Two-Stage, Pipelined 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 (e.g., processor issue width, processor window size, and processor checkpoint count).**
Pipelining of the register renaming logic can help avoid restricting the processor clock frequency. This work presents a full-custom, two-stage register renaming implementation in a 130-nm fabrication technology. The latency of non-pipelined and two-stage, pipelined renaming is compared, and the underlying performance and complexity tradeoffs are discussed. The two-stage pipelined design reduces the renaming logic depth from 23 fan-out-of-four (FO4) down to 9.5 FO4.

***Index Terms*—Computer architecture, latency, map table,microproces-sors, pipelining, register alias table, register renaming.**

I. 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 among architectural (the logical register names, which are used by instructions) and physical registers (the physical storage elements' names, which hold the values at runtime). The RAT is a performance-critical component because it is read and updated by all instructions in order as they are decoded. Hence, the renaming unit must operate at the processor frequency, or it must be pipelined to avoid limiting the clock period. While previous work discussed the possibility of RAT pipelining [3], [7], none discussed how the renaming logic can be pipelined, and none measured how pipelining affects renaming latency. To fill this gap, this work investigates a two-stage renaming design in a commercial fabrication technology, analyzes the latency in terms of fan-out-of-four (FO4), and explains the underlying performance and complexity tradeoffs. This work also presents optimizations enabled by pipelining that reduce RAT energy (e.g., RAT reads are disabled when dependencies exist among co-renamed instructions). Latency results show that the logic depth per pipeline stage (which determines the processor's clock period [12]) of a 4-way superscalar processor is reduced from 23 FO4 to 9.5 FO4 using two-stage renaming.

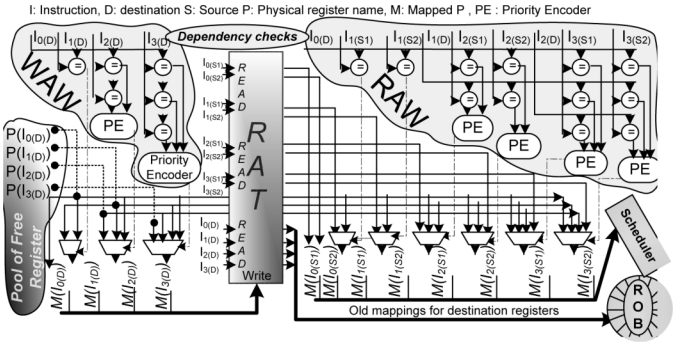

Fig. 1. Single-cycle superscalar register renaming.

II. Register Renaming Background

Renaming a single instruction proceeds as follows: *(i)* reading the physical register names for the source register operands; *(ii)* reading the current mapping of the destination register to be saved in the *re-order buffer* (ROB) for supporting speculative execution [1], [4]; and *(iii)* updating the RAT for the destination register with a physical register name allocated from the pool of free registers. Actions *(i)* and *(ii)* proceed in parallel, while action *(iii)* follows. Acquiring a free physical register name can be done at any time prior to action *(iii)*.

Superscalar processors simultaneously rename several instructions (called a *rename group*) per cycle. Hence, the renaming unit checks intra-group dependencies to handle intra-group write-after-write (WAW) and read-after-write (RAW) dependencies. To reduce latency, RAT reads proceed in parallel with dependency checks. All the RAT writes are performed, last taking into account WAW dependencies, so that only the latest update per destination register is saved. Fig. 1 shows the hardware block diagram of the 4-way superscalar renaming. The actions involved are as follows.

(i) RAT reads for the source $(\mathrm{I}_{0\ldots3(\mathrm{S}1\ldots2)})$ and destination $(\mathrm{I}_{0\ldots3(\mathrm{D}1\ldots2)})$ operands of the co-renamed instructions $(\mathrm{I}_{0\ldots3})$ are initiated.

(ii) In parallel with *(i)*, new physical registers $\mathrm{P}(\mathrm{I}_{0\ldots3(\mathrm{D}1\ldots2)})$ are allocated for all the destination registers.

