

# Design and Implementation of a 100 MHz Reorder Buffer

Steven Wallace, Nirav Dagi, and Nader Bagherzadeh

Department of Electrical and Computer Engineering  
University of California, Irvine  
Irvine, CA 92717  
swallace, niravd, or nader@ece.uci.edu

## Abstract

A reorder buffer is an important part of a superscalar microprocessor architecture. It allows out-of-order issue and completion of instructions, which contribute to overall performance enhancement. A custom layout of a 32-entry reorder buffer was implemented using a 1.0  $\mu\text{m}$  (drawn) triple-metal CMOS technology. It measures 2 mm by 2.85 mm. The design was verified using several benchmark programs.

## Introduction

Growing demand for high performance computing has resulted in the development of advanced microprocessors based on superscalar architectures [1]. One of the main forces behind improving performance in a superscalar architecture is the ability to identify independent instructions (at the assembly level) and execute them in parallel; i.e., instruction-level parallelism. A large number of current microprocessor designs are aimed at utilizing this form of concurrency.

At UC Irvine, researchers have been designing a superscalar architecture called SDSP (Superscalar Digital Signal Processor) [2]. One of the main components of this design is the scheduling unit which consists of an instruction window, a reorder buffer, and a register file. The reorder buffer contains the lookahead state, while the register file maintains the in-order state [3]. One function of a reorder buffer is to rename the destination identifier (register) to a unique tag identifier; i.e., register renaming. The result from a functional unit uses its tag to write to an allocated entry. The reorder buffer updates the register file with the results in the original program order by operating the reorder buffer in first-in, first-out (FIFO) fashion.

The current design outperforms the previously published reorder buffer in several features [4]. First, the new design decodes four instructions instead of two and has a total of thirty-two entries instead of eight. Secondly, the current design runs at 100 MHz as oppose to 20 MHz, using a smaller, three metal technology. Thirdly, instead

Table 1: Reorder Buffer Fields

Name	Qty	Type	Description
DATA	32	8-port RAM	Holds 32-bit data value or 5-bit tag.
READY	1	8-port RAM	Indicates if data or tag is in DATA field.
DEST	10	4-port CAM	Holds destination register and compares source registers.
TAG	5	8-port CAM	Holds destination tag and compares against result tags.
CURRENT	8	lookup	Searches for most current destination register.
VALID	1	special	Indicates if entry is valid.

of a current bit cell, a lookup array is used to dynamically determine the most current entry. This has performance benefits in handling mispredicted branches, since selected entries may be invalidated. Finally, the size of all cells was reduced dramatically by reducing the number of transistors per cell and by combining the shift and storage portions of each cell.

## Cell Architecture

The reorder buffer is composed of a destination register field, data field, ready field, destination tag field, current field, and valid field. The name, quantity, type of cell CAM = Content Addressable Memory, RAM = Random Access Memory), and description of each field are given in Table 1.

Figure 1 is the floor plan of the reorder buffer. It consists of 32 identical rows/entries, except for the lookup array. A group of four entries is classified as a block. When the reorder buffer shifts, each block shifts down by one (effectively, each entry shifts by four). As a result, the block at the top accepts new destination identifiers, while the block at the bottom commits its results to the register file.

The center of the reorder buffer contains 33 bit-slices of data cells (8-port RAM cells). One cell is used for the ready bit. The remaining 32 cells hold a 32-bit result or a 5-bit tag. Initially, the 5-bit tag is shifted into 5 cells

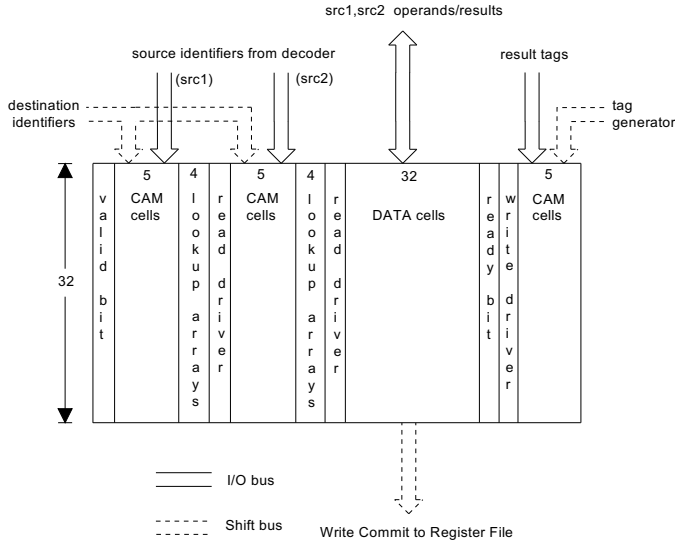


Figure 1: Floor Plan of the Reorder Buffer

of the DATA field, and a ‘0’ is shifted into the ready bit. When the result arrives, a ‘1’ is written into the ready bit, and it will overwrite the 5 bits of tag, since it is no longer needed for decoding. By storing the tags in the data cells, the need for a dual purpose CAM and RAM cell has been eliminated.

