Available online at www.sciencedirect.com
SCIENCE DIRECT®
Microprocessors and Microsystems xx (0000) 1–10
MICROPROCESSORS AND MICROSYSTEMS
www.elsevier.com/locate/micpro

An effective out-of-order execution control scheme for an embedded floating point coprocessor

Cheol-Ho Jeong[a,*], Woo-Chan Park[a], Tack-Don Han[a], Sung-Bong Yang[a], Moon-Key Lee[b]

[a]*Department of Computer Science, Yonsei University, 134 Shinchon-Dong, Seodaemoon-Gu, Seoul 120-749, South Korea*
[b]*Department of Electrical Engineering, Yonsei University, 134 Shinchon-Dong, Seodaemoon-Gu, Seoul 120-749, South Korea*

Received 18 June 2002; revised 18 December 2002; accepted 12 January 2003

Abstract

This paper proposes an out-of-order execution control scheme that can be effectively applied to a coprocessor for embedded systems. A floating-point coprocessor has generally multiple pipelines such as a floating-point adder, a floating-point multiplier, a floating-point divider and a load/store pipelines. In order to utilize fully these pipelines, a constraint-based dynamic control scheme is designed for a coprocessor. This control scheme can be achieved by a data dependency checking, a resource conflict checking, and an exception prediction technique. With this control scheme a coprocessor can execute its instructions out of order without an extra hardware unit for out-of-order execution control.

Keywords: Floating-point unit; Coprocessor; Out-of-order execution; Embedded system

1. Introduction

The embedded systems are the preferred choices of major semiconductor companies and mobile device manufacturers since they need a simple, light, and low power micro-controller rather than a high-performance microprocessor. There are numerous examples of embedded processors in the market. For example, ARM9 and ARM 10 architectures from ARM, SH4 and SH5 architectures from HITACHI, and VR5000/5500 series from NEC Electronics. Those embedded processors have different characteristics for specific applications.

The SH4 processor is a two-way superscalar processor for home entertainment game consoles. It has four pipelines: integer, floating-point, load/store, and branch. It supports in-order issue, in-order exception, and out-of-order completion. Especially, it supports 3D graphics operations with a floating-point vector unit (Inner-production unit) [1]. The latest VR5500 series processor has about two times of pipeline resources compared with the SH4 processor. It is a two-way superscalar processor and has six pipelines: two for integer, two for floating-point, load/store, and branch. It supports out-of-order issue, and out-of-order completion [2]. On the other hand, ARM9 and ARM10 series processors are not for high performance, but for small area and low power consumptions. Every ARM core has a general integer pipeline which executes integer, load/store, and branch instructions. But they can expand their architecture with a coprocessor interface on a single chip. For example, VFP10 (Vector Floating-point coprocessor) can supports single and double precision floating-point arithmetic and has a 5-stage load/store pipeline and a 7-stage execution pipeline [3]. The expandable architecture is a more important feature in the embedded systems for various applications.

The latest design goal of a micro-controller is to achieve lower power and higher performance within certain constraints. Flexible configurations of peripheral devices such as a floating-point unit (FPU), caches, and other I/O blocks, should be provided to meet the requirements for various applications. A coprocessor like an FPU is an auxiliary device that may or may not be used according to

* Corresponding author. Tel.: +82-2-2123-2715; fax: +82-2-365-2579.
E-mail addresses: chjeong@kurene.yonsei.ac.kr (C.H. Jeong), chan@kurene.yonsei.ac.kr (W.C. Park), hantack@kurene.yonsei.ac.kr (T.D. Han), yang@mythos.yonsei.ac.kr (S.B. Yang), mklee@spark.yonsei.ac.kr (M.K. Lee).

0141-933/03/$ - see front matter © 2003 Published by Elsevier Science B.V.
doi:10.1016/S0141-9331(03)00023-1

the requirement of the applications. Therefore, the control interface for a coprocessor should be simple and flexible. In addition, a coprocessor for an embedded system should consume low power while maintaining high performance within certain constraints, if possible.

Generally, an FPU has multiple data-paths such as a floating-point adder, a floating-point multiplier (FMUL), and a floating-point divider (FDIV). The latencies of a floating-point operation are varied with arithmetic instructions in the execution pipelines [4]. If the in-order execution control scheme is applied in designing an FPU controller, the overall performance may be degraded considerably due to the long latency instructions such as floating-point division (FDIV) and floating-point square root. Thus the out-of-order execution control scheme is essential to achieve high performance. Scoreboarding and Tomasulo's algorithms, which need special hardware blocks such as scoreboard registers and an instruction queue, can support out-of-order execution and completion [5]. Scoreboarding is an instruction issue control mechanism with a big central scoreboard register, which stores information of instruction status, functional unit status, and register status. The performance of scoreboarding can be limited by the number of scoreboard entries, the number and type of functional units. Also, Tomasulo's approach uses distributed reservation stations instead of a central scoreboard and its dependency checking and execution control are done by the reservation station [6]. However, the design cost and complexity of these techniques are too high for microcontroller applications.

In this paper, a constraint-based dynamic control scheme is proposed for a coprocessor with a data dependency checking, a resource conflict checking, and an exception prediction. Constraint-based dynamic scheduling is similar to a reduced set of scoreboarding. A data dependency checking and pipeline resource checking are done by a small amount of information settings on the register file and the instruction decode unit. It has advantages of area because it needs small register bits for instruction control to achieve out-of-order execution instead of a scoreboard or a reservation stations. Also an exception prediction technique can eliminate a special hardware unit like a reorder buffer. All operands of arithmetic instructions are checked for exception at the first stage of the pipeline. If an exception occurs, the coprocessor executes a checked instruction in order for handling the exceptional condition properly, otherwise the coprocessor performs the instructions out of order. With this technique, the coprocessor is able to execute instructions out of order without extra hardware blocks, if the coprocessor does not generate an exception. Also the exception prediction technique eliminates a special hardware unit like a reorder buffer for precise exception.

