Formal Design Verification for Correctness of Pipelined Microprocessors with Out-of-order Instruction Execution

Takashi Takenaka, Junji Kitamichi,
Teruo Higashino and Kenichi Taniguchi
Department of Informatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University
Toyonaka, Osaka 560–8531, Japan
e-mail:{t-takena,kitamiti,higashino,taniguchi}@ics.es.osaka-u.ac.jp

Abstract— In this paper, we propose a verification method for pipelined microprocessors with out-of-order execution. We define a class of pipelined microprocessors with out-of-order execution and give a sufficient condition that guarantees the correctness of implementation. Each microprocessor in this class has a pipeline $\mathbf{stg}_1, \ldots, \mathbf{stg}_n$ such that the stages $\mathbf{stg}_c, \ldots, \mathbf{stg}_n$ are so-called "*in-order* pipeline" and changes the execution order of instructions within the stages of $\mathbf{stg}_1, \ldots, \mathbf{stg}_{c-1}$. Using our method, we carried out the correctness proof of a practical 6-stage pipelined microprocessor that has a so-called scoreboard[1]. We used a verifier having a decision procedure for Presburger sentences. The total CPU time spent in the proof was about 8 hours.

I. Introduction

Out-of-order instruction execution[1] is one of the techniques to overcome data hazards with dynamic scheduling. Therefore almost all modern pipelined microprocessor architectures, such as the PowerPC and the DEC Alpha, employ it. Recently, some efforts have gone into verifying *out-of-order* architectures[2, 3, 4]. In those methods, however, the class of the implementations that the method can be applied for is not defined clearly. It is exceedingly important to define a suitable class of the implementations that the method can be applied for and to give a sufficient condition to guarantee that a given implementation in this class satisfies its specification.

In this paper, we give a design constraint C in order to specify a class of the pipelined microprocessors with out-of-order instruction execution. Moreover we propose a sufficient condition V to guarantee that a given implementation designed under the design constraint C satisfies its specification.

The *design constraint* C claims as follows. First, each instruction must execute the stages stg_1, stg_2, ..., stg_n in this order, provided that it is possible for each instruction to abort its execution before executing a certain stage stg_c, Secondly, the stages $\mathrm{stg}_c, \ldots, \mathrm{stg}_n$ are so-called "*in-order* pipeline". Therefore, the implementation changes the execution order of instructions within the stages of $\mathrm{stg}_1, \ldots, \mathrm{stg}_{c-1}$. Moreover, w.r.t each visible register (except for program counter), the order of instructions that read values from/write values into this register is the same as the order of instructions which execute the stage stg_c. The microprocessors in this class can have data forwarding, speculative instruction fetch, and so on.

The *sufficient condition* V claims that (a) each instruction must not enter the pipeline consisting of the stages $\mathrm{stg}_c, \ldots, \mathrm{stg}_n$ without guaranteeing that it would not cause any of RAW, WAR, WAW hazards, and that (b) each instruction must perform specified operations correctly, and so on. We can check whether a given implementation satisfies condition (a) by showing directly that the control of the implementation guarantees this condition without considering the values calculated by each instruction.

We can check condition (b) under the assumption that each instruction can read correct values without hazards. Moreover, since we assume that the stages $\mathrm{stg}_c, \ldots, \mathrm{stg}_n$ are "*in-order* pipeline" by C, we can use the method similar to ones proposed for in-order architecture[5].

Moreover, we designed a practical pipelined microprocessor with out-of-order instruction execution under the design constraint C. This microprocessor has 6-stage pipeline and some buffers to record the program order of instructions and so on. We proved that this microprocessor satisfies the sufficient condition V as follows. First, we introduced lemmas concerned with properties of the buffers and so on, and proved them. Secondly, we proved that the implementation satisfies the sufficient condition V under the lemmas. To prove those, we used a decision procedure for the prenex normal form Presburger sentences bounded by only universal quantifiers[6]. Although it was necessary to decide the truth of the sentence whose length[1] was over 8,000, the total CPU time spent in the proof was the practical time of 8 hours.