(iii) The physical register names read in *(i)* do not reflect map changes from the co-renamed instructions. The names read are the ones left by the immediately-preceding rename group. 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 *(i)* and *(ii)*. When the RAT reads and the dependency checks complete, the comparison results are priority encoded and drive multiplexers selecting the most recent preceding physical register name assigned to each source architectural register.

(iv) WAW dependencies among co-renamed instructions must be detected (a) to write only the latest update per architectural register in the RAT and (b) to select the appropriate previous mapping to store in the ROB for each destination register. This mapping is either the RAT mapping read in *(i)*, or the physical register assigned to a preceding write within the same co-rename group in *(ii)*. As with RAW dependencies, these comparisons proceed in parallel with *(i)–(iii)*.

A. Checkpointed, SRAM-Based RAT

This work focuses on the SRAM-based RAT implementation, which is more scalable and energy-efficient for today's instruction window sizes (e.g., 128) than the CAM-based implementation [3]. This multi-port SRAM RAT has one entry per architectural registers. Each entry keeps a physical register name (e.g., 7 bits for 128 physical registers).

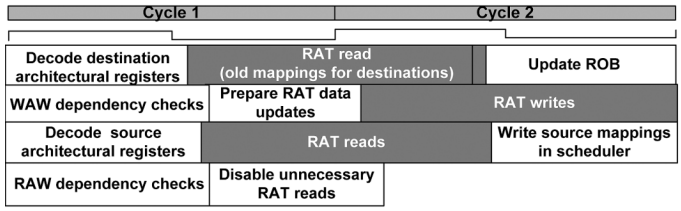

Fig. 2. Two-cycle register renaming.

Modern processors speculatively rename and execute instructions before certainly knowing that they should [1]. On mispeculations, erroneously-executed instructions are "squashed," i.e., any changes they made (including RAT changes) are undone. Modern RAT designs incorporate a set of *global checkpoints* (GCs) to recover from mispeculations. 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 is released when the instruction commits. To recover the RAT from a mispeculation, the GC is copied to the RAT. In addition to GCs, which provide fast recovery but are few as they are expensive to build, modern processors use the ROB, an instruction-by-instruction log of all changes done to the RAT. This work uses a mechanism that organizes GCs in small bi-directional shift registers embedded next to each RAT bit. This mechanism scales better than the alternative [6].

III. Pipelined Renaming

This section discusses a two-stage, pipelined renaming design as well as its latency and energy as a function of several parameters. This section also discusses the impact of deeper pipelining.

Figs. 2 and 3 depict the high-level block diagram of two-stage renaming and the actions that take place at each stage. We first explain how single instruction renaming can be pipelined, and then proceed to superscalar renaming. Fig. 3 shows a 4-way superscalar renaming unit. We assume instructions with two source and one destination register. 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 stage 1 decodes the source and destination registers of instruction *B*. The decoder outputs are latched so that they remain stable while reading from the RAT (starting at the second half of stage 1 and continuing into the first half of stage 2) and while writing the destination register's new mapping (starting at the first half of stage 2).

In parallel with decoding of the source and destination registers for *B*, dependency checking with *A* is performed. This parallel operation is feasible because dependency checking uses architectural register names of *A* and *B*. 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 ROB. This value is either *A*'s destination physical register in the case of a WAW dependency between instructions *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 start during the second half of stage 1 and complete 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, appropriate source and destination physical register 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*).