To the right side of the data cells, CAM cells are used for comparing result tags for writing, while CAM cells to the left of the data cells are used for comparing source identifiers to destination identifiers. Also to the left are special lookup cells and a valid bit that are used in conjunction with the CAM cells for evaluation. The destination CAM section is split into two identical pieces, each evaluating four operands, so that its height would be smaller to match the height of the data cells and save area overall.

### Shift Cell

Since each entry in the entire reorder buffer shifts by four each cycle (unless it stalls), every type of cell must be able to store its information and shift it to the fourth entry below. Routing the data through three cells below is accomplished using the third level of metal. Shifting and storage are both handled with a single modified D flip-flop, as shown in Figure 2. Normally, the *shift* control signal is low, and the first transmission gate sets up the input for shifting. When *shift* goes high, this gate is turned off and the next gate opens and updates the *data* output. After *shift* returns low in the next phase, the feedback transmission gate opens to maintain its state, since the reorder buffer could stall (by keeping *shift* low) for an indeterminate amount of time.

### Data Cell

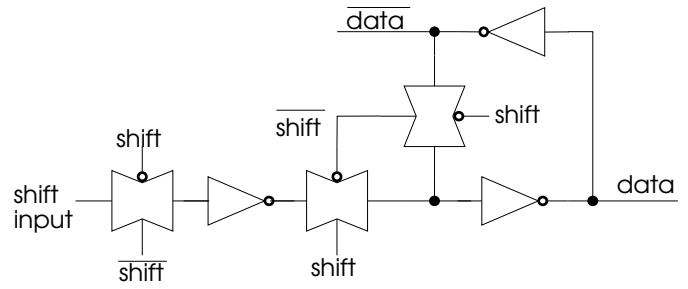


Figure 2: Shift/Storage Cell

The entire reorder buffer is designed with consideration to the data cell, both in size and in timing. Figure 3 is the data cell, which allows both reading and writing to its storage cell through eight different ports. Each port has its own n-transistor and *match* control signal. Four ports access *data*, while the other four ports access  $\overline{data}$ . Consequently, eight operands may be read (four from *bit* and four from  $\overline{bit}$ ), and four results may be written (each result on *bit* and corresponding  $\overline{bit}$ ).

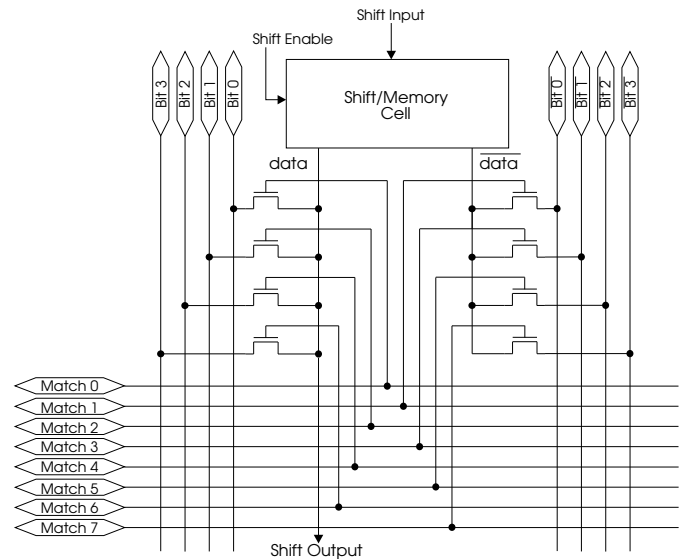


Figure 3: Data Cell

### Valid Cell

In case of a mispredicted branch or an exception, the entries in the reorder buffer following that instruction need to be invalidated. This control is achieved with the help of the valid cell which is a resettable shift/storage cell. If an entry is invalid, then all its *match\_dest* lines are discharged, irrespective of a match. This ensures the operands are read only from a valid entry.

