A modified framework for determination of next cache address

By
Shishir Kumar[1], Durgesh Pant[2], Vipin Tyagi[3]
Address- HOD, Dept. of Computer Science & Engineering, Jaypee Institute of Engineering & Technology, Raghogarh, A-B Road, Guna (MP) India Pin – 473226
E mail – **dr.shishir@yahoo.com** Phone number: +91 9826711482

Abstract:

Recent processors issue multiple instructions per cycle and employ multiple functional units and hardware scheduling techniques to achieve maximum parallelism at the instruction level. To exploit maximum efficiency such multiple issue processors must be fed by high instruction fetch bandwidth. The instruction fetch unit must fetch enough instructions every cycle to keep the functional units busy. No clock cycle should go idle and thus several instructions need to be fetched at every clock cycle.

In conventional multi-way set-associative I-cache, an instruction is read in the following way. The address generated by the processor is divided into two parts - tag and index. The index selects the set of the I-cache to be accessed. The tag is compared simultaneously with the cache blocks tags of all the blocks in the set. The data is read from the block whose tag matches with the instruction tag. If none of the stored tags matches with the instruction tag, then a cache miss occurs and the instruction is read from the other levels of the memory hierarchy. With very fast clock cycles, this whole procedure requires more than one clock cycle to complete. It is expected that the future processors, which are likely to have a very deep pipeline, will require several pipeline stages to fetch instructions.

The two main factors that affect the performance of a multi-issue processor are: cache access time and fetching correct instructions every cycle. In this paper we will address both these problems by predicting the address of the Instruction cache from where the next instruction has to be fetched.

1. Introduction

For a set associative cache, the address comprises of the selection of the set as well as the block within the set. This kind of prediction will have two advantages. First, since we are also predicting the block within the cache-set from where the instruction has to be fetched, the tag part of the instruction address need not be compared with the tags of the various other blocks in the set for selecting the block. This will make the cache access faster and also reduce power consumption of the I-cache. [4]

A misprediction is known by comparing the tag and index of the selected block with that of the instruction address. A misprediction however is associated with the penalty of killing wrongly fetched instructions. This scheme reduces the need for sophisticated branch prediction scheme as well. At the fetch stage itself; we are predicting the next cache address for fetching the next instruction, a correct prediction, can thus reduce control hazards.

2. Scheme for determination of next Cache Address:

The factors that affect the performance of a multi-issue processor are the I-cache access time and prediction of the next set of instructions to be fetched into the pipeline. The suggested scheme predicts the location of the cacheline, i.e. the set and way, where the next instruction(s) reside. Since, in this scheme, we need not compare the tag of an instruction address with the tag stored in cache blocks, in a particular set, it makes cache access faster and reduces the I-cache power consumption.

2.1 The Design

The main feature of this scheme is a Set Way Prediction Table, which is a small cache memory whose entries contain the following information-

current set, current way → next set, next way

As the instructions are fetched from an address, an entry in the table is searched for the next set and next way. The current set and way for this entry should be same as the set and way for the current instructions.

The next set and way is the predicted set and way in the cache from where the next instruction is to be fetched. Whenever an instruction is fetched, from a given set, way; the table is looked up for a corresponding entry. If the current set and way is found in the table, then the next instruction is fetched from the predicted set and way.
If no entry is found corresponding to the current set and way, then the default prediction is used. The default prediction is based on the spatial locality principle. Instructions that are located in neighboring positions in memory, when fetched to cache, will tend to fill up the cache column by column. Hence, if the current instruction comes from set s, way w; then the next instruction is likely to be in set (s + l) and way w. This is the default prediction of this scheme.

2.2 Working Principle

Initially the prediction buffer is empty. The cache is accessed in the traditional way to translate the address in the PC to correct cache address (set and way) to read the instruction. After this, the current set and way is used to predict the next set and way and the instruction is speculatively read from the predicted cacheline. If the prediction is not correct, then the speculatively fetched instructions are killed and the prediction table is updated. The correct PC is then used to access the cache in traditional way and fetch the correct instruction.

2.3 Cold Start

At the beginning of the program execution, most predicted cache addresses correspond to invalid cachelines. Accessing these cachelines results in cache miss. In this case, the address of the instruction is constructed using the address of the previous instruction read, and the instruction is read from this address.
It can be noticed that, once the entire cache is filled up by some instructions, all the speculative cache reads will result in a hit. However, as it will be shown later, there is no guarantee that all these speculative reads fetch the desired instruction.

