Efficient Implementation of Precise Exception for Processor Based on Pre-detection

Qingyu Chen
Xi'an Microelectronics Technology Institute
Xi'an, China
e-mail: chenqingyu2006@163.com

Longsheng Wu
Xi'an Microelectronics Technology Institute
Xi'an, China
e-mail: chenqingyu2006@163.com

Erjuan Zhu
Shaanxi Changling Photovoltaic Electric Co. Ltd
Bao ji, China
e-mail: erjuan1022@126.com

Abstract—Embedded systems have higher requirements on real-time performance of processor. However exception, which can interrupt normal execution of program, will decrease the processor performance. For improving the exception handling efficiency of processor, a precise exception handling method for embedded pipeline processor based on pre-detection is proposed in this paper. When the precise exception occurs in any pipeline stage, the precise exception flag is set valid immediately and advanced to the next pipeline stage. Based on pre-detection on the exception flag, a single-cycle NOP instruction is provided for the processor by a dedicated hardware module. That separates the processor from spending large number of clock cycles requesting main instruction memory for instructions which will be flushed in the process of exception handling. This method has been implemented in a SPARC V8 processor which has successfully taped out. Test results of chip show that the precise exception detection and response efficiency is increased by 35.47% without increasing the processor critical path. The proposed method can be used to improve response efficiency of the precise exception at low hardware overheads.

Keywords-exception flag; pre-detection; dedicated module; NOP instruction; processor

I. INTRODUCTION

Most modern embedded processors develop the instruction-level parallelism and improve the real-time performance of processor by the means of deep pipeline and multiple issue[1]. These techniques make the control of the processor, especially the accurate control of exception more difficult. The implementation of the precise exception is the important to the processor to satisfy the sequential architecture [2].

In order to implement the precise exception, the controller in the processor will flush all instructions in pipeline to ensure that the results of instruction execution will be committed in-order when it responses to precise exception. References [3-5] introduce some techniques, which are commonly used in out-of-order processor to implement the precise exception, and the authors had studied the implementation method of the hardware and the overhead, but did not consider the response efficiency of exception. And the implementation principles and schemes of precise exception in single issue processor are concluded in [6-9]. Those schemes can accelerate the speed of recovery processor context by increasing the backup buffer or future buffer, but they didn't take the time of fetching instruction into account as those instructions will be flushed finally in the response of exception.

Aiming at the defects of the existing method for handling precise exception, this paper presents an optimization method to implement precise exception for single issue processor, and conducts an experimental verification with a SPARC V8 processor developed by Xi'an micro-electronics technology institute. The method focuses on reduce the time of fetching invalid instruction which is in the pipeline and have not arrived at write-back stage when processor is handling the precise exception. Firstly, the paper introduces the definition of precise exception and its implementation in SPARC V8 architecture, and then based on the analysis of the existing methods, the precise exception implementation method based on pre-detection are illustrated. Finally, we analyze and conclude the testing process and results of this method.

II. PRECISE EXCEPTION

The exception is like an unexpected procedure call and may be caused by an instruction-induced exception or an external interrupt request not directly related to a particular instruction. There are 256 distinct types of exception by 8-bit TT (trap type) field identification in SPARC V8 architecture — half for hardware exceptions and half for software exceptions and the transfer addresses of all traps are decided by the exceptional type. All exceptions can be divided into three categories of exception: precise exception, deferred exception or interrupting exception.

A precise exception is induced by a particular instruction and occurs before any program-visible state has been changed by the trap-inducing instruction. Several conditions must hold: (1) The instructions before the one which induced the exception have completed execution. (2)The instructions after the one which induced the exception remain unexecuted. (3)The PC saved in local register1 points to the instruction which induced the exception, and the NPC saved in local register2 points at the instruction which was to be executed next [10].

III. IMPLEMENTATION OF PRECISE EXCEPTION

The precise exception handling method based on pre-detection has been proved to be effective in a SPARC V8 processor which has been successfully taped out. The processor achieves 159 kinds of exceptions, of which 144 types of precise exceptions and 15 types of external interrupt which will induce deferred exceptions. This section mainly introduces the process of exception handle, the traditional method for implementing precise exception, and the method based on pre-detection which is proposed in this paper.