### CAM Cell

A four-port CAM cell, shown in Figure 4, is used to de-

termine which destination entries or tag entries match for reading or writing. Each match line is initially discharged and the register or tag used for comparison is put on its corresponding *bit* and *bit* lines. To evaluate, each match line uses an active p-transistor that attempts to charge it. If all bits match, then all the n-transistors in the CAM cells will be open and the match line will charge. On the other hand, if one or more bits do not match, then the n-transistors will pull the match line close to ground since the transistor gain factor of the pull-down transistors is six times that of the pull-up transistor [5]. Using this method, only entries that match will charge up the match lines. Alternatively, if a precharge method is used, then additional precharge control transistors would be required for each cell, and more importantly, an additional synchronization step would be required between the destination CAM cells and the lookup cells. Therefore, we chose the former method at the cost of power consumption.

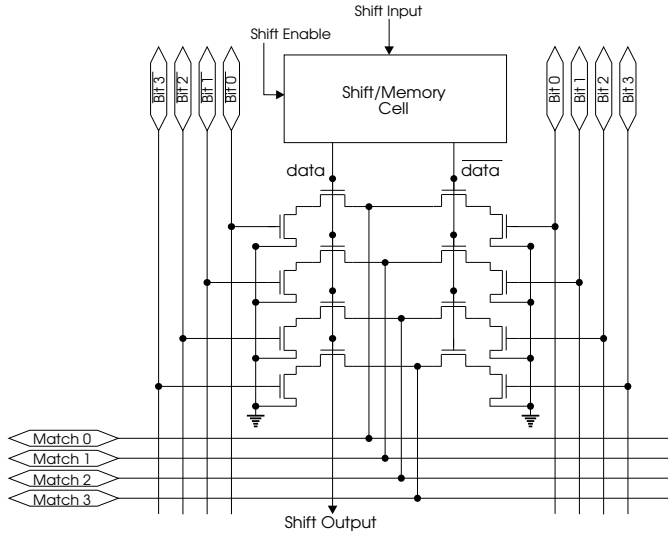


Figure 4: CAM Cell

### Lookup Cell

More than one destination CAM cell can match a source register identifier. The value or tag desired comes from the cell matching the most current destination identifier, which is closest to the top of the reorder buffer. All other matching entries below it must be discharged. Therefore, we designed a circuit that performs this task in constant time with respect to the number of entries,  $n$ , and requires  $\lg n$  space.

Let  $m = \lg n$ ,  $i = 0 \dots n - 1$ , and  $j = 0 \dots m - 1$ . If  $n = 32$ , Figure 5 shows the schematic for the  $i$ th entry (starting from the top). In the schematic, there are circles with a variable inside them. If the variable evaluates to a value of 0, then the circle represents a short circuit. Otherwise, it is an open circuit, and the connecting transistor need

not be present. These variables,  $p$  (precharge lookup) and  $d$  (discharge lookup), conform to the following equations:

$$p_{i,j} = i \bmod 2^{j+1}$$

and

$$d_{i,j} = \left\lfloor \frac{i \bmod 2^{j+1}}{j+1} \right\rfloor.$$

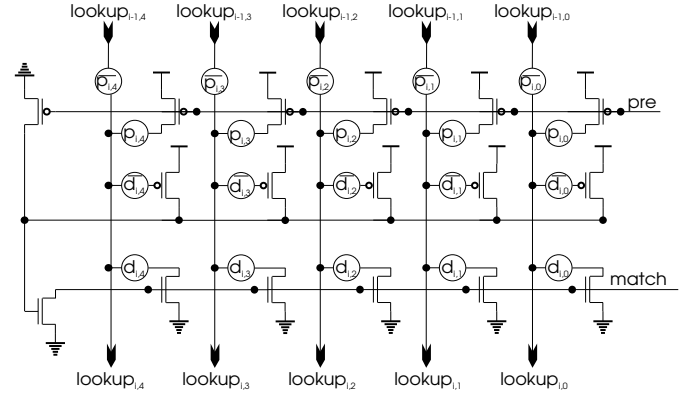


Figure 5: Lookup Cell in the  $i$ th Row

### Driver Cells

Figure 6 shows the read and write drivers used in the design for driving the *match\_local* signal. The *match\_local* signal represents a control signal that opens up the corresponding ports on all the data cells in that specific row for read/write operation. The two drivers function as a distributed multiplexer, driving either the *match\_dest* signal (during read phase) or the *match\_tag* signal (during write phase) onto the *match\_local* signal, and keep the *match\_local* discharged during other phases. The extra n-transistors controlled by *write* and *read* are needed to assure when one driver is trying to pull-up *match\_local*, the other does not attempt to pull it down.

