A Physical Level Study of Two-Stage Register Renaming

ABSTRACT

Register renaming is a performance critical component of modern, dynamically-scheduled processors. Register renaming latency increases as a function of several architectural parameters such as issue width, window size and number of recovery checkpoints. To avoid impacting the processor’s clock frequency, pipelining of the register renaming logic is desirable. This work demonstrates a detailed, full-custom, two-stage register renaming implementation in a 130nm fabrication technology. The resulting latency is expressed in fan-out-of-four (FO4) and, the underlying performance and complexity tradeoffs are discussed. Latency results show that the processor’s logic depth is reduced from 23 down to 9.5 FO4 going from single-stage to two-stage renaming.

Categories and Subject Descriptors
B.7.1 Integrated Circuits , C.1 Processor Architecture.

General Terms

Keywords
Register renaming, pipelining, latency.

1. Introduction

Modern high-performance processors increase instruction level parallelism (ILP) by removing artificial data dependencies via register renaming. The register alias table (RAT), the core of register renaming, maintains mappings between architectural (the register names used by instructions) and physical registers (the names of the physical storage elements that hold the values at runtime). RAT implementations can be based on content-addressable (CAM) or static random access (SRAM) memory structures. The SRAM-based RAT, which this work focuses on, is more scalable and energy efficient for larger instruction window sizes [5]. In the rest of this paper, the terms renaming and RAT are used interchangeably.

As addressed in Section 2, RAT complexity and size, and hence its latency and energy, vary as a function of several architectural parameters. The RAT is a performance-critical component as it is read and updated by all instructions in order as these instructions are decoded. Accordingly, the RAT should operate at the processor frequency or it must be pipelined.

While previous work have discussed the possibility of pipelining the RAT [5][8], no previous work has detailed how the renaming logic can be pipelined and measured how pipelining affects its latency. Pipelining introduces overheads such as bypassing and potentially bubbles in case of bypassing. Hence, pipelining does not necessarily lead to better performance or latency. This work investigates a detailed two-stage register renaming implementation at the physical level. It presents a custom implementation in a commercial fabrication technology. Using this implementation, this work analyzes the renaming latency in fan-out-of-four (FO4) and discusses the underlying performance and complexity tradeoffs. Simple analytical models are also presented that express RAT latency as a function of the issue width (IW) and window size (WS). These models are useful at the early stages of architectural exploration where actual implementations are often unavailable. Latency results show that the logic depth of a 4-way superscalar processor is reduced from 23 to 9.5 FO4 going from single-stage to two-stage renaming.

2. Register Renaming Background

Register renaming removes false name dependencies that artificially limit instruction level parallelism (ILP). Renaming maps the architectural register names used by instructions into the physical registers implemented in the processor. False dependencies are introduced when the same architectural register is used to hold different values over time. Renaming assigns different physical registers to those values eliminating the false dependencies. At any given point of time, a RAT read for an architectural register returns its most recently assigned physical register name. The number of physical registers is larger than the number of architectural registers so that many instructions can be active in the processor concurrently. The SRAM-based RAT is a multi-ported register file with as many entries as the number of architectural registers. The RAT entry width is equal to the physical register name (e.g., 7 bits for 128 physical registers).

Renaming a single instruction proceeds as follows: 1) Reading the physical register names for the source register operands; 2) Reading the current mapping of the destination register to be saved in the reorder buffer (ROB) for supporting speculative execution [2][6]; and 3) Updating the RAT for the destination register with a physical register name allocated by the pool of free registers. Actions (1) and (2) proceed in parallel, while action (3) follows. Acquiring a free physical register name can be done at any time prior to action (3).

In superscalar processors several instructions are simultaneously renamed per cycle (a rename group), and hence dependencies may exist among the co-renamed instructions. In this case, to reduce latency, RAT reads proceed in parallel with intra-group dependency checks. The dependency checks are used to appropriately handle intra-group WAW and RAW dependencies. At the end, all the RAT writes are performed taking into account WAW dependencies so that only the latest update per destination register is saved in the RAT. Specifically, superscalar single-cycle renaming involves the following actions:
1. RAT reads for the source operands and the destination operands (for saving into the ROB) of the co-renamed instructions are initiated.