Pipelined superscalar renaming introduces the possibility of intra-group and inter-group dependencies among adjacent groups as

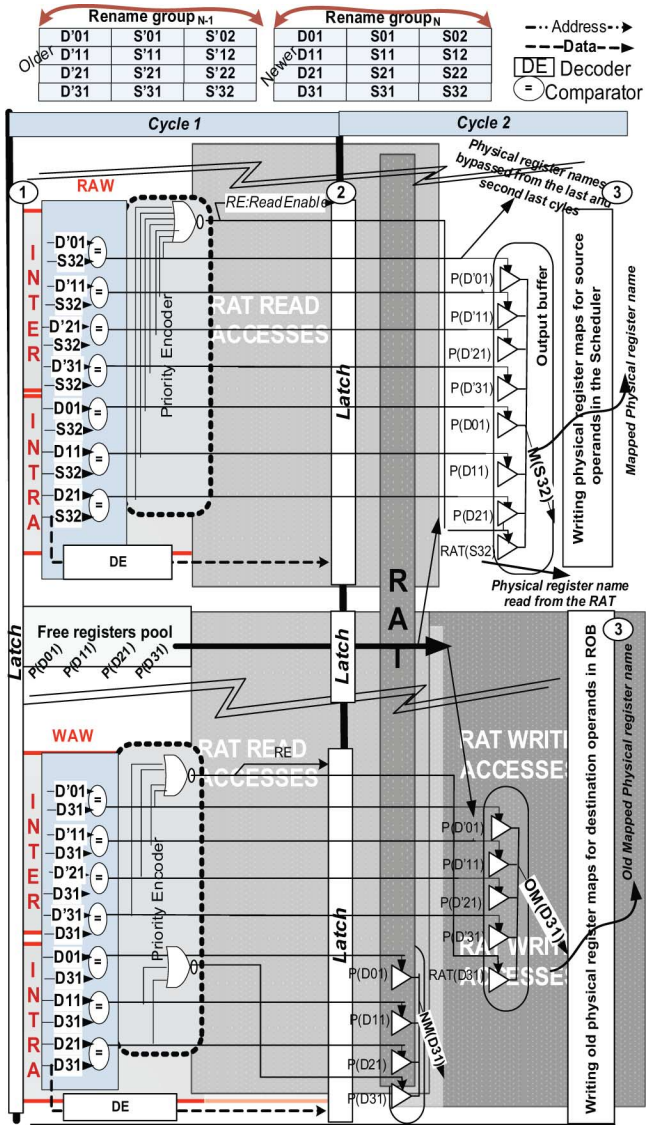

Fig. 3. Two-cycle register renaming implementation.

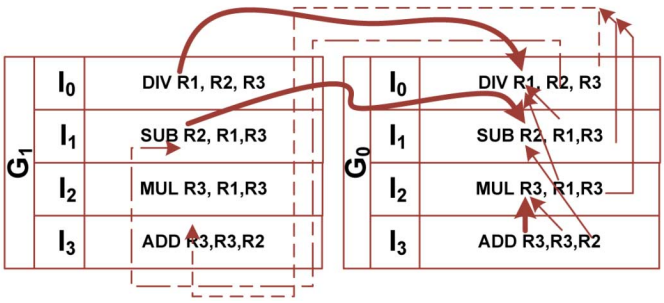

Fig. 4. Inter-group and intra-group dependencies.

Fig. 4 shows. These dependencies must be handled appropriately. 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-group and inter-group dependencies are detected so that the appropriate mapping is assigned to each source register. This mapping is either the register name that will read from the RAT, or the 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. The outputs of RAW dependency comparators outputs are priority encoded (design is detailed in [9]) to select the most recently-assigned physical register name for the architectural source operand during stage 2.
2) Starting at the beginning of stage 1, the source architectural registers are decoded and latched to remain stable during the RAT reads. The RAT reads, extending over both stages, start after decoding. The source's map RAT read is required if no RAW dependency exists. Hence, once the RAW dependency checks are finalized, unnecessary RAT reads can be stopped to reduce energy consumption. Such optimization is desirable because sense amplifiers are a major source of RAT's energy consumption [6]. The comparators' outputs are NORed to activate the sense-amplifier logic of the associated bitlines. If one of the comparators detects a match, NOR's output becomes zero to deactivate the sense amplifiers. Decode and bitline precharge latencies are overlapped with comparator and NOR latencies.
3) Up to issue width (IW) free physical registers are assigned to the destination architectural registers by the end of stage 1. The names are latched and used in stage 2. These names are provided by the register free list, implemented by a circular first-input–first-output (FIFO) capable of providing up to IW 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 attempt write to the same architectural registers within the current group; accordingly, only the latest update should be recorded into the RAT. Inter-group WAW dependencies must also be detected so that the appropriate values are stored in ROB. The WAW dependency check results are used to prepare data for RAT updates. To reduce energy, unnecessary RAT reads for old mappings can be stopped once a WAW dependency is detected.
5) Starting at the beginning of stage 1, the destination architectural register addresses are decoded and latched to remain stable during the RAT reads for the purposes of ROB checkpointing and updating the RAT with new mappings. Once decoding is completed, RAT reads are initiated for reading the old mappings to record them in ROB. RAT updates start once the old map reads complete.

