超純量微處理器中支援高指令發出率與完成率之多組庫重排序緩衝器

Multi-Bank Reorder Buffer to Support High Issuing and Retiring Rate for Superscalar Microprocessors

黃樹林

Shu-Lin Hwang

摘要

超級純量微處理器大多支援預測式平行執行以提升系統效能，這代表指令能被亂序執行。因此需要一個能重新排列指令為原始程式中順式的機制以保持正確的程式狀態。另一方面，當指令被亂序執行時也需能提供暫存器改名機制以解決資料相依問題(WAR 或 WAW)。一個重排序緩衝器通常在達成這兩個需求中扮演重要的角色。本論文主要提出一個於超級純量微處理器中能達到高指令發出率與完成率(最高為 8 個固定長度的 RISC 指令或稱為微運算碼)的重排序緩衝器的設計。並且為了簡化切換及讀/寫埠的複雜度，全部的資料位置使用交錯方式被分成八個獨立的組庫來運作。

關鍵字:超級純量，重排序緩衝器，多組庫

ABSTRACT

Superscalar microprocessors mostly support for speculative parallel execution to enhance system performance, this methodology means the instructions can be out-of-order executed. A mechanism is required for rearranging the instruction in original program sequence to keep a correct state. Another requirement, the register renaming mechanism is provided for resolving data dependence (WAR or WAW) when the instructions are executed out-of-order. A Reorder Buffer plays the role for achieving these two requirements. This paper proposes a design of high issuing and retiring rate (up to eight fixed-length RISC instructions referred to as micro-ops) Reorder Buffer of a superscalar microprocessor. In order to reduce the complexity of switching and read/write ports, the total entries are divided into eight independent banks by an interleaved way.

Keyword: Superscalar, Reorder Buffer, multi-bank

1. Introduction

It is difficult to determine a state precisely in a superscalar architecture since the processor or system state is not modified in a sequential order with respect to the other instructions. Thus when an exception occurs, it is necessary to perform a certain amount of work to obtain the correct processor state that should exist at the time the exception occurred. In order to achieve precise exception, Reorder Buffer (ROB) is one of hardware mechanisms can maintain the correct processor state when the instructions are speculative executed. Therefore, it can be used to restore the correct state as the exception has been resolved. The ROB is an instructions pool. It accepts the micro-ops from decoder's dispatcher in program sequential order. Because the dispatcher

黃樹林　明志科技大學電子工程系副教授

issues the micro-ops into the ROB in program order, so the micro-ops have been kept in original program order in ROB. After the micro-ops are placed in the ROB, the reservation station (RS) can copy multiple micro-ops from the ROB in any order and dispatch them to the appropriate execution units for execution. The criteria for selecting a micro-op for execution is that the appropriate execution unit and all necessary data items required by the micro-op are available – the micro-ops do not have to be executed in any particular order. As each micro-op in the ROB has been executed, it is marked as ready for retirement and its results are retained in the micro-op's respective entry in the ROB (rather than in the processor's actual register set).

In order to achieve precise exception, the ROB's micro-ops must be retired to actual register files in original program order. The retirement logic constantly checks the status of the oldest micro-ops in the ROB to determine whether they can be retired or not. If they can be retired, the micro-ops executed results are copied into the processor's real register set from the ROB entries and the respective ROB entry is then deleted (by updating the ROB's head pointer).

A diagram, as shown in Figure 1, Micro-ops are decoded and dispatched to the ROB from the Decoder/Dispatcher Unit in original program order. In the meanwhile, Source operand identifiers are sent to the ROB for searching the tag or value from the Dispatcher, then the searching information combining with the destination operand tag from the ROB are dispatched to the reservation station for preparing execution. The micro-ops do not have to be executed in program order. But they can be executed if and only if the function of which the micro-op executed is available and all necessary data items required by the execution are available.