2.4 A Modified Design

As a modification of this scheme, we can use the Branch Target Buffer(BTB) as an additional source for prediction the address to the next instruction to be fetched. As in the earlier scheme, the current set and way is used to predict a new pair of set and way. Simultaneously, the address of the current instruction is used to refer to the BTB as usual, and predict the PC of the next instruction. The tag and index of the predicted PC are compared with the tag and index of the speculatively fetched instruction. If the tag and index of the fetched instruction do not match that of the predicted PC, the speculatively fetched instruction is killed, and the cache is accessed in the traditional way, using the predicted PC. In such a case, the set way prediction table is also updated. It has been seen that this scheme gives better performance when coupled with the BTB. Using this scheme, the focus shifts from address prediction, to predicting the exact location in the cache where the instruction resides. A cacheline replacement (due to cache miss) can have serious implications on the performance.

3. Performance Issues

In a processor using a branch prediction table, the outcome of a instruction branch is predicted. Similarly in a Branch Target Buffer based prediction, the target address of a branch instruction is predicted. Essentially, in these branch prediction schemes, the address to the next instruction is predicted. In proposed scheme, we do not predict the next memory address from where the instruction is to be fetched, but rather predict the position in cache, where the next instruction is likely to be present is predicted. Thus in proposed scheme there is a shift of focus from instruction address, to the actual location of the instruction in the cache.
This distinction, which affects many performance issues of this scheme. If a BTB based scheme makes a prediction of control transfer from an instruction address PC1 to another instruction address PC2, then it signifies that there is a control transfer instruction at address PC1. This control transfer may not always take place (as in the case of a conditional branch). However the control transfer instruction will be present throughout the lifetime of the program. So an entry in the BTB always conveys some relevant piece of information. Now, consider the setway prediction table of proposed scheme. Suppose, due to a cache miss, the instruction in a particular cacheline of the cache gets replaced, then all information stored about that set and way of the cache loses all relevance. This unfortunately gives rise to certain drawbacks in this scheme. The various problems that may arise in such a system are as follows-

3.1 Problem 1

The most common problem occurs when an instruction that is the source or target of a control transfer instruction gets replaced due to cache replacement policy. It can be explained with an example (fig.1). Suppose there is a jump from address0xl06d4 to 0xl0700. Suppose the first instruction is in way 0 of set 53 and the second instruction is in way 0 of set 0. The set way prediction table will contain an entry (0,53) → (0,0), corresponding to this jump. Now, suppose, the instruction 0xl0700 gets replaced due to cache miss. Now, the corresponding entry in the prediction table loses relevance. If at some point later the control reaches at instruction 0xl06d4, then the next set and way will be wrongly predicted as (0,0) where the instruction has been replaced. One solution to this problem could be to invalidate all the entries in the set way prediction table corresponding to a set-way which just got replaced. But, even then this will not substantially improve the situation. Supposing, the entry in the prediction table is invalidated. When control will be transferred to 0xl06d4 at set 53, way 0, the prediction table will be looked up. When no corresponding entry will be found, it will predict by default to fetch instruction from set 54, way 0, which in this case contains 0xl06d8. Now suppose the branch instruction at 0xl06d4 actually takes the branch. Then also a misprediction occurs. In both the cases the performance penalty is the same. However if the branch is not taken after the cacheline gets replaced, then there will be some advantage.