The proposed constraint-based dynamic control scheme has been used for the design of a floating-point coprocessor for an embedded system [7]. A coprocessor has been implemented with a standard-cell library to save the design time and cost of its implementation. A hard-macro block is used for a large conventional block—fraction multiplier, barrel shifter, adder, and subtractor.

The rest of this paper is organized as follows. Section 2 illustrates the architecture of a coprocessor. Section 3 describes the out-of-order execution control scheme. Some of examples of coprocessor instruction execution are given in Section 4. Section 5 describes the implementation details. Section 6 presents the conclusions.

2. The architecture of a coprocessor

A coprocessor is composed of a hardware floating-point arithmetic and logic unit (FALU), an FMUL, and an FDIV as shown in Fig. 1. It has an independent instruction decoder, a load/store unit, a register file and a coprocessor interface unit. It can execute several instructions simultaneously within some constraints and can execute the instructions out of order if an arithmetic exception is not generated by the current instruction. It has a reduced interrupt recovery mechanism with a simple coprocessor interface and an exception prediction technique.

The coprocessor does not need any hardware unit for the memory access. Instead, it can perform data transfer only with the host processor and the memory through a coprocessor interface. Thus the host processor takes the responsibilities for instruction fetching, address generation and data arrangements for the memory read or write operations. Since it has neither an instruction fetch unit nor a memory address generation unit, the design complexity of a coprocessor is quite low.

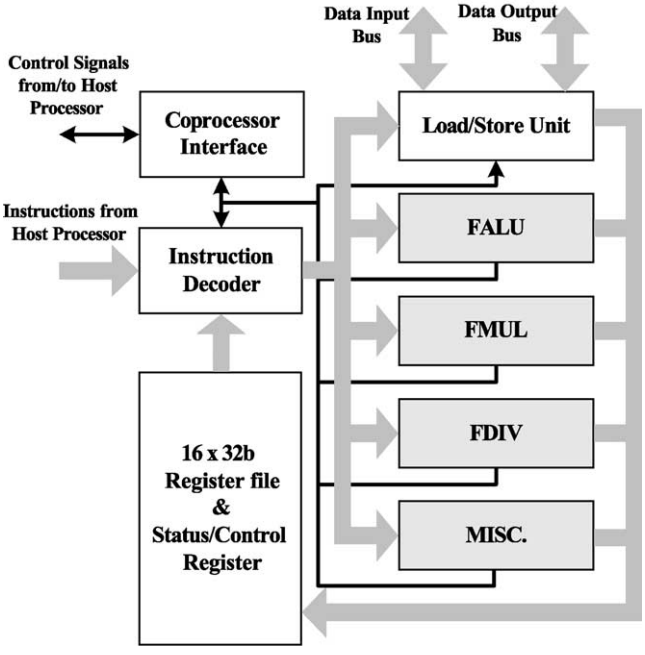


Fig. 1. The block diagram of the coprocessor.

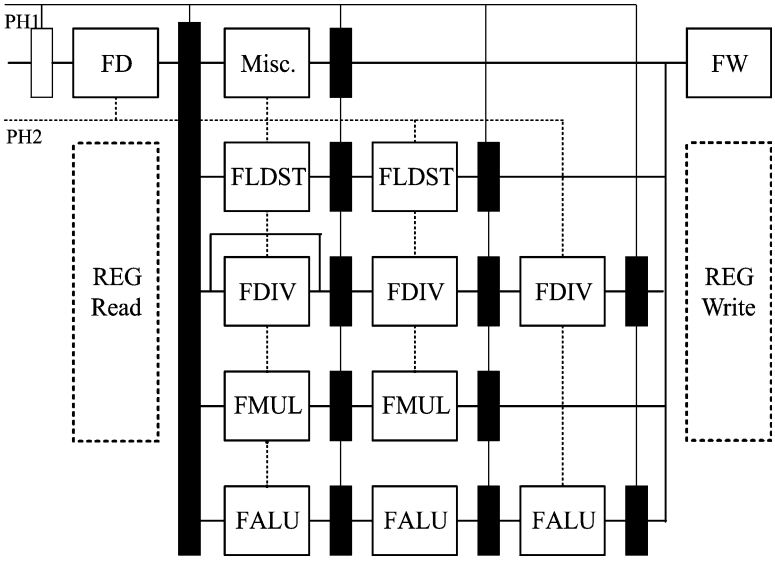


Fig. 2. The pipeline diagram of the coprocessor.

2.1. The FPU execution pipelines

Fig. 2 shows the pipeline layout of the coprocessor. It has five separate pipelines; the floating-point ALU pipeline, the FMUL pipeline, the FDIV pipeline, the floating-point load/store pipeline (FLDST), and the miscellaneous pipeline (Misc). As shown in Fig. 2, these pipelines have different operation latencies and all pipelines except FDIV are fully pipelined. The first stage of FDIV has an iterative path that takes 15 clock cycle latencies. Also each arithmetic pipeline has an exception prediction unit at the first pipeline stage, respectively.

FALU is composed of a floating-point addition (FADD)/-subtraction (FSUB) unit and a comparison unit. FADD/FSUB takes four processing steps: alignment, fraction addition/subtraction, normalization, and rounding. In order to reduce the rounding overhead, the parallel-rounding algorithm proposed in Ref. [8] is adopted. With this algorithm, fraction addition/subtraction and rounding are executed simultaneously at the second pipeline stage.

FMUL consists of the following two stages. At the first stage, floating-point fraction multiplication and addition of partial products are performed. At the next stage, fraction rounding and normalization are executed. It is designed with a conventional FMUL algorithm. In order to reduce the design time and efforts, a hard macro in a target library is used for an integer multiplier.