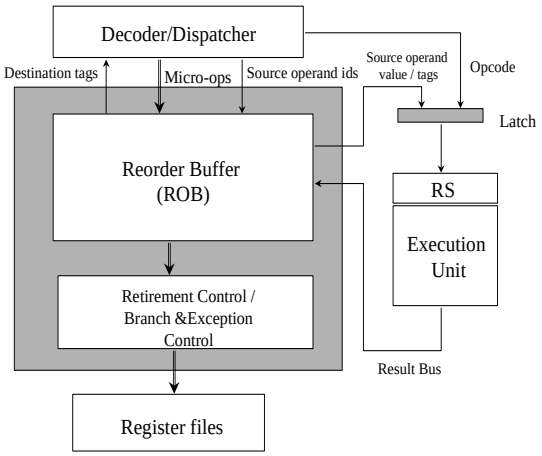

Figure 1. The Reorder Buffer Block Diagram

After execution, the results are written to the corresponding entries in the ROB through the result bus according to the destination tags. The Retirement Control/Branch&Exeception Control unit checks the oldest micro-ops in the ROB to determine whether they can be retired or not. If does, the unit writes the results to the register files. If doesn't, exception or branch misprediction might have occurred. This unit must send flush signal to the ROB and other units for flushing incorrect micro-ops.

In this paper, we will briefly introduce the traditional Reorder Buffer architecture, and then describe our multi-bank scheme for the superscalar.

2. Reorder Buffer Method

This method allows instructions to complete out of order, but retains the results of each instruction in a reorder buffer. The processor uses the reorder buffer to reorder out–of–order completion instructions before they modify the processor state. The processor updates the register file from this reorder buffer only after it knows that all previously issued instructions are free of exception conditions. This approach has an advantage over the in-order completion method because it allows multiple instructions to be executed concurrently. Each entry of the result shift register needs an extra field, called the Tag field, for controlling the scheduling of instructions. An

organization of the reorder buffer method is shown in Figure 2.

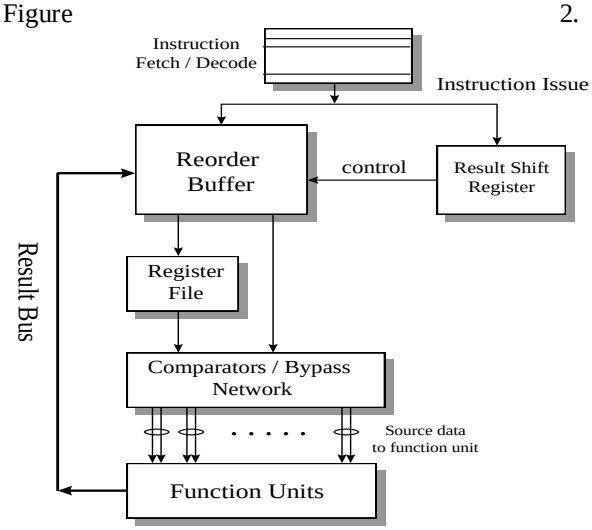

Figure 2. Reorder Buffer Method

Figure 3 presents the operation of the result shift register for a sample program, using the reorder buffer method. The reorder buffer takes the form of a circular queue when the processor issues an instruction, it places the current program counter and a tag to the destination register in the reorder buffer entry that the tail pointer points to. Simultaneously, the tag value is placed into the Tag field of the corresponding entry in the result shift register, along with the execution unit field indicating which functional unit the instruction will use. After the processor then increments the tail pointer, modulo the reorder buffer depth, it is ready for the next instruction. In the reorder buffer, the tail pointer points to the most recently issued instruction and the head pointer points to the earliest issued instruction, as shown in Figure 3.

