Modified Architectural Support to Implement Tomasulo's Algorithm on Tournament Branch Predictor

Rubina Khanna, Vinay Chopra, and Sweta Verma

Abstract—**This paper proposes modified architecture of 21264, Out-Of-Order, six-way issue microprocessor. The proposed modified architecture implements Tomasulo's algorithm using tournament branch prediction scheme to improve the performance of processor. Tomasulo's Algorithm controls the operation of the Common Data Bus (CDB) by means of tag mechanism. A tag is a 4-bit number used to identify separately each of eleven sources which can feed the CDB. The proposed modified architecture will evaluate branch outcome by taking both local and global history. The choice of global-versus-local branch prediction is made dynamically on a path-based predictor that decides which predictor to use, based on the past correctness of choice.**

Index Terms—**common data bus (CDB), tomasulo's algorithm, tournament branch predictor.**

I. Introduction

The IBM 360/91 floating point used a sophisticated scheme to allow out of order execution. This scheme invented by Robert Tomasulo's, tracks when operands for instructions are available, to minimize RAW hazards, and introduces Register Renaming, to minimize WAW & WAR hazards. IBM's goal was to achieve high floating point performance from an instruction set. Tomasulo's algorithm is designed to overcome long memory access and long floating point delays. It also supports overlapped execution of multiple iteration of a loop [2].

The 21264 is a superscalar microprocessor that can fetch and execute up to four instructions per cycle. It also features out-of-order execution. With this the instructions execute as soon as possible and in parallel with other nondependent work, which results in faster execution. The processor also employs speculative execution to maximize performance. The 21264 implements a sophisticated tournament branch prediction scheme. The scheme dynamically chooses between two types of branch predictors- one using local history and one using global history-to predict the direction of given branch [9].

A. Local Branch Predictor

The local branch predictor bases predictions on the past behavior of the specific branch instruction being fetched. The local branch predictor holds 10 bits of branch pattern history for up to 1,024 branches. This 10–bit pattern picks from one of 1,024 prediction counters. It maintains a PC-indexed history table of branch patterns which, in turn, index a table of prediction counters, which supply the prediction. The history table records the last 10 taken/not-taken branch decisions for 1K branches (indexed by 10 bits of the program counter). As branch history is accumulated, a given history table entry may, in succession, index a different prediction counter. For example, a branch that is taken on every third iteration will generate, in succession taken/not-taken patterns of 0010010010, 0100100100 and 1001001001 (assume "taken" is denoted as "1" and "not-taken" as "0"). When the branch is issued, resolved and committed, the history table is updated with the true branch direction and the referenced counter is incremented or decremented in the direction which reinforces the prediction [8].

B. Global Branch Predictor

The global branch predictor bases its prediction on the behavior of branches that have been fetched prior to the current branch. The global predictor is a 4,096-entry table of 2-bit saturating counters indexed by the path, or global history of last 12 branches. Consider the following code sequence:

```
loop :
  //modify a and b
  if (a == 100) {…}     //1
  if (b % 10 == 0) {…….} //2
  if (a % b == 0) {…....} //3
```

Prediction based on program flow would conclude that if the first two branches were taken then the third branch should be predicted-taken.

C. Choice predictor

The choice of global-versus-local branch prediction is made dynamically on a path-based predictor that decides which predictor must be used based on the past correctness of choice. The chooser is a table of prediction counters; indexed by path history that dynamically selects either local or global predictions for each branch invocation. It is trained to select the global predictor when global prediction was correct and local prediction was incorrect. The choice predictor or chooser is also a 4,096-entry table of two-bit prediction counters indexed by the path history.

II. Proposed Modified Architecture

In this proposed modified architecture, we are going to

Manuscript received March 10, 2011; revised July 29, 2011.