FDIV has an iterative stage and a couple of non-iterative stages. At the first stage, the radix-4 SRT division algorithm is used [9,10]. At the second stage, quotient addition is performed. Rounding and normalization are performed at the final stage.

FLDST has two stages to comply with the memory access stage (MEM) in the host processor. At the first stage of FLDST, there is no operation because the coprocessor has no memory access stage, but at the next stage a data read or write operation is executed through the data input/output buses with the help of host-coprocessor interface signals. The host and coprocessor interface scheme is described in the next section. The Misc pipeline can execute register move, absolute value, negation value and constant (0.0 or 1.0) load operations.

2.2. Instruction execution

At first, the host processor reads an instruction from the memory and checks the type of the instruction. If the instruction is a coprocessor instruction, the instruction code is transferred directly to the coprocessor with several control signals. The coprocessor decodes the received instruction and executes it on an appropriate execution pipeline. For load/store operations, the host processor performs the memory address generation and data arrangement for transferring data to the coprocessor or to the memory. The coprocessor has the only responsibility for data arrangement and for data transfer to the host processor through the data bus.

The host processor can continue issuing instructions to the coprocessor until a data dependency or a resource conflict on the coprocessor occurs. The issued instructions are executed simultaneously in the execution pipelines and their operations are completed out of order if a resource conflict is resolved and any of the other pipelines does not generate an exception prediction signal. If an exception prediction signal is generated, the host processor stops issuing instructions until the instruction that has generated the exception prediction signal completes its execution. If an exception prediction is false, the host processor continues the program execution.

At the FW (FPU Write-back) stage, if two or more write-back data are available, only one write-back data is selected among these five pipelines according to the predefined

priority and a pipeline stall signal is generated to stop the pipeline advance of other pipelines. On the next cycle, the write-back data with the highest priority is advanced to the FW stage.

3. The out-of-order execution control scheme

This section describes the host-coprocessor interface, the proposed out-of-order execution control scheme, and the precise exception support. A simple and effective host-coprocessor interface unit can support precise exception as well as the dynamic control scheme with some constraints. The host processor issues instructions sequentially, but the issued instruction can be executed out of order within the coprocessor.

3.1. The coprocessor interface

The coprocessor needs to be synchronized in some cases with the host processor; for example, instruction issue may not be allowed when the coprocessor has suffered a stall and cannot get further instructions from the host processor, the host processor is stalled and cannot provide the data for coprocessor data transfer instruction, etc [11]. Therefore, the special control signals are defined to communicate with the coprocessor and the host processor: these signals are STXEN, STMEN, STWEN, COPXEN, COPMEN, and COPWEN as shown in Fig. 3. These signals are active in the low state, where 'ST' indicates the host processor status; 'COP' denotes the coprocessor status; and 'X', 'M' and 'W' stand for execute stage, memory stage, and write-back stage, respectively. The last two letters 'EN' denotes enable. For example, if STMEN is high, the instructions in the host processor can advance to the next memory stage; if it is low, the instruction in the host processor cannot advance to the next stage. If COPXEN is in the low state, some pipeline stall conditions such as data dependencies, resource conflicts, and pipeline full have occurred. Instruction issue must be stopped until COPXEN goes high. Instruction issue control is, therefore, accomplished with these control signals.

3.2. The constraint-based dynamic control scheme

The constraint-based dynamic control scheme provides out-of-order execution with data dependency checking, resource conflict checking, and exception prediction technique with the host interface signals. This scheme controls instruction issue from the host processor according to some constraints such as data dependencies, resource conflicts, and exceptions. The instruction that is received from the host processor is decoded and is checked for data dependencies and resource conflicts. If a data dependency is found, a COPXEN signal is generated to stop instruction issue. As soon as the data dependency is resolved, the COPXEN signal goes high.

If a resource conflict is found, a COPXEN signal is used to stop instruction issue as in the case of a data dependency. There are two possible resource conflicts in the coprocessor. The first one occurs when two or more execution pipelines attempt to advance simultaneously to the coprocessor write-back stage. Since only one write-back data can advance to the write-back stage at each clock cycle, other instructions in the pipelines must wait until the next clock cycle. Write-back data selection is scheduled based on the pre-defined

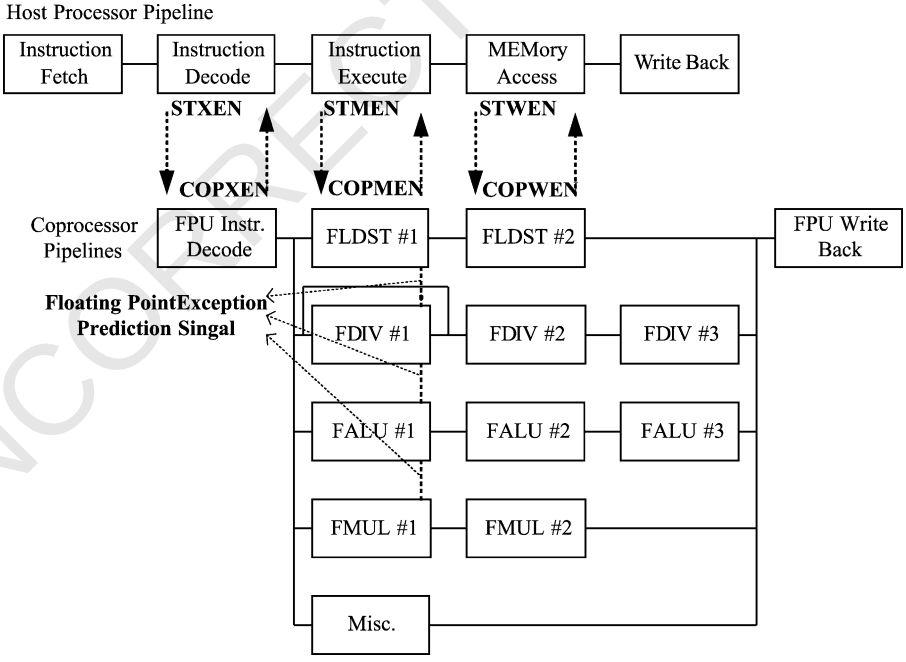


Fig. 3. The host and coprocessor interface signals.