II. Specification

We assume that a pipelined microprocessor has a program counter(PC) and some registers and memories as visible registers. All instructions, which the microprocessor fetches and executes, are stored in an instruction memory(IMEM). We assume that IMEM keeps its contents unchanged in execution. The instruction set is supposed to be so-called RISC-type ISA. Each instruction consists of an op-code and some operands. It is supposed that the registers that an instruction reads values from/writes values into are specified by its operands. In the

[1]The length of a sentence is the total number of occurrences of the variables, constants, operations($+$, $-$), predicates($>$, $=$), and logical connectives($\wedge$, $\vee$, $\neg$) in the sentence.

following discussion, let $\mathcal{F}^{src}(\mathcal{I})$, $\mathcal{F}^{dest}(\mathcal{I})$ denote the set of the registers that instruction $\mathcal{I}$ reads values from/writes values into, respectively.

The description(abstraction) level of specifications is so-called ISA level. We write *cycle* to denote the the transition function of the specification S. The transition function *cycle* specifies the operations of the microprocessor executing each instruction independently. Let $\sigma(t^s)$ denote the state after the transition *cycle* is executed from the initial state t^s times. We abbreviate $\sigma(t^s)$ as t^s and the state after *cycle* is executed from the state t^s as *cycle*(t^s), respectively.

III. Implementation

We assume that an implementation has special registers and controller to enable out-of-order execution, and pipeline registers in addition to the visible registers. The datapath is divided into a fixed number of pipelined stages, for example n stages, $\mathrm{stg}_1, \ldots, \mathrm{stg}_n$.

The description(abstraction) level of implementations is so-called RT level. We write *clk* to denote the transition function of the implementation I. The transition function *clk* specifies the operations that are executed in each stage during a clock cycle. Let $\sigma(t)$ denote the state after the transition *clk* is executed from the initial state t times. We abbreviate $\sigma(t)$ as t and the state after *clk* is executed from the state t as *clk*(t), respectively.

We give a design constraint C in order to specify a class of the pipelined microprocessors as follows.

(C1) Each instruction must execute the stages stg_1, stg_2, ..., stg_n in this order. However, it is possible for each instruction to abort its execution before executing a certain stage stg_c.

(C2) For each stage stg_i such that $c \leq i \leq n$, the order of instructions that execute this stage is the same as the order of instructions that execute the stage stg_c.

(C3) For each register F, there exists only one stage stg_{i_F} such that all instructions can write values into F at executing the stage stg_{i_F} independently of the kind of instructions. Provided that for each visible register, $c \leq i_F \leq n$. For PC, there exist two stages, stg_1 and stg_p such that $c \leq p \leq n$.

(C4) W.r.t each register F, all instruction can read values from F only during executing between stg_1 and stg_{i_F+1}. However w.r.t each visible register(except for PC), during executing between stg_c and stg_{i_F+1}.

The implementation designed under the design constraint C changes the execution order of instructions within the stages of $\mathrm{stg}_1, \ldots, \mathrm{stg}_{c-1}$. The stages $\mathrm{stg}_c, \ldots, \mathrm{stg}_n$ are *in-order* pipeline. Moreover, w.r.t each visible register (except for PC), the order of instructions that read values from/write values into this register is the same as the order of instructions that execute the stage stg_c.

IV. Correctness

In this section, we give a definition that implementation satisfies specification.

