSTRUCTION QUEUE

A conventional instruction queue, shown in Figure 1, is typically implemented with a RAM storing the destination tags and a CAM storing the source operands and searches for dependent instructions associatively. The destination tag for every issued instruction is read from the RAM and sent to the comparator input port of the CAM where a ready bit is set when the associated source tag is identified. When both source operands of an entry are ready, an issue request is sent to the select logic. When an issue slot is won, the instruction's destination tags are read and broadcast to the CAM to wakeup their dependent instructions in the next cycle.

Figure 1. Conventional wake/select logic.

A critical delay loop in the pipeline is formed by sequentially reading destination tags of issued instructions, waking up dependent instructions, and selecting ready instructions for issue.

The read tag delay is primarily due to wire delays of the RAM bitline. The select logic delay is determined by the delay from a sequence of priority encoders. The wakeup delay is defined by three factors:

- Tag drive: The destination tag read from the RAM is driven to the tag bitline of the CAM.
- Tag match: The CAM word matchline, shown in Figure 2, is precharged to VDD and held if the data on the tag bitline is equal to the data stored in the memory cell. If there are any mismatches, the matchline is pulled to ground.
- Match-line OR: OR gates combine the match results. If any of the match lines are high, the ready bit of the corresponding operand is set to indicate the availability of the operand.

A decomposition of the wakeup/select loop delays for a 64 entry, 4-issue instruction queue are shown in Figure 3. These simulation results identify the wakeup delay as the largest contributor to the total loop delay.

Figure 2. A CAM word with one comparator.

Figure 3. Delay decomposition of an instruction queue (64 entry, 4-issue wide) for TSMC 0.18um technology.

In the conventional design, the physical register identifiers are used both to wakeup waiting instructions and to index the physical register at later cycles. This unnecessarily links register access to the wakeup/select process and limits options to optimize the instruction queue design. However, decoupling these tags increases the instruction queue design space and allows the development of tag coding methods that target different delay components in the wakeup/select loop.

3. REDUCING TAG MATCH DELAY

As indicated in Figure 3, tag match delay takes up more than 30% of the total wakeup delay. At the precharge stage, the matchline in Figure 2 is precharged to VDD through a PMOS transistor. At the evaluation stage, if there are any mismatch bits between the tag and the data in memory cells, the matchline is discharged through the NMOS transistors of the mismatching cells. The worst case

delay occurs when only one bit is mismatched, turning on only one pair of discharge NMOS to pull down the match line.

To reduce the tag match delay, one option is to increase the size of the pull-down transistors. However, this also increases the capacitance of the matchline due to transistor junction capacitance, increasing discharge time and offsetting the speed benefit of large transistor. Simulation shows less than 2% improvement in delay using large pull-down transistors.

The tag match delay also depends on the number of simultaneously active pull-down transistors, which is determined by the Hamming distance between tags being compared. In the conventional IQ design, the tags are physical registers identifiers, and the minimum Hamming distance (MHD) between tags is one. However, if the tags are encoded so that the minimum Hamming distance is greater than one, tag match delay can be reduced by turning on more pull-down transistors simultaneously.

3.1.Tag Encoding

Decoupling the tags for instruction wakeup from physical register identifiers needs to implement a separated set of tags in the instruction queue. This requires additional hardware to store the tags and link the tags of the issued instructions to the physical register identifiers. Binary linear codes [1] are suitable for this application. A linear code C is referred to as an $[n, k]$ code, where n is the length and k is the dimension of the code. A linear code can be considered as having two parts. The first part of a codeword has k bits representing the information content. In this case, it is the physical register identifier. The second part has $n-k$ bits. These data are redundant bits that are used to increase the code Hamming distance. Therefore, there remains a direct mapping from the tags in the instruction queue to the physical register identifier. Only $n-k$ memory cells are needed to store the redundant bits.

The minimum Hamming distance of a linear code C is determined by the minimum weight of C [1], where the nonzero weight $w(C)$ is determined by the number of logic-high bits in the codeword. A minimum Hamming distance of d requires each codeword has at least d logic-high bits. MHD=2 can be achieved using the check parity code, which ensures all nonzero codes have an even number of, at least two, logic-high bits. A greedy algorithm can be used to find a linear set of codes with $MHD > 2$, further reducing the average tag match delay. Figure 4 shows that the number of redundant bits required grows as MHD increases. With $MHD > 5$, more redundant bits than information bits are needed, significantly increasing the size of the tag. Larger tags require more memory and incur larger wire and diffusion capacitance on the matchline, offsetting the benefit of increasing the discharge path. Based on these considerations, only codes with MHD less than four have been considered

Figure 4. The number of redundant bits versus the minimum hamming distance (8 information bits).