In single-cycle renaming, the SRAM latching delay is completely exposed. In the pipelined implementation, part of the SRAM latching (see Fig. 5) is hidden by the pipeline latches. Referring to Fig. 5, RAT accesses are pipelined as follows:

(I) during clk1-high, the address is decoded and latched, and the bitlines are also pre-charged;
(II) for writes, in clk2-high, the write input data are latched and the write transaction happens;
(III) for reads, in clk1-high, the bitline voltages start to swing and sensed by sense-amplifier whose output is then latched;
(IV) for a write followed by a read for the old mapping, because the read data latch signal defines the end of the read operation, it also triggers a write operation.

A. Scalability of Pipelined Register Renaming

This section discusses the components along the critical path in two-stage renaming, and describes 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 $\mathrm{Log}_2(number\ of\ architectural\ registers)$. The maximum number of comparators needed per source operand is $(2 \times \mathrm{IW} - 1)$. Increasing IW increases the number of comparators. However, all comparators work in parallel, hence

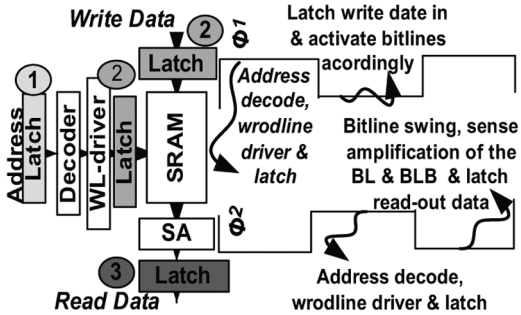

Fig. 5. Pipelined SRAM's timing.

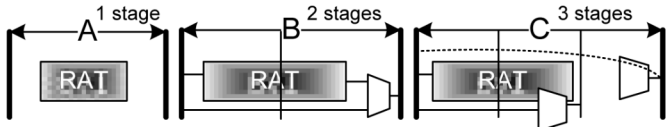

Fig. 6. Multistage renaming.

increasing IW does not affect the comparison latency, but noticeably increases power.

(ii) **Priority Encoder**: Increasing IW increases the latencies of the IW-input priority encoder and the $(2 \times \mathrm{IW} - 1)$-input NORs. The encoders are driven by the comparators of (i).

(iii) **Output Multiplexers**: The width of each multiplexer is $\log_2(\mathrm{WS})$. The maximum number of multiplexers needed per source operand is $(2 \times \mathrm{IW})$. Increasing IW increases the number of pass gates; however, they run in parallel, so increasing IW does not affect the multiplexer delay significantly, although it noticeably increases power.

(iv) ***RAT***: Non-checkpointed RATs are essentially multi-ported register files. RAT access latency grows linearly with entry count and quadratically with port count. The RAT size and power grow linearly with the entry count and faster than linearly and slower than quadratically with port count. For checkpointed RATs, previous work has determined quantitatively how RAT latency and energy vary as a function of WS, IW, and GC count [6]. Increasing WS increases RAT entry width, and hence increases delay logarithmically. The latency increases exponentially with increasing number of GCs (NoGCs). Increasing IW increases latency quadratically as it increases port count [6].