Direction of movement

Stage	Functional Unit Source	Valid	Tag
1		0	
2	Integer ADD	1	4
3		0	
4		0	
5	Float-Point ADD	1	5
:			
N		0	

Tag	Dest. Reg.	Result	Exceptions	Valid	Program Counter
3					
head → 4	R4			0	6
5	R0			0	7
tail → 6					
⋮					

Figure 3. The relations between the result shift register and the reorder buffer

As an instruction finishes execution (completed, and leaves the result shift register), the reorder buffer pointer from the result shift register guides the instruction results and exception condition to the correct reorder buffer entry. When this reorder buffer entry becomes the head of the queue and contains valid results, the destination register pointer guides the results, which are now stored in the reorder buffer, into the destination register. If the processor detects no exceptions, it writes the results into the destination register. If it detects an exception, it will stop issue instruction and inhibits all further writings into the registers. After the processor updates the register file, it no longer needs the reorder buffer entry, and it increments the pointer to the head of the reorder buffer, discarding the entry. [1][5][9]

3. Multi-Bank Reorder Buffer Design

Our architecture of the reorder buffer is introduced in this section which includes the specific and function. We explain how it achieves high issuing and retiring rate performance.

The Intel Pentium Pro microprocessor has a reorder buffer design with an average three micro-ops of issuing and retiring rate. In order to

achieve higher performance, we propose a new reorder buffer architecture with higher issued and retired micro-ops per cycle comparing to Pentium Pro microprocessor. However, because reorder buffer is the control center of a superscalar processor which implies the circuit design complexity is very high.

In order to achieve compatibility to x86 instructions, the microprocessor's each general purpose register is divided into three parts which are eXtension (X) part, High byte (H) part, and Low bye (L) part. So, the data dependence checking is not only checks full register but also checks partial register that instructions retrieve. Therefore, the complexity is increased when dealing with data dependence. Because there is only one flag, that is not a problem to find latest micro-op's flag when we do source operands searching or write the latest updated flags into Eflag register when a micro-op is retiring.

The dispatcher has two choices to log micro-ops into ROB, one is logging into ROB's entry that top pointer points to, the other is logging into the location by tag matching. The numbers of micro-ops can be logged-in or retired also have two choices, fixed number or various numbers. The latter is more flexible but also more complex to design.

The misprediction handling can be done in various time: the time as soon as the misprediction occurs or the time when the misprediction micro-op is going to be retired. Exception handling needs to consider the information sent exception handler, including EIP (32 bit x86 instruction pointer) for recover instruction execution when the exception has been processed.

Our reorder buffer has the following main features:

- Keep the lookahead state (register file maintains the in-order state)
- Divide ROB into eight independent banks in interleaved round-robin way, they can be considered as a big ROB conceptually
- ROB can accept up to eight log-in micro-ops and retire up to eight micro-ops in one cycle
- Retrieve the source operands and flags modified by previous instructions for up to eight micro-ops simultaneously
- Commit the result values and flags for four micro-ops simultaneously
- Rename destination register and flags
- Retire (update register files) all the micro-ops following the program sequence correctly
- Handle flushing caused by the misprediction and the exception, and call the Exception Handler to recover the exception.

As mentioned before, the ROB is a First-In First-Out (FIFO) circular queue. In order to maintain program state correctly, the ROB controls completed micro-ops to update register files in original program order. As exception occurs, processor could recover to correct state after the exception processed. It is also used to reserve micro-ops' information that have been completed but not retired temperately. The ROB's function controls when the result from the functional units will update register files.

The decoder decodes micro-ops in program order, and then dispatches micro-ops to the ROB banks individually and sequentially from dispatcher&scheduler. How to keep micro-ops' original order among the ROB banks is a key to decide if the micro-ops could be retired in

program order later.

The Reorder Buffer (ROB) consists of the following five major components: eight discrete Banks, Issue Pointer & Tag Dispatch Circuit, Source Operands Searching Circuit, Result Write Arbiter Circuit, and Retirement Circuit/Branch & Exception Control Circuit as shown in shadow parts of Fig 4.

The main functions of the ROB include three aspects. The first one is to rename the destination register by the physical ROB addresses where the corresponding micro-ops are stored. Renaming can move the following two false dependencies: write after read (WAR), and write after write (WAW), and can speed up the process of the true dependence – read after write (RAW).