2. In parallel with (1), new physical registers are allocated for the destination registers of all co-renamed instructions.

3. The physical register names available by the end of the RAT read stage do not reflect map changes from the co-renamed instructions. These names are the ones left by the immediately preceding renaming group. Read-after-write (RAW) dependencies among co-renamed instructions need to be detected; each source register name is compared with the destination register names of all its preceding instructions. These comparisons proceed in parallel with actions (1) and (2). When the RAT reads and the dependency checks complete, the comparison results are priority encoded to drive multiplexers selecting the most recent, preceding physical register name assigned to each source architectural register. In case of a RAW dependency, the physical name read from the RAT for a source register of an instruction is replaced with the physical register name assigned from the pool of free registers, in (2), to the most recent, preceding co-renamed instruction that writes to that register.

4. Write-after-write (WAW) dependencies may exist among co-renamed instructions and must be detected for two reasons: a) to write only the latest update per architectural register in the RAT when multiple co-renamed instructions write to the same architectural register; b) to select the appropriate previous mapping to store in the ROB for each destination register (either the one returned after the RAT reads of (1), or the one written by a preceding write within the same co-renamed group in (2)). As with RAW dependencies, these comparisons proceed in parallel with (1), (2) and (3). Figure 1 shows the hardware block diagram of the 4-way superscalar renaming (single-cycle).

### 2.1 Checkpointed RAT

Modern processors support speculative execution where instructions are renamed and possibly executed before it is certain that they should. On misspeculations, erroneously executed instructions are “squashed”, i.e., all changes made by them must be undone including any RAT changes. The RAT must be restored so that it does not contain any of the mappings introduced by the incorrectly speculated instructions. Modern RAT designs incorporate a set of global checkpoints (GCs) to recover from misspeculations. A GC is a complete snapshot of all relevant processor state including the RAT. A GC is taken when an instruction initiates speculation (e.g., a branch whose target address is predicted) and released when the corresponding instruction commits. To recover from a mispeculation, the GC is copied back to the RAT.

In addition to GCs, which provide fast recovery but are few as they are expensive to build [7], modern processors use the ROB, an instruction-by-instruction log of all changes done to the RAT. Recovery using the ROB requires a time proportional to the number of instructions that should be reversed. Recovery using a GC is instantaneous, i.e., it requires a small, fixed latency.

This work uses a GC organization mechanism that organizes GCs in small bidirectional shift registers embedded next to each RAT bit, as shown in Figure 11; this mechanism scales better than the alternative [7].

### 2.2 RAT Functionality and Requirements

The RAT operations are as follows: (1) read the mapping for a register from the RAT, (2) write the mapping for a register to the RAT, (2) allocate a GC, and (3) restore from a GC. To simultaneously rename N instructions, having up to two sources and one destination, the RAT needs 3xN read ports and N write ports. Up to 2xN read ports to locate the current mappings for the two source registers and the other N to retrieve the mappings the destination registers had prior to renaming so that they can be recorded in the ROB. The N write ports are used to update the RAT with up to N new mappings for the destination registers.

The RAT’s complexity and size, and hence its latency varies as a function of several architectural parameters. Section 3.1 discusses how the latency and energy of this two-stage pipeline varies as key architectural parameters change. Section 3.2 discusses the impact of deeper pipelining. Finally, Section 3.3 reviews other ways of reducing renaming latency.

Figure 3 shows the high-level block diagram of two-stage renaming. For simplicity, we first explain how non-superscalar (i.e., where only one instruction is renamed per cycle) can be pipelined. Then, we discuss superscalar renaming. Without the loss of generality, we first assume that each instruction can have up to two sources and up to one destination register.

