Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Implementation of Backward Taken and Forward Not Taken Prediction Techniques in SimpleScalar

Prof. Bhargav C Goradiya[1] , Trusit Shah[2]
Electronics & Telecommunications Department,
BVM Engg College,
Gujarat, India

***Abstract:** In today's world Computer Architecture Design has become rapid developing field. Researchers developing new Instruction Set Architectures, modify current Instruction Set Architectures according to their needs. They are also adding features in processor organization like adding additional floating point unit, increasing the size of the cache memory, changing the pipeline structure, designing new branch prediction logics etc. Now after designing new architecture tests are carried out to check the Performance Parameters of the architecture or processor. Generally HDL is used to test the processor design which requires the lots of time and resources to design whole processor and testbenches. SimpleScalar is the simulator which simulates the processor and gives performance parameters. SimpleScalar consists of different tools to simulate the processor with different perspectives. This tool set consists of basic compiler, cross compiler, assembler, linker, simulator, benchmarks (test bench programs) and visual tools. This tool also enable user to simulate C codes which gives more flexibility than the conventional HDL design flow. SimpleScalar supports PISA, APLHA, x86, SPARC and ARM Instruction Set Architectures. As it is open source you can design your own architecture and simulate it. New Branch Prediction Techniques (Backward Taken and Forward Not Taken) are developed in SimpleScalar and verified.*

***Keywords:** Branch prediction, Simple Scalar tool set, Sim-bpred, BTFNT, Pipeline architecture*

I. INTRODUCTION

The SimpleScalar tool set is a system software infrastructure used to build modelling applications for program performance analysis, detailed micro architectural modelling, and hardware-software co-verification. Using the SimpleScalar tools, users can build modelling applications that simulate real programs running on a range of modern processors and systems. The tool set includes sample simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. In addition to simulators, the SimpleScalar tool set includes performance visualization tools, statistical analysis resources, and debug and verification infrastructure. In simple words SimpleScalar is a toolset which compares two processor and give us their performance parameters without designing processors from the scratch. In this paper in first phase SimpleScalar Tool Set Described then study of different Architectures of SimpleScalar is explained then simulation of functional parameters is explained. After that some basic experiments and tests are covered and finally two branch predictions techniques which is designed as a part of this paper is explained with the proper simulation results.