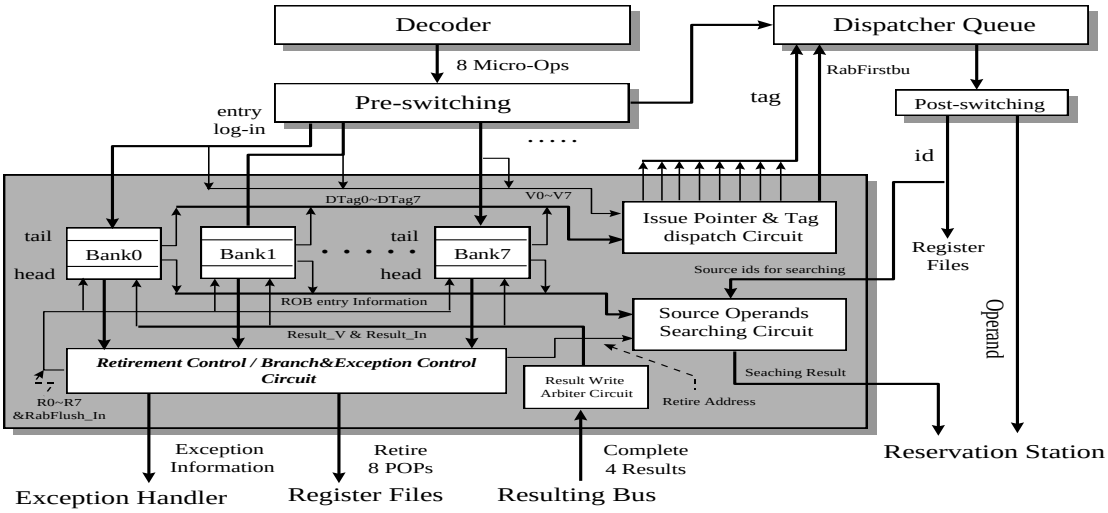

Figure 4. Multi-Bank Reorder Buffer Block Diagram

The second function is to correctly retire all the micro-ops following the program sequence. To retire up to eight micro-ops in one cycle, the Retirement Control needs to check the following three special cases: destinations conflict or overlapping, the store micro-op and misprediction and exception. Those three special cases will make the ROB unable to retire the eight micro-ops in one cycle even if the eight micro-ops can be retired at least. In following sections we will describe how to take care of the retirement.

The third function is to handle the flushing caused by the misprediction and the exceptions, and to start the Exception Handler to recover the exceptions. In the meanwhile, the ROB must send the EIP to the Decoder to restart decoding the micro-ops.

Basically, the ROB is the only mechanism in processor which can know the order of all the micro-ops, so it has to be responsible for controlling the out–of–order execution flow to guarantee that the result can be retired correctly in whatever condition. In following sections, we introduce the detail of individual blocks that compose of the ROB.

4. *ROB Banks*

This block is constructed of a FIFO circular queue structure which stores the micro-ops temporary information from the Decoder/Dispatcher or the Result Bus (after executing the micro-ops, then the operated result is output to the result bus). Our ROB has 64 entries totally. A tail pointer is used to point the entry that will be allocated from the dispatcher in

the next cycle (negative edge). Additionally, the ROB also has a head pointer that points to the entry to be retired (recognizes whether it could be retired or not by retirement logic) in the current cycle (negative edge). Owing to these two pointers, the processor can point out the valid entries region and judge the full or empty condition for the ROB.

According to our design, the ROB Bank block is comprised of eight independent banks. All the eight banks are of the same structure. They can be considered as a big ROB conceptually. Physically they are divided into eight independent pieces in order to reduce the complexity of switching and read/write ports. Each bank has eight entries, a header and a tailer to point out the valid entries region. To match the logical concept of a big ROB, the micro-ops in the dispatcher queue are interleaved and logged into the banks in round–robin way.

In each cycle, the Dispatcher can issue up to eight micro-ops into the individual ROB bank entry pointed by its own tail pointer. Because the issuing quantum of micro-ops from dispatcher is various in each cycle. In this case, the dispatcher needs to know the starting ROB bank number for micro-ops logged-in in the next cycle. This assures that the processor can retire micro-ops in original program order. However, the quantum of the micro-ops could be retired is also various. The eight ROB banks' entries pointed by tail pointers individually are the micro-ops to be retired by retirement logic circuit. So the processor needs to maintain two pointers that record the information about the start point for issuing (Ip pointer) and retiring (Rp pointer) in the next cycle. After issuing and retiring micro-op, the tail and head pointers need to be incremented individually.