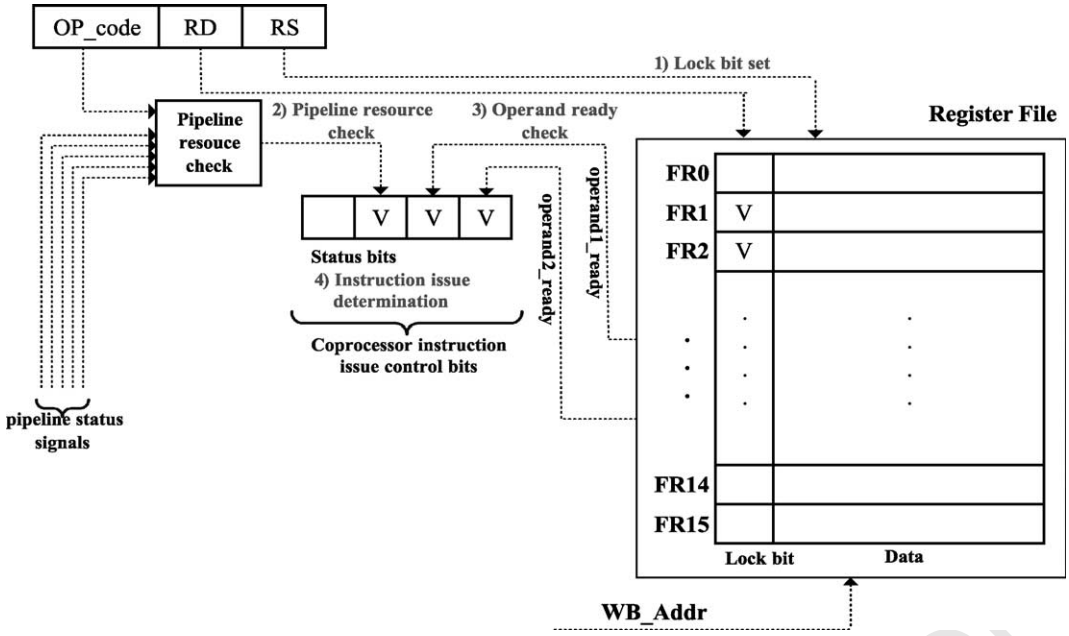


Fig. 4. The constraint-based dynamic control scheme.

priority. The second one happens when one pipeline is full with instructions due to a lower priority; no more instructions that use the pipeline can be executed. But an instruction that uses a empty pipeline can be executed if a data dependency has not occurred. A pipeline resource conflict is controlled with COPXEN signals for instruction scheduling in the host processor.

To implement the constraint-based dynamic control scheme effectively, the register lock bits, the pipeline status bit, and the operand ready bits are added in the control block as shown in Fig. 4. The register lock bits show whether or not the register is the write-back destination of any instruction. The pipeline status bit shows which pipeline is available for instruction execution. That is, it shows that the pipeline is full of instructions or has empty execution stages. The operand ready bits show whether or not two source operands are available for execution. That is, if data are forwarded from the FW stage or if the lock bits of two source registers are not set, the operand ready bit is set for instruction execution. Note that a coprocessor instruction is two-operand type.

Data-dependency checking is achieved by the lock bit setting in the register file and checking operand ready bit at the decode stage. First, the register destination (RD) of the current the instruction determines the lock bit position in the register file. Then the operand read bit reads the lock bit of the register on the current instruction. If the lock bit of register source (RS) is set by the previous instruction, a data dependency has occurred.

As soon as the coprocessor instruction is decoded, the lock bit of the destination register is set for a data dependency checking. And then the pipeline status bit and the operand ready bit determine whether the instruction is issued or not. If one of these bits is not set, a data dependency or a resource conflict is found. Both the coprocessor and the host processor cannot issue the instruction to the coprocessor until these bits are set, that is, the data dependency has been resolved or write-back scheduling is completed. After the execution of the previous instruction the result of the execution is stored into the destination register, and the lock bit of RD is cleared. The next instruction can then be executed.

As shown in Fig. 5, the first FADD instruction and the first FMUL instruction have a data dependency. As soon as the FMUL instruction is decoded, we set the lock bit of the destination register (FR2) and read operand ready bits. But, because the source register of the current instruction (FR0) is the destination register of the previous instruction, one of the operand ready bits is not set. Therefore, the coprocessor generates an interface signal to stop instruction issue and waits for the data dependency resolving. After the first FADD instruction has been executed, the result can be forwarded so that instruction execution and instruction issue can be performed.

For the data load/store operation, the load/store pipeline is designed to communicate with a host processor pipeline. Because the coprocessor cannot access the memory, the host processor reads data from the memory and writes the data to the global data bus, and then the coprocessor reads this data when a load operation for some coprocessor data is executed. For the coprocessor data store operation, the host processor generates the memory addresses and the coprocessor writes the data to be stored to the global data bus. In general the load/store operation could not be done in one-clock cycle when a cache miss occurs. Therefore, the host processor generates coprocessor interface signals to stall the coprocessor on a cache miss. In this case the load/store operation can take multiple clock cycles. But, in case of a cache hit, the load/store operation does not