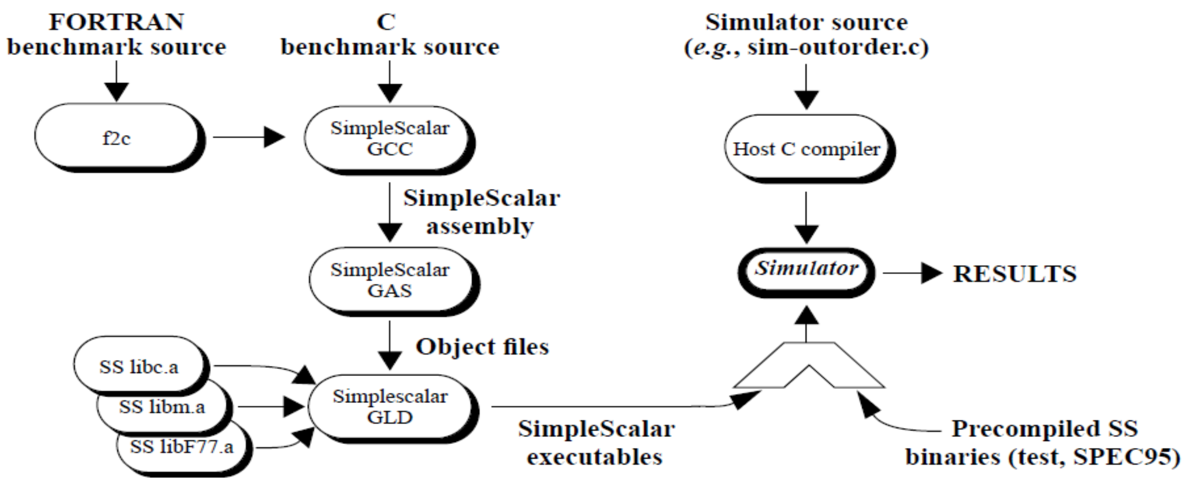


Figure 1 : SimpleScalar Toolset

II. SIMPLESCALAR TOOLSET

SimpleScalar tool set consists of basic compiler, cross compiler, assembler, linker, simulator, benchmarks (test bench programs) and visual tools. Detail description is explained in Figure-1.We can simulate both C and FORTRAN source code in this tool set. For FORTRAN source code it is converted into C through f2c convertor. After that it assembled and compiled which gives you the executable. Now using different SimpleScalar Tool Set we can simulate and get performance parameters of the processor. SimpleScalar consists of following Programs to simulate: sim-fast, sim-safe, sim-cache, sim-cheetah, sim-bpred, sim-profile, sim-outorder. sim-safe: Same as sim-fast but it checks for memory exceptions. Slightly slower instruction interpreter, as it checks for memory alignment and memory access permission on all memory operations. This simulator can be used if the simulated program causes sim-fast to crash without explanation. sim-fast: Fast instruction interpreter, optimized for speed. This simulator does not account for the behaviour of pipelines, caches, or any other part of the micro architecture. It performs only functional simulation using in-order execution of the instructions (i.e. they are executed in the order they appear in the program). sim-cache: It is used to simulate the cache stat of the processor.Cache parameters like cache size , block size , no of ways of cache, no of sets in cache given as input and hit-miss ratio we get as performance parameter. Separate instruction, data and TLB caches can be configured. sim-chetaah: Same as sim-cache but it can configured for multiple caches structure. sim-bpred: It is branch prediction logic for the simulation. It supports taken, not taken, bimod branch prediction logics. sim-profile: Instruction interpreter and profiler. This simulator keeps track of and reports dynamic instruction counts, instruction class counts, usage of address modes, and profiles of the text and data segments. Different types of profiling like instruction profiling, branch profiling, address mode profiling, segment profiling is supported in SimpleScalar. sim-Outorder: Detailed micro architectural simulator. This tool models in detail and out-of-order microprocessor with all of the bells and whistles, including branch prediction, caches, and external memory. This simulator is highly parameterized and can emulate machines of varying numbers of execution units. Figure 2 shows the comparison of different simulator supported by simplescalar toolset.