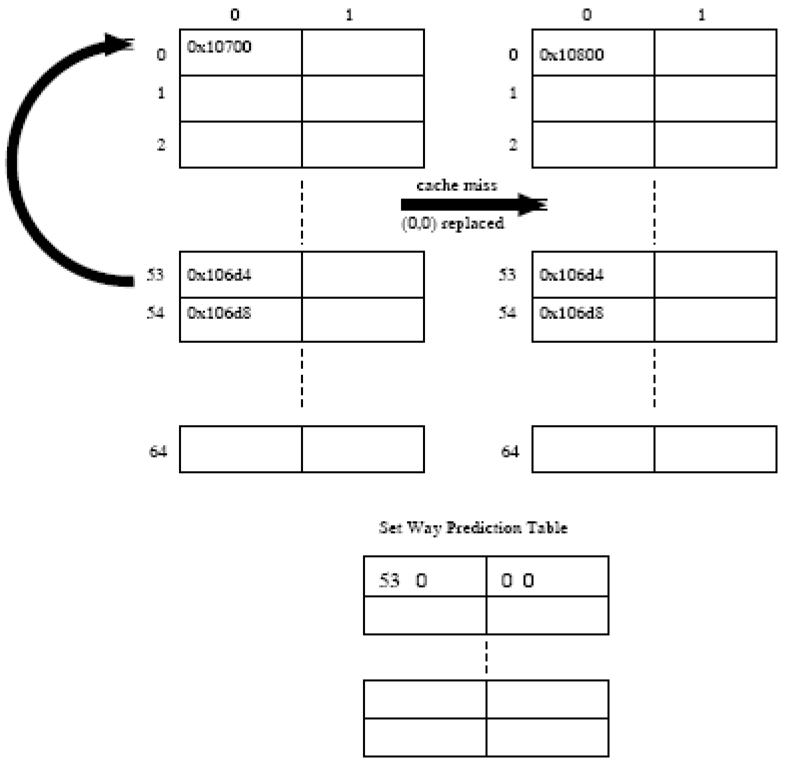

Figure 1: Wrong Prediction due to Cache Replacement

3.2 Problem 2

This problem is explained with an example. Supposing, the instructions 0xl06d4, 0xl06d8 and 0xl06dc are stored in way 0 of the adjacent sets 53, 54 and 55 respectively (fig. 2). Initially there are no entries corresponding to any of these instructions in the prediction table. Now suppose the instruction 0xl06d8 gets replaced by instruction 0xl05d8. Then when the control is transferred to 0xl06d4 at set 53, way 0, the prediction scheme predicts the next instruction to be fetched from set 54, way 0. It is not a cache miss, but the wrong instruction is fetched. A normal cache read will result in a cache miss. After the cache penalty the correct instruction will be read from the memory and placed in the cache at way 1 of set 54.

The mistake will be discovered after a few clock cycles. By that time some other wrong instructions have been fetched and all these instructions have to be killed. The subsequent next prediction will try to fetch an instruction from way 1 of set 55. This prediction will meet a similar fate, since the expected instruction is in way 0 of set 55. So again there will be a penalty associated with it, although the desired instruction is already present in the cache. This type of pseudo cache miss degrades the performance of the system.

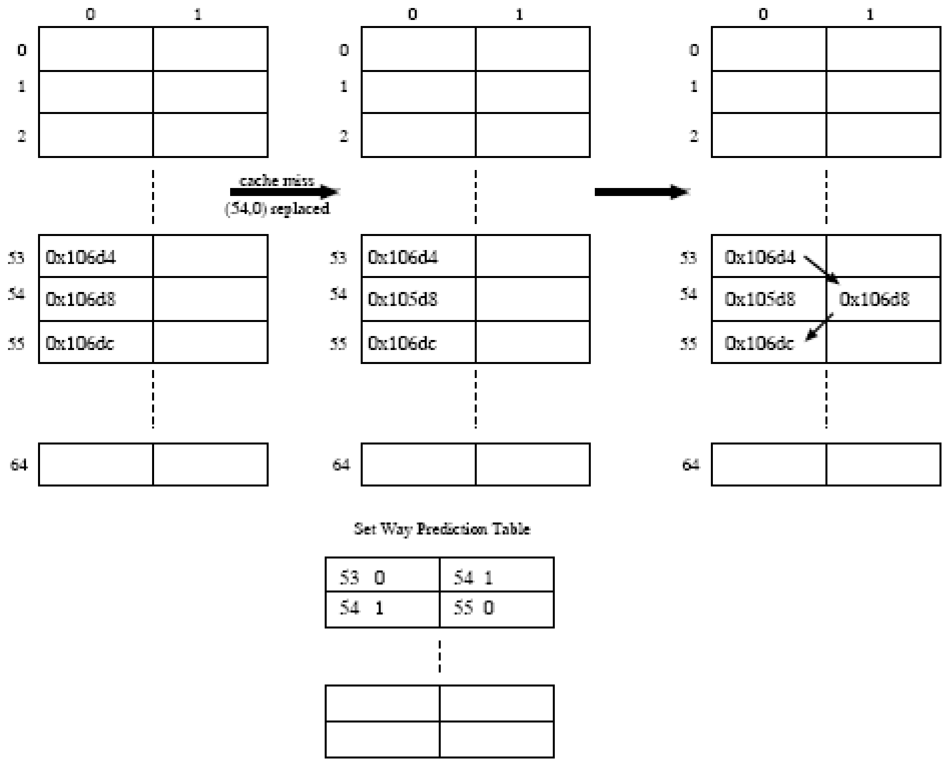

Figure 2: Pseudo Cache Miss

As an added penalty, the set way prediction table will be updated with two entries (53,0) → (54,1) and (54,1) → (55,0). These two entries signify that after set 53, way 0 is accessed, the next instruction is to be fetched from set 54, way 1 and then from set 55 way O. Although there is no jump from 0xl06d4 to0xl06d8 or from 0xl06d8 to 0xl06dc, the entries corresponding to these sequential flow of control, will remain in the prediction table. Thus the prediction table can have entries, which actually depicts sequential flow of control, rather than a jump.