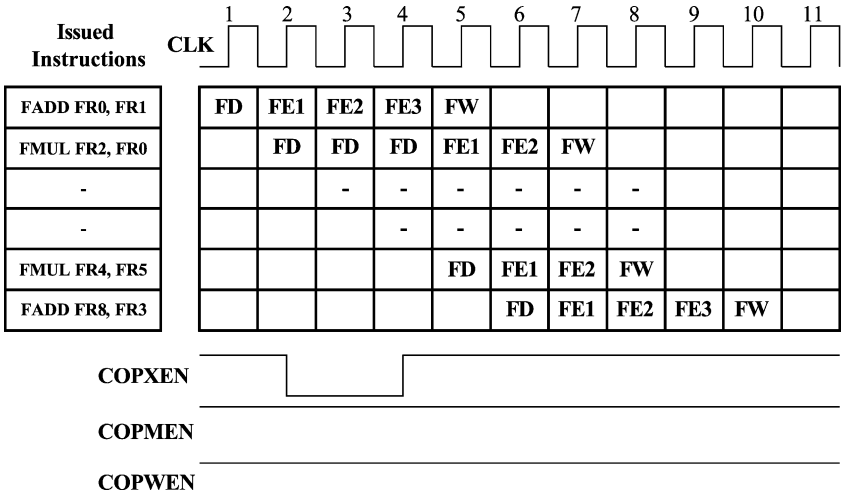


Fig. 5. The pipeline diagram of data dependency case.

generate special control signals by the host processor. On a cache miss, however, the coprocessor must wait for the end of the memory access and data preparation in the host processor. In that case, one or more control signals from the host processor go low to stall a coprocessor load/store pipeline (STMEN, STWEN), but other execution pipelines in the coprocessor can execute another instructions if any of a data dependency, a resource conflict, and an exception prediction does not occur.

3.3. Exception predictions and precise exception

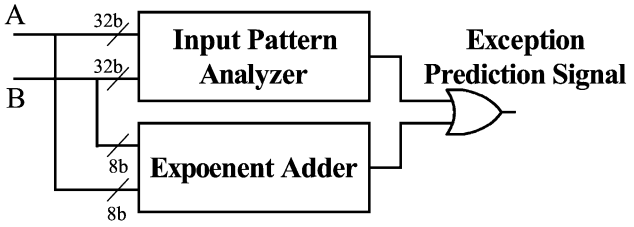
In the proposed architecture an exception prediction technique is employed for precise exception. At the first stage of the arithmetic pipeline, each instruction is checked for the possibility of a exception. If the instruction can make an exception, an exception prediction signal is generated. This signal makes the host processor stall with COPMEN and COPXEN signals. At the last stage of the arithmetic pipeline, a true exception is generated. If any exception does not occur, the host processor continues issuing instructions; otherwise the host processor jumps to the coprocessor a exception handling routine.

The coprocessor has no special exception recovery unit, while each instruction is checked at the first stage of the corresponding execution pipeline. No further instruction can be issued if an exception prediction signal is active. Thus if all exception prediction signals of the execution pipeline are inactive, it guarantees that arithmetic exceptions never happen on the execution of the instruction. This technique has an advantage in area and design costs because a special hardware unit like a reorder buffer is not required.

Exception predictors are designed individually for each arithmetic pipeline. Exception prediction is to check the result of a floating-point operation for a exception at the first stage of the execution pipeline as shown in Fig. 6. An exception is predicted by the input pattern checking and exponent calculation. First, the input pattern checking examines the abnormal conditions of an input operand. That is, if the input number is not a number (NaN) and it is negative or positive infinity ($\pm$Inf) on FADD, an invalid exception occurs. If the divisor is floating-point zero on a FDIV, the division by zero exception occurs. Second, the possibility of an overflow or underflow exception is checked by the exponent calculation of the two input numbers. For example, on a FMUL, if the addition result of an input number exceeds the maximum value of the corresponding exponent (254, in case of a single precision floating-point number), an overflow exception should be generated. If the result of the exponent addition is equal to the maximum value of the corresponding exponent, an overflow exception may be generated after the rounding stage. These two cases are included in determining exception prediction.

4. Execution examples

This section describes the execution of some coprocessor instructions with a sample code. In the following pipeline diagram, the letter 'FD', 'FE*x*', and 'FW' mean 'Decode',



$$\text{if} \left\{ \begin{array}{l} m_a = \textit{DenormalizedNumber or NaN or} \pm \textit{INF} \\ m_b = \textit{Denormalized Number or NaN or} \pm \textit{INF or } 0 \end{array} \right\}$$

$$\text{if} \left\{ \begin{array}{l} e_a + e_b \geq MAX\_e \textit{ or } e_a - e_b \geq MAX\_e \\ e_a + e_b \leq MIN\_e \textit{ or } e_a - e_b \leq MIN\_e \end{array} \right\}$$

Fig. 6. The exception prediction.

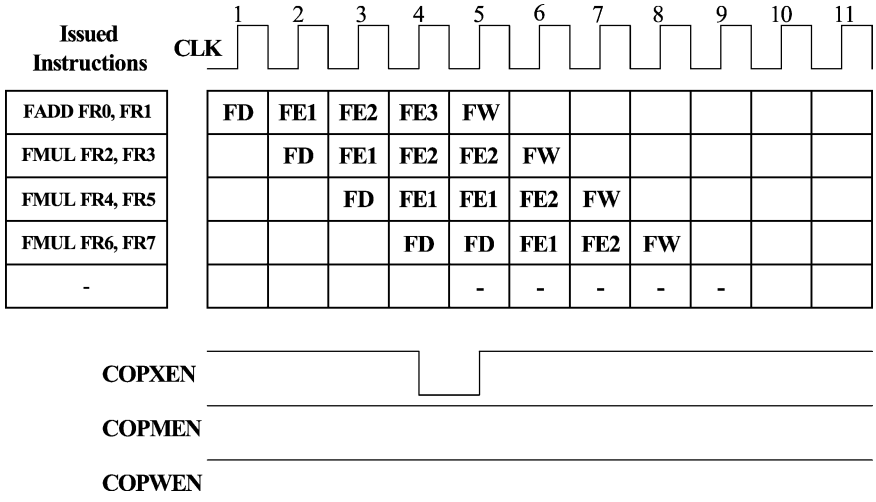


Fig. 7. The pipeline diagram of the exception impossible case.

