A 20MHz CMOS Reorder Buffer for a Superscalar Microprocessor

John Lenell
Standard Microsystems

Steve Wallace and Nader Bagherzadeh
Department of Electrical and Computer Engineering
University of California, Irvine
Irvine CA 92717

Abstract- **Superscalar processors can achieve increased performance by issuing instructions out-of-order from the original sequential instruction stream. Implementing an out-of-order instruction issue policy requires a hardware mechanism to prevent incorrectly executed instructions from updating register values. A reorder buffer can be used to allow a superscalar processor to issue instructions out-of-order, and maintain program correctness. This paper describes the design and implementation of a 20Mhz CMOS reorder buffer for superscalar processors. The reorder buffer is designed to accept and retire two instructions per cycle. A full-custom layout in 1.2 micron has been implemented, measuring 1.1058 mm by 1.3542 mm.**

1 Introduction

A superscalar processor can improve performance by looking ahead into the sequential instruction stream, and executing independent instructions out-of-order. However, issuing instructions out-of-order can cause the processor to execute instructions which should not have been executed. For example, a branch instruction in the instruction stream is predicted by the processor to be taken. The processor begins to execute instructions from the target of the branch before the actual direction of the branch is determined. If the branch is determined to be not taken, then the instructions from the target of the branch have been executed incorrectly. For the program to complete correctly, the results of the instructions which have been executed incorrectly must not write results to the register file. Program correctness can be maintained by only allowing instructions to update the register file in the original program order. A reorder buffer allows a superscalar processor to execute instructions out-of-order by allowing the register file to maintain the in-order state of the processor[?, ?]. The reorder buffer temporarily holds all results computed by instructions after execution regardless of the order of execution, and then updates the register file with the results in the original program order. Results are written to the register file in-order by operating the reorder buffer in FIFO fashion. As entries in the reorder buffer reach the bottom of the FIFO, the completed results are written to the register file[?].

The organization of a reorder buffer in a superscalar processor is shown in Figure 1. Each decoded instruction is allocated an entry at the top of the reorder buffer. During allocation, the instruction's destination register identifier and a unique tag identifier for the instruction