3.3 Branch Target Buffer& next address determination with this scheme

The penalties associated with the problems mentioned above, can be reduced if the mispredictions are detected earlier. We incorporated BTB based prediction also in proposed scheme assuming that BTB gives a more accurate prediction for the addresses. Experimental results confirms it. It can be noticed that, even in the modified scheme, the above mentioned problems will not be eliminated, since the primary source of prediction is still the set way prediction buffer. But, in the modified scheme, whenever we speculatively fetch a wrong instruction, the following comparison with the predicted PC, will result in a mismatch and the error will be detected at a much earlier stage of the pipeline, thus reducing the penalty.

3.4 Problem 3

Association of BTB along with the cache prediction table brings in some more problems. It has been shown that the set way prediction has to store information about instructions which are not of control transfer type.

Since the set way prediction table has a limited number of entries, entries in the table can get replaced, which may be entries for control transfer instructions. However the BTB may still have the information about the branch instruction.

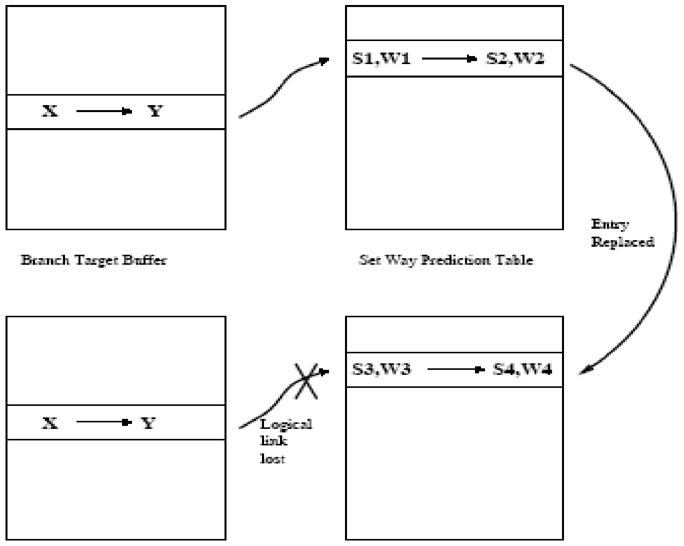

Figure 3: Mismatch Due to Decoupling of BTB and Set Way Prediction Table: Case 1

Sometimes, such inconsistencies between the two tables can lead to degradation in performance as shown in the above example (fig 3). Suppose we have a jump from address X to address Y. The instructions were stored in the cache at (s1,w1) and (s2,w2) respectively. The BTB contains an entry X → Y. Similarly the set way prediction table contains the entry (s1,w1) →(s2,w2) . Now assume that the entry (s1,w1) →(s2,w2) is replaced by some other unrelated entry (s3,w3) →(s4,w4) .Now suppose the control is transferred to the instruction X, stored at set s1 way w1. The set way prediction table has no entry for this cacheline and will mispredict the next instruction to be fetched from set (s1 + 1 way w1), The BTB based prediction provides the next predicted PC to be Y, which is a correct prediction if the branch is taken. A misprediction is detected and all the fetched instructions are killed. After this the correct instruction will be fetched, but after a considerable penalty.

Just the reverse case can also happen (fig .4). Instead of the set way prediction entry being replaced, it may be the case that the entry X →Y in BTB got replaced by another entry A → B. Now, if the control goes to the instruction X, the set way prediction table will predict the next instruction to be in set s2 way w2, which is correct prediction. Later, when the BTB is consulted, and no entry is found for address X, it will predict the next PC to be X+4 (assume that the instructions are all 4 bytes long), which is a misprediction. A mismatch will be detected and the correctly fetched instruction will be killed. A wrong instruction will be fetched as a result of BTB misprediction and the set way prediction table will be updated as a result of which the correct entry of (s1 w1) → (s2 w2) will be removed. This mistake will be detected at the later stage in the pipeline and the correct instruction will be fetched, but after a considerable penalty. It can be noticed that although, it was a misprediction of the BTB and there would have been a penalty even if we had BTB only; the penalty would have been lesser than the penalty in this case.