As Figure 3 shows, renaming proceeds in two stages, each comprising two sub-stages. We assume that the current instruction B enters stage 1, while the immediately preceding instruction A enters stage 2. The first half of the first stage decodes the source and destination registers of instruction B. The decoding outputs are latched so that they remain stable while reading from the RAT the mappings for the source and the destination registers (starts at the second half of stage 1 and continues to stage 2) and while writing the destination register’s new mapping (starts at the first half of stage 2). In parallel with decoding the source and destination registers for B, dependency checking with instruction A is performed; dependency checking uses the architectural names of registers in A and B which are available. The WAW and RAW dependency checks complete before the RAT writes in stage 2 and some time while the RAT reads are in progress. The WAW dependency results select the value to be stored for B in the ROB. This value is either A’s destination physical register in the case of a WAW dependency between B and A or the mapping read from the RAT otherwise. The RAW dependency results are used to select the appropriate physical register for B’s source register (either the value read from the RAT, or A’s destination). The RAT reads are initiated during the second half of stage 1 and completed by the end of the first half of stage 2. As soon as the RAT reads complete, the RAT write for B’s destination starts. The write updates the entry for B’s destination register with a free
physical register obtained from the free list. During the second half of stage 2, the appropriate source and destination physical names are selected either from those read from the RAT (no dependency between A and B), or those provided by the free list for A (dependency between A and B).

Compared to single instruction renaming, superscalar renaming introduces the possibility of inter-group dependencies. Compared to the previous discussion, both intra-group and inter-group dependencies between adjacent groups must be detected and handled appropriately. Specifically, superscalar, pipelined renaming entails the following actions:

1. Starting at the beginning of stage 1, the RAW dependencies among the current and the previous rename group are detected. Both intra- and inter-group dependencies must be detected so that the appropriate mapping is assigned to each source register. This mapping is either the register that will be read from the RAT or the physical register name assigned to the closest producer. Inter-group RAW dependencies must be detected since the previous group’s mappings have not been written into the RAT by the time the new group reads from the RAT. These pending RAT updates have to be bypassed. RAT updates start at stage 2 and do not complete before the end of it. Accordingly, reading from the RAT reflects the updates performed two rename groups back.

In case of a RAW match, the most recent physical register name assigned to the producer instruction’s architectural destination register will be the map used for the consumer instruction’s source register. Any source architectural register that does not have a RAW dependency on any of the instructions in the current or the previous group is renamed to the physical register name read from the RAT. The outputs of the RAW dependency comparators are priority encoded to select the most recently-assigned physical register name for the architectural source operand during stage 2.

**Circuit considerations:** The 8-bit comparator’s and 8-bit priority encoder’s transistor-level implementations are depicted in Figures 9 and 7 respectively. These 8-bit macro cells can be cascaded to form a larger encoder as shown in Figure 10 [10]. The N-input encoder receives N input requests and sets one of N outputs considering the input priority. The priority encoder outputs drive the select lines of the output multiplexer’s transmission gates in stage 2. Figure 12 shows the structure of the output multiplexer which is an N-to-1 multiplexer with N, M-bit inputs. The N input select control signals are driven by the priority-encoder.

2. Starting at the beginning of stage 1, the source architectural registers are decoded, and then are latched to be stable during the RAT reads. The RAT reads are started as soon as decoding is finished and occupy part of both stages.

**Circuit considerations:** The source’s map read from the RAT is required only if no RAW dependency exists. Hence, once the RAW dependency check results are finalized, unnecessary RAT reads can be stopped to reduce energy. This is desirable since sense amplifiers can be a major source of energy consumption in the RAT [7]. The comparators’ outputs are NORed to activate the sensing logic of the associated bitlines. If one of the comparators detects a match, the NOR output becomes zero, thereby deactivating the sense amplifiers. Decode and bitline precharge latencies are overlapped with comparator and NOR latencies.

3. Up to issue width (IW) currently free physical registers are allocated to the destination architectural registers by the end of stage 1. The names are latched to be used in stage 2. The free register names are provided by the register free list. Different register free list implementations are possible. Our implementation uses a circular FIFO capable of providing up to IW physical register names per cycle.

4. For updating the RAT, intra-group WAW dependencies are detected starting at the beginning of stage 1. Multiple instructions may write to the same architectural registers within the current group; accordingly, only the latest update should be recorded into the RAT. However, Inter-group WAW dependencies must also be detected so that the appropriate values are stored in the ROB. This is because the previous group’s mappings have not been written into the RAT by the time the new group is supposed to read the old mappings from the RAT and save them into the ROB.