Rubina Khanna is a post graduate student with DAVIET, Jalandhar, Punjab, India (rukukapoor@gmail.com)

Vinay Chopra is Assistant Professor with DAVIET, Jalandhar, Punjab, India (vinaychoprs22@yahoo.co.in)

Sweta Verma is Associate Professor with GCET, Greater Noida, Uttar Pradesh, India. (sweta_verma@yahoo.com)

apply branch prediction on Tomasulo's algorithm by using Tournament branch prediction scheme, to improve the performance of processor. The modified architecture is shown in Fig.1 below:

A. Description of modified architecture

Tomasulo's Algorithm was designed to control the flow of data between a set of programmable floating-point registers and a group of parallel arithmetic units. Tomasulo's Algorithm attempts to minimize delays between the production of a result by one operation and the start of a subsequent operation which requires that result as an input. The algorithm also deals with the register renaming mechanism by providing additional registers, known as Reservation Stations, at the inputs to the arithmetic units and a system of tags which direct result operands to where they are next needed, rather than necessarily to where they would have gone when the instructions which produced them were issued. [1].

Instructions are prepared from the Instruction Unit pipeline and entered in sequence, at a maximum rate of one per clock cycle, into the Floating-point Operation Stack (FLOS). Instructions are taken from the FLOS in the same sequence, decoded, and routed to the appropriate execution unit. The Instruction Unit maps both storage-to-register and register-to-register instructions into a pseudo register-to-register format, in which the equivalent of the R1 field always refers to one of the four Floating-point Registers (FLR), while R2 can be a Floating-point

Register, a Floating-point Buffer (into which operands are received from store), or a Store Data Buffer (from which operands are written to store). In the first two cases R2 defines the source of an operand; in the last case it defines a sink. The most significant features of this floating-point system are the Common Data Bus (CDB), the Reservation Stations at the inputs to the arithmetic units and the Tag mechanism used by Tomasulo's Algorithm to control the interactions between the units attached to the CDB. The CDB allows data produced as the result of an operation to be forwarded directly to the next execution unit or back to the store without first going through a floating-point register, thus reducing the effective pipeline length for *read after write* dependencies, as found.

Tomasulo's Algorithm controls the operation of the Common Data Bus (CDB) by means of a tag mechanism. A tag is a 4-bit number used to identify separately each of the eleven sources which can feed the CDB. These are the six floating-point buffers, the three Reservation Stations associated with the add unit and the two Reservation Stations associated with the multiply/divider unit. Tag registers are associated with each of the four Floating-Point Registers, with the Source and Sink registers of each of the five Reservation Stations, and with each of the three Store Data Buffers. There is also a *busy bit* associated with each of the Floating-Point Registers. This bit is set whenever the FLOS issues an instruction designating that register as a sink and re-set when a result is returned to the register.

Before the decoder issues an instruction possibilities of branch instruction is tested and if the branch instruction has occurred then the branch predictions are used to resolve the branch occurrence (taken/not-taken). Branch prediction is the most important requirement for maintaining the high performance of modern processors. The highly pipelined nature of most modern processors mean that is control dependencies such as conditional branches, return instructions, etc, has the potential to introduce pipeline stalls. To avoid this, two actions have to be carried out:

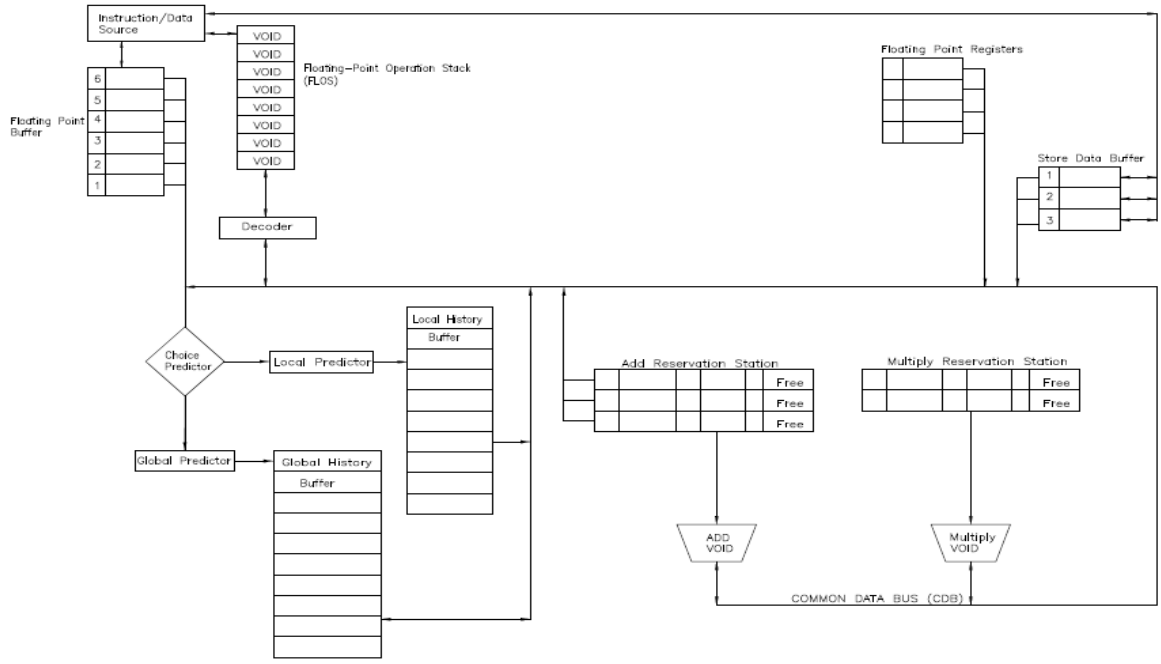

Fig. 1. Proposed Modified Architecture of 21264 Processor

Firstly, prediction on the ***direction*** of the branch must be made, which will allow speculative execution on the predicted path until the branch is resolved.

Secondly, Branch **target address** must be quickly determined, so that the pipeline may be continued to be filled with a minimum number of pipeline bubbles introduced.

The paper proposes modified architecture implementing Tomasulo's algorithm in tournament branch prediction

scheme, which will evaluate the branch outcome by taking both local and global history. The choice of global-versus-local branch prediction is made dynamically on a path-based predictor that decides which predictor must be used based on the past correctness of choice. The tournament predictor consists of 4K 2-bit counters to choose from among a global predictor and a local predictor. The global predictor also has 4K entries and indexed by history of last 12 branches; each entry in the global predictor is a standard 2-bit predictor i.e. the global predictor consist of 12 bit pattern in which ith bit 0 => ith prior branch not taken and ith bit 1 => ith prior branch taken.

The local predictor consist of a 2-level predictor which maintains the local history table of 1024 10-bit entries, in which each 10-bit entry corresponds to most recent 10 branch outcomes for the entry.

So, after receiving instructions from the Floating-Point Operand Stack the decoder firstly use choice predictor for resolving the branch instructions to be taken or not-taken by identifying their local and global history. The choice predictor will choose the type of history used for a particular branch by first checking the threshold value. If the threshold value is greater than or equal to 2, then global predictor bit is set and global history is updated else local predictor is set and the result appears on the CDB, which will broadcast to all destinations. As the local predictor holds its history in local history buffer, the busy bit is set when it overflows and resets to zero when the previous instructions are completed.

Before the Decoder issues an instruction, it checks the busy bit of each of the specified floating-point registers. If the busy bit is zero, the contents of the register are sent to the selected Reservation Station; if the busy bit is set to one, the current value of the corresponding tag register is sent instead. The busy bit of the designated Sink register is then set to 1, and the tag number of the selected Reservation Station is entered into its tag register. Thus the tag register of a busy floating-point register identifies the last unit (in proper program sequence) which will produce a result for it.

Whenever a result appears on the CDB, the tag corresponding to its Reservation Station is broadcast to all destinations. Each active Reservation Station (selected but awaiting a register operand) compares its Sink and Source tags with the CDB tag. If a match occurs, the Reservation Station takes the data from the CDB. In a similar manner, the CDB tag is compared with the contents of the tag registers associated with the Floating-Point Registers and the Store Data Buffers. All busy registers with tags matching that on the CDB are set to the value on the CDB and their busy bits re-set.