Conclusions

In proposed scheme, whenever the instruction cache is accessed, associative tag comparison need not be done. This makes the instruction fetch operation faster. Since we are bypassing the tag comparison step, we are reducing the length of the fetch pipeline. Essentially, there are two parallel instruction fetch paths. The shorter one is taken most of the time. The longer traditional path is taken, only when we detect a misprediction. As observed in [4], eliminating tag comparison reduces power consumption, proposed scheme has the added advantage of saving power. The performance of the scheme for predicting the next instruction is comparable to the other existing schemes.

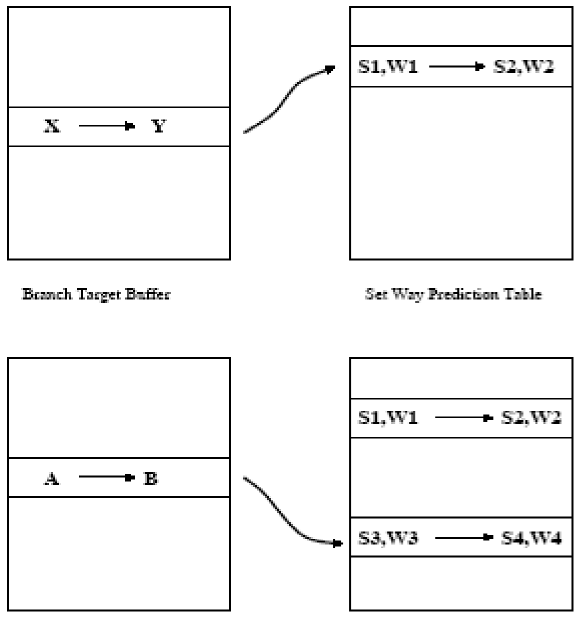

Figure 4: Mismatch Due to Decoupling of BTB and Set Way Prediction Table: Case 2

From the statistical data available after finding simulation results of the proposed scheme it can be concluded that generally proposed scheme gives comparable performance as the other branch prediction based schemes. Only in the cases of Perl Interpreter and Lisp Interpreter program, did the scheme perform poorly. For loop intensive programs, like Whetstone and Matrix Multiplication, the proposed scheme gave good performance. So this scheme can act as an alternative to the different existing sophisticated branch prediction mechanism. Since tag comparison is not performed every time the cache is accessed, the fetch pipeline gets shorter. There are actually two parallel fetch paths in proposed scheme. The shorter path (which is taken more often) bypasses the tag comparison step. The other longer path accesses the cache in traditional way, whenever a misprediction occurs. So, in general, proposed scheme will make cache access faster without compromising with the hit rate.

Bibliography

1. BRAD CALDER, D. G., AND EMER, J. Predictive Sequential Associative Cache. *2nd International Symposium on High Performance Computer Architecture* (February 1996), 244-253. http://www.cs.colorado.edu/~grunwald/Papers/HPCA96-SeqAssocCache / paper. ps.
2. CALDER, B., AND GRUNWALD, D. Next Cache Line and Set Prediction. *Proceedings of the 22nd annual international symposium on Computer architecture* (June 1995), 287 -295. http://dev.acm.org/pubs/articles/proceedings/isca/223982/p287-calder/p287-calder. pdf.
3. HENNESSEY, J. 1., AND PATTERSON, D. *Computer Architecture: A Quantitative Approach,* 2 ed. Harcourt Asia PTE LTD.
4. KOJI INOUE, T. 1., AND MURAKAMI, K. Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption. *Proceedings of* 1999 *International Symposium on Low Power Electronics and Design (ISLPED* '99) (August 1999), 273- 275. http://www.kasuga.csce.kyushu-u.ac.jp/-pparam/paper/PPRAM-TR-42.ps.gz/ .
5. LEE, B. Dynamic Branch Prediction. www.ece.orst.edu/-benl/Projects/branch_pred/ .
6. S. PETER SONG, M. D., AND CHANG, J. The PowerPC 604 RISC microprocessor. *IEEE Micro* (October 1994).
7. YEH, T., AND PATT, Y. Two-level Adaptive Branch Prediction. *24th ACM/ IEEE International Symposium on Micro architecture* (November 1991)
8. http://www.iitk.ac.in
9. http://www.iitd.ac.in