Definition 1 *Suppose that a specification S and an implementation I are given and each has the same instruction memory IMEM and the same initial value of each visible register.*

The implementation I satisfies the specification S if for each visible register F_j and for each state t of the specification from the initial state, there exists a value of correspondence function $conc_{F_j}(t)$ such that $conc_{F_j}(t+1)$ follows or equals to $conc_{F_j}(t)$, and that equation

$$F_j^{spec}(t) = F_j^{impl}(conc_{F_j}(t))$$

holds, where $F_j^{spec}(t)$ and $F_j^{impl}(t)$ denote the contents of F in the state t of the specification S and the implementation I, respectively. □

In this paper, we define the correspondence function $conc_{F_j}$ inductively in the following way. Suppose that the value of $conc_{F_j}(t)$ is determined for each visible register F_j and for each state t from the initial time to the state t^s. Let $\mathcal{I}_t$ denote an instruction that is first fetched after $conc_{PC}(t^s)$. If $F_j \in \mathcal{F}^{dest}(\mathcal{I}_t)$, the value of $conc_{F_j}(t^s+1)$ is the state in which $\mathcal{I}_t$ completes to execute the stage stg_{i_F}. Otherwise, the value of $conc_{F_j}(t^s+1)$ is the same as the value of $conc_{PC}(t^s+1)$. However if $conc_{F_j}(t^s)$ follows $conc_{PC}(t^s+1)$, the value of $conc_{F_j}(t^s+1)$ is the same as the value of $conc_{F_j}(t^s)$.

V. Sufficient condition for correctness

A. Sufficient condition

In this section, we give a sufficient condition V for the correctness of microprocessor. We have only to check whether a given implementation satisfies the sufficient condition V in order to verify that this implementation satisfies its specification.

Lemma 1 (Sufficient condition V) *Suppose that an implementation I designed under the design constraint C and its specification S are given. If the implementation I satisfies the following six conditions, the implementation I satisfies the specification S.*

(V1) *Each instruction must enter the pipeline consisting of the stages $\mathrm{stg}_c, \ldots, \mathrm{stg}_n$, provided that every instruction fetched speculatively must abort before executing the stage stg_c.*

(V2) *There are no RAW hazards*[1]. The instruction $\mathcal{I}_j$ must not enter the pipeline consisting of the stages $\mathrm{stg}_c, \ldots, \mathrm{stg}_n$, if there exists the instruction $\mathcal{I}_i$ such that $\mathcal{I}_i$ precedes $\mathcal{I}_j$, $\mathcal{I}_i$ has not executed the stage stg_c, and there exists the register F_k such that $F_k \in \mathcal{F}^{dest}(\mathcal{I}_i)$ and $F_k \in \mathcal{F}^{src}(\mathcal{I}_j)$.

(V3) *There are no WAR hazards*[1]. The instruction $\mathcal{I}_j$ must not enter the pipeline consisting of the stages $\mathrm{stg}_c, \ldots, \mathrm{stg}_n$, if there exists the instruction $\mathcal{I}_i$ such that $\mathcal{I}_i$ precedes $\mathcal{I}_j$, $\mathcal{I}_i$ has not executed the stage stg_c, and there exists the register F_k such that $F_k \in \mathcal{F}^{src}(\mathcal{I}_i)$ and $F_k \in \mathcal{F}^{dest}(\mathcal{I}_j)$.

(V4) *There are no WAW hazards*[1]. The instruction $\mathcal{I}_j$ must not enter the pipeline consisting of the stages $\mathrm{stg}_c, \ldots, \mathrm{stg}_n$, if there exists the instruction $\mathcal{I}_i$ such that $\mathcal{I}_i$ precedes $\mathcal{I}_j$, $\mathcal{I}_i$ has not executed the stage stg_c, and there exists the register F_k such that $F_k \in \mathcal{F}^{dest}(\mathcal{I}_i)$ and $F_k \in \mathcal{F}^{dest}(\mathcal{I}_j)$.

(V5) *Every instruction must perform specified operations correctly.* Let t^s, t denote the state of the specification S and the implementation I, respectively. If there exists the instruction

$\mathcal{I}_t$ that begins its execution in the state t, the following condition holds;

$$\left[\begin{array}{l} PC(t^s) = PC(t) \\ \wedge \bigwedge\limits_{F_j \in \mathcal{F}_v} F_j(t^s) = F_j(clk_{c+1,n}^{(F_j)}(t')) \end{array}\right] \rightarrow \bigwedge_{F_j \in \mathcal{F}_v} F_j(cycle(t^s)) = F_j(clk_{c,n}^{(F_j)}(t')) \quad (1)$$

where $F_j(clk_{c,n}^{(F_j)}(t'))$ denotes a content of the register F_j in which all instructions executing the stages of $\mathrm{stg}_c, \ldots, \mathrm{stg}_n$ would complete their own executions, and $\mathcal{F}_v$ denotes a set of all visible registers except PC.