5. Starting at the beginning of stage 1, the destination architectural register addresses are decoded and latched to be kept stable during the RAT reads for the purposes of ROB checkpointing and for updating the RAT with new mappings. Once decoding is completed, RAT reads are initiated for reading the old mappings to record them in the ROB. RAT updates start once the old map reads complete.

**Circuit considerations:** To reduce energy, unnecessary RAT reads for old mappings can be stopped once a WAW dependency is detected. The transistor level implementations for the 8-bit priority encoder, the large priority encoder, the comparator, latches, and the output multiplexers are shown in Figures 7, 8, 9, 10 and 12 respectively. In single-cycle renaming, the SRAM latching delay is completely exposed. In the pipelined implementation, part of the SRAM latching (Figure 5) is hidden by the pipeline latches. Referring to Figure 5, RAT accesses are pipelined as follows: (1) during clk1-high, the address is decoded and latched and the bitlines are also pre-charged. (2) For writes, in clk2-high, the write input data are latched and the write transaction happens. (3) For reads, in clk1-high, the bit line voltages start to swing and sensed by sense-amplifier whose output is then latched. (4) For a write followed by a read for the old mapping, as the read data latch signal defines the end of the read operation, it also triggers a write operation.

### 3.1 Scalability of Two-Stage Renaming

This section discusses the components along the critical path in two-stage renaming, and how their latency and power scale as a function of the window size (WS) and issue width (IW).

The components along the critical paths of the first and second stages are: (i) **Comparators:** the width of each comparator is Log2(# architectural registers). The maximum number of comparators needed per source operand is (2*1W-1). Increasing IW increases the number of comparators; however, all comparators work in parallel, hence increasing IW does not affect the comparison latency, but noticeably increases power. (ii) **Priority Encoder:** increasing IW increases the latency of the IW-input priority encoder and the (2*1W-1)-input NORs. The encoders are driven by the comparators of (i). (iii) **Output
Multiplexers: the width of each multiplexer is \( \log_2(WS) \). The maximum number of multiplexers needed per source operand is \( 2 \times IW \). Increasing IW increases the number of pass gates; however, they work in parallel, so increasing IW does not affect the multiplexer delay, although noticeable increases power.

(iv) RAT: non-checkpointed RATs are essentially multi-ported register files. Their access latency grows linearly with entry count and quadratically with the number of ports. The \( RAT \) size and power grow linearly with the entry count and faster that linearly and slower that quadratically with the number of ports. For checkpointed RATs, previous work has determined quantitatively how the latency and energy vary as a function of WS, IW and the number of GCs (NoGSs) \cite{7}. The equations representing the relations among architectural parameters—IW, WS and NoGSs—and the physical organization parameter—entry count, entry width, the number of read and write ports—are given by equations \((1)-(3)\). Increasing WS increases RAT entry width, and hence increases delay logarithmically \cite{6}. The latency increases exponentially with increasing NoGSs. Increasing IW increases latency quadratically as it increases the number of ports.

3.2 Deeper Pipelining

Using a similar methodology, renaming can be pipelined further. However, complexity increases rapidly as dependency checks must be extended across more groups and additional bypass paths are needed for inter-group dependencies. It is important to observe that after the RAT reads complete, a set of multiplexers selects the appropriate value. For example, Figure 4 shows the organization of a three-stage pipelined renaming where a RAT write followed by a read to the same entry takes two cycles. In this case, the multiplexers that necessarily operate after the RAT reads have one more option to select from, hence making them slower. In general, the deeper the pipeline, the more options the final multiplexers have to select from and the slower they become. Hence, there is a point past which this delay becomes detrimental to performance.

Besides the increased complexity, other reasons may make deeper pipelining undesirable. While deeper pipelining can improve performance by providing a faster clock, it suffers from performance-degrading side effects. Specifically, due to the extra pipeline stages that have to be re-filled, deeper pipelining in the front-end increases the mispredicted branch penalties and instruction cache miss penalties. Deeper pipelining will be useful only if the performance improvement it provides (faster clock) surpasses the previously mentioned performance-degrading side effects \cite{8}. Executing dependent instructions in consecutive cycles is required for high performance, especially for programs with limited parallelism; otherwise, pipeline bubbles can cause significant performance loss.