A. Traditional Implementation

The implementation of precise exception typically is divided into four processes. The first is the exception detection and response, namely detecting the exception in instructions, and beginning to handle. Secondly the context preservation, mainly refers to save instruction PC, processor state PSR, generate subroutine entry address of exception handler. And then the controller begins to execute exception handler programs. Finally recovering state/control fields, such as PSR, from exception handler subroutine, at the same time resuming the main program. The main process of exception handling and time consumption is shown in Fig .1. Different from the public methods which aim at decreasing t2 or t4, this paper focuses on optimizing t1 since the t1 time overhead is more serious than t2 and t4 in the deep pipeline processor.

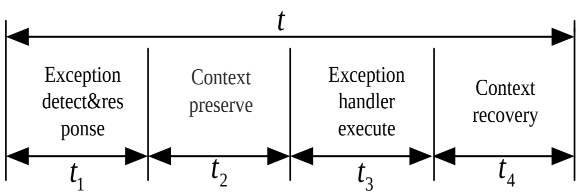

Figure 1. Process of exception handing & time consumption

The V8 processor mentioned in this paper has a typical 5-stage pipeline, which are IF (instruction fetch), ID(instruction decode), EX(execute), MEM(memory access) and WB(write back). The traditional method to implement precise exception is shown in Fig .2, including two steps: (1) distributional detection. In order to balance the payload of each pipeline stage, the detection and priority arrangement of precise exception is distributed to the all pipeline stage. Ten kinds of exception related to instruction execution, such as illegal instruction, windows overflow, privileged instruction and floating exception, can be detected in ID stage and the highest priority exception is advanced to the EX stage after priority arrangement. Some precise exceptions which has relationship with memory accessing will be detected in MEM stage. (2) centralized control. All exceptions change the state of the processor in WB stage, which can simplify and reduce the degree of difficulty in achieving precise exception to put exception controller logic on WB stage. When finding the instruction exception in any pipeline stage except for WB stage, the pipeline controller executes it continually as a normal instruction instead of handling immediately and sets the exceptional flag and reason (TT) into pipeline registers to transfer it to the WB at the same time. As soon as the exception controller in WB detects the exceptional flags, it will generate exceptional service address according to exceptional reason (TT) and preserve the context of program execution. In addition, the exception controller will flush all instructions which are in pipeline at the same time. These flushed instruction execution will not change processor state, so these instructions are executed uselessly. We call these flushed instructions as invalid instruction. As is shown in Fig.2, there are four pipeline stages before WB stage. So, in the worst case, four invalid instructions would be in the pipeline. The operation on flushing these instructions will lead the time consumption in fetching invalid instructions useless. When instruction SUB induces an exception, which is shown in Fig .3, its subsequent instruction ‘AND’, ‘XOR’, ‘SMUL’ and ‘SDIV’ has been fetched and executed normally before the SUB reaching the WB. Exception controller will flush all instructions in the pipeline, ‘AND’, ‘XOR’, ‘SMUL’ and ‘SDIV’ in Fig.3, when the SUB arrives at the WB. Although the mechanism of central controller can simplify and reduce the implementation difficulty of precise exceptions, it cannot reduce the time consumption in instruction fetching and execution of invalid instruction.