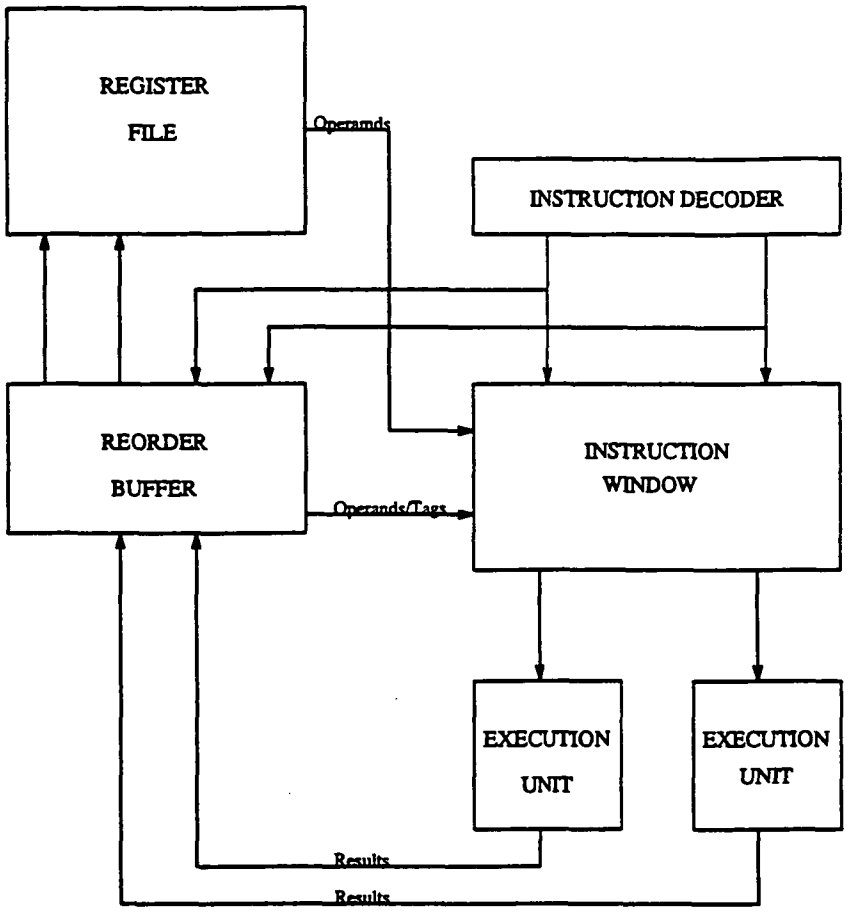

Figure 1: Location of a Reorder Buffer in a Superscalar Processor

the operand value, or neither, indicating that the value must be read from the register file. After the functional unit receives its operands and executes, the result is stored in the result field of the reorder buffer. When a completed instruction reaches the bot

2 Design

The reorder buffer presented here has been designed for a superscalar processor with a two instruction fetch unit, and two execution units which execute concurrently. The reorder buffer allows the instruction decoder to enter two instructions into the buffer, read the tag or data for four source registers specified by the instructions in the decoder, and write the results for two functional units each cycle. In terms of processor pipeline stages, the reorder buffer supports the instruction decode, write back, and register update stages.

2.1 Operation

During instruction decode, the reorder buffer is associatively searched with the source register identifiers of each instruction in the decoder. The source register identifiers are compared with a field of the reorder buffer which holds the destination register identifier of preceding instructions. If a match is found, three control fields of the reorder buffer are used to further

search of the completed instructions tag identifier with the tag field of the buffer. A match enables a write into the data field of the corresponding buffer entry, and sets the Ready bit.

As instructions reach the bottom of the reorder buffer, they are written to the global register file during the register update stage. Register update is controlled by the Ready and Valid bits of the entries at the bottom. If both bits are set, the contents of the data field can be written to the register file to the location specified by the destination register field.

2.2 Configuration

A block diagram for the reorder buffer is shown in Figure 2. The reorder buffer is split into two banks. Each bank contains four reorder buffer entries. Each entry has the following fields:

Destination Register Identifier This field specifies the true destination register of the instruction before register renaming takes place.

Destination Register Tag This field holds the renamed value of the destination register. Each Tag in the reorder buffer is unique so that every decoded instruction can be located by its tag.

Data This field temporarily holds the results of the corresponding instruction which have been computed by the functional units.

Valid This is a single bit field that defines whether the associated reorder buffer entry contains valid information. This field is set when instructions are entered into the reorder buffer, and cleared if the entire reorder buffer needs to be invalidated (ie. mispredicted branch).

Current This is a single bit field which tracks the most recent instruction to update a destination register. As each destination register is entered into the reorder buffer, this bit is set for the entry, and the current bit of any other entries with the same destination register are reset.

Ready This is a single bit field that is set when an instruction entry in the reorder buffer has completed execution by a functional unit. It signifies that the result of the corresponding instruction is ready to write back to the register file.

One entry from the instruction decoder can be written to each bank every cycle. The first instruction in the decoder is always written to the first reorder buffer bank, and the second instruction is written to the second reorder buffer bank. One result can be read from each bank every cycle from the bottom of the reorder buffer. and the entries will be shifted out of the reorder buffer if a shift is requested by the instruction decoder for incoming instructions. FIFO operation of the reorder buffer is achieved by shifting all the fields of each reorder buffer entry down one row. Both of the banks shift together so that when entries reach the bottom, they are paired with the same entry with which they entered the reorder buffer.

Phase	Function	Description
1	Precharge	The match lines of the content addressable memory cells are precharged prior to evaluation.
	Evaluate	A self timed signal turns off precharge mid-phase, and enables evaluation of the cam cells.
2	Read	Valid, Ready, and Tag or Data is read for each source register identifier in the decoder after match lines have evaluated. Data is read from the bottom of the reorder buffer and written to the register file if the Ready bit is set.
3	Precharge	The match lines of the CAM cells are precharged prior to evaluation.
	Shift	The entire reorder buffer shifts in a FIFO fashion. New entries are shifted into the top of the FIFO.
	Evaluate	Evaluation follows precharge.
4	Write	Results are written from the functional units to the reorder buffer for the entries whose tags match the tags of the completed instructions. Ready bits are set for entries which receive results. Current bits are reset for entries which no longer contain the most recent update to a destination register.

