Verifying Tomasulo's Algorithm by Refinement*

T. Arons and A. Pnueli
Department of Applied Mathematics
Weizmann Institute of Science
Rehovot, 76100, Israel
{tamarah, amir}@wisdom.weizmann.ac.il

Abstract

In this paper Tomasulo's algorithm for out-of-order execution is shown to be a refinement of the sequential instruction execution algorithm. Correctness of Tomasulo's algorithm is established by proving that the register files of Tomasulo's algorithm and the sequential algorithm agree once all instructions have been completed.

1. Introduction

Modern out-of-order super-scalar microprocessors use dynamic scheduling to increase the number of instructions executed per cycle. These processors maintain a fixed-size window into the instruction stream, analyzing the instructions in the window to determine which can be executed out of order to improve performance. Branch prediction and register renaming are employed in order to keep the window full, while result-buffering techniques maintain the in-order-execution model required by the architecture.

In this paper we propose a proof-method for proving correctness of such processor designs based on *refinement*, and illustrate it by showing correctness of the Tomasulo algorithm [3] for out-of-order execution. Our approach is based closely on [2] and shares the advantages of this approach, in particular, the ability to cope with *generic* designs, establishing correctness for an arbitrary number of functional units performing arbitrary arithmetic operations. Arithmetic operations are modeled as *uninterpreted functions*, allowing us to verify instruction scheduling without performing arithmetic verification. Our approach greatly simplifies the proof by obviating the need for the intermediate abstraction used by Damm and Pnueli. Instead, we introduce a *predicted value* field to allow us to directly compare system TOMASULO to system SEQ. We believe that the use of predicted values, as presented in this paper, provides an effective means of verifying refinement between two systems which run at different rates. Our proof is intuitively simple and has been verified using the PVS [5] theorem prover.

Recent papers [4, 6] propose new techniques for verifying out-of-order processing units. In [4] Tomasulo's algorithm is verified using the SMV verifier, and an impressive level of automation is achieved. The proof, however, is dependent on the configuration and arithmetic operators and any modification requires that the new system be verified afresh.

A more complex model of out-of-order execution is verified in [6] using the Stanford Validity Checker. The implementation machine is shown to refine an intermediate abstraction. Incremental flushing is then used to show that the intermediate abstraction is functionally equivalent to the specification machine. In comparison, our refinement is to the specification level, obviating the need for a second stage.

This paper is structured as follows: The next section presents our definition of refinement. Ultimately, the implementation has to be compatible with the sequential reference model developed in Section 3. Section 4 introduces the Tomasulo algorithm. In section 5 we sketch the proof of correctness.

2. Refinement

An *abstract system* S_A is designed to serve as a *specification* capturing all the acceptable correct computations of the *concrete system* S_C. We take SEQ and TOMASULO as the abstract and concrete system respectively.

Observation functions $\mathcal{O}_A$ and $\mathcal{O}_C$ of S_A and S_C are defined. These functions indicate the parts of the systems which are compared by the refinement relation.

The transition relations ρ_A and ρ_C define how the system moves from one state to another. For example, in one step the sequential algorithm may execute an instruction, updating the program counter and register file. Typically, ρ_A is non-deterministic (e.g. SEQ may either execute an instruction or idle). We define a transition relation ρ_A^* which can use any

*This research was supported in part by a gift from Intel, a grant from the Minerva foundation, and an *Infrastructure* grant from the Israeli Ministry of Science.

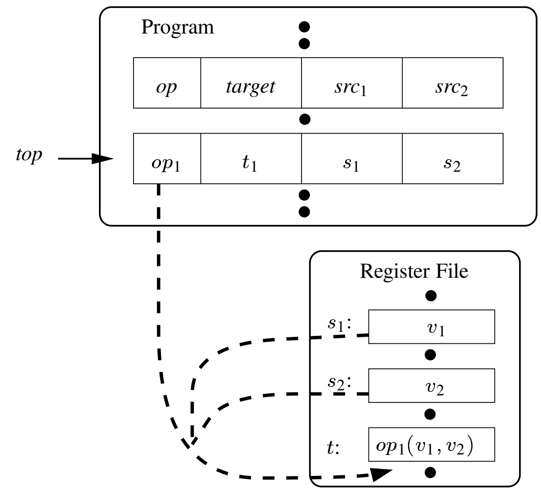

Figure 1. Execution of one instruction in system SEQ

details of the current state in system S_A and S_C, or of the next state in system S_C, to choose one of the possibilities offered by ρ_A. (E.g. SEQ executes an instruction if and only if TOMASULO dispatched an instruction.)

