Results of the Verification of a Complex Pipelined Machine Model

Jun Sawada[1] and Warren A. Hunt, Jr.[2]

[1] Department of Computer Sciences, University of Texas at Austin
[2] IBM Austin Research Laboratory

Abstract. Using a theorem prover, we have verified a microprocessor design, FM9801. We define our correctness criterion for processors with speculative execution and interrupts. Our verification approach defines an invariant on an intermediate abstraction that records the history of instructions. We verified the invariant first, and then proved the correctness criterion. We found several bugs during the verification process.

1 FM9801 and Correctness Criterion

We argue that even complex microprocessor design can be formally verified. As an evidence of our claim, we have mechanically verified our FM9801 microprocessor design. It has various features such as out-of-order issue and completion of instructions with Tomasulo's algorithm, speculative execution with branch prediction, precise handling of internal exceptions and external interrupts, and supervisor/user modes.

The FM9801 is formally specified in the ACL2 logic[KM96] at the *instruction-set architecture* (ISA) level and the *microarchitecture* (MA) level. These formal definitions are publicly available along with the FM9801 verification scripts[Saw]. The ISA sequentially executes instructions. Its behavior is specified with function $\texttt{ISA-step}(\mathit{ISA}, \mathit{intr})$, which returns the ISA state after executing one instruction from state *ISA*, with interrupt signal *intr*. The MA is a clock cycle accurate model of the pipelined hardware design. Its behavioral function $\texttt{MA-step}(\mathit{MA}, \mathit{sigs})$ returns the MA state after one clock cycle of execution with external signals *sigs*. We define $\texttt{ISA-stepn}(\mathit{ISA}, \mathit{intr\text{-}list}, m)$ as the recursive function that repeatedly applies the next state function $\texttt{ISA-step}$ to state *ISA* m times, where *intr-list* is a list of interrupt signals for each execution step. Similarly, we define $\texttt{MA-stepn}(\mathit{MA}, \mathit{sig\text{-}list}, n)$ as n applications of $\texttt{MA-step}$ with a list of signals *sig-list*. Projection function $\texttt{proj}(\mathit{MA})$ returns the ISA state consisting of the program counter, the register file, and the memory in *MA*.

Our correctness criterion is whether our machine designs satisfy the commutative diagram shown in Fig. 1. For an arbitrary initial MA state MA_0, a list of signals *sig-list*, and a natural number n, if the initial state MA_0 and the final state $MA_n = \texttt{MA-step}(MA_0, \mathit{sig\text{-}list}, n)$ are both pipeline flushed states, then

$$\texttt{proj}(\texttt{MA-stepn}(MA_0, \mathit{sig\text{-}list}, n)) = \texttt{ISA-step}(\texttt{proj}(MA_0), \mathit{intr\text{-}list}, m)$$

should hold for an appropriate list of interrupt signals *intr-list* and a natural number m. We additionally assume that the executed program does not modify itself.

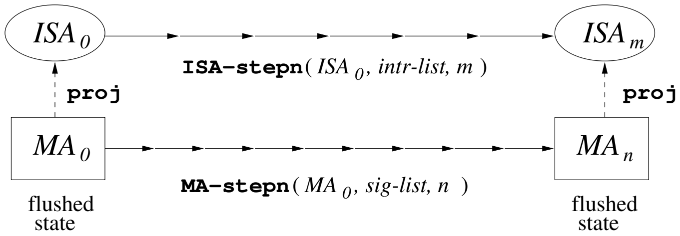

Fig. 1. Correctness Diagram

2 Invariant and Correctness Proof

We use an intermediate model of the MA state that mimics the behavior of speculative execution, exceptions, and external interrupts. This abstraction, which is called a *MAETT*, records executed instructions, each of which is represented with a data-structure holding the values related to the instruction[SH98].

Table 1 gives a list of properties we defined during our verification. Let Π be the set of properties in the table. Additionally, we define predicate `CMI-p` that holds iff the MA has committed any self-modifying code. Then $\bigwedge_{P\in\Pi} P$ is an *invariant under the constraint* $\neg$`CMI-p` [LL90], that is, every property in Π is preserved as long as no self-modifying program is committed. Since $\bigwedge_{P\in\Pi} P$ holds for any flushed pipeline states, $\bigwedge_{P\in\Pi} P$ is true for any reachable state from a flushed state, as long as no self-modifying code is committed.

Given that the final MA state MA_n in Fig. 1 satisfies $\bigwedge_{P\in\Pi} P$, we can show the commutative diagram holds. In fact, all we need to know is that the properties labeled 1 through 6 hold for MA_n. The rest of the properties are necessary for our inductive proofs.

The properties in Table 1 were obtained interactively during the verification process. Initially we started invariant verification by only considering the conjunction of properties labeled 1 through 6. Naturally, our first proof attempt failed. Then, we analyzed the failed proof, and added more properties to the conjunction. Eventually we identified all properties in Π. This was the most time consuming part of the verification.

The proof of the correctness criterion must bridge the complex time abstraction between the ISA level and the MA level. The state of each programmer visible component in the MA is related to different ISA states. These relations are expressed with properties labeled 1 through 4 in Table 1. The proof of the criterion can be found in our report.[SH].

3 Verification Summary

The FM9801 verification was carried out with the ACL2 theorem prover. First, we simulated our FM9801 specification using ACL2's execution capability. This eliminated most of the bugs in our original design before we started the formal verification process.