Table 1: Functions of Timing Phases

2.3 Timing

A four phase clock is used to trigger the multiple functions that the reorder buffer must perform each cycle. The functions performed during the four phases are given in Table 1. Figure 3 is the timing diagram of the four phases. An additional timing signal is generated by the circuitry to trigger two events in a single phase. This self-timed signal is generated by the precharging match lines of the content addressable memory cells. Precharge begins at the start of phase one and three. When precharge has completed, the signal turns off the precharge drivers and enables evaluation of the CAM cells. This self-timed signal is important since precharging the match lines is completed under a nanosecond. Otherwise, if two more phases are used, the cycle time would increase dramatically.

3 Implementation

The regular structure of the reorder buffer is ideal for a full custom implementation. Custom cells were designed for each field of the reorder buffer described in the previous section. A total of six custom cells were used with minor variations occurring within cells depending on the placement of the cell in the buffer. Additionally, a self timed signal is generated by a custom circuit to enable evaluation after precharge in phases one and three. A universal shift cell was designed to allow data to shift between memory elements.

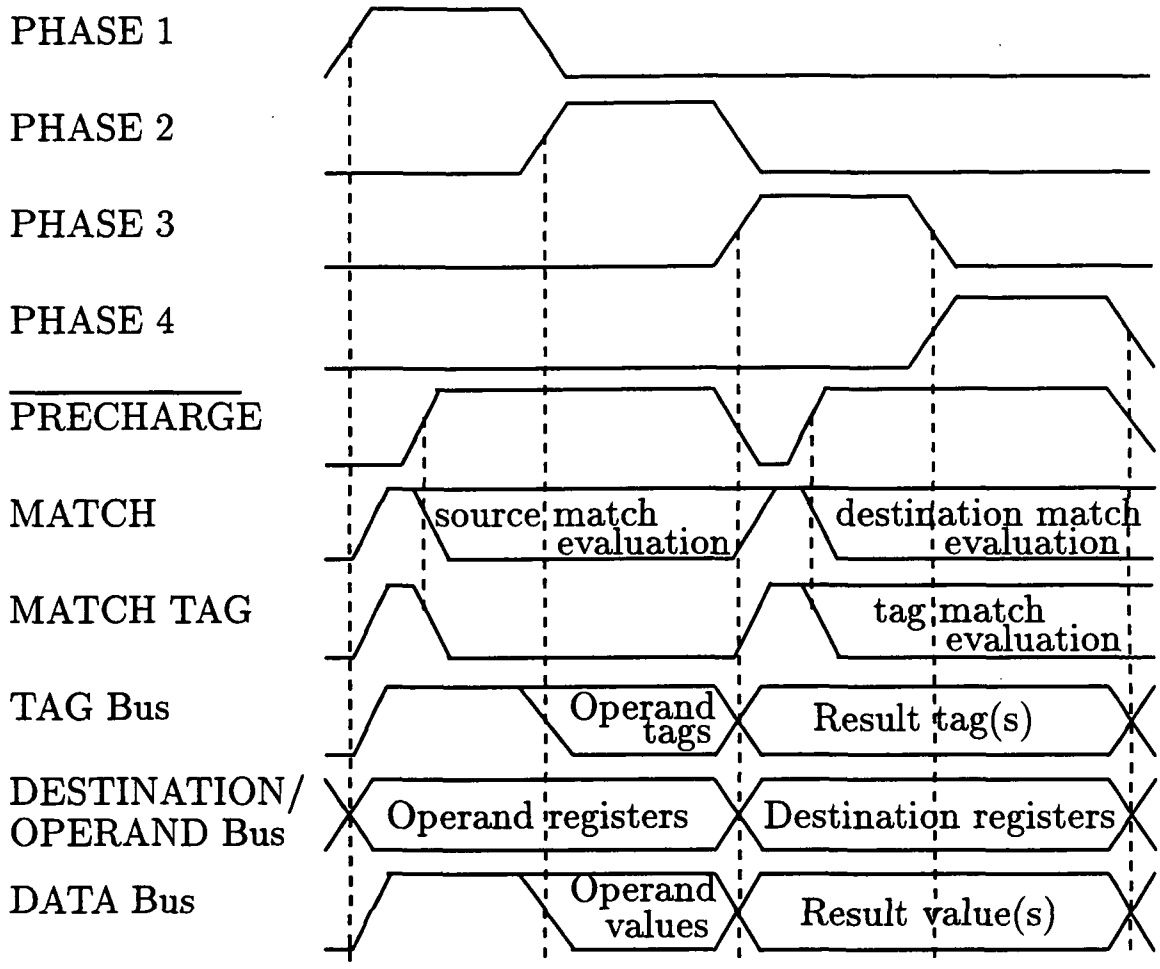

Figure 3: Timing Diagram of Reorder Buffer

3.1 Destination Identifier Cell

The destination identifier field of the reorder buffer is implemented with four-port CAM cells[4]. The four port CAM cell is able to evaluate four match signals simultaneously. The schematic of the circuit are shown in Figure 4(a). Operating the four port CAM cell involves precharging the four match lines and placing the value to be addressed on the bit lines. During precharge, the discharge paths for the match lines are off to prevent evaluation before the bit lines have settled. After precharge, if the value on a bit line does not match the content of the cell, the corresponding match line is discharged. The match lines of the CAM cell in the reorder buffer are precharged and evaluated twice each cycle. During the evaluate phase, the four source register values from the instruction decoder are placed on the bit lines of the CAM cell, and the match lines select the data value to be read from the reorder buffer. Then, in the write phase, the destination register of bo