B. Deeper Pipelining

Using a methodology similar to that used for two-stage pipelining, 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. After the RAT reads complete, a set of multiplexers selects the appropriate value. For example, Fig. 6 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, making them slower. In general, the deeper the pipeline, the wider and slower the final multiplexers need to be. Hence, there is a point past which this delay becomes detrimental to performance.

While deeper pipelining can improve performance by providing a faster clock, it suffers from performance-degrading side effects. Due to the extra pipeline stages that have to be refilled, deeper pipelining in the front-end increases mispredicted branch penalties and instruction cache miss penalties. Hence, deeper pipelining will be useful only if performance improvement it provides (faster clock) surpasses its side effects [7]. Executing dependent instructions in consecutive cycles is needed for high performance, especially for programs with limited parallelism; otherwise, pipeline bubbles can cause significant performance loss.

C. Other RAT Latency Reduction Techniques

Aside from pipelining register renaming, renaming latency can be reduced in several ways. First, the RAT can be duplicated to reduce the number of ports on each RAT copy reducing latency to some extent. Second, considering that not all instructions have two operands and that co-renamed instructions are highly likely to have common operands, the RAT port count can be reduced at the cost of insignificant IPC performance loss. However, this requires a routing network whose latency may negate any of the benefits. Third, using GC prediction and selective GC allocation, the number of RAT GCs can be reduced while maintaining performance [1]. Pipelining is orthogonal to all aforementioned techniques. Given a specific design, any or all of the above techniques can be used to reduce latency to the desired level if possible.

IV. Evaluation

Section IV-A details the design and evaluation methodologies. Section IV-B validates that our underlying circuit implementations are reasonably tuned. Section IV-C presents simple empirical models for RAT delay and uses the models to justify why a two-stage pipeline is an appropriate choice for this technology. Finally, Section IV-D analyzes the latency of the pipelined renaming unit.

A. Methodology

We developed full-custom layouts using the Cadence tool set in a commercial 130 nm technology with a 1.3 V supply voltage (the best technology available to us at the time of these experiments). The key observations can apply, for the most part, to smaller technologies. The main differences will be in the relative importance of dynamic versus static power consumption.

We implemented a 4-way superscalar RAT, a width representative of the processors built in the given fabrication technology and of many modern processors. We assume 64 and 128 architectural and physical registers respectively, common sizes today. We also assume that WS and the number of physical registers are the same as the exact implementation of the scheduler and the window is orthogonal to this study. We assume a RAT with 16 GCs since it has been shown that with GC prediction, at most 16 GCs are sufficient to achieve performance close to what is possible with infinite GCs [1].

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 four write ports using CACTI 4.2 [8]. We used Spectre for circuit simulations, and we report worst-case latency results.

For delay analysis, we used the FO4 delay metric as estimated by formula (1). For the 130-nm technology, FO4 is about 40 ps. 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 metric) optimized designs [12]. Given by (2), the logic depth/computation delay is calculated by subtracting the pipeline logic depth from the pipelining clock overhead including clock skew and jitter, and latch delay. Clock overhead including margins for setup time and hold time is thus estimated at 4–6 FO4:

(i) pulse-mode Latch 1–1.5 FO4/flip-flops 2–3 FO4;

(ii) skew 2–4 FO4;

(iii) jitter 1–2 FO4. (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 using techniques such as pulsed clocks and/or direct domino pipelines. Accordingly, a 3 FO4 pipelining overhead is a reasonable approximation [12].

B. Validation

This section demonstrates that the proposed circuit implementations are reasonably tuned by comparing the results to published measurements for a commercial design. Unfortunately, to the best of our knowledge no public results exist for a two-stage pipelined implementation. Accordingly, we used the same design methodology to implement a single-stage, non-pipelined renaming, and then we compared its latency with the latency of a renaming unit for a commercial processor implementation. The latency was measured at 28 FO4 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 23 FO4. With a pipeline overhead of 3 FO4, this delay is comparable to the clock period of processors implemented in 130-nm technology and considered single-cycle renaming (e.g., 800 MHz SR71010B MIPS), shown as follows.