3.3 Other Latency Reduction Techniques

Aside from pipelining, register renaming latency can be reduced in several ways: First, the RAT can be duplicated to reduce the number of ports on each RAT copy. Second, considering that not all instructions have two operands and that co-renamed instructions are highly likely to have common operands, the RAT ports can be reduced at the cost of insignificant IPC performance loss. Third, using GC prediction and selective GC allocation, the number of RAT GCs can be reduced while maintaining performance \cite{1}\cite{2}. Pipelining is orthogonal to these techniques. Any or all of the above techniques can be used to reduce latency to the desired level if possible.

### 4. Evaluation

Section 4.1 details the design and evaluation methodologies. Section 4.2 validates that our circuit implementations are reasonably tuned. Section 4.3 presents a simple analytical model for \( RAT \) delay and uses it to justify why a two-stage pipeline was an appropriate choice for this technology. Finally, Section 4.4 reports the latency breakdown for the pipelined renaming unit.

#### 4.1 Methodology

We developed full-custom layouts using Cadence\textsuperscript{\textregistered} tools in a commercial 0.13 \( \mu m \) technology with a 1.3V supply voltage. This was the best technology available to us at the time of this experiment. The key observations can apply, for the most part, to smaller technologies. The main differences may be in the relative importance of dynamic versus static power dissipation.

We implemented a 4-way superscalar RAT, a width representative of the processors built in the given fabrication technology and of many modern processors as well. We assume 64 and 128 architectural and physical registers respectively, two very common sizes today. We also assume that the window size and the number of physical registers are the same since the exact implementation of the scheduler and the window is orthogonal to this study. We consider a RAT with 16 GCs since previous work show that with \( GC \) prediction, 16 GCs or even less are sufficient to achieve performance close to what is possible with infinite GCs when focusing only on control-flow misspeculations \cite{1}\cite{2}.

We initially used minimum size transistors, and then increased dimensions to achieve latency less than an upper bound. We arrived at this upper bound by estimating the delay of a 64-bit, 64-entry \( SRAM \) with 12 read and 4 write ports using CACTI 4.2 \cite{9}. We used Spectre\textsuperscript{\textreg} for circuit simulations, and we report worst case latency results.

For delay analysis, we use the FO4 delay metric. CMOS FO4 delay can be estimated for a fabrication technology using formula \((4)\). The FO4 delay for the specific 130 nm technology FO4 is measured at 40 ps by simulations. In typical commercial designs of dynamically-scheduled, superscalar processors, the optimal pipeline depth is around 10-14 FO4 for performance-only optimized designs and 24-28 FO4 for power-performance \((Energy \times delay^2 \text{ metric})\) optimized designs \cite{11}. Given by \((5)\), the logic depth is calculated by subtracting the pipeline depth from the pipelining clock overhead. The overhead includes clock skew and jitter and latch delay. In a custom design flow, most of the clock skew and jitter overheads can be hidden by circuit techniques such as time borrowing. The latch overhead could be reduced by using techniques such as pulsed clocks and/or direct domino pipelines.

<table>
<thead>
<tr>
<th>Rename width: ( N ) (equal to IW)</th>
<th>Number of checkpoints: ( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of source registers per instruction: ( S )</td>
<td>Number of destination registers per instruction: ( D )</td>
</tr>
<tr>
<td>Number of architectural registers: ( A )</td>
<td>Number of physical registers: ( P ) (equal to WS)</td>
</tr>
</tbody>
</table>

\( A \times N = (S+D) \times N, D \times N \)
Accordingly, a 3FO4 pipelining overhead is a reasonable approximation. Clock overhead including margins for setup/hold time is estimated at 4-6 FO4 without the mentioned techniques: (i) pulse-mode Latch 1-1.5 FO4/ flip-flops 2-3 FO4, (ii) skew 2-4 FO4, and (iii) jitter 1-2 FO4.