3.2 Current Cell

The current cell is placed physically adjacent to the four-port CAM cell and shares the four match lines of the CAM cell. The current cell performs three functions each cycle: read, write, and reset. During the read phase, the current cell discharges all of the match lines passing through the cell if the value of the cell is logically zero. This prevents matches of duplicate destination registers to a single source register by allowing only the most recent destination register entry in the reorder buffer to match. Write access is provided to the current cell to provide shifting capabilities between reorder buffer rows. Reset of the current cell occurs during the write phase if match0 or match1 stay high. The schematic of the current cell is shown in Figure 4(b).

3.3 Tag Identifier Cell

The tag identifier field is implemented with a single memory cell with two-port CAM and four port RAM capabilities. Figure 5(a) shows the schematic of the tag cell. During the read phase, the cell allows four port read access on the bit lines. The corresponding word lines are enabled by the destination match lines which pass through the cell. The two-port CAM is implemented in the same way as the four-four port CAM, and provides two match lines for signaling matches between the reorder buffer tag and the tag of instructions completing execution in the functional units. Evaluation of the tag match lines is enabled in the write phase, during which, the tag match lines control the word lines of the data field.

3.4 Ready Cell

The ready cell provides a control signal from the reorder buffer during the read phase for each of the four source registers whose value is being read from the reorder buffer. The signal indicates whether the tag or data value read from the reorder buffer should be used for the corresponding source register. Logically, the ready signal will be the select signal for a multiplexor which chooses between the tag and data value. For this function, the ready signal is designed as a four-port RAM cell. The destination match lines, which pass through