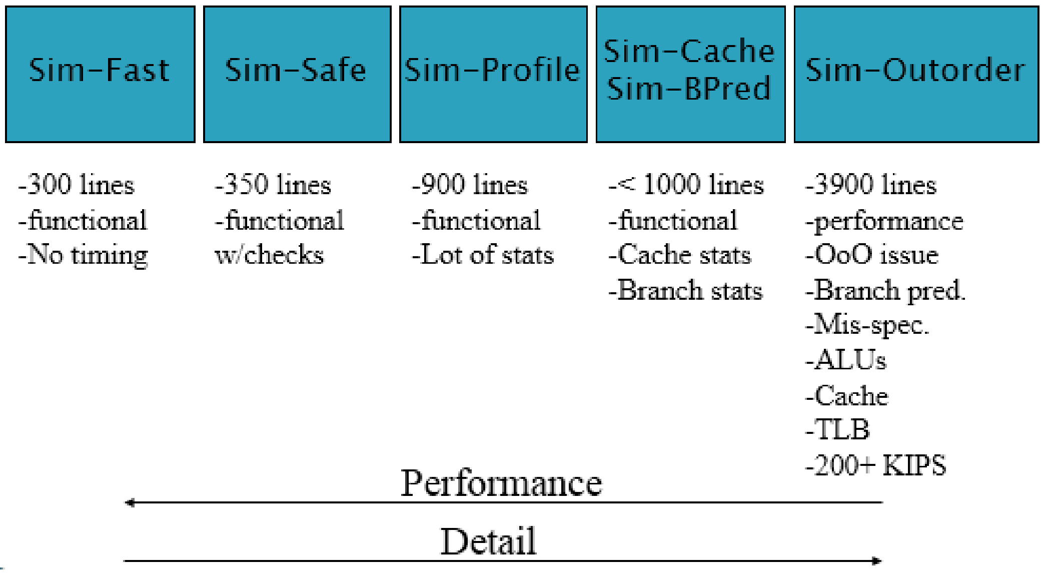


Figure 2: Comparison of different simulator

III. Branch Prediction Technique In Simplescalar

Branch Prediction is essential part of the pipelined based architecture. In pipeline based architecture processor is executing an instruction and at a same time it is prefetching next instructions. Now when a branch comes in the execution stage if it is taken the whole instructions which are pre-fetch before must be removed from queue and start a new queue which disturbs the flow of the pipeline. Now if we able to predict the branch immediately after the fetch we can save the time and increase the performance of the processor. SimpleScalar supports various branch prediction techniques like taken, not taken, bimod, 2level etc. Here we have implemented this technique in sim-bpred.c, bpred.c and bpred.h source codes. As part of the study two more techniques Backward Taken and Forward Not Taken technique is implemented and verified in the simplescalar. As a result of Branch Prediction Technique simulator gives following performance parameters. Number of Branches, Number of Branches whose direction is predicted correctly, Number of Branches whose direction and address both predicted correctly and Prediction parameter for the register relative jump

A. *NotTaken Branch Prediction:*

In this technique we always assume that the branch which is occurred during program is not taken. In the sense we fetch the immediate next instruction of the running program. Now if we find that at the execution state of the branch operation that the branch is taken then we flush the all the prefetch data and fill it with the next jump address which is pointed by the branch. The main drawback of this technique is that the all of the assembly level programs which we are designing are jump if not zero kind of programs where most of the time branch is taken instead of not taken. Due to this problem Not Taken Branch Prediction will not give the desired branch prediction for the program.

B. *Taken Branch Prediction:*
This technique assumes that the branch which is occurring always going to be taken. For taking up the branch we must require an addition hardware which will fetch the target address of branch and fetch the new instruction from that points only. This gives us the advantage for the assembly level programs and gives much better result than the Not Taken Prediction Technique.

C. *Backward Taken and Forward Not Taken Branch Prediction:*
This technique is the combination of the above two techniques. Now according to the survey most of the loop are backward jump loops and which will be taken most of the time. Same way Forward braches are normally not taken. This will enhance the performance of the taken prediction to certain extend. But for the programs which has more if else ladders will not give proper prediction for this technique. This BTFNT technique is not implemented in the SimpleScalar. By changing the source code of the SimpleScalar We have implemented this technique in Simplescalar.

D. Dynamic Branch Prediction:
The techniques which we have seen up till now is static techniques in which predict the branch compile time not run time. Now some new techniques are developed based on history table which stores the previous branch status and used it for branch prediction. 2level, bimod and gskew are the example of this type of technique.

E. 2-bit Branch Prediction:
In this technique k bit counter is runs in the table to maintain the branch history and prediction logic of the branch. From the value of that counter we can predict that whether the branch is taken branch or not taken branch. We have assumed it as two bit counter. If branch is taken it increases its value else decrease the value. At prediction time if the value of the prediction table is more than 2 than we predict the branch as taken branch else take it as not taken branch.