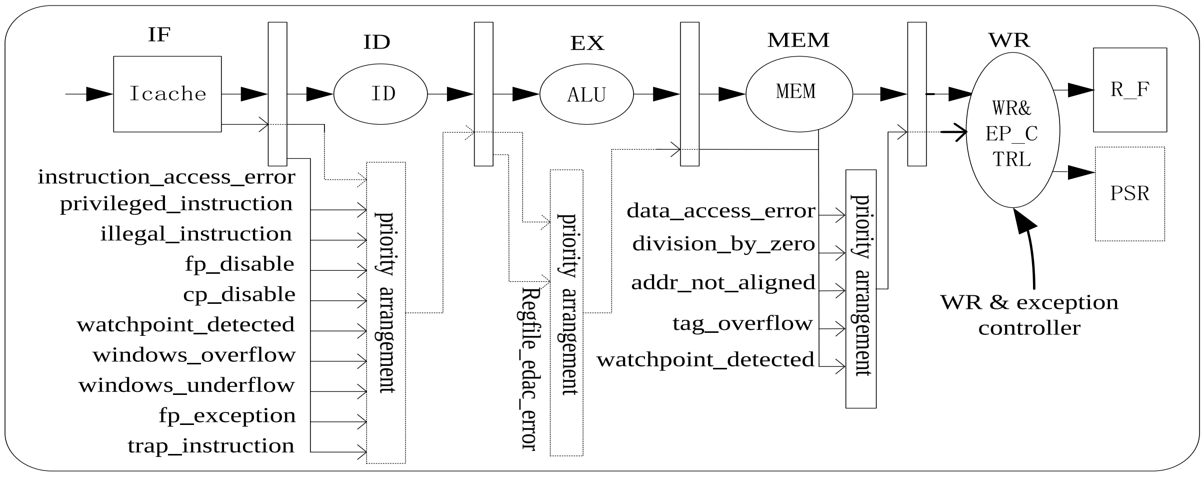

Figure 2. Traditional method to achieve precise exception

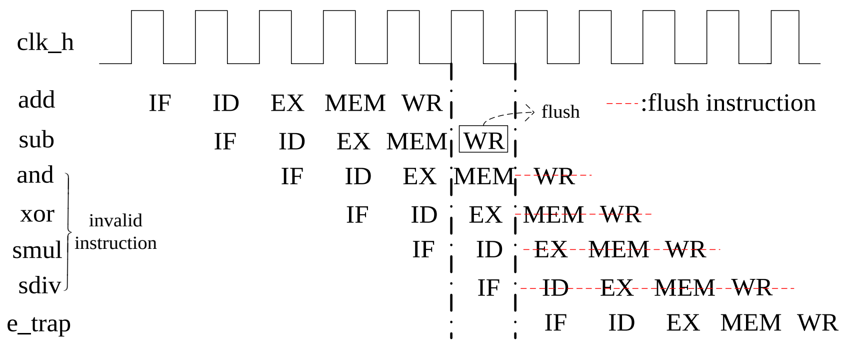

Figure 3. Instructions flush in the precise exception

B. *Implementation based on Pre-detection*

In order to reduce the time consumption of invalid instruction fetching, the method which is based on the mechanism of exception distributional detection, by pre-detecting the precise exception flags, immediately provides a single-cycle NOP instruction for the ID stage through a dedicated hardware module and stops IF stage to request instructions to slower main memory as soon as the EFD (exception flag detection) module finds the exceptional flags valid in any pipeline stage. The method can effectively reduce the time consumption of fetch instruction operation of the invalid instructions.

The concrete implementation of this method is shown in Fig .4. When detecting the instruction exception in any pipeline stage, the exception flag is set valid in that pipeline stage. Two modules, namely EFD (exception flag detect) and ROR (read only register), are added in the IF stage of the processor. Considering the exception flags from the EX, MEM and WR as the input of the OR gate, the output of the OR connects to EFD module. If the output of the OR is effective, EFD would ban processor to request instructions to the main memory through inull signal connected to the Icache (instruction cache) and select a NOP instruction saved in the ROR module for ID stage by a multi-channel selector. The exceptional flags will be valid until the instruction inducing exception is completely out of the pipeline where the EFD inull signal becoming invalid, IF resumes to request instructions to main memory and the processor is transferred to exception handler program to execute. Because invalid instructions replaced by NOP instruction which does not cause the change of processor state, The method can simplify the refresh logic for the whole pipeline with only refresh logic in EX and MEM stages retained. On the others hand, such a mechanism avoids the processor from spending plenty of clock cycles requesting instructions to slower main memory and reduces the time consumption of invalid instruction fetching.