The main features of this block are shown as the following:

- *Log-in micro-ops into ROB bank every cycle*
- *Write result to ROB bank according to tag address*
- *Output operand information to Source Operands Searching Circuit block to find source operand tag or value*
- *Output retiring micro-ops' (the entry pointed by head pointer) information to the Retirement Control/Branch&Exception Control Circuit block which decides whether it can be retired or not*
- *Flush ROB entries when RabFlush_ signal arrives and issues ROB bank full and empty signals*

In each ROB bank (Figure 5.), the DpRabreq bus is the input data line recording the micro-ops' information from the dispatcher for log in. The RabSrc buses send out the information about source operand for source operand searching circuit to resolve data dependency. One RabSrc bus exhibits one entry information. The Result_In bus accepts the completed data from the result bus for changing the corresponding entry (tag match) execution status (ex. misprediction, exception).

The Result_V signal exhibits if the result from the result bus is valid. The RabRetire bus sends out the information of the entry pointed by head pointer, the retirement logic circuit will use this information for judging whether it could be retired or not. The R is a signal from the retirement logic circuit to decide whether the entry pointed by head pointer can be retired. The RabEmpty_ and RabFull_ indicate the ROB bank's condition is empty or full. The RabFlushIn_ signal decides whether we need to flush the ROB bank. The E_Buf signal stores the exception information caused by delay exception (EN=16, Floating Point Error). Finally, the DTag

is an assigned tag data for the current issued micro-ops entry in this cycle. This information is the destination operand's tag that the dispatcher needs.