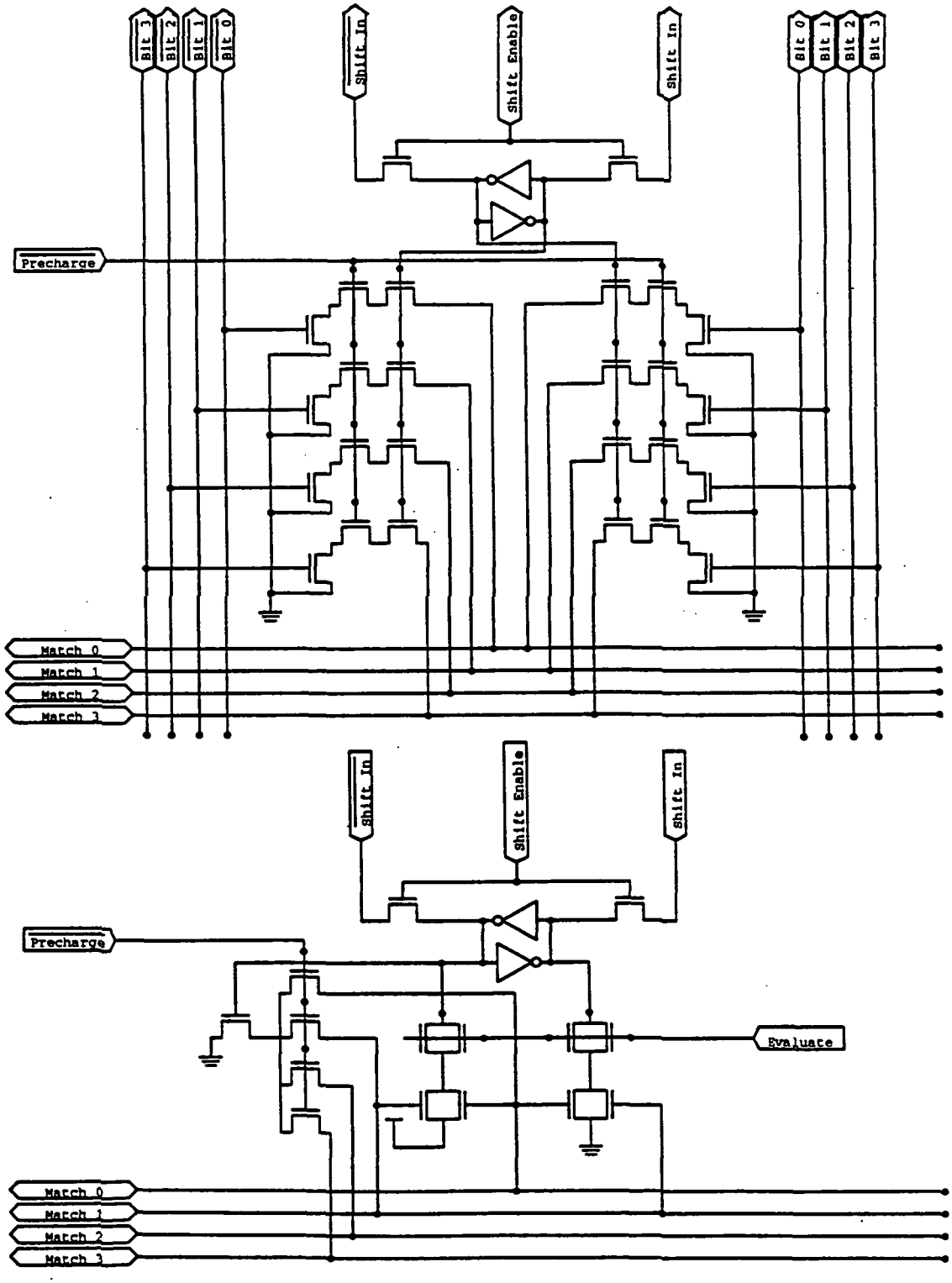

Figure 4: Schematic of a) Destination Iddentifier Cell (top) and b) Current Cell (bottom)

the cell, enable word access to the cell, allowing the value of the cell to be placed on the bit lines.

The tag match lines also pass through the cell as shown in the schematic of the cell in Figure 5(b), and they enable a set function during the write phase. When the tag for a reorder buffer entry matches during the write phase, the data from a functional unit is being written to the entry, and the ready bit is set.

3.5 Valid Cell

The function of the valid cell is to provide reset capabilities to the reorder buffer, and control the use of tags and data read from the reorder buffer for source operands. The cell is implemented as a four-port resetable RAM cell. The word lines of the cell are controlled by the destination match lines which pass through the cell. The bit lines of the valid cell are used to select between the value obtained from the reorder buffer, or from the register file for each source operand.

Reseting the reorder buffer can be achieved by clearing all of the valid cells in the buffer. The reset line of the valid cell asynchronously resets the cell, and indicates that the values in the reorder buffer should not be used. Typical uses of the reset capabilities are for reset of the microprocessor or to recover from mispredicted branches. The valid bit is set as each instruction enters the reorder buffer from the instruction decoder.

3.6 Data Cell

The data cell is a modified four-port RAM cell. The cell has two sets of bit lines for data access controlled by four word line enables. The word lines of the cell are controlled by the destination match lines during the read phase, allowing four values to be read from the data field. Then, during the write phase, the tag match lines are muxed onto the word lines. The tag match lines enable writes to the data field from functional units who have completed execution. Additional read and write control inputs to the cell, enable the path between the cross-coupled inverter pair and the pass transistors to the bit lines. These enables prevent the reading and writing of data during the evaluate phases of the cycle. Read is enabled by phase two, and write is enabled by phase four.