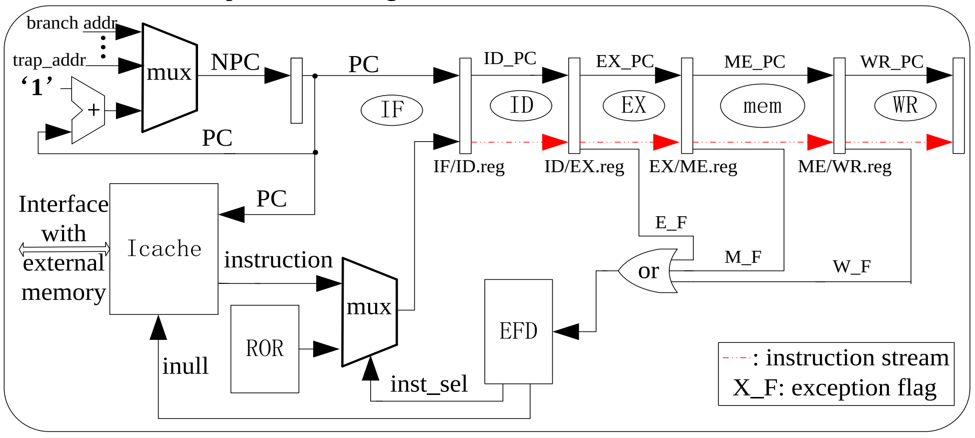

Figure 4. Implementation based on pre-detection

IV. Verification and Results

The exception response time of the processor is one of the important standards for the real-time performance. This scheme verifies the same two silicon circuits except for the method to implement precise exception on ASIC development board by the TIMER integrated into the processor[11]. The process and environment of testing are shown in Fig .5 and 6. In the process of testing, through the DSU we upload test case to SRAM with 5-waiting clock cycles in the GRMON environment and started running the processor[12]. When executing instruction ta1 (soft trap), the processor enters debug mode. Then checking the TIMER count value, we can calculate the exception detection and response time t1 in specific test case according to the clock frequency. Test results are listed in the table 1 and 2 between different cases, of which case1 is suitable for illegal instruction exception, case2 for register file check error, case3 for unaligned address exception. case1, case2 and case3 are respectively detected in ID, EX and MEM stage. The verification results are catalogued in Table 1 and 2.

Testing results show that: (1) when cache hit, this method has no effect on the precise exception response efficiency. (2)When cache un-hit, the approach can significantly reduce the exception detection and response time t1. The earlier the exception flag is detected, the more obvious this method optimizes the performance. The design implements 144 types of precise exception, of which 138 kinds are detected in ID stage, one kind is detected in EX stage, 5 types of precise exception are detected in MEM stage. With comprehensive consideration of various exception cases, the exception detection and response time t1 reduced by 35.47%.

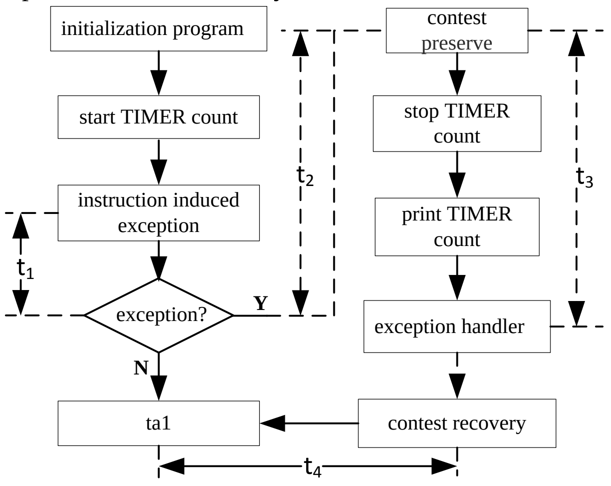

Figure 5. Exception detect and response time testing process

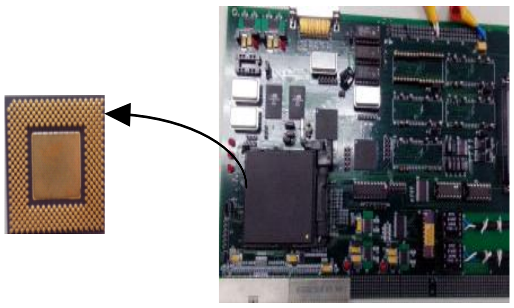

Figure 6. Silicon circuit verification environment

TABLE I. TESTING RESULT WITH CACHE-UNHIT

Testing case	case1	case2	case3
pre-optimization /ns	330	330	330
post-optimization /ns	210	250	290
improvement	36.4%	24.2%	12.1%