We say that *system* S_A *refines system* S_C *according to the observation pair* $(\mathcal{O}_C, \mathcal{O}_A)$ if

- **R1.** It is always possible to take a ρ_A^*-step.
- **R2.** Every ρ_A^*-step is a legal ρ_A-step.
- **R3.** When S_C advances using ρ_C and S_A using ρ_A^*, $\mathcal{O}_C = \mathcal{O}_A$.

That is, S_A refines S_C if using ρ_A^* a legal (R2) computation of S_A can be generated (R1) such that the observation functions $\mathcal{O}_C$ and $\mathcal{O}_A$ always agree (R3).

A more rigorous definition of refinement can be found in [1].

3. The Reference Model: System SEQ

In this section we present system SEQ which is to serve as a reference model. System SEQ executes in a strictly sequential manner an input program consisting of non-branching register-to-register instructions. It accepts two parameters, N, the number of instructions, and R the maximum register index. System SEQ is presented in Fig. 1.

Instructions are stored in an array *prog* of length N. Each instruction has an *operation*, a *target* and two *source* operands. A program counter, *top*, points to the next instruction in *prog*. A register file *reg* records the current values of each register $0..R$.

At each step system SEQ either delays, in which case no change is made in the system, or executes the instruction pointed to by *top*. The value computed by the instruction is stored in *reg*[*prog*[*top*].*target*] and *top* is incremented by one.

4. The Tomasulo Algorithm

In this section we give a brief overview of the Tomasulo algorithm [7] for data-driven instruction execution. Our definitions are based on the descriptions in [2, 3].

The program in system TOMASULO is identical to that in system SEQ, and like SEQ TOMASULO has a program counter called *top*. In addition, system TOMASULO contains a register file RF, a finite number U of functional units and an array rs of reservation stations in which each functional unit has Z slots. Each functional unit owns one *result* register in which results are stored until they can be put on the bus. The parameters of system TOMASULO are thus N, R, U and Z. The data structures are presented in Fig. 2.

Instructions are issued from the instruction stream to a reservation station slot. The reservation station entry records the status of the instruction, the operation of the instruction, and information about its operands. The operand information is taken from the register file which records, for every register, a value for the register, a busy-flag indicating whether an instruction targeting the register is pending, and a tag pointing to the reservation station entry which will produce the result. Either the value or the tag (*prod*) is valid, depending on the busy flag. All this information about the operand is copied to the reservation station. The reservation station is marked as occupied and a busy-bit is set, indicating that the instruction has not been completed.

The reservation stations continuously snoop the bus, waiting for the values of any pending results to be written back. Once both operands of an instruction are available the instruction may be executed. On each cycle each functional unit can execute at most one such instruction from its block of reservation stations.

The result of the instruction which has been executed is passed to the *result* register of the functional unit. The active bit of the result register is set exactly when the result register contains a value and is waiting for control of the bus.

When a result is put on the bus it is read by both the register file and the reservation stations. If the tag on the bus matches the producer field of a busy register, the register file saves the value and clears the busy-flag. When a reservation station notices its own address as the tag on the bus, it marks itself as unoccupied, and becomes available to receive a new instruction.