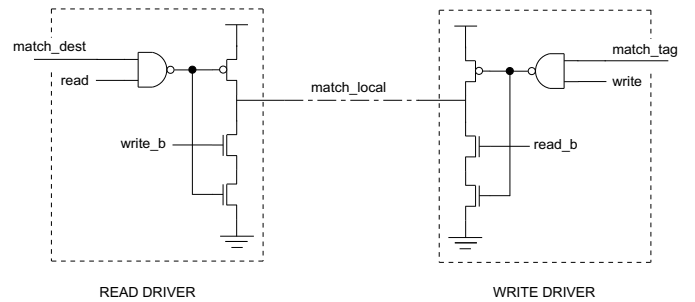


Figure 6: Driver Cells

Table 2: Phase Timing

Phase	Field	Operation
1	TAG CURRENT DEST	<i>Match_tag</i> lines evaluate. Lookup lines precharge. Dest fields shift by four; new destination identifiers are accepted.
2	DATA READY DEST CURRENT	Results write into matching entries. Matching entries are set. <i>Match_dest</i> lines evaluate. Lookup lines evaluate and discharge appropriate <i>match_dest</i> lines.
3	DATA, READY TAG	Ready and Data fields shift by four. I/O buses precharge for reading. Tag field shifts by four.
4	DATA READY	Data reads from matching entries. Ready bits are read.

### Timing

A four phase clock generates all the necessary control signals for the reorder buffer. Table 2 shows the operation of each part during each phase. Figure 7 is a timing diagram of signals in the reorder buffer, including the worst-case times of some signals. The first phase precharges the lookup lines for evaluation, and the destination fields shift, thereby creating new entries for new instructions. The second phase writes result values into the data cells and evaluates the *match\_dest* lines for reading in the fourth phase. In the third phase, I/O buses on the data cells are precharged for reading, and the rest of the reorder buffer shifts. During the fourth and last phase, the values/tags and the ready bit are read from the reorder buffer. Since evaluation of *match\_dest* immediately precedes reading, the destination register field can not shift at the same time as the other fields. Therefore, it shifts two phases earlier.

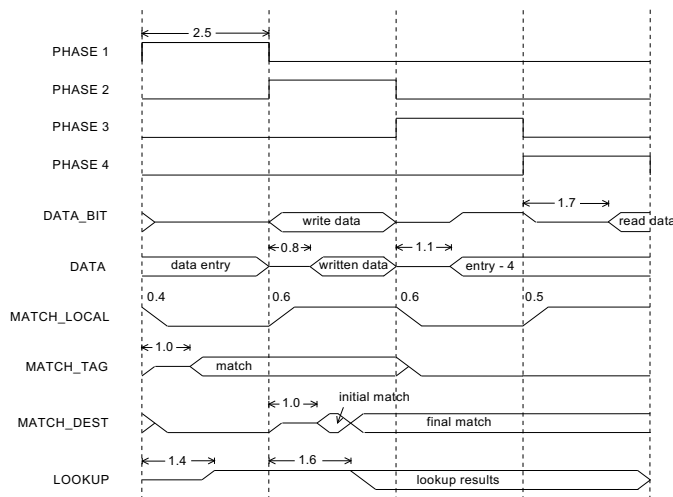


Figure 7: Timing Diagram (nsec)

The last phase is the critical phase since it takes 1.7 ns to discharge four source operands from the same entry. Since no amplification is used, it is probable that this time could be improved. Although a much faster clock could be used, a 100 MHz clock is chosen.

For design verification, the help of the SDSP simulator developed at UCI was sought [2]. Various benchmarks were run on the simulator, and the inputs to the reorder buffer were converted to IRSIM stimulus. For the first 1,000 cycles of each program, the layout was simulated, and the resulting outputs were successfully compared against the expected outputs from the SDSP simulator.

### Conclusion

In summary, we presented our design and implementation of a 32 entry reorder buffer capable of decoding four instructions per cycle. It was verified by simulation on several different benchmarks and can tolerate a 100 MHz clock rate with considerable safety margins.

### References

- [1] Mike Johnson, *Superscalar Microprocessor Design*, Prentice Hall, Englewood Cliffs, 1991.
- [2] Steven Wallace, "Performance Analysis of a Superscalar Architecture," Master's thesis, University of California, Irvine, 1993.
- [3] J. E. Smith and A. R. Pleszkun, "Implementing Precise Interrupts in Pipelined Processors," *IEEE Transactions on Computers*, C-37:562-573, May 1988.
- [4] J. Lenell, S. Wallace, and N. Bagherzadeh, "A 20 MHz CMOS Reorder Buffer for a Superscalar Microcomputer," *4th Annual NASA VLSI Symposium*, November 1992.
- [5] Neil Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, Reading, MA, 1993.