FO4 Delay Estimation:			
(360-500) ps/µm × (0.13/L_{drawn}), $L_{drawn} = 0.7\ L_{eff}$			(1)
Pipeline depth	**Pipeline overhead**	**Pipeline length**	(2)
10-14 (FO4) = 400- 560ps	3 FO4 = 120ps	7-11 FO4 = 280-440ps	

C. Two-Stage Pipeline Justification

Based on our full-custom implementation, we developed a simple empirical delay model for single-cycle checkpointed, superscalar RATs. To develop the empirical models, we applied curve fitting over 60 data points, based on the measurements from our physical-level implementations in a 130-nm technology. We present the latency models for the 4-way RAT; Equations (3) and (4) estimate read and write latencies. In this model, NOP stands for the number of ports, WS for window size, NoGCs for the number of GCs, and IW for the issue width. The worst-case relative error of the model predictions is on average within 7.8% of the Spectre circuit simulation results for the data points we used for curve fitting ($0 <= WS <= 512$, $0 <= NoGCs <= 16, IW = 4$). These data points cover values of the WS, IW, and NoGCs that would be typical for modern processors, and model predictions have a rage of accuracy that is acceptable for architectural-level studies. For predicting the delay of the configurations outside of this range, extrapolation can be used. However, the extrapolation results are often subject to greater uncertainty.

The model given by (3) and (4) provides an upper bound on RAT read and write. We used this model to decide how many pipeline stages would be appropriate for the given technology. In particular, we first estimated an upper bound (X) on non-pipelined renaming latency. We then considered the clock frequency of the Pentium 4 processor implemented in a 130-nm technology [11] as the target clock frequency (Z) that the specification is supposed to meet. Diving the two and rounding Floor (Z/X) suggests a two-stage pipeline as follows.

$$Delay_{Read} = 1.7543 \times e^{0.1524 \times NoGCs} + 2.77 \times \ln(WS) + 0.0835 \times NOP^2 + 0.4908 \times NOP + 449.31 \quad (3)$$

$$Delay_{Write} = 5.5818 \times e^{0.1322 \times NoGCs} + 13.946 \times \ln(WS) + 0.1664 \times NOP^2 + 1.8378 \times NOP + 449.31 \quad (4)$$

$NOP = 4 \times IW \qquad NOP > 16$

Number of source registers per instruction: 2

Number of destination registers per instruction: 1

D. Latency Measurements

Finally, we report the latency measurements for the two-stage renaming and show how much faster it is compared to the single-cycle renaming. The latencies of the components in FO4 are given by (5). The critical path latencies of the first and second stages are as follows.

Component Delay		(5)
6-64 decoder and wordline driver	5 FO4	
Latch delay	2.5 FO4	
Wordline activation to sense-amp output	2.4 FO4	
Wordline activation to write in data cell	2.8 FO4	
8-bit comparator	3 FO4	
8-bit dynamic NOR	3 FO4	
Output multiplexer	2.5 FO4	
Precharge	3 FO4	

(i) 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 is about 9.2 FO4.

(ii) 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 9.5 FO4. 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, defining the clock frequency, is about 12.5 FO4. In brief, the latency results show that the processor's logic depth is reduced from 23 FO4 to 9.5 FO4 going from one to two-stage renaming.

V. Conclusion

The RAT, the core of register renaming, is a performance-critical component of modern processors because it is read and updated by all instructions in order as they are decoded. Register renaming latency increases as a function of the issue width, window size and number of recovery checkpoints. The RAT is on the processor's critical path and can limit the processor clock frequency. To avoid restricting of processor's clock frequency, pipelining of the register renaming logic is a solution. This work presents a full-custom, two-stage register renaming implementation in a 130-nm fabrication technology. Latency analyses show that the renaming logic depth is reduced from 23 to 9.5 FO4 going from single-stage to two-stage renaming for a 4-way processor. In our two-stage pipelined implementation, pipelining enables the early termination of RAT reads to reduce power consumption. Furthermore, this work contributes to other proposed architectural techniques that consider pipelined renaming as their requirement [4], [7].

References