3.7 Shift Operation

The FIFO operation of this reorder buffer is achieved by shifting data between buffer entries. The shift is accomplished with a row of shift cells between each entry of the buffer, and with shift inputs and outputs on each of the cells of the buffer. During a shift phase, the data content of each cell in a column is shifted down one row or entry in the buffer, and the top entry is loaded with new instruction data. The shift cell performs two functions. First, it isolates entries of the buffer during shift with a transmission gate to prevent shift data from a single cell from propagating to other cells in the same column except for the target cell. Secondly, it uses the gate capacitance of inverters to store the data to be shifted, and then drive the shift inputs of the target cell. The transmission gate is enabled during phase 1

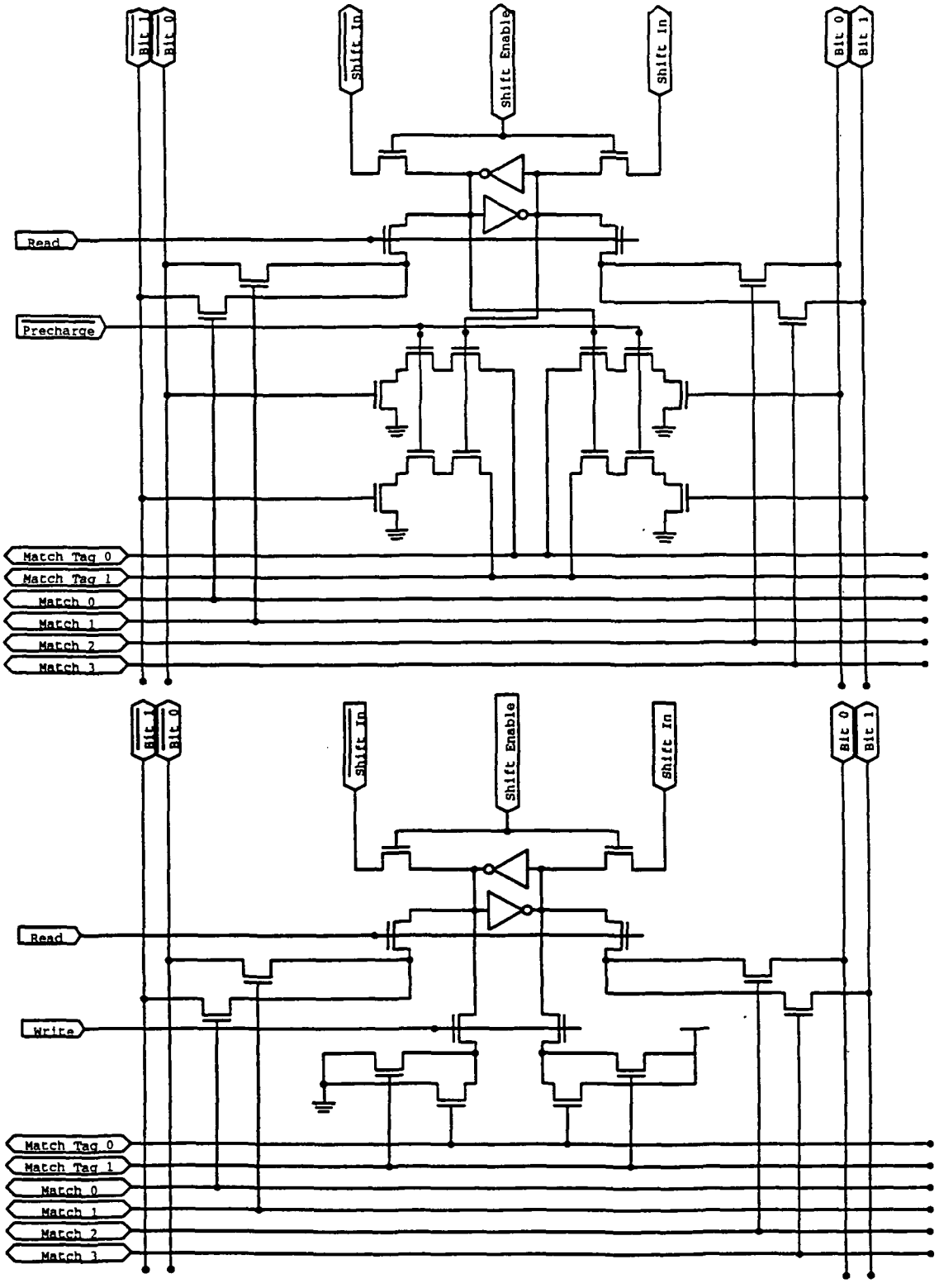

Figure 5: Schematic of a) Tag Cell (top) and b) Ready Cell (bottom)

to charge the shift cell with the data on the input. After phase 1, the transmission gate is disabled, and the data is shifted into the target cell during phase 3.