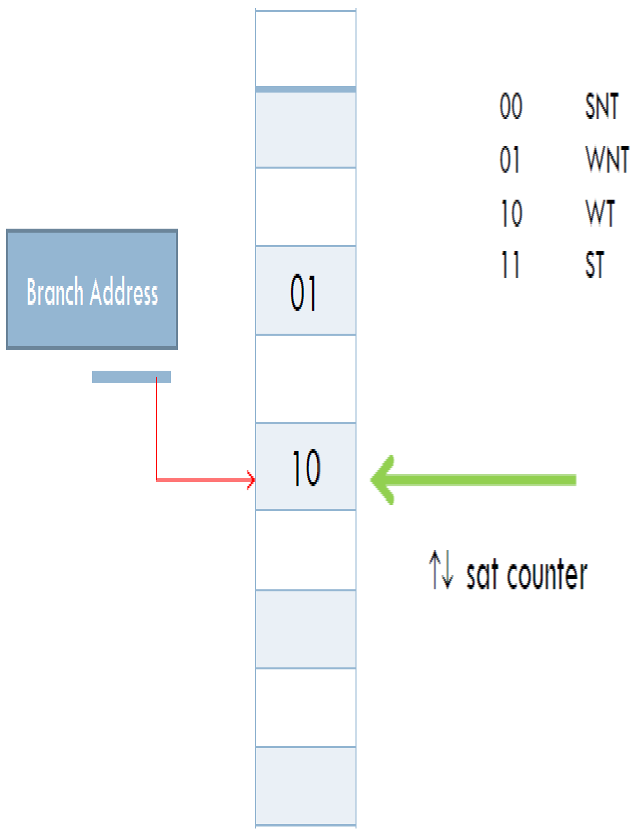


Figure 3: 2-bit Branch Prediction

2-level Branch Prediction:
In this technique two levels of table are used to predict the branch. Sometimes instead of first table single register is used which is known as the Global History Register and the table is known as Branch History Table. An example diagram of the 2level branch prediction is as follow. First Table selects the PHT (Pattern History Table) using PC and previous

branch predictions. After branch BHT is rotated and according to taken and not taken of the branch it is updated. In second table we use same fundamental we are using in the 2 bit prediction.

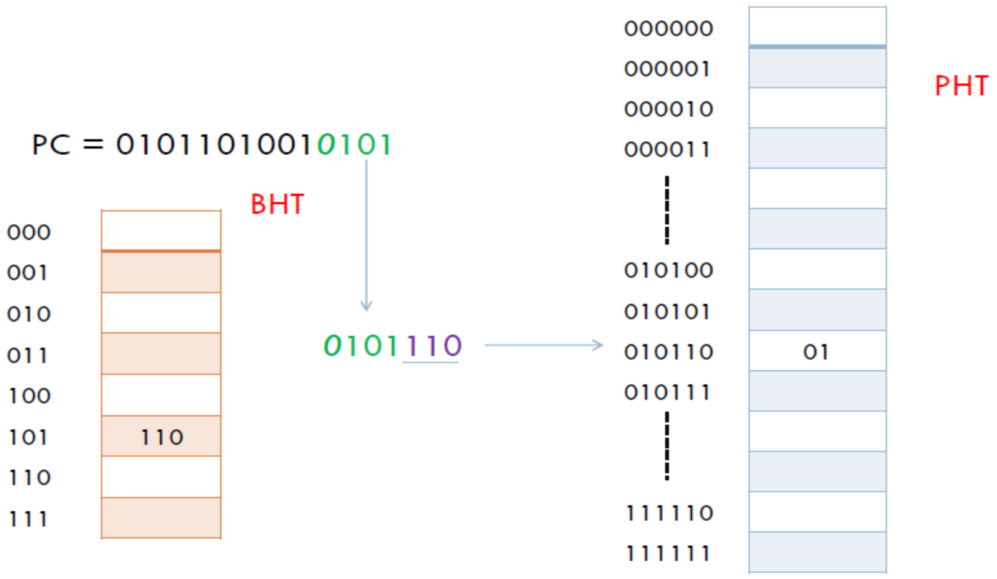
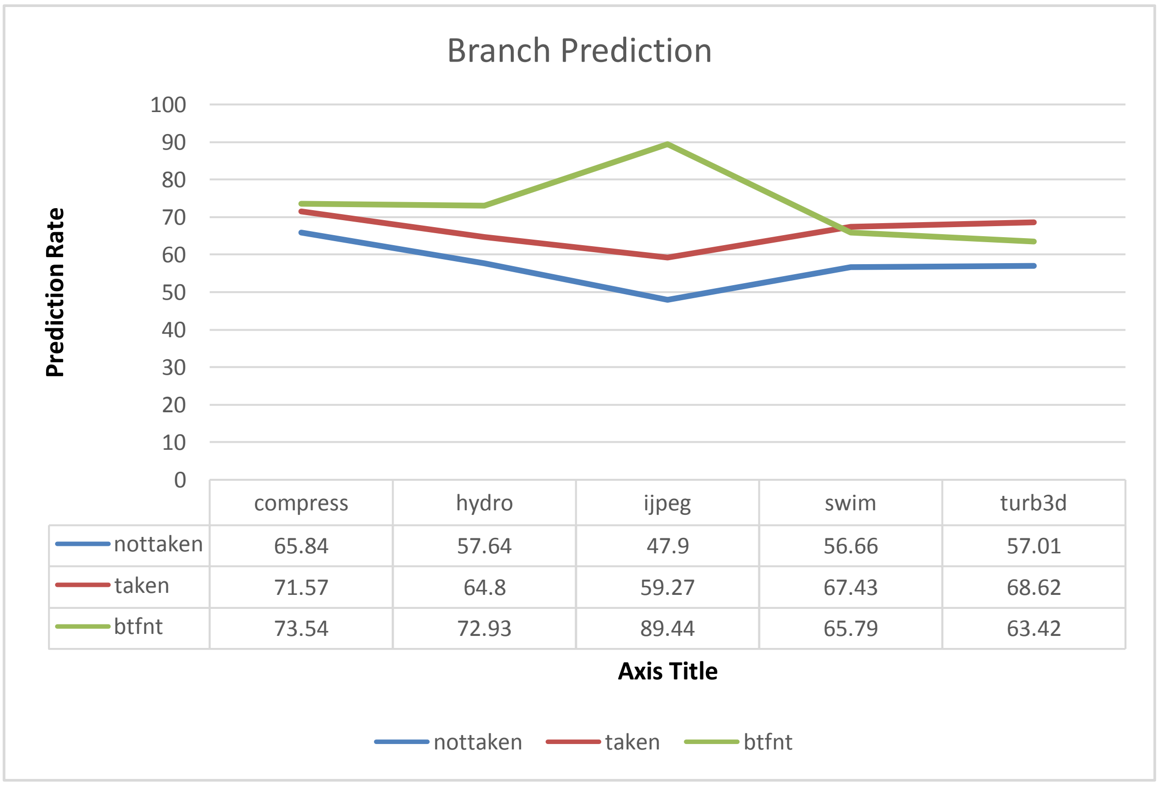