Every value field in the system is paired with an auxiliary *predicted value* (*pv*) field. Auxiliary variables are only updated and copied from one record to another but they never affect the flow of control. Hence, it is easy to prove that deleting these will not affect valuations of other sys-

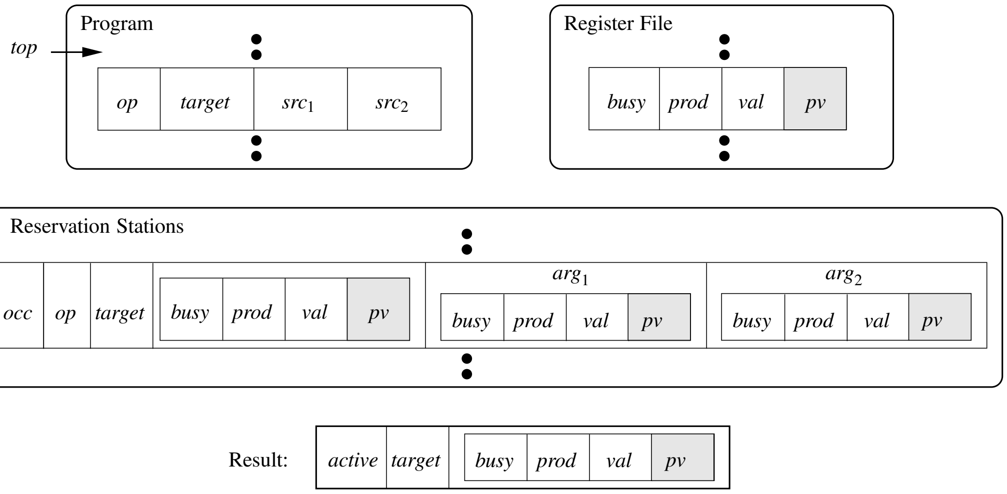

Figure 2. Data structures for TOMASULO

tem variables. In Fig. 2 auxiliary fields are distinguished by shading.

The predicted value field is fundamental to our proof of correctness: while system TOMASULO may complete instructions long after SEQ does, it calculates predicted values at dispatch time, in tandem with the execution of the same instructions by SEQ. Predicted values thus provide a natural means of comparing the two systems.

When instructions are dispatched, the predicted values of their operands are read from the register file and stored in the reservation station. The predicted value of the instruction, calculated by applying the operation to the predicted values of the operands, is stored in both the register file and reservation station. This value is not modified until a new instruction is dispatched to these records. When an instruction completes, its predicted value is copied to the result register.

5. System TOMASULO is correct

In this section we sketch the proof that system TOMASULO is correct, i.e., the final values calculated by TOMASULO equal those calculated by SEQ. We first prove that the predicted values in the register file of system TOMASULO equal the values calculated by SEQ, and then that the calculated and predicted values in system TOMASULO agree when all instructions have been completed. A more detailed explanation of the proof, which has been verified using the PVS theorem prover, can be found in [1].

Since TOMASULO may complete instructions out-of-order a direct comparison of the values in TOMASULO with the values in SEQ would be complicated. Instead, we ensure that the two systems dispatch instructions simultaneously and compare the *predicted* values in TOMASULO with the values computed by SEQ. This comparison is simpler as the predicted value in TOMASULO becomes available in the same step in which the value is calculated in SEQ. Letting top_A and top_C be the program counters *top* of systems SEQ and TOMASULO respectively, simultaneous dispatch of the instructions is equivalent to requiring that top_A equals top_C.

Our observation functions are thus:

$$\begin{array}{lcl} \mathcal{O}_A & : & (top_A, reg) \\ \mathcal{O}_C & : & (top_C, \lambda r : [0..R].RF[r].pv) \end{array}$$

We assume that both computations start at the first instruction with all registers storing value zero and all reservation stations unoccupied.

The transition relation ρ^*_A is defined such that SEQ executes one instruction whenever an instruction is issued in TOMASULO. It is easy to show that it is always possible to take a ρ^*_A-step and that every ρ^*_A-step is a legal ρ_A-step (premises **R1** and **R2** of our refinement rule).

To prove that premise **R3** holds we need to show that the observation functions of the two systems agree. The program counters agree by definition of ρ^*_A. We further require that the values in the register file of SEQ equal the predicted values in the register file of TOMASULO. The predicted value of the new instruction is obtained by applying the instruction operation to the predicted values of the operands. The value in system SEQ is obtained by applying the same operation to the values in reg. Under the assumption that the observation functions are equal in the current state, the predicted values of the operands in system TOMASULO equal the values calculated for them in system SEQ, and so $\mathcal{O}_C = \mathcal{O}_A$ in the new state. This provides us with an inductive proof that the observation functions of the two system are equal.