'*x*-th Execute' and 'Write-back' stages in the coprocessor, respectively.

4.1. Exception impossible case and exception possible case

An example in Fig. 7 shows the pipeline of the constraint-based dynamic scheduling in the case of a resource conflict, assuming that all instructions are predicted as exception impossible and that the write-back priority is FLST > FDIV > FALU > FMUL > Misc. As in Fig. 6, the first FMUL instruction cannot advance to the FW stage because of low priority at the 4th clock. The second FMUL instruction cannot also go further because the next pipeline stage is not empty. The third FMUL instruction cannot advance either, although all instructions have no data dependencies. The coprocessor generates a COPXEN signal to stop instruction issue by the host processor.

The next example in Fig. 8 shows an exception possible case. We assume that the FADD instruction has an exception possible operand. At the first execution pipeline stage of the FADD instruction, the exception prediction signal is generated, and then a COPMEN signal is set to stop instruction execution in the host processor. Although there is no data dependency between the FADD instruction and the first FMUL instruction, the FMUL instruction cannot advance to the next pipeline stage to support precise exception. And a COPXEN signal is generated to stop instruction issuing by the host processor until the truth of exception is decided at the last stage of the pipeline. At the 4th clock, the first FMUL instruction starts to execute because an exception is not actually occurred. The following instructions then continue their execution.

4.2. Out-of-order execution examples

Fig. 9 shows some out-of-execution examples in the coprocessor. All the operands for the instructions are

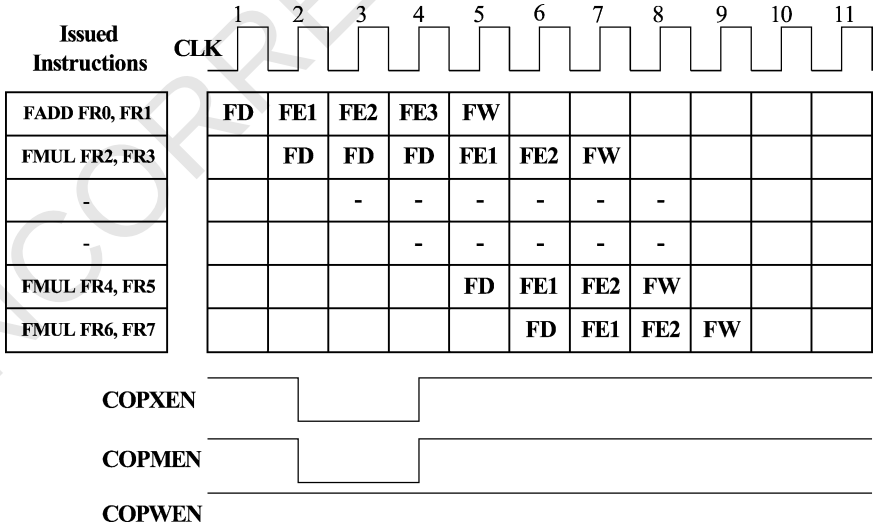


Fig. 8. The pipeline diagram of the exception possible case.

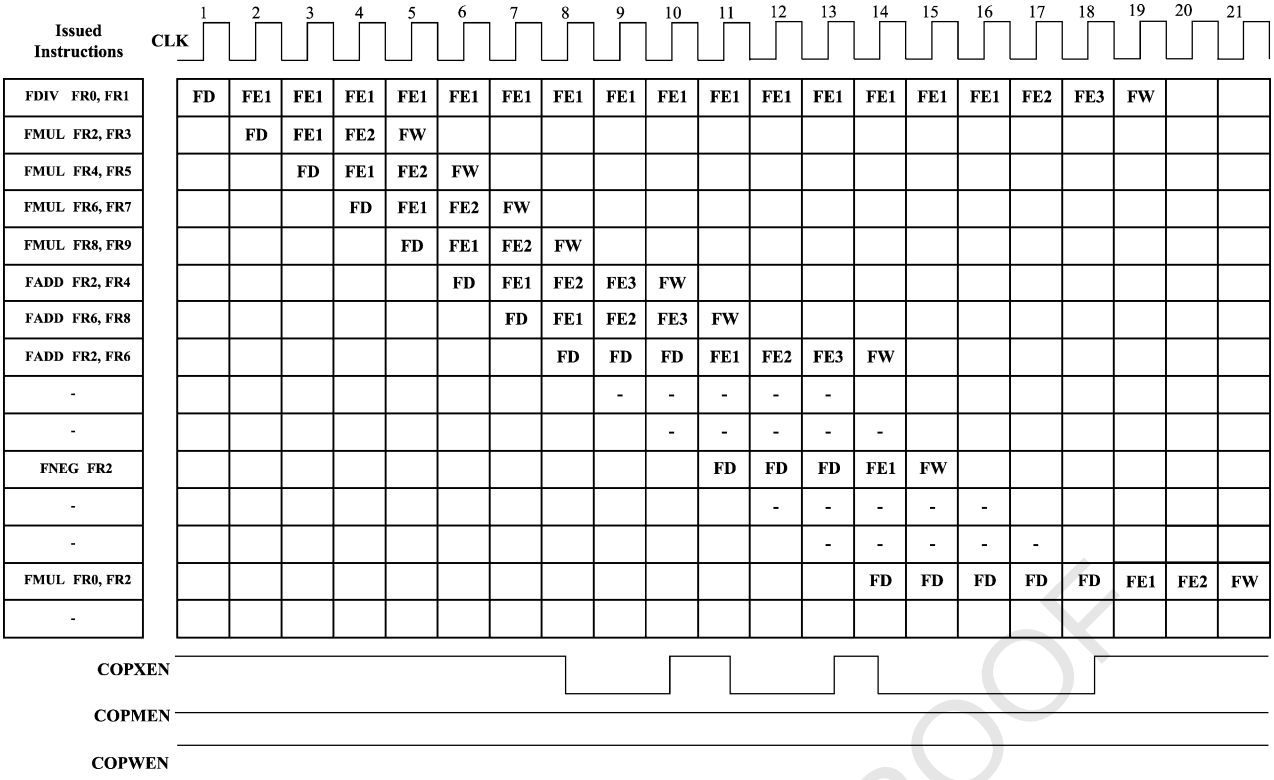


Fig. 9. The pipeline diagram of the out-of-order execution case.