\[
\begin{array}{|c|c|c|}
\hline
\text{Component} & \text{Delay} & \text{Remarks} \\
\hline
6-64 decoder and wordline driver & 5 FO4 & \text{NOP}2 + 1.8378 \times \text{NOP} + 449.31 \\
Latch delay & 2.5 FO4 & \text{NOP} \times \text{NOP}^2 + 1.8378 \times \text{NOP} + 449.31 \\
Wordline activation to sense-amp output & 2.4 FO4 & \text{NOP} = 4 \times \text{I}W, \text{NOP} > 16 \\
Wordline activation to write in data cell & 2.8 FO4 & \text{NOP} = 4 \times \text{I}W, \text{NOP} > 16 \\
8-bit comparator & 3 FO4 & \text{NOP}^2 + 1.3878 \times \text{NOP} + 449.31 \\
8-bit dynamic NOR & 3 FO4 & \text{NOP} \times \text{NOP}^2 + 1.8378 \times \text{NOP} + 449.31 \\
output Multiplexer & 2.5 FO4 & \text{NOP}^2 + 1.3878 \times \text{NOP} + 449.31 \\
Precharge & 3 FO4 & \text{NOP}^2 + 1.3878 \times \text{NOP} + 449.31 \\
\hline
\end{array}
\]

4.2 Validation

This section demonstrates that the underlying circuit implementations are reasonably tuned by comparing to published measurements for a commercial design. Accordingly, we used the same design methodology to implement a single-cycle renaming unit and then compared its latency to that of a commercial processor implementation. The latency was measured at 28FO4 for the specific 4-way RAT. By overlapping the decoding and wordline select drivers for RAT writes with the preceding RAT reads, this delay was further reduced to 23FO4. Adding up the pipelining overhead of 3FO4, this delay is comparable to the clock period of processors implemented in 130 nm technology that used single-cycle renaming (e.g., 800MHz SR71010B MIPS).

4.3 Two-Stage Pipeline Justification

Based on our full-custom implementation we developed a simple analytical delay model for single-cycle checkpointed, superscalar RATs. This model, shown in formulas (7) and (8), gives an upper bound on RAT read and write latencies as a function of NoGCs, WS, and I. In this model, NOP stands for the number of ports. We used this model to decide how many pipeline stages would be appropriate in the given technology. Specifically, given the delay estimated by (7) and (8) we decided to use a two-stage pipeline guided by the frequency of a high-end processor build in a similar technology. In particular, we estimated an upper bound (X) on renaming latency using the aforementioned models. We considered the clock frequency of the Pentium 4 processor implemented in a 130nm technology [11] as the target clock frequency (Z) that the specification is supposed to meet. Dividing the two and rounding (Floor (Z/X)) suggests a two-stage pipeline.

\[
\begin{align*}
\text{Delay}_{\text{Read}} &= 1.7543 \times e^{0.1524 \times \text{NoGCs} + 1.2777 \times \text{ln}(\text{WS}) + 0.0835 \times \text{NOP}^2 + 0.4908 \times \text{NOP} + 449.31} \\
\text{Delay}_{\text{Write}} &= 5.5818 \times e^{0.1322 \times \text{NoGCs} + 13.946 \times \text{ln}(\text{WS}) + 0.1664 \times \text{NOP}^2 + 1.3878 \times \text{NOP} + 449.31} \\
\text{NOP} &= 4 \times \text{I}W, \text{NOP} > 16
\end{align*}
\]

4.4 Latency Measurements

Finally, we report the latency measurements for the two-stage renaming and demonstrate how much faster it is compared to the single-cycle renaming. The latencies of the components in FO4 are given by (10). The critical path latencies of the first and second stages are as follows: (a) The critical path of the first stage comprises the latencies of the comparator, the NOR gate within the priority encoder, and the partial RAT read. The first stage latency measured at about 9.2 FO4. (b) The critical path of the second stage comprises the latencies of the partial RAT read delay, the RAT write excluding encoding and wordline select. This is measured at 9.5FO4. Accordingly, the logic delay in each stage is about 9.5 FO4. By adding up the pipelining overhead of 3 FO4, the overall pipeline depth, which defines the clock frequency, is about 12.5FO4. In summary, the latency results show that the processor’s logic depth is reduced from 23 to 9.5FO4 going from one to two-stage renaming.

References