W.r.t PC, both of following two conditions (a) and (b) hold. (a) If $\mathcal{I}_t$ is a branch that is taken, the following condition holds;

$$\left[\begin{array}{l} PC(t^s) = PC(t) \\ \wedge \bigwedge\limits_{F_j \in \mathcal{F}_v} F_j(t^s) = F_j(clk_{c+1,n}^{(F_j)}(t')) \end{array}\right] \rightarrow PC(cycle(t^s)) = PC(clk_{c,n}^{(PC)}(t')). \quad (2)$$

(b) If $\mathcal{I}_t$ is not a branch or is a branch that is not taken, (b1) $\mathcal{I}_t$ must not write a value into PC in the stage stg_p such that $PC \in \mathcal{F}_{\mathrm{stg}_p}$, (b2) the following condition holds;

$$PC(t^s) = PC(t) \rightarrow PC(cycle(t^s)) = PC(clk_1(t)) \quad (3)$$

where clk_1 denotes the transition function that performs only operations of stg_1. □

We are certain that for the class provided by the design constraint C it is difficult to relax the sufficient condition given above.

B. Proof of sufficientness of condition V

In this section, we briefly prove that when a given implementation I designed under the design constraint C satisfies the sufficient condition V, the implementation I satisfies the specification S. We prove it by induction on the number of instructions executed by the specification S. Since the specification and implementation have the same initial values of each visible register, we can prove the basis step obviously.
(Hypothesis step) Let k denote a state in which the specification S completes the execution of $(k-1)$-th instruction. Suppose that for each visible register F_j and for each state i from the initial state through the state k, there exists the value of $conc_{F_j}(i)$ and equation (4) holds.

$$F_j(i) = F_j(conc_{F_j}(i)) \quad (4)$$

(Induction step) Let $\mathcal{I}_t$ denote the first instruction fetched after $conc_{PC}(k)$ in the implementation I, and let t denote the state in which $\mathcal{I}_t$ is fetched. W.r.t. PC, equation (5) holds since the content of PC is kept unchanged until $\mathcal{I}_t$ completes to execute the stage stg_1 from C2 and **V1**.

$$PC(k) = PC(t) \quad (5)$$

From **V1**, $\mathcal{I}_t$ must execute the stage stg_c. Let t' denote the state in which $\mathcal{I}_t$ executes stg_c. Now, suppose that the implementation I would not execute any instruction following $\mathcal{I}_t$. In this case, let $F_s(clk_{c+1,n}^{(F_s)}(t'))$ denote a content of F_j in which all instructions executing the stages of $\mathrm{stg}_{c+1}, \ldots, \mathrm{stg}_n$ would complete their own execution. W.r.t the value of $F_s(clk_{c+1,n}^{(F_s)}(t'))$, the following property holds under the hypothesis, if the implementation I satisfies **V1**, **V2**, **V3** and **V4**.

Property 1 *An equation $F_s(k) = F_s(clk_{c+1,n}^{(F_s)}(t'))$ holds for each register F_s such that $F_s \in \mathcal{F}^{src}(\mathcal{I}_t)$.* □

From Property 1, equation (5) and **V5**, equation (6) holds w.r.t each register F_d such that $F_d \in \mathcal{F}^{dest}(\mathcal{I}_t)$.

$$F_d(k+1) = F_d(clk_{c+1,n}^{(F_d)}(t')) \quad (6)$$

Even if there exist instructions following $\mathcal{I}_t$, equation (7) holds from the design constraint C since any instruction can not read values written by the instructions that execute the stage stg_c later than them.