assumed to be safe for an exception. The entire pipeline stall is caused by the data dependency that cannot eliminate due to the sequence of the calculation. With an in-order execution control scheme no other coprocessor instruction can be executed during the FDIV instruction execution and it wastes the execution clock cycles. However, as in Fig. 8, the execution clock cycle loss can be alleviated with the out-of-execution control scheme and the code optimization.

5. Implementation

The coprocessor has been designed with a 0.25 μm standard-cell library which has four metal layers. The

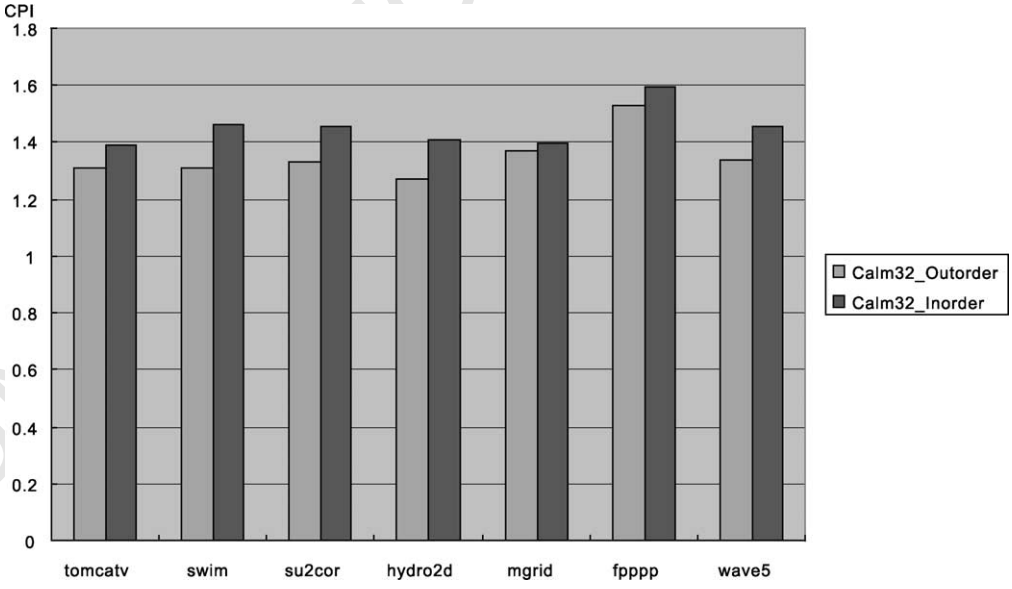


Fig. 10. The performance comparison of in-order issue and out-of-order issue processors the CalmRISC32.

Table 1
The comparison of instruction issue and execution control

Processors	ARM10 + VFP10	Calm32 + Coprocessor	SH4	VR5500
Inst. issue	In-order	In-order	In-order (2-way)	Out-of-order (2-way)
Inst. execution	In-order	Out-of-order	Out-of-order	Out-of-order
Pipeline resources	1 Integer[a], 1 FPU	1 Integer, 1 FPU	1 Integer, 1 FPU, 1 Brach, 1 Load/Store	2 Integer, 2 FPU, 1 Branch, 1 Load/Store

[a] ARM10 core can simultaneously execute data processing instructions under the load/store multiple instructions.

coprocessor supports 32 bit single precision floating-point arithmetic instructions; FADD, FSUB, FMUL, FDIV, format conversion (FTOI, ITOF), comparison (FCMP), rounding (FRND), load, store (CLD), etc. The IEEE-754 standard rounding mode and all exception conditions are also supported. For the development of a coprocessor, the behavioral HDL model has been implemented and verified with a simple test vector. All arithmetic data-paths have been verified by comparing the result of the data-path calculation with that of a C program. Then the Synopsys design analyzer has generated a gate-level model. Finally, timing verification with the Samsung's in-house CAD tool, *CubicWare*, has been performed.

5.1. Performance evaluation

To evaluate the performance potential of the out-of-order execution coprocessor, an architectural simulation is conducted with the SimpleScalar Tool set [12]. And the sim-outorder simulator is modified with a CalmRISC32 host processor and the coprocessor architecture. The SPEC floating-point benchmark program is also used for simulation. It can estimate the performance potential of the out-of-order execution coprocessor. Fig. 10 shows the simulation results in CPI (Clock per Instruction). Because the host processor CalmRISC32 is an in-order issue processor, Calm32_Inorder shows the performance of the out-of-order execution coprocessor, and Calm32_Outorder shows out-of-order issue and out-of-order execution results.

Observe that the result of the out-of-order issue and out-of-order execution architecture simulation is not greatly improved. Because the CalmRISC32 host processor has instruction issue limitation and the amount of its pipeline resources is less than the other processor. Therefore, within limited resources the in-order issue and out-of-order execution control scheme can be an appropriate choice.

To implement the out-of-control scheme, scoreboarding uses a centralized scoreboard register and scoreboard control. And Tomasulo's algorithm needs an instruction queue and a separate reservation station for each arithmetic pipeline. Constraint-based dynamic scheduling, however, needs a small number of registers for lock bits, pipeline status bits, and a resource checker. It has advantages of area and design complexity because of efficient dependency checking as well as simple resource checking method.