4 Simulation

A simulator program in C++ was developed to accurately represent a superscalar reorder buffer. It was used to verify the functionality of the extracted layout. The simulator is represented at the register transfer level and updates the contents of each cell at the end of each phase. Every type of cell in the design is defined as a class in C++. The simulator reads instruction information, generates an output file of the state of each cell at the end of phases 2 and 4, and generates a command (script) file. IRSIM uses that command file on the extracted layout to generate output which can then be compared against the simulated output. Optionally, the simulator has an user interactive feature whereby the complete state of the reorder buffer is represented in a user-friendly way.

The simulator reads in two (optionally one) instruction at a time from a .asm file and cycles through four phases until there are no more instructions left to execute. The .asm file input provides two source registers, a destination register, the latency involved in the operation, and the expected output for this virtual instruction. Note that no real operations are performed, the simulator behaves as if these values were passed to it from the instruction window.

As the simulator executes through each phase of a cycle, it generates appropriate commands that IRSIM should execute into a .cmd file. For instance, before phase 2 is executed, the source registers must be placed on the Destination Register Identifier bus for match comparison. This is done by the "set" vector command. At the end of phase 2, IRSIM is told to display the Valid, Ready, Tag, and Data values read from the reorder buffer, while at the end of phase 4, IRSIM is told to display the contents of the reorder buffer. Since the simulator knows the theoretical contents of the reorder buffer at that point, it outputs the expected value into a .out file, in a format identical to the output of IRSIM. After both the simulation and the layout simulation (IRSIM) have been executed, another program "cleans" the output of IRSIM, compares both outputs, and indicates any differences in a separate file.

4.1 Results

Simulation could be performed at 10 ns for each phase (25 MHz). During phase 1 and 3, precharging of the match lines took approximately 1 ns. The shifting during phase 3 took up to 3 ns to complete. Thus the self-timed circuit must be wary of this and not should not start before either condition is completed to avoid corruption of data or premature discharge of the match lines. During phase 1 evaluation of the match lines, if all the bits of the source operands compared the Destination Identifier Cell match except one, then the corresponding match line will have to discharge through a single transistor. This worst case took approximately 7 ns. Reading the reorder buffer during phase 2 lasted 3 ns for data bus,

6 ns for ready and valid bits, and up to 7.5 ns for the tag bus. The tag takes longer to read because it has additional internal load on it. During phase 3 evaluation of the match tag lines, if all the bits of the result tag compared to the Tag Cell match except one, then the corresponding ma

The limiting phase is phase 2. Since no external loads were used during simulation, the actual read time will take longer than stated. Since the tag bus has internal load, with additional external load, the worst case discharge time will take longer than 7.5 ns and probably longer than 10 ns. In addition, since no dead time was used, this will further increase the effective cycle time. In reality, taking in consideration for self-timed precautions, external load, and dead time, we can expect our 1.2 micron implementation would operate at 20 MHz.

5 Conclusion

The reorder buffer is an important part of an advanced computer architecture design that relies on run-time analysis of the concurrency at the instruction-level. This paper described the design and VLSI implementation of a reorder buffer. The results of this paper could be used to estimate and evaluate the design of superscalar microprocessors.

References

[1] Mike Johnson. *Superscalar Microprocessor Design.* Prentice Hall, Englewood Cliffs, 1991.

[2] J. Lenell. Superscalar and VLIW processors: A performance analysis. Master's thesis, University of California, Irvine, 1992.

[3] James Smith and Andrew Pleszkun. Implementation of precise interrupts in pipelined processors. In *Proceedings of the 12th Annual International Symposium on Computer Architecture*, pages 36–44, June 1985.

[4] Neil Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective.* Addison Wesley, 1985.