$$F_d(conc_{F_d}(k+1)) = F_d(clk_{c+1,n}^{(F_d)}(t')) \quad (7)$$

Therefore equation (8) is yielded by equations (6) and (7).

$$F_d(k+1) = F_d(conc_{F_d}(k+1)) \quad (8)$$

W.r.t PC, the same discussion can be done. Moreover from **V5**, $\mathcal{I}_t$ does not write any value into F_j such that $F_j \notin \mathcal{F}^{dest}(\mathcal{I}_t)$. Therefore equations (9) and (10) hold.

$$PC(k+1) = PC(conc_{PC}(k+1)) \quad (9)$$

$$F_j(k+1) = F_j(conc_{F_j}(k+1)) \quad (10)$$

Finally, from C2 and **V4**, $conc_{F_j}(k+1)$ follows or equals to $conc_{F_j}(k)$ for each register F_j. □

[Proof of Property1]

Let F_s denote a register such that $F_s \in \mathcal{F}^{src}(\mathcal{I}_t)$. In order to prove Property 1, we need to consider two cases w.r.t the instructions that precede $\mathcal{I}_t$ and are specified to write values into F_s: (a) when all of such instructions complete to write the values into F_s in the state t', and (b) when some of such instructions are executing the stages $\mathrm{stg}_{c+1}, \ldots, \mathrm{stg}_n$ in the state t'.

For case (a), $F_s(t') = F_s(conc_{F_s}(k))$ holds. Moreover there does not exist any instruction specified to write values into F_s in the stages of $\mathrm{stg}_{c+1}, \ldots, \mathrm{stg}_n$. These are yielded by following three reasons. (a1) All instructions that are speculatively fetched and abandoned can not write values into F_s(from **V1** and C4). (a2) All instructions preceding $\mathcal{I}_t$ can not write values into F_s after $conc_{F_s}(k)$(from the hypothesis). (a3) All instructions that follow $\mathcal{I}_t$ and are specified to write values into F_s can not execute the stage stg_c before $\mathcal{I}_t$ executes the stage stg_c(from **V3**). Therefore Property (1) holds from C4.

For case (b), from C2, C4, **V1**, **V3**, **V4**, the value of $F_s(clk_{c+1,n}^{(F_s)}(t'))$ equals to the value written into F_s by the latest instruction of all instructions that are specified to write values into F_s and that are executing the stages

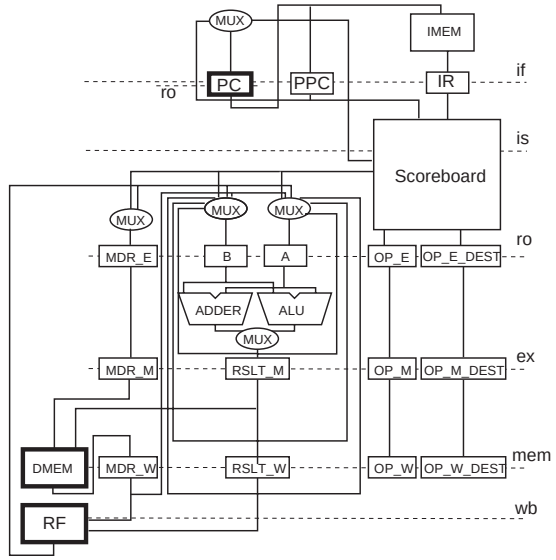

Fig. 1. An overview of our example microprocessor.

of $\mathrm{stg}_{c+1}, \ldots, \mathrm{stg}_n$ in the state t'. Therefore the value of $F_s(clk^{(F_s)}_{c+1,n}(t'))$ equals to the value of $F_s(conc_{F_s}(k))$, since all instructions following $\mathcal{I}'$ can not affect the execution of $\mathcal{I}'$ from the design constraint C. Therefore Property (1) holds from the hypothesis. □

VI. An experimental proof

We designed a out-of-order pipelined microprocessor under the design constraint C(See Fig. 1). Our implementation is based on FDDP, which has been designed as an implementation with in-order instruction execution by NTT and is similar to DLX[1]. Our microprocessors has a program counter(PC), a register file(RF), and a data memory(DMEM) as visible registers. Moreover it has 6-stage pipeline and a buffer, named scoreboard[1]. The scoreboard of our microprocessor has the entries of six instructions currently executed in the microprocessor and records the program order and execution status of these instructions, and so on. The instruction set of it is a RISC-style ISA with four operation class: 3-register ALUs, Load, Store, and Branches.

