International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Modified Architecture of Tomasulo's algorithm using Adaptive branch predictor

Sanjay Kumar Yadav[1], Shivam Vishwakarma[2], Vishvnath Pratap Singh[3], Sweta Verma[4]

[1,2,3]B.Tech (IT Final Year) Galgotia College of Engineering & Technology
[4]Associate Prof (IT) Galgotia College of Engineering & Technology

ABSTRACT
Tomasulo's algorithm minimizes RAW, WAW and WAR hazards. It attempts to minimize delay between the prediction of a result by one operation and the start of subsequent operation. This paper proposes modified architecture to implement the Tomasulo's Algorithm using Adaptive branch prediction scheme, which evaluate the branch outcome by taking both local and global history dynamically. The choice of local vs. global branch prediction is made dynamically on a path based predictor that decides which branch must be taken on past correctness of choice.

Keywords: adaptive branch predictor, common data bus (CDB), Tomasulo's algorithm.

INRODUCTION

Tomasulo's algorithm uses a sophisticated scheme for out of order execution when operands for instructions are available. Tomasulo's algorithm minimizes RAW hazards and introduces register renaming to minimize WAW & WAR hazards. The goal was to achieve high floating point performance from an instruction set. Tomasulo's algorithm was designed to overcome the long memory access and floating delay. This algorithm also supports overlapped execution of multiple iteration of a loop [1].
Branch prediction schemes can be classified into static schemes and dynamic schemes depending on the information used to make predictions. Static branch prediction schemes can be as simple as predicting that all branches are not taken or predicting that all branches are taken [2]. Dynamic branch prediction takes advantage of the knowledge of branches run-time behavior to make predictions. Lee and Smith proposed a structure they called a Branch Target Buffer [3] which uses 2-bit saturating up-down counters to collect history information which is then used to make predictions. The execution history dynamically changes the state of the branch's entry in the buffer. In their scheme, branch prediction is based on the state of the entry. The Branch Target Buffer design can also be simplified to record only the result of the last execution of the branch.

Adaptive Predictor

The Two-Level Adaptive scheme, which alters the branch prediction algorithm on the basis of information collected at run-time. Several configurations of the Two-Level Adaptive Branch predictor are introduced, simulated, and compared to simulations to other static and dynamic branch schemes. The Two-Level Adaptive Training Branch Prediction scheme has the following characteristics:

•Branch prediction is based on the history of branches executed during the current execution of the program.

• Execution history pattern information is collected on the fly of the program execution by updating the pattern history information in the branch history pattern table of the predictor. Hence, no run of the program are necessary.

The Two-Level Adaptive Training scheme has two major data structures, the branch history register (HR) and the branch history pattern table (PT).In two level adaptive prediction, instead of accumulating statistics by profiling the programs, the execution history information on which branch predictions are based is collected by updating the contents of the history registers and the pattern history bits in the entries of the pattern table depending on the outcomes of the branches. The history register is a shift register which shifts in bits representing the branch results of the most recent history information. All the history registers are contained in a history register table (HRT).

The structure of Two-Level Adaptive Branch Prediction is shown in Figure 1.1. The prediction of a branch is based on the history pattern of the last k outcomes of executing the branch; therefore, k bits are needed in the history register for each branch to keep track of the history. If the branch was taken, then a “1" is recorded; if not, a "0" is recorded. Since there are k bits in the history register, at most 2k different patterns appear in the history register. In order to keep track of the history of the patterns, there are 2k entries in the pattern table; each entry is indexed by one distinct history pattern [2].

When a conditional branch B is being predicted, the contents of its history register, HR_i, whose content is denoted as R_i, R_{c-k}, R_i, R_{c-k+1} for the last k outcomes of executing the branch, is used to address the pattern table. The pattern history bits Sc in the addressed entry PTR _k+l ,r>>c-1 in the pattern table are then used for predicting the branch.

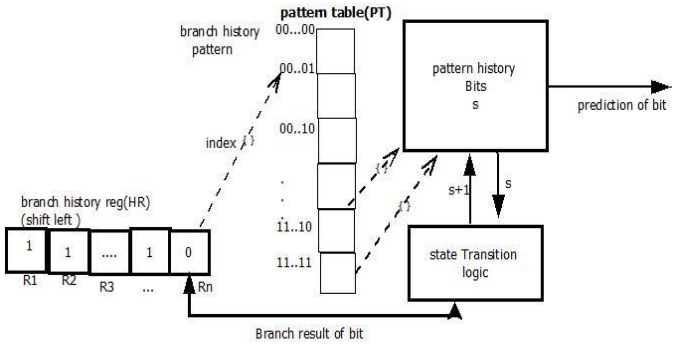

Fig1.1 the structure of two level adaptive branch scheme

Since the pattern history bits change in Two-Level Adaptive Training, the predictor can adjust to the current branch execution behavior of the program to make proper predictions. With the updates, Two-Level Adaptive Training can still be highly accurate over many different programs and data sets. Static Training, on the contrary, may not predict well if changing data sets results in different execution behavior.

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 [2].

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.

2.DESCRIPTION OFPROPOSED ARCHITECTURE

In this proposed modified architecture two level branch predictions on Tomasulo's algorithm by using Adaptive Branch prediction scheme, to improve the performance of processor. The modified architecture is shown in Fig.1.2 .