Figure 4: 2 level branch Prediction

IV RESULTS



	compress	hydro	ijpeg	swim	turb3d
nottaken	65.84	57.64	47.9	56.66	57.01
taken	71.57	64.8	59.27	67.43	68.62
btfnt	73.54	72.93	89.44	65.79	63.42

Figure 5 Result Chart

A BTFNT technique is verified on different binaries of the spec95 benchmark programs. Result graph is shown above in figure 5. Here comparison between BTFNT, taken and not taken is shown. From the result graph it is clear that the result of the branch prediction is increased in the BTFNT technique than the not taken and taken techniques.

V CONCLUSION

In this paper we have overview of the simplescalar branch prediction technique along with that new implemented technique is also described with the results. These techniques can be easily implemented on any architecture which consists of pipelining. By implementing the BTFNT one can easily avoid the pipeline hazards and remove pipeline stalls from the architecture.

References

[1] Druglas C. Burger and Taud M. Austin "The Simplescalar Version 2.0", in Computer Architecture News, 25131, pp-13-25, June 1997

[2] Taud Austin, Eric Larson and Dan Ernst, "Simplescalar: An infrastructure for Computer System Modelling", IEEE computer, February 2002.

[3] Druglas C. Burger and Taud M. Austin "Recent Extensions to and simplescalar tool suite", ACM SIGMETRICS Performance Evaluation Review Vol.31, No.4, March 2004.

[4] Dharmesh Parikh, Kevin Skadron, Yan Zhang, Mircea Stan "Power Aware Branch Prediction: Characterization and Design" IEEE transaction on computers, Vol 53, No. 2 pp 168-186 February, 2004

[5] www.simplescalar.com

[6] http://www.ecs.umass.edu/ece/koren/architecture/Simplescalar/SimpleScalar_introduction.htm

[7] http://web.engr.oregonstate.edu/~benl/Projects/branch_pred/