In Table 1 the comparison of instruction issue and execution control scheme is provided. In general, ARM series processors are optimized for small area and power consumption. Hence, the instruction issue and execution control scheme are not greatly considered. Because those control schemes need special storage elements; both increase area and power consumption would be increased. On the other hand, a high performance embedded processor like VR5500 series processor has multiple pipeline resources, and for the better use of those resources, the reservation station of 20 entries is used for the out-of-order execution control. SH4 processor has a moderate approach for out-of-order execution. Only a pair of possible combination of instructions is checked for instruction issue in the instruction fetch stage. With such a constraint, SH4 can reduce area and control overheads, sacrificing the performance. CalmRISC32 has a minimum effort for out-of-order execution. It issues instructions in-order and executes instructions in-order in the host processor. A coprocessor instruction is issued sequentially by the host processor, but the coprocessor can execute coprocessor instruction out-of-order. It can effectively execute other instructions during the long latency floating-point instruction such as FDIV instruction. It should provide performance improvement potentials for the floating-point intensive applications and give program optimization opportunity.

6. Conclusion

In this paper a floating-point co-processor for an embedded system that supports out-of-order execution control with the constraint-based dynamic control scheme has been designed and verified with the standard cell library. The proposed scheme is achieved by the data dependency checking at the decode stage, the resource conflict checking at the write-back stage, and the exception prediction at the first stage of each arithmetic pipeline. It has an advantage in area and design costs because a special hardware unit for out-of-order execution control like an instruction queue or a scoreboard register is not required. For precise exception, the coprocessor instruction stream is scheduled by the simple host

interface and the exception prediction technique, which can eliminate a reorder buffer. It has been applied to a floating-point coprocessor which is a RISC-type coprocessor. The coprocessor instructions can be executed simultaneously in all pipelines and can be completed out of order. The performance of the coprocessor can be improved with the constraints-based dynamic control scheme and code optimization.

References

[1] F. Arakawa, O. Nishi, K. Uchiyama, N. Nakagawa, SH4 RISC multimedia microprocessor, IEEE Micro 18 (2) (1998) 26–34.

[2] User's Manual VR5500™ 64/32-Bit Micro processor, NEC electronics, 2002, pp. 25–38.

[3] S. Futber, ARM System-on-Chip Architecture, Addison-Wesley, London, 2000, pp. 317–346.

[4] A.R. Omondi, Computer Arithmetic Systems: Algorithms Architecture and Implementation, Prentice-Hall, New York, 1994.

[5] J. Hennessy, D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufman, San Francisco, 1996, pp. 240–261.

[6] R.M. Tomasulo, An efficient algorithm for exploiting multiple arithmetic units, IBM Journal of Research and Development 11 (1) (1967) 25–33.

[7] C.H. Jeong, W.C. Park, S.W. Kim, T.D. Han, M.K. Lee, In-order issue out-of-order execution floating point coprocessor for CalmRISC32, 15th IEEE Symposium on Computer Arithmetic (2001) 195–200.

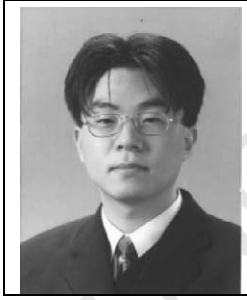
[8] W.C. Park, S.W. Lee, O.Y. Kwon, T.D. Han, Floating-point adder/subtractor performing IEEE rounding and addition/subtraction in parallel, IEICE Transaction on Information and System E79-D (4) (1996) 297–305.

[9] D.E. Atkins, Higher-Radix division using estimates of the divisor and partial remainder, IEEE Transactions on Computers 17 (10) (1968) 925–934.

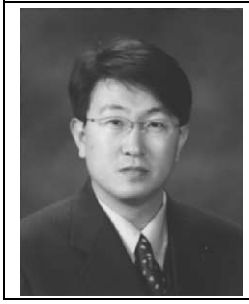
[10] S. Oberman, M. Flynn, Division algorithms and implementations, IEEE Transaction on Computers 46 (8) (1997) 833–854.

[11] S.Y. Cho, S.H. Park, Y.G. Kim, S.W. Jeong, B.Y. Chung, H.L. Roh, C.H. Lee, H.M. Yang, S.H. Kwak, M.K. Lee, CalmRISC™-32: a 32-bit low-power MCU core, The Second IEEE Asia Pacific Conference on ASICs (2000) 285–289.

[12] D. Burger, T. Austin, The SimpleScalar Tool Set, Version 2.0, Computer Science Department, University of Wisconsin, Madison, 1997.

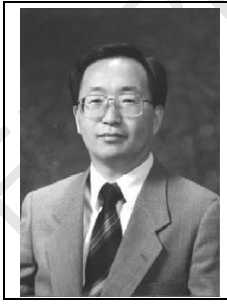


Cheol-Ho Jeong is a graduate student in the Department of Computer Science at Yonsei University, Korea. He is currently researching computer arithmetic and system for 3D computer graphics. He received an MS in computer science from Yonsei University.



Woo-Chan Park is a research professor in the Department of Computer Science at Yonsei University, Korea. His research interests include 3D computer graphics accelerator architecture, micro-architecture, and computer arithmetic. He received a PhD in computer science from Yonsei University.

Tack-Don Han is a professor in the Department of Computer Science at the Yonsei University, Korea. His research interests include high performance computer architecture, media system architecture, and wearable computing. He received a PhD in computer engineering from the University of Massachusetts.



Sung-Bong Yang is an associate professor in the Department of Computer Science at Yonsei University, Korea. His research interests include Mobile system, 3D computer graphics, and Electronics Commerce. He received a PhD in computer science from the University of Oklahoma.

Moon-Key Lee is a professor in the Department of Electrical Engineering at Yonsei University, Korea. His research interests include VLSI design, CAD and embedded system. He received a PhD in electrical engineering from Yonsei University and a PhD in electrical engineering from the University of Oklahoma.