We proved that our microprocessor satisfies the sufficient condition V. This proof was carried out as follows. First, we introduced lemmas concerned with properties of the scoreboard and so on, and proved them. For example, we introduced the lemmas describing that the scoreboard always records the program order of instructions, and so on. Secondly, we proved that the implementation satisfies the sufficient condition V under those lemmas. For example, to show that our microprocessor satisfies **V3**, we proved following condition under the lemma describing that the scoreboard always records the program order of instructions and so on :"w.r.t each pair $\mathcal{I}_i, \mathcal{I}_j$ of instructions such that the scoreboard has their entries, $\mathcal{I}_j$ does not execute the ro[2] stage, if (1) the program order of $\mathcal{I}_i$, which is recorded by scoreboard, is less than the order of $\mathcal{I}_j$, (2) $\mathcal{I}_i$ has not executed the ro stage, (3) the operation class of $\mathcal{I}_j$ is ALUs or Load, and (4) the destination register of $\mathcal{I}_j$ agrees with either of the source registers of $\mathcal{I}_i$".

[2]The ro stage of our microprocessor is correspond to stg_c.

TABLE I
CPU time used for proof.

Conditions	CPU time(sec)
Lemmas	18,339
V1	4,031
V2	7,383
V3	2,511
V4	33
V5	48

(PentiumII 300MHz, 128MB Memory)

To prove them, we used a decision procedure for the prenex normal form Presburger sentences bounded by only universal quantifiers. This procedure is based on the transformation rule called "quantifier elimination" which is used in Cooper's algorithm. For speed-up, we added many devices to the algorithm[6]. Although our decision procedure can decide the elimination-order for deleting variables efficiently depending on the form of a given expression, in a few cases we specified this ordering. Although there existed some sentences whose length was over 8,000, the total CPU time spent in the proof was the practical time of 8 hours(See TABLE I).

VII. Conclusions

In this paper, we specified the class of the implementation by imposing a design constraint C on the implementation and proposed a sufficient condition V of the correctness of pipelined microprocessors with out-of-order instruction execution. Experimental proof showed that each conditions in the sufficient condition was able to be proved, although the correctness of the whole microprocessor was difficult to be proved.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1996.

[2] J. Sawada and W. A. Hunt, Jr., "Trace table based approach for pipelined microprocessor verification," in *Proc. 9th CAV*, vol. 1254 of *LNCS*, pp. 364–375, Springer-Verlag, June 1997.

[3] R. Hosabettu, M. Srivas and G. Gopalakrishnan, "Decomposing the proof of correctness of pipelined microprocessors," in *Proc. 10th CAV*, vol. 1427 of *LNCS*, pp. 122–134, Springer-Verlag, June 1998.

[4] J. U. Skakkebæk, R. B. Jones and D. L. Dill, "Formal verification of out-of-order execution using incremental flushing," in *Proc. 10th CAV*, vol. 1427 of *LNCS*, pp. 98–109, Springer-Verlag, June 1998.

[5] J. R. Burch, "Techniques for verifying superscalar microprocessors," in *Proc. 33rd Design Automation Conference*, (Las Vegas), pp. 552–557, June 1996.

[6] S. Morioka, N. Shibata, T. Higashino and K. Taniguchi, "Techniques to reduce computation time in decision procedure for prenex normal form Presburger sentences bounded only by existential quantifiers," in *Trans. IPSJ*, Vol. 38, No. 12, Dec. 1997(in Japanese).