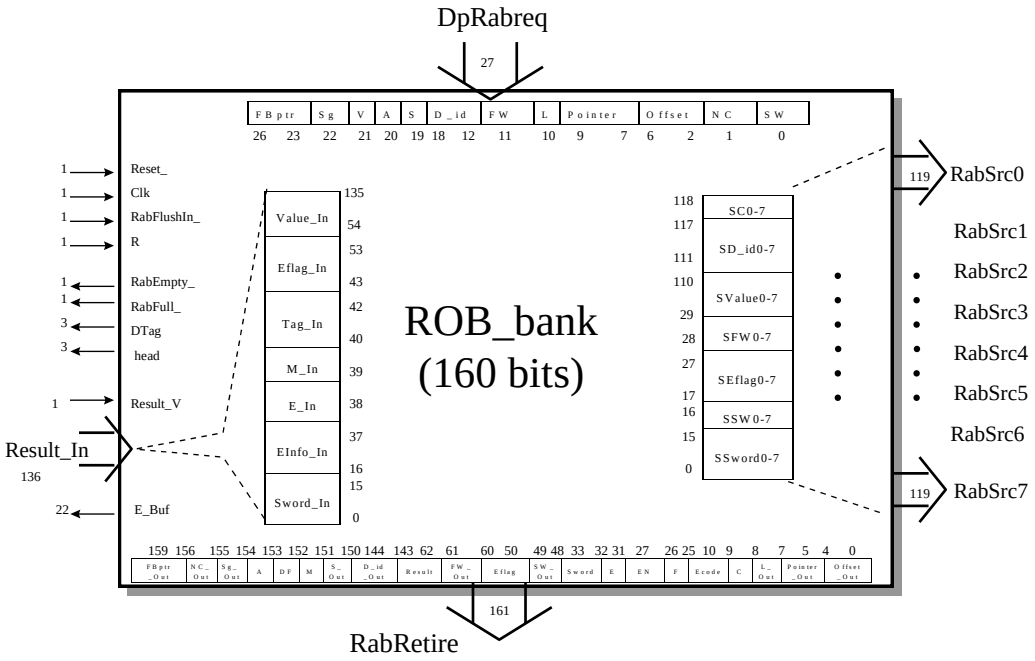

Figure 5. The individual ROB bank I/O interface

5. ***Numerical Results:***

We design a verilog test pattern file for the simulation to verify the correctness of the ROB function. Decoder starts logging multiple micro-ops into the ROB before the falling edge of clock 3. The RabFirstbu and RabTag signals are updated before the rising edge of the clock after the micro-ops logged-in. Before the rising edge of the clock 5, the first set result comes to the ROB for updating. The contents of the ROB entry stores (address from 0~8) is shown in the figure 6. About retirement aspect, the R0~R7 signals show that there are four micro-ops retired in clock 6, and two micro-ops retired in clock 7. In the meanwhile, an exception occurs in the micro-op of the bank 6 is detected in clock 7. So it activates the Rabflush_ signal simultaneously. After RabFlush_ active, the RabFirstbu and Rp reset that indicates the start point for logging-in and retiring micro-ops is bank 0.

6. Conclusions

The modern superscalar microprocessor supports for speculative execution to achieve high performance. Out-of-order execution needs a mechanism to maintain the instruction's original program order. With this mechanism, the processor can recover to the correct state after exception occurs. Another objective is to provide result data temporary storage to achieve register renaming [7]. This mechanism can be supported by the following approaches, Ex. History Buffer, Reorder Buffer and Future File. Among these, Reorder Buffer is the most popular method. That's the reason why we choose the reorder buffer method to develop our design. The popular microprocessor likes: Intel Pentium Pro and AMD K5 also use the reorder buffer mechanism.

According to the databook of the Intel Pentium Pro, it could log-in and retire three micro-ops per cycle. In AMD K5 the micro-ops logging-in and retiring rate is four. However, our reorder buffer design can log-in and retire up to eight micro-ops per cycle. In order to reduce the complexity of switching and read/write ports, a 64 entries reorder buffer is

divided into eight independent banks by an interleaved way. Increasing on hardware cost to achieve higher performance is a trade off to design the reorder buffer. This new idea for Dividing banks, Source Operand Searching and Retirement data dependence to resolve are our major contribution.

References

[1] J.L. Hennessy, D.A. Patterson, "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publisher, San Francisco, Calif., 1996

[2] James E. Smith, and Andrew R. Pleszkun, "Implementation of Precise Interrupts in Pipelined Processors," *Proc. 12th Ann. Int'I Symp. Computer Architecture*, Los Alamitos, Calif., 1985, pp. 36-44.

[3] W.M. Hwu and Y.N. Patt, "Checkpoint Repair for Out-of-Order Execution Machines," *Proc. 14th Ann. Int'I Symp. Computer Architecture*, 1987, pp. 18-26.

[4] Gurindar S. Sohi and Sriram Vajapeyam, "Instruction Issue Logic for High-Performance, Interruptable Pipelined Processors," Computer Sciences Department, University of Wisconsin-Madison 1987

[5] W.M. Johnson, "Superscalar Microprocessor Design," Prentice-Hall, Englewood Cliffs, NJ, 1991.

[6] H. Dwyer and H.C. Torng, "Out-of-Order Superscalar Processor with Speculative Execution and Fast, Precise Interrupts," *In proceedings of the 25th Annual International Symposium on Microarchitecture*, 1992., pp. 272-281

[7] M. Moudgill, K. Pingali, and S. Vassiliadis, "Register Renaming and Dynamic Speculation: an Alternative Approach," Department of Computer Science, Cornell University. August 1993.

[8] C.-J. Wang and F. Emmett, "Implementing Precise Interruptions in Pipelined RISC Processors," IEEE Micro, Vol. 13, No. 4, Aug. 1993, pp. 36-43.

[9] Mayan Moudgill and Stamatis Vassiliadis, "Precise Interrupts," IBM T.J. Watson Research Center and Delft University of Technology. IEEE Micro, Feb. 1996.

[10] Tom Shanley, "Pentium Pro Processor System Architecture," MindShare, Inc. 1996.

[11] Mayan Moudgill and Stamatis Vassiliadis, "Precise Interrupts," IBM T.J. Watson Research Center and Delft University of Technology. IEEE Micro, Feb. 1996.