3.2. Implementation

Operand tags are obtained at the instruction decoding stage when the names of logical registers are translated into physical register identifiers. It is assumed that a CAM structure mapping table is used to keep track the mapping from logical registers to physical registers [2]. Each entry of the mapping table contains the name of a logical register and a valid bit to indicate the latest mapping of logical register with multiple entries. The word lines corresponding to the matched entry are used to access a ROM and retrieve the physical register identifier code.

In the traditional design, each ROM entry stores the binary code of a physical register identifier. Thus the length of an entry is logarithmically (base 2) proportional to the register file size. In this design, each entry stores the codeword of a linear code. The same number of cells store the register identifier, and extra cells store the redundant bits. Assuming, for example, the check parity code is used, only one extra cell is needed in the ROM representing the parity of the physical register identifier. No extra delay is introduced because the redundant bits are stored in ROM and accessed in parallel with the information bits.

When a ready instruction is selected for issue, its destination tag, consisting of the physical register identifier and the redundant bits, are broadcast to the instruction queue. However, the redundant bits are not sent to the pipeline register to access the register file and thus have no effect on any other hardware units.

3.3. Results

Figure 5 shows the wakeup logic delay versus different coding schemes for TSMC 0.18 μ m technology at 1.8v. Tags with MHD=1 represents the conventional binary code. Tags with MHD=2 represents the check parity code. Results show that the check parity code reduces the tag match delay by 42% over the traditional design. The tag OR delay is also reduced by 17% due to improving the slope of the matchline signal that, in turn, increases the switching speed of the buffer and NOR gates at the end of the match line. The tag drive delay is dependent on the size of the instruction queue and therefore remains constant. Overall, the wakeup delay is reduced by 19% with the check parity code.

Figure 5. Wakeup delay of an instruction queue (4-issue, 64 entries, 8-bit register file identifier).

Using tags with MHD=3 achieves an additional 10% reduction in the tag match delay. The improvement in delay is not linear with the MHD because the number of cells storing the redundancy bits grows from one to four as the tag distance is increased from two to three. Overall, using tags with MHD=3 reduces the wakeup delay by 24%, resulting in 12% improvement in the whole wakeup/select loop.

4. REMOVING TAG-OR DELAY

The tag OR delay, as indicated in Figure 3, contributes 33% of the total wakeup delay. A different encoding method that can eliminate this delay component has also been developed.

Consider that each source operand in an instruction queue entry corresponds to IW match lines that are ORED to set the ready bit of the operand. Let the destination tags of issued instructions be $c^1, c^2, \dots, c^{IW}$, and the source tag of entry k in the instruction queue be c_k . In the traditional design, c_k is encoded in the binary code leading to the following comparison operation. Let the compare operation of tags a and b be $a \oplus b = \sum_m a_i \oplus b_i$.

Then the function to set the ready bit is

$r_k = c^1 \oplus c_k + c^2 \oplus c_k + \dots + c^{IW} \oplus c_k$, and the ready bit is high if any of $c^1, c^2, \dots, c^{IW}$ are equal to c_k .

However, if c_k is instead encoded with one-hot code, it is possible to remove the OR operations from this comparison as follows. Define the dot operation of tags a and b as $a \cdot b = \sum_m a_i \cdot b_i$ and notice that $c_i \cdot c_j = 1$ for $i=j$ and

$c_i \cdot c_j = 0$ for $i \neq j$. Now consider the function

$r_k' = c^1 \cdot c_k + c^2 \cdot c_k + \dots + c^{IW} \cdot c_k = (c^1 + c^2 + \dots + c^{IW}) \cdot c_k$,

where if any of $c^1, c^2, \dots, c^{IW}$ are equal to c_k then

$r_k' = c_k \cdot c_k = 1$, otherwise $r_k' = 0$. This function can be used

in the instruction queue for associative search operation, and $c^1 + c^2 + \dots + c^{IW}$ corresponds to the broadcast signal that update the instruction queue. By mapping the active bit in the one-hot code to the grant signal that is set high when the corresponding entry wins an issue slot, all of the grant lines realize logic $c^1 + c^2 + \dots + c^{IW}$; therefore no OR gates are needed to implement the function r_k' .

The combination of the grant lines is equivalent to the destination tags read from the RAM in the conventional design. Thus, one-hot encoding also removes the RAM read operation from the wakeup/select loop, further reducing the loop delay. The wakeup logic that implements one-hot encoding is similar to the dependence matrices described in [3].

4.1. Implementation

The register rename logic keeps track of the mapping from logic registers to both physical register identifiers and instruction queue identifiers. An entry of the register map table is shown in Figure 7. It contains the physical register identifier (R_tag) and the instruction queue identifier (I_tag) mapped to the same logical register. The logical register identifier is used to index the map table. An instruction reads the table to obtain the R_tag and I_tag for each source register. The tags of the allocated physical register and the instruction queue are also written to the map table. The