Table 1. List of the properties used to define our invariant.

#	Property Name	Brief Description
0	**weak-invariants:**	A well-formedness predicate for a MAETT.
1	**pc-match-p:**	Correct state of the program counter.
2	**sregs-match-p:**	Correct state of the special register file.
3	**regs-match-p:**	Correct state of the general register file.
4	**mem-match-p:**	Correct state of the memory.
5	**no-speculative-commit-p:**	No speculatively executed instruction commits.
6	**MT-inst-invariants:**	Valid intermediate data values in the pipeline.
7	**correct-speculation-p:**	Instructions following a mis-predicted branch are speculatively executed.
8	**correct-exintr-p:**	Externally interrupted instructions retires immediately.
9	**in-order-dispatch-commit-p:**	Instructions dispatch and commit in program order.
10	**in-order-DQ-p:**	The dispatch queue is a FIFO queue.
11	**in-order-ROB-p:**	The re-order buffer is a FIFO queue.
12	**no-stage-conflict:**	No structural conflict at pipeline stages.
13	**no-robe-conflict:**	No structural conflict in the re-order buffer.
14	**in-order-LSU-inst-p:**	Certain orders are preserved for instructions in the load-store unit.
15	**consistent-RS-p:**	Reservation stations keep track of instruction dependencies.
16	**consistent-reg-tbl-p:**	The register reference table keeps track of the newest instruction that updates each general register.
17	**consistent-sreg-tbl-p:**	The register reference table keeps track of the newest instruction that updates each special register.
18	**consistent-MA-p:**	The conjunction of miscellaneous conditions.
19	**misc-invariants:**	The conjunction of miscellaneous conditions.

The size of the ACL2 verification scripts and the time to certify the proofs for each stage of the verification are given in Table 2. The whole verification project took about 15 months. The verification of invariant $\bigwedge_{P \in \Pi} P$ occupied the largest portion in the ACL2 proof scripts and in our verification effort. This is not surprising because the proof of the invariant is the core of our verification process. We found several bugs that were not detected in the simulation, and all these bugs were detected during the verification of the invariant.

Table 2. ACL2 script size and CPU time with Pentium Pro 200MHz.

Type of ACL2 Script	ACL2 Script Size	CPU Time to Certify
Definitions of ISA and MA	140 KBytes	14 minutes
MAETT modeling	55 KBytes	6 minutes
Definitions of Our Invariant	89 KBytes	7 minutes
Proof of Shared Lemmas	481 KBytes	58 minutes
Proof of Our Invariant	1034 KBytes	211minutes
Proof of Criterion	37 KBytes	11 minutes

We found 14 design faults in our machine design during the verification process. For instance, one bug was found in the control logic for the speculative execution and the branch prediction. A prediction is usually made for the branch instruction at the instruction fetch unit (IFU). If the branch instruction stalls in the IFU, then more than one branch predictions are made for a single branch instruction. In the original design, if the branch prediction outcomes differed, the machine did not correctly execute instructions from the branching point.

Table 3. Sizes of ACL2 proof scripts for different machines.

Verified Machine	Machine Spec	Total Verification
Small Example Machine	13 KBytes	169 KBytes
Pipelined Design presented in CAV '97	78 KBytes	757 KBytes
FM9801	140 KBytes	1909 KBytes

Although this verification was labor intensive, our technique seems to scale well with the size of machine. In Table 3, we compare the size of our machine specification and verification scripts with two other proof efforts, each with different machine sizes, where we employed a similar approach. The ratio of the size of the verified machine design and its verification script does not change much. We also note that the CPU time in Table 2 is relatively small. This is because we decompose a complex verification problem into small lemmas to avoid case explosions. Typically, the ACL2 theorem prover proves single lemmas in less than a minutes during our verification.

We have demonstrated that the pipelined machine with complex control features can be mechanically verified. Although the verification cost was high, we do not see any major difficulty in scaling the verification process for a more complex design. Until now, we have only used an theorem prover, but the combination of algorithmic approach could improve the verification efficiency. Improving invariant verification processes will make our technique more practical.

References

[KM96] Matt Kaufmann and J Strother Moore. ACL2: An industrial strength version of nqthm. In *Eleventh Annual Conference on Computer Assurance (COMPASS-96)*, pages 23–34. IEEE computer Society Press, June 1996.

[LL90] Leslie Lamport and Nancy Lynch. Distributed computing models and methods. In *Handbook of Theoretical Computer Science*, volume B, pages 1159–1199. The MIT Press, Cambridge, Ma., 1990.

[Saw] Jun Sawada. Verification scripts for FM9801 pipelined microprocessor design. Web page `http://www.cs.utexas.edu/users/sawada/FM9801/`.

[SH] Jun Sawada and Warren A. Hunt, Jr. Verification of FM9801: Out-of-order processor with speculative execution and exceptions that may execute self-modifying code. Unpublished Report. Personal contact: sawada@cs.utexas.edu.

[SH98] Jun Sawada and Warren A. Hunt, Jr. Processor verification with precise exceptions and speculative execution. In Alan J. Hu and Moshe Y. Vardi, editors, *computer Aided Verification (CAV '98)*, volume 1427 of *LNCS*, pages 135–146. Springer Verlag, 1998.