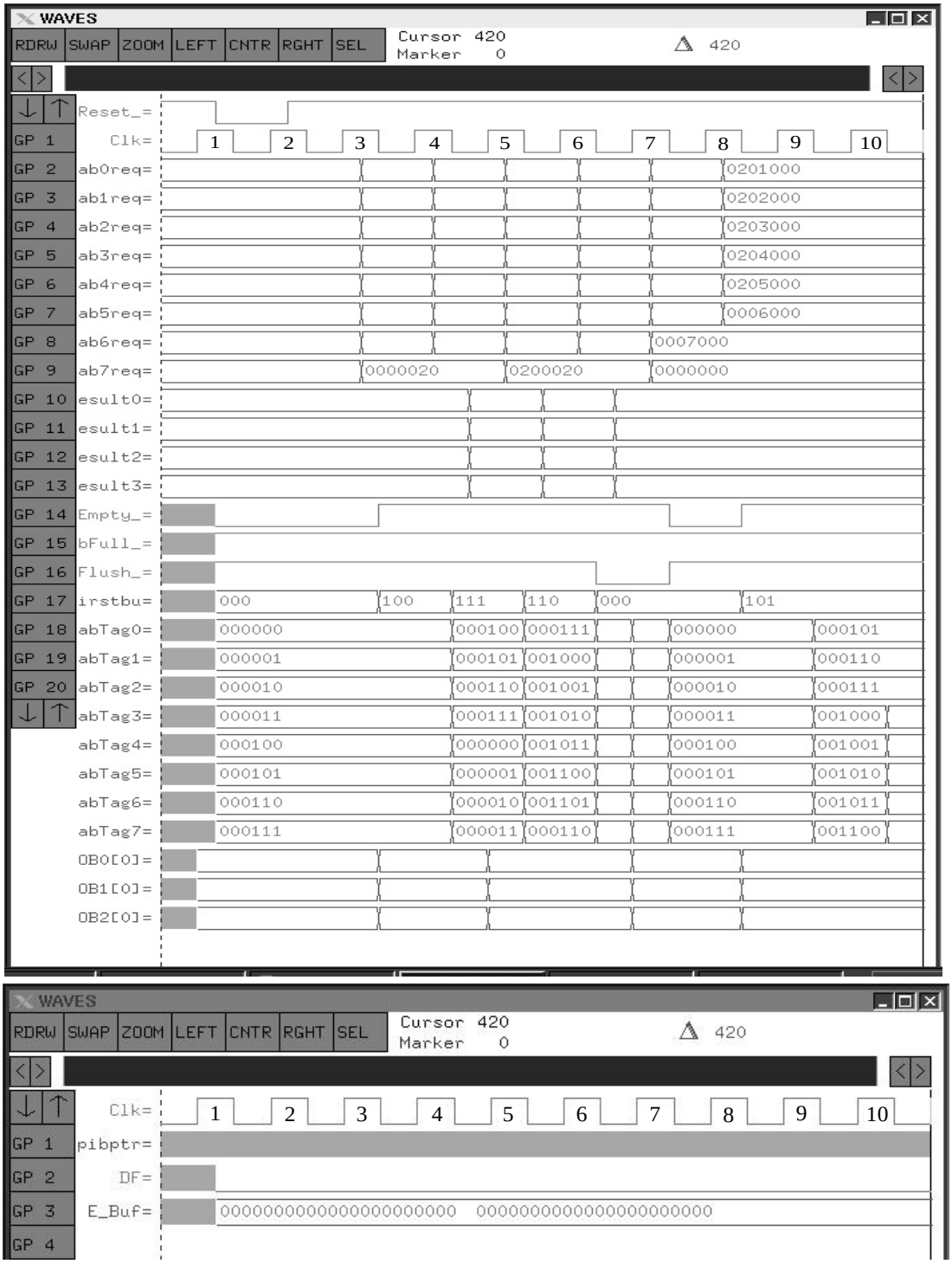

Figure 6. The ROB Verilog Simulation Result