Issuing an instruction in this system only requires that a Reservation Station be available for whichever execution unit is required. If a source register is awaiting the result of a previously issued, but as yet uncompleted instruction, or if a floating-point buffer register is awaiting an operand from store, the tag associated with that register is transmitted instead to the Reservation Station, which then waits for that tag to appear at its input. Thus it is the Reservation Stations which do the waiting for operands, rather than the execution circuitry, which is free to be engaged by whichever Reservation Station fills first. Execution of an instruction starts when a Reservation Station has received both operands.

Algorithm for modified architecture:

```
if (threshold value >=  2)
{
selected = Global;
global selected++;
gp = global_addrs;
}
else
{
selected = local;
local_selected++;
lp = local_addrs;
}
```

Where lp – Local predictor

gp- Global predictor

Threshold value has been calculated via the expression:

Threshold = pow (2, SaturatingCounterSize) / 2;

Hence, the threshold value is taken as greater than equal to 2 after calculation.

III. Conclusion

This paper works for a modified architecture to implement Tomasulo's algorithm on tournament branch predictor. Instructions are prepared from the instruction Unit pipeline and entered in sequence, at a maximum rate of one per clock cycle, into the Floating-point Operation Stack (FLOS). Instructions are taken from the FLOS in the same sequence, decoded, and routed to the appropriate execution unit. The Instruction Unit maps both storage-to-register and register-to-register instructions into a pseudo register-to-register format. Before the decoder issues an instruction, possibility of branch instruction occurrence is tested and branch prediction can be done using local and global history as selected by the choice predictor by checking threshold value against the histories, to resolve the branch occurrence to be taken / not-taken.

References

[1] Rubina Khanna, Sweta Verma, Vinay Chopra, Ranjit Biswas, " Implementation of load in Tomasulo's algorithm-out-of-order execution in Superscalar processor" in proceeding of ICDM 2010, 11-12 March 2010,IMT, Ghaziabad.

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

[3] Grunwald, D., Klauser, A., Manne, S. and pleszkun, A. (1998)' Confidence estimation for speculation control', Proceedings of the 25th annual International Symposium on Computer Architecture, Barcelona, Spain, pp.122-131.

[4] T.-Y. Yeh and Y.N. Patt, "A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History", Proc. 20th Annual ACM/IEEE Intl. Symposium on Computer Architecture, 1993.

[5] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. Patt, "Branch Classification: a New Mechanism for Improving Branch Predictor Performance", Proc. 27th Annual Intl. Symp. On Microarchitecture, 1994.

[6] Hennessy J., Patterson D., "Computer Architecture, aQuantitative Approach", Morgan Kaufmann, 2003.

[7] Jurij Silc, Theo Ungerer, Borut Robic,(2007) 'Introduced dynamic branch prediction techniques for Superscalar processors' Int. J. High Performance Systems Architecture, Vol. 1, No. 1, pp.2–13.

[8] M.-C. Chang and Y.-W. Chou, (2002) 'Branch prediction using both global and local branch history information' Proceedings of IEEE on Computer Digit technology, Vol 149, No. 2, March (2002).
[9] R. E. Kessler, E. J. McLellan, 'The Alpha 21264 Microprocessor architecture' in the proceedings of the International conference on computer design, Vol. 19, March 1999.

Mrs. Rubina Khanna is a Post Graduate student with Computer Science & Engineering Department, DAVIET, Jalandhar, Punjab, India. She has completed her undergraduate course in Information Technology from Lovely Institute of Engineering and technology, Jalandhar in year 2006.Presently she is doing her post graduate course in Computer Science from DAVIET, Jalandhar, Punjab, India. Her areas of interest are Computer Architecture.
She has 1 research papers in International Journal, 2 in international paper and 3 in national conference.

Vinay Chopra is working as Assistant Professor with Computer Science & Engineering Department, DAVIET, Jalandhar, Punjab, India. His areas of Interest are software Engineering, Automata & computer Graphics. He has 12 research publications in national, international journals and conferences.

Mrs. Sweta Verma is Associate Professor with GCET, Greater Noida, Uttar Pradesh, India. She has completed her undergraduate course in CSE and post graduate course in IT. Presently pursuing her PhD in CS&E/IT. Her areas of interest are computer architecture. She is the author of five books and has 10 publications in International journal and international conferences each.