TABLE II. TESTING RESULT WITH CACHE-HIT

Testing case	case1	case2	case3
pre-optimization /ns	210	250	320
post-optimization /ns	210	250	290
improvement	0%	0%	0%

V. SUMMARY

Based on the analysis on definition of SPARC V8 precision exception, this paper proposes a method to implement precise exception for single issue pipeline processor aiming at the deficiency of existing precise exception implementation. When an exception occurs, the method, by pre-detecting the precise exception flags, immediately provides a single-cycle NOP instruction for the processor through a dedicated hardware module. Such a mechanism can effectively reduce the time consumption of fetch instruction operation of the invalid instructions and simplify the flush operation of the whole instruction pipeline. This method has been implemented in a SPARC V8 processors developed by Xi'an microelectronics technology institute. The exception detection and response time of the silicon circuit is verified through 6 cases. The results show that this method makes the precise exception detection and response time t1 reduced by 35.47%. The proposed method can be used to improve response efficiency of the precise exception at low hardware overheads.

ACKNOWLEDGMENT

The research work is supported by National High-Tech Research and Development Program of China(No. 2010ZX01021, No. 2009ZX01023). And I appreciate that the help from Xi'an Microelectronics Technology Institute.

REFERENCES

[1] D. A. Patterson and J. L. Hennessy, Computer organization and design: the hardware/software interface, 2nd ed., Newnes, 2013, pp.483-495.

[2] J. L. Hennessy and D. A. Patterson, Computer architecture: a quantitative approach, 3rd ed., Elsevier, 2012, pp.68-75.

[3] J. E. Smith and A.R. Pleszkun, "Implementing precise interrupts in pipelined processors," Computers, IEEE Transactions, vol.37, no.5, 1988, pp.562-573.

[4] M. Pericas, A. Cristal, R. González, D. A. Jiménez and M. Valero, "A decoupled kilo-instruction processor," Proc. The Twelfth International Symposium on High-Performance Computer Architecture, IEEE Press, 2006, pp.53-64.

[5] Z. Y. Liu and J. Y. Qi, "Implementation of precise exception in a 5-stage pipeline embedded processor," Proc. The 5th International Conference on ASIC, IEEE press, 2003, pp.447-451.

[6] W. J. Li, "A 5-stage pipelined embedded processor with optimized handling exception," Proc. 2011 International Conference on Computer Science and Network Technology (ICCSNT), IEEE press, 2011, pp.2773-2777.

[7] X.Chen, S.B. Zhang and X.B. Shen, "Optimized Realization of Precise Interrupt on Microprocessor Longtium R2," Application Research of Computers, vol. 7, 2007, pp. 60-65.

[8] Z. Y. Liu and J. Y. Qi, "High Performance RISC Microprocessor Hardware Simulator Design," Journal of Computer Research and Development, vol. 41, 2004, pp. 1436-1440.

[9] B. Edouard, "Binary translator with precise exception synchronization mechanism," U.S. Patent No. 8,296,551. 23 Oct. 2012.

[10] Internationa S, The SPARC architecture manual Version 8, SPARC International Inc, 1998.

[11] X. F. Yin, S. H. Yuan and J. B. Hu, "Experimental Research on Interrupt Latency Time of ARM Microprocessor," Computer Engineering, vol. 37, 2011, pp.252-563.

[12] Q. Y. Chen, T. Y. Sheng and Q. Y. Duan , "The design of reusable interface for AHB bus slave," Microelectronics & Computer, vol. 23, no.7, 2008, pp.129-133.