We have thus shown that TOMASULO refines SEQ and that the predicted values in the register file of TOMASULO equal the values computed by SEQ. The proof of correctness can now be completed by proving that the predicted values in registers match the values finally written to them. This proof proceeds in two stages: We first show that all predicted values referring to the same instruction agree, and then that values are predicted correctly.

Uniformity of predicted values. The quadruple $\langle$*busy*, *prod*, *val*, *pv*$\rangle$ appears repeatedly in the system and in Fig. 2 each occurrence is enclosed in a rectangle. We call it a record of type QUAD. A QUAD-record is called *active* if it is a busy register, part of an active result, the main QUAD-field of an occupied reservation station, or a busy operand field of an occupied slot. When two active QUAD-records have the same producer field they are intuitively pointing to the same reservation station entry, either waiting for its result to become available (e.g. busy registers or operand fields in the reservation stations), or recording the source of the value already obtained (active result register). It is clear that all such QUAD-records should record the same predicted values, and this can be proved to be the case.

Correctness of prediction. Consider a reservation station slot in which an instruction is being executed. The predicted value stored in the slot's QUAD-record was obtained by applying the instruction operation to the predicted values of its operands. The values of the operands have been calculated previously, and we assume, for induction, that their values agree with their predicted values. The value for the instruction is calculated by applying the instruction operation to the values of the two operands and will thus agree with the predicted value. The calculated value is propagated to the result register, and from there to any operand fields and registers which are waiting for it. By the uniformity of predicted values, the predicted value in each of these active QUAD-records equals that in the reservation station which calculated the instruction. Thus, wherever the value is written it will equal the predicted value. This proves that every value written to a QUAD-record is correctly predicted by the record's predicted value field. (A QUAD-record may be overwritten without having stored a value. We claim only that values which *are* written to QUAD-records are correctly predicted.)

We compare systems SEQ and TOMASULO after all instructions have completed. It can be shown that at this point all registers do contain a final value. The principle of correct prediction asserts that the value and predicted value fields in all registers agree. By the refinement relation the predicted values in system TOMASULO agree with the values recorded in system SEQ, and thus system TOMASULO is shown to be correct.

6. Conclusion

We have presented a refinement-based proof-method for the verification of modern processor architectures, and have demonstrated its applicability by showing the correctness of a data-path involving multiple functional units, register-renaming, dynamic scheduling, and out-of-order execution.

Our use of predicted values has significantly simplified the refinement and obviated the need for an intermediate level. The strength of our proof is its generality: we prove correctness for arbitrary configurations of unlimited size. The proof is also independent of the operations appearing in the instructions, and checks only the correctness of the out-of-order scheduling, not the calculation of mathematical expressions.

Still, this work needs to be extended in a number of ways:

- We would like to increase the degree of automation in the proofs. We are currently investigating the possibility of making better use of PVS strategies. Compilation of the employed notation for STS could ideally replace the current hand-translation.
- Our model is limited to non-branching programs in which no loads, stores or exceptions occur. Ongoing work considers extending the framework of this paper to incorporate some of these features.

References

[1] T. Arons and A. Pnueli. Verifying Tomasulo's algorithm by refinement. Technical report, Dept. of Comp. Sci., Weizmann Institute, Oct 1998.

[2] W. Damm and A. Pnueli. Verifying out-of-order exectutions. *CHARME'97*:23-47, Chapman & Hall, 1997.

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

[4] K.L. McMillan. Verification of an implementation of Tomasulo's algorithm by compositional model checking. *CAV'98*:110-121, 1998.

[5] S. Owre, J.M. Rushby, N. Shankar, and M.K. Srivas. A tutorial on using PVS for hardware verification. In R. Kumar and T. Kropf, editors, *Proceedings of the Second Conference on Theorem Provers in Circuit Design*, pages 167–188. FZI Publication, Universität Karlsruhe, 1994. Preliminary Version.

[6] J.U. Skakkebaek, R.B. Jones and D.L. Dill. Formal verification of out-of-order execution using incremental flushing. *CAV'98*:98-110, 1998.

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