Modified Architecture Description

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 [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.

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 [8] 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 Buffer storage 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 [1].

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 sources which can feed the common data bus CDB.

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.

Conditional branches have to wait for condition codes in order to decide the branch targets. Subroutine return branches can be predicted by using a return address stack. A return address is pushed onto the stack when a subroutine is called and is popped as the prediction for the branch target address when a return instruction is detected. The return address prediction may miss when the return address stack overflows. For instruction sets without special instructions for returns from subroutines, the double stacks scheme pro in [3] is able to perform the return address prediction.

Before the decoder issues 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:

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.

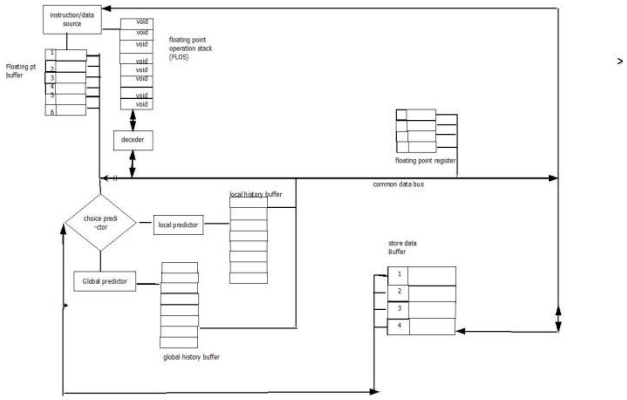

Fig1.2 proposed architecture for 21264 processor

The paper proposes modified architecture implementing Tomasulo's algorithm in adaptive 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 adaptive branch 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 consists of 12 bit pattern in which i^{th} bit 0 => i^{th} prior branch not taken and i^{th} bit 1 => i^{th} 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 at the run time of the execution.

Firstly the instruction are come in buffer data storage and it is connected to common data bus (CDB),on behalf of the execution of the previous branch result from the pattern history table updated from the local/global history table in the form of taken/not-taken by the caching concept.

Branching mechanism is very similar to the pointer concept, which can modifiable at the run time. In this branching the threshold value is identified on behalf of the previously generated result in the form of pattern like "000" or "001".this will defined the next instruction are taken or not .let the sequencing of bit is stored in T.T is a variable which stores the pattern bit sequence, generated by the previous output, as per the local and global history.

Issuing an instruction in this system only requires that a Reservation Station be available for whichever execution unit is required. Store data buffer act here as a reservation station for storing the previous result. 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 or "pattern of bit" associated with that register is transmitted instead to the data buffer, which then waits for that tag to appear at its input. Thus it is the data buffer storage 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 result of previous data sets are stored in data buffer storage and received both operands.

let instr_1 is the start instruction value and

```
instr_n is the last instruction value
for (int i=instr_1;i<instr_ n;i++)
{
If (threshold value==T)
{
Selected =local;      //previous sequence
                              selected
T=local;
local selected++; //data buffer incremented
                        for storing next sequence
lp=local_addrs;
}
Else
{
Selected=Global; //sequence from history
                        result
T =Global;
Global selected++;
gp=Global_addrs;
}
}
```

Where lp - Local predictor gp- Global predictor
Threshold value has been calculated via the sequencing of previous result in Store Data Buffer.

3. CONCLUSION

Dynamic branch prediction take advantages of knowledge of branches and their run time behavior to make prediction. The execution History dynamically changes the state branch's entry in the buffer storage. The adaptive branch prediction scheme alters the branch prediction algorithm on the information collected at the run time. It achieves ninety seven percent accuracy.

REFERENCES:

1. T. Arons, A. Pnueli, "*Verifying Tomasulo's Algorithm by Refinement*". Twelfth International Conference On VLSI Design, 1999. Page(s): 306 – 309
2. Tse-Yu Yeh,Patt, Y.N.. *"Alternative Implementations of Two-Level Adaptive Branch Prediction"*. In The 19th Annual International Symposium on Computer Architecture, 1992.Page(s): 124 - 134
3. J.Lee and A. J. Smith, "Branch Prediction Strategies and Branch Target Bufer Design", IEEE Computer, (January 1984), pp.6-22.
4. A.R. Robertson, R.N *Ibbett "HASE: A Flexible High Performance Architecture Simulator"*. In Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, 1994.Volume: 1, Page(s): 261 – 270
5. T. Juan, S. Sanjeevan, J.J Navarro *"Dynamic History-Length Fitting: A third level of adaptivity for branch prediction,"* In The 25th Annual International Symposium on Computer Architecture, 1998.Page(s): 155 - 166

6. R. Khanna, V. Chopra, S. Verma*,"implementation of branch delay in superscalar processors by reducing branch penalties"2010 ieee 2nd international advance computing conference.*
7. C. Egan, G.B. Steven, Shim Won,L. Vintan, *"Applying Caching to Two-level Adaptive Branch Prediction"*. Euromicro Symposium on Digital Systems, Design, 2001.Page(s): 186 – 193
8. R. Khanna, V. Chopra, S. Verma, *"Modified Architectural Support to implement Tomasulo's Algorithm on Tournament Branch Predictor"* in International Journal of Computer Theory and Engineering, Vol